Group Assignment Report

Group Members:

Bruno Simione Beltrame
Nilton Antonio De Oliveira Paes
Fabio Cesar Torres De Araujo Hercules
Oleksii Kachan
Francisco Isaac De La Rosa Rivero

Dataset of Stock Market in the S&P 500 index which shows the information of all stock available in the index from Feb 2013 to Feb 2018

- Date – day of when the data was taken
- Open – the initial value of the stock
- High – the highest value of the day
- Low – the lowest value of the day
- Close – the closing value of the stock
- Volume – total amount of trades
- Name – stock name

Source of dataset https://www.kaggle.com/camnugent/sandp500

Although we can get all the different stocks in the index we decided to predict the closing value of Amazon (AMZN) taking the open value with the max and min of the day.

First we take only the AMZN's stock we create two files one for the training and one for the testing

We load the dataset to train and we take the values to train and to predict in this case is the close column

```
training_data_df =
pd.read_csv("/dbfs/FileStore/tables/all_stocks_5yr_amazon_train.csv",
dtype=float)
X_training = training_data_df.drop('close', axis=1).values
Y_training = training_data_df[['close']].values
```

And likewise we take the test data and we use the same close column

```
test_data_df =
pd.read_csv("/dbfs/FileStore/tables/all_stocks_5yr_amazon_test.csv",
dtype=float)
X_testing = test_data_df.drop('close', axis=1).values
Y_testing = test_data_df[['close']].values
```

Taking the range to preprocess the data we take 0 to 1

```
X_scaler = MinMaxScaler(feature_range=(0, 1))
Y_scaler = MinMaxScaler(feature_range=(0, 1))
```

Using this we going scale the values that was taken previously for both the training set and testing set

```
X_scaled_training = X_scaler.fit_transform(X_training)
Y_scaled_training = Y_scaler.fit_transform(Y_training)

X_scaled_testing = X_scaler.transform(X_testing)
Y_scaled_testing = Y_scaler.transform(Y_testing)
```

Now we can define some values to use in the neural network model, first we the define the learning_rate at which the weight is going to change and how many times we are going to process this using with training_epochs, in this case we set a step of 0.001 and 200 epochs

```
learning_rate = 0.001
training_epochs = 200
```

We currently have 4 inputs and 1 output so we declare as follows

```
number_of_inputs = 4
number_of_outputs = 1
```

And finally we define the number of nodes per layer

```
layer_1_nodes = 75
layer_2_nodes = 100
layer_3_nodes = 75
```

When we have this initial values for the neural network model we now can use them to define the layers

Input Layer

```
with tf.variable_scope('input'):
    X = tf.placeholder(tf.float32, shape=(None, number_of_inputs))
```

Layer 1

```
with tf.variable_scope('layer_1'):
    weights = tf.get_variable("weights1", shape=[number_of_inputs,
layer_1_nodes], initializer=tf.contrib.layers.xavier_initializer())
    biases = tf.get_variable(name="biases1", shape=[layer_1_nodes],
initializer=tf.zeros_initializer())
    layer_1_output = tf.nn.relu(tf.matmul(X, weights) + biases)
```

Layer 2

```
with tf.variable_scope('layer_2'):
    weights = tf.get_variable("weights2", shape=[layer_1_nodes,
layer_2_nodes], initializer=tf.contrib.layers.xavier_initializer())
    biases = tf.get_variable(name="biases2", shape=[layer_2_nodes],
initializer=tf.zeros_initializer())
    layer_2_output = tf.nn.relu(tf.matmul(layer_1_output, weights) + biases)
```

Layer 3

```
with tf.variable_scope('layer_3'):
    weights = tf.get_variable("weights3", shape=[layer_2_nodes,
layer_3_nodes], initializer=tf.contrib.layers.xavier_initializer())
    biases = tf.get_variable(name="biases3", shape=[layer_3_nodes],
initializer=tf.zeros_initializer())
    layer_3_output = tf.nn.relu(tf.matmul(layer_2_output, weights) + biases)
```

Output Layer

```
with tf.variable_scope('output'):
    weights = tf.get_variable("weights4", shape=[layer_3_nodes,
number_of_outputs], initializer=tf.contrib.layers.xavier_initializer())
    biases = tf.get_variable(name="biases4", shape=[number_of_outputs],
initializer=tf.zeros_initializer())
    prediction = tf.matmul(layer_3_output, weights) + biases
```

Later we make the cost that will determine how accurate the prediction is while the training is going

```
with tf.variable_scope('cost'):
    Y = tf.placeholder(tf.float32, shape=(None, 1))
    cost = tf.reduce_mean(tf.squared_difference(prediction, Y))
```

We define the optimizer that will work in the neural network model

```
with tf.variable_scope('train'):
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

And the summary for the log process on the neural network

```
with tf.variable_scope('logging'):
    tf.summary.scalar('current_cost', cost)
    summary = tf.summary.merge_all()
saver = tf.train.Saver()
```

We all this layer defined we can start the process of the session running TensorFlow

