Black Swan Anomalies Stock Market Analysis

Authors

Oluwatamilore (Tami) Ajibade, Melina Nguyen, Oluwatobi (Tobi) Oyinloye

Introduction

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving weekly_IBM.csv to weekly_IBM (2).csv
```

```
import pandas as pd

stocks_df = pd.read_csv("weekly_IBM.csv")
stocks_df.head()
```

	timestamp	open	high	low	close	volume
0	2023-12-01	154.99	160.590	154.75	160.55	21900644
1	2023-11-24	152.51	155.705	152.35	155.18	11362696
2	2023-11-17	148.46	153.500	147.35	152.89	19547595
3	2023-11-10	147.89	149.680	145.28	149.02	18357944
4	2023-11-03	143.19	148.445	142.58	147.90	22959464

We also will define our start dates and end dates for our three anomalies.

```
## anomaly 1 - 9/11
# 4 months of 9/11 financial crisis
anom_1a_start = "2001-09-11"
anom_1a_end = "2002-01-11"

# 4 months BEFORE 9/11 financial crisis
anom_1b_start = "2001-05-10"
anom_1b_end = "2001-09-10"

# 4 months AFTER 9/11 financial crisis
anom_1c_start = "2002-01-11"
anom_1c_end = "2002-05-01"

## anomaly 2 - 2008 Housing Market Crash
# 1 year of 2008 Housing financial crisis
anom_2a_start = "2008-01-01"
anom_2a_end = "2008-12-31"

# 1 year BEFORE 2008 Housing financial crisis
anom_2b_start = "2007-01-01"
anom_2b_end = "2007-12-31"

# 1 year AFTER 2008 Housing financial crisis
anom_2c_start = "2009-01-01"
anom_2c_end = "2009-12-31"

## anomaly 3 - COVID-19 pandemic
# 2 months of COVID-19 financial crisis
anom_2a_start = "2020-03-01"
anom_2a_end = "2020-04-31"

# 2 months BEFORE COVID-19 financial crisis
anom_2b_start = "2020-01-01"
anom_2b_end = "2020-02-29"

# 2 months AFTER COVID-19 financial crisis
anom_2c_start = "2020-05-01"
anom_2c_end = "2009-07-01"
```

Methodology and Results

Data Preparation

We apply a log transformation to our data. Stock prices are typically positively skewed and exhibit heteroscedasticity (the spread of data points changes as the values of the independent variable change). Applying a log transformation to the stock prices can help stabilize variances and make the data more suitable for time series analysis or modeling.

```
import numpy as np
def logs_to_stocks(x):
  if pd.api.types.is_numeric_dtype (x):
    return np.log(x)
  else:
    return x

logs_df = stocks_df.apply(logs_to_stocks)
logs_df.head()
```

	timestamp	open	high	low	close	volume
0	2023-12-01	5.043361	5.078855	5.041811	5.078605	16.902027
1	2023-11-24	5.027230	5.047963	5.026181	5.044586	16.245846
2	2023-11-17	5.000316	5.033701	4.992811	5.029719	16.788363
3	2023-11-10	4.996469	5.008500	4.978663	5.004081	16.725573
4	2023-11-03	4.964172	5.000215	4.959903	4.996536	16.949241

Exploratory Statistics

We perform t-tests to compare the mean values of the stock price metrics (e.g., close, adjusted close) on days with scientifically verifiable black swan anomalies and days without anomalies. This will help identify if and when there are significant differences in stock price.

```
import pandas as pd
from scipy.stats import ttest_ind
from datetime import datetime


#T-TEST FUNCTION
def perform_t_tests(start_date_anomaly, end_date_anomaly, start_date_before, end_date_before, start_date_after, end_date_after):
  # Filter the DataFrame for the specified date ranges: ANOMALY, BEFORE, AFTER
  selected_days_anomaly = logs_df[(logs_df['timestamp'] >= start_date_anomaly) & (logs_df['timestamp'] <= end_date_anomaly)]
  selected_days_before = logs_df[(logs_df['timestamp'] >= start_date_before) & (logs_df['timestamp'] <= end_date_before)]
  selected_days_after = logs_df[(logs_df['timestamp'] >= start_date_after) & (logs_df['timestamp'] <= end_date_after)]

  columns_to_compare = ['close', 'open', 'low', 'volume', 'high']
  significance_level=0.05

    # T-TESTS
  for column in columns_to_compare:
    t_statistic_before_after, p_value_before_after = ttest_ind(selected_days_before[column], selected_days_after[column], equal_v
    t_statistic_anomaly_before, p_value_anomaly_before = ttest_ind(selected_days_anomaly[column], selected_days_before[column], e
    t_statistic_anomaly_after, p_value_anomaly_after = ttest_ind(selected_days_anomaly[column], selected_days_after[column], equa

    # BEFORE VS AFTER
    if p_value_before_after < significance_level:
      print(f'T-Test for {column} - Before vs After:')
      print(f'T-Statistic: {t_statistic_before_after}')
      print(f'P-Value: {p_value_before_after}')
      print('\n')

