Faculty of computer science and artificial intelligence, Helwan University
Department of Data Science

HELWAN UNIVERSITY

Gold Prices Documentation

Supervised by: Prof. Sayed AbdelGaber

Prepared by

Dana Aboelyazid Elshrbiny
Yomna Tarek Gamal

December 27, 2023

Overview

The Egypt Gold Prices project is for people who want to know how much gold costs in Egypt every day. It looks at the prices in Egyptian money for each gram of gold. The aim is to help people, like investors and analysts, get a clear picture of what's happening with gold in Egypt. The project collects and keeps a record of daily gold prices, giving useful information about the Egyptian gold market. It's like a tool to understand how gold prices in Egypt relate to the global market.

Tools and technologies

- Programming languages: Python
- Data analysis tools: Pandas, NumPy.
- Data visualization libraries: Matplotlib, Seaborn.
- Data visualization platforms: Power bi.
- Code Platforms: Jupyter notebooks or Google Colab.
- NoSQL Database: Mongo DB.

Starting the Project

1. Start by getting the important tools and libraries ready. This is called the **import** part of the code.

Imports Section.

In [145…

```python
"""Standard library imports."""

import pandas as pd  # Pandas for data manipulation and analysis.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler  # MinMaxScaler from scikit-learn for feature scaling
from sklearn.model_selection import train_test_split  # train_test_split from scikit-learn for splitting the dataset
from sklearn.linear_model import LinearRegression  # LinearRegression from scikit-learn for linear regression modeling
from sklearn.linear_model import Ridge  # Ridge regression is a linear regression model with L2 regularization
from sklearn.model_selection import cross_val_score  # cross_val_score for cross-validation
from sklearn.feature_selection import RFE  # Recursive Feature Elimination (RFE) for feature selection
from sklearn.tree import DecisionTreeRegressor  #  DecisionTreeRegressor from scikit-learn for Decision Tree Regressor m
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor  #  RandomForestRegressor from scikit-learn for Random Forest Regress
from sklearn.ensemble import GradientBoostingRegressor #  GradientBoostingRegressor from scikit-learn for Gradient Boost
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Lasso
from keras.models import load_model
import h5py
```

2. Dataset Acquisition from kaggle

- The dataset containing daily gold prices in Egyptian pounds per gram. This dataset is available for download from Kaggle.
- Can we take a look to read csv and show this dataset.

In [58]:

```python
# Read the CSV file into a Pandas DataFrame.
# The read_csv function reads the data from the CSV file and stores it in a tabular format.
# You can specify various parameters like delimiter, header, and encoding based on the file's characteristics.

data = pd.read_csv('/content/data/data.csv')
```

In [59]:

```python
# Display the first few rows of the 'data' DataFrame.
# The head() method is used to show a preview of the data, which is useful for initial exploration.
# By default, it displays the first 5 rows, but you can specify the number of rows by passing an argument.

data.head()
```

Out[59]:

	Date	24K - Local Price/Sell	24K - Local Price/Buy	22K - Local Price/Sell	22K - Local Price/Buy	21K - Local Price/Sell	21K - Local Price/Buy	18K - Local Price/Sell	18K - Local Price/Buy	14K - Local Price/Sell	14K - Local Price/Buy	12K - Local Price/Sell	Pri
0	2022-11-09	1394.0	1401.0	1278.0	1284.0	1220.0	1226.0	1046.0	1051.0	813.0	817.0	697.0	
1	2022-11-10	1398.0	1402.0	1281.0	1285.0	1223.0	1227.0	1048.0	1052.0	815.0	818.0	699.0	
2	2022-11-11	1431.0	1435.0	1312.0	1316.0	1252.0	1256.0	1073.0	1077.0	835.0	837.0	715.0	
3	2022-11-12	1446.0	1457.0	1325.0	1336.0	1265.0	1275.0	1084.0	1093.0	843.0	850.0	723.0	
4	2022-11-13	1429.0	1440.0	1310.0	1320.0	1250.0	1260.0	1071.0	1080.0	833.0	840.0	714.0	

3. Check for Missing.

- **Ignore the tuples:**

  This approach is suitable when the dataset is quite large and multiple values are missing within a tuple.

- **Fill the Missing values:**

  Fill the missing values manually, by attribute mean or the most probable value.

- In this dataset, we calculate the number of missing values in each column of the 'data' Data Frame.

Check For Missing.

