WQD7005 Data Mining (Assignment)

Name: Liong Qiao Hui *Matric No.: 17043935/1 (WQD180039)*

Github link: https://github.com/QiaoHui213/data_mining

Video 1: https://www.loom.com/share/f505c53d852e433fa4ab62cddbedd17e

Video 2: https://www.loom.com/share/5ac650128d234834801bb6d28df271b4

Video 3: https://www.loom.com/share/b0d7878ccc4341c482e3aa4d1d09ad94

Video 4: https://www.loom.com/share/5a35913897ad420e88c097c3a57fcad0

Video 5: https://www.loom.com/share/284b2ee0b6a54acaa0c85d0b63f43f94

Milestone 1: Web Crawling the Real Time Data

Data is crawled from the website https://www.investing.com/commodities/gold-historical-data

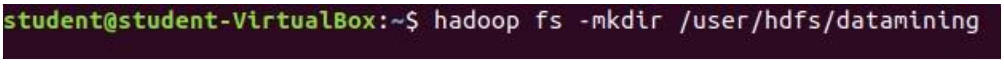
This website showed the gold future historical data and the data is crawled from this website to do further prediction.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

driver = webdriver.Chrome(executable_path="C:\\Users\\User\\Desktop\\data_mining\\chromedriver.exe")

driver.set_page_load_timeout(30)
driver.get("https://www.investing.com/commodities/gold-historical-data")

for i in range(5):
    dateRangeElements = driver.find_elements_by_xpath("//div[@id='widgetFieldDateRange']")

    if len(dateRangeElements) > 0:
        dateRangeElements[0].click()
        break
    else:
        print(str(i + 1) + " trial, can't find element")

    time.sleep(1)
    i = i + 1

elementStartDate = driver.find_element_by_xpath("//input[@id='startDate']")
elementStartDate.clear()
elementStartDate.send_keys("01/01/2019")

driver.find_element_by_xpath("//a[@id='applyBtn']").click()
```

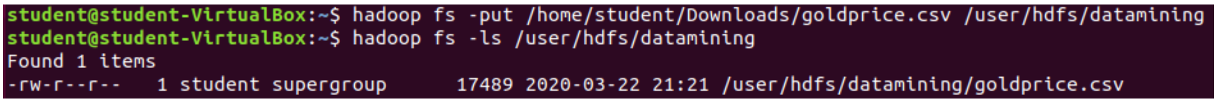
```python
time.sleep(3)

tableData = driver.find_elements_by_xpath("//table[@id='curr_table']/tbody/tr/td")

resultFile = open("goldprice_" + time.strftime("%d-%b-%Y %H-%M-%S", time.localtime()) + ".csv", 'w')

resultFile.write("Date,Price,Open,High,Low,Volume,Change %\n")

column = 1
for data in tableData:
    if column != 7:
        resultFile.write((data.text).replace(',', '') + ", ")
        column = column + 1
    else:
        resultFile.write(data.text + "\n")
        column = 1

resultFile.close()

