goldpriceprediction

August 27, 2023

[1]:
```python
# Import packages
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import cross_val_score
from xgboost import XGBRegressor
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import mutual_info_regression
from sklearn.decomposition import PCA
```

[2]:
```python
from google.colab import files
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving FINAL_USO.csv to FINAL_USO.csv
```

[3]:
```python
df = pd.read_csv("FINAL_USO.csv")

y = df['Adj Close']

# We will start out by selecting features gold ETF features
gold_features = ['Open','High', 'Low', 'Volume']
X = df[gold_features]
X.head()
```

[3]:

	Open	High	Low	Volume
0	154.740005	154.949997	151.710007	21521900
1	154.309998	155.369995	153.899994	18124300
2	155.479996	155.860001	154.360001	12547200
3	156.820007	157.429993	156.580002	9136300
4	156.979996	157.529999	156.130005	11996100

[4]:
```
# There are no null values
df.isnull().values.any()
```

[4]: False

[5]:
```
# Define Model
gold_model = LinearRegression()

#Fit Model
gold_model.fit(X, y)

print("Making predicitons for the first 5 entries\n")
print(X.head())
print("\nThe predictions are:\n")
print(gold_model.predict(X.head()))
print("\nThe actual values are:\n")
print(y.head())
```

```
Making predicitons for the first 5 entries

         Open        High         Low    Volume
0  154.740005  154.949997  151.710007  21521900
1  154.309998  155.369995  153.899994  18124300
2  155.479996  155.860001  154.360001  12547200
3  156.820007  157.429993  156.580002   9136300
4  156.979996  157.529999  156.130005  11996100

The predictions are:

[152.55743325 154.81709905 154.92457233 157.14066214 156.77663033]

The actual values are:

0    152.330002
1    155.229996
2    154.869995
3    156.979996
4    157.160004
Name: Adj Close, dtype: float64
```

[6]:
```
predicted_adj_close = gold_model.predict(X.head())
print(mean_absolute_error(y.head(),predicted_adj_close))

predicted_adj_close = gold_model.predict(X)
print(mean_absolute_error(y, predicted_adj_close))
```

```
0.2477890674518619
0.21905793913855734
```

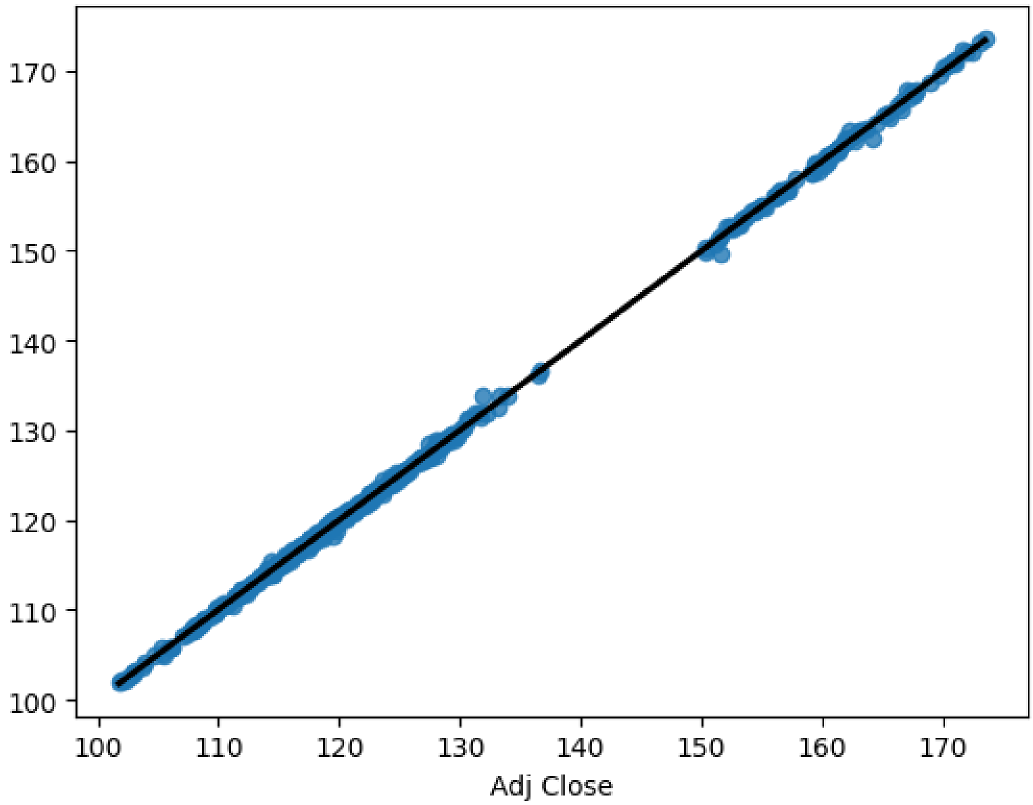
[7]:
```
# Partition data into training and validation groups
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
# Define a new model for training set
gold_model = LinearRegression()
# Fit model
gold_model.fit(train_X, train_y)

#get predicted prices on validation data
val_predictions = gold_model.predict(val_X)
print(mean_absolute_error(val_y,val_predictions))
```

```
0.22434694336395747
```