    # ANOMALY VS BEFORE
    if p_value_anomaly_before < significance_level:
      print(f'T-Test for {column} - Anomaly vs Before:')
      print(f'T-Statistic: {t_statistic_anomaly_before}')
      print(f'P-Value: {p_value_anomaly_before}')
      print('\n')

    # ANOMALY VS AFTER
    if p_value_anomaly_after < significance_level:
      print(f'T-Test for {column} - Anomaly vs After:')
      print(f'T-Statistic: {t_statistic_anomaly_after}')
      print(f'P-Value: {p_value_anomaly_after}')
      print('\n')


#THIS CODE BOX CONTAINS 9/11 T-TEST

#4 Months ANAMOLY 9/11 Financial Crisis
anomaly_1_start_date = '2001-09-11'
anomaly_1_end_date = '2002-01-11'

#4 Months  BEFORE 9/11 Financial Crisis

b_anomaly_1_start_date = '2001-05-10'
b_anomaly_1_end_date = '2001-09-10'

#4 Months AFTER 9/11 Financial Crisis
a_anomaly_1_start_date = '2002-01-11'
a_anomaly_1_end_date = '2002-05-01'

perform_t_tests('2001-09-11','2002-01-11', '2001-05-10','2001-09-10', '2002-01-11','2002-05-01' )
```

```
T-Test for close - Before vs After:
T-Statistic: 2.111784983046713
P-Value: 0.04524875041666402


T-Test for close - Anomaly vs After:
T-Statistic: 2.429314204473413
P-Value: 0.02111703762250402


T-Test for low - Before vs After:
T-Statistic: 2.1588812563206043
P-Value: 0.04144662691352722


T-Test for volume - Before vs After:
T-Statistic: -2.5162782857218877
```

```
    P-Value: 0.017260649616976455


    T-Test for high - Anomaly vs After:
    T-Statistic: 2.071689124225216
    P-Value: 0.04678480903365964
```

```
#THIS CODE-BOX CONTAINS 2008 T-TEST

#1 Year ANAMOLY 2008 Housing Financial Crisis
anomaly_2_start_date = '2008-01-01'
anomaly_2_end_date = '2008-12-31'

#1 Year BEFORE 2008 Financial Crisis

b_anomaly_2_start_date = '2007-01-01'
b_anomaly_2_end_date ='2007-12-31'

#1 Year AFTER 9/11 Financial Crisis
a_anomaly_2_start_date = '2009-01-01'
a_anomaly_2_end_date = '2009-12-31'


perform_t_tests('2008-01-01','2008-12-31', '2007-01-01','2007-12-31', '2009-01-01','2009-12-31')
```

```
    T-Test for volume - Anomaly vs Before:
    T-Statistic: 2.703680566429757
    P-Value: 0.008039183516076407


    T-Test for volume - Anomaly vs After:
    T-Statistic: 2.3893412999003156
    P-Value: 0.01874745320618078


    T-Test for high - Anomaly vs Before:
    T-Statistic: 2.0613814883498924
    P-Value: 0.04267754786835433
```

```
#THIS CODE-BOX CONTAINS COVID-19 T-TEST

#2 Month ANAMOLY COVID-19 Financial Crisis
anomaly_3_start_date = '2020-03-01'
anomaly_3_end_date = '2020-04-31'

#2 Months BEFORE COVID-19 Financial Crisis
b_anomaly_3_start_date ='2020-01-01'
b_anomaly_3_end_date ='2020-02-29'

#2 Months AFTER COVID-19 Financial Crisis
a_anomaly_3_start_date ='2020-05-01'
a_anomaly_3_end_date =' 2020-07-01'

perform_t_tests('2020-03-01','2020-04-31', '2020-01-01','2020-02-29', '2020-05-1',' 2020-07-1 ')
```

```
    T-Test for close - Anomaly vs Before:
    T-Statistic: -5.555126166757153
    P-Value: 0.00018585660045399157


    T-Test for open - Anomaly vs Before:
    T-Statistic: -5.390561496627213
    P-Value: 0.00038636806876509366


    T-Test for low - Anomaly vs Before:
    T-Statistic: -6.328028509682065
    P-Value: 6.838526834613872e-05