In [66]:
```python
# Calculate the number of missing values in each column of the 'data' DataFrame.

missing = data.isnull().sum()
missing
```

Out[66]:
```
Date                     0
24K - Local Price/Sell   0
24K - Local Price/Buy    0
22K - Local Price/Sell   0
22K - Local Price/Buy    0
21K - Local Price/Sell   0
21K - Local Price/Buy    0
18K - Local Price/Sell   0
18K - Local Price/Buy    0
14K - Local Price/Sell   0
14K - Local Price/Buy    0
12K - Local Price/Sell   0
12K - Local Price/Buy    0
24K - Global Price       0
22K - Global Price       0
21K - Global Price       0
18K - Global Price       0
14K - Global Price       0
12K - Global Price       0
9K - Global Price        0
dtype: int64
```

Before initiating the preprocessing process, we extract features and labels from the data. In this context, features represent buy prices, while labels represent sell prices.

- ## Labels

```
In [70]:
# Extracting buy prices as labels
# iloc[:, [1, 3, 5, 7, 9, 11]] selects all rows (:) and columns at indices 1, 3, 5, 7, 9, 11

labels = data.iloc[:, [1, 3, 5, 7, 9, 11]]
labels.head()
```

Out[70]:

	24K - Local Price/Buy	22K - Local Price/Buy	21K - Local Price/Buy	18K - Local Price/Buy	14K - Local Price/Buy	12K - Local Price/Buy
0	1401.0	1284.0	1226.0	1051.0	817.0	701.0
1	1402.0	1285.0	1227.0	1052.0	818.0	701.0
2	1435.0	1316.0	1256.0	1077.0	837.0	718.0
3	1457.0	1336.0	1275.0	1093.0	850.0	729.0
4	1440.0	1320.0	1260.0	1080.0	840.0	720.0

- ## Features

```
In [71]:
# Extracting sell prices as features
# iloc[:,[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]] selects all rows (:) and columns at indices 0, 2, 4, 6, 8, 10, 12,

features = data.iloc[:, [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]]
features.head()
```

Out[71]:

	24K - Local Price/Sell	22K - Local Price/Sell	21K - Local Price/Sell	18K - Local Price/Sell	14K - Local Price/Sell	12K - Local Price/Sell	24K - Global Price	21K - Global Price	14K - Global Price	9K - Global Price	Date_2022-11-10	Date_2022-11-12
0	1394.0	1278.0	1220.0	1046.0	813.0	697.0	1339.0	1171.0	780.84	501.97	0	0
1	1398.0	1281.0	1223.0	1048.0	815.0	699.0	1378.0	1206.0	803.74	516.69	1	0
2	1431.0	1312.0	1252.0	1073.0	835.0	715.0	1387.0	1213.0	808.95	520.04	0	0
3	1446.0	1325.0	1265.0	1084.0	843.0	723.0	1387.0	1214.0	809.33	520.28	0	1
4	1429.0	1310.0	1250.0	1071.0	833.0	714.0	1388.0	1215.0	809.92	520.66	0	0

Now, we can begin preprocessing our data by transforming features and labels.

4. Data Transformation

   Converting the data into a suitable format for analysis and the Common techniques include:

   - **Normalization**

     I. Used to handle data with different units and scales.

II. Scale the data to a common range

III. Common normalization techniques include min-max normalization, z-score normalization, and decimal scaling.

- **Standardization**

  Transform the data to have zero mean and unit variance.

- **Discretization.**

I. Convert continuous data into discrete categories.

II. Achieved through techniques such as equal width binning, equal frequency binning, and clustering

- **Concept Hierarchy Generation**

  Attributes are converted from lower level to higher level in hierarchy.