driver.close()
```

Milestone 2: Store data into hive data warehouse

In terminal, create a directory called ‘datamining’ in hadoop folder. Then, put the *goldprice.csv* file into the folder created.

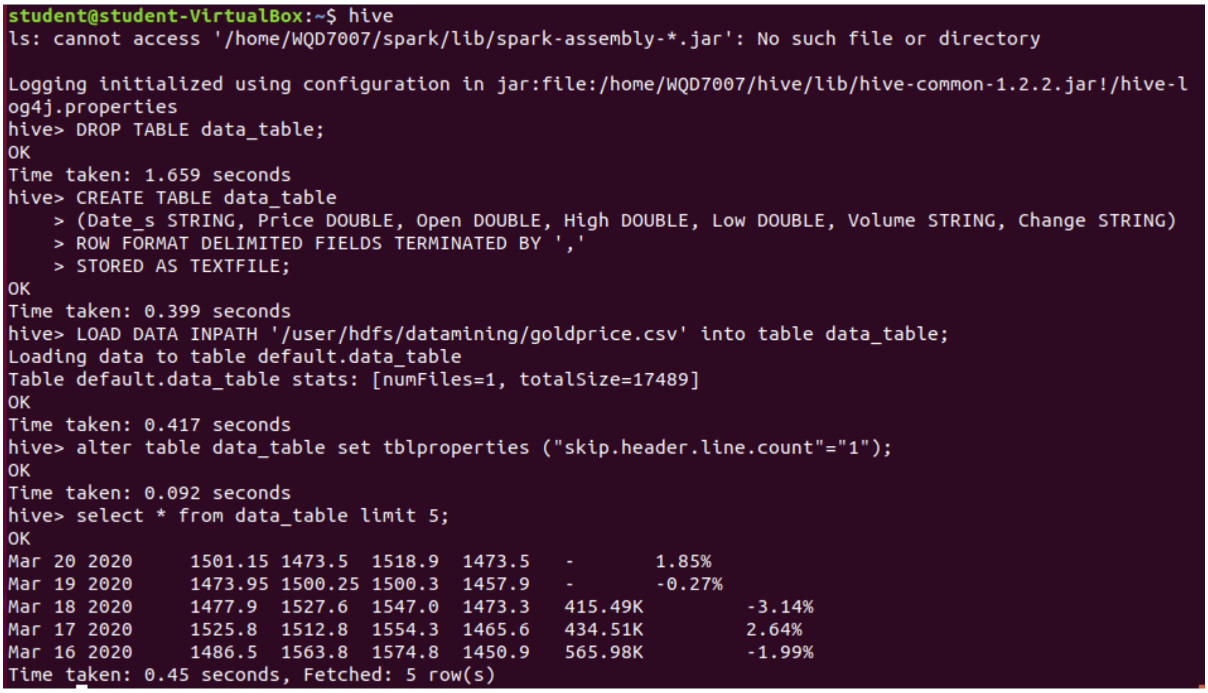


```
student@student-VirtualBox:~$ hadoop fs -mkdir /user/hdfs/datamining
```



```
student@student-VirtualBox:~$ hadoop fs -put /home/student/Downloads/goldprice.csv /user/hdfs/datamining
student@student-VirtualBox:~$ hadoop fs -ls /user/hdfs/datamining
Found 1 items
-rw-r--r--   1 student supergroup      17489 2020-03-22 21:21 /user/hdfs/datamining/goldprice.csv
```

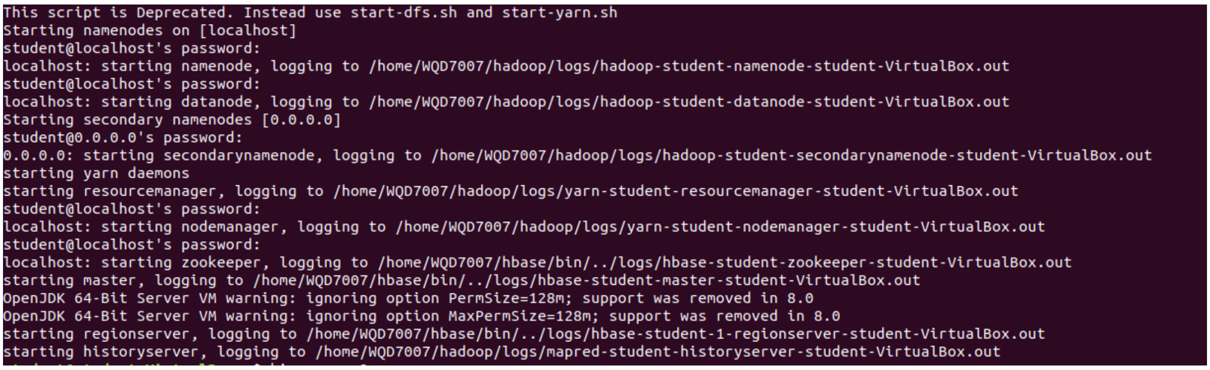
In hive terminal, create a table called ‘data_table’ and stored the data into hive warehouse. Then, the header of the table is removed and the first 5 rows of data is selected and print out in the result.



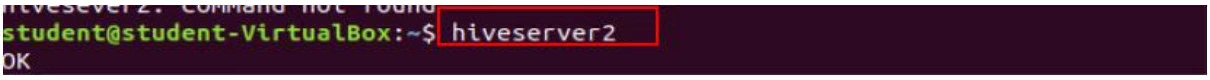
```
student@student-VirtualBox:~$ hive
ls: cannot access '/home/WQD7007/spark/lib/spark-assembly-*.jar': No such file or directory

