databricks latest

(https://databricks.com)

Qualcomm Stock Prediction using RNNs and LSTM

```
# Installing necessary libraries
%pip install numpy
%pip install tensorflow
```

```
Python interpreter will be restarted.
Requirement already satisfied: numpy in /databricks/python3/lib/python3.9/site
-packages (1.21.5)
Python interpreter will be restarted.
Python interpreter will be restarted.
Collecting tensorflow
  Using cached tensorflow-2.13.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (524.1 MB)
Collecting termcolor>=1.1.0
  Using cached termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in /databricks/
python3/lib/python3.9/site-packages (from tensorflow) (4.1.1)
Requirement already satisfied: setuptools in /databricks/python3/lib/python3.
9/site-packages (from tensorflow) (61.2.0)
Collecting flatbuffers>=23.1.21
  Using cached flatbuffers-23.5.26-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: packaging in /databricks/python3/lib/python3.9/
site-packages (from tensorflow) (21.3)
Collecting astunparse>=1.6.0
  Using cached astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Collecting grpcio<2.0,>=1.24.3
```

```
%pip install yfinance
```

```
Python interpreter will be restarted.
Collecting yfinance
  Using cached yfinance-0.2.26-py2.py3-none-any.whl (62 kB)
Collecting appdirs>=1.4.4
  Using cached appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Collecting lxml>=4.9.1
  Using cached lxml-4.9.3-cp39-cp39-manylinux_2_28_x86_64.whl (8.0 MB)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /databricks/python3/l
ib/python3.9/site-packages (from yfinance) (4.11.1)
```

```
Collecting pytz>=2022.5
  Using cached pytz-2023.3-py2.py3-none-any.whl (502 kB)
Requirement already satisfied: numpy>=1.16.5 in /local_disk0/.ephemeral_nfs/en
vs/pythonEnv-24e9da4a-2568-43aa-a122-653f235c23fb/lib/python3.9/site-packages
(from yfinance) (1.24.3)
Collecting html5lib>=1.1
  Using cached html5lib-1.1-py2.py3-none-any.whl (112 kB)
Requirement already satisfied: pandas>=1.3.0 in /databricks/python3/lib/python
3.9/site-packages (from yfinance) (1.4.2)
Collecting multitasking>=0.0.7
  Using cached multitasking-0.0.11-py3-none-any.whl (8.5 kB)
```

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from keras import backend
from keras import metrics
```

```
/databricks/python/lib/python3.9/site-packages/scipy/__init__.py:146: UserWarn
ing: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciP
y (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
# Function for Data Preprocessing
#def preprocess_data(data, steps):
steps=60
# Read data
#qcomData = spark.read.option("header", "true").option("inferSchema",
"true").csv('dbfs:/FileStore/tables/QCOM.csv')
import yfinance as yf
data = yf.download("QCOM", start="2018-07-16", end="2023-07-16")

data = data.reset_index()
qcomData = spark.createDataFrame(data)
#Selecting only relevant columns
close_data = qcomData.select('Close').collect()

#MinMax Scaling
mmscale = MinMaxScaler(feature_range=(0, 1))
mmscaled_data = mmscale.fit_transform(close_data)

#New dataset for training
train_size = int(len(close_data) * 0.8)
train_data = mmscaled_data[0:train_size]

train_x, train_y = [], []
for i in range(train_size - steps):
    train_x.append(train_data[i:i + steps, 0])
    train_y.append(train_data[i + steps, 0])

train_x, train_y = np.array(train_x), np.array(train_y)
train_x = np.reshape(train_x, (train_size - steps, steps, 1))

# Define the number of time steps for LSTM
steps = 60
```

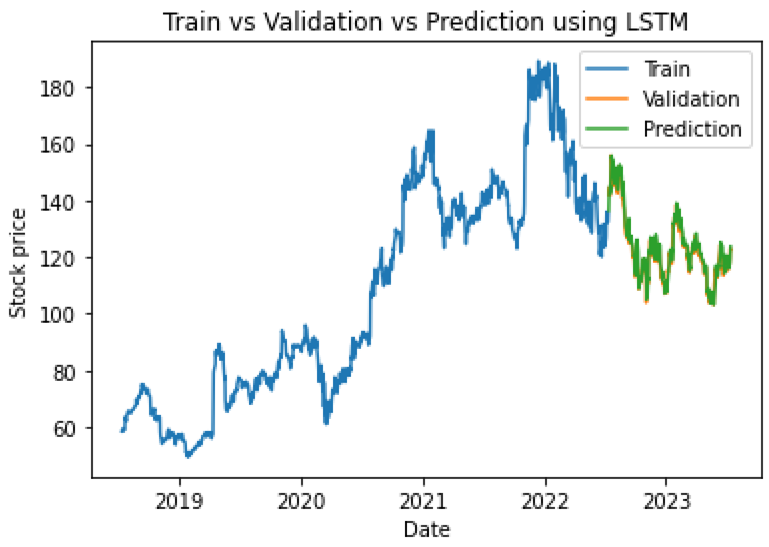
```
[*********************100%***********************]  1 of 1 completed
```