```
with tf.Session() as session:
  session.run(tf.global_variables_initializer())
  training_writer = tf.summary.FileWriter('./logs/training', session.graph)
  testing_writer = tf.summary.FileWriter('./logs/testing', session.graph)
  for epoch in range(training_epochs):
          session.run(optimizer, feed_dict={X: X_scaled_training, Y:
Y_scaled_training})
          # Every 5 training steps, log our progress
          if epoch % 5 == 0:
            training_cost, training_summary = session.run([cost, summary],
feed_dict={X: X_scaled_training, Y:Y_scaled_training})
            testing_cost, testing_summary = session.run([cost, summary],
feed_dict={X: X_scaled_testing, Y:Y_scaled_testing})
            training_writer.add_summary(training_summary, epoch)
            testing_writer.add_summary(testing_summary, epoch)
            print("Epoch: {} - Training Cost: {}  Testing Cost:
{}".format(epoch, training_cost, testing_cost))
            final_training_cost = session.run(cost, feed_dict={X:
X_scaled_training, Y: Y_scaled_training})
            final_testing_cost = session.run(cost, feed_dict={X:
X_scaled_testing, Y: Y_scaled_testing})
  print("Final Training cost: {}".format(final_training_cost))
  print("Final Testing cost: {}".format(final_testing_cost))
  Y_predicted_scaled = session.run(prediction, feed_dict={X:
X_scaled_testing})
  Y_predicted = Y_scaler.inverse_transform(Y_predicted_scaled)
  real_earnings = test_data_df['close'].values[0]
  predicted_earnings = Y_predicted[0][0]
  print("The actual close were ${}".format(real_earnings))
  print("Our neural network predicted close of
${}".format(predicted_earnings))

  save_path = saver.save(session, "logs/trained_model.ckpt")
  print("Model saved: {}".format(save_path))
```

And we can see the process during each iterations

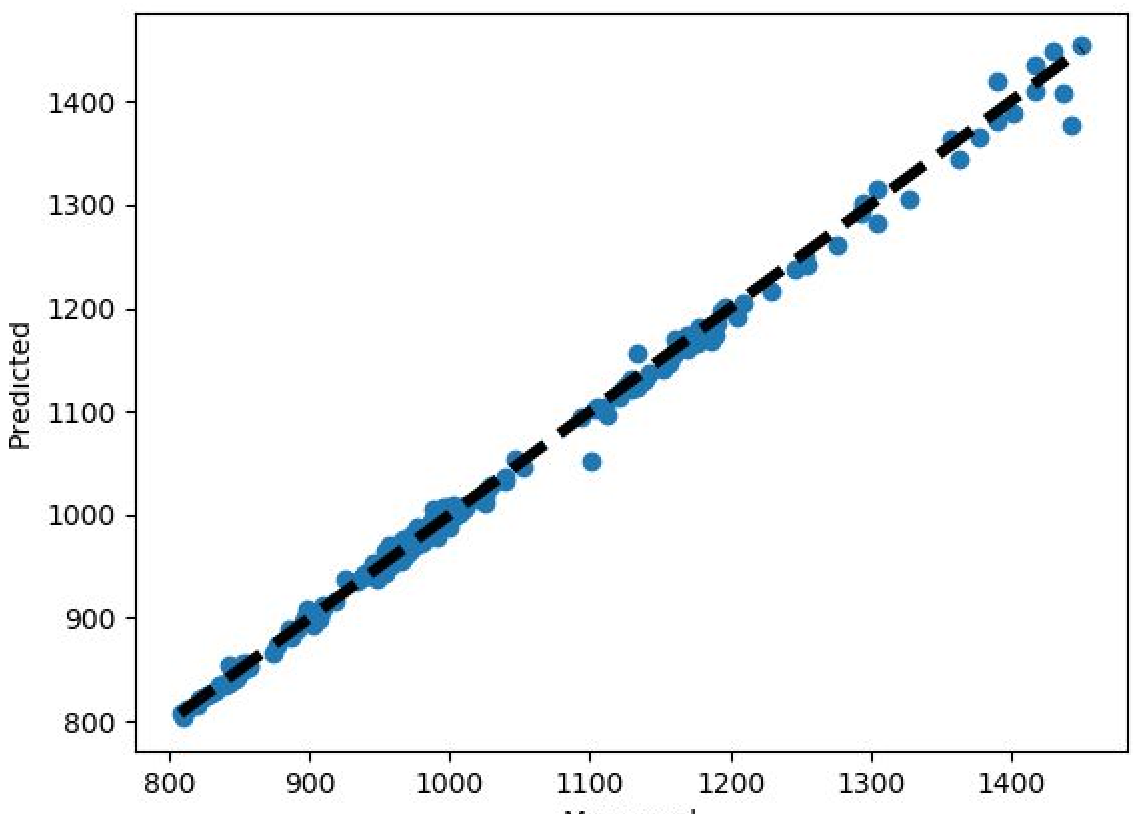
```
Epoch: 80 - Training Cost: 8.377773337997496e-05  Testing Cost: 0.0004893926670774817
Epoch: 85 - Training Cost: 8.059653191594407e-05  Testing Cost: 0.0005367605481296778
Epoch: 90 - Training Cost: 7.346444908762351e-05  Testing Cost: 0.000354340358171612
Epoch: 95 - Training Cost: 6.783728167647496e-05  Testing Cost: 0.0004179234674666077
Epoch: 100 - Training Cost: 6.558767199749127e-05  Testing Cost: 0.000383748731110245
```

