Coca Cola (KO) Stock Closing Price Prediction & Modelling Using Linear & Polynomial Regression

31st March 2022

1 Introduction

In this report I elaborate on my problem formulation for predicting the Coca Cola's next day's closing stock price of the Coca Cola Stock (stock market ticker: 'KO').

The Coca Cola Company is a multinational beverage corporation involved in manufacturing, retailing, and marketing of the aforementioned. It has a history of more than a century in which it has managed to become a household brand in many regions of the world.

However, the specific niche of beverages The Coca Cola Company is involved in is heavily reliant on two ingredients: sugar and caffeine (derived from coffee beans and coffee's stock market ticker: 'KC=F'). From the lens of Finance, there exists a hint at the relation between commodity stock price of Sugar (stock market ticker: 'SB=F') and KO as stated in an article by Stevenson (2017), however, little literature is found on the impact of commodity (sugar and coffee) prices on the KO stock.

Application domain of implementing a machine learning model involving the above will be a step towards expanding the stock price prediction machine learning models to involve inter-stock dependance variability and its subsequent impact on the principal stock price. Additionally, it could be used to improve accuracy of the already existing price prediction models.

Following Section 2 is where the exact problem formulation is discussed alongside exact dataset description and source. In Section 3 the methodology used to apply the machine learning models chosen is described, implemented, and subsequently reasoned. Section 4 is where the results and conclusion is presented.

2 Problem Formulation

The problem I intend to advance in is: given historical stock data of the KO stock alongside data of sugar and coffee commodities, can the KO next day's closing price be predicted accurately?

The **datapoints** of the problem are the stock market data on a single day. The stock market data will be having the following instances/characteristics (data type's are mentioned in square brackets): KO's Opening Price ($) [float], KO's Highest Price ($) [float], KO's Lowest Price ($) [float], KO's Closing Price ($) [float], KO's Volume (No.) [integer], KC=F's Opening Price ($) [float], KC=F's Closing Price ($) [float], KC=F's Average Price ($) [float], SB=F's Opening Price ($) [float], SB=F's Closing Price($) [float], SB=F's Average Price ($) [float]. One datapoint has the characteristics of the stock market data as its candidate **features**. The KC=F's Average Price ($) and SB=F's Average Price ($) feature will be calculated using their historical stock market data by implementing the formula: (opening price + closing price)/2. The resulting average price will be used in as a feature.

The target value is KO's Next Day's Closing Price ($) [float], which is the **label** of a datapoint.

3 Methods

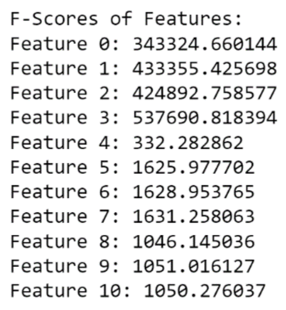
3.1 Methods: Dataset

The range of the set of datapoints we will use is from 1st of January 2010 to 1st of January 2020 (exactly 2513 datapoints), excluding the datapoints with missing features. Our sources of data are as follows:

- KO's Historical Stock Market Data: Yahoo! Finance (Yahoo, 2022c)
- SB=F's Historical Stock Market Data: Yahoo! Finance (Yahoo, 2022b)

- [KC=F’s Historical Stock Market Data: Yahoo! Finance](https://finance.yahoo.com) (Yahoo, 2022a)

3.1 Methods: Feature Selection



```
F-Scores of Features:
Feature 0: 343324.660144
Feature 1: 433355.425698
Feature 2: 424892.758577
Feature 3: 537690.818394
Feature 4: 332.282862
Feature 5: 1625.977702
Feature 6: 1628.953765
Feature 7: 1631.258063
Feature 8: 1046.145036
Feature 9: 1051.016127
Feature 10: 1050.276037
```

Figure 1: F-Score's of Features (in the order mentioned in section 2)

After **visualizing** the data with **scatterplots**, as displayed in Figure 2, a few of the features will be dropped from the dataset. We additionally, verified this using **correlation** statistically through Pearson’s correlation using the *f_regression()* function and *SelectKBest* class, as shown in Figure 1. While, KO’s opening, highest, lowest, and closing price does correlate highly with the label column, KO’s volume is highly uncorrelated to it and we are not doing a weighted financial stock prediction model where volume traded’s impact needs to be taken into account, which is why we will not be selecting this. Moreover, if we have taken into account KC=F’s and SB=F’s average prices, we will be dropping their opening and closing prices because they are highly correlated, and intuitively, it is a repetition of columns because we have calculated average prices using the opening and closing prices.

Additionally, we also notice that KC=F’s Average Price and SB=F’s Average Price has less stronger correlation with our label but we will be investigating this further since it is part of our problem formulation to find out whether their impact optimistically impacts KO’s price prediction model.

