aapl_regression_scikit_learn

December 28, 2021

1 Building a Regression Model for a Financial Dataset

In this notebook, you will build a simple linear regression model to predict the closing AAPL stock price. The lab objectives are: * Pull data from BigQuery into a Pandas dataframe * Use Matplotlib to visualize data * Use Scikit-Learn to build a regression model

[1]:
```
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

[2]:
```
!pip install --user google-cloud-bigquery==1.25.0
```

```
Collecting google-cloud-bigquery==1.25.0
  Downloading google_cloud_bigquery-1.25.0-py2.py3-none-any.whl (169 kB)
     |                  | 169 kB 9.0 MB/s
Collecting google-resumable-media<0.6dev,>=0.5.0
  Downloading google_resumable_media-0.5.1-py2.py3-none-any.whl (38 kB)
Collecting google-auth<2.0dev,>=1.9.0
  Downloading google_auth-1.35.0-py2.py3-none-any.whl (152 kB)
     |                  | 152 kB 55.9 MB/s
Collecting google-api-core<2.0dev,>=1.15.0
  Downloading google_api_core-1.31.5-py2.py3-none-any.whl (93 kB)
     |                  | 93 kB 1.8 MB/s
Requirement already satisfied: protobuf>=3.6.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(3.19.1)
Requirement already satisfied: six<2.0.0dev,>=1.13.0 in
/opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0)
(1.16.0)
Collecting google-cloud-core<2.0dev,>=1.1.0
  Downloading google_cloud_core-1.7.2-py2.py3-none-any.whl (28 kB)
Requirement already satisfied: setuptools>=40.3.0 in
/opt/conda/lib/python3.7/site-packages (from google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (59.6.0)
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in
/opt/conda/lib/python3.7/site-packages (from google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (1.53.0)
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in
/opt/conda/lib/python3.7/site-packages (from google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2.26.0)
```

```
Requirement already satisfied: packaging>=14.3 in /opt/conda/lib/python3.7/site-
packages (from google-api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0)
(21.3)
Requirement already satisfied: pytz in /opt/conda/lib/python3.7/site-packages
(from google-api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2021.3)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.7/site-
packages (from google-auth<2.0dev,>=1.9.0->google-cloud-bigquery==1.25.0) (4.8)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/opt/conda/lib/python3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-
cloud-bigquery==1.25.0) (0.2.7)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-
cloud-bigquery==1.25.0) (4.2.4)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from packaging>=14.3->google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (3.0.6)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/opt/conda/lib/python3.7/site-packages (from pyasn1-modules>=0.2.1->google-
auth<2.0dev,>=1.9.0->google-cloud-bigquery==1.25.0) (0.4.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2.0.9)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0dev,>=2.18.0->google-api-
core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (3.1)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-
api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (1.26.7)
Installing collected packages: google-auth, google-api-core, google-resumable-
media, google-cloud-core, google-cloud-bigquery
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
explainable-ai-sdk 1.3.2 requires xai-image-widget, which is not installed.
google-cloud-storage 1.43.0 requires google-resumable-media<3.0dev,>=1.3.0;
python_version >= "3.6", but you have google-resumable-media 0.5.1 which is
incompatible.
cloud-tpu-client 0.10 requires google-api-python-client==1.8.0, but you have
google-api-python-client 2.33.0 which is incompatible.
Successfully installed google-api-core-1.31.5 google-auth-1.35.0 google-cloud-
```

```
bigquery-1.25.0 google-cloud-core-1.7.2 google-resumable-media-0.5.1
```

Note: Restart your kernel to use updated packages.

Kindly ignore the deprecation warnings and incompatibility errors related to google-cloud-storage.

[2]:
```
%%bash

bq mk -d ai4f
bq load --autodetect --source_format=CSV ai4f.AAPL10Y gs://cloud-training/ai4f/
 ↪AAPL10Y.csv
```

```
BigQuery error in mk operation: Dataset 'qwiklabs-gcp-00-78295dc8d771:ai4f'
already exists.