Logging initialized using configuration in jar:file:/home/WQD7007/hive/lib/hive-common-1.2.2.jar!/hive-l
og4j.properties
hive> DROP TABLE data_table;
OK
Time taken: 1.659 seconds
hive> CREATE TABLE data_table
    > (Date_s STRING, Price DOUBLE, Open DOUBLE, High DOUBLE, Low DOUBLE, Volume STRING, Change STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.399 seconds
hive> LOAD DATA INPATH '/user/hdfs/datamining/goldprice.csv' into table data_table;
Loading data to table default.data_table
Table default.data_table stats: [numFiles=1, totalSize=17489]
OK
Time taken: 0.417 seconds
hive> alter table data_table set tblproperties ("skip.header.line.count"="1");
OK
Time taken: 0.092 seconds
hive> select * from data_table limit 5;
OK
Mar 20 2020     1501.15 1473.5  1518.9  1473.5  -       1.85%
Mar 19 2020     1473.95 1500.25 1500.3  1457.9  -       -0.27%
Mar 18 2020     1477.9  1527.6  1547.0  1473.3  415.49K         -3.14%
Mar 17 2020     1525.8  1512.8  1554.3  1465.6  434.51K         2.64%
Mar 16 2020     1486.5  1563.8  1574.8  1450.9  565.98K         -1.99%
Time taken: 0.45 seconds, Fetched: 5 row(s)
```

Milestone 3: Accessing Hive Data Warehouse by using Python

In terminal, start Hadoop and hiveserver2

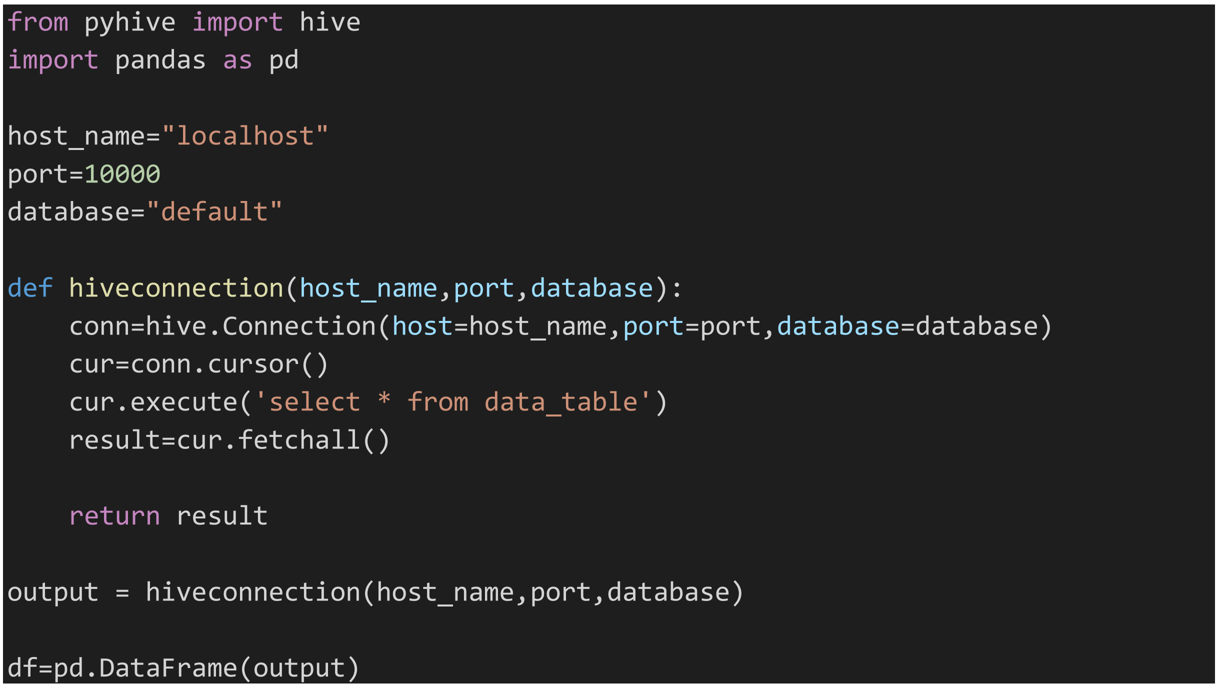