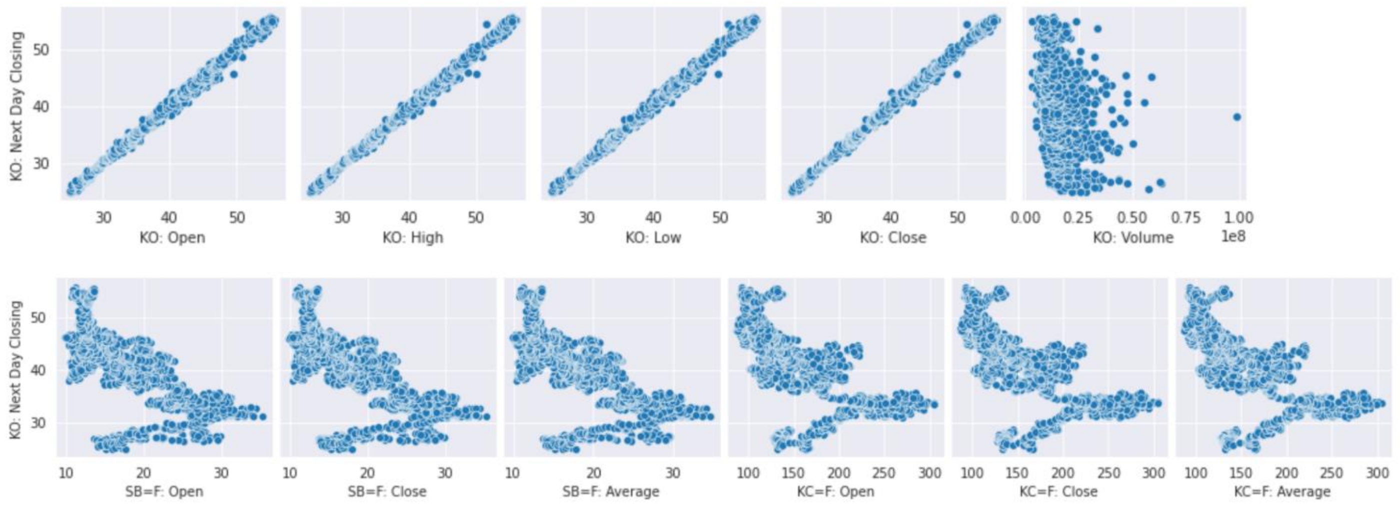


Figure 2: Visual Scatterplots with respect to the Label

3.2 Methods: Visual Summary of Datapoint (Example)

Datapoints	Features						Label
Date	KO: Open	KO: High	KO: Low	KO: Close	KC=F: Average	SB=F: Average	KO: Next Day’s Closing Price
05/01/10	60.00	61.12	59.88	60.60	250.74	18.45	59.97

3.3 Methods: Model (Hypothesis Space) & Motivation

Linear Regression is the first model that is used because a mostly linear relationship can be observed between features and label. Moreover, this model is used because we want to predict a continuous dependent variable (label) from a number of independent continuous variables (features). Additionally, it is simple to implement in a stock price prediction model as it tries to fit the data in the best way linearly which in a financial sense can be employed to predict long-term stock price trends for long-term trend investors and traders.

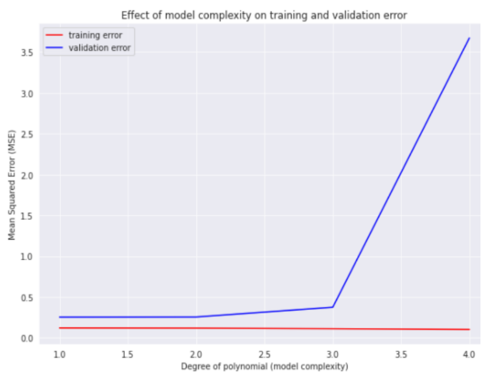


Figure 3

Polynomial Regression is the second model that is used because we are predicting a continuous dependent variable (label) from several independent continuous variables (features). Our motivation to use the model stems from its ability to implement several degrees of fitting to investigate the improvement in the model's accuracy. Moreover, since, there is varying correlation between the features and the labels, as per Fig. 1, it is a recommended method to test varying degrees of polynomial model fitting in search of greater accuracy.

The degrees used to fit the data are 1, 2, and 3. Degree 1 is principally the same as Linear Regression. We had investigated increasing the degrees, however, it immensely continuously increases the validation error (Ev) relative to the training error (Et) after degree 3, as shown in Figure 3, which according to machine learning diagnosis, starts to overfit the data i.e. $Et \ll Ev$.

3.4 Methods: Loss Function Used & Motivation

The mean squared error (MSE) loss is chosen as it allowed the use of a ready-made library for linear regression. The loss is the mean of the squared differences between the true and predicted values. Additionally, other loss functions, such as those less sensitive to outliers (e.g. Huber loss, etc.), are not used here because the data here has very minimum outliers with a minimum deviation from the principle trend of the data because the KO stock has remained stable with no severe or highly volatile market stock price movements over the period we have chosen for our ML methods.

3.5 Methods: Design Choice

The design choice for the model had an 80/20 split between the training and remaining set. I chose such a split ratio because it is common for such ML models related to stock prediction to use such relatively large set for training and a smaller set for the remaining set which constitutes of validation set and testing set. We have then split the remaining set with a 60/40 ratio for the validation and testing set respectively. In all, we had done a 80:12:8 split of the data into training, validation, and testing sets, respectively. Another reason for such a split are the academic reasonings given in the scientific report by Gholamy, Kreinovich and Kosheleva (2018). Additionally, since several machine learning stock prediction models exist online which can be used as reference for splitting, for the scope of this problem, such a split was a common split. The intuitiveness behind it stems from the unpredictable nature of the stock price prediction data.

4.1 Results of the Methods

For each of the implemented method, plots were created and/or calculations pertaining to errors was carried out to analyze the predictions.

Following, in Figure 4, are the plots for the validation set for each of the models along with their validation and training errors mentioned on top.

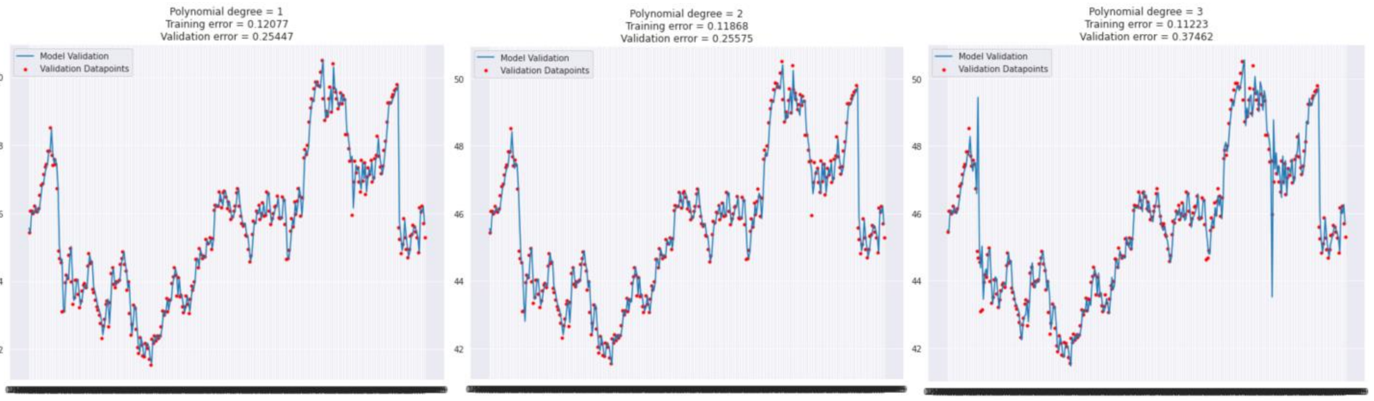


Figure 4: Resulting Plots of Regression used on Validation Set

As can be seen, the closest validation and training error are to each other is when the degree is 1, the same as linear regression. Hence, we then used it on the testing set. And, the result is shown in Figure 5 which seems to be quite accurate to the original data.