Waiting on bqjob_r6a6f3218b8be9872_0000017e01074006_1 … (1s) Current status:
DONE
```

[3]:
```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

plt.rc('figure', figsize=(12, 8.0))
```

1.1 Pull Data from BigQuery

In this section we'll use a magic function to query a BigQuery table and then store the output in a Pandas dataframe. A magic function is just an alias to perform a system command. To see documentation on the "bigquery" magic function execute the following cell:

The query below selects everything you'll need to build a regression model to predict the closing price of AAPL stock. The model will be very simple for the purposes of demonstrating BQML functionality. The only features you'll use as input into the model are the previous day's closing price and a three day trend value. The trend value can only take on two values, either -1 or +1. If the AAPL stock price has increased over any two of the previous three days then the trend will be +1. Otherwise, the trend value will be -1.

Note, the features you'll need can be generated from the raw table `ai4f.AAPL10Y` using Pandas functions. However, it's better to take advantage of the serverless-ness of BigQuery to do the data pre-processing rather than applying the necessary transformations locally.

[4]:
```
%%bigquery df
WITH
  raw AS (
  SELECT
```

```
    date,
    close,
    LAG(close, 1) OVER(ORDER BY date) AS min_1_close,
    LAG(close, 2) OVER(ORDER BY date) AS min_2_close,
    LAG(close, 3) OVER(ORDER BY date) AS min_3_close,
    LAG(close, 4) OVER(ORDER BY date) AS min_4_close
  FROM
    `ai4f.AAPL10Y`
  ORDER BY
    date DESC ),
  raw_plus_trend AS (
  SELECT
    date,
    close,
    min_1_close,
    IF (min_1_close - min_2_close > 0, 1, -1) AS min_1_trend,
    IF (min_2_close - min_3_close > 0, 1, -1) AS min_2_trend,
    IF (min_3_close - min_4_close > 0, 1, -1) AS min_3_trend
  FROM
    raw ),
  train_data AS (
  SELECT
    date,
    close,
    min_1_close AS day_prev_close,
    IF (min_1_trend + min_2_trend + min_3_trend > 0, 1, -1) AS trend_3_day
  FROM
    raw_plus_trend
  ORDER BY
    date ASC )
SELECT
  *
FROM
  train_data
```

View the first five rows of the query's output. Note that the object `df` containing the query output is a Pandas Dataframe.

[5]:
```
print(type(df))
df.dropna(inplace=True)
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

[5]:

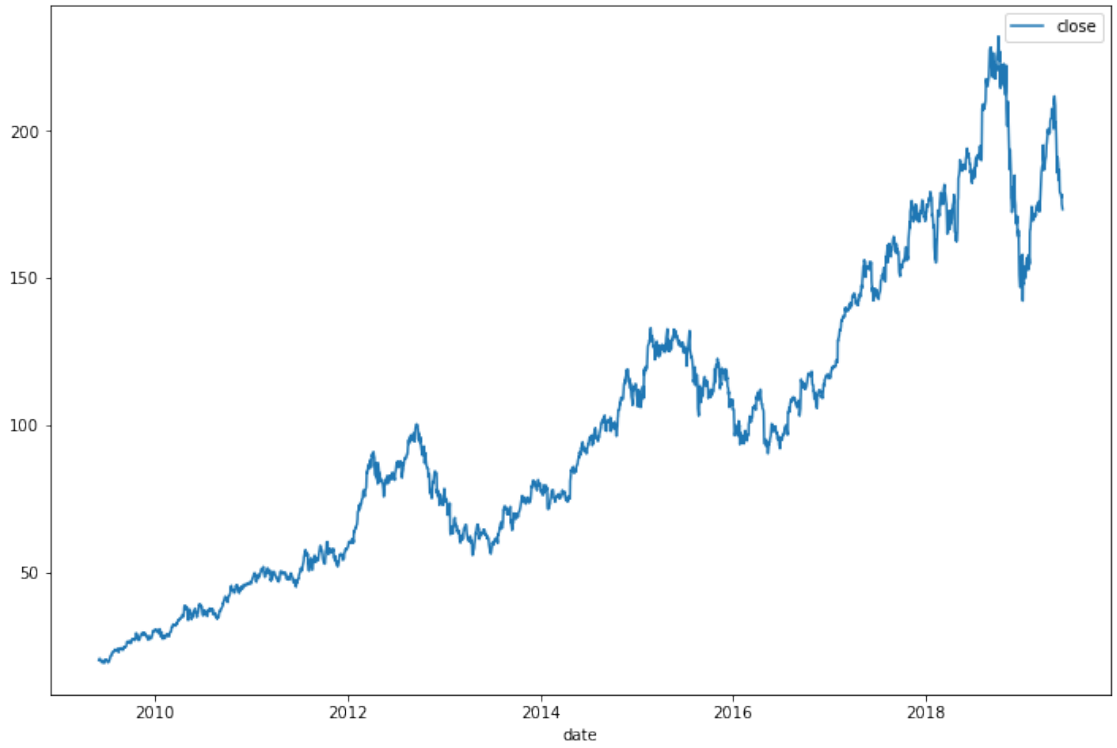
	date	close	day_prev_close	trend_3_day
1	2009-06-03	20.1357	20.1357	-1
2	2009-06-04	20.5343	20.5343	-1
3	2009-06-04	20.5343	20.1357	-1

4	2009-06-05	20.6671	20.6671	1
5	2009-06-05	20.6671	20.5343	-1

1.2 Visualize data

The simplest plot you can make is to show the closing stock price as a time series. Pandas DataFrames have built in plotting funtionality based on Matplotlib.