```
# Build LSTM Model
def build_lstm_model(steps):
    model = Sequential()
    model.add(LSTM(50, input_shape=(steps, 1), return_sequences=True))
    model.add(LSTM(50, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error',
                  metrics=['mean_squared_error', 'mean_absolute_error',
                           'mean_absolute_percentage_error', root_mean])
    return model


# Root Mean Squared Error function
def root_mean(actual_y, predicted_y):
    return backend.sqrt(backend.mean(backend.square(predicted_y - actual_y)))


# Build the LSTM model
lstm_model = build_lstm_model(steps)

# Fit the model

batch_1=lstm_model.fit(train_x, train_y, batch_size=1, epochs=40,verbose=True)
batch_2=lstm_model.fit(train_x, train_y, batch_size=6, epochs=40,verbose=True)
batch_3=lstm_model.fit(train_x, train_y, batch_size=11, epochs=40,verbose=True)
```

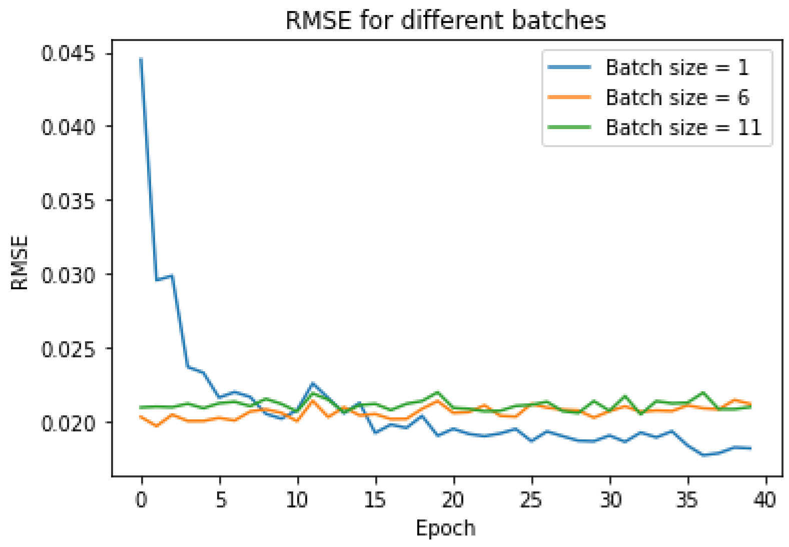
```
Epoch 1/40
112/946 [==>...........................] - ETA: 34s - loss: 0.0108 - mean_squa
red_error: 0.0108 - mean_absolute_error: 0.0728 - mean_absolute_percentage_err
or: 26.7865 - root_mean: 0.0731

*** WARNING: max output size exceeded, skipping output. ***

86/86 [==============================] - 5s 59ms/step - loss: 5.0175e-04 - mea
n_squared_error: 5.0175e-04 - mean_absolute_error: 0.0154 - mean_absolute_perc
entage_error: 12049.9473 - root_mean: 0.0210
```

```
# Testing Dataset for X
test_data = mmscaled_data[train_size-steps:]
x_test = []
x_test = [test_data[i:i + steps, 0] for i in range(len(test_data) - steps)]
x_test = np.array(x_test)
x_test = np.reshape(x_test, (len(test_data) - steps, steps, 1))

# Testing Dataset for Y
y_test = close_data[train_size:]

# Predicting values
predictions = lstm_model.predict(x_test)
predictions = mmscale.inverse_transform(predictions)

# Calculate RMSE
rmse_val = np.sqrt(np.mean(((predictions - y_test) ** 2)))
print("Root Mean Square Error:", rmse_val)

# Plot the results
plt.title('Train vs Validation vs Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('Stock price')
plt.plot(qcomData.select('Date').collect()[:train_size],
qcomData.select('Close').collect()[:train_size])
plt.plot(qcomData.select('Date').collect()[train_size:],
qcomData.select('Close').collect()[train_size:])
plt.plot(qcomData.select('Date').collect()[train_size:], predictions)
plt.legend(['Train', 'Validation', 'Prediction'])
plt.show()
```

```
8/8 [==============================] - 1s 17ms/step
Root Mean Square Error: 3.066162970386304
```