    T-Test for high - Anomaly vs Before:
    T-Statistic: -5.878570899432568
    P-Value: 5.904096192407748e-05
```

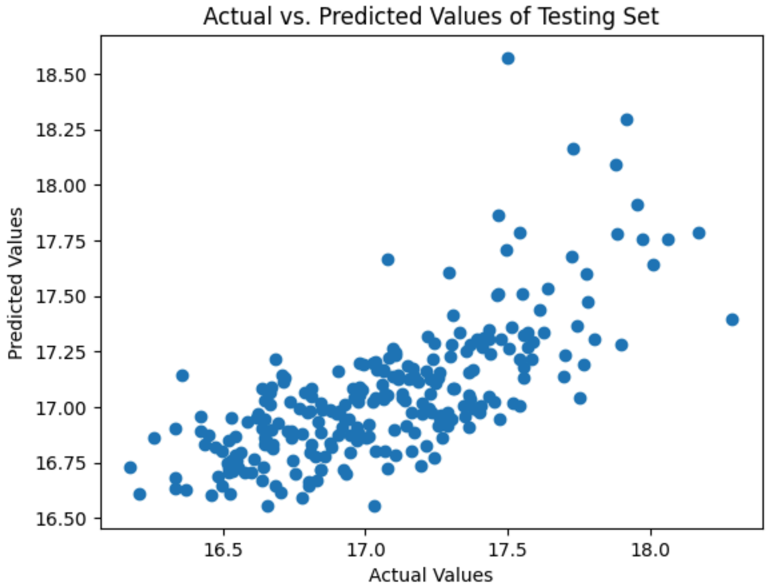
Machine Learning

We perform multiple regression to analyze the relationship between a single dependent (target) variable — volume — and several independent (predictor) variables, such as open, close, high and low stock price. We also created regressor trees in which our target variable took on continuous values instead of class labels in the leaves.

We varied all the parameters with the log transformation. We also performed our machine learning methods on all four of our independent variables and tried to take out additional ones to see how it would change our results. For example, we would look at high and low stock prices and omit open and close stock prices.

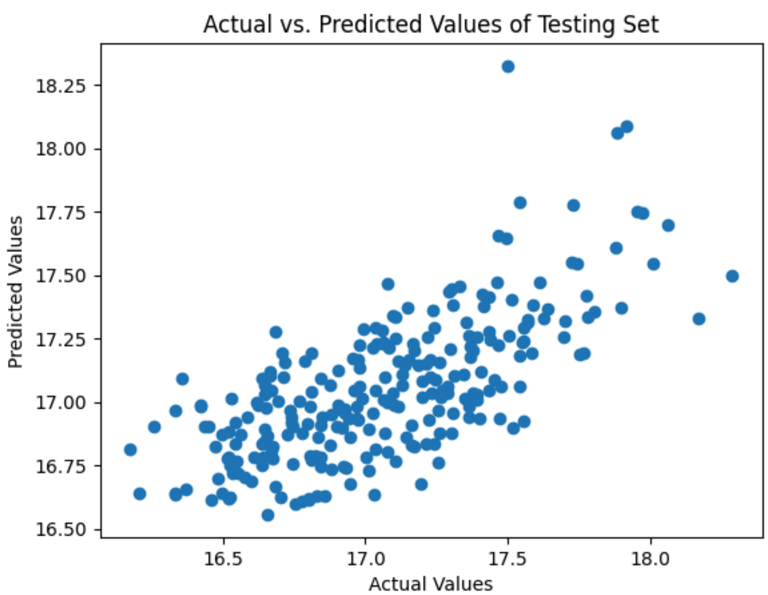
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from statsmodels.stats.outliers_influence import variance_inflation_factor

def mult_lin_reg(df, features, target):
  # convert columns into numpy arrays
  X = df[features].values
  y = df[target].values

  # split data into training and testing sets
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

  #train linear regression model
  lin_model = LinearRegression().fit(X_train, y_train)

  # Calculate VIF for each feature
  df_features = pd.DataFrame(X, columns=features)
  vif_data = pd.DataFrame()
  vif_data["Variable"] = df_features.columns
  vif_data["VIF"] = [variance_inflation_factor(df_features.values, i) for i in range(df_features.shape[1])]

  print("Variance Inflation Factor:")
  print(vif_data)

  # generate predictions
  y_train_pred = lin_model.predict(X_train)
  y_test_pred = lin_model.predict(X_test)

  # evaluate scores (coefficient of determination, R^2)
  train_r2 = lin_model.score(X_train, y_train)
  test_r2 = lin_model.score(X_test, y_test)

  print(f"Training R-squared: {train_r2}")
  print(f"Testing R-squared: {test_r2}")

  # plot final multiple regression model
  plt.scatter(y_test, y_test_pred)
  plt.xlabel("Actual Values")
  plt.ylabel("Predicted Values")
  plt.title("Actual vs. Predicted Values of Testing Set")
  plt.show()


mult_lin_reg(logs_df, ["open", "high", "low", "close"], "volume")
```

```
Variance Inflation Factor:
  Variable            VIF
0     open  120073.014719
1     high  139226.667768
2      low  119722.461206
3    close  130485.120974
Training R-squared: 0.5344901432537585
Testing R-squared: 0.5195177560977121
```