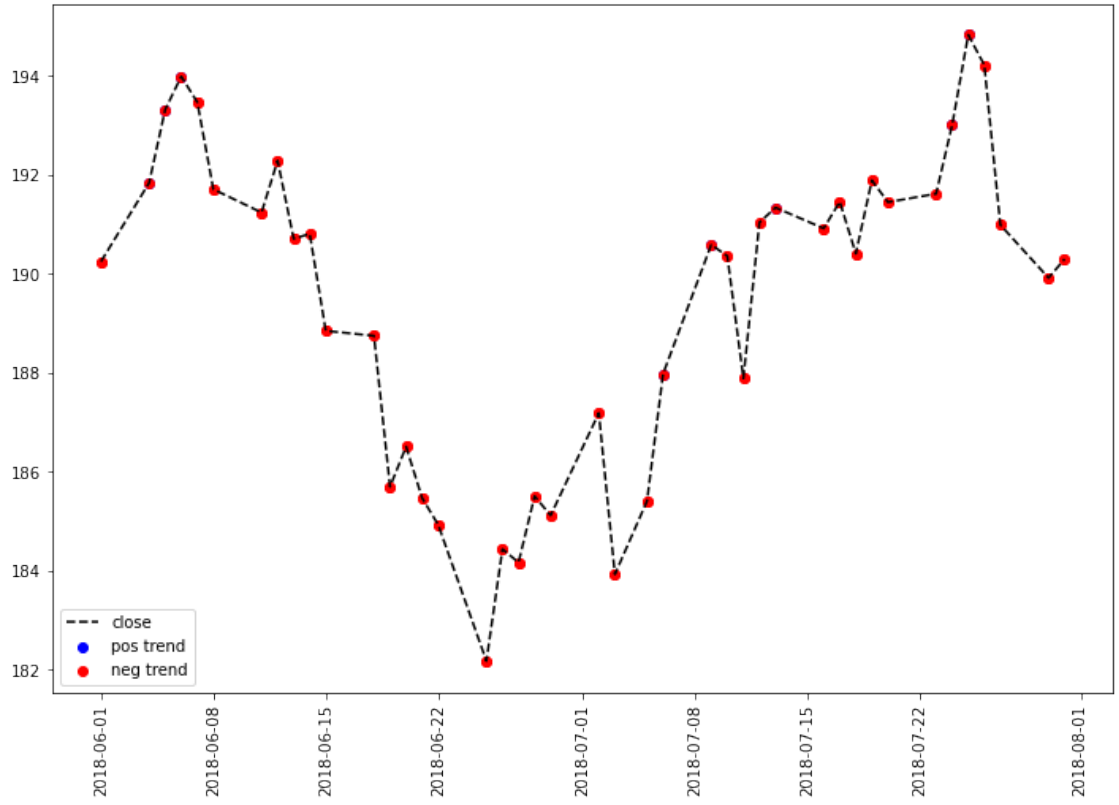
[6]:
```
df.plot(x='date', y='close');
```



You can also embed the `trend_3_day` variable into the time series above.

[7]:
```
start_date = '2018-06-01'
end_date = '2018-07-31'

plt.plot(
    'date', 'close', 'k--',
    data = (
        df.loc[pd.to_datetime(df.date).between(start_date, end_date)]
    )
)

plt.scatter(
```

```
    'date', 'close', color='b', label='pos trend',
    data = (
        df.loc[df.trend_3_day == 1 & pd.to_datetime(df.date).
 ↪between(start_date, end_date)]
    )
)

plt.scatter(
    'date', 'close', color='r', label='neg trend',
    data = (
        df.loc[(df.trend_3_day == -1) & pd.to_datetime(df.date).
 ↪between(start_date, end_date)]
    )
)

plt.legend()
plt.xticks(rotation = 90);
```



[8]:
```
df.shape
```

[8]: (5033, 4)

1.3 Build a Regression Model in Scikit-Learn

In this section you'll train a linear regression model to predict AAPL closing prices when given the previous day's closing price `day_prev_close` and the three day trend `trend_3_day`. A training set and test set are created by sequentially splitting the data after 2000 rows.

