AIM357 Documentation

Release latest

Dec 05, 2019

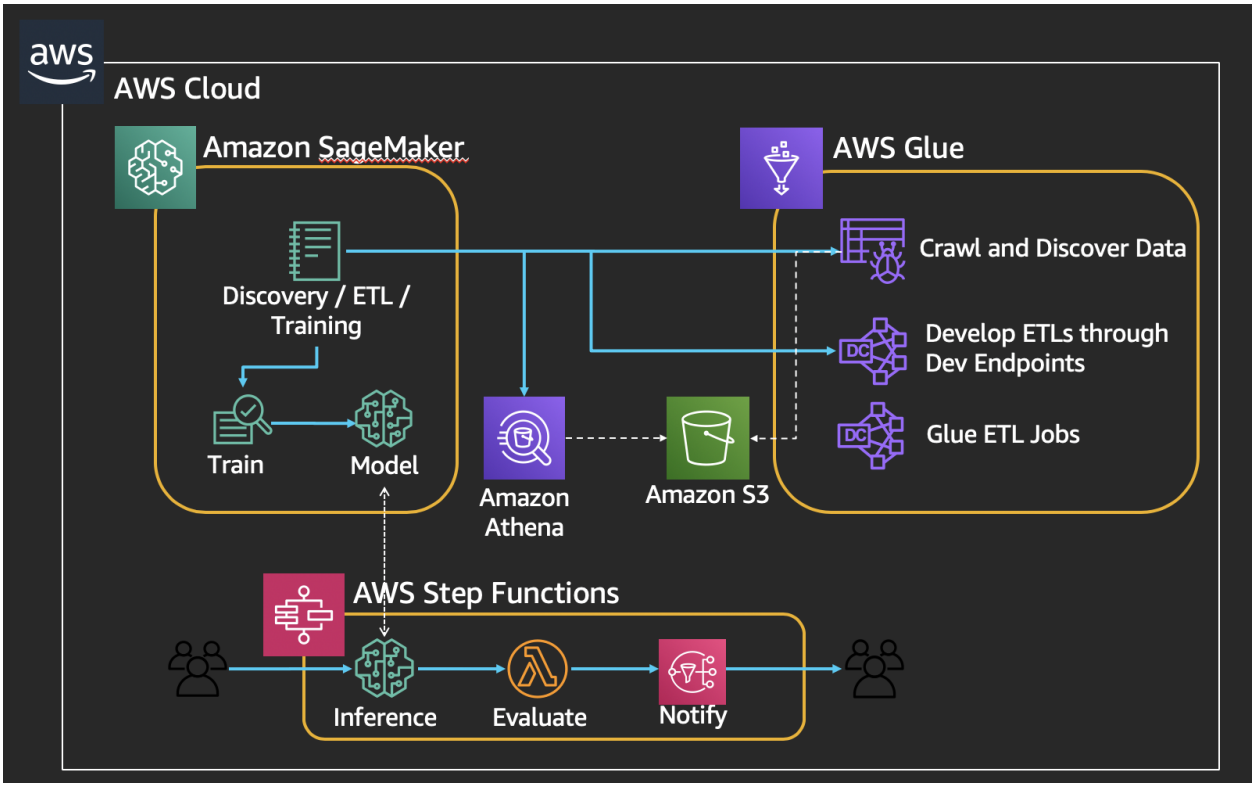
Contents

Machine learning involves more than just training models; you need to source and prepare data, engineer features, select algorithms, train and tune models, and then deploy those models and monitor their performance in production. Learn how to set up an ETL pipeline to analyze customer data using Amazon SageMaker, AWS Glue, and AWS Step Functions.

This workshop will be around the ETL and full pipeline to perofrm Time-series forecasting using NYC Taxi Dataset. It includes the following steps:

- Crawl, Discover, and Explore the new datasets in a Data lake
- Perform Extract, Transform, Load (ETL) jobs to clean the data
- Train a Machine Learning model and run inference
- Assess the response
- Send an alert if value is outside specified range

The workshop uses the following architecture:



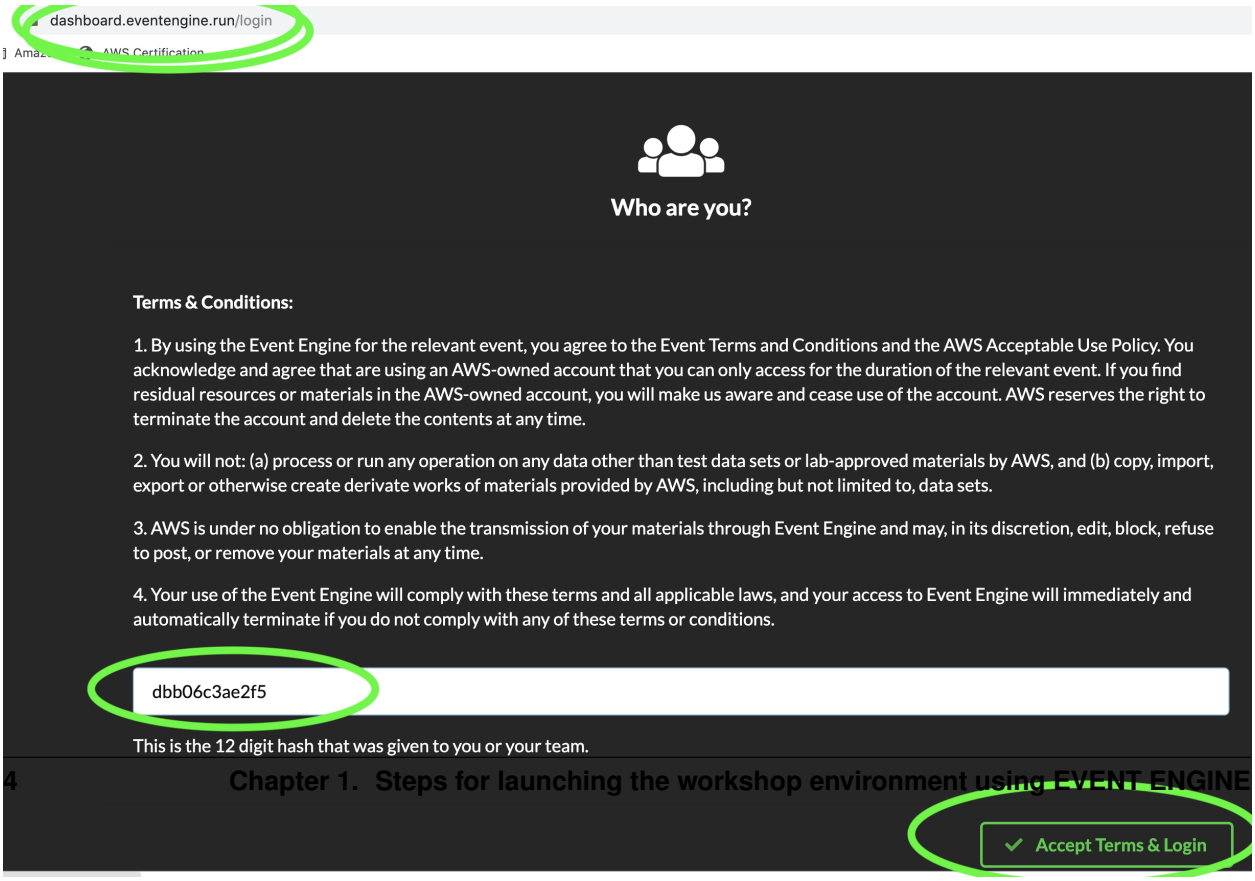
CHAPTER 1

Steps for launching the workshop environment using EVENT ENGINE

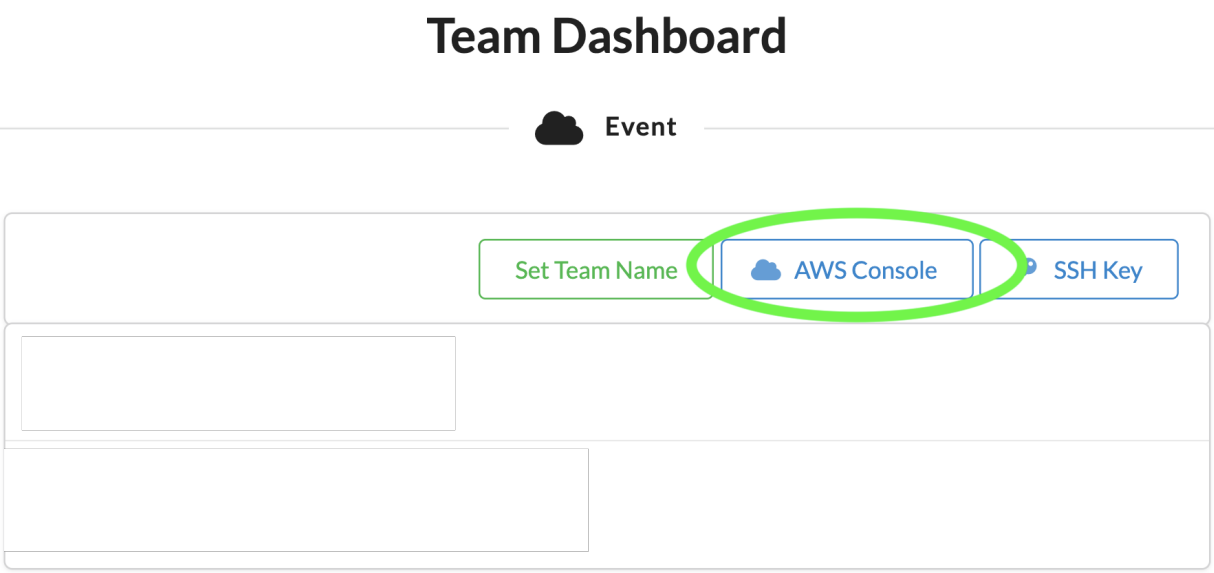
1.1 open a browser and navigate to https://dashboard.eventengine.run/login

1.2 Enter a 12-character "hash" provided to you by workshop organizer.

1.3 Click on "Accpet Terms & Login"

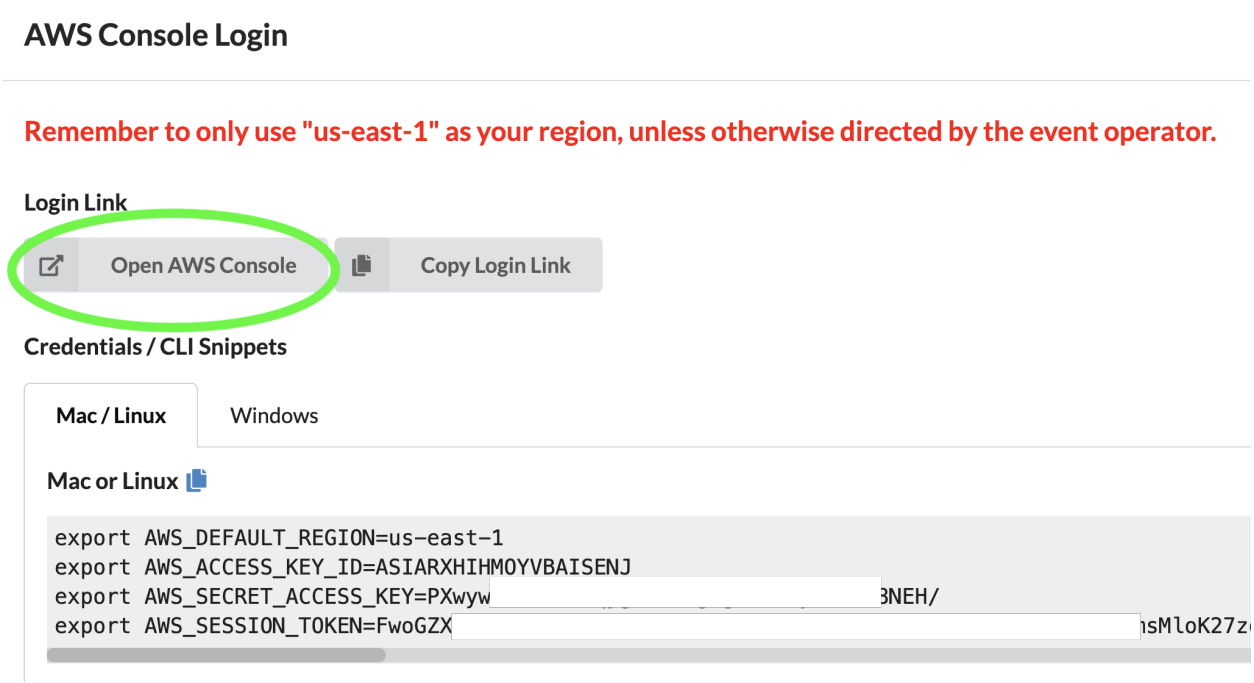


1.4 Click on "AWS Console"



1.5 Please, log off from any other AWS accounts you are currently logged into

1.6 Click on "Open AWS Console"



1.7 You should see a screen like this.

1.7.1 Notebooks

Data Discover and Transformation

in this section of the lab, we'll use Glue to discover new transportation data. From there, we'll use Athena to query and start looking into the dataset to understand the data we are dealing with.

We've also setup a set of ETLs using Glue to create the fields into a canonical form, since all the fields call names different things.

After understanding the data, and cleaning it a little, we'll go into another notebook to perform feature engineering and time series modeling.

What are Databases and Tables in Glue:

When you define a table in the AWS Glue Data Catalog, you add it to a database. A database is used to organize tables in AWS Glue. You can organize your tables using a crawler or using the AWS Glue console. A table can be in only one database at a time.

Your database can contain tables that define data from many different data stores.

A table in the AWS Glue Data Catalog is the metadata definition that represents the data in a data store. You create tables when you run a crawler, or you can create a table manually in the AWS Glue console. The Tables list in the AWS Glue console displays values of your table's metadata. You use table definitions to specify sources and targets when you create ETL (extract, transform, and load) jobs.

```
import boto3

database_name = '2019reinventWorkshop'

## lets first create a namespace for the tables:
glue_client = boto3.client('glue')
create_database_resp = glue_client.create_database(
    DatabaseInput={
        'Name': database_name,
        'Description': 'This database will contain the tables discovered through both␣
↪crawling and the ETL processes'
    }
)
```

This will create a new database, or namespace, that can hold the collection of tables

https://console.aws.amazon.com/glue/home?region=us-east-1#catalog:tab=databases

Name	Description
2019reinventworkshop	This database will contain the tables discovered through both crawling and the ETL processes

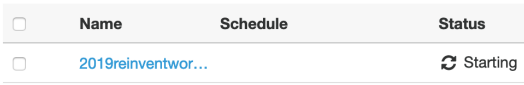
Fig. 1: create db response

You can use a crawler to populate the AWS Glue Data Catalog with tables. This is the primary method used by most AWS Glue users. A crawler can crawl multiple data stores in a single run. Upon completion, the crawler creates or updates one or more tables in your Data Catalog. Extract, transform, and load (ETL) jobs that you define in AWS

Glue use these Data Catalog tables as sources and targets. The ETL job reads from and writes to the data stores that are specified in the source and target Data Catalog tables.

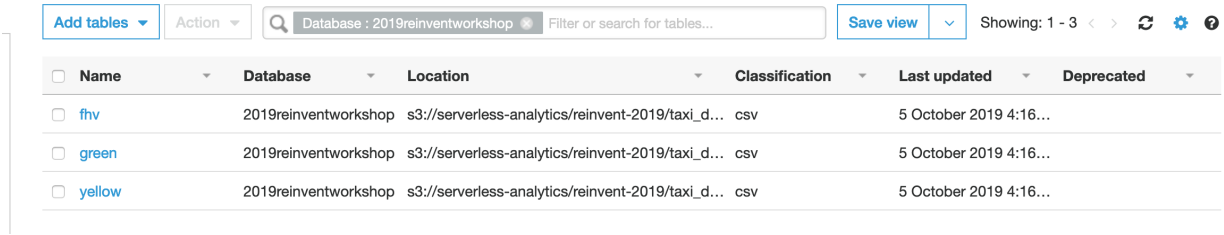
```
crawler_name = '2019reinventworkshopcrawler'
create_crawler_resp = glue_client.create_crawler(
    Name=crawler_name,
    Role='GlueRole',
    DatabaseName=database_name,
    Description='Crawler to discover the base tables for the workshop',
    Targets={
        'S3Targets': [
            {
                'Path': 's3://serverless-analytics/reinvent-2019/taxi_data/',
            },
        ]
    }
)
response = glue_client.start_crawler(
    Name=crawler_name
)
```

After starting the crawler, you can go to the glue console if you'd like to see it running.



https://console.aws.amazon.com/glue/home?region=us-east-1#catalog:tab=crawlers

After it finishes crawling, you can see the datasets (represeted as "tables") it automatically discovered.



Waiting for the Crawler to finish

```
import time

response = glue_client.get_crawler(
    Name=crawler_name
)
while (response['Crawler']['State'] == 'RUNNING') | (response['Crawler']['State'] ==
↪'STOPPING'):
    print(response['Crawler']['State'])
    # Wait for 40 seconds
    time.sleep(40)

    response = glue_client.get_crawler(
        Name=crawler_name
    )

print('finished running', response['Crawler']['State'])
```

```
RUNNING
RUNNING
STOPPING
STOPPING
finished running READY
```

Querying the data

We'll use Athena to query the data. Athena allows us to perform SQL queries against datasets on S3, without having to transform them, load them into a traditional sql datastore, and allows rapid ad-hoc investigation.