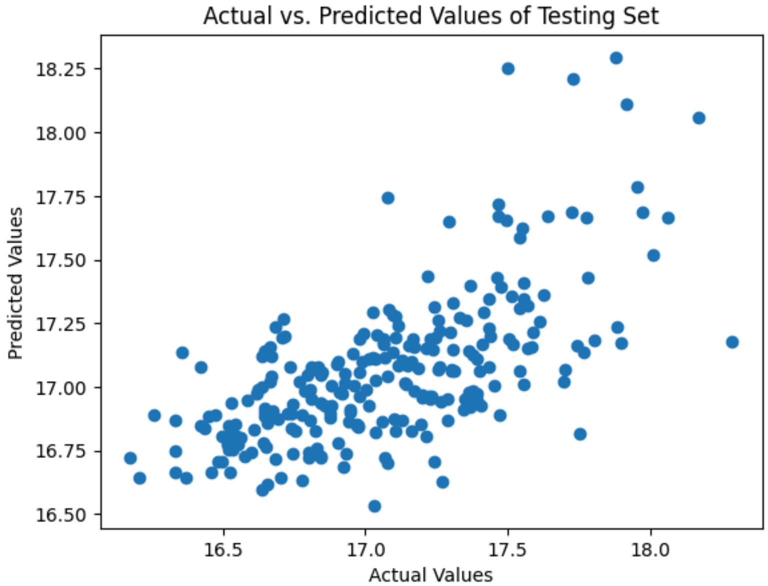

```
mult_lin_reg(logs_df, ["open", "close", "high"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0     open  41768.493671
1    close  39768.307110
2     high  83995.761698
Training R-squared: 0.4790791402045186
Testing R-squared: 0.49567243765317404
```



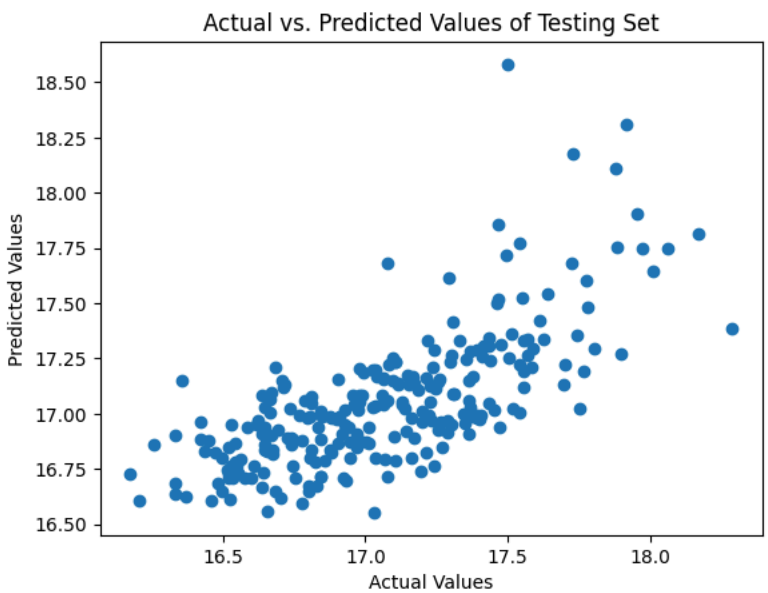

```
mult_lin_reg(logs_df, ["open", "close", "low"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0     open  35110.499663
1    close  40738.182948
2      low  72228.830026
Training R-squared: 0.47390902251660805
Testing R-squared: 0.4266113601708894
```



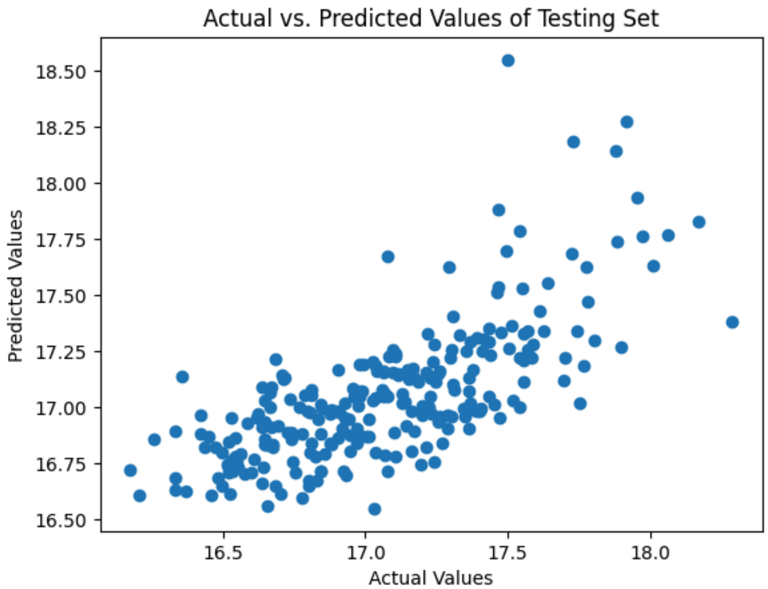

```
mult_lin_reg(logs_df, ["open", "high", "low"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0     open  64033.012674
1     high  43467.342640
2      low  36488.141864
Training R-squared: 0.5342192971911763
Testing R-squared: 0.5152884393953694
```