```
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
student@localhost's password:
localhost: starting namenode, logging to /home/WQD7007/hadoop/logs/hadoop-student-namenode-student-VirtualBox.out
student@localhost's password:
localhost: starting datanode, logging to /home/WQD7007/hadoop/logs/hadoop-student-datanode-student-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
student@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /home/WQD7007/hadoop/logs/hadoop-student-secondarynamenode-student-VirtualBox.out
starting yarn daemons
starting resourcemanager, logging to /home/WQD7007/hadoop/logs/yarn-student-resourcemanager-student-VirtualBox.out
student@localhost's password:
localhost: starting nodemanager, logging to /home/WQD7007/hadoop/logs/yarn-student-nodemanager-student-VirtualBox.out
student@localhost's password:
localhost: starting zookeeper, logging to /home/WQD7007/hbase/bin/../logs/hbase-student-zookeeper-student-VirtualBox.out
starting master, logging to /home/WQD7007/hbase/bin/../logs/hbase-student-master-student-VirtualBox.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
starting regionserver, logging to /home/WQD7007/hbase/bin/../logs/hbase-student-1-regionserver-student-VirtualBox.out
starting historyserver, logging to /home/WQD7007/hadoop/logs/mapred-student-historyserver-student-VirtualBox.out
```



```
hiveserver2: command not found
student@student-VirtualBox:~$ hiveserver2
OK
```

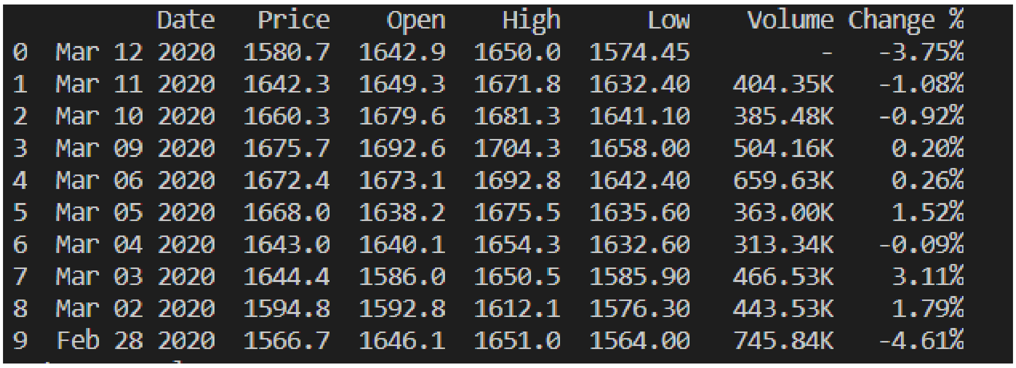
To access data from hive, we import pyhive library in python script to connect to hive server. host_name=localhost, port=port & database=database



```
from pyhive import hive
import pandas as pd

host_name="localhost"
port=10000
database="default"

def hiveconnection(host_name,port,database):
    conn=hive.Connection(host=host_name,port=port,database=database)
    cur=conn.cursor()
    cur.execute('select * from data_table')
    result=cur.fetchall()

    return result

output = hiveconnection(host_name,port,database)

df=pd.DataFrame(output)
```

Output:

The data are loaded and top 10 row of data is read.