Later we'll use Spark to do ETL and feature engineering.

```
!pip install --upgrade pip > /dev/null
!pip install PyAthena > /dev/null
```

Athena uses S3 to store results to allow different types of clients to read it and so you can go back and see the results of previous queries. We can set that up next:

```
import sagemaker
sagemaker_session = sagemaker.Session()
athena_data_bucket = sagemaker_session.default_bucket()
```

Next we'll create an Athena connection we can use, much like a standard JDBC/ODBC connection

```
from pyathena import connect
import pandas as pd

sagemaker_session = sagemaker.Session()

conn = connect(s3_staging_dir="s3://" + athena_data_bucket,
               region_name=sagemaker_session.boto_region_name)

df = pd.read_sql('SELECT \'yellow\' type, count(*) ride_count FROM "' + database_name␣
↪+ '"."yellow" ' +
                 'UNION ALL SELECT \'green\' type, count(*) ride_count FROM "' +␣
↪database_name + '"."green"' +
                 'UNION ALL SELECT \'fhv\' type, count(*) ride_count FROM "' +␣
↪database_name + '"."fhv"', conn)
print(df)
df.plot.bar(x='type', y='ride_count')
```

```
     type  ride_count
0   green    12105351
1  yellow   147263398
2     fhv   292722358
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c9ad19828>
```

```
green_etl = '2019reinvent_green'

response = glue_client.start_job_run(
    JobName=green_etl,
    WorkerType='Standard', # other options include: 'G.1X'|'G.2X',
```

(continues on next page)

(continued from previous page)

```
    NumberOfWorkers=5
)
print('response from starting green')
print(response)
```

```
response from starting green
{'JobRunId': 'jr_d51a70e617a0c7459b3af986ff047ee211696c13ce509736ba01f4778b45b759',
↪'ResponseMetadata': {'RequestId': '40ef03ea-1387-11ea-a9c8-7df52ce46fb6',
↪'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sat, 30 Nov 2019 15:37:02 GMT',
↪'content-type': 'application/x-amz-json-1.1', 'content-length': '82', 'connection':
↪'keep-alive', 'x-amzn-requestid': '40ef03ea-1387-11ea-a9c8-7df52ce46fb6'},
↪'RetryAttempts': 0}}
```

After kicking it off, you can see it running in the console too: https://console.aws.amazon.com/glue/home?region=us-east-1#etl:tab=jobs

WAIT UNTIL THE ETL JOB FINISHES BEFORE CONTINUING! ALSO, YOU MUST CHANGE THE BUCKET PATH IN THIS CELL - FIND THE BUCKET IN S3 THAT CONTAINS '2019reinventetlbucket' in the name

```
#let's list the s3 bucket name:
!aws s3 ls | grep '2019reinventetlbucket' | head -1
```

```
2019-11-30 14:38:27 reinvent-2019reinventetlbucket-656uo7rzqlvu
```

```
# syntax should be s3://...
normalized_bucket = 's3://reinvent-2019reinventetlbucket-656uo7rzqlvu'

assert(normalized_bucket != 's3://FILL_IN_BUCKET_NAME')

create_crawler_resp = glue_client.create_crawler(
    Name=crawler_name + '_normalized',
    Role='GlueRole',
    DatabaseName=database_name,
    Description='Crawler to discover the base tables for the workshop',
    Targets={
        'S3Targets': [
            {
                'Path': normalized_bucket + "/canonical/",
            },
        ]
    }
)
response = glue_client.start_crawler(
    Name=crawler_name + '_normalized'
)
```

Let's wait for the next crawler to finish, this will discover the normalized dataset.

```
import time

response = glue_client.get_crawler(
    Name=crawler_name + '_normalized'
)
```

(continues on next page)

(continued from previous page)

```
while (response['Crawler']['State'] == 'RUNNING') | (response['Crawler']['State'] ==
↪'STOPPING'):
    print(response['Crawler']['State'])
    # Wait for 40 seconds
    time.sleep(40)

    response = glue_client.get_crawler(
        Name=crawler_name + '_normalized'
    )

print('finished running', response['Crawler']['State'])
```

```
RUNNING
RUNNING
STOPPING
STOPPING
finished running READY
```

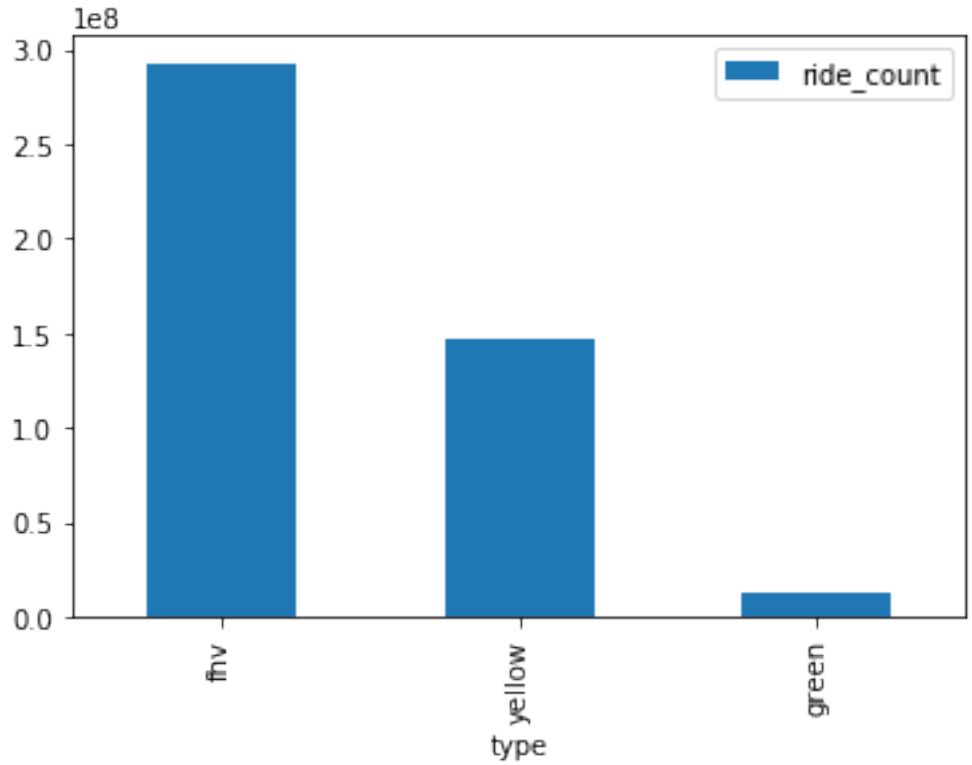
Querying the Normalized Data

Now let's look at the total counts for the aggregated information

```
normalized_df = pd.read_sql('SELECT type, count(*) ride_count FROM "' + database_name␣
↪+ '"."canonical" group by type', conn)
print(normalized_df)
normalized_df.plot.bar(x='type', y='ride_count')
#
```

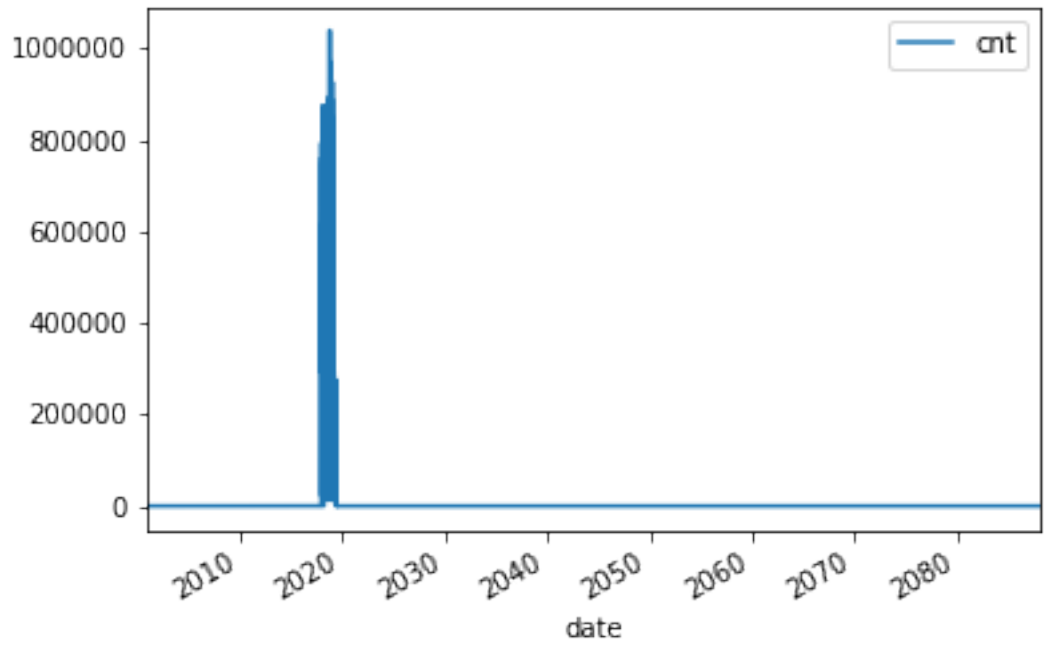
```
     type  ride_count
0     fhv   292722358
1  yellow   147263386
2   green    12105351
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f4570f0>
```




```
query = "select type, date_trunc('day', pickup_datetime) date, count(*) cnt from \""
↪+ database_name + "\".canonical where pickup_datetime < timestamp '2099-12-31'
↪group by type, date_trunc(\'day\', pickup_datetime) "
typeperday_df = pd.read_sql(query, conn)
typeperday_df.plot(x='date', y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f2fc1d0>
```



We see some bad data here...

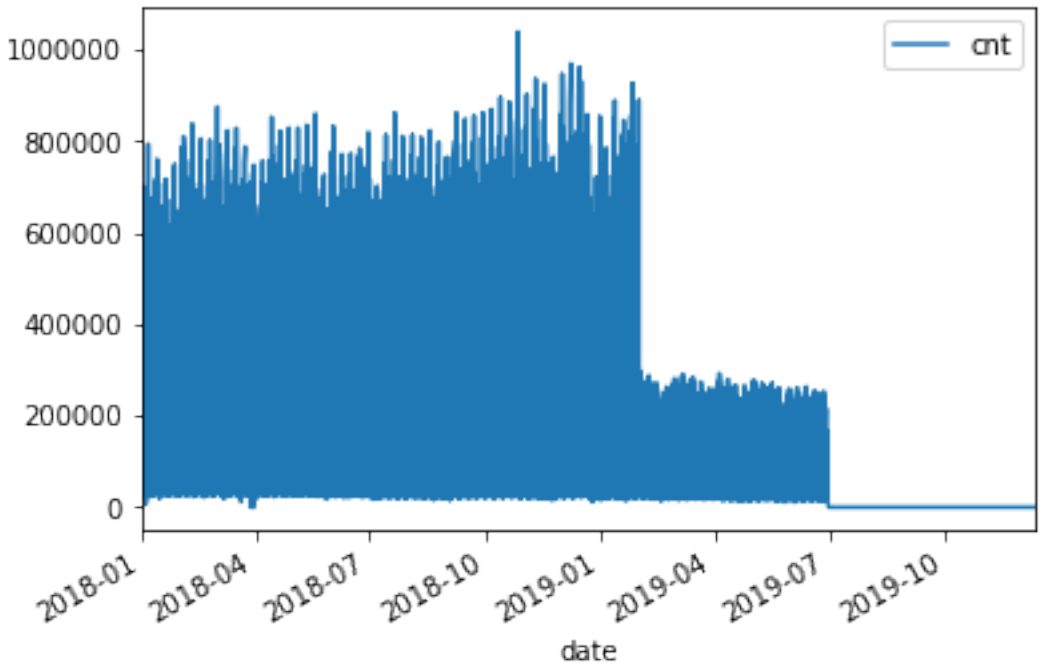
We are expecting only 2018 and 2019 datasets here, but can see there are records far into the future and in the past. This represents bad data that we want to eliminate before we build our model.

```
# Only reason we put this conditional here is so you can execute the cell multiple␣
↪times
# if you don't check, it won't find the 'date' column again and makes interacting w/␣
↪the notebook more seemless
if type(typeperday_df.index) != pd.core.indexes.datetimes.DatetimeIndex:
    print('setting index to date')
    typeperday_df = typeperday_df.set_index('date', drop=True)

typeperday_df.head()
```

```
setting index to date
```

```
typeperday_df.loc['2018-01-01':'2019-12-31'].plot(y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f2c6198>
```



Let's look at some of the bad data now:

All the bad data, at least the bad data in the future, is coming from the yellow taxi license type.

Note, we are querying the transformed data.

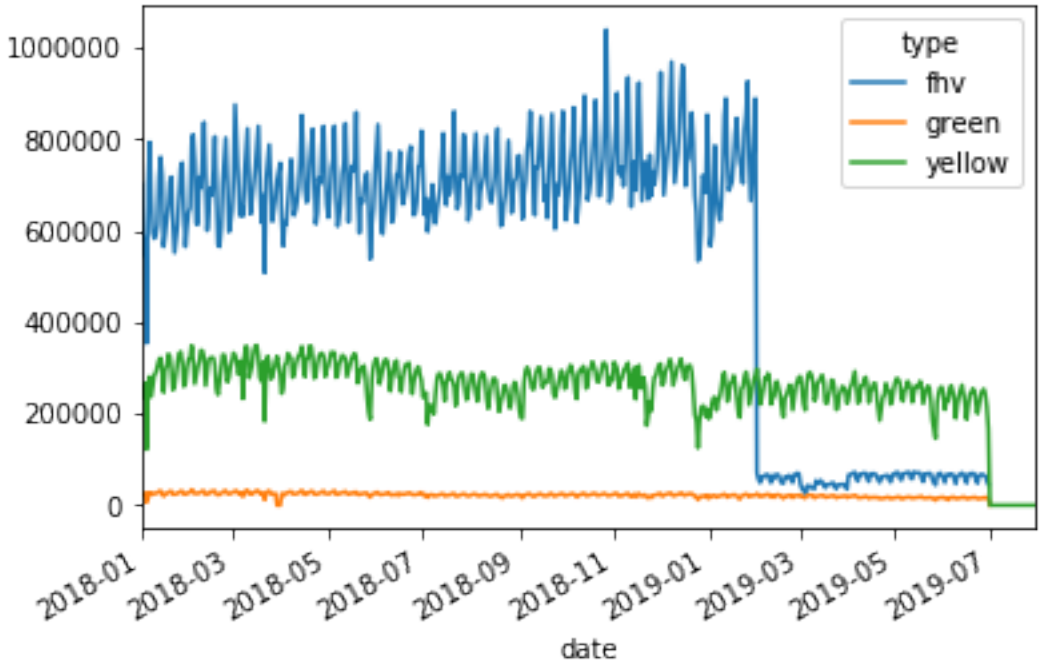
We should check the raw dataset to see if it's also bad or something happened in the ETL process

Let's find the two 2088 records to make sure they are in the source data

```
pd.read_sql("select * from \"" + database_name + "\".yellow where tpep_pickup_
↪datetime like '2088%'", conn)
```

```
## Next let's plot this per type:
typeperday_df.loc['2018-01-01':'2019-07-30'].pivot_table(index='date',
                                                         columns='type',
                                                         values='cnt',
                                                         aggfunc='sum').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f111a58>
```



Fixing our Time Series data

Some details of what caused this drop: #### On August 14, 2018, Mayor de Blasio signed Local Law 149 of 2018, creating a new license category for TLC-licensed FHV businesses that currently dispatch or plan to dispatch more than 10,000 FHV trips in New York City per day under a single brand, trade, or operating name, referred to as High-Volume For-Hire Services (HVFHS). This law went into effect on Feb 1, 2019

Let's bring the other license type and see how it affects the time series charts:

```
create_crawler_resp = glue_client.create_crawler(
    Name=crawler_name + '_fhvhv',
    Role='GlueRole',
    DatabaseName=database_name,
    Description='Crawler to discover the base tables for the workshop',
    Targets={
        'S3Targets': [
            {
                'Path': 's3://serverless-analytics/reinvent-2019_moredata/taxi_data/
↪fhvhv/',
            },
        ]
    }
)
response = glue_client.start_crawler(
    Name=crawler_name + '_fhvhv'
)
```

Wait to discover the fhvhv dataset...

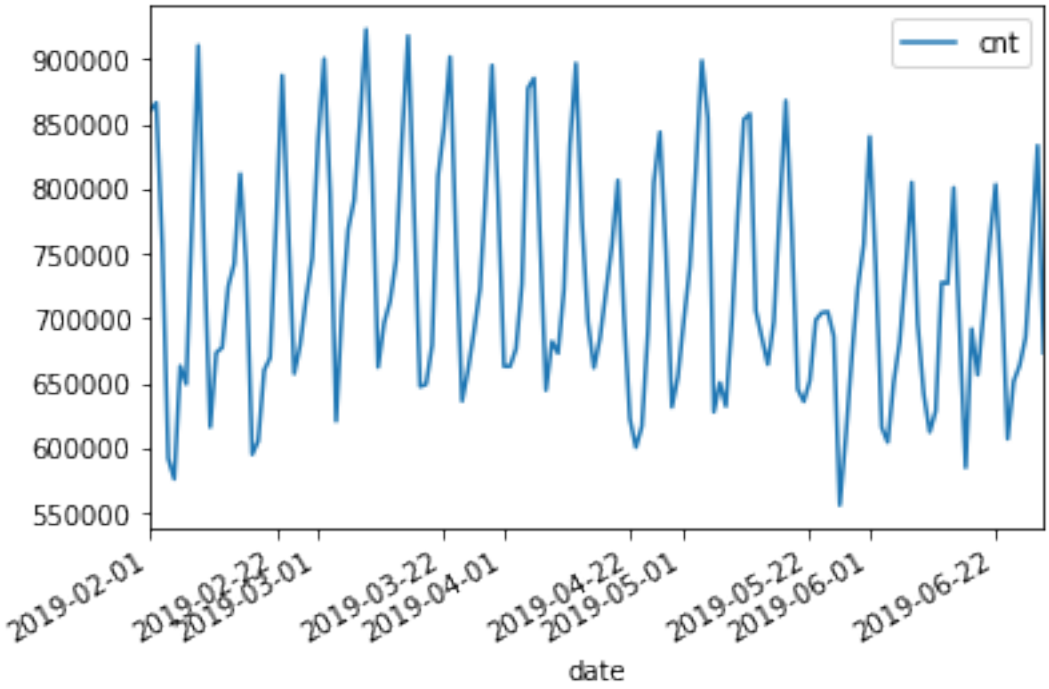
```
import time

response = glue_client.get_crawler(
    Name=crawler_name + '_fhvhv'
)
while (response['Crawler']['State'] == 'RUNNING') | (response['Crawler']['State'] ==
↪'STOPPING'):
    print(response['Crawler']['State'])
    # Wait for 40 seconds
    time.sleep(40)

    response = glue_client.get_crawler(
        Name=crawler_name + '_fhvhv'
    )

print('finished running', response['Crawler']['State'])
```

```
RUNNING
RUNNING
STOPPING
STOPPING
finished running READY
```

```
query = 'select \'fhvhv\' as type, date_trunc(\'day\', cast(pickup_datetime as␣
↪timestamp)) date, count(*) cnt from "' + database_name + '"."fhvhv" group by date_
↪trunc(\'day\',  cast(pickup_datetime as timestamp)) '
typeperday_fhvhv_df = pd.read_sql(query, conn)
typeperday_fhvhv_df = typeperday_fhvhv_df.set_index('date', drop=True)
print(typeperday_fhvhv_df.head())
typeperday_fhvhv_df.plot(y='cnt')
```

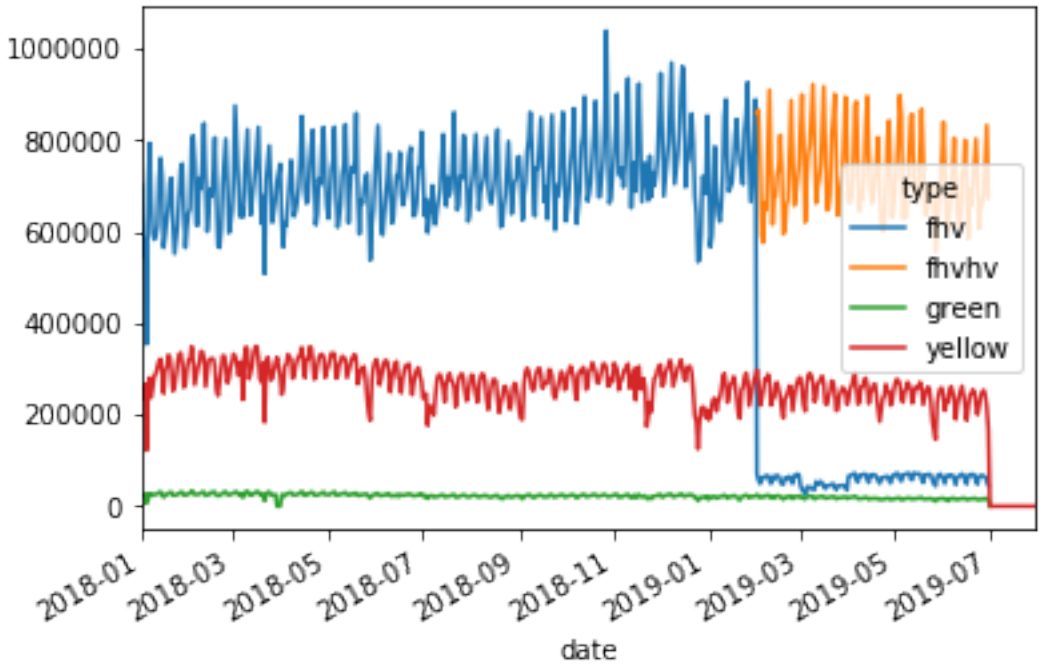
```
             type     cnt
date
2019-05-30  fhvhv  723800
2019-04-23  fhvhv  600870
2019-05-23  fhvhv  698940
2019-03-31  fhvhv  794717
2019-03-17  fhvhv  779620
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f1cb320>
```




```
pd.concat([typeperday_fhvhv_df, typeperday_df], sort=False).loc['2018-01-01':'2019-07-
↪30'].pivot_table(index='date',
                                                                columns='type',
                                                                values='cnt',
                                                                aggfunc='sum').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8f3a9be0>
```



That looks better – let's start looking at performing EDA now. Please open the other notebook file in your SageMaker notebook instance.