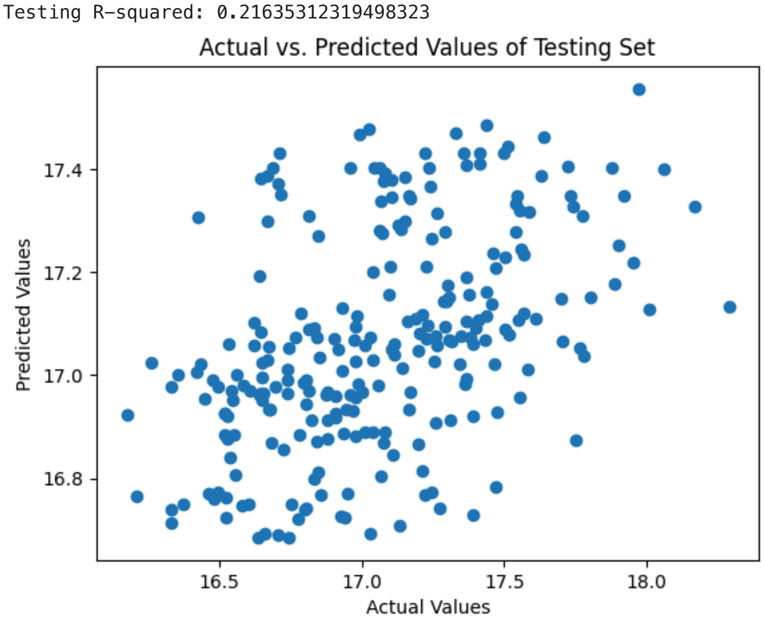

```
mult_lin_reg(logs_df, ["close", "high", "low"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0    close  69585.621921
1     high  40711.211284
2      low  41646.550433
Training R-squared: 0.5336692907921204
Testing R-squared: 0.5170137688249116
```



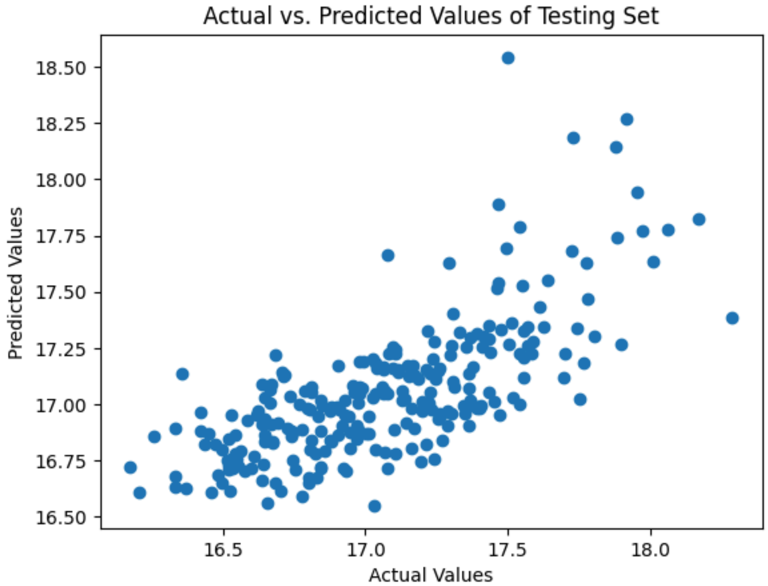

```
mult_lin_reg(logs_df, ["open", "close"], "volume")
```

```
Variance Inflation Factor:
  Variable          VIF
0     open  19757.53683
1    close  19757.53683
Training R-squared: 0.2591800642165357
Testing R-squared: 0.21635312319498323
```



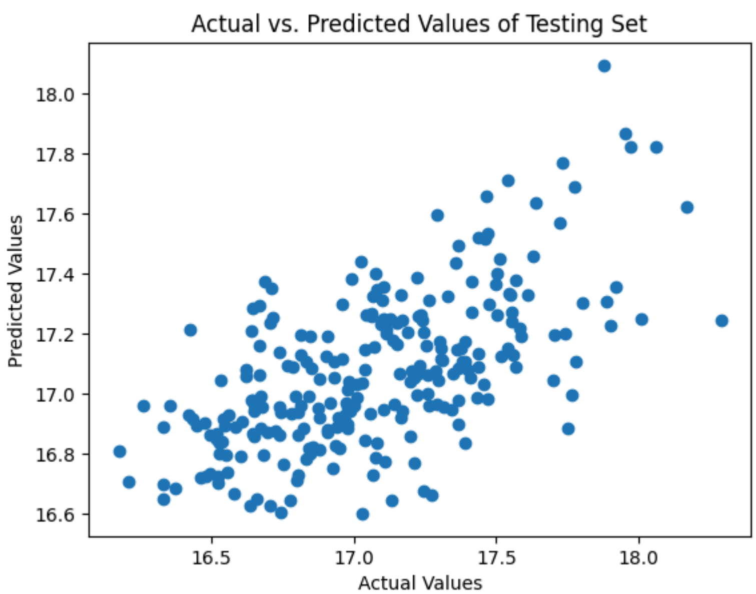

```
mult_lin_reg(logs_df, ["high", "low"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0     high  23779.399401
1      low  23779.399401
Training R-squared: 0.5336512682659659
Testing R-squared: 0.5181592393363503
```