[8]:
```
gold_model = LinearRegression()

# Bundle preporcessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('gold_model', gold_model)])
# Preprocessing of training data, fit model
my_pipeline.fit(train_X, train_y)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(val_X)

# Evaluate the model
mae_score = mean_absolute_error(val_y, preds)
print('MAE:', mae_score)

# Display Model
sns.regplot(x=val_y, y=preds, line_kws={"color":"black"})
```

```
MAE: 0.22434694336395747
```

[8]:
```
<Axes: xlabel='Adj Close'>
```



[9]:
```
# Multiply by -1 since sklearn calculates *negative* MAE
scores = -1 * cross_val_score(my_pipeline, X, y,
                              cv=10,
                              scoring = 'neg_mean_absolute_error')
print("MAE scores:\n",scores,"\n")
print("Average MAE score (across all ten folds):")
print(scores.mean())

rmse = math.sqrt(mean_squared_error(val_y,preds))
print("\nRMSE is",rmse)

r2 = r2_score(val_y, preds)
print("\nr2 score is", r2)
```

```
MAE scores:
 [0.33869539 0.28749731 0.27608857 0.18376062 0.19862309 0.20854433
 0.23916281 0.16176519 0.17072235 0.14091063]

Average MAE score (across all ten folds):
0.22057702723377437
```

```
RMSE is 0.3257335889369903

r2 score is 0.9996725196712021
```

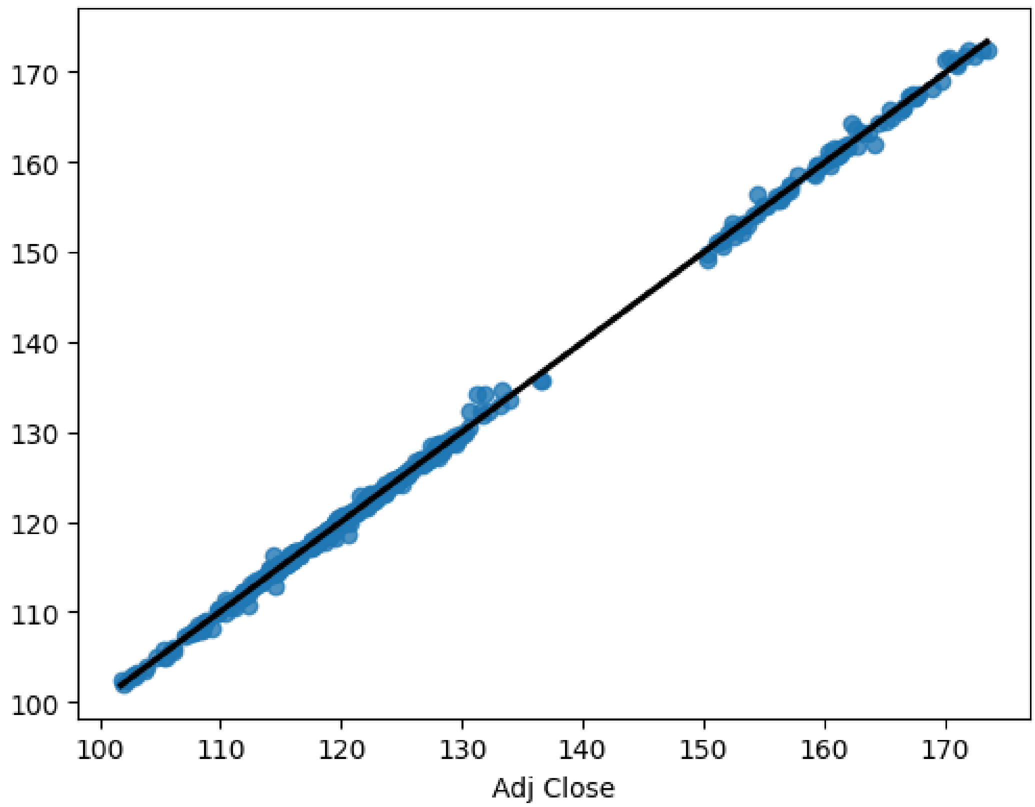
[10]:
```python
my_model = XGBRegressor()
my_model.fit(train_X, train_y)

# Make predictions using XGBoost model
predictions = my_model.predict(val_X)
print("Mean Absolute Error: ",mean_absolute_error(predictions, val_y))
```

```
Mean Absolute Error:  0.3337985221452574
```

[11]:
```python
my_model = XGBRegressor(n_estimators=1000,
                        learning_rate=0.03,
                        n_jobs=4)
my_model.fit(train_X, train_y,
            early_stopping_rounds=5,
            eval_set=[(val_X, val_y)],
            verbose=False)

predictions = my_model.predict(val_X)
print("Mean Absolute Error",
      mean_absolute_error(predictions,val_y))

rmse = math.sqrt(mean_squared_error(val_y,predictions))
print("\nRMSE is", rmse)

r2 = r2_score(val_y,predictions)
print("\nr2 score is", r2)

sns.regplot(x=val_y, y=predictions, line_kws={"color": "black"})
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:835: UserWarning:
`early_stopping_rounds` in `fit` method is deprecated for better compatibility
with scikit-learn, use `early_stopping_rounds` in constructor or`set_params`
instead.
  warnings.warn(

Mean Absolute Error 0.3232501925031789

RMSE is 0.4868649270362122

r2 score is 0.9992683942525465
```