Feature Engineering and Training our Model

We'll first setup the glue context in which we can read the glue data catalog, as well as setup some constants.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())

database_name = '2019reinventWorkshop'
canonical_table_name = "canonical"
```

```
Starting Spark application
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
SparkSession available as 'spark'.
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

Reading the Data using the Catalog

Using the glue context, we can read in the data. This is done by using the glue data catalog and looking up the data

Here we can see there are **500 million** records

```
taxi_data = glueContext.create_dynamic_frame.from_catalog(database=database_name,␣
↪table_name=canonical_table_name)
print("2018/2019 Taxi Data Count: ", taxi_data.count())
taxi_data.printSchema()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
2018/2019 Taxi Data Count:  452091095
root
|-- vendorid: string
|-- pickup_datetime: timestamp
|-- dropoff_datetime: timestamp
|-- pulocationid: long
|-- dolocationid: long
|-- type: string
```

Caching in Spark

We'll use the taxi dataframe a bit repeatitively, so we'll cache it ehre and show some sample records.

```
df = taxi_data.toDF().cache()
df.show(10, False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+--------+-------------------+-------------------+------------+------------+-----+
|vendorid|pickup_datetime    |dropoff_datetime   |pulocationid|dolocationid|type |
+--------+-------------------+-------------------+------------+------------+-----+
|null    |null               |null               |null        |null        |green|
|null    |2018-01-01 00:18:50|2018-01-01 00:24:39|null        |null        |green|
|null    |2018-01-01 00:30:26|2018-01-01 00:46:42|null        |null        |green|
|null    |2018-01-01 00:07:25|2018-01-01 00:19:45|null        |null        |green|
|null    |2018-01-01 00:32:40|2018-01-01 00:33:41|null        |null        |green|
|null    |2018-01-01 00:32:40|2018-01-01 00:33:41|null        |null        |green|
|null    |2018-01-01 00:38:35|2018-01-01 01:08:50|null        |null        |green|
|null    |2018-01-01 00:18:41|2018-01-01 00:28:22|null        |null        |green|
|null    |2018-01-01 00:38:02|2018-01-01 00:55:02|null        |null        |green|
|null    |2018-01-01 00:05:02|2018-01-01 00:18:35|null        |null        |green|
+--------+-------------------+-------------------+------------+------------+-----+
only showing top 10 rows
```

Removing invalid dates

When we originally looked at this data, we saw that it had a lot of bad data in it, and timestamps that were outside the range that are valid. Let's ensure we are only using the valid records when aggregating and creating our time series.

```
from pyspark.sql.functions import to_date, lit
from pyspark.sql.types import TimestampType

dates = ("2018-01-01",  "2019-07-01")
date_from, date_to = [to_date(lit(s)).cast(TimestampType()) for s in dates]

df  = df.where((df.pickup_datetime > date_from) & (df.pickup_datetime < date_to))
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

We need to restructure this so that each time is a single row, and the time series values are in the series, followed by the numerical and categorical features

Creating our time series (from individual records)

Right now they are individual records down to the second level, we'll create a record at the day level for each record and then count/aggregate over those.

Let's start by adding a ts_resampled column

```
from pyspark.sql.functions import col, max as max_, min as min_

## day = seconds*minutes*hours
unit = 60 * 60 * 24
epoch = (col("pickup_datetime").cast("bigint") / unit).cast("bigint") * unit

with_epoch = df.withColumn("epoch", epoch)

min_epoch, max_epoch = with_epoch.select(min_("epoch"), max_("epoch")).first()

# Reference range
ref = spark.range(
    min_epoch, max_epoch + 1, unit
).toDF("epoch")

resampled_df = (ref
    .join(with_epoch, "epoch", "left")
    .orderBy("epoch")
    .withColumn("ts_resampled", col("epoch").cast("timestamp")))

resampled_df.cache()

resampled_df.show(10, False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+----------+--------+-------------------+-------------------+------------+------------
↪+------+-------------------+
|epoch     |vendorid|pickup_datetime    |dropoff_datetime   ␣
↪|pulocationid|dolocationid|type  |ts_resampled       |
+----------+--------+-------------------+-------------------+------------+------------
↪+------+-------------------+
|1514764800|fhv     |2018-01-01 04:01:19|2018-01-01 04:06:54|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|null    |2018-01-01 10:55:25|2018-01-01 10:57:42|null        |null        ␣
↪|yellow|2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 03:43:11|2018-01-01 03:53:41|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 04:12:23|2018-01-01 04:36:15|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 05:27:22|2018-01-01 06:01:18|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 04:50:57|2018-01-01 04:56:17|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 04:23:56|2018-01-01 05:17:40|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 17:03:23|2018-01-01 17:33:46|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 17:48:59|2018-01-01 17:58:42|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
|1514764800|fhv     |2018-01-01 15:57:23|2018-01-01 16:09:00|null        |null        ␣
↪|fhv   |2018-01-01 00:00:00|
+----------+--------+-------------------+-------------------+------------+------------
↪+------+-------------------+
only showing top 10 rows
```

Creating our time series data

You can see now that we are resampling per day the resample column, in which we can now aggregate across.

```
from pyspark.sql import functions as func

count_per_day_resamples = resampled_df.groupBy(["ts_resampled", "type"]).count()
count_per_day_resamples.cache()
count_per_day_resamples.show(10, False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+-------------------+------+------+
|ts_resampled       |type  |count |
+-------------------+------+------+
|2019-04-10 00:00:00|green |17165 |
|2018-02-21 00:00:00|green |25651 |
|2018-11-11 00:00:00|yellow|257698|
|2019-02-22 00:00:00|fhv   |65041 |
|2018-03-15 00:00:00|yellow|348198|
|2018-12-30 00:00:00|fhv   |683406|
|2019-03-07 00:00:00|yellow|291098|
|2018-11-28 00:00:00|green |22899 |
|2018-03-05 00:00:00|yellow|290631|
|2018-11-20 00:00:00|yellow|278900|
+-------------------+------+------+
only showing top 10 rows
```

We restructure it so that each taxi type is it's own column in the dataset.

```
time_series_df = count_per_day_resamples.groupBy(["ts_resampled"])\
.pivot('type')\
.sum("count").cache()

time_series_df.show(10,False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+-------------------+------+-----+------+
|ts_resampled       |fhv   |green|yellow|
+-------------------+------+-----+------+
|2019-06-18 00:00:00|69383 |15545|242304|
|2018-12-13 00:00:00|822745|24585|308411|
|2019-03-21 00:00:00|47855 |20326|274057|
|2018-09-09 00:00:00|794608|20365|256918|
|2018-01-31 00:00:00|640887|26667|319256|
|2018-08-16 00:00:00|717045|22113|277677|
|2018-03-21 00:00:00|508492|11981|183629|
|2018-09-20 00:00:00|723583|23378|298630|
|2018-05-15 00:00:00|689620|25458|309023|
|2018-12-24 00:00:00|640740|19314|185895|
```

(continues on next page)

(continued from previous page)

```
+-------------------+------+-----+------+
only showing top 10 rows
```

Local Data Manipulation

Now that we an aggregated time series that is much smaller – let's send this back to the local python environment off the spark cluster on Glue.

```
%%spark -o time_series_df
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

We are in the local panda/python environment now

```
%%local
import pandas as pd
print(time_series_df.dtypes)

time_series_df = time_series_df.set_index('ts_resampled', drop=True)
time_series_df = time_series_df.sort_index()

time_series_df.head()
```

```
ts_resampled    datetime64[ns]
fhv                      int64
green                    int64
yellow                   int64
dtype: object
```

```
VBox(children=(HBox(children=(HTML(value='Type:'), Button(description='Table',␣
↪layout=Layout(width='70px'), st...
```

```
Output()
```

We'll create the training window next, We are going to predict the next week

```
%%local

## number of time-steps that the model is trained to predict
prediction_length = 14

n_weeks = 4
end_training = time_series_df.index[-n_weeks*prediction_length]
print('end training time', end_training)
```

(continues on next page)

(continued from previous page)

```
time_series = []
for ts in time_series_df.columns:
    time_series.append(time_series_df[ts])

time_series_training = []
for ts in time_series_df.columns:
    time_series_training.append(time_series_df.loc[:end_training][ts])
```

```
end training time 2019-05-06 00:00:00
```

We'll install matplotlib in the local kernel to visualize this.

```
%%local
!pip install matplotlib > /dev/null
```

Visualizing the training and test dataset:

In this next cell, we can see how the training and test datasets are split up. Since this is time series, we don't do a random split, instead, we look at how far in the future we are predicting and using that a a knob.

```
%%local
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
#cols_float = time_series_df.drop(['pulocationid', 'dolocationid'], axis=1).columns
cols_float = time_series_df.columns
cmap = matplotlib.cm.get_cmap('Spectral')
colors = cmap(np.arange(0,len(cols_float))/len(cols_float))


plt.figure(figsize=[14,8]);
for c in range(len(cols_float)):
    plt.plot(time_series_df.loc[:end_training][cols_float[c]], alpha=0.5,␣
↪color=colors[c], label=cols_float[c]);
plt.legend(loc='center left');
for c in range(len(cols_float)):
    plt.plot(time_series_df.loc[end_training:][cols_float[c]], alpha=0.25,␣
↪color=colors[c], label=None);
plt.axvline(x=end_training, color='k', linestyle=':');
plt.text(time_series_df.index[int((time_series_df.shape[0]-n_weeks*prediction_
↪length)*0.75)], time_series_df.max().max()/2, 'Train');
plt.text(time_series_df.index[time_series_df.shape[0]-int(n_weeks*prediction_length/
↪2)], time_series_df.max().max()/2, 'Test');
plt.xlabel('Time');
plt.show()
```

Cleaning our Time Series

FHV still has the issue – the time series drops when the law is in place.

we still need to pull in the FHV HV dataset starting in Feb. This represents the rideshare apps going to a difference licence type under the NYC TLC.

```
## we are running back on spark now
fhvhv_data = glueContext.create_dynamic_frame.from_catalog(database=database_name,␣
↪table_name="fhvhv")
fhvhv_df = fhvhv_data.toDF().cache()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

Let's filter the time range just in case we have additional bad records here.

```
fhvhv_df = fhvhv_df.where((fhvhv_df.pickup_datetime > date_from) & (fhvhv_df.pickup_
↪datetime < date_to)).cache()

from pyspark.sql.functions import to_timestamp
fhvhv_df = fhvhv_df.withColumn("pickup_datetime", to_timestamp("pickup_datetime",
↪"yyyy-MM-dd HH:mm:ss"))
fhvhv_df.show(5, False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+-----------------+-------------------+------------------+-------------------+------
↪------+-----------+-------+
|hvfhs_license_num|dispatching_base_num|pickup_datetime    |dropoff_datetime   ␣
↪|pulocationid|dolocationid|sr_flag|
+-----------------+-------------------+------------------+-------------------+------
↪------+-----------+-------+
|HV0003           |B02867              |2019-02-01 00:05:18|2019-02-01 00:14:57|245  ␣
↪      |251         |null   |
|HV0003           |B02879              |2019-02-01 00:41:29|2019-02-01 00:49:39|216  ␣
↪      |197         |null   |
|HV0005           |B02510              |2019-02-01 00:51:34|2019-02-01 01:28:29|261  ␣
↪      |234         |null   |
|HV0005           |B02510              |2019-02-01 00:03:51|2019-02-01 00:07:16|87   ␣
↪      |87          |null   |
|HV0005           |B02510              |2019-02-01 00:09:44|2019-02-01 00:39:56|87   ␣
↪      |198         |null   |
+-----------------+-------------------+------------------+-------------------+------
↪------+-----------+-------+
only showing top 5 rows
```

Let's first create our rollup column for the time resampling

```
from pyspark.sql.functions import col, max as max_, min as min_

## day = seconds*minutes*hours
unit = 60 * 60 * 24

epoch = (col("pickup_datetime").cast("bigint") / unit).cast("bigint") * unit

with_epoch = fhvhv_df.withColumn("epoch", epoch)

min_epoch, max_epoch = with_epoch.select(min_("epoch"), max_("epoch")).first()

ref = spark.range(
    min_epoch, max_epoch + 1, unit
).toDF("epoch")

resampled_fhvhv_df = (ref
    .join(with_epoch, "epoch", "left")
    .orderBy("epoch")
    .withColumn("ts_resampled", col("epoch").cast("timestamp")))

resampled_fhvhv_df = resampled_fhvhv_df.cache()

resampled_fhvhv_df.show(10, False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+----------+-----------------+-------------------+------------------+---------------
↪----+-----------+-----------+-------+-------------------+
|epoch     |hvfhs_license_num|dispatching_base_num|pickup_datetime    |dropoff_
↪datetime   |pulocationid|dolocationid|sr_flag|ts_resampled       |
+----------+-----------------+-------------------+------------------+---------------
↪----+-----------+-----------+-------+-------------------+
```

(continues on next page)

(continued from previous page)

```
|1548979200|HV0003           |B02867              |2019-02-01 00:05:18|2019-02-01␣
↪00:14:57|245          |251          |null    |2019-02-01 00:00:00|
|1548979200|HV0003           |B02879              |2019-02-01 00:41:29|2019-02-01␣
↪00:49:39|216          |197          |null    |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:51:34|2019-02-01␣
↪01:28:29|261          |234          |null    |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:03:51|2019-02-01␣
↪00:07:16|87           |87           |null    |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:09:44|2019-02-01␣
↪00:39:56|87           |198          |null    |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:59:55|2019-02-01␣
↪01:06:28|198          |198          |1       |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:12:06|2019-02-01␣
↪00:42:13|161          |148          |null    |2019-02-01 00:00:00|
|1548979200|HV0005           |B02510              |2019-02-01 00:45:35|2019-02-01␣
↪01:14:56|148          |21           |null    |2019-02-01 00:00:00|
|1548979200|HV0003           |B02867              |2019-02-01 00:10:48|2019-02-01␣
↪00:20:23|226          |260          |null    |2019-02-01 00:00:00|
|1548979200|HV0003           |B02867              |2019-02-01 00:32:32|2019-02-01␣
↪00:40:25|7            |223          |null    |2019-02-01 00:00:00|
+----------+-----------------+--------------------+-------------------+---------------
↪----+------------+------------+-------+-------------------+
only showing top 10 rows
```

Create our Time Series now

```
from pyspark.sql import functions as func
count_per_day_resamples = resampled_fhvhv_df.groupBy(["ts_resampled"]).count()
count_per_day_resamples.cache()
count_per_day_resamples.show(10, False)
fhvhv_timeseries_df = count_per_day_resamples
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
+-------------------+------+
|ts_resampled       |count |
+-------------------+------+
|2019-06-18 00:00:00|692171|
|2019-03-21 00:00:00|809819|
|2019-05-03 00:00:00|815626|
|2019-05-12 00:00:00|857727|
|2019-04-25 00:00:00|689853|
|2019-03-10 00:00:00|812902|
|2019-04-30 00:00:00|655312|
|2019-06-26 00:00:00|663954|
|2019-06-06 00:00:00|682378|
|2019-02-06 00:00:00|663516|
+-------------------+------+
only showing top 10 rows
```

Now we bring this new time series back locally to join it w/ the existing one.

```
%%spark -o fhvhv_timeseries_df
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',␣
↪layout=Layout(height='25px', width='50%'),...
```

We rename the count column to be fhvhv so we can join it w/ the other dataframe

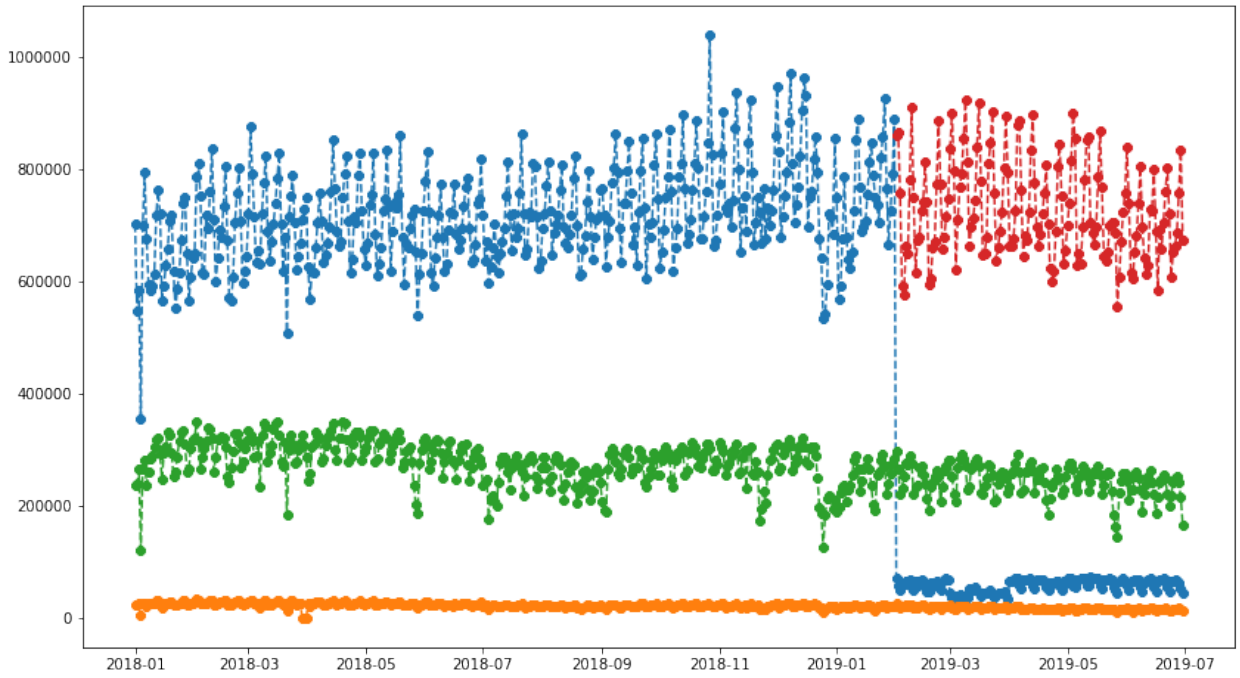
```
%%local
fhvhv_timeseries_df = fhvhv_timeseries_df.rename(columns={"count": "fhvhv"})
fhvhv_timeseries_df = fhvhv_timeseries_df.set_index('ts_resampled', drop=True)
```

Visualizing all the time series data

When we look at the FHVHV dataset starting in Feb 1st, you can see the time series looks normal and there isn't a giant drop in the dataset on that day.