We transform the features and labels using Min-Max scaling to achieve a specified range, with the default being [0, 1].

```
In [72]:  # Creating a MinMaxScaler instance
          scale = MinMaxScaler()

          # Transforming the features using Min-Max scaling
          # fit_transform() scales the features to a specified range (default is [0, 1])

          features = scale.fit_transform(features)
          features

Out[72]:  array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                  0.        ],
                 [0.00236407, 0.00193424, 0.00202703, ..., 0.05651757, 1.        ,
                  0.        ],
                 [0.02186761, 0.02192134, 0.02162162, ..., 0.06937992, 0.        ,
                  0.        ],
                 ...,
                 [0.87825059, 0.87814313, 0.87837838, ..., 0.92378576, 0.        ,
                  0.        ],
                 [0.87470449, 0.87491941, 0.875     , ..., 0.91556921, 0.        ,
                  0.        ],
                 [0.90543735, 0.90522244, 0.90540541, ..., 0.93737762, 0.        ,
                  0.        ]])
```

```
In [73]: scale2 = MinMaxScaler()

         # Transforming the labels using Min-Max scaling
         # fit_transform() scales the labels to a specified range (default is [0, 1])

         labels = scale2.fit_transform(labels)
         labels

Out[73]: array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                 0.00000000e+00, 0.00000000e+00],
                [5.74052813e-04, 6.26174076e-04, 6.56167979e-04, 7.65696784e-04,
                 9.84251969e-04, 0.00000000e+00],
                [1.95177956e-02, 2.00375704e-02, 1.96850394e-02, 1.99081164e-02,
                 1.96850394e-02, 1.95402299e-02],
                ...,
                [8.75430540e-01, 8.75391359e-01, 8.75328084e-01, 8.75191424e-01,
                 8.75984252e-01, 8.75862069e-01],
                [8.58783008e-01, 8.59110833e-01, 8.58923885e-01, 8.59111792e-01,
                 8.59251969e-01, 8.59770115e-01],
                [8.88633754e-01, 8.88541014e-01, 8.88451444e-01, 8.88208270e-01,
                 8.88779528e-01, 8.88505747e-01]])
```

Now, we split the dataset into training and testing sets, where features and labels are divided into **x_train**, **x_test**, **y_train**, and **y_test**. Setting **test_size=0.2** indicates that 20% of the data will be used for testing, and 80% for training.

```
In [74]: # Splitting the dataset into training and testing sets
         # features and labels are split into x_train, x_test, y_train, and y_test
         # test_size=0.2 indicates that 20% of the data will be used for testing, and 80% for training
         x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.2)

In [75]: print(len(features))

349

In [76]: print(len(x_train))

279

In [77]: print(len(x_test))

70
```

As seen in this picture, we print the lengths of the **features**, **x_train**, and **x_test** to display the sizes or lengths of these specific datasets. This is done for observation and validation purposes, allowing us to verify the sizes of the training and testing sets in our dataset.

Now, we'll use different methods to predict our models and check how well they perform. There are various methods like linear regression, decision trees, and more. In this case, we're using Lasso Regression to figure out how close our predictions are to the actual values.

We chose Lasso Regression because it's good at picking important features and not getting confused by too many details. This is helpful when working with datasets that have a lot of features because it prevents the model from getting too specific and makes it easier to understand.