[11]:
```
<Axes: xlabel='Adj Close'>
```



[12]:
```
# Refresh on what all of the features look like
# There are 79 predictor columns. I am not including Adj Close and Close of the␣
↪81 total.

plt.style.use("seaborn-whitegrid")

df.head()
```

```
<ipython-input-12-6c5ddbdd1776>:4: MatplotlibDeprecationWarning: The seaborn
styles shipped by Matplotlib are deprecated since 3.6, as they no longer
correspond to the styles shipped by seaborn. However, they will remain available
as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```

[12]:

	Date	Open	High	Low	Close	Adj Close	\
0	2011-12-15	154.740005	154.949997	151.710007	152.330002	152.330002	
1	2011-12-16	154.309998	155.369995	153.899994	155.229996	155.229996	
2	2011-12-19	155.479996	155.860001	154.360001	154.869995	154.869995	
3	2011-12-20	156.820007	157.429993	156.580002	156.979996	156.979996	
4	2011-12-21	156.979996	157.529999	156.130005	157.160004	157.160004	

```
     Volume     SP_open     SP_high      SP_low  …    GDX_Low  GDX_Close  \
0  21521900  123.029999  123.199997  121.989998  …  51.570000  51.680000
1  18124300  122.230003  122.949997  121.300003  …  52.040001  52.680000
2  12547200  122.059998  122.320000  120.029999  …  51.029999  51.169998
3   9136300  122.180000  124.139999  120.370003  …  52.369999  52.990002
4  11996100  123.930000  124.360001  122.750000  …  52.419998  52.959999

   GDX_Adj Close  GDX_Volume   USO_Open   USO_High    USO_Low  USO_Close  \
0      48.973877    20605600  36.900002  36.939999  36.049999  36.130001
1      49.921513    16285400  36.180000  36.500000  35.730000  36.270000
2      48.490578    15120200  36.389999  36.450001  35.930000  36.200001
3      50.215282    11644900  37.299999  37.610001  37.220001  37.560001
4      50.186852     8724300  37.669998  38.240002  37.520000  38.110001

   USO_Adj Close  USO_Volume
0      36.130001    12616700
1      36.270000    12578800
2      36.200001     7418200
3      37.560001    10041600
4      38.110001    10728000

[5 rows x 81 columns]
```

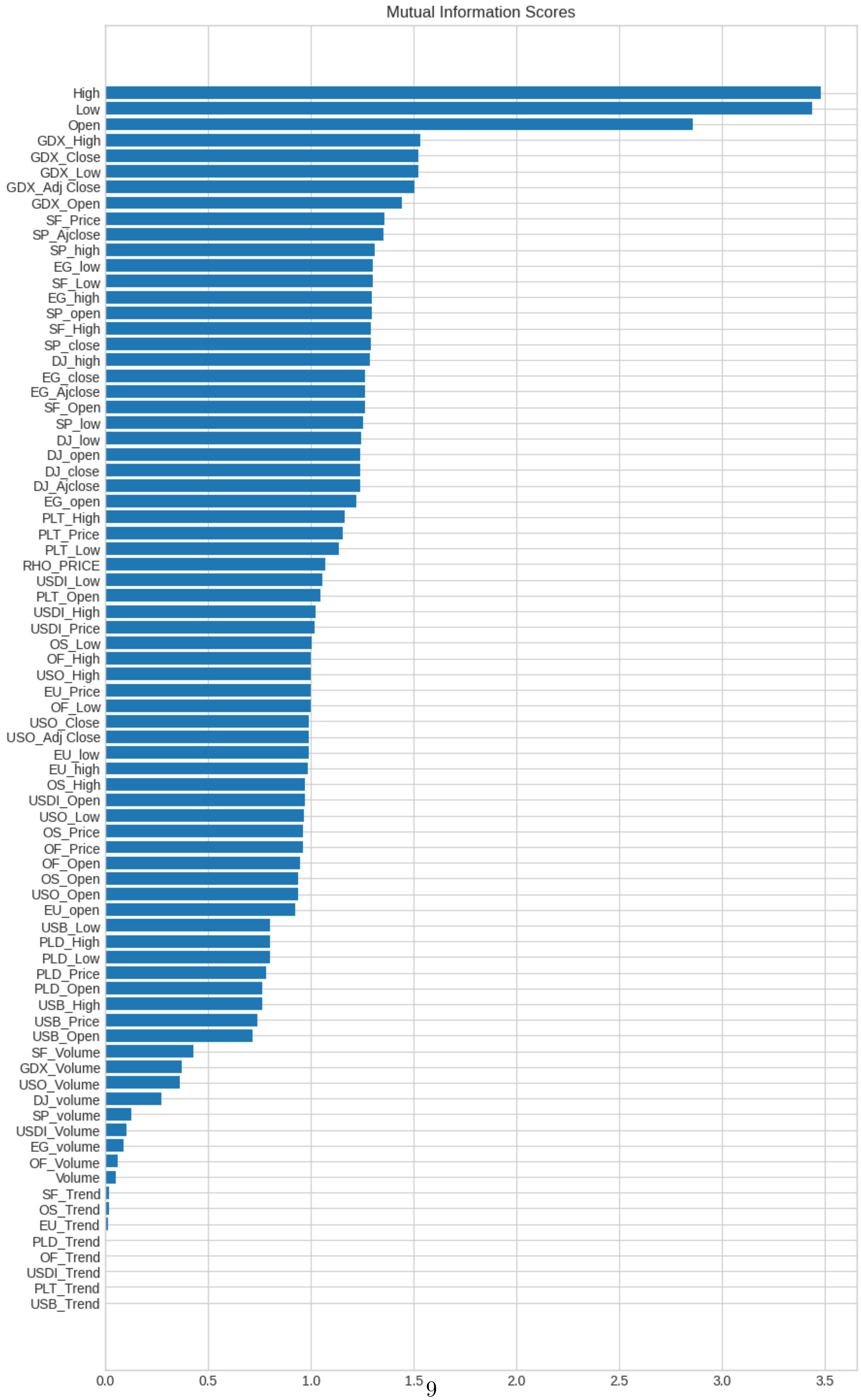
[13]:
```python
# Create new ds with all predictor features. Take Adj Close as Y
# Remove Close because it is too close to Adj Close
X = df.copy()
y = X.pop('Adj Close')
date = X.pop('Date')
X.pop('Close')
```

[13]:
```
0       152.330002
1       155.229996
2       154.869995
3       156.979996
4       157.160004
           …
1713    120.019997
1714    119.660004
1715    120.570000
1716    121.059998
1717    121.250000
Name: Close, Length: 1718, dtype: float64
```

[14]:
```python
# Create mutual info scores