```
%%local
plt.figure(figsize=[14,8]);
plt.plot(time_series_df.join(fhvhv_timeseries_df), marker='8', linestyle='--')
```

```
[<matplotlib.lines.Line2D at 0x7f31f90e32e8>,
 <matplotlib.lines.Line2D at 0x7f31f90aa518>,
 <matplotlib.lines.Line2D at 0x7f31f90aa6d8>,
 <matplotlib.lines.Line2D at 0x7f31f90aa828>]
```



But now we need to combine the FHV and FHVHV dataset

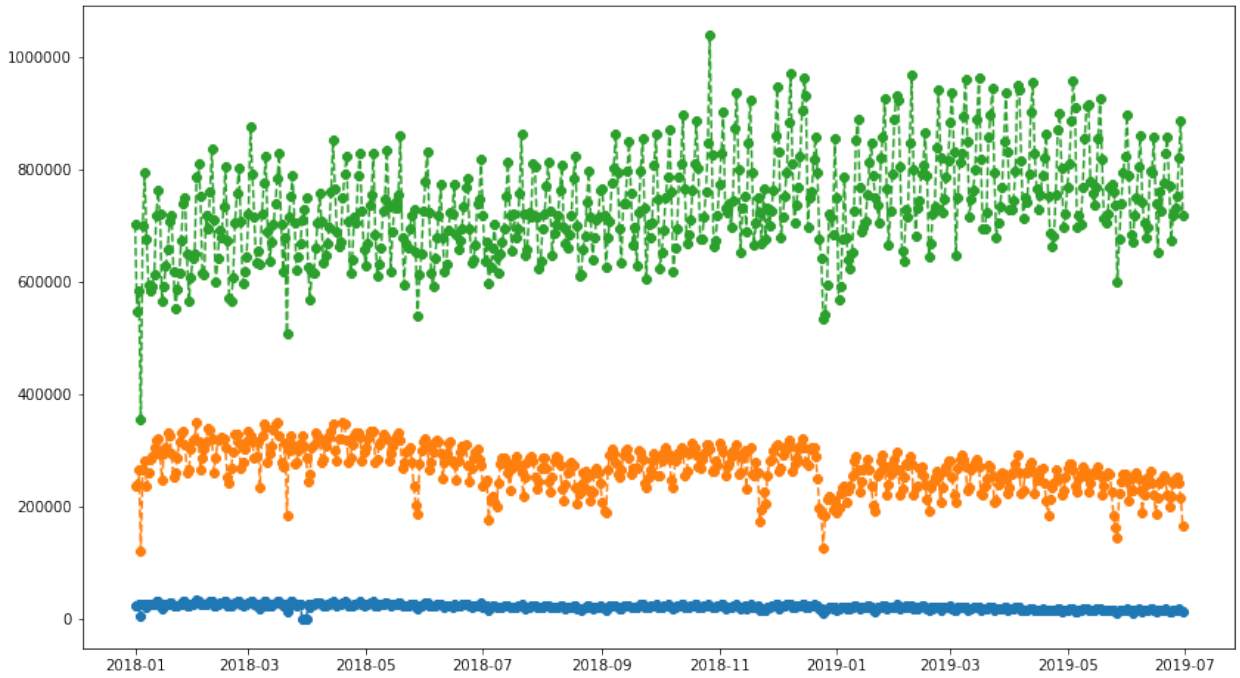
Let's create a new dataset and call it full_fhv meaning both for-hire-vehicles and for-hire-vehicles high volume.

```
%%local
full_timeseries = time_series_df.join(fhvhv_timeseries_df)
full_timeseries = full_timeseries.fillna(0)
full_timeseries['full_fhv'] = full_timeseries['fhv'] + full_timeseries['fhvhv']
full_timeseries = full_timeseries.drop(['fhv', 'fhvhv'], axis=1)

full_timeseries = full_timeseries.fillna(0)
```

Visualizing the joined dataset

```
%%local
plt.figure(figsize=[14,8]);
plt.plot(full_timeseries, marker='8', linestyle='--')
```

```
[<matplotlib.lines.Line2D at 0x7f31f9064080>,
 <matplotlib.lines.Line2D at 0x7f31f9027780>,
 <matplotlib.lines.Line2D at 0x7f31f9027860>]
```



Looking at the training/test split now

```
%%local
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import numpy as np
#cols_float = time_series_df.drop(['pulocationid', 'dolocationid'], axis=1).columns
cols_float = full_timeseries.columns
cmap = matplotlib.cm.get_cmap('Spectral')
colors = cmap(np.arange(0,len(cols_float))/len(cols_float))


plt.figure(figsize=[14,8]);
for c in range(len(cols_float)):
    plt.plot(full_timeseries.loc[:end_training][cols_float[c]], alpha=0.5,␣
↪color=colors[c], label=cols_float[c]);
plt.legend(loc='center left');
for c in range(len(cols_float)):
    plt.plot(full_timeseries.loc[end_training:][cols_float[c]], alpha=0.25,␣
↪color=colors[c], label=None);
plt.axvline(x=end_training, color='k', linestyle=':');
plt.text(full_timeseries.index[int((full_timeseries.shape[0]-n_weeks*prediction_
↪length)*0.75)], full_timeseries.max().max()/2, 'Train');
plt.text(full_timeseries.index[full_timeseries.shape[0]-int(n_weeks*prediction_length/
↪2)], full_timeseries.max().max()/2, 'Test');
plt.xlabel('Time');
plt.show()
```


```
%%local
import json
import boto3

end_training = full_timeseries.index[-n_weeks*prediction_length]
print('end training time', end_training)

time_series = []
```

(continues on next page)

(continued from previous page)

```
for ts in full_timeseries.columns:
    time_series.append(full_timeseries[ts])

time_series_training = []
for ts in full_timeseries.columns:
    time_series_training.append(full_timeseries.loc[:end_training][ts])

import sagemaker
sagemaker_session = sagemaker.Session()
bucket = sagemaker_session.default_bucket()

key_prefix = '2019workshop-deepar/'

s3_client = boto3.client('s3')
def series_to_obj(ts, cat=None):
    obj = {"start": str(ts.index[0]), "target": list(ts)}
    if cat:
        obj["cat"] = cat
    return obj

def series_to_jsonline(ts, cat=None):
    return json.dumps(series_to_obj(ts, cat))

encoding = "utf-8"
data = ''

for ts in time_series_training:
    data = data + series_to_jsonline(ts)
    data = data + '\n'

s3_client.put_object(Body=data.encode(encoding), Bucket=bucket, Key=key_prefix +
↪'data/train/train.json')


data = ''
for ts in time_series:
    data = data + series_to_jsonline(ts)
    data = data + '\n'

s3_client.put_object(Body=data.encode(encoding), Bucket=bucket, Key=key_prefix +
↪'data/test/test.json')
```

```
end training time 2019-05-06 00:00:00
```

```
{'ResponseMetadata': {'RequestId': '080F10A207131EEC',
  'HostId': 'QunHqencw40NUjnNNHS/
↪tFdSLN45HBmNRNPG2VNRqUxbZAZV3gg1Yc5caHB1IN+bl0VnnLNinaY=',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amz-id-2': 'QunHqencw40NUjnNNHS/
↪tFdSLN45HBmNRNPG2VNRqUxbZAZV3gg1Yc5caHB1IN+bl0VnnLNinaY=',
   'x-amz-request-id': '080F10A207131EEC',
   'date': 'Sat, 30 Nov 2019 16:26:35 GMT',
   'etag': '"3d0c723b9f128d637f003391b7546c16"',
   'content-length': '0',
   'server': 'AmazonS3'},
  'RetryAttempts': 0},
```

(continues on next page)

(continued from previous page)

```
 'ETag': '"3d0c723b9f128d637f003391b7546c16"'}
```

Setting our data and output locations

```
%%local
import boto3
import s3fs
import sagemaker
from sagemaker import get_execution_role
sagemaker_session = sagemaker.Session()
role = get_execution_role()

s3_data_path = "{}/{}data".format(bucket, key_prefix)
s3_output_path = "{}/{}output".format(bucket, key_prefix)
```

Setting up the DeepAR Algorithm settings

```
%%local

region = sagemaker_session.boto_region_name
image_name = sagemaker.amazon.amazon_estimator.get_image_uri(region, "forecasting-
↪deepar", "latest")

estimator = sagemaker.estimator.Estimator(
    sagemaker_session=sagemaker_session,
    image_name=image_name,
    role=role,
    train_instance_count=1,
    train_instance_type='ml.c4.2xlarge',
    base_job_name='DeepAR-forecast-taxidata',
    output_path="s3://" + s3_output_path
)

## context_length = The number of time-points that the model gets to see before␣
↪making the prediction.
context_length = 14

hyperparameters = {
    "time_freq": "D",
    "context_length": str(context_length),
    "prediction_length": str(prediction_length),
    "num_cells": "40",
    "num_layers": "3",
    "likelihood": "gaussian",
    "epochs": "100",
    "mini_batch_size": "32",
    "learning_rate": "0.001",
    "dropout_rate": "0.05",
    "early_stopping_patience": "10"
}

estimator.set_hyperparameters(**hyperparameters)
```

Kicking off the training

```
%%local

estimator.fit(inputs={
    "train": "s3://{}/train/".format(s3_data_path),
    "test": "s3://{}/test/".format(s3_data_path)
})
```

```
2019-11-30 16:26:45 Starting - Starting the training job...
2019-11-30 16:27:13 Starting - Launching requested ML instances.........
2019-11-30 16:28:18 Starting - Preparing the instances for training...
2019-11-30 16:29:07 Downloading - Downloading input data...
2019-11-30 16:29:41 Training - Training image download completed. Training in␣
↪progress..[31mArguments: train[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Reading default configuration␣
↪from /opt/amazon/lib/python.7/site-packages/algorithm/resources/default-
↪input.json: {u'num_dynamic_feat': u'auto', u'dropout_rate': u'0.10',␣
↪u'mini_batch_size': u'128', u'test_quantiles': u'[0.1, 0.2, 0.3, 0.4, 0.
↪5, 0.6, 0.7, 0.8, 0.9]', u'_tuning_objective_metric': u'', u'_num_gpus':␣
↪u'auto', u'num_eval_samples': u'100', u'learning_rate': u'0.001', u'num_
↪cells': u'40', u'num_layers': u'2', u'embedding_dimension': u'10', u'_
↪kvstore': u'auto', u'_num_kv_servers': u'auto', u'cardinality': u'auto',␣
↪u'likelihood': u'student-t', u'early_stopping_patience': u''}[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Reading provided configuration␣
↪from /opt/ml/input/config/hyperparameters.json: {u'dropout_rate': u'0.05',
↪ u'learning_rate': u'0.001', u'num_cells': u'40', u'prediction_length':␣
↪u'14', u'epochs': u'100', u'time_freq': u'D', u'context_length': u'14',␣
↪u'num_layers': u'3', u'mini_batch_size': u'32', u'likelihood': u'gaussian',␣
↪u'early_stopping_patience': u'10'}[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Final configuration:
↪{u'dropout_rate': u'0.05', u'test_quantiles': u'[0.1, 0.2, 0.3, 0.4, 0.5, 0.
↪6, 0.7, 0.8, 0.9]', u'_tuning_objective_metric': u'', u'num_eval_samples':␣
↪u'100', u'learning_rate': u'0.001', u'num_layers': u'3', u'epochs': u'100',
↪ u'embedding_dimension': u'10', u'num_cells': u'40', u'_num_kv_servers':␣
↪u'auto', u'mini_batch_size': u'32', u'likelihood': u'gaussian', u'num_
↪dynamic_feat': u'auto', u'cardinality': u'auto', u'_num_gpus': u'auto',␣
↪u'prediction_length': u'14', u'time_freq': u'D', u'context_length': u'14',␣
↪u'_kvstore': u'auto', u'early_stopping_patience': u'10'}[0m
[31mProcess 1 is a worker.[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Detected entry point for␣
↪worker worker[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Using early stopping with␣
↪patience 10[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] [cardinality=auto] cat field␣
↪was NOT found in the file /opt/ml/input/data/train/train.json and will NOT␣
↪be used for training.[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] [num_dynamic_feat=auto]␣
↪dynamic_feat field was NOT found in the file /opt/ml/input/data/train/train.
↪json and will NOT be used for training.[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Training set statistics:[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Integer time series[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] number of time series: 3[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] number of observations: 1473[0m
```

```
[31m[11/30/2019 16:29:43 INFO 140657760761664] mean target length: 491[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] min/mean/max target: 5.0/
↪342890.394433/1039874.0[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] mean abs(target): 342890.
↪394433[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] contains missing values: no[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Small number of time series.␣
↪Doing 10 number of passes over dataset per epoch.[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Test set statistics:[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Integer time series[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] number of time series: 3[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] number of observations: 1638[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] mean target length: 546[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] min/mean/max target: 5.0/
↪342546.620269/1039874.0[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] mean abs(target): 342546.
↪620269[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] contains missing values: no[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] nvidia-smi took: 0.
↪0251910686493 secs to identify 0 gpus[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Number of GPUs being used: 0[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Create Store: local[0m
[31m#metrics {"Metrics": {"get_graph.time": {"count": 1, "max": 72.
↪59106636047363, "sum": 72.59106636047363, "min": 72.59106636047363}},␣
↪"EndTime": 1575131383.872348, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131383.79892}
[0m
[31m[11/30/2019 16:29:43 INFO 140657760761664] Number of GPUs being used: 0[0m
[31m#metrics {"Metrics": {"initialize.time": {"count": 1, "max": 179.
↪97288703918457, "sum": 179.97288703918457, "min": 179.97288703918457}},␣
↪"EndTime": 1575131383.979019, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131383.872428}
[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[0] Batch[0] avg_epoch_
↪loss=13.084117[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=0, batch=0 train loss <loss>=13.0841169357[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[0] Batch[5] avg_epoch_
↪loss=12.561649[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=0, batch=5 train loss <loss>=12.5616491636[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[0] Batch [5]#011Speed:␣
↪1082.40 samples/sec#011loss=12.561649[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] processed a total of 304␣
↪examples[0m
[31m#metrics {"Metrics": {"epochs": {"count": 1, "max": 100, "sum": 100.0,␣
↪"min": 100}, "update.time": {"count": 1, "max": 438.4138584136963, "sum":␣
↪438.4138584136963, "min": 438.4138584136963}}, "EndTime": 1575131384.417562,
↪ "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm":␣
↪"AWS/DeepAR"}, "StartTime": 1575131383.97908}
[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=693.261703397 records/second[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #progress_metric: host=algo-1,␣
```

```
↪completed 1 % of epochs[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=0, train loss <loss>=12.4054102898[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_69c66750-0a52-494a-838b-eff95388ff56-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 14.
↪506101608276367, "sum": 14.506101608276367, "min": 14.506101608276367}},␣
↪"EndTime": 1575131384.432697, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131384.417626}
[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[1] Batch[0] avg_epoch_
↪loss=11.543509[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=1, batch=0 train loss <loss>=11.5435094833[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[1] Batch[5] avg_epoch_
↪loss=11.853811[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=1, batch=5 train loss <loss>=11.8538109461[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[1] Batch [5]#011Speed:␣
↪1200.96 samples/sec#011loss=11.853811[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] processed a total of 303␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 313.
↪37714195251465, "sum": 313.37714195251465, "min": 313.37714195251465}},␣
↪"EndTime": 1575131384.746183, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131384.432759}
[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=966.559590765 records/second[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 2 % of epochs[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=1, train loss <loss>=11.612717247[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_180ffbc0-0b42-4349-aeb2-3dd997e411ee-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 14.
↪13583755493164, "sum": 14.13583755493164, "min": 14.13583755493164}},␣
↪"EndTime": 1575131384.760916, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131384.746256}
[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[2] Batch[0] avg_epoch_
↪loss=11.307412[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=2, batch=0 train loss <loss>=11.3074121475[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[2] Batch[5] avg_epoch_
↪loss=11.342184[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=2, batch=5 train loss <loss>=11.3421843847[0m
[31m[11/30/2019 16:29:44 INFO 140657760761664] Epoch[2] Batch [5]#011Speed:␣
↪1186.24 samples/sec#011loss=11.342184[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[2] Batch[10] avg_epoch_
↪loss=11.268009[0m
```

```
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=2, batch=10 train loss <loss>=11.1789993286[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[2] Batch [10]#011Speed:␣
↪1131.09 samples/sec#011loss=11.178999[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] processed a total of 328␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 351.
↪1309623718262, "sum": 351.1309623718262, "min": 351.1309623718262}},␣
↪"EndTime": 1575131385.112164, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131384.760976}
[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=933.823852992 records/second[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 3 % of epochs[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=2, train loss <loss>=11.2680093592[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_cb8228c2-d13f-4282-a8be-6c0110ee4205-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 13.
↪696908950805664, "sum": 13.696908950805664, "min": 13.696908950805664}},␣
↪"EndTime": 1575131385.126428, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131385.112241}
[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[3] Batch[0] avg_epoch_
↪loss=11.775387[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=3, batch=0 train loss <loss>=11.7753868103[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[3] Batch[5] avg_epoch_
↪loss=11.569252[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=3, batch=5 train loss <loss>=11.5692516963[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[3] Batch [5]#011Speed:␣
↪1216.54 samples/sec#011loss=11.569252[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] processed a total of 296␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 320.
↪6939697265625, "sum": 320.6939697265625, "min": 320.6939697265625}},␣
↪"EndTime": 1575131385.447235, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131385.126488}
[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=922.675933702 records/second[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 4 % of epochs[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=3, train loss <loss>=11.6022842407[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[4] Batch[0] avg_epoch_
↪loss=11.529972[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=4, batch=0 train loss <loss>=11.5299720764[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[4] Batch[5] avg_epoch_
```

```
↪loss=11.531543[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=4, batch=5 train loss <loss>=11.531542778[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[4] Batch [5]#011Speed:␣
↪1128.21 samples/sec#011loss=11.531543[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] processed a total of 308␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 336.
↪6720676422119, "sum": 336.6720676422119, "min": 336.6720676422119}},␣
↪"EndTime": 1575131385.784423, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131385.447313}
[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=914.529829482 records/second[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 5 % of epochs[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=4, train loss <loss>=11.496231842[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] Epoch[5] Batch[0] avg_epoch_
↪loss=11.429693[0m
[31m[11/30/2019 16:29:45 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=5, batch=0 train loss <loss>=11.429693222[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[5] Batch[5] avg_epoch_
↪loss=11.502197[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=5, batch=5 train loss <loss>=11.5021974246[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[5] Batch [5]#011Speed:␣
↪1210.92 samples/sec#011loss=11.502197[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[5] Batch[10] avg_epoch_
↪loss=11.288384[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=5, batch=10 train loss <loss>=11.0318073273[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[5] Batch [10]#011Speed:␣
↪1166.00 samples/sec#011loss=11.031807[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] processed a total of 336␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 367.095947265625,
↪ "sum": 367.095947265625, "min": 367.095947265625}}, "EndTime": 1575131386.
↪152045, "Dimensions": {"Host": "algo-1", "Operation": "training",␣
↪"Algorithm": "AWS/DeepAR"}, "StartTime": 1575131385.7845}
[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=915.037687483 records/second[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 6 % of epochs[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=5, train loss <loss>=11.288383744[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[6] Batch[0] avg_epoch_
↪loss=11.120630[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=6, batch=0 train loss <loss>=11.1206302643[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[6] Batch[5] avg_epoch_
```