```
import matplotlib.pyplot as plt

real_earningsAll = test_data_df['close']
fig, ax = plt.subplots()
ax.scatter(real_earningsAll, Y_predicted)
ax.plot([real_earningsAll.min(), real_earningsAll.max()],
[real_earningsAll.min(),
real_earningsAll.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
display(fig)
```

<u>Test #1</u>

```
Final Training cost: 4.9574733566259965e-05
Final Testing cost: 0.00021704452228732407
The actual close were $835.77
Our neural network predicted close of $835.803466796875
```

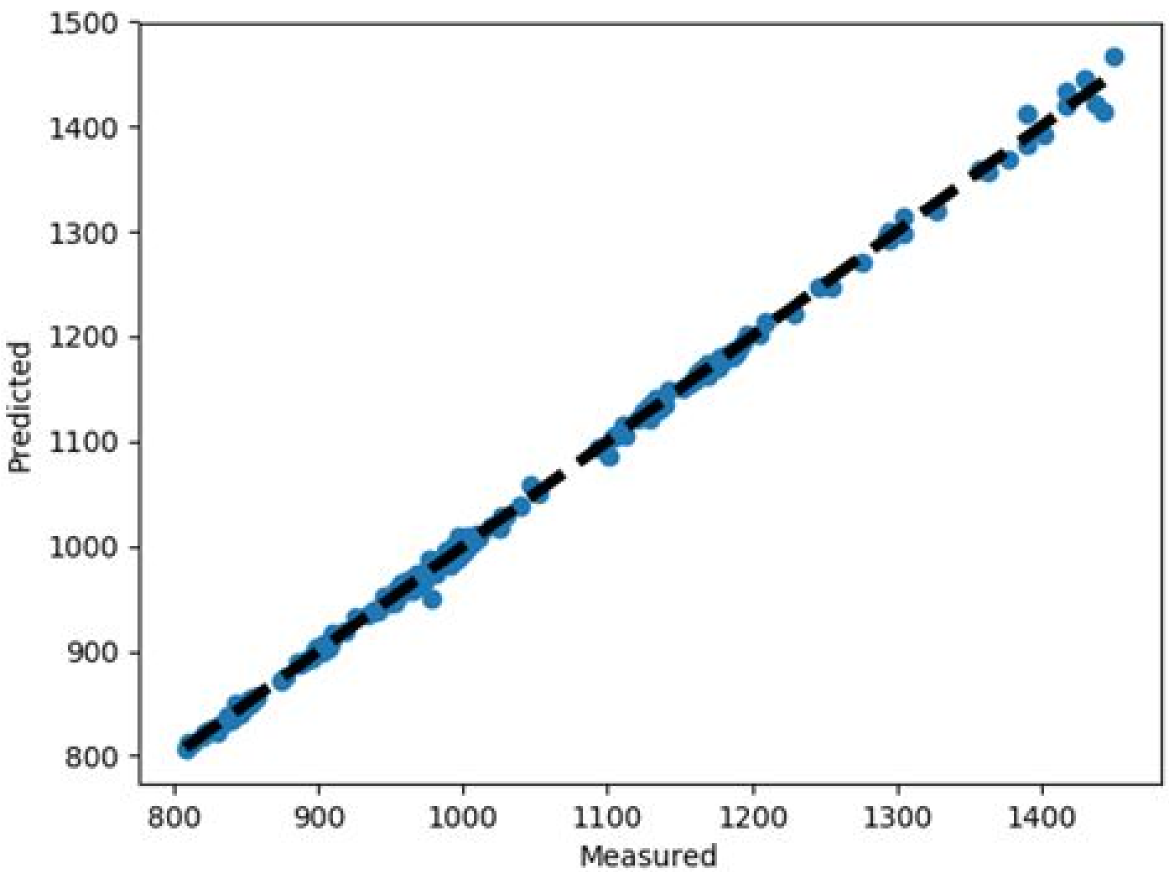


Findings:
Using the following parameters, we were able to predict the value **1436.1113** (Actual value: 1416.78)

Parameters:
learning_rate = 0.001
training_epochs = 200
layer_1_nodes = 75
layer_2_nodes = 100
layer_3_nodes = 75

Test #2