def make_mi_scores (X, y):
    mi_scores = mutual_info_regression(X, y)
```

```
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

mi_scores = make_mi_scores(X, y)
```

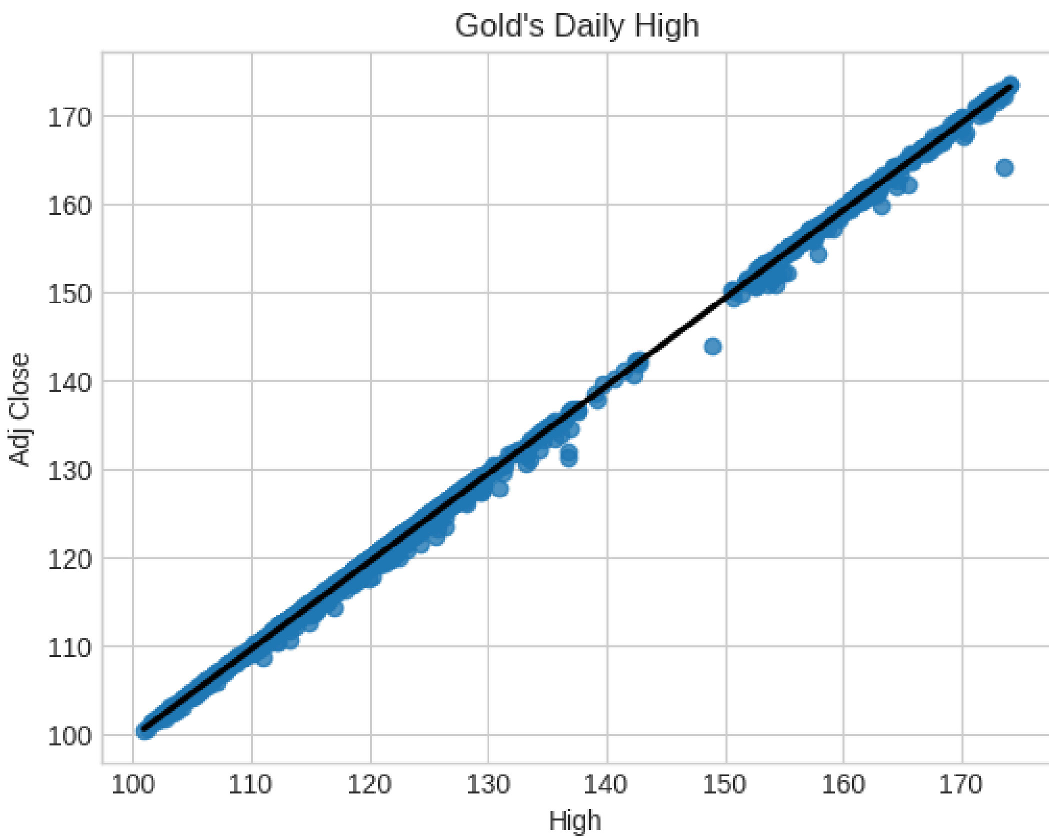
[15]:
```
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")