```
↪loss=11.133206[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=6, batch=5 train loss <loss>=11.1332060496[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[6] Batch [5]#011Speed:␣
↪1149.90 samples/sec#011loss=11.133206[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] processed a total of 305␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 314.
↪59808349609375, "sum": 314.59808349609375, "min": 314.59808349609375}},␣
↪"EndTime": 1575131386.467154, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131386.15211}
[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=969.142829437 records/second[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 7 % of epochs[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=6, train loss <loss>=11.1763834[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_bc44f47d-8f44-4ede-b4e3-9e726c1c9b8c-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 14.
↪055013656616211, "sum": 14.055013656616211, "min": 14.055013656616211}},␣
↪"EndTime": 1575131386.481823, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131386.46723}
[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[7] Batch[0] avg_epoch_
↪loss=11.204243[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=7, batch=0 train loss <loss>=11.2042427063[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[7] Batch[5] avg_epoch_
↪loss=11.445836[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=7, batch=5 train loss <loss>=11.4458359083[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[7] Batch [5]#011Speed:␣
↪1099.40 samples/sec#011loss=11.445836[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] processed a total of 316␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 322.
↪6950168609619, "sum": 322.6950168609619, "min": 322.6950168609619}},␣
↪"EndTime": 1575131386.804633, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131386.481883}
[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=978.940348457 records/second[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 8 % of epochs[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=7, train loss <loss>=11.4341204643[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] Epoch[8] Batch[0] avg_epoch_
↪loss=11.525706[0m
[31m[11/30/2019 16:29:46 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=8, batch=0 train loss <loss>=11.5257062912[0m
```

```
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[8] Batch[5] avg_epoch_
↪loss=11.321895[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=8, batch=5 train loss <loss>=11.3218946457[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[8] Batch [5]#011Speed:␣
↪1002.87 samples/sec#011loss=11.321895[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[8] Batch[10] avg_epoch_
↪loss=10.957104[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=8, batch=10 train loss <loss>=10.5193548203[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[8] Batch [10]#011Speed:␣
↪1042.35 samples/sec#011loss=10.519355[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] processed a total of 326␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 383.
↪8310241699219, "sum": 383.8310241699219, "min": 383.8310241699219}},␣
↪"EndTime": 1575131387.189002, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131386.8047}
[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=849.088935738 records/second[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 9 % of epochs[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=8, train loss <loss>=10.9571038159[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_3517a004-cfca-454a-a835-341f5c4e6434-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 21.
↪010875701904297, "sum": 21.010875701904297, "min": 21.010875701904297}},␣
↪"EndTime": 1575131387.210623, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131387.189077}
[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[9] Batch[0] avg_epoch_
↪loss=11.336051[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=9, batch=0 train loss <loss>=11.3360509872[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[9] Batch[5] avg_epoch_
↪loss=11.312789[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=9, batch=5 train loss <loss>=11.3127894402[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[9] Batch [5]#011Speed:␣
↪1033.45 samples/sec#011loss=11.312789[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[9] Batch[10] avg_epoch_
↪loss=11.353988[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=9, batch=10 train loss <loss>=11.403427124[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[9] Batch [10]#011Speed:␣
↪881.09 samples/sec#011loss=11.403427[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] processed a total of 336␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 413.
↪4540557861328, "sum": 413.4540557861328, "min": 413.4540557861328}},␣
↪"EndTime": 1575131387.624198, "Dimensions": {"Host": "algo-1", "Operation":␣
```

```
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131387.210689}
[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=812.447549835 records/second[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 10 % of epochs[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=9, train loss <loss>=11.3539883874[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[10] Batch[0] avg_epoch_
↪loss=10.794859[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=10, batch=0 train loss <loss>=10.7948589325[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[10] Batch[5] avg_epoch_
↪loss=11.195995[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=10, batch=5 train loss <loss>=11.1959945361[0m
[31m[11/30/2019 16:29:47 INFO 140657760761664] Epoch[10] Batch [5]#011Speed:␣
↪949.42 samples/sec#011loss=11.195995[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[10] Batch[10] avg_epoch_
↪loss=11.216783[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=10, batch=10 train loss <loss>=11.2417282104[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[10] Batch [10]#011Speed:␣
↪899.09 samples/sec#011loss=11.241728[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] processed a total of 350␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 435.
↪26315689086914, "sum": 435.26315689086914, "min": 435.26315689086914}},␣
↪"EndTime": 1575131388.060028, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131387.624271}
[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=803.899002688 records/second[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 11 % of epochs[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=10, train loss <loss>=11.2167825699[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[11] Batch[0] avg_epoch_
↪loss=11.323181[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=11, batch=0 train loss <loss>=11.3231811523[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[11] Batch[5] avg_epoch_
↪loss=11.015685[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=11, batch=5 train loss <loss>=11.0156849225[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[11] Batch [5]#011Speed:␣
↪847.25 samples/sec#011loss=11.015685[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[11] Batch[10] avg_epoch_
↪loss=11.156606[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=11, batch=10 train loss <loss>=11.3257108688[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[11] Batch [10]#011Speed:␣
```

```
↪1025.19 samples/sec#011loss=11.325711[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] processed a total of 333␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 455.
↪63697814941406, "sum": 455.63697814941406, "min": 455.63697814941406}},␣
↪"EndTime": 1575131388.516234, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131388.060105}
[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=730.671786467 records/second[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 12 % of epochs[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=11, train loss <loss>=11.1566058072[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[12] Batch[0] avg_epoch_
↪loss=11.386516[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=12, batch=0 train loss <loss>=11.3865156174[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[12] Batch[5] avg_epoch_
↪loss=11.044365[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=12, batch=5 train loss <loss>=11.044365406[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[12] Batch [5]#011Speed:␣
↪1024.15 samples/sec#011loss=11.044365[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[12] Batch[10] avg_epoch_
↪loss=11.160591[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=12, batch=10 train loss <loss>=11.3000627518[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] Epoch[12] Batch [10]#011Speed:␣
↪1059.63 samples/sec#011loss=11.300063[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] processed a total of 335␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 406.
↪9790840148926, "sum": 406.9790840148926, "min": 406.9790840148926}},␣
↪"EndTime": 1575131388.923674, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131388.516307}
[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=822.915883581 records/second[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 13 % of epochs[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=12, train loss <loss>=11.1605914723[0m
[31m[11/30/2019 16:29:48 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[13] Batch[0] avg_epoch_
↪loss=11.314456[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=13, batch=0 train loss <loss>=11.314455986[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[13] Batch[5] avg_epoch_
↪loss=11.248668[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=13, batch=5 train loss <loss>=11.2486680349[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[13] Batch [5]#011Speed:␣
```

```
↪1105.80 samples/sec#011loss=11.248668[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[13] Batch[10] avg_epoch_
↪loss=11.205602[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=13, batch=10 train loss <loss>=11.1539218903[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[13] Batch [10]#011Speed:␣
↪995.09 samples/sec#011loss=11.153922[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] processed a total of 348␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 397.
↪4108695983887, "sum": 397.4108695983887, "min": 397.4108695983887}},␣
↪"EndTime": 1575131389.321589, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131388.923749}
[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=875.46058706 records/second[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 14 % of epochs[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=13, train loss <loss>=11.2056016055[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[14] Batch[0] avg_epoch_
↪loss=10.776047[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=14, batch=0 train loss <loss>=10.7760467529[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[14] Batch[5] avg_epoch_
↪loss=10.890142[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=14, batch=5 train loss <loss>=10.890141805[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[14] Batch [5]#011Speed:␣
↪1192.29 samples/sec#011loss=10.890142[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[14] Batch[10] avg_epoch_
↪loss=10.867761[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=14, batch=10 train loss <loss>=10.8409036636[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[14] Batch [10]#011Speed:␣
↪1086.88 samples/sec#011loss=10.840904[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] processed a total of 328␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 359.
↪12179946899414, "sum": 359.12179946899414, "min": 359.12179946899414}},␣
↪"EndTime": 1575131389.681233, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131389.321647}
[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=913.043344833 records/second[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 15 % of epochs[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=14, train loss <loss>=10.8677608317[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_0fec42b3-c584-4660-9bba-d7bcd3a29e84-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 17.
```

```
↪184019088745117, "sum": 17.184019088745117, "min": 17.184019088745117}},␣
↪"EndTime": 1575131389.69897, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131389.681311}
[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[15] Batch[0] avg_epoch_
↪loss=10.758276[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=15, batch=0 train loss <loss>=10.7582759857[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[15] Batch[5] avg_epoch_
↪loss=11.068750[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=15, batch=5 train loss <loss>=11.0687500636[0m
[31m[11/30/2019 16:29:49 INFO 140657760761664] Epoch[15] Batch [5]#011Speed:␣
↪1117.68 samples/sec#011loss=11.068750[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] processed a total of 300␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 329.
↪8060894012451, "sum": 329.8060894012451, "min": 329.8060894012451}},␣
↪"EndTime": 1575131390.028907, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131389.699043}
[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=909.302658336 records/second[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 16 % of epochs[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=15, train loss <loss>=10.9549746513[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[16] Batch[0] avg_epoch_
↪loss=11.169759[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=16, batch=0 train loss <loss>=11.1697587967[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[16] Batch[5] avg_epoch_
↪loss=11.076884[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=16, batch=5 train loss <loss>=11.0768841108[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[16] Batch [5]#011Speed:␣
↪1164.08 samples/sec#011loss=11.076884[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] processed a total of 318␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 340.
↪06404876708984, "sum": 340.06404876708984, "min": 340.06404876708984}},␣
↪"EndTime": 1575131390.369544, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131390.028989}
[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=934.826960924 records/second[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 17 % of epochs[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=16, train loss <loss>=10.9921466827[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[17] Batch[0] avg_epoch_
↪loss=11.118506[0m
```

```
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=17, batch=0 train loss <loss>=11.1185064316[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[17] Batch[5] avg_epoch_
↪loss=10.985323[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=17, batch=5 train loss <loss>=10.9853231112[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[17] Batch [5]#011Speed:␣
↪1146.54 samples/sec#011loss=10.985323[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] processed a total of 309␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 323.
↪4109878540039, "sum": 323.4109878540039, "min": 323.4109878540039}},␣
↪"EndTime": 1575131390.693494, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131390.369612}
[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=955.139269164 records/second[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 18 % of epochs[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=17, train loss <loss>=11.0881063461[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[18] Batch[0] avg_epoch_
↪loss=10.437140[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=18, batch=0 train loss <loss>=10.4371395111[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[18] Batch[5] avg_epoch_
↪loss=11.002823[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=18, batch=5 train loss <loss>=11.0028233528[0m
[31m[11/30/2019 16:29:50 INFO 140657760761664] Epoch[18] Batch [5]#011Speed:␣
↪1213.98 samples/sec#011loss=11.002823[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] processed a total of 318␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 339.
↪57600593566895, "sum": 339.57600593566895, "min": 339.57600593566895}},␣
↪"EndTime": 1575131391.03361, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131390.693562}
[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=936.147746678 records/second[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 19 % of epochs[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=18, train loss <loss>=10.9492453575[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[19] Batch[0] avg_epoch_
↪loss=11.144054[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=19, batch=0 train loss <loss>=11.1440544128[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[19] Batch[5] avg_epoch_
↪loss=11.232653[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=19, batch=5 train loss <loss>=11.2326534589[0m
```

```
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[19] Batch [5]#011Speed:␣
↪1090.45 samples/sec#011loss=11.232653[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[19] Batch[10] avg_epoch_
↪loss=11.190658[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=19, batch=10 train loss <loss>=11.1402643204[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[19] Batch [10]#011Speed:␣
↪1222.95 samples/sec#011loss=11.140264[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] processed a total of 335␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 382.
↪02691078186035, "sum": 382.02691078186035, "min": 382.02691078186035}},␣
↪"EndTime": 1575131391.416161, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131391.033688}
[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=876.649359433 records/second[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 20 % of epochs[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=19, train loss <loss>=11.1906583959[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[20] Batch[0] avg_epoch_
↪loss=10.614120[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=20, batch=0 train loss <loss>=10.6141204834[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[20] Batch[5] avg_epoch_
↪loss=10.965454[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=20, batch=5 train loss <loss>=10.9654542605[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[20] Batch [5]#011Speed:␣
↪1217.96 samples/sec#011loss=10.965454[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[20] Batch[10] avg_epoch_
↪loss=11.028810[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=20, batch=10 train loss <loss>=11.1048358917[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[20] Batch [10]#011Speed:␣
↪973.56 samples/sec#011loss=11.104836[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] processed a total of 333␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 387.
↪83907890319824, "sum": 387.83907890319824, "min": 387.83907890319824}},␣
↪"EndTime": 1575131391.804478, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131391.416235}
[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=858.351303134 records/second[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 21 % of epochs[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=20, train loss <loss>=11.0288095474[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:51 INFO 140657760761664] Epoch[21] Batch[0] avg_epoch_
↪loss=10.710171[0m
```

```
[31m[11/30/2019 16:29:51 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=21, batch=0 train loss <loss>=10.7101707458[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[21] Batch[5] avg_epoch_
↪loss=10.991527[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=21, batch=5 train loss <loss>=10.9915272395[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[21] Batch [5]#011Speed:␣
↪1129.43 samples/sec#011loss=10.991527[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] processed a total of 303␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 352.
↪0510196685791, "sum": 352.0510196685791, "min": 352.0510196685791}},␣
↪"EndTime": 1575131392.15703, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131391.804556}
[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=860.387525515 records/second[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 22 % of epochs[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=21, train loss <loss>=10.8595946312[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_afd195b9-b81b-481a-b83f-fc3da9cc67d7-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 20.
↪630836486816406, "sum": 20.630836486816406, "min": 20.630836486816406}},␣
↪"EndTime": 1575131392.178267, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131392.15711}
[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[22] Batch[0] avg_epoch_
↪loss=10.801229[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=22, batch=0 train loss <loss>=10.8012285233[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[22] Batch[5] avg_epoch_
↪loss=10.897065[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=22, batch=5 train loss <loss>=10.8970645269[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[22] Batch [5]#011Speed:␣
↪1235.31 samples/sec#011loss=10.897065[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[22] Batch[10] avg_epoch_
↪loss=10.918552[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=22, batch=10 train loss <loss>=10.9443367004[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[22] Batch [10]#011Speed:␣
↪1233.26 samples/sec#011loss=10.944337[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] processed a total of 336␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 355.
↪93295097351074, "sum": 355.93295097351074, "min": 355.93295097351074}},␣
↪"EndTime": 1575131392.534308, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131392.178325}
[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=943.719874562 records/second[0m
```

```
[31m[11/30/2019 16:29:52 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 23 % of epochs[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=22, train loss <loss>=10.9185518785[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[23] Batch[0] avg_epoch_
↪loss=10.825591[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=23, batch=0 train loss <loss>=10.8255910873[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[23] Batch[5] avg_epoch_
↪loss=10.959688[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=23, batch=5 train loss <loss>=10.9596881866[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[23] Batch [5]#011Speed:␣
↪1207.47 samples/sec#011loss=10.959688[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] processed a total of 304␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 330.
↪36303520202637, "sum": 330.36303520202637, "min": 330.36303520202637}},␣
↪"EndTime": 1575131392.865151, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131392.534381}
[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=919.812798202 records/second[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 24 % of epochs[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=23, train loss <loss>=10.7987992287[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_4d952657-a43f-4806-b881-55f084041e01-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 19.
↪411087036132812, "sum": 19.411087036132812, "min": 19.411087036132812}},␣
↪"EndTime": 1575131392.885128, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131392.865255}
[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] Epoch[24] Batch[0] avg_epoch_
↪loss=11.502991[0m
[31m[11/30/2019 16:29:52 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=24, batch=0 train loss <loss>=11.5029907227[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[24] Batch[5] avg_epoch_
↪loss=11.037862[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=24, batch=5 train loss <loss>=11.0378623009[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[24] Batch [5]#011Speed:␣
↪1136.21 samples/sec#011loss=11.037862[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[24] Batch[10] avg_epoch_
↪loss=11.062090[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=24, batch=10 train loss <loss>=11.0911626816[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[24] Batch [10]#011Speed:␣
↪1111.99 samples/sec#011loss=11.091163[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] processed a total of 341␣
↪examples[0m
```

```
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 378.
↪1099319458008, "sum": 378.1099319458008, "min": 378.1099319458008}},␣
↪"EndTime": 1575131393.263366, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131392.885207}
[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=901.606249338 records/second[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 25 % of epochs[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=24, train loss <loss>=11.0620897466[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[25] Batch[0] avg_epoch_
↪loss=10.812670[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=25, batch=0 train loss <loss>=10.812669754[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[25] Batch[5] avg_epoch_
↪loss=10.899194[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=25, batch=5 train loss <loss>=10.8991940816[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[25] Batch [5]#011Speed:␣
↪1145.95 samples/sec#011loss=10.899194[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] processed a total of 312␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 329.
↪3180465698242, "sum": 329.3180465698242, "min": 329.3180465698242}},␣
↪"EndTime": 1575131393.593195, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131393.263439}
[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=947.004459947 records/second[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 26 % of epochs[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=25, train loss <loss>=10.7822431564[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_9b76b505-ec67-40b3-94df-bb134f1035d5-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 19.
↪988059997558594, "sum": 19.988059997558594, "min": 19.988059997558594}},␣
↪"EndTime": 1575131393.613767, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131393.593273}
[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[26] Batch[0] avg_epoch_
↪loss=10.912527[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=26, batch=0 train loss <loss>=10.9125270844[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[26] Batch[5] avg_epoch_
↪loss=10.956731[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=26, batch=5 train loss <loss>=10.9567310015[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[26] Batch [5]#011Speed:␣
↪1206.31 samples/sec#011loss=10.956731[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[26] Batch[10] avg_epoch_
```

```
↪loss=11.084283[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=26, batch=10 train loss <loss>=11.2373447418[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] Epoch[26] Batch [10]#011Speed:␣
↪1168.12 samples/sec#011loss=11.237345[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] processed a total of 323␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 364.
↪0120029449463, "sum": 364.0120029449463, "min": 364.0120029449463}},␣
↪"EndTime": 1575131393.977882, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131393.613821}
[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=887.077150741 records/second[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 27 % of epochs[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=26, train loss <loss>=11.0842827017[0m
[31m[11/30/2019 16:29:53 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[27] Batch[0] avg_epoch_
↪loss=11.405628[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=27, batch=0 train loss <loss>=11.4056282043[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[27] Batch[5] avg_epoch_
↪loss=11.410955[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=27, batch=5 train loss <loss>=11.4109549522[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[27] Batch [5]#011Speed:␣
↪1171.53 samples/sec#011loss=11.410955[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] processed a total of 318␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 337.
↪0828628540039, "sum": 337.0828628540039, "min": 337.0828628540039}},␣
↪"EndTime": 1575131394.315439, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131393.977955}
[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=943.111383263 records/second[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 28 % of epochs[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=27, train loss <loss>=11.2308731079[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[28] Batch[0] avg_epoch_
↪loss=10.775253[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=28, batch=0 train loss <loss>=10.7752532959[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[28] Batch[5] avg_epoch_
↪loss=11.065414[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=28, batch=5 train loss <loss>=11.0654142698[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[28] Batch [5]#011Speed:␣
↪1148.01 samples/sec#011loss=11.065414[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] processed a total of 310␣
```

```
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 341.
↪9628143310547, "sum": 341.9628143310547, "min": 341.9628143310547}},␣
↪"EndTime": 1575131394.657948, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131394.315508}
[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=906.236249471 records/second[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 29 % of epochs[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=28, train loss <loss>=11.0119464874[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[29] Batch[0] avg_epoch_
↪loss=11.090364[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=29, batch=0 train loss <loss>=11.0903635025[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[29] Batch[5] avg_epoch_
↪loss=10.968114[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=29, batch=5 train loss <loss>=10.9681135813[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] Epoch[29] Batch [5]#011Speed:␣
↪1245.21 samples/sec#011loss=10.968114[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] processed a total of 316␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 319.
↪1089630126953, "sum": 319.1089630126953, "min": 319.1089630126953}},␣
↪"EndTime": 1575131394.97759, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131394.658024}
[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=989.922319409 records/second[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 30 % of epochs[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=29, train loss <loss>=10.9914549828[0m
[31m[11/30/2019 16:29:54 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[30] Batch[0] avg_epoch_
↪loss=11.099952[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=30, batch=0 train loss <loss>=11.0999517441[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[30] Batch[5] avg_epoch_
↪loss=10.965332[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=30, batch=5 train loss <loss>=10.9653320312[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[30] Batch [5]#011Speed:␣
↪1226.64 samples/sec#011loss=10.965332[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] processed a total of 293␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 325.
↪0389099121094, "sum": 325.0389099121094, "min": 325.0389099121094}},␣
↪"EndTime": 1575131395.303113, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131394.977663}
[0m
```