```
          Date   Price    Open    High      Low    Volume Change %
0  Mar 12 2020  1580.7  1642.9  1650.0  1574.45         -   -3.75%
1  Mar 11 2020  1642.3  1649.3  1671.8  1632.40   404.35K   -1.08%
2  Mar 10 2020  1660.3  1679.6  1681.3  1641.10   385.48K   -0.92%
3  Mar 09 2020  1675.7  1692.6  1704.3  1658.00   504.16K    0.20%
4  Mar 06 2020  1672.4  1673.1  1692.8  1642.40   659.63K    0.26%
5  Mar 05 2020  1668.0  1638.2  1675.5  1635.60   363.00K    1.52%
6  Mar 04 2020  1643.0  1640.1  1654.3  1632.60   313.34K   -0.09%
7  Mar 03 2020  1644.4  1586.0  1650.5  1585.90   466.53K    3.11%
8  Mar 02 2020  1594.8  1592.8  1612.1  1576.30   443.53K    1.79%
9  Feb 28 2020  1566.7  1646.1  1651.0  1564.00   745.84K   -4.61%
```

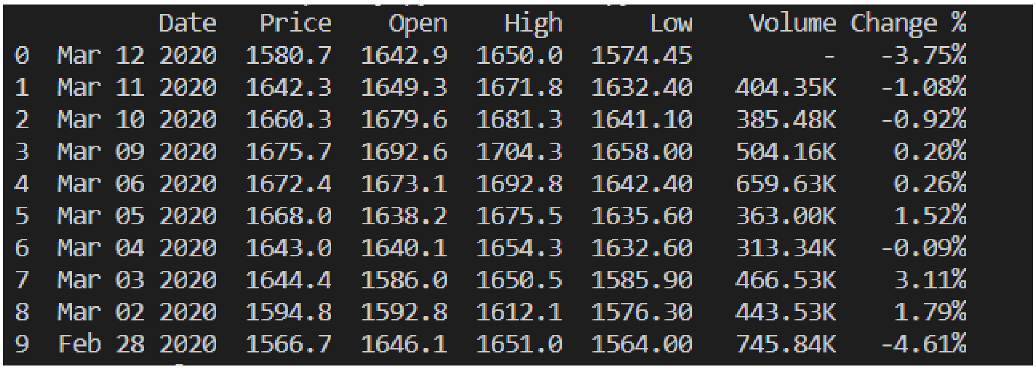
Milestone 4: Interpretation of data & Communication of Insight of data

The module needed are imported & the dataset is read.

```
# Import module
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn
import yfinance as yf