```
Final Training cost: 1.9963938029832207e-05
Final Testing cost: 7.552812894573435e-05
The actual close were $835.77
Our neural network predicted close of $833.1156616210938
```



Findings:
Using the following parameters, we were able to predict the value **1433.7349**
(Actual value: 1416.78)
Parameters:

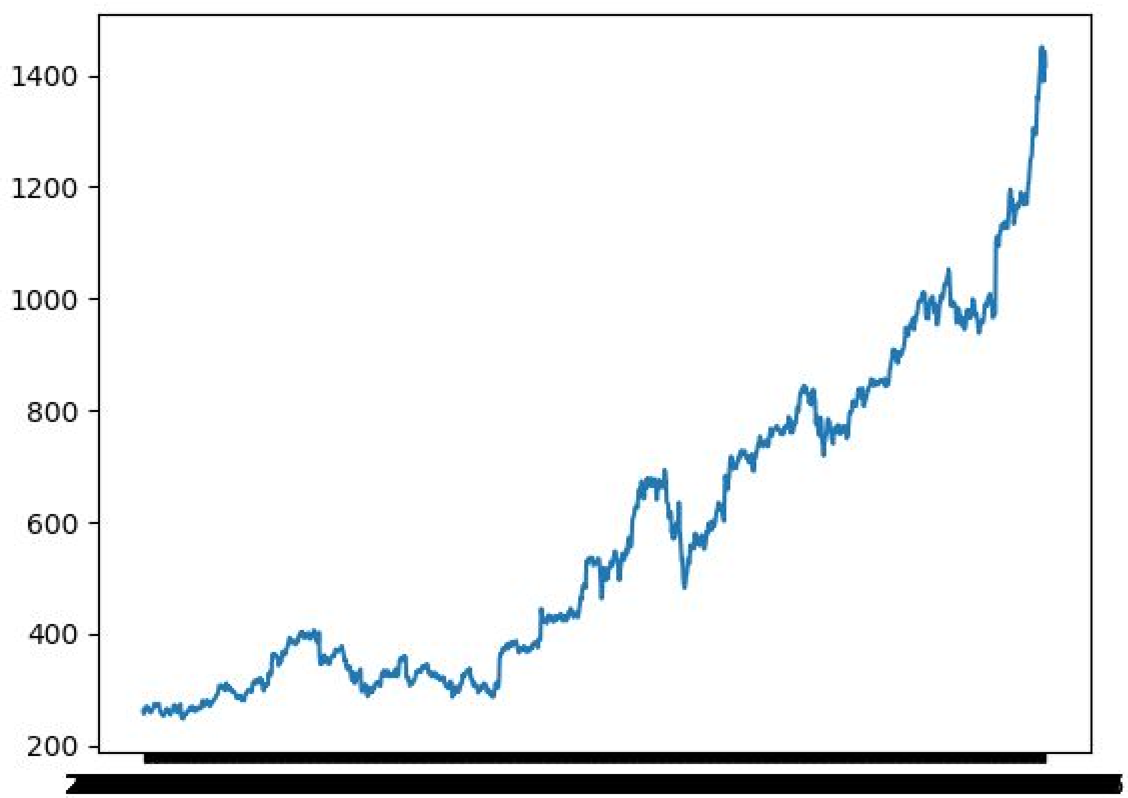
learning_rate = 0.001
training_epochs = 800
layer_1_nodes = 1024
layer_2_nodes = 512
layer_3_nodes = 256

```
import matplotlib.pyplot as plt

data_df =
pd.read_csv("/dbfs/FileStore/tables/all_stocks_5yr_amazon_full.csv")

fig, ax = plt.subplots()
y_plot = data_df['close']
x_plot = data_df['date']
plt.plot(x_plot, y_plot, '-')

display(fig)
```



Graph containing the evolution of the AMZN stock in 5 years.

Now using MLlib with same dataset
We load the data

```
regressionDataFrame =
spark.read.csv('/FileStore/tables/all_stocks_5yr_amazon_full.csv',header=True
, inferSchema = True)
```

```
regressionDataFrame.show(5)
```

```
+-------------------+------+------+-------+------+-------+----+
|               date|  open|  high|    low| close| volume|Name|
+-------------------+------+------+-------+------+-------+----+
|2013-02-08 00:00:00| 261.4|265.25|260.555|261.95|3879078|AMZN|
|2013-02-11 00:00:00| 263.2|263.25|  256.6|257.21|3403403|AMZN|
|2013-02-12 00:00:00|259.19|260.16|  257.0| 258.7|2938660|AMZN|
|2013-02-13 00:00:00|261.53|269.96|  260.3|269.47|5292996|AMZN|
|2013-02-14 00:00:00|267.37|270.65|  265.4|269.24|3462780|AMZN|
+-------------------+------+------+-------+------+-------+----+
```

We get the resilient distributed dataset (RDD) to transform it to labeled point

```
regressionDataRDDDict = regressionDataFrame.rdd
```

```
regressionDataRDDDict = regressionDataFrame.rdd
regressionDataRDDDict.take(5)

Out[142]:
[Row(date=datetime.datetime(2013, 2, 8, 0, 0), open=261.4, high=265.25, low=260.555, close=261.95, volume=3879078,
Name='AMZN'),
 Row(date=datetime.datetime(2013, 2, 11, 0, 0), open=263.2, high=263.25, low=256.6, close=257.21, volume=3403403,
Name='AMZN'),
 Row(date=datetime.datetime(2013, 2, 12, 0, 0), open=259.19, high=260.16, low=257.0, close=258.7, volume=2938660,
Name='AMZN'),
 Row(date=datetime.datetime(2013, 2, 13, 0, 0), open=261.53, high=269.96, low=260.3, close=269.47, volume=5292996,
Name='AMZN'),
 Row(date=datetime.datetime(2013, 2, 14, 0, 0), open=267.37, high=270.65, low=265.4, close=269.24, volume=3462780,
Name='AMZN')]
```

Since the RDD has key with values and we only want to have values we use map for that