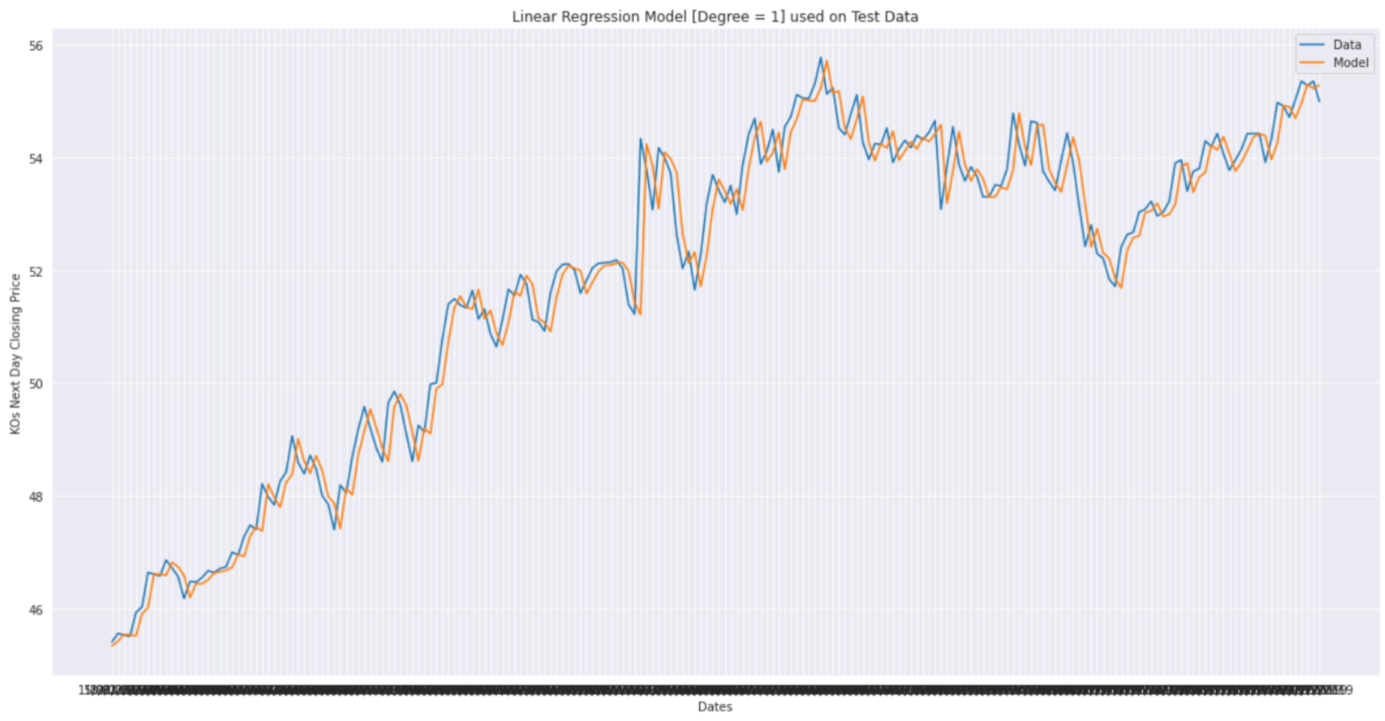


Figure 5: Linear Regression (minimum MSE) used on Testing Set

This is further verified, by calculation of the testing error i.e. 0.2381. Hence, incorporation of the sugar and coffee data alongside KO's own stock data resulted in a very accurate model for stock price prediction.

References

Gholamy, A., Kreinovich, V. and Kosheleva, O. (2018). Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. [online] ScholarWorks@UTEP. Available at: https://scholarworks.utep.edu/cs_techrep/1209/ [Accessed 10 Mar. 2022].

Stevenson, A. (2017). *Cramer's charts predict Coca-Cola on the rise thanks to the price of sugar*. [online] CNBC. Available at: https://www.cnbc.com/2017/01/31/cramers-charts-predict-coca-cola-on-the-rise-thanks-to-the-price-of-sugar.html [Accessed 10 Feb. 2022].

Yahoo (2022a). Coffee May 22 (KC=F) Stock Historical Prices & Data - Yahoo Finance. [online] Yahoo! Finance. Available at: https://finance.yahoo.com/quote/KC%3DF/history?period1=1262304000&period2=1577836800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true [Accessed 1 Mar. 2022].

Yahoo (2022b). Sugar #11 May 22 (SB=F) Stock Historical Prices & Data - Yahoo Finance. [online] Yahoo! Finance. Available at: https://finance.yahoo.com/quote/SB%3DF/history?period1=1262304000&period2=1577836800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true [Accessed 1 Mar. 2022].

Yahoo (2022c). The Coca-Cola Company (KO) Stock Historical Prices & Data - Yahoo Finance. [online] Yahoo! Finance. Available at: https://finance.yahoo.com/quote/KO/history?period1=1262304000&period2=1577836800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true [Accessed 1 Mar. 2022].

Appendix

Stage 3

March 31, 2022

[1]:
```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score
#
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from matplotlib import pyplot
#
import random
sns.set_style("darkgrid")

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
```