plt.figure(dpi=100, figsize=(10,18))
plot_mi_scores(mi_scores)
```



Mutual Information Scores
High
Low
Open
GDX_High
GDX_Close
GDX_Low
GDX_Adj Close
GDX_Open
SF_Price
SP_Ajclose
SP_high
EG_low
SF_Low
EG_high
SP_open
SF_High
SP_close
DJ_high
EG_close
EG_Ajclose
SF_Open
SP_low
DJ_low
DJ_open
DJ_close
DJ_Ajclose
EG_open
PLT_High
PLT_Price
PLT_Low
RHO_PRICE
USDI_Low
PLT_Open
USDI_High
USDI_Price
OS_Low
OF_High
USO_High
EU_Price
OF_Low
USO_Close
USO_Adj Close
EU_low
EU_high
OS_High
USDI_Open
USO_Low
OS_Price
OF_Price
OF_Open
OS_Open
USO_Open
EU_open
USB_Low
PLD_High
PLD_Low
PLD_Price
PLD_Open
USB_High
USB_Price
USB_Open
SF_Volume
GDX_Volume
USO_Volume
DJ_volume
SP_volume
USDI_Volume
EG_volume
OF_Volume
Volume
SF_Trend
OS_Trend
EU_Trend
PLD_Trend
OF_Trend
USDI_Trend
PLT_Trend
USB_Trend
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

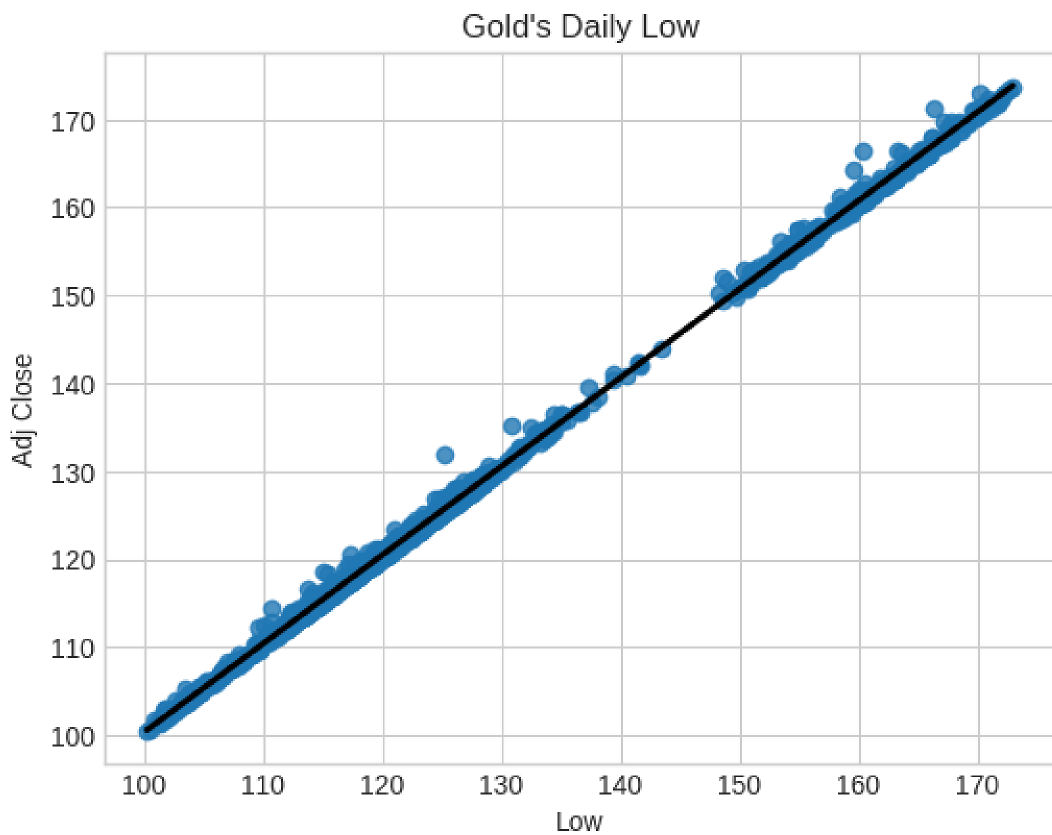
[16]:

```
daily_high = sns.regplot(x="High", y="Adj Close", data=df, line_kws={"color":
↪"black"}).set(title="Gold's Daily High")
```

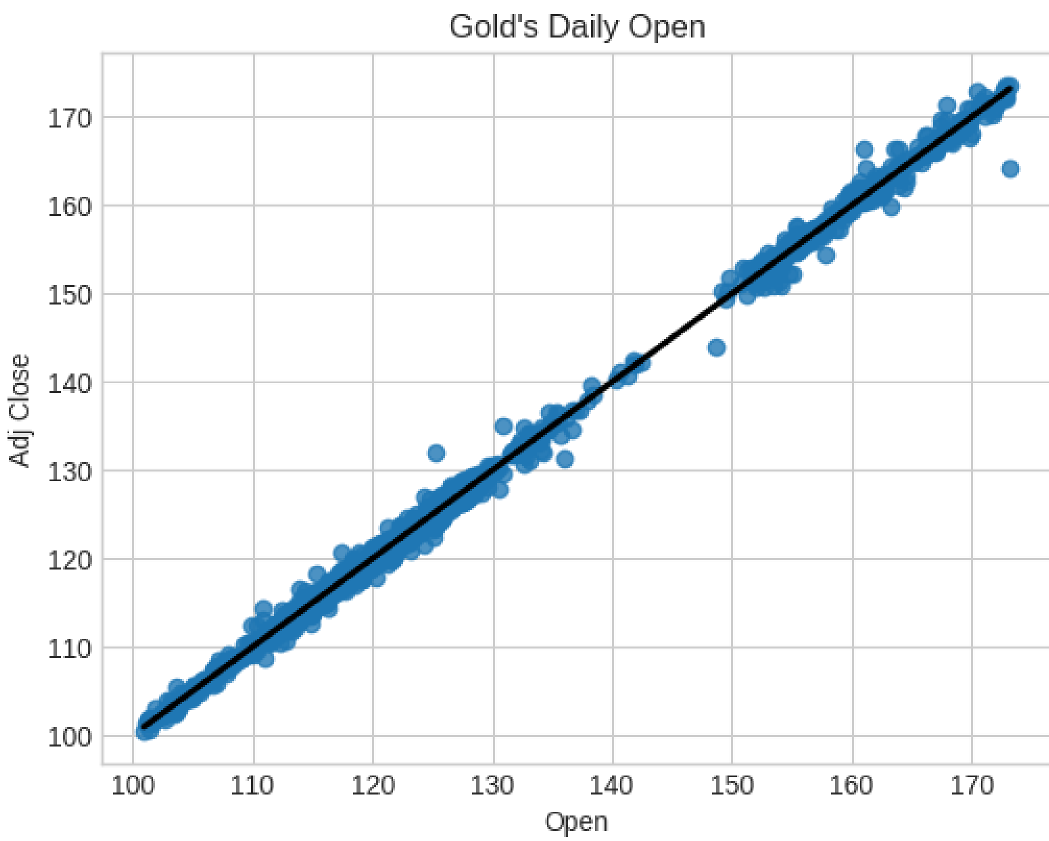


[17]:

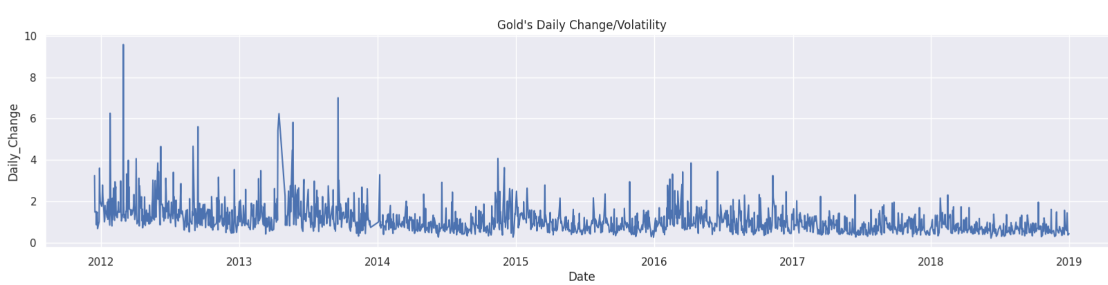
```
daily_low = sns.regplot(x="Low", y="Adj Close", data=df, line_kws={"color":
↪"black"}).set(title="Gold's Daily Low")
```



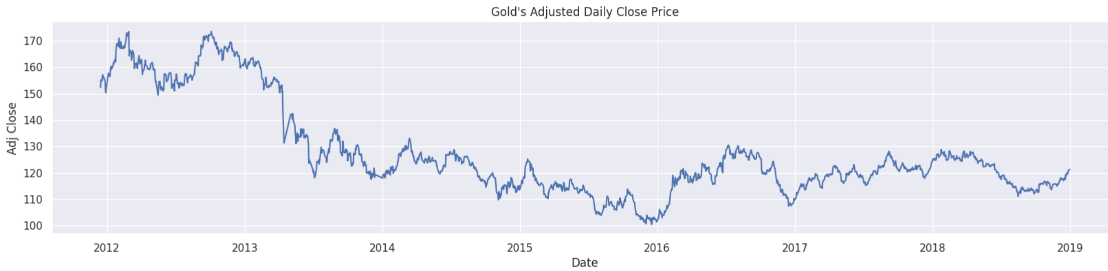
[18]:
```
daily_close = sns.regplot(x="Open", y="Adj Close", data=df, line_kws={"color":
    "black"}).set(title="Gold's Daily Open")
```



[19]:
```
df["Daily_Change"] = abs(X.High - X.Low)

# Convert Date from string to datetime to give us yearly ticks on the X-axis
df['Date'] = pd.to_datetime(df['Date'], format = '%Y-%m-%d')

# Plot volatility
sns.set(rc={"figure.figsize":(20, 4)})
daily_change = sns.lineplot(x="Date", y="Daily_Change", data=df).
 ↪set(title="Gold's Daily Change/Volatility")
```



[20]:
```
# Adjusted Close with Time Series
sns.set(rc={"figure.figsize":(20, 4)})
daily_change = sns.lineplot(x="Date", y="Adj Close", data=df).set(title="Gold's␣
↪Adjusted Daily Close Price")
```



[21]:
```
features = ["High", "Low", "Open", "GDX_High", "GDX_Low", "GDX_Close"]

X = df.copy()
y = X.pop('Adj Close')
date = X.pop('Date')
X.pop('Close')
X = X.loc[:, features]

# Standardize the new df. PCA is sensitive to scale.
X_scaled = (X - X.mean(axis=0)) / X.std(axis=0)
```

[22]:
```
# Create principal componenets
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Convert to dataframe
component_names = [f"PC{i+1}" for i in range (X_pca.shape[1])]
X_pca = pd.DataFrame(X_pca, columns=component_names)