```
regressionDataRDD = regressionDataFrame.rdd.map(list)
```

```
regressionDataRDD = regressionDataFrame.rdd.map(list)
regressionDataRDD.take(5)

Out[143]:
[[datetime.datetime(2013, 2, 8, 0, 0),
  261.4,
  265.25,
  260.555,
  261.95,
  3879078,
  'AMZN'],
```

Once we have only the values we can process with LabeledPoint using the close value as the label and the open, high and low as the features.

```
from pyspark.mllib.regression import LabeledPoint
regressionDataLabelPoint = regressionDataRDDDict.map(lambda data :
LabeledPoint(data[4],data[1:4]))
```

```
regressionDataLabelPoint.take(5)

Out[145]:
[LabeledPoint(261.95, [261.4,265.25,260.555]),
 LabeledPoint(257.21, [263.2,263.25,256.6]),
 LabeledPoint(258.7, [259.19,260.16,257.0]),
 LabeledPoint(269.47, [261.53,269.96,260.3]),
 LabeledPoint(269.24, [267.37,270.65,265.4])]
```

We going to split this new data set randomly 70% to train and 30% to test

```
regressionLabelPointSplit = regressionDataLabelPoint.randomSplit([0.7,0.3])
```

Training set

```
regressionLabelPointTrainData = regressionLabelPointSplit[0]
regressionLabelPointTrainData.take(5)

Out[147]:
[LabeledPoint(257.21, [263.2,263.25,256.6]),
 LabeledPoint(258.7, [259.19,260.16,257.0]),
 LabeledPoint(269.47, [261.53,269.96,260.3]),
 LabeledPoint(269.24, [267.37,270.65,265.4]),
 LabeledPoint(269.75, [265.91,270.11,264.5])]
```

```
regressionLabelPointTrainData.count()

Out[148]: 884
```

Testing set

```
regressionLabelPointTestData = regressionLabelPointSplit[1]
regressionLabelPointTestData.take(5)

Out[149]:
[LabeledPoint(261.95, [261.4,265.25,260.555]),
 LabeledPoint(265.09, [267.63,268.92,263.11]),
 LabeledPoint(266.41, [270.2,274.3,266.371]),
 LabeledPoint(263.25, [259.4,265.83,256.86]),
 LabeledPoint(273.11, [265.36,273.3,264.14])]
```

```
regressionLabelPointTestData.count()

Out[150]: 375
```

Using the Training Data we set it with 200 iterations and a step of 0.00001 to make a model with linear regression to test it later

```
from pyspark.mllib.regression import LinearRegressionWithSGD as lrSGD
ourModelWithLinearRegression = lrSGD.train(data =
regressionLabelPointTrainData, iterations = 200, step = 0.00001, intercept =
True)
```

```
ourModelWithLinearRegression.intercept
```

```
Out[152]: 1.0004267774059894
```

```
ourModelWithLinearRegression.weights
```

```
Out[153]: DenseVector([0.3296, 0.3386, 0.3306])
```

```
ourModelWithLinearRegression.save(sc, 'dbfs:/FileStore/tables
/ourModelWithLinearRegressionAmazonStock_201904152024')
```

Now we can test the model and validate how accurate the model is

actualDataandLinearRegressionPredictedData = regressionLabelPointTestData.map(lambda data : (float(data.label) , float(ourModelWithLinearRegression.predict(data.features))))

```
actualDataandLinearRegressionPredictedData.take(5)
```

```
Out[160]:
[(261.95, 263.1058758900237),
 (265.09, 267.24644204215144),
 (266.41, 270.9932822821445),
 (263.25, 261.4215935924259),
 (273.11, 268.32203675609145)]
```

We calculate the value of root-mean-square error and finally the r2

```
from pyspark.mllib.evaluation import RegressionMetrics as rmtrcs
ourLinearRegressionModelMetrics = rmtrcs(actualDataandLinearRegressionPredictedData)
ourLinearRegressionModelMetrics.rootMeanSquaredError
```

```
Out[161]: 5.514359415419885
```

```
ourLinearRegressionModelMetrics.r2
```

```
Out[162]: 0.9996232102011257
```

We can validate the model showing that r2 is greater than 0.5 therefore is a good model

Further data application

S&P Index Based investment

1. Get daily change

```python
# calculate daily rate changes
def calcPercent(close, open):
   return ((close - open) / open) * 100

dataFrame = spDataFrame.withColumn('change', calcPercent(spDataFrame['close'], spDataFrame['open']))
dataFrame.show()
```