[2]:
```
data = pd.read_csv('Data.csv')
data.head(3)
```

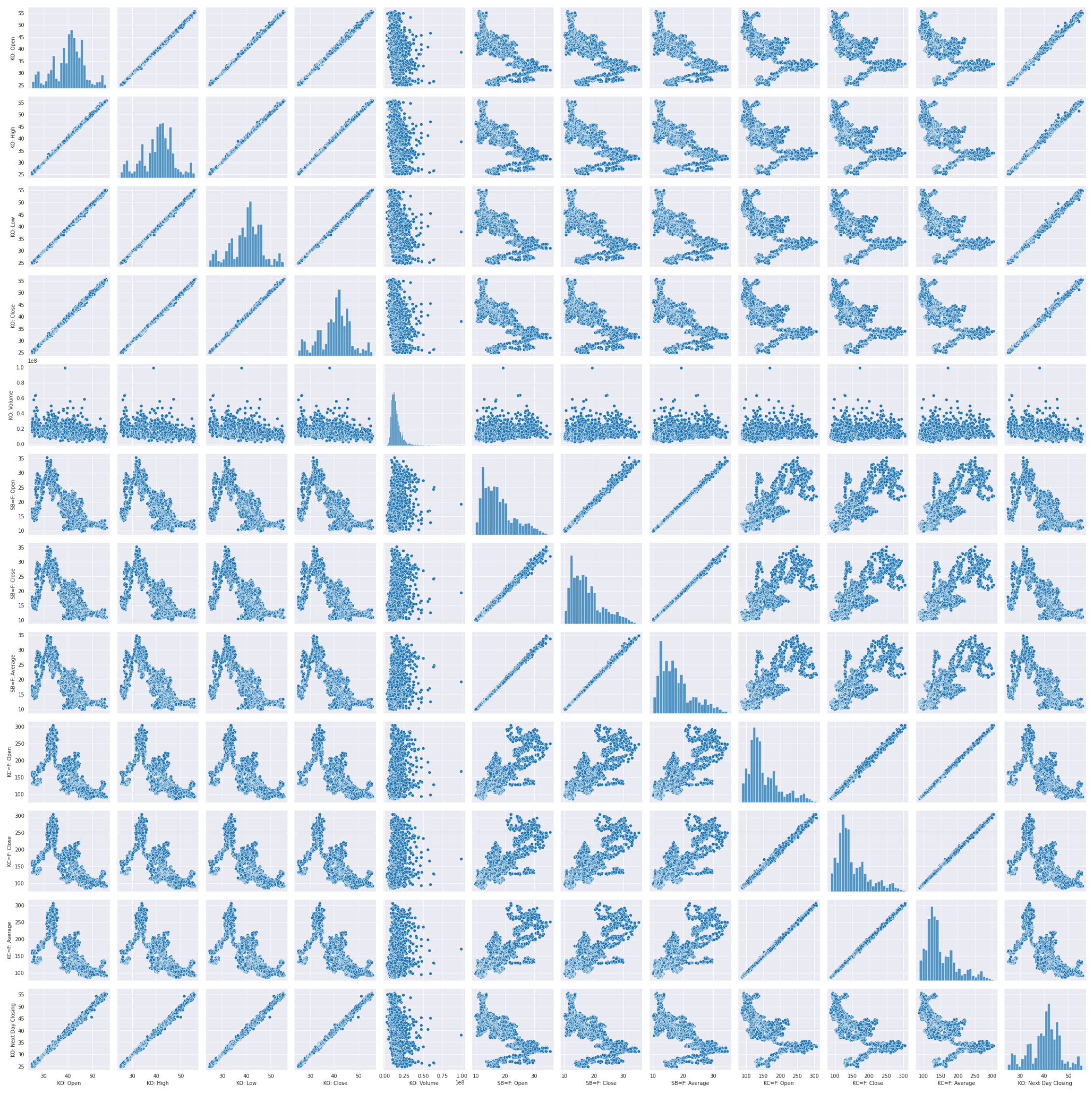
[2]:
```
         Date  KO: Open  KO: High  KO: Low  KO: Close  KO: Volume  SB=F: Open  \
0  04/01/2010     28.58     28.61    28.45      28.52    13870400       27.00
1  05/01/2010     28.42     28.50    28.07      28.17    23172400       27.50
2  06/01/2010     28.17     28.22    27.99      28.17    19264600       27.63

   SB=F: Close  SB=F: Average  KC=F: Open  KC=F: Close  KC=F: Average  \
0        27.62          27.31      136.00       141.85         138.93
1        27.64          27.57      141.85       141.00         141.43
2        28.41          28.02      141.60       141.60         141.60

   KO: Next Day Closing
```

```
0                  28.17
1                  28.17
2                  28.09
```

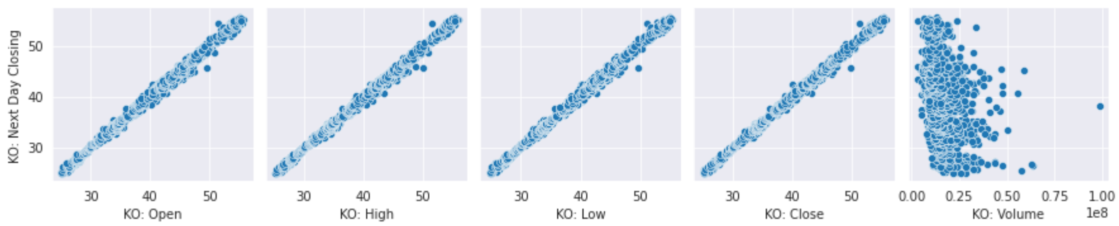
[3]:
```
X = data.loc[:, 'KO: Open':'KC=F: Average'].to_numpy()
y = data.loc[:, 'KO: Next Day Closing'].to_numpy()
```

[4]:
```
sns.pairplot(data)
sns.pairplot(
    data,
    x_vars=["KO: Open", "KO: High", "KO: Low", "KO: Close", "KO: Volume"],
    y_vars=["KO: Next Day Closing"]
)
sns.pairplot(
    data,
    x_vars=["SB=F: Open", "SB=F: Close", "SB=F: Average", "KC=F: Open", "KC=F:␣
 ↪Close", "KC=F: Average"],
    y_vars=["KO: Next Day Closing"]
)
```

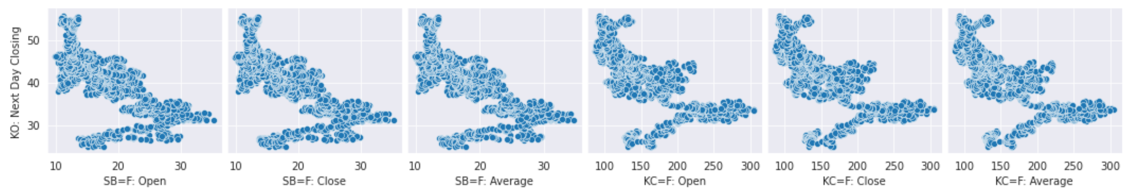
[4]:
```
<seaborn.axisgrid.PairGrid at 0x7fddc45bbc10>
```