[9]:
```
features = ['day_prev_close', 'trend_3_day']
target = 'close'

X_train, X_test = df.loc[:2000, features], df.loc[2000:, features]
y_train, y_test = df.loc[:2000, target], df.loc[2000:, target]
```

[10]:
```
# Create linear regression object
regr = linear_model.LinearRegression(fit_intercept=False)
```

[11]:
```
# Train the model using the training set
regr.fit(X_train, y_train)
```

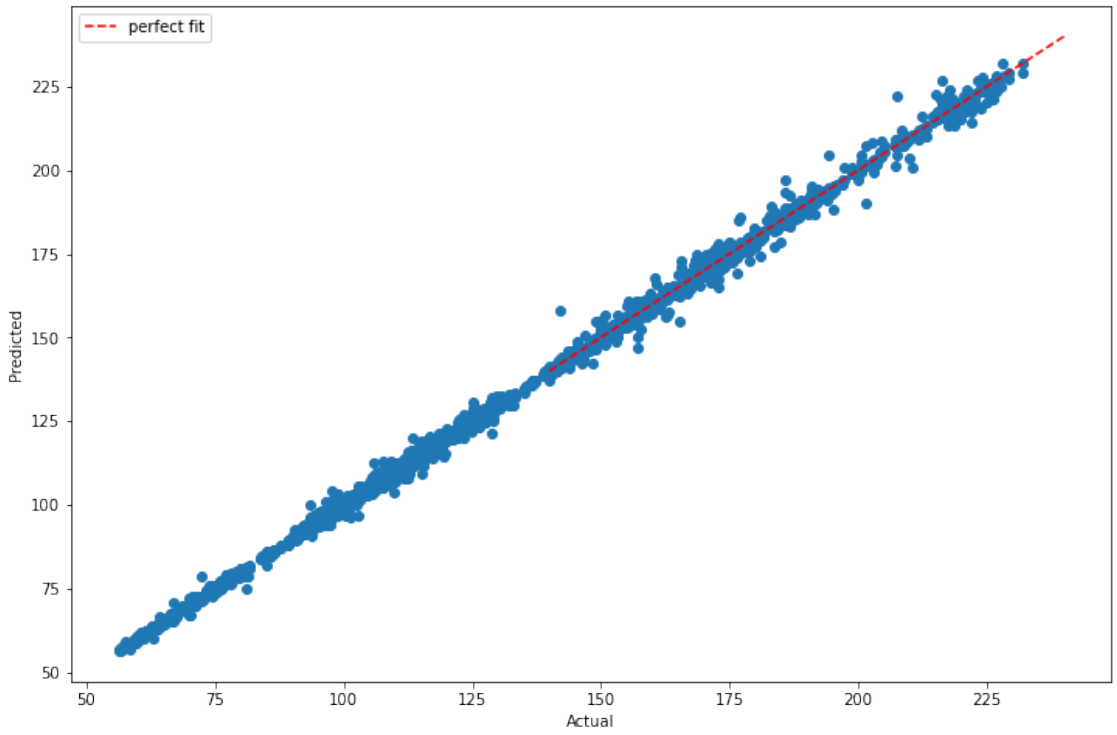
[11]:
```
LinearRegression(fit_intercept=False)
```

[12]:
```
# Make predictions using the testing set
y_pred = regr.predict(X_test)
```

[13]:
```
# The mean squared error
print('Root Mean Squared Error: {0:.2f}'.format(np.
 ↪sqrt(mean_squared_error(y_test, y_pred))))

# Explained variance score: 1 is perfect prediction
print('Variance Score: {0:.2f}'.format(r2_score(y_test, y_pred)))
```

```
Root Mean Squared Error: 1.50
Variance Score: 1.00
```

[14]:
```
plt.scatter(y_test, y_pred)
plt.plot([140, 240], [140, 240], 'r--', label='perfect fit')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.legend();
```



The model's predictions are more or less in line with the truth. However, the utility of the model depends on the business context (i.e. you won't be making any money with this model). It's fair to question whether the variable `trend_3_day` even adds to the performance of the model:

[15]:
```
print('Root Mean Squared Error: {0:.2f}'.format(np.
↪sqrt(mean_squared_error(y_test, X_test.day_prev_close))))
```

```
Root Mean Squared Error: 1.50
```

Indeed, the RMSE is actually lower if we simply use the previous day's closing value as a prediction! Does increasing the number of days included in the trend improve the model? Feel free to create new features and attempt to improve model performance!

[ ]:
```

```