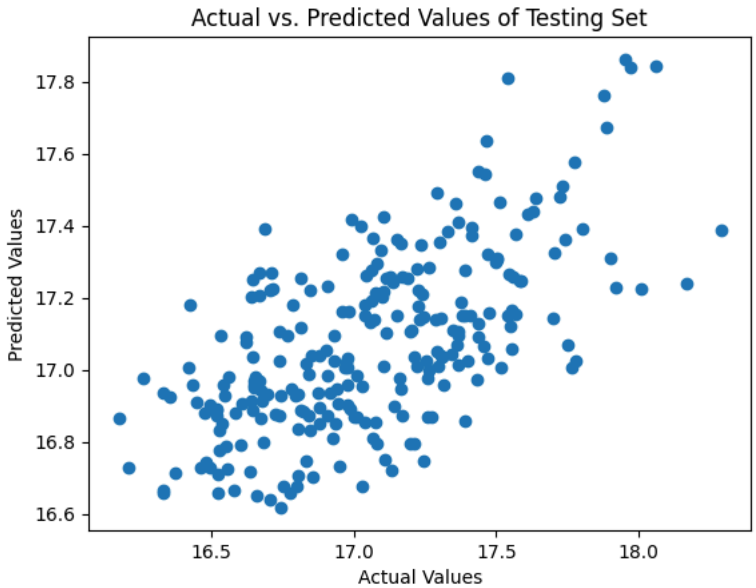

```
mult_lin_reg(logs_df, ["open", "low"], "volume")
```

```
Variance Inflation Factor:
  Variable           VIF
0     open  35030.128154
1      low  35030.128154
Training R-squared: 0.37468329081261353
Testing R-squared: 0.3938097885600982
```




```
mult_lin_reg(logs_df, ["close", "high"], "volume")
```

```
Variance Inflation Factor:
  Variable          VIF
0    close  39732.085347
1     high  39732.085347
Training R-squared: 0.34955603303675586
Testing R-squared: 0.4030852708397965
```



With our multiple regression machine learning method, we notice that our models have high correlation but extremely high variance inflation factors. This means we are seeing multicollinearity, which is expected since our variables are related to each other — open, close, high and low prices rise and fall in similar patterns.

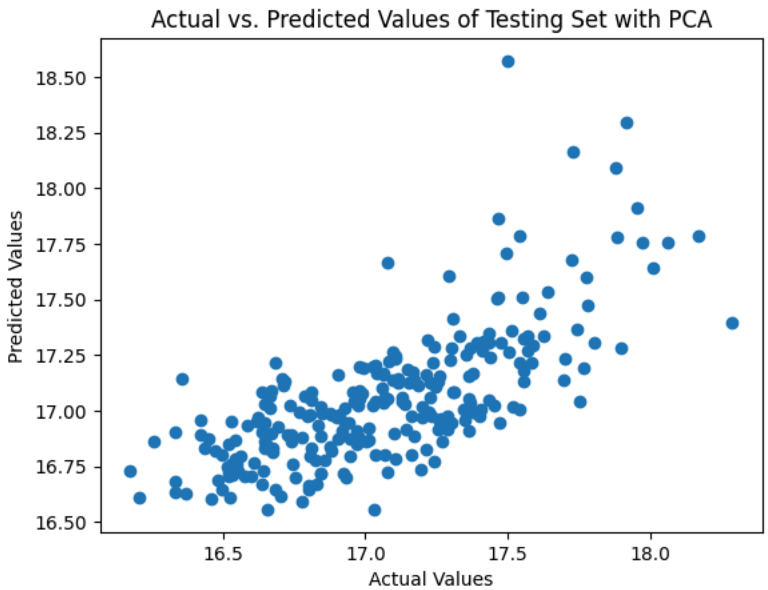
Therefore, we discovered that principal component analysis (PCA) could be incorporated to transform our original variables into a set of linearly uncorrelated variables. We created a function that adds PCA to our multiple regression in attempts to reduce multicollinearity. Hopefully, we can preserve as much information as possible with our modified variables.

```
def mult_lin_reg_with_pca(df, features, target, n_components=None):
  # Convert columns into numpy arrays
  X = df[features].values
  y = df[target].values

  # Split data into training and testing sets
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

  # Standardize the data
  scaler = StandardScaler()
  X_train_scaled = scaler.fit_transform(X_train)
  X_test_scaled = scaler.transform(X_test)

  # Apply PCA
  pca = PCA(n_components=n_components)
  X_train_pca = pca.fit_transform(X_train_scaled)
  X_test_pca = pca.transform(X_test_scaled)

  # Train linear regression model with PCA components
  lin_model_pca = LinearRegression().fit(X_train_pca, y_train)

  # Calculate VIF for each PCA component
  vif_data_pca = pd.DataFrame()
  vif_data_pca["Principal Component"] = range(1, pca.n_components_ + 1)
  vif_data_pca["VIF"] = [variance_inflation_factor(X_train_pca, i) for i in range(pca.n_components_)]

  print("Variance Inflation Factor (PCA Components):")
  print(vif_data_pca)

  # Generate predictions
  y_train_pred_pca = lin_model_pca.predict(X_train_pca)
  y_test_pred_pca = lin_model_pca.predict(X_test_pca)

  # Evaluate scores (coefficient of determination, R^2)
  train_r2_pca = lin_model_pca.score(X_train_pca, y_train)
  test_r2_pca = lin_model_pca.score(X_test_pca, y_test)
```

```
    print(f"Training R-squared with PCA: {train_r2_pca}")
    print(f"Testing R-squared with PCA: {test_r2_pca}")

    # Plot final multiple regression model with PCA
    plt.scatter(y_test, y_test_pred_pca)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title("Actual vs. Predicted Values of Testing Set with PCA")
    plt.show()
```