KO: Open
KO: High
KO: Low
KO: Close
KO: Volume
SB=F: Open
SB=F: Close
SB=F: Average
KC=F: Open
KC=F: Close
KC=F: Average
KO: Next Day Closing
1e8



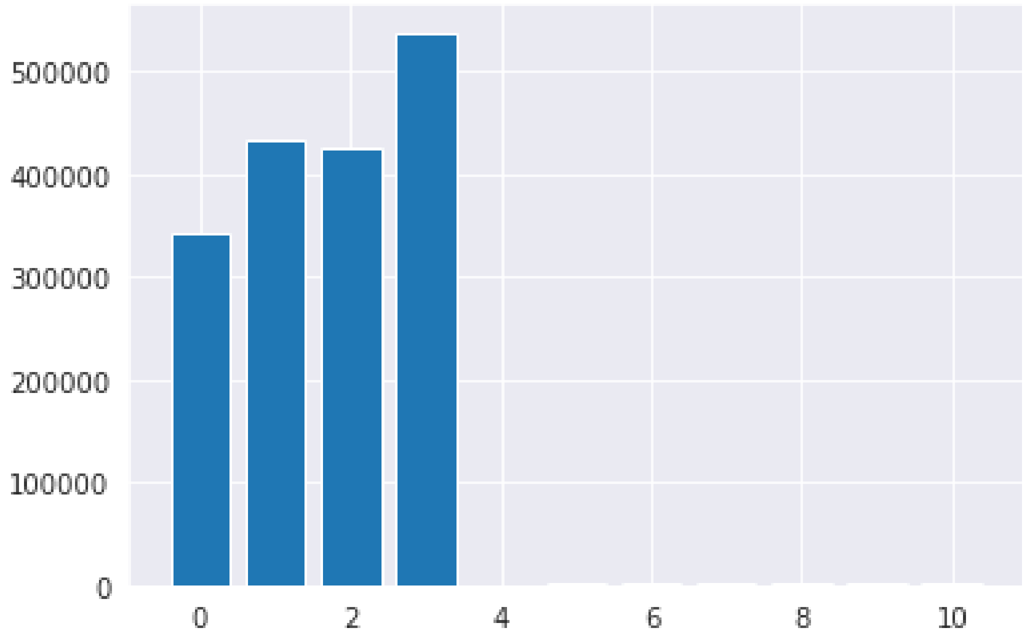
KO: Next Day Closing
KO: Open
KO: High
KO: Low
KO: Close
KO: Volume
1e8



```
[ ]:
```

```
[5]: # feature selection
def select_features(X_train, y_train, X_test):
        # configure to select all features
        fs = SelectKBest(score_func=f_regression, k='all')
        # learn relationship from training data
        fs.fit(X_train, y_train)
        # transform train input data
        X_train_fs = fs.transform(X_train)
        # transform test input data
        X_test_fs = fs.transform(X_test)
        return X_train_fs, X_test_fs, fs

# split X set into training and remaining sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# feature selection
print("F-Scores of Features:")
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test)
# what are scores for the features
for i in range(len(fs.scores_)):
        print('Feature %d: %f' % (i, fs.scores_[i]))
# plot the scores
pyplot.bar([i for i in range(len(fs.scores_))], fs.scores_)
pyplot.show()
```

```
F-Scores of Features:
Feature 0: 343324.660144
Feature 1: 433355.425698
Feature 2: 424892.758577
Feature 3: 537690.818394
Feature 4: 332.282862
Feature 5: 1625.977702
Feature 6: 1628.953765
Feature 7: 1631.258063
Feature 8: 1046.145036
Feature 9: 1051.016127
Feature 10: 1050.276037
```



```
[6]: data = pd.read_csv('Data.csv')
     data.columns
```

```
[6]: Index(['Date', 'KO: Open', 'KO: High', 'KO: Low', 'KO: Close', 'KO: Volume',
            'SB=F: Open', 'SB=F: Close', 'SB=F: Average', 'KC=F: Open',
            'KC=F: Close', 'KC=F: Average', 'KO: Next Day Closing'],
           dtype='object')
```

```
[7]: # reprocess data by dropping Volume, KC=F: Open, KC=F: Close, SB=F: Open, SB=F:␣
      ↪Close.
     data = data.drop(columns=['KO: Volume', 'KC=F: Open', 'KC=F: Close', 'SB=F:␣
      ↪Open', 'SB=F: Close'])
     data.head(3)


     X = data.loc[:, 'KO: Open':'KC=F: Average'].to_numpy()
     y = data.loc[:, 'KO: Next Day Closing'].to_numpy()

     # split X set into training and remaining sets

     split = int(len(X)*0.8)
     X_train, X_rem, y_train, y_rem = X[:split], X[split:], y[:split], y[split:]

     dates = np.arange(len(X_rem))
```

```
# split remaining set into validation and testing sets
X_val, X_test, y_val, y_test, d1, d2 = train_test_split(X_rem, y_rem, dates,␣
↪test_size=0.4, random_state=42, shuffle=False)
```

[ ]:
```

```

[8]:
```
dates_train = np.array(data.iloc[:split]['Date'])
print(dates_train.shape)
#dates_train
```

```
(2009,)
```

[9]:
```
dates_test = np.array(data.iloc[d2+split]['Date'])
print(dates_test.shape)
#dates_test
```

```
(202,)
```

[10]:
```
dates_val = np.array(data.iloc[d1+split]['Date'])
print(dates_val.shape)
#dates_val
```

```
(301,)
```

[11]:
```
data.head(5)
```

[11]:
```
         Date  KO: Open  KO: High  KO: Low  KO: Close  SB=F: Average  \
0  04/01/2010     28.58     28.61    28.45      28.52          27.31
1  05/01/2010     28.42     28.50    28.07      28.17          27.57
2  06/01/2010     28.17     28.22    27.99      28.17          28.02
3  07/01/2010     28.17     28.18    27.88      28.09          28.12
4  08/01/2010     27.73     27.82    27.38      27.58          27.73

   KC=F: Average  KO: Next Day Closing
0         138.93                 28.17
1         141.43                 28.17
2         141.60                 28.09
3         141.72                 27.58
4         143.90                 28.14
```