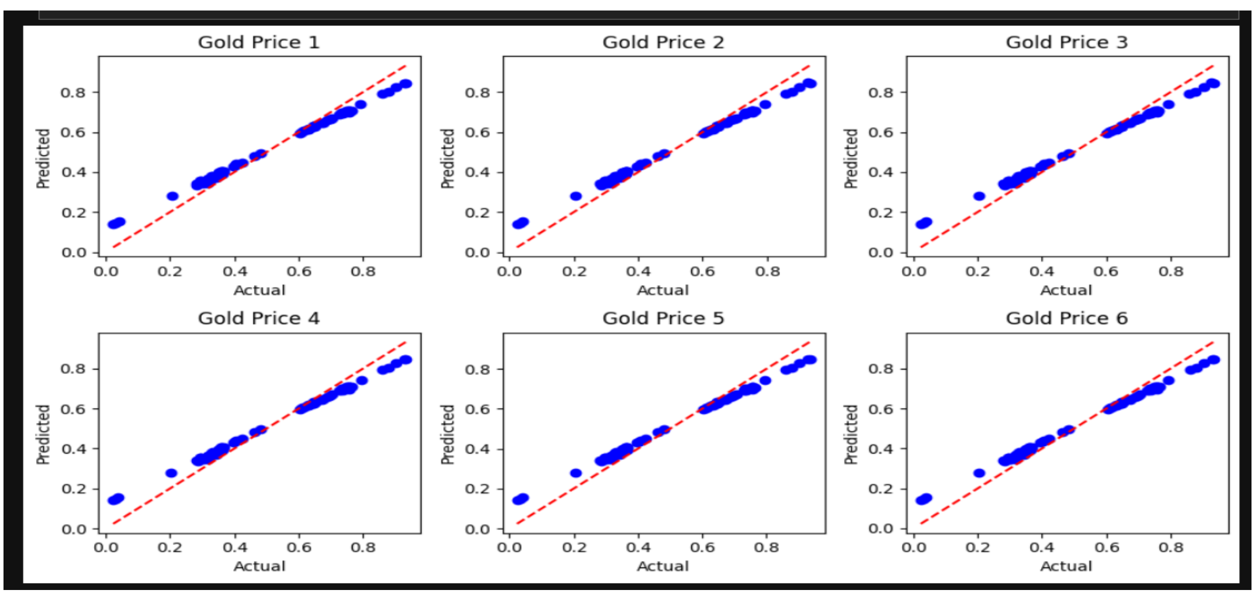
Lasso Regression

In [132…

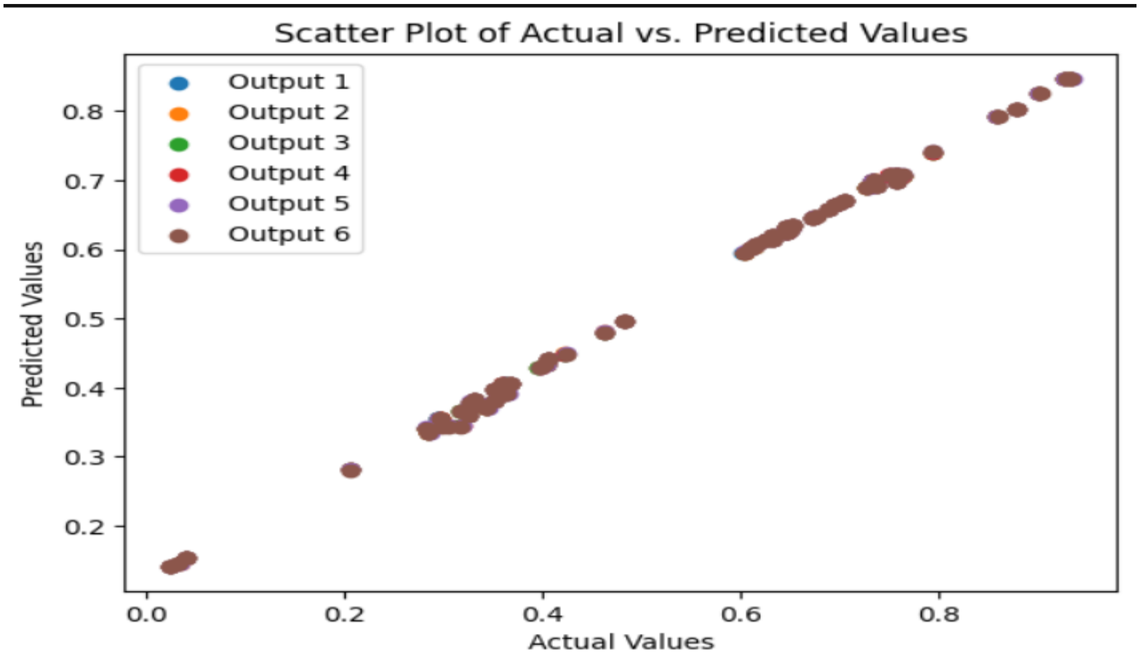
```python
# Create Lasso regression model with alpha (regularization strength)
lasso_model = Lasso(alpha=0.01)

# Train the Lasso model
lasso_model.fit(X_train, y_train)

# Make predictions on the test set
predictions_lasso = lasso_model.predict(X_test)

# Calculate R-squared and Mean Absolute Error
r2_lasso = r2_score(y_test, predictions_lasso)
mae_lasso = mean_absolute_error(y_test, predictions_lasso)

print("Lasso Regression - R-squared:", r2_lasso)
print("Lasso Regression - MAE:", mae_lasso)
```

```
Lasso Regression - R-squared: 0.9533143615951639
Lasso Regression - MAE: 0.04139741931027106
```