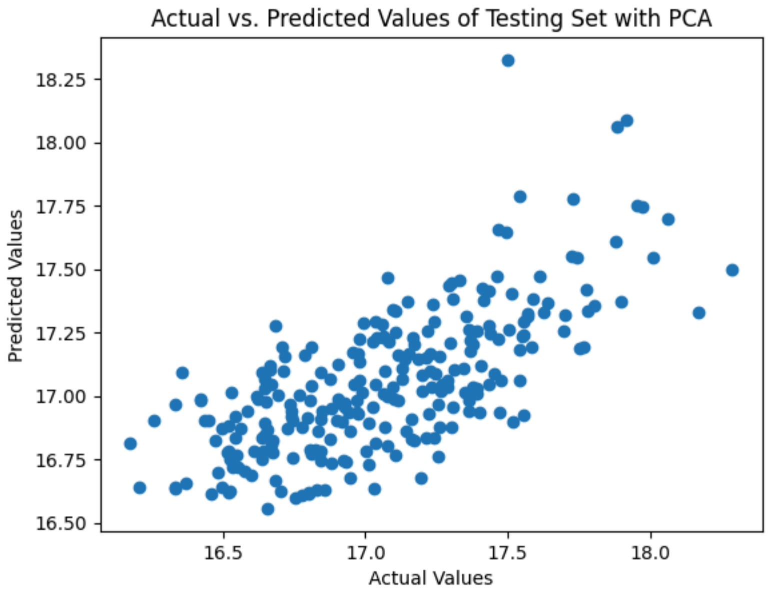
```
mult_lin_reg_with_pca(logs_df, ["open", "close", "high", "low"], "volume")
```

```
Variance Inflation Factor (PCA Components):
   Principal Component  VIF
0                    1  1.0
1                    2  1.0
2                    3  1.0
3                    4  1.0
Training R-squared with PCA: 0.5344901432537585
Testing R-squared with PCA: 0.5195177560977086
```



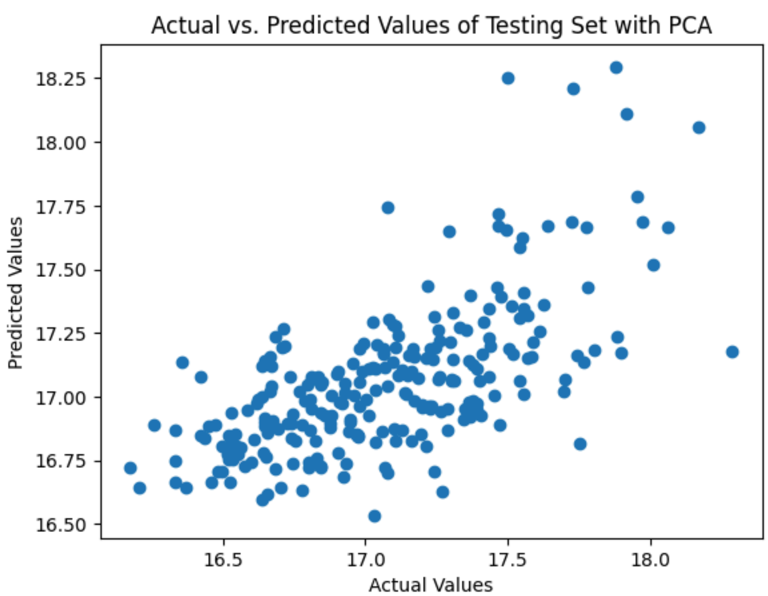

```
mult_lin_reg_with_pca(logs_df, ["open", "close", "high"], "volume")
```

```
Variance Inflation Factor (PCA Components):
   Principal Component  VIF
0                    1  1.0
1                    2  1.0
2                    3  1.0
Training R-squared with PCA: 0.4790791402045186
Testing R-squared with PCA: 0.4956724376531755
```




```
mult_lin_reg_with_pca(logs_df, ["open", "close", "low"], "volume")
```

```
Variance Inflation Factor (PCA Components):
   Principal Component  VIF
0                    1  1.0
1                    2  1.0
2                    3  1.0
Training R-squared with PCA: 0.4739090225166084
Testing R-squared with PCA: 0.42661136017088497
```




```
mult_lin_reg_with_pca(logs_df, ["open", "high", "low"], "volume")
```

```
mult_lin_reg_with_pca(logs_df, ["close", "high", "low"], "volume")
```

```
mult_lin_reg_with_pca(logs_df, ["open", "close"], "volume")
```

```
mult_lin_reg_with_pca(logs_df, ["high", "low"], "volume")
```