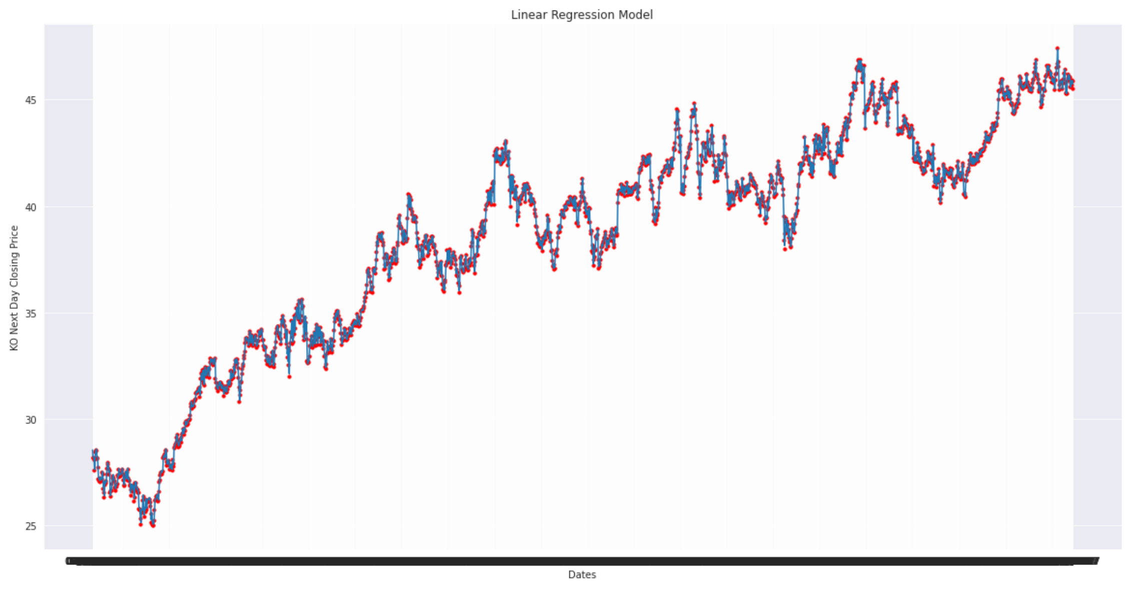
[12]:
```
print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_val: ", X_val.shape)
print("Shape of y_val: ", y_val.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (2009, 6)
Shape of y_train:  (2009,)
Shape of X_val:  (301, 6)
Shape of y_val:  (301,)
Shape of X_test:  (202, 6)
Shape of y_test:  (202,)
```

[13]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)

y_train_pred = lr.predict(X_train)
tr_error = mean_squared_error(y_train, y_train_pred)
accuracy = 1-tr_error
y_pred_val = lr.predict(X_val)
val_error = mean_squared_error(y_val, y_pred_val)

print('Linear Regression\nTraining error = {:.5}\nValidation error = {:.5}'.
 ↪format(tr_error, val_error))
```

```
Linear Regression
Training error = 0.12077
Validation error = 0.25447
```

[17]:
```
plt.figure(figsize=(20, 10))
plt.plot(dates_train, y_train_pred, label = "Model")
plt.scatter(dates_train, y_train, color="r", s=10, label="Train Datapoints")
plt.title('Linear Regression Model')
plt.xlabel('Dates')
plt.ylabel('KO Next Day Closing Price')
plt.show()
```



```
[19]: print('The shape of y_pred is: ', y_train_pred.shape) #to check that the shape␣
       ↪of the predicted y is the same as the training y

      print('The prediction accuracy is: ', accuracy)    # print the training error
      print("w1 = ", lr.coef_)   # print the learnt w1
      print("w0 = ", lr.intercept_)
```

```
The shape of y_pred is:  (2009,)
The prediction accuracy is:  0.879227983530783
w1 =  [-1.65261362e-02  1.30033830e-01  6.70219746e-02  8.16205513e-01
 -1.21143599e-03  2.19497010e-05]
w0 =  0.14131042370286906
```

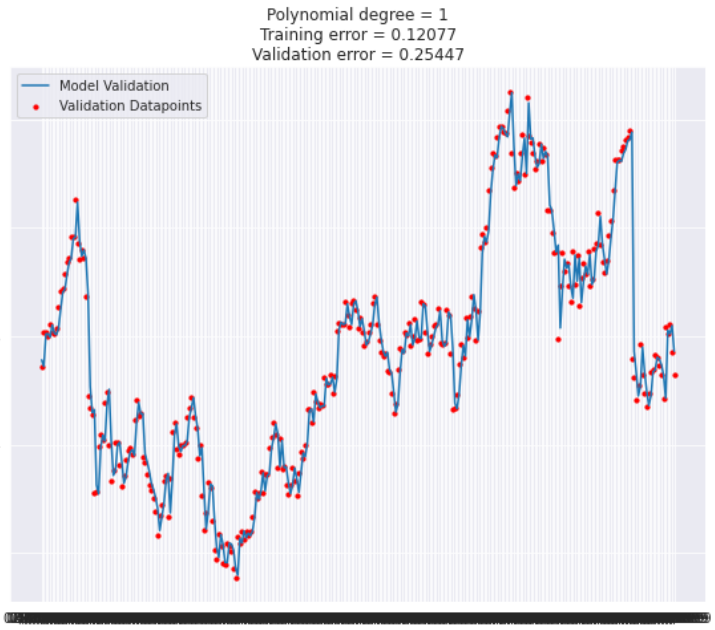
```
[ ]:
```

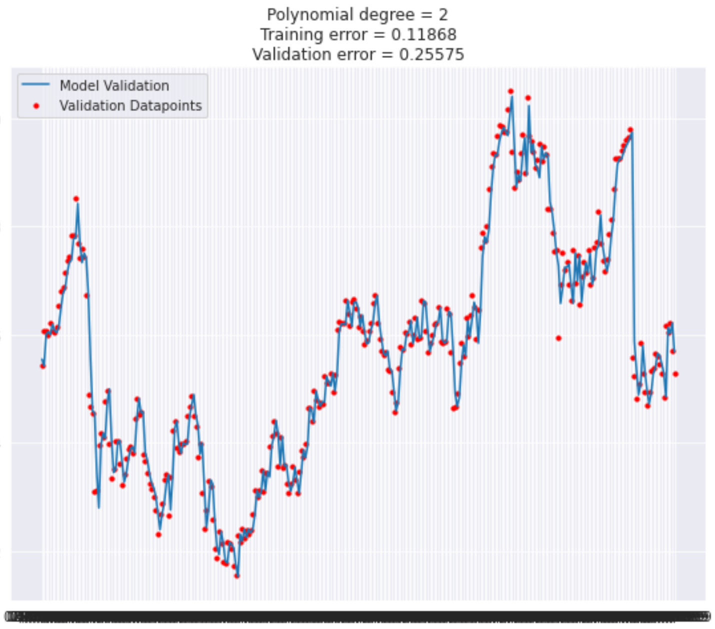
```
[ ]:
```

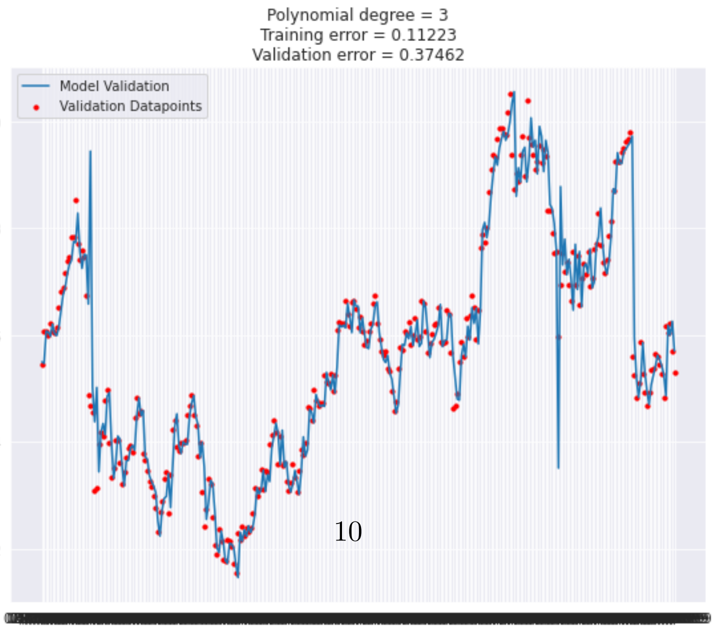
```
[20]: ## define a list of values for polynomial degrees
      degrees = [1, 2, 3]