```
▸ (1) Spark Jobs
▸ dataFrame: pyspark.sql.dataframe.DataFrame = [date: timestamp, open: double ... 6 more fields]
+-------------------+-----+-----+-----+-----+--------+----+--------------------+
|               date| open| high|  low|close|  volume|Name|              change|
+-------------------+-----+-----+-----+-----+--------+----+--------------------+
|2013-02-08 00:00:00|15.07|15.12|14.63|14.75| 8407500| AAL|  -2.123424021234242|
|2013-02-11 00:00:00|14.89|15.01|14.26|14.46| 8882000| AAL| -2.8878441907320327|
|2013-02-12 00:00:00|14.45|14.51| 14.1|14.27| 8126000| AAL|  -1.245674740484427|
|2013-02-13 00:00:00| 14.3|14.94|14.25|14.66|10259500| AAL|  2.5174825174825135|
|2013-02-14 00:00:00|14.94|14.96|13.16|13.99|31879900| AAL|  -6.358768406961174|
|2013-02-15 00:00:00|13.93|14.61|13.93| 14.5|15628000| AAL|   4.091888011486003|
|2013-02-19 00:00:00|14.33|14.56|14.08|14.26|11354400| AAL|-0.48848569434752465|
|2013-02-20 00:00:00|14.17|14.26|13.15|13.33|14725200| AAL|  -5.928016937191248|
|2013-02-21 00:00:00|13.62|13.95| 12.9|13.37|11922100| AAL| -1.8355359765051396|
|2013-02-22 00:00:00|13.57| 13.6|13.21|13.57| 6071400| AAL|                 0.0|
|2013-02-25 00:00:00| 13.6|13.76| 13.0|13.02| 7186400| AAL|  -4.264705882352942|
|2013-02-26 00:00:00|13.14|13.42| 12.7|13.26| 9419000| AAL|  0.9132420091324142|
|2013-02-27 00:00:00|13.28|13.62|13.18|13.41| 7390500| AAL|  0.9789156626506084|
|2013-02-28 00:00:00|13.49|13.63|13.39|13.43| 6143600| AAL|-0.44477390659748334|
|2013-03-01 00:00:00|13.37|13.95|13.32|13.61| 7376800| AAL|  1.7950635751682888|
|2013-03-04 00:00:00| 13.5|14.07|13.47| 13.9| 8174800| AAL|  2.9629629629629655|
|2013-03-05 00:00:00|14.01|14.05|13.71|14.05| 7676100| AAL|   0.285510349750185|
|2013-03-06 00:00:00|14.52|14.68|14.25|14.57|13243200| AAL| 0.34435261707989473|
|2013-03-07 00:00:00| 14.7|14.93| 14.5|14.82| 9125300| AAL|  0.8163265306122517|
|2013-03-08 00:00:00|14.99| 15.2|14.84|14.92|10593700| AAL| -0.4669779853235509|
+-------------------+-----+-----+-----+-----+--------+----+--------------------+
only showing top 20 rows
```

2. Reduce columns

```python
# Select specific columns from dataframe
reducedDF = dataFrame.select('date', 'Name', 'change')
reducedDF.show()
```

```
▸ (1) Spark Jobs
▸ reducedDF: pyspark.sql.dataframe.DataFrame = [date: timestamp, Name: string ... 1 more fields]
+-------------------+----+--------------------+
|               date|Name|              change|
+-------------------+----+--------------------+
|2013-02-08 00:00:00| AAL|  -2.123424021234242|
|2013-02-11 00:00:00| AAL| -2.8878441907320327|
|2013-02-12 00:00:00| AAL|  -1.245674740484427|
|2013-02-13 00:00:00| AAL|  2.5174825174825135|
|2013-02-14 00:00:00| AAL|  -6.358768406961174|
|2013-02-15 00:00:00| AAL|   4.091888011486003|
|2013-02-19 00:00:00| AAL|-0.48848569434752465|
|2013-02-20 00:00:00| AAL|  -5.928016937191248|
|2013-02-21 00:00:00| AAL| -1.8355359765051396|
|2013-02-22 00:00:00| AAL|                 0.0|
|2013-02-25 00:00:00| AAL|  -4.264705882352942|
|2013-02-26 00:00:00| AAL|  0.9132420091324142|
|2013-02-27 00:00:00| AAL|  0.9789156626506084|
|2013-02-28 00:00:00| AAL|-0.44477390659748334|
|2013-03-01 00:00:00| AAL|  1.7950635751682888|
|2013-03-04 00:00:00| AAL|  2.9629629629629655|
|2013-03-05 00:00:00| AAL|   0.285510349750185|
|2013-03-06 00:00:00| AAL| 0.34435261707989473|
|2013-03-07 00:00:00| AAL|  0.8163265306122517|
|2013-03-08 00:00:00| AAL| -0.4669779853235509|
+-------------------+----+--------------------+
only showing top 20 rows
```

3. Get daily rate change mean

```python
# get total average
from pyspark.sql.functions import mean, col

groupedDF = reducedDF.groupBy("Name").mean("change").select("Name", "avg(change)")
groupedDF = groupedDF.select(col("Name").alias("name"), col("avg(change)").alias("mean"))
groupedDF.show()
```

▸ (3) Spark Jobs

▸ groupedDF: pyspark.sql.dataframe.DataFrame = [name: string, mean: double]