```
[31m[11/30/2019 16:29:55 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=901.119879863 records/second[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 31 % of epochs[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=30, train loss <loss>=10.8014466286[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[31] Batch[0] avg_epoch_
↪loss=11.109714[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=31, batch=0 train loss <loss>=11.1097135544[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[31] Batch[5] avg_epoch_
↪loss=11.098782[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=31, batch=5 train loss <loss>=11.0987817446[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[31] Batch [5]#011Speed:␣
↪1119.19 samples/sec#011loss=11.098782[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] processed a total of 309␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 325.
↪37198066711426, "sum": 325.37198066711426, "min": 325.37198066711426}},␣
↪"EndTime": 1575131395.628996, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131395.303189}
[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=949.356007788 records/second[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 32 % of epochs[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=31, train loss <loss>=11.0050726891[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[32] Batch[0] avg_epoch_
↪loss=10.975577[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=32, batch=0 train loss <loss>=10.9755773544[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[32] Batch[5] avg_epoch_
↪loss=10.815983[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=32, batch=5 train loss <loss>=10.8159825007[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Epoch[32] Batch [5]#011Speed:␣
↪1165.20 samples/sec#011loss=10.815983[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] processed a total of 313␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 330.
↪68108558654785, "sum": 330.68108558654785, "min": 330.68108558654785}},␣
↪"EndTime": 1575131395.96022, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131395.629072}
[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=946.214077417 records/second[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 33 % of epochs[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=32, train loss <loss>=10.7815012932[0m
```

```
[31m[11/30/2019 16:29:55 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:55 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_2dd9a9a7-00b8-449a-aa62-2ae9701037b1-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 19.
↪906044006347656, "sum": 19.906044006347656, "min": 19.906044006347656}},␣
↪"EndTime": 1575131395.980681, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131395.960296}
[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[33] Batch[0] avg_epoch_
↪loss=10.885527[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=33, batch=0 train loss <loss>=10.8855266571[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[33] Batch[5] avg_epoch_
↪loss=11.092681[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=33, batch=5 train loss <loss>=11.0926809311[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[33] Batch [5]#011Speed:␣
↪1115.08 samples/sec#011loss=11.092681[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[33] Batch[10] avg_epoch_
↪loss=11.012475[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=33, batch=10 train loss <loss>=10.91622715[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[33] Batch [10]#011Speed:␣
↪1119.30 samples/sec#011loss=10.916227[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] processed a total of 345␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 376.
↪2049674987793, "sum": 376.2049674987793, "min": 376.2049674987793}},␣
↪"EndTime": 1575131396.356996, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131395.980739}
[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=916.768011333 records/second[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 34 % of epochs[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=33, train loss <loss>=11.0124746669[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[34] Batch[0] avg_epoch_
↪loss=10.840313[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=34, batch=0 train loss <loss>=10.8403129578[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[34] Batch[5] avg_epoch_
↪loss=10.774440[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=34, batch=5 train loss <loss>=10.7744396528[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[34] Batch [5]#011Speed:␣
↪1189.46 samples/sec#011loss=10.774440[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[34] Batch[10] avg_epoch_
↪loss=10.920055[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=34, batch=10 train loss <loss>=11.094792366[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[34] Batch [10]#011Speed:␣
↪1196.73 samples/sec#011loss=11.094792[0m
```

```
[31m[11/30/2019 16:29:56 INFO 140657760761664] processed a total of 335␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 352.
↪2989749908447, "sum": 352.2989749908447, "min": 352.2989749908447}},␣
↪"EndTime": 1575131396.709889, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131396.357078}
[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=950.608038168 records/second[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 35 % of epochs[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=34, train loss <loss>=10.9200545224[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[35] Batch[0] avg_epoch_
↪loss=10.686173[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=35, batch=0 train loss <loss>=10.686173439[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[35] Batch[5] avg_epoch_
↪loss=10.848715[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=35, batch=5 train loss <loss>=10.8487146695[0m
[31m[11/30/2019 16:29:56 INFO 140657760761664] Epoch[35] Batch [5]#011Speed:␣
↪1058.96 samples/sec#011loss=10.848715[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[35] Batch[10] avg_epoch_
↪loss=10.778698[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=35, batch=10 train loss <loss>=10.6946788788[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[35] Batch [10]#011Speed:␣
↪1053.95 samples/sec#011loss=10.694679[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] processed a total of 321␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 404.
↪58083152770996, "sum": 404.58083152770996, "min": 404.58083152770996}},␣
↪"EndTime": 1575131397.114952, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131396.70996}
[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=793.207624003 records/second[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 36 % of epochs[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=35, train loss <loss>=10.778698401[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_0116ea36-118a-4989-b7ab-41a84fce5f3d-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 19.
↪61493492126465, "sum": 19.61493492126465, "min": 19.61493492126465}},␣
↪"EndTime": 1575131397.135088, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131397.115025}
[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[36] Batch[0] avg_epoch_
↪loss=10.761388[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
```

```
↪epoch=36, batch=0 train loss <loss>=10.761387825[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[36] Batch[5] avg_epoch_
↪loss=10.424281[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=36, batch=5 train loss <loss>=10.4242806435[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[36] Batch [5]#011Speed:␣
↪1225.75 samples/sec#011loss=10.424281[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[36] Batch[10] avg_epoch_
↪loss=10.717050[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=36, batch=10 train loss <loss>=11.0683725357[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[36] Batch [10]#011Speed:␣
↪1217.64 samples/sec#011loss=11.068373[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] processed a total of 323␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 345.
↪71099281311035, "sum": 345.71099281311035, "min": 345.71099281311035}},␣
↪"EndTime": 1575131397.480908, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131397.135147}
[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=934.01719999 records/second[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 37 % of epochs[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=36, train loss <loss>=10.7170496854[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] best epoch loss so far[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/state_eb6eba5a-a99c-4c25-9f73-934897b081a4-0000.params"[0m
[31m#metrics {"Metrics": {"state.serialize.time": {"count": 1, "max": 19.
↪63210105895996, "sum": 19.63210105895996, "min": 19.63210105895996}},␣
↪"EndTime": 1575131397.501094, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131397.480983}
[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[37] Batch[0] avg_epoch_
↪loss=10.980392[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=37, batch=0 train loss <loss>=10.9803915024[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[37] Batch[5] avg_epoch_
↪loss=10.693951[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=37, batch=5 train loss <loss>=10.6939511299[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[37] Batch [5]#011Speed:␣
↪1228.86 samples/sec#011loss=10.693951[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[37] Batch[10] avg_epoch_
↪loss=10.766361[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=37, batch=10 train loss <loss>=10.8532535553[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[37] Batch [10]#011Speed:␣
↪1114.16 samples/sec#011loss=10.853254[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] processed a total of 346␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 365.476131439209,
↪ "sum": 365.476131439209, "min": 365.476131439209}}, "EndTime": 1575131397.
```

```
↪866699, "Dimensions": {"Host": "algo-1", "Operation": "training",␣
↪"Algorithm": "AWS/DeepAR"}, "StartTime": 1575131397.501155}
[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=946.428097497 records/second[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 38 % of epochs[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=37, train loss <loss>=10.7663613233[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] Epoch[38] Batch[0] avg_epoch_
↪loss=10.943336[0m
[31m[11/30/2019 16:29:57 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=38, batch=0 train loss <loss>=10.9433355331[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[38] Batch[5] avg_epoch_
↪loss=10.928860[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=38, batch=5 train loss <loss>=10.9288597107[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[38] Batch [5]#011Speed:␣
↪1245.15 samples/sec#011loss=10.928860[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[38] Batch[10] avg_epoch_
↪loss=10.751878[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=38, batch=10 train loss <loss>=10.5395008087[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[38] Batch [10]#011Speed:␣
↪1212.10 samples/sec#011loss=10.539501[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] processed a total of 334␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 355.
↪93605041503906, "sum": 355.93605041503906, "min": 355.93605041503906}},␣
↪"EndTime": 1575131398.223124, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131397.866775}
[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=938.076110925 records/second[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 39 % of epochs[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=38, train loss <loss>=10.7518783916[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[39] Batch[0] avg_epoch_
↪loss=10.241104[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=39, batch=0 train loss <loss>=10.241104126[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[39] Batch[5] avg_epoch_
↪loss=10.674151[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=39, batch=5 train loss <loss>=10.6741507848[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[39] Batch [5]#011Speed:␣
↪983.99 samples/sec#011loss=10.674151[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] processed a total of 312␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 375.
↪98586082458496, "sum": 375.98586082458496, "min": 375.98586082458496}},␣
```

```
↪"EndTime": 1575131398.599621, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131398.223193}
[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=829.564361432 records/second[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 40 % of epochs[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=39, train loss <loss>=10.7625462532[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[40] Batch[0] avg_epoch_
↪loss=10.869785[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=40, batch=0 train loss <loss>=10.8697853088[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[40] Batch[5] avg_epoch_
↪loss=10.707974[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=40, batch=5 train loss <loss>=10.7079737981[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[40] Batch [5]#011Speed:␣
↪1014.36 samples/sec#011loss=10.707974[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[40] Batch[10] avg_epoch_
↪loss=10.762103[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=40, batch=10 train loss <loss>=10.8270570755[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] Epoch[40] Batch [10]#011Speed:␣
↪1179.95 samples/sec#011loss=10.827057[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] processed a total of 324␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 385.
↪1900100708008, "sum": 385.1900100708008, "min": 385.1900100708008}},␣
↪"EndTime": 1575131398.985327, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131398.599699}
[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=840.900762962 records/second[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 41 % of epochs[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=40, train loss <loss>=10.7621025606[0m
[31m[11/30/2019 16:29:58 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[41] Batch[0] avg_epoch_
↪loss=11.000257[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=41, batch=0 train loss <loss>=11.0002565384[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[41] Batch[5] avg_epoch_
↪loss=10.934035[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=41, batch=5 train loss <loss>=10.9340354602[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[41] Batch [5]#011Speed:␣
↪1200.30 samples/sec#011loss=10.934035[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] processed a total of 306␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 347.
↪9650020599365, "sum": 347.9650020599365, "min": 347.9650020599365}},␣
```

```
↪"EndTime": 1575131399.333776, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131398.985401}
[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=879.120496626 records/second[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 42 % of epochs[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=41, train loss <loss>=10.8994800568[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[42] Batch[0] avg_epoch_
↪loss=10.722577[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=42, batch=0 train loss <loss>=10.722577095[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[42] Batch[5] avg_epoch_
↪loss=10.752093[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=42, batch=5 train loss <loss>=10.7520933151[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[42] Batch [5]#011Speed:␣
↪1227.94 samples/sec#011loss=10.752093[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] processed a total of 314␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 323.
↪21691513061523, "sum": 323.21691513061523, "min": 323.21691513061523}},␣
↪"EndTime": 1575131399.657482, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131399.333851}
[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=971.18650822 records/second[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 43 % of epochs[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=42, train loss <loss>=10.7396873474[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[43] Batch[0] avg_epoch_
↪loss=10.040716[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=43, batch=0 train loss <loss>=10.0407161713[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[43] Batch[5] avg_epoch_
↪loss=10.671568[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=43, batch=5 train loss <loss>=10.671567599[0m
[31m[11/30/2019 16:29:59 INFO 140657760761664] Epoch[43] Batch [5]#011Speed:␣
↪1073.07 samples/sec#011loss=10.671568[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[43] Batch[10] avg_epoch_
↪loss=10.726690[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=43, batch=10 train loss <loss>=10.7928371429[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[43] Batch [10]#011Speed:␣
↪1078.08 samples/sec#011loss=10.792837[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] processed a total of 342␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 369.
↪65417861938477, "sum": 369.65417861938477, "min": 369.65417861938477}},␣
```

```
↪"EndTime": 1575131400.027668, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131399.657553}
[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=924.914045559 records/second[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 44 % of epochs[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=43, train loss <loss>=10.726690119[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[44] Batch[0] avg_epoch_
↪loss=11.044625[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=44, batch=0 train loss <loss>=11.0446252823[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[44] Batch[5] avg_epoch_
↪loss=10.733619[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=44, batch=5 train loss <loss>=10.733619372[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[44] Batch [5]#011Speed:␣
↪965.37 samples/sec#011loss=10.733619[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[44] Batch[10] avg_epoch_
↪loss=10.951829[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=44, batch=10 train loss <loss>=11.2136800766[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[44] Batch [10]#011Speed:␣
↪1201.88 samples/sec#011loss=11.213680[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] processed a total of 321␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 390.
↪2130126953125, "sum": 390.2130126953125, "min": 390.2130126953125}},␣
↪"EndTime": 1575131400.418395, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131400.027744}
[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=822.391450467 records/second[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 45 % of epochs[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=44, train loss <loss>=10.9518287832[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[45] Batch[0] avg_epoch_
↪loss=10.878129[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=45, batch=0 train loss <loss>=10.8781290054[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[45] Batch[5] avg_epoch_
↪loss=10.760768[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=45, batch=5 train loss <loss>=10.7607679367[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[45] Batch [5]#011Speed:␣
↪1154.22 samples/sec#011loss=10.760768[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] processed a total of 320␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 336.
↪11607551574707, "sum": 336.11607551574707, "min": 336.11607551574707}},␣
```

```
↪"EndTime": 1575131400.75503, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131400.418471}
[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=951.730033682 records/second[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 46 % of epochs[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=45, train loss <loss>=10.7703885078[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[46] Batch[0] avg_epoch_
↪loss=10.351963[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=46, batch=0 train loss <loss>=10.3519630432[0m
[31m[11/30/2019 16:30:00 INFO 140657760761664] Epoch[46] Batch[5] avg_epoch_
↪loss=10.757680[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=46, batch=5 train loss <loss>=10.7576799393[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Epoch[46] Batch [5]#011Speed:␣
↪1133.89 samples/sec#011loss=10.757680[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] processed a total of 319␣
↪examples[0m
[31m#metrics {"Metrics": {"update.time": {"count": 1, "max": 352.
↪22411155700684, "sum": 352.22411155700684, "min": 352.22411155700684}},␣
↪"EndTime": 1575131401.107782, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131400.755107}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #throughput_metric: host=algo-
↪1, train throughput=905.382885068 records/second[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 47 % of epochs[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪epoch=46, train loss <loss>=10.7637651443[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] loss did not improve[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Loading parameters from best␣
↪epoch (36)[0m
[31m#metrics {"Metrics": {"state.deserialize.time": {"count": 1, "max": 8.
↪539915084838867, "sum": 8.539915084838867, "min": 8.539915084838867}},␣
↪"EndTime": 1575131401.116903, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131401.107859}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] stopping training now[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #progress_metric: host=algo-1,␣
↪completed 100 % of epochs[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Final loss: 10.7170496854␣
↪(occurred at epoch 36)[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪train final_loss <loss>=10.7170496854[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Worker algo-1 finished␣
↪training.[0m
[31m[11/30/2019 16:30:01 WARNING 140657760761664] wait_for_all_workers will␣
↪not sync workers since the kv store is not running distributed[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] All workers finished.␣
↪Serializing model for prediction.[0m
```

```
[31m#metrics {"Metrics": {"get_graph.time": {"count": 1, "max": 108.
↪70194435119629, "sum": 108.70194435119629, "min": 108.70194435119629}},␣
↪"EndTime": 1575131401.226341, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131401.116966}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Number of GPUs being used: 0[0m
[31m#metrics {"Metrics": {"finalize.time": {"count": 1, "max": 153.
↪6271572113037, "sum": 153.6271572113037, "min": 153.6271572113037}},␣
↪"EndTime": 1575131401.271223, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131401.226416}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Serializing to /opt/ml/model/
↪model_algo-1[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Saved checkpoint to "/opt/ml/
↪model/model_algo-1-0000.params"[0m
[31m#metrics {"Metrics": {"model.serialize.time": {"count": 1, "max": 7.
↪069826126098633, "sum": 7.069826126098633, "min": 7.069826126098633}},␣
↪"EndTime": 1575131401.278404, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131401.271295}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Successfully serialized the␣
↪model for prediction.[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] Evaluating model accuracy on␣
↪testset using 100 samples[0m
[31m#metrics {"Metrics": {"model.bind.time": {"count": 1, "max": 0.
↪03695487976074219, "sum": 0.03695487976074219, "min": 0.03695487976074219}
↪}, "EndTime": 1575131401.279076, "Dimensions": {"Host": "algo-1",␣
↪"Operation": "training", "Algorithm": "AWS/DeepAR"}, "StartTime":␣
↪1575131401.278445}
[0m
[31m#metrics {"Metrics": {"model.score.time": {"count": 1, "max": 417.
↪0551300048828, "sum": 417.0551300048828, "min": 417.0551300048828}},␣
↪"EndTime": 1575131401.6961, "Dimensions": {"Host": "algo-1", "Operation":␣
↪"training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1575131401.279135}
[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1, RMSE):␣
↪26350.5937694[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1, mean_
↪wQuantileLoss): 0.036173888[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.1]): 0.026595287[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.2]): 0.040024366[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.3]): 0.046109695[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.4]): 0.04796083[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.5]): 0.044627175[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.6]): 0.041932512[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.7]): 0.03474249[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
```

```
↪wQuantileLoss[0.8]): 0.027418833[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #test_score (algo-1,␣
↪wQuantileLoss[0.9]): 0.016153822[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪test mean_wQuantileLoss <loss>=0.0361738875508[0m
[31m[11/30/2019 16:30:01 INFO 140657760761664] #quality_metric: host=algo-1,␣
↪test RMSE <loss>=26350.5937694[0m
[31m#metrics {"Metrics": {"totaltime": {"count": 1, "max": 18092.56100654602,
↪ "sum": 18092.56100654602, "min": 18092.56100654602}, "setuptime":
↪{"count": 1, "max": 9.248971939086914, "sum": 9.248971939086914, "min":␣
↪9.248971939086914}}, "EndTime": 1575131401.711991, "Dimensions": {"Host":␣
↪"algo-1", "Operation": "training", "Algorithm": "AWS/DeepAR"}, "StartTime":␣
↪1575131401.696185}
[0m