X_pca.head()
```

[22]:

	PC1	PC2	PC3	PC4	PC5	PC6
0	4.786447	1.084283	0.062709	0.089771	0.020374	-0.008956
1	4.895857	1.091385	-0.013283	-0.007822	-0.004370	-0.009334
2	4.823785	0.920197	0.005722	0.050612	-0.030129	-0.008412
3	5.092355	0.949527	-0.042882	-0.010260	0.000476	0.002431
4	5.095494	0.961803	-0.020048	0.008791	0.007451	0.000329

[23]:
```
# Wrap the PCA loadings up in a dataframe
loadings = pd.DataFrame(
    pca.components_.T,        # Transpose the matrix of loadings
```

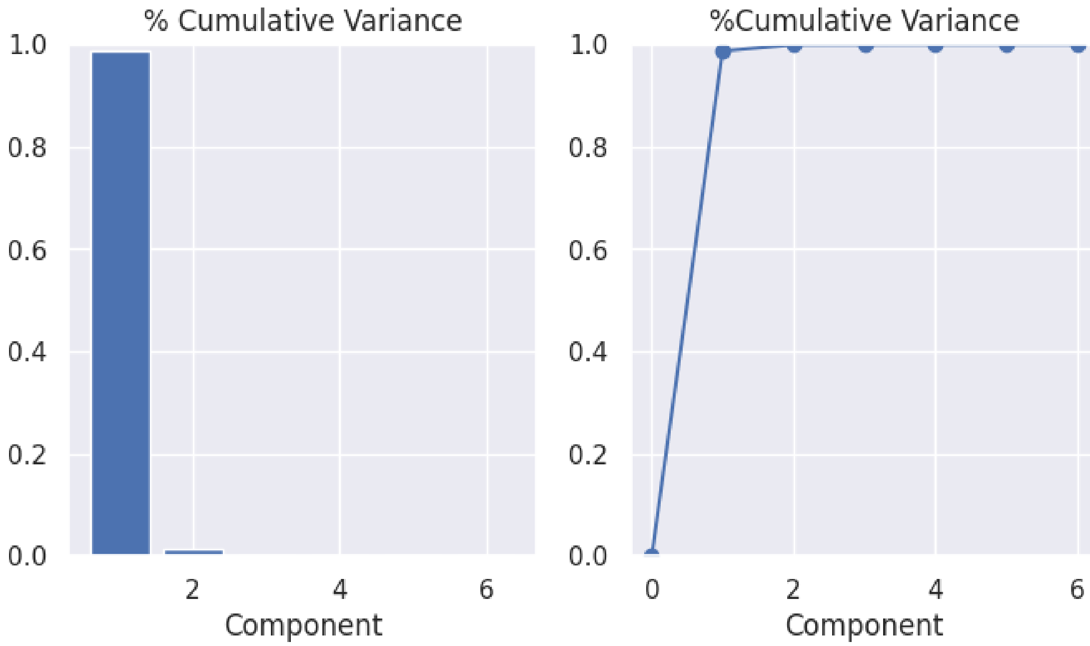
```
    columns=component_names, # to turn columns into principal components
    index = X.columns,       # and the rows are original features, so we can␣
 ↪identify them
)
loadings
```

[23]:

	PC1	PC2	PC3	PC4	PC5	PC6
High	0.408326	-0.401039	0.529359	-0.274509	0.192727	-0.528883
Low	0.408168	-0.413142	-0.558814	-0.298090	-0.510041	0.037945
Open	0.408236	-0.410488	0.040841	0.563978	0.321422	0.491724
GDX_High	0.408251	0.408309	0.433426	0.271772	-0.632694	0.067782
GDX_Low	0.408318	0.402371	-0.466389	0.320687	0.285792	-0.518978
GDX_Close	0.408190	0.413960	0.021448	-0.583966	0.342671	0.450686

[24]:

```
def plot_variance(pca, width=8, dpi=100):
    # Create figure
    fig, axs = plt.subplots(1,2)
    n = pca.n_components_
    grid = np.arange(1, n + 1)

    # Explained variance
    evr = pca.explained_variance_ratio_
    axs[0].bar(grid, evr)
    axs[0].set(
        xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)
    )
    # Cumulative Variance
    cv = np.cumsum(evr)
    axs[1].plot(np.r_[0, grid], np.r_[0,cv], "o-")
    axs[1].set(
        xlabel="Component", title="%Cumulative Variance", ylim=(0.0,1.0)
    )
    # Set up figure
    fig.set(figwidth=8, dpi=100)
    return axs