      # declare an array variable to store the resulting training and validation␣
       ↪errors for each polynomial degree
      tr_errors, val_errors = [], []

      plt.figure(figsize=(8, 20))
      for i, degree in enumerate(degrees):    # use for-loop to fit polynomial␣
       ↪regression models with different degrees
          plt.subplot(len(degrees), 1, i + 1)

          print("Polynomial degree = ",degree)

          lin_regr = LinearRegression(fit_intercept = False)

          poly = PolynomialFeatures(degree=degree)
          X_train_poly = poly.fit_transform(X_train)
          lin_regr.fit(X_train_poly, y_train)

          # y_pred_train = ...    # predict values for the training data using the␣
       ↪linear model
          # tr_error = ...    # calculate the training error
          # X_val_poly = ... # transform the raw features for the validation data
          # y_pred_val = ... # predict values for the validation data using the␣
       ↪linear model
          # val_error = ... # calculate the validation error
```

```
    # YOUR CODE HERE
    y_pred_train = lin_regr.predict(X_train_poly)
    tr_error = mean_squared_error(y_train, y_pred_train)
    X_val_poly = poly.fit_transform(X_val)
    y_pred_val = lin_regr.predict(X_val_poly)
    val_error = mean_squared_error(y_val, y_pred_val)

    tr_errors.append(tr_error)
    val_errors.append(val_error)
    print('Polynomial degree = {}\nTraining error = {:.5}\nValidation error = {:
↪.5}'.format(degree, tr_error, val_error))    # set the title    # show the␣
↪plot

    plt.tight_layout()
    plt.plot(dates_val, y_pred_val, label="Model Validation")
    plt.scatter(dates_val, y_val, color="r", s=10, label="Validation␣
↪Datapoints")
    plt.legend(loc="best")
    plt.title(f'Polynomial degree = {degree}\nTraining error = {tr_error:.
↪5}\nValidation error = {val_error:.5}')    # set the title


# sanity check
assert len(tr_errors) == 3 # check the length of array tr_errors
```

```
Polynomial degree =  1
Polynomial degree = 1
Training error = 0.12077
Validation error = 0.25447
Polynomial degree =  2
Polynomial degree = 2
Training error = 0.11868
Validation error = 0.25575
Polynomial degree =  3
Polynomial degree = 3
Training error = 0.11223
Validation error = 0.37462
```

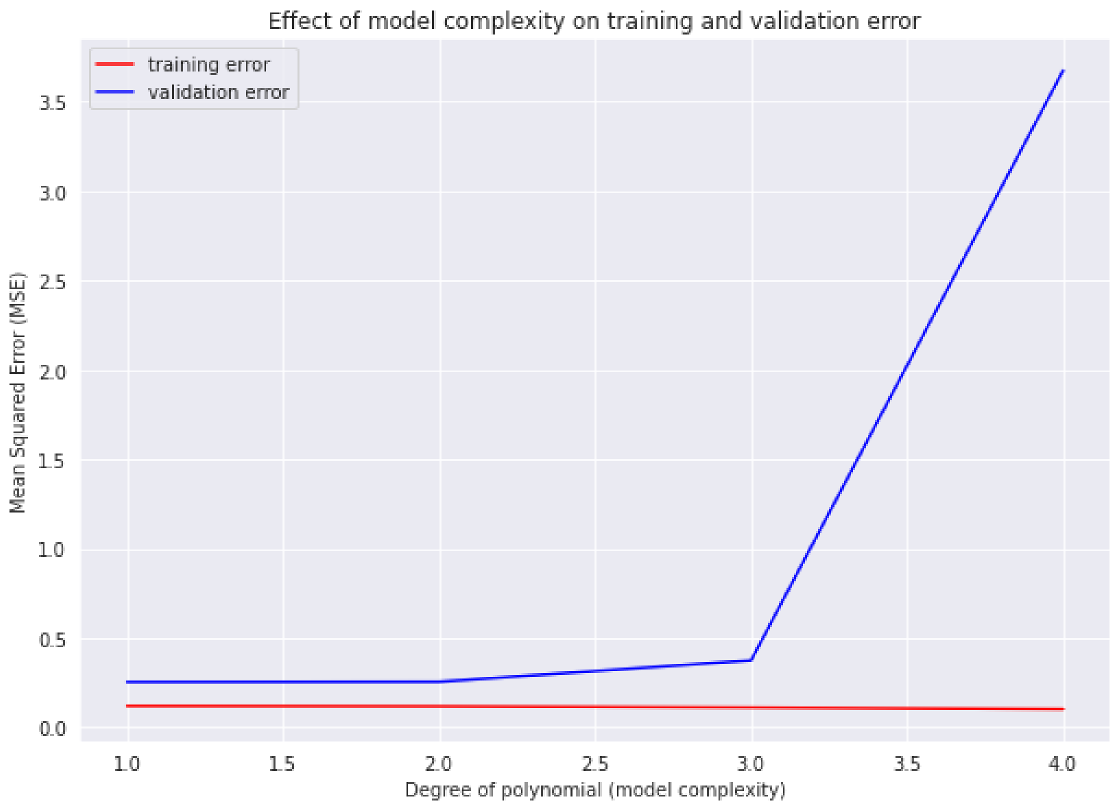

Polynomial degree = 1
Training error = 0.12077
Validation error = 0.25447
Model Validation
Validation Datapoints
50
48
46
44
42


Polynomial degree = 2
Training error = 0.11868
Validation error = 0.25575
Model Validation
Validation Datapoints
50
48
46
44
42


Polynomial degree = 3
Training error = 0.11223
Validation error = 0.37462
Model Validation
Validation Datapoints
50
48
46
44
42

[98]:
```
train_errs = []
val_errs = []