2019-11-30 16:30:13 Uploading - Uploading generated training model
2019-11-30 16:30:13 Completed - Training job completed
Training seconds: 66
Billable seconds: 66
```

DeepAR Deep Dive

While the training is happening, Let's elaborate on the DeepAR model's architecture by walking through an example. When interested in quantifying the confidence of the estimates produced, then it's probabilistic forecasts that are wanted. The data we're working with is real-valued, so let's opt for the Gaussian likelihood:

$$\ell(y_t|\mu_t, \sigma_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-(y_t - \mu_t)^2}{2\sigma^2}.$$

θ represents the `parameters of the likelihood`. In the case of Gaussian, θ_t will represent the mean and standard deviation:

$$\theta_t = \{\mu_t, \sigma_t\}.$$

The neural network's last hidden layer results in $h_{d,t}$. This $h_{d,t}$ will undergo 1 activation function per likelihood parameter. For example, for the Gaussian likelihood, $h_{d,t}$ is transformed by an affine activation function to get the mean:

$$\mu_t = w_\mu^T h_{d,t} + b_\mu,$$

$$and then : math : `h` is transformed by a softplus activation to get the$$

standard deviation:

$$\sigma_t = \log\left(1 + \exp(w_\sigma^T h_{d,t} + b_\sigma)\right).$$

The `activation parameters` are the $w_\mu, b_\mu, w_\sigma, b_\sigma$ parameters within the activation functions. The NN is trained to learn the fixed constants of the activation parameters. Since the $h_{d,t}$ output vary given each time-step's input, this still allows the likelihood parameters to vary over time, and therefore capture dynamic behaviors in the time-series data.

From the above diagram, the green input at each time-step is the data point preceding the current time-step's data, as well as the previous network's output. For simplicity, on this diagram we aren't showing covariates which would also be input.

The LSTM layers are shown in red, and the final hidden layer produces the $h_{d,t}$ value, which we saw in the previous slide will undergo an activation function for each parameter of the specified likelihood. To learn the activation function

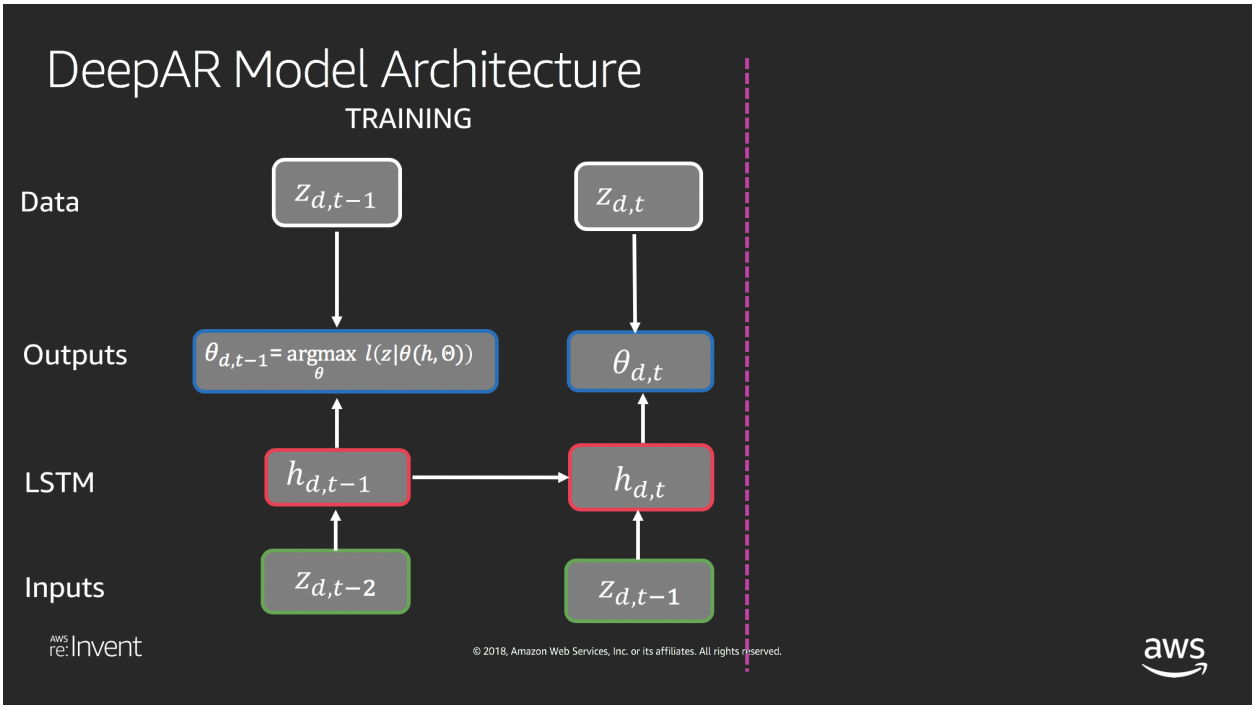


Fig. 2: DeepAR Training

parameters, the NN takes the $h_{d,t}$ at time t and the data up until time t, and performs Stochastic Gradient Descent (SGD) to yield the activation parameters which maximize the likelihood at time t. The blue output layer uses the SGD-optimized activation functions to output the maximum likelihood parameters.

This is how DeepAR trains its model to your data input. Now we want to DeepAR to give us probabilistic forecasts for the next time-step.

The pink line marks our current point in time, divides our training data from data not yet seen. For the first input, it can use the data point of the current time. The input will be processed by the trained LSTM layers, and subsequently get activated by the optimized activation functions to output the maximum-likelihood theta parameters at time $t + 1$.

Now that DeepAR has completed the likelihood with its parameter estimates, DeepAR can simulate `Monte Carlo (MC) samples` from this likelihood and produce an empirical distribution for the predicted datapoint - the probabilistic forecasts shown in purple. The MC samples produced at time $t+1$ are used as input for time $t+2$, etc, until the end of the prediction horizon. In the interactive plots below, we'll see how Monte Carlo samples are able to provide us a confidence interval about the point estimate.

```
%%local
class DeepARPredictor(sagemaker.predictor.RealTimePredictor):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, content_type=sagemaker.content_types.CONTENT_TYPE_
↪JSON, **kwargs)

    def predict(self, ts, cat=None, dynamic_feat=None,
                num_samples=100, return_samples=False, quantiles=["0.1", "0.5", "0.9
↪"]):
        """Requests the prediction of for the time series listed in `ts`, each with␣
↪the (optional)
        corresponding category listed in `cat`.
```

(continues on next page)

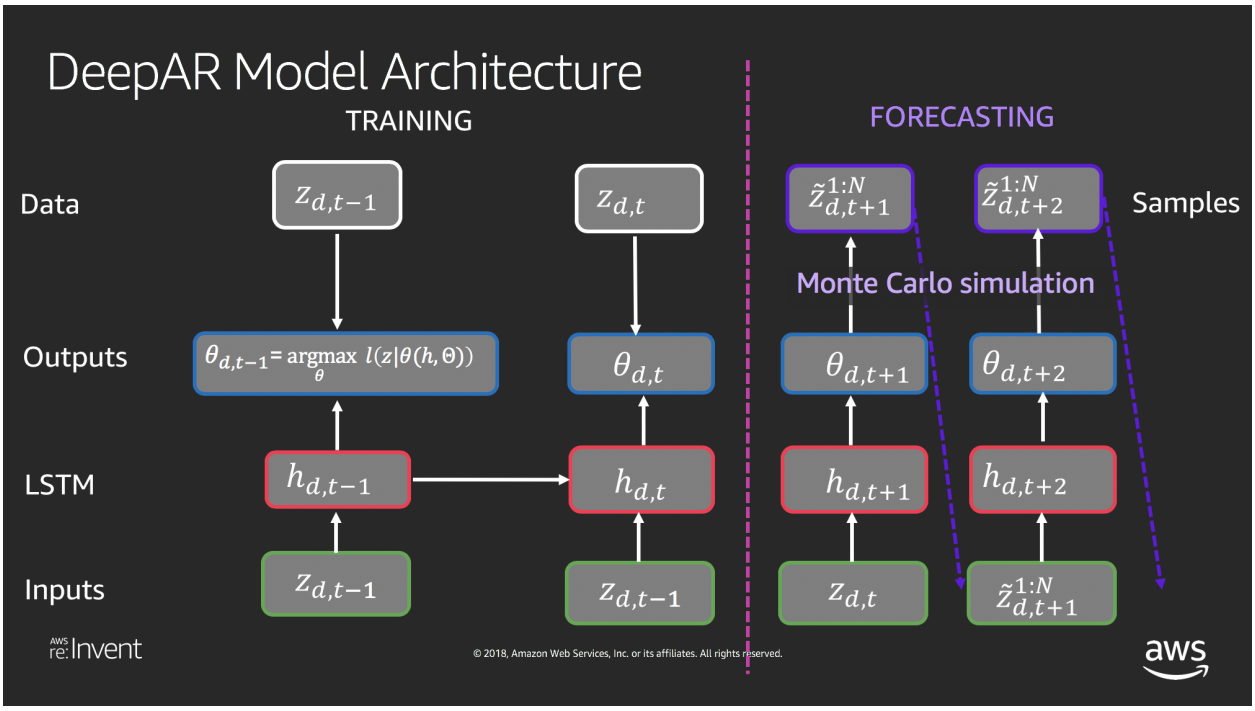


Fig. 3: DeepAR Forecast

(continued from previous page)

```
        ts -- `pandas.Series` object, the time series to predict
        cat -- integer, the group associated to the time series (default: None)
        num_samples -- integer, number of samples to compute at prediction time␣
↪(default: 100)
        return_samples -- boolean indicating whether to include samples in the␣
↪response (default: False)
        quantiles -- list of strings specifying the quantiles to compute (default: [
↪"0.1", "0.5", "0.9"])

        Return value: list of `pandas.DataFrame` objects, each containing the␣
↪predictions
        """
        prediction_time = ts.index[-1] + 1
        quantiles = [str(q) for q in quantiles]
        req = self.__encode_request(ts, cat, dynamic_feat, num_samples, return_
↪samples, quantiles)
        res = super(DeepARPredictor, self).predict(req)
        return self.__decode_response(res, ts.index.freq, prediction_time, return_
↪samples)

    def __encode_request(self, ts, cat, dynamic_feat, num_samples, return_samples,␣
↪quantiles):
        instance = series_to_dict(ts, cat if cat is not None else None, dynamic_feat␣
↪if dynamic_feat else None)

        configuration = {
            "num_samples": num_samples,
            "output_types": ["quantiles", "samples"] if return_samples else [
↪"quantiles"],
```

(continues on next page)

(continued from previous page)

```
            "quantiles": quantiles
        }

        http_request_data = {
            "instances": [instance],
            "configuration": configuration
        }

        return json.dumps(http_request_data).encode('utf-8')

    def __decode_response(self, response, freq, prediction_time, return_samples):
        # we only sent one time series so we only receive one in return
        # however, if possible one will pass multiple time series as predictions will␣
↪then be faster
        predictions = json.loads(response.decode('utf-8'))['predictions'][0]
        prediction_length = len(next(iter(predictions['quantiles'].values())))
        prediction_index = pd.DatetimeIndex(start=prediction_time, freq=freq,␣
↪periods=prediction_length)
        if return_samples:
            dict_of_samples = {'sample_' + str(i): s for i, s in␣
↪enumerate(predictions['samples'])}
        else:
            dict_of_samples = {}
        return pd.DataFrame(data={**predictions['quantiles'], **dict_of_samples},␣
↪index=prediction_index)

    def set_frequency(self, freq):
        self.freq = freq

def encode_target(ts):
    return [x if np.isfinite(x) else "NaN" for x in ts]

def series_to_dict(ts, cat=None, dynamic_feat=None):
    """Given a pandas.Series object, returns a dictionary encoding the time series.

    ts -- a pands.Series object with the target time series
    cat -- an integer indicating the time series category

    Return value: a dictionary
    """
    obj = {"start": str(ts.index[0]), "target": encode_target(ts)}
    if cat is not None:
        obj["cat"] = cat
    if dynamic_feat is not None:
        obj["dynamic_feat"] = dynamic_feat
    return obj
```

Deploying a realtime predictor

Next we will deploy a predictor, this may take a few minutes

```
%%local
predictor = estimator.deploy(
    initial_instance_count=1,
```

(continues on next page)

(continued from previous page)

```
    instance_type='ml.m4.xlarge',
    predictor_cls=DeepARPredictor
)
```

```
--------------------------------------------------------------------------------
↪!
```

Running Predictions on the Endpoint

```
%%local
ABB = full_timeseries.asfreq('d')
print('Green Rides:')
print(predictor.predict(ts=ABB.loc[end_training:, 'green'], quantiles=[0.10, 0.5, 0.
↪90], num_samples=100).head())
print('\nYellow Rides:')
print(predictor.predict(ts=ABB.loc[end_training:, 'yellow'], quantiles=[0.10, 0.5, 0.
↪90], num_samples=100).head())
print('\nFHV Rides:')
print(predictor.predict(ts=ABB.loc[end_training:, 'full_fhv'], quantiles=[0.10, 0.5,␣
↪0.90], num_samples=100).head())
```

```
Green Rides:
                    0.1           0.5           0.9
2019-07-01   9721.018555  13791.329102  17538.664062
2019-07-02  11092.904297  14636.870117  17553.996094
2019-07-03  11470.556641  14888.299805  18519.500000
2019-07-04  11351.886719  15973.585938  20251.890625
2019-07-05  13768.468750  16813.535156  20389.726562

Yellow Rides:
                     0.1            0.5            0.9
2019-07-01  159293.500000  194948.109375  227773.562500
2019-07-02  184475.156250  215566.890625  241038.796875
2019-07-03  203786.625000  233359.468750  265797.250000
2019-07-04  210038.375000  249315.265625  285376.531250
2019-07-05  225598.546875  251787.218750  281431.187500

FHV Rides:
                   0.1          0.5          0.9
2019-07-01  585261.7500  675419.6250  758377.4375
2019-07-02  601097.9375  668050.7500  754662.2500
2019-07-03  638367.8750  716293.1875  772745.3125
2019-07-04  703654.0625  780941.1875  852240.3750
2019-07-05  768018.5625  832646.1875  923119.3750
```

```
%%local
endpoint_name = predictor.endpoint

%store ABB
%store endpoint_name
%store end_training
%store prediction_length
```

```
Stored 'ABB' (DataFrame)
Stored 'endpoint_name' (str)
Stored 'end_training' (Timestamp)
Stored 'prediction_length' (int)
```

We'll show you in the next notebook, how to recreate the predictor and evaluate the results more.

Examine notebook used to visualize results

First we will load the endpoint name, training time, prediction length and seom of the data

```
%store -r
print('endpoint name ', endpoint_name)
print('end training', end_training)
print('prediction_length', prediction_length)
```

```
endpoint name  DeepAR-forecast-taxidata-2019-11-30-16-26-45-053
end training 2019-05-06 00:00:00
prediction_length 14
```

Sample data being used:

```
print('data sample')
ABB.head(5)
```

```
data sample
```

This next cell creates the predictor using the endpoint_name. Ideally we'd have the DeepARPredictor in a seperate .py rather than repeated in the two notebooks.

```
import sagemaker
from sagemaker import get_execution_role
from sagemaker.tuner import HyperparameterTuner
import numpy as np
import json
import pandas as pd

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

class DeepARPredictor(sagemaker.predictor.RealTimePredictor):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, content_type=sagemaker.content_types.CONTENT_TYPE_
↪JSON, **kwargs)

    def predict(self, ts, cat=None, dynamic_feat=None,
                num_samples=100, return_samples=False, quantiles=["0.1", "0.5", "0.9
↪"]):
        """Requests the prediction of for the time series listed in `ts`, each with␣
↪the (optional)
        corresponding category listed in `cat`.
```

(continues on next page)

(continued from previous page)

```
        ts -- `pandas.Series` object, the time series to predict
        cat -- integer, the group associated to the time series (default: None)
        num_samples -- integer, number of samples to compute at prediction time␣
↪(default: 100)
        return_samples -- boolean indicating whether to include samples in the␣
↪response (default: False)
        quantiles -- list of strings specifying the quantiles to compute (default: [
↪"0.1", "0.5", "0.9"])

        Return value: list of `pandas.DataFrame` objects, each containing the␣
↪predictions
        """
        prediction_time = ts.index[-1] + 1
        quantiles = [str(q) for q in quantiles]
        req = self.__encode_request(ts, cat, dynamic_feat, num_samples, return_
↪samples, quantiles)
        res = super(DeepARPredictor, self).predict(req)
        return self.__decode_response(res, ts.index.freq, prediction_time, return_
↪samples)

    def __encode_request(self, ts, cat, dynamic_feat, num_samples, return_samples,␣
↪quantiles):
        instance = series_to_dict(ts, cat if cat is not None else None, dynamic_feat␣
↪if dynamic_feat else None)

        configuration = {
            "num_samples": num_samples,
            "output_types": ["quantiles", "samples"] if return_samples else [
↪"quantiles"],
            "quantiles": quantiles
        }

        http_request_data = {
            "instances": [instance],
            "configuration": configuration
        }

        return json.dumps(http_request_data).encode('utf-8')

    def __decode_response(self, response, freq, prediction_time, return_samples):
        # we only sent one time series so we only receive one in return
        # however, if possible one will pass multiple time series as predictions will␣
↪then be faster
        predictions = json.loads(response.decode('utf-8'))['predictions'][0]
        prediction_length = len(next(iter(predictions['quantiles'].values())))
        prediction_index = pd.DatetimeIndex(start=prediction_time, freq=freq,␣
↪periods=prediction_length)
        if return_samples:
            dict_of_samples = {'sample_' + str(i): s for i, s in␣
↪enumerate(predictions['samples'])}
        else:
            dict_of_samples = {}
        return pd.DataFrame(data={**predictions['quantiles'], **dict_of_samples},␣
↪index=prediction_index)

    def set_frequency(self, freq):
```

(continues on next page)

(continued from previous page)

```
        self.freq = freq

def encode_target(ts):
    return [x if np.isfinite(x) else "NaN" for x in ts]

def series_to_dict(ts, cat=None, dynamic_feat=None):
    """Given a pandas.Series object, returns a dictionary encoding the time series.

    ts -- a pands.Series object with the target time series
    cat -- an integer indicating the time series category

    Return value: a dictionary
    """
    obj = {"start": str(ts.index[0]), "target": encode_target(ts)}
    if cat is not None:
        obj["cat"] = cat
    if dynamic_feat is not None:
        obj["dynamic_feat"] = dynamic_feat
    return obj

predictor = DeepARPredictor(endpoint_name)
```

```
import matplotlib
import matplotlib.pyplot as plt

def plot(
    predictor,
    target_ts,
    cat=None,
    dynamic_feat=None,
    forecast_date=end_training,
    show_samples=False,
    plot_history=7 * 12,
    confidence=80,
    num_samples=100,
    draw_color='blue'
):
    print("Calling endpoint to generate {} predictions starting from {} ...".
↪format(target_ts.name, str(forecast_date)))
    assert(confidence > 50 and confidence < 100)
    low_quantile = 0.5 - confidence * 0.005
    up_quantile = confidence * 0.005 + 0.5

    # we first construct the argument to call our model
    args = {
        "ts": target_ts[:forecast_date],
        "return_samples": show_samples,
        "quantiles": [low_quantile, 0.5, up_quantile],
        "num_samples": num_samples
    }


    if dynamic_feat is not None:
        args["dynamic_feat"] = dynamic_feat
        fig = plt.figure(figsize=(20, 6))
        ax = plt.subplot(2, 1, 1)
```

(continues on next page)

(continued from previous page)

```
    else:
        fig = plt.figure(figsize=(20, 3))
        ax = plt.subplot(1,1,1)

    if cat is not None:
        args["cat"] = cat
        ax.text(0.9, 0.9, 'cat = {}'.format(cat), transform=ax.transAxes)

    # call the end point to get the prediction
    prediction = predictor.predict(**args)

    # plot the samples
    mccolor = draw_color
    if show_samples:
        for key in prediction.keys():
            if "sample" in key:
                prediction[key].asfreq('D').plot(color='lightskyblue', alpha=0.2,␣
↪label='_nolegend_')

    # the date didn't have a frequency in it, so setting it here.
    new_date = pd.Timestamp(forecast_date, freq='d')
    target_section = target_ts[new_date-plot_history:new_date+prediction_length]
    target_section.asfreq('D').plot(color="black", label='target')
    plt.title(target_ts.name.upper(), color='darkred')

    # plot the confidence interval and the median predicted
    ax.fill_between(
        prediction[str(low_quantile)].index,
        prediction[str(low_quantile)].values,
        prediction[str(up_quantile)].values,
        color=mccolor, alpha=0.3, label='{}% confidence interval'.format(confidence)
    )
    prediction["0.5"].plot(color=mccolor, label='P50')
    ax.legend(loc=2)

    # fix the scale as the samples may change it
    ax.set_ylim(target_section.min() * 0.5, target_section.max() * 1.5)

    if dynamic_feat is not None:
        for i, f in enumerate(dynamic_feat, start=1):
            ax = plt.subplot(len(dynamic_feat) * 2, 1, len(dynamic_feat) + i,␣
↪sharex=ax)
            feat_ts = pd.Series(
                index=pd.DatetimeIndex(start=target_ts.index[0], freq=target_ts.index.
↪freq, periods=len(f)),
                data=f
            )
            feat_ts[forecast_date-plot_history:forecast_date+prediction_length].
↪plot(ax=ax, color='g')
```

Let's interact w/ the samples and forecast values now.