```
+----+--------------------+
|name|                mean|
+----+--------------------+
|ALXN|-0.03819572050231...|
| GIS|0.056748417622980586|
|   K|0.030945186822567604|
| LEN|-0.02993897166939417|
|SPGI| 0.09052882241446729|
| AIV| 0.05187183560618326|
| AVY| 0.04834277090589205|
|BF.B| 0.04351933136018874|
| MMM| 0.07236058899879151|
| PKI| 0.07594906776537622|
| PPG|  0.0440281761788716|
|  RF| 0.03740940624892114|
| AXP|0.022083036046032792|
|  CI| 0.08535447752213957|
| IRM|0.021304890793449105|
| WEC|0.057792724373221784|
|INFO| 0.13868457521732652|
| PFG| 0.05860423437773062|
|  PM| 0.04034244258475278|
| SNA|0.034608708151195086|
+----+--------------------+
```

4. Sort

```
# sort data frame
sortedDF = groupedDF.sort(col("mean").desc())
sortedDF.show()

```

(1) Spark Jobs

sortedDF: pyspark.sql.dataframe.DataFrame = [name: string, mean: double]

```
+----+-------------------+
|name|               mean|
+----+-------------------+
| HPE| 0.1915503478274737|
| HPQ| 0.1909562023928024|
|INFO|0.13868457521732652|
| BBY|0.12799839289891546|
|NVDA|0.12357063024288786|
| HUM|0.11422863658447747|
|  EA|0.11386175916937684|
|FISV|0.10907777744128025|
|INTU|0.10733630762579978|
| RMD|0.10530298930080509|
|ANTM|0.10479424202254987|
| HII| 0.1031360210136624|
| TSS|0.10231258710884215|
|NTAP|0.10205940980971352|
|ABBV|0.10162815874338921|
| ACN|0.09826030744984464|
| MAR|0.09806690664753333|
| VAR| 0.0979728303444835|
| HCA|0.09761553318365702|
|ILMN|0.09716045291125679|
+----+-------------------+
only showing top 20 rows
```

5. Get top N (10 in this example)

```
# get top 10
top10 = sortedDF.head(10)
top10
```

▸ (1) Spark Jobs

```
Out[134]:
[Row(name='HPE', mean=0.1915503478274737),
 Row(name='HPQ', mean=0.1909562023928024),
 Row(name='INFO', mean=0.13868457521732652),
 Row(name='BBY', mean=0.12799839289891546),
 Row(name='NVDA', mean=0.12357063024288786),
 Row(name='HUM', mean=0.11422863658447747),
 Row(name='EA', mean=0.11386175916937684),
 Row(name='FISV', mean=0.10907777744128025),
 Row(name='INTU', mean=0.10733630762579978),
 Row(name='RMD', mean=0.10530298930080509)]
```

6. Calculate index weight

```
# calculate total index
df_stats = reducedDF.select(
    mean(col('change')).alias('mean')
).collect()
totalMarketWeight = df_stats[0]['mean']
totalMarketWeight
```

▸ (1) Spark Jobs

```
Out[147]: 0.03527777096687718
```

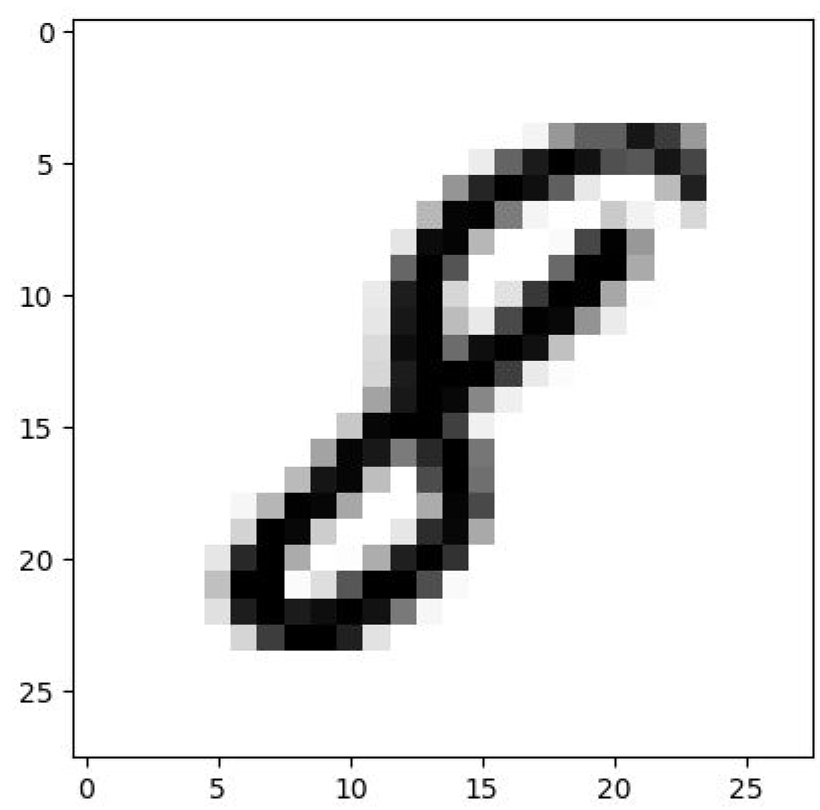
Keras dataset
Using this API we can test and train multiple images with different shapes to detect which number it is

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

import matplotlib.pyplot as plt

You may select anything up to 60,000