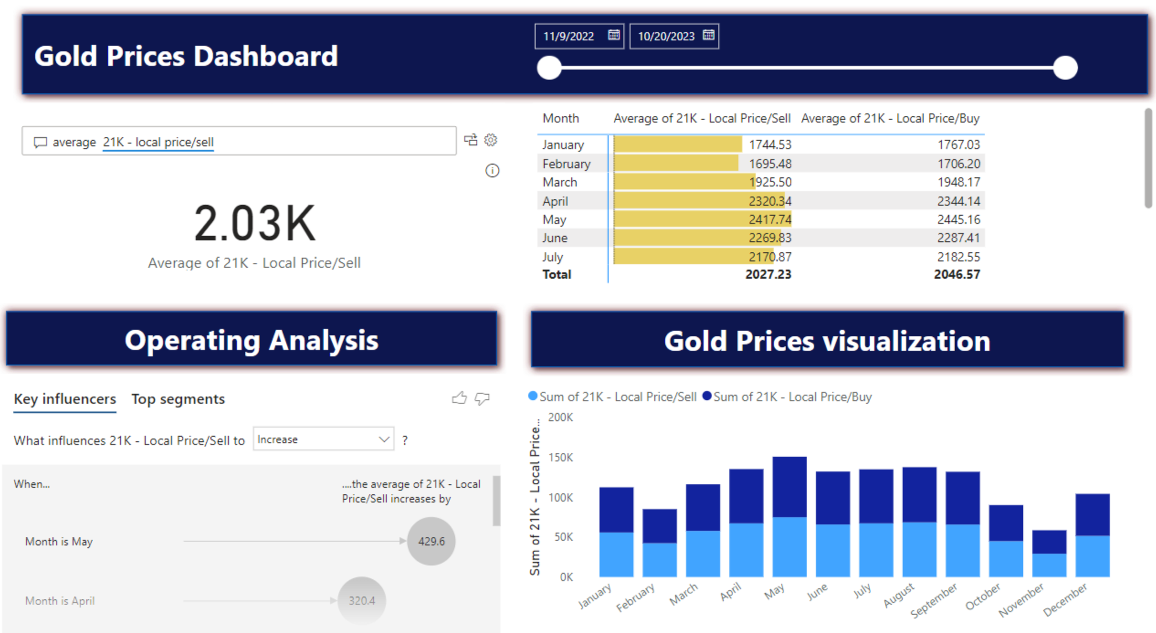
We can compare the actual values (y_test) with the predicted values (predictions_lasso)



We can make more plots to see the number of output variables. This will give us a full picture of how well the model is doing in different ways.



Let's create a complete visualization for this project by building a dashboard using Power BI.



In this dashboard, we've created five visualizations to help you understand the local market for 21k gold

- **Q&A Visualization**

  This shows the average monthly sales for 21k gold in the local market. It's like asking a question and getting the average sales answer each month.

- **Key Influencers**

  This visualization highlights the factors influencing the monthly sales of 21k gold. It points out what's causing increases or decreases in sales for each month.

- **Bar Chart**

  The bar chart provides a clear visual representation of the monthly sales for 21k gold, making it easy to compare sales performance across different months.

- **Matrix**

  This matrix displays both the sales and purchase data for 21k gold each month. You can filter the data by day and month, allowing you to see the total average sales for each month.

- **Slicer**

  The slicer is a tool that lets you filter data and the entire dashboard based on specific start and end dates. This way, you can focus on and see the actual data within a specified time range.

This dashboard is always changing with the latest information. If there are new sales or any updates, the dashboard shows them right away. It's a helpful tool to stay updated and learn more about what's happening in the local market for 21k gold.

Conclusion

This project is about finding out how much gold costs every day in Egypt. We collected and checked data, and we made cool pictures to help us see what's happening with gold in Egypt.

Target

We made this for investors, analysts, local businesses, and anyone who wonders about gold prices in Egypt.

Goals

We wanted to connect the prices of gold around the world with what's happening in Egypt. We also wanted to give helpful information for making decisions. And, we made a dashboard that's easy to use and always has the newest info. This project is like a tool that makes it simple to see what's happening with gold in Egypt.

References

[1] Python programming language

https://www.python.org/doc/

[2] Pandas Documentation

https://pandas.pydata.org/docs/

[3] NumPy Documentation

https://numpy.org/doc/stable/

[4] Matplotlib Documentation

https://matplotlib.org/stable/index.html

[5] Scikit-Learn Documentation

https://scikit-learn.org/stable/

[6] Scikit-Learn Lasso Regression Documentation

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[7] Kaggle Dataset

https://www.kaggle.com/datasets/mohamedmagdy11/egypt-gold-prices-daily-updated/

[8] Project Code on GitHub

https://github.com/danaelshrbiny10/Gold-Prices