We also created regressor trees.

Regressor trees can be used to build predictive models that learn patterns and relationships that help predict future stock prices.

Regressor trees also provide an intuitive way to assess the importance of different features in predicting stock prices. The interpretations of the trees help us understand which variables have the most significant impact on the model's predictions.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# create a function for regressor trees
def decision_tree_reg(logs_df, features, target):
    # Convert columns into numpy arrays
    X = logs_df[features].values
    y = logs_df[target].values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train Decision Tree Regressor
    tree_reg = DecisionTreeRegressor(random_state=42)
    tree_reg.fit(X_train, y_train)

    # Generate predictions
    y_train_pred = tree_reg.predict(X_train)
    y_test_pred = tree_reg.predict(X_test)

    # Evaluate model
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("Train RMSE:", train_rmse)
    print("Test RMSE:", test_rmse)

    # Plot final model
    plt.scatter(y_test, y_test_pred)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title("Actual vs. Predicted Values of Testing Set")
    plt.show()
```

We realized we needed to cross-validate our regressor trees to better estimate our model's performance, ensure it can react well to unseen testing data and detect overfitting.

```
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error

def decision_tree_reg_with_r2(logs_df, features, target, max_depth=10, min_samples_leaf=50):
    # Convert columns into numpy arrays
    X = logs_df[features].values
    y = logs_df[target].values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train Decision Tree Regressor with limited depth and minimum samples per leaf
    tree_reg = DecisionTreeRegressor(min_samples_leaf=min_samples_leaf, random_state=42)
    tree_reg.fit(X_train, y_train)

    # Generate predictions
    y_train_pred = tree_reg.predict(X_train)
    y_test_pred = tree_reg.predict(X_test)

    # Evaluate model
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print("Train RMSE:", train_rmse)
    print("Test RMSE:", test_rmse)

    # Perform cross-validation with R-squared as the scoring metric
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    r2_scores = cross_val_score(tree_reg, X, y, cv=kfold, scoring='r2')

    print("Cross-Validation R-squared Scores:", r2_scores)
    print("Average R-squared:", np.mean(r2_scores))

    # Plot final model
    plt.scatter(y_test, y_test_pred)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title("Actual vs. Predicted Values of Testing Set")
    plt.show()

    # Visualize the Decision Tree
    plt.figure(figsize=(30, 8))
    plot_tree(tree_reg, filled=True, feature_names=features, rounded=True, fontsize=10)
    plt.show()


decision_tree_reg(logs_df, ["open", "close", "high", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["open", "close", "high"], "volume")
```

```
decision_tree_reg(logs_df, ["open", "close", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["open", "high", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["close", "high", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["open", "close"], "volume")
```

```
decision_tree_reg(logs_df, ["high", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["open", "low"], "volume")
```

```
decision_tree_reg(logs_df, ["close", "high"], "volume")
```

Visualizations

Line graph

We create line graphs showing the progression of stock prices. We generated a line plot comparing the the average values of "high" and "low" stock prices from 1991 to 2023, with emphasis on the speicifed anomaly dates which are easily identified and when the line graphs become their lowest point. The resulting line plot display a visual representation of how low the stock dropped during the specified anomaly times.

```
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

def plot_stocks(df, col1, col2, interval_val, start_date=None, end_date=None):
    df["timestamp"] = pd.to_datetime(df["timestamp"])

    if start_date is not None and end_date is not None:
        start_date = pd.to_datetime(start_date)
        end_date = pd.to_datetime(end_date)
        df = df[(df["timestamp"] >= start_date) & (df["timestamp"] <= end_date)]

    df = df.sort_values(by='timestamp')

    # Plot
    plt.figure(figsize=(15, 10))
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d"))
    plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=interval_val))

    plt.plot(df["timestamp"], df[col1], label=str(col1).capitalize())
    plt.plot(df["timestamp"], df[col2], label=str(col2).capitalize())

    plt.xlabel("Timestamp")
    plt.ylabel("Price (in USD)")
    plt.legend()
    plt.gcf().autofmt_xdate()
    plt.title(f"Difference in {str(col1).capitalize()} Stock Prices and {str(col2).capitalize()} Stock Prices over {start_date}
    plt.show()
```