```
image_index = 7777
print(y_train[image_index]) # The label is 8
plt.imshow(x_train[image_index], cmap=plt.get_cmap('gray_r'))
display(plt.show())
x_train.shape
```



Reshaping the array to 4-dims so that it can work with the Keras API

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

```
input_shape = (28, 28, 1)
```

Making sure that the values are float so that we can get decimal points after division

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

Normalizing the RGB codes by dividing it to the max RGB value.

```
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
```

```
x_train shape: (60000, 28, 28, 1) Number of images in x_train 60000 Number of
images in x_test 10000
```

Importing the required Keras modules containing model and layers

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

Using TensorFlow backend.

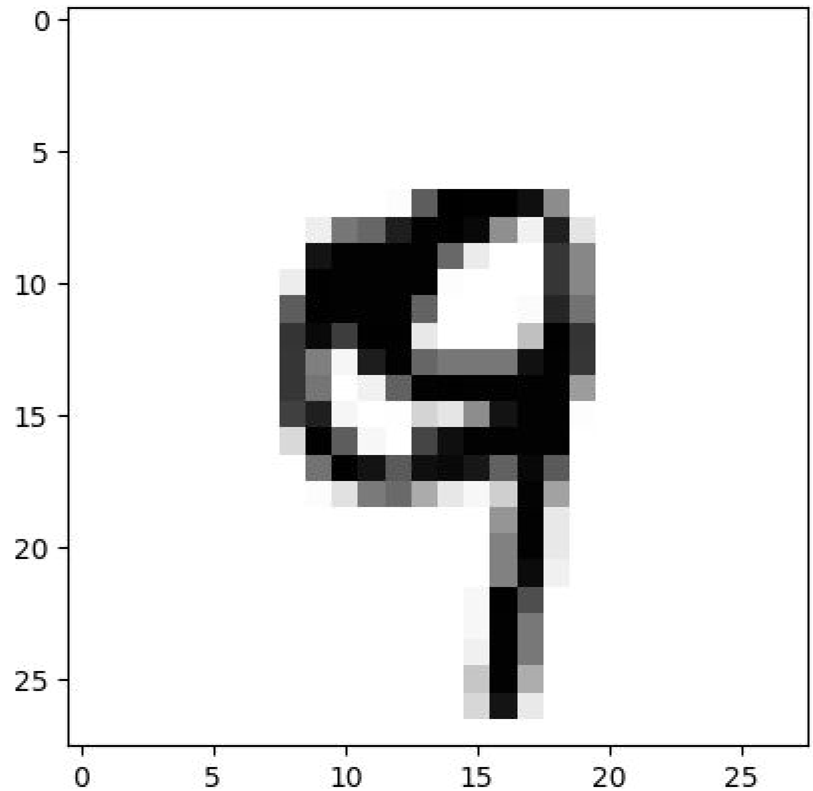
```
model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)
```

```
59072/60000 [============================>.] - ETA: 0s - loss: 0.0203 - acc: 0.9931
59200/60000 [============================>.] - ETA: 0s - loss: 0.0203 - acc: 0.9931
59328/60000 [============================>.] - ETA: 0s - loss: 0.0202 - acc: 0.9931
59456/60000 [============================>.] - ETA: 0s - loss: 0.0202 - acc: 0.9931
59552/60000 [============================>.] - ETA: 0s - loss: 0.0202 - acc: 0.9931
59680/60000 [============================>.] - ETA: 0s - loss: 0.0202 - acc: 0.9931
59808/60000 [============================>.] - ETA: 0s - loss: 0.0202 - acc: 0.9931
59904/60000 [============================>.] - ETA: 0s - loss: 0.0201 - acc: 0.9931
60000/60000 [==============================] - 29s 479us/step - loss: 0.0201 - acc: 0.9931
Out[28]: <keras.callbacks.History at 0x7efdf4c86e48>
```

model.evaluate(x_test, y_test)

```
 7776/10000 [======================>.......] - ETA: 0s
 7968/10000 [======================>.......] - ETA: 0s
 8256/10000 [=======================>......] - ETA: 0s
 8544/10000 [========================>.....] - ETA: 0s
 8832/10000 [=========================>....] - ETA: 0s
 9120/10000 [==========================>...] - ETA: 0s
 9440/10000 [===========================>..] - ETA: 0s
 9728/10000 [============================>.] - ETA: 0s
 9984/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 2s 193us/step
Out[29]: [0.06136563551748695, 0.9845]
```

```
image_index = 4444
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
display(plt.show())
```




```
pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
print(pred.argmax())
```

```
9
```

At the end we can predict using the test to evaluate and show which number looks like