def train_validate_poly(X_train, y_train, X_val, y_val, model=LinearRegression,␣
 ↪degree=5):
    # train a polynomial model and validate it

    # generate polynomial features
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    X_train_poly = poly_features.fit_transform(X_train)
    X_val_poly = poly_features.transform(X_val)

    # learn the model and validate it
    reg = model()
    reg = reg.fit(X_train_poly, y_train)
    y_train_pred = reg.predict(X_train_poly)
    y_val_pred = reg.predict(X_val_poly)
    train_err = mean_squared_error(y_train, y_train_pred)
    val_err = mean_squared_error(y_val, y_val_pred)
    return reg, train_err, val_err


# get training and validation error for different degrees
orders = range(1, 5)
for order in orders:
    _, train_err, val_err = train_validate_poly(X_train, y_train, X_val, y_val,␣
 ↪degree=order)
    train_errs.append(train_err)
    val_errs.append(val_err)

# create the plot
fig, axs = plt.subplots(figsize=(10, 7))
axs.plot(orders, train_errs, color='red', label='training error')
axs.plot(orders, val_errs, color='blue', label='validation error')
axs.set_xlabel('Degree of polynomial (model complexity)')
axs.set_ylabel('Mean Squared Error (MSE)')
axs.set_title('Effect of model complexity on training and validation error')
axs.legend()
```

[98]:
```
<matplotlib.legend.Legend at 0x7fc0f292b790>
```

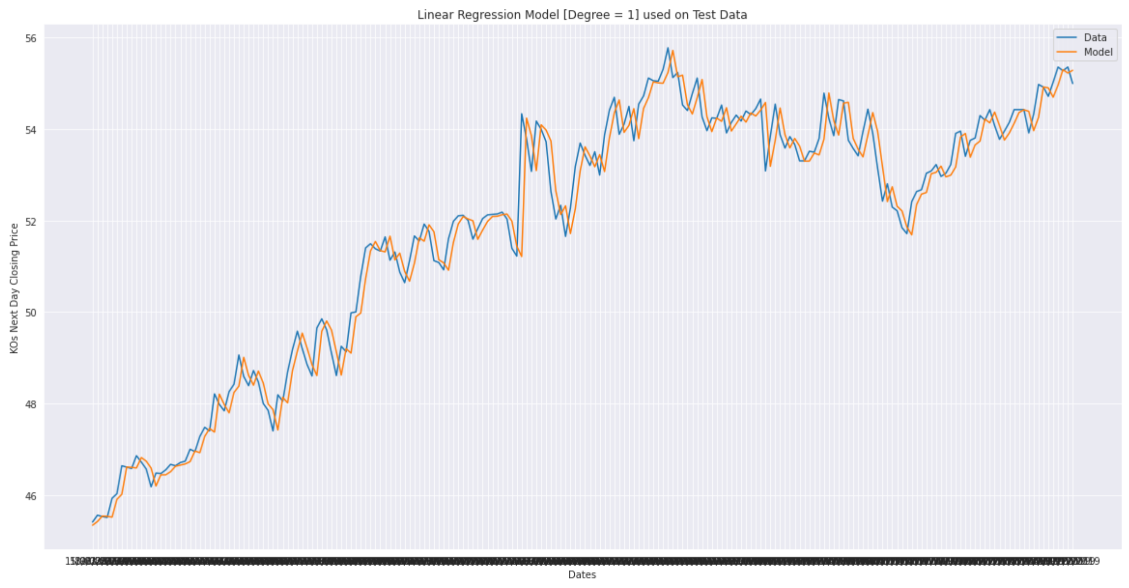


[21]:
```
lin_regr = LinearRegression(fit_intercept=False)
poly = PolynomialFeatures(degree=1)     # generate polynomial features with␣
 ↪degree of 1
X_train_poly = poly.fit_transform(X_train)     # fit the test features
lin_regr.fit(X_train_poly, y_train)

X_test_poly = poly.fit_transform(X_test)
y_pred_test = lin_regr.predict(X_test_poly)
test_error = mean_squared_error(y_test, y_pred_test)
```

[22]:
```
print("The final test error using the Linear Regression (Polynomial Degree = 1)␣
 ↪is: ",test_error)
```

```
The final test error using the Linear Regression (Polynomial Degree = 1) is:
0.2381176511514003
```

[23]:
```
plt.figure(figsize=(20, 10))
plt.plot(dates_test, y_test, label = "Data")
plt.plot(dates_test, y_pred_test, label = "Model")
plt.title('Linear Regression Model [Degree = 1] used on Test Data')
plt.xlabel('Dates')
plt.ylabel('KOs Next Day Closing Price')
plt.legend(loc="best")
plt.show()
```



```
[ ]:
```