```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
```

(continues on next page)

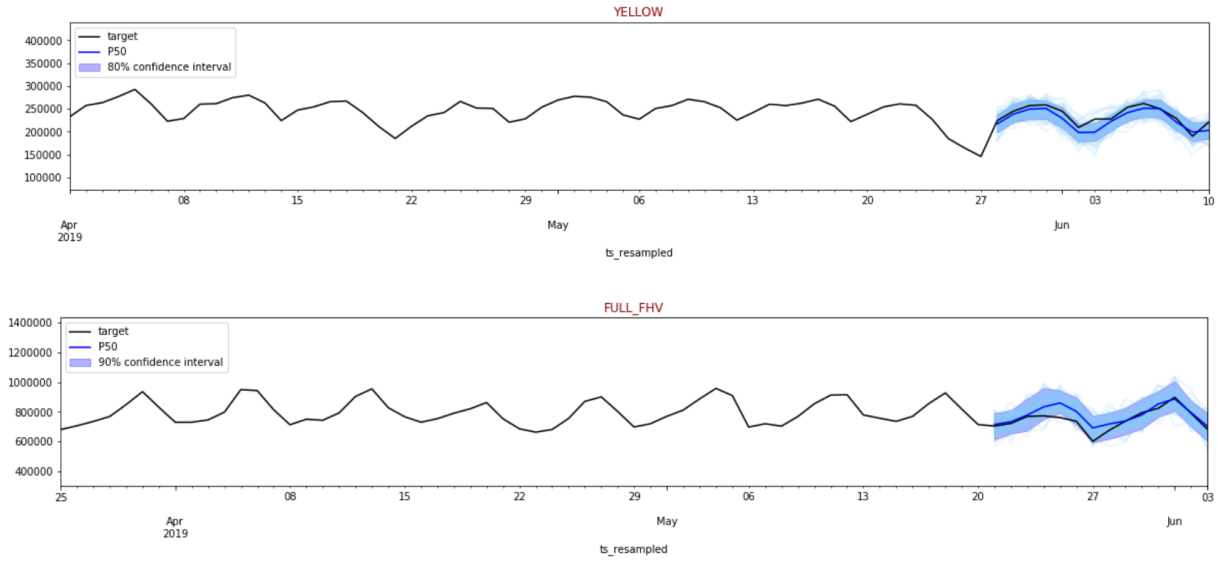
(continued from previous page)

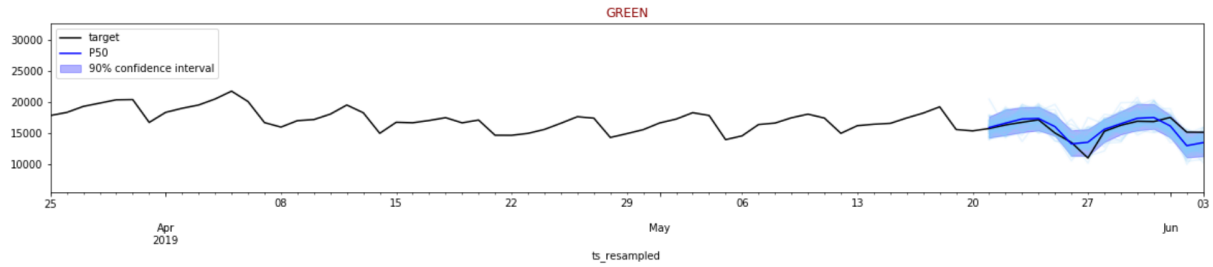
```
import ipywidgets as widgets
from ipywidgets import IntSlider, FloatSlider, Checkbox, RadioButtons
import datetime

style = {'description_width': 'initial'}

@interact_manual(
    series_type=RadioButtons(options=['full_fhv', 'yellow', 'green'], value='yellow',␣
↪description='Type'),
    forecast_day=IntSlider(min=0, max=100, value=21, style=style),
    confidence=IntSlider(min=60, max=95, value=80, step=5, style=style),
    history_weeks_plot=IntSlider(min=1, max=20, value=4, style=style),
    num_samples=IntSlider(min=100, max=1000, value=100, step=500, style=style),
    show_samples=Checkbox(value=True),
    continuous_update=False
)

def plot_interact(series_type, forecast_day, confidence, history_weeks_plot, show_
↪samples, num_samples):
    plot(
        predictor,
        target_ts=ABB[series_type].asfreq(freq='d', fill_value=0),
        forecast_date=end_training + datetime.timedelta(days=forecast_day),
        show_samples=show_samples,
        plot_history=history_weeks_plot * prediction_length,
        confidence=confidence,
        num_samples=num_samples
    )
```

```
interactive(children=(RadioButtons(description='Type', index=1, options=('full_fhv',
↪'yellow', 'green'), value...
```

Testing and Understanding the results

You can test the results and see different results. Here are some examples below:





Running the Step Functions inference workflow

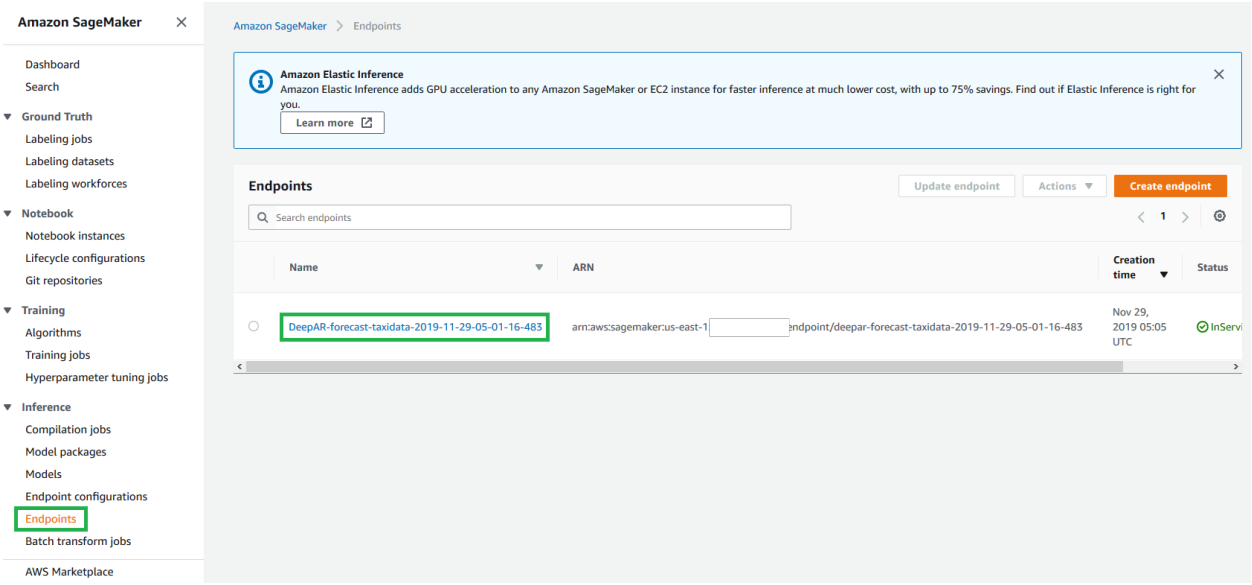
We're going to use AWS Step Functions to automate a workflow that will invoke our model endpoint and send and text us with the results. This is to exemplify that we can get notified about how our inferences are performing.

Activity 1: Update our Lambda function with the model endpoint name

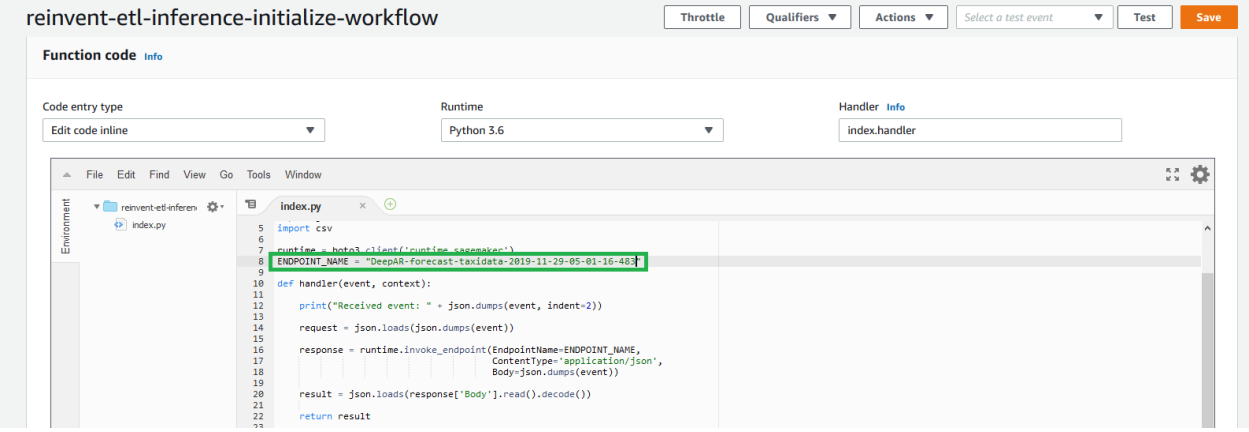
A Lambda function has been created before the workshop, which will invoke our model endpoint

Steps:

1. In the AWS Console, click **Services** in the top, left-hand corner of the screen
2. Type **SageMaker** into the search field and hit Enter
3. Select **Endpoints** from the menu on the left-hand side of the screen (see screenshot below)



1. Copy the name of the endpoint to a text editor
2. Next, click **Services** in the top, left-hand corner of the screen
3. Type **Lambda** into the search field and hit Enter
4. Click on the lambda function name ("**reinvent-etl-inference-initialize-workflow**")
5. Scroll down to the **Function code** area, and replace the value of the **ENDPOINT_NAME* variable with the name of the endpoint that you copied from the SageMaker console (see screenshot below)



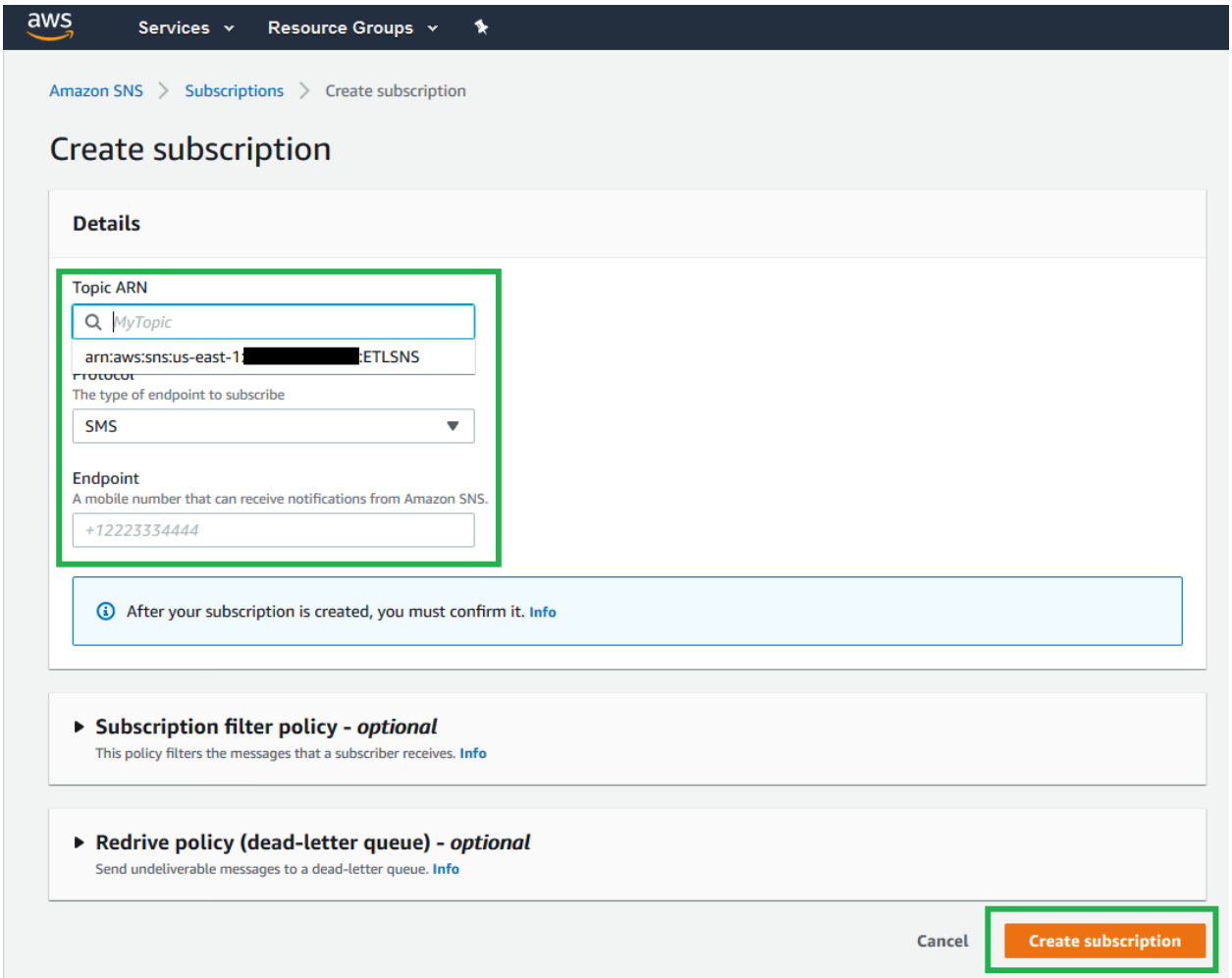
1. Click **Save** in the top, right-hand corner of the screen

Activity 2: Subscribe to the SNS topic

An SNS topic for our workflow has also been created before the workshop, and you can subscribe your cell-phone number or email address to this topic to receive a notification with the model inference results. (We recommend using the SMS option and your cell-phone number, because the email method requires verifying your email address, which takes longer).

Steps:

1. In the AWS Console, click **Services** in the top, left-hand corner of the screen
2. Type **SNS** into the search field and hit Enter
3. Select **Subscriptions** from the menu on the left-hand side of the screen
4. Click **Create subscription**
5. In the **Topic ARN** field, select the ETLSNS topic ARN (see screenshot below)



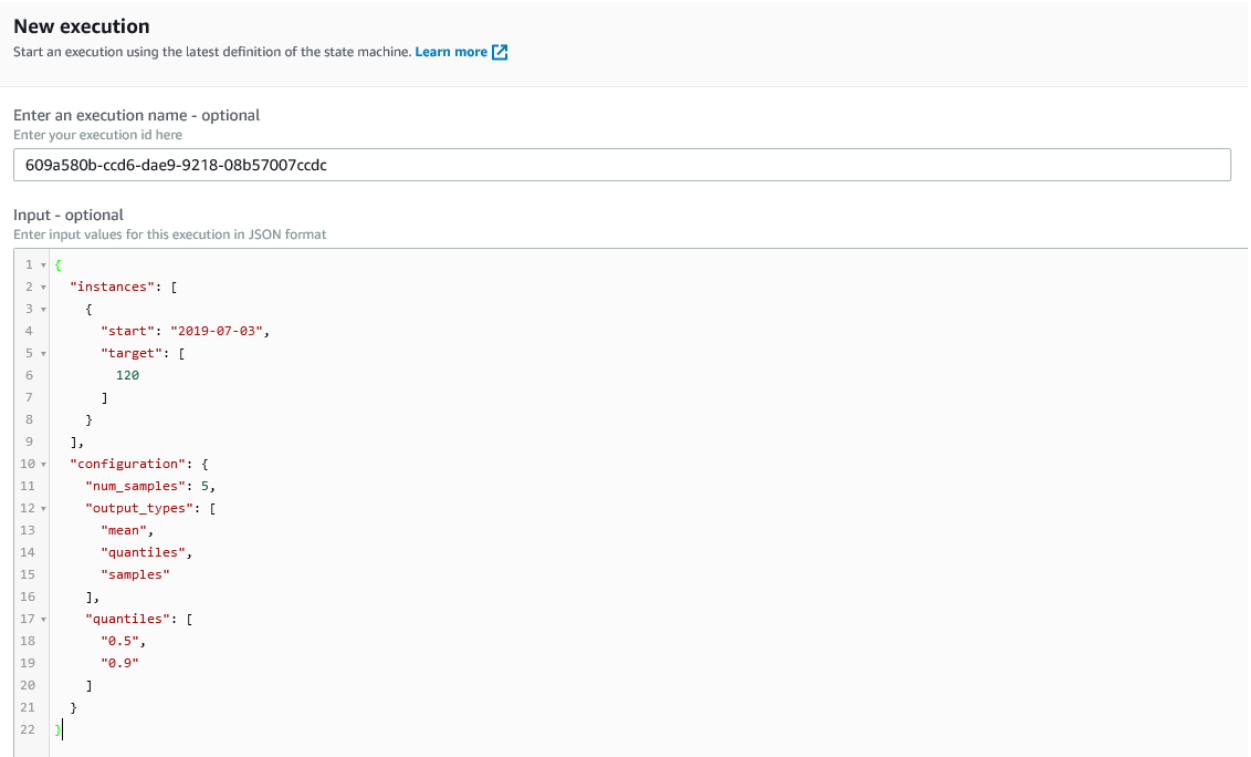
1. In the **Protocol** field, select SMS
2. In the **Endpoint** field, enter your cell-phone number (note that these accounts and all data stored within them will be deleted directly after this workshop, so all of these details will be deleted)
3. Click **Create subscription**

Activity 3: Let's test our workflow!

Steps:

1. In the AWS Console, click **Services** in the top, left-hand corner of the screen
2. Type **Step Functions** into the search field and hit Enter
3. Click **State machines** in the top, right-hand corner of the screen.
4. Click on the **reinvent-etl-inference-workflow** state machine
5. Click **Start execution**
6. Paste the following json text into the input field (see screenshot below), and click **Start execution**. Note the version of the dataset that we are using for training, as denoted by the "DataDate" parameter: .. code-block:

```
{
"instances": [
 {
   "start": "2019-07-03",
   "target": [
     120
   ]
 }
],
"configuration": {
 "num_samples": 5,
 "output_types": [
   "mean",
   "quantiles",
   "samples"
 ],
 "quantiles": [
   "0.5",
   "0.9"
 ]
}
}
```



1. Now we can watch the workflow progress through each of the states. Be sure to to inspect the inputs and outputs of each state, and you should receive an SMS on your cell-phone with the inference results when it completes!