# Look at the explained variance from PCA
plot_variance(pca);
```



[25]:
```
# View MI Scores for the principal components
mi_scores = make_mi_scores(X_pca, y)
mi_scores
```

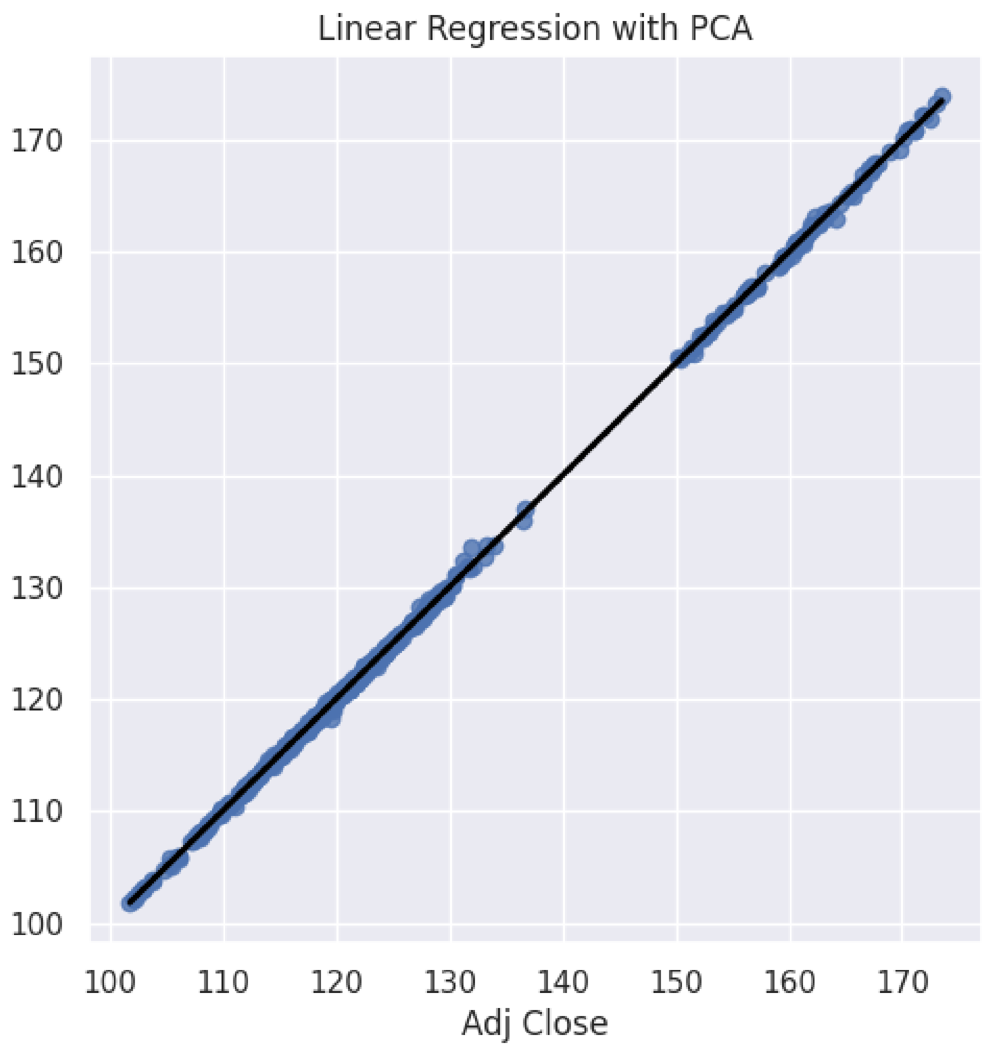
[25]:
```
PC1    2.185745
PC2    0.509278
PC3    0.113900
PC5    0.095533
PC4    0.034827
PC6    0.010998
Name: MI Scores, dtype: float64
```

[26]:
```
# Partition the PCA dataframe into training and validation groups
train_X, val_X, train_y, val_y = train_test_split(X_pca, y, random_state = 0)

gold_model = LinearRegression()

# Bundle preporcessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('gold_model', gold_model)])
# Preprocessing of training data, fit model
my_pipeline.fit(train_X, train_y)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(val_X)
```

```
# Evaluate the model
mae_score = mean_absolute_error(val_y, preds)
print('MAE:', mae_score)

# Display Model
sns.set(rc={"figure.figsize":(6,6)})
sns.regplot(x=val_y, y=preds, line_kws={"color":"black"}).set(title="Linear␣
↪Regression with PCA")
```

```
MAE: 0.2011137068925935
```

[26]: `[Text(0.5, 1.0, 'Linear Regression with PCA')]`



[27]:
```python
# Multiply by -1 since sklearn calculates *negative* MAE
scores = -1 * cross_val_score(my_pipeline, X_pca, y,
                              cv=10,
                              scoring = 'neg_mean_absolute_error')
print("MAE scores:\n",scores,"\n")
print("Average MAE score (across all ten folds):")
print(scores.mean())
rmse = math.sqrt(mean_squared_error(val_y,preds))
print("\nRMSE is", rmse)
r2 = r2_score(val_y,preds)
print("\nr2 score is", r2)
```

```
MAE scores:
 [0.29200777 0.27589726 0.24365488 0.15945436 0.17239864 0.17891691
 0.19767476 0.13222747 0.15118348 0.12333965]

Average MAE score (across all ten folds):
0.19267551793350662

RMSE is 0.27542712552287113

r2 score is 0.9997658611136764
```

[28]:
```python
results = [['Linear Regression', 0.221, 0.326, 0.999672],
           ['Gradient Boosting (XGBoost)', 0.325, 0.490, 0.999259],
           ['Linear Regression with PCA', 0.193, 0.275, 0.999766]]
results_df = pd.DataFrame(results, columns = ['Model Type', 'MAE', 'RMSE',␣
 ↪'r2'])
results_df
```

[28]:

	Model Type	MAE	RMSE	r2
0	Linear Regression	0.221	0.326	0.999672
1	Gradient Boosting (XGBoost)	0.325	0.490	0.999259
2	Linear Regression with PCA	0.193	0.275	0.999766

[ ]:
```python

```