```
plt.title('RMSE for different batches')
plt.plot(batch_1.history['root_mean'])
plt.plot(batch_2.history['root_mean'])
plt.plot(batch_3.history['root_mean'])
plt.xlabel('Epoch')
plt.ylabel('RMSE')
plt.legend(['Batch size = 1','Batch size = 6','Batch size = 11'])
plt.show()
```



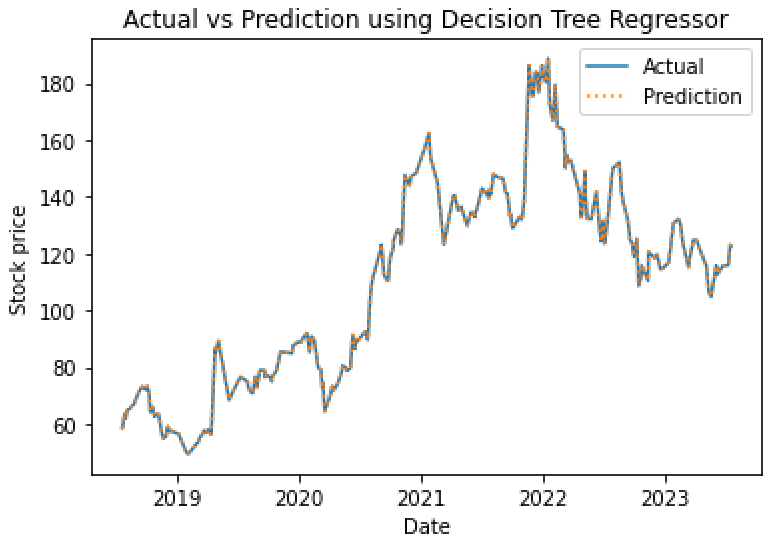
Prediction using Decision Tree Regressor

```
from __future__ import print_function

from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
# $example on$
from pyspark.ml.feature import OneHotEncoder, VectorAssembler, StringIndexer
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window

#qcomData = spark.read.option("header", "true").option("inferSchema",
"true").csv('dbfs:/FileStore/tables/QCOM.csv')
data = yf.download("QCOM", start="2018-07-16", end="2023-07-16")

data = data.reset_index()
qcomData = spark.createDataFrame(data)
w = Window.partitionBy().orderBy("Date")

qcomData = qcomData.withColumn('diffOpenClose', qcomData.Open - qcomData.Close)
qcomData = qcomData.withColumn('diffHighLow', qcomData.High - qcomData.Low)
qcomData = qcomData.withColumn('target', F.when(F.lag(qcomData.Close).over(w) <
qcomData.Close, 'yes').otherwise('no'))
qcomData.drop('Date')
cat_Columns = ['High', 'Low', 'Open', 'Close','Adj Close','Volume']
stages = []

for cat_Col in cat_Columns:
    stringIndexer = StringIndexer(inputCol=cat_Col, outputCol=cat_Col +
'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],
outputCols=[cat_Col + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol='target', outputCol='label')
stages += [label_stringIdx]

assembler = VectorAssembler(inputCols=[c + "classVec" for c in cat_Columns],
outputCol="features")
stages += [assembler]

pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(qcomData)
qcomData = pipelineModel.transform(qcomData)

qcomData.select('Close', 'label', 'features').show()
```

```
(trainingData, testData) = qcomData.randomSplit([0.8, 0.2])

dr = DecisionTreeRegressor(labelCol="label", featuresCol="features")

model = dr.fit(trainingData)
predictions = model.transform(testData)

evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
[*********************100%***********************]  1 of 1 completed
+------------------+-----+--------------------+
|             Close|label|            features|
+------------------+-----+--------------------+
|58.349998474121094|  1.0|(7219,[822,2011,3...|
| 58.90999984741211|  0.0|(7219,[823,2005,3...|
|  58.7599983215332|  1.0|(7219,[44,2012,31...|
|59.310001373291016|  0.0|(7219,[829,2014,3...|
| 58.61000061035156|  1.0|(7219,[825,1236,3...|
| 59.08000183105469|  0.0|(7219,[826,2009,3...|
|58.849998474121094|  1.0|(7219,[830,2013,3...|
| 59.41999816894531|  0.0|(7219,[828,2007,3...|
| 63.58000183105469|  0.0|(7219,[840,2019,3...|
|62.689998626708984|  1.0|(7219,[834,2026,3...|
|62.040000915527344|  1.0|(7219,[833,2021,3...|
| 64.08999633789062|  0.0|(7219,[846,2037,3...|
|  64.3499984741211|  0.0|(7219,[848,2038,3...|
|  64.7699966430664|  0.0|(7219,[850,2041,3...|
|  65.4000015258789|  0.0|(7219,[856,2047,3...|
|  65.7300033569336|  0.0|(7219,[861,2057,2...|
| 65.44000244140625|  1.0|(7219,[860,2054,3...|
```

```
#Reformatting data for Plotting

from pyspark.sql.functions import to_date
from pyspark.sql import Row
from pyspark.sql.functions import col
from pyspark.sql.types import FloatType
testData = testData.withColumn("Date_d", to_date("Date", "yyyy-MM-dd"))
predictions = predictions.withColumn("Date_d", to_date("Date", "yyyy-MM-dd"))

testData = testData.withColumn("Close_f", col("Close").cast(FloatType()))
predictions = predictions.withColumn("Close_f", col("Close").cast(FloatType()))

test_size = testData.count()
```


Actual vs Prediction using Decision Tree Regressor
Actual
Prediction
Stock price
180
160
140
120
100
80
60
2019
2020
2021
2022
2023
Date