# Read Data
df = pd.read_csv("C:\\Users\\User\\Desktop\\data_mining\\data_mining\\goldpric
e_12-Mar-2020_22-26-27.csv")
print(df.head(10))
```

Output:

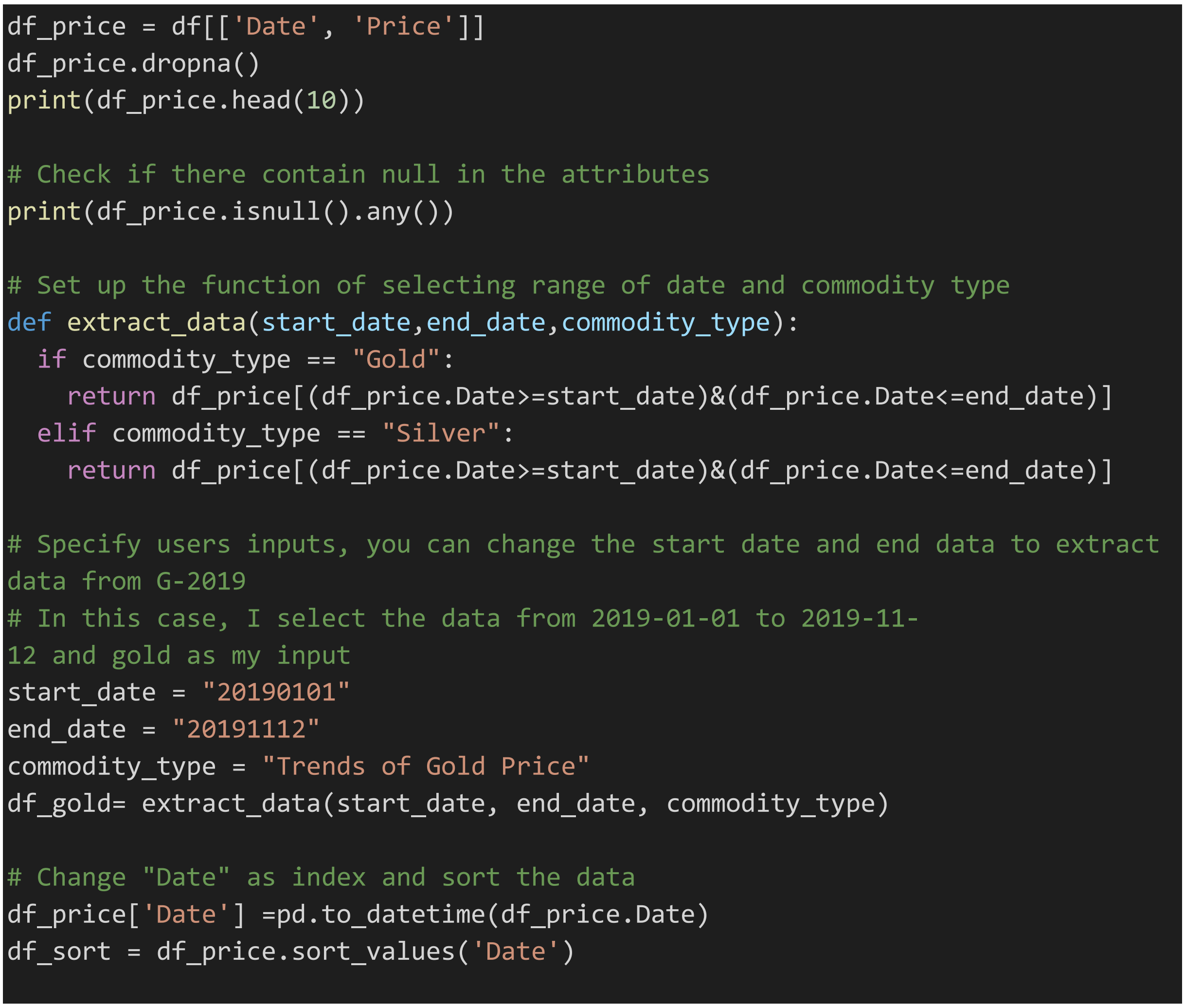
The top 10 rows of dataset are printed out.



```
           Date   Price   Open    High      Low    Volume Change %
0  Mar 12 2020  1580.7  1642.9  1650.0  1574.45        -   -3.75%
1  Mar 11 2020  1642.3  1649.3  1671.8  1632.40  404.35K   -1.08%
2  Mar 10 2020  1660.3  1679.6  1681.3  1641.10  385.48K   -0.92%
3  Mar 09 2020  1675.7  1692.6  1704.3  1658.00  504.16K    0.20%
4  Mar 06 2020  1672.4  1673.1  1692.8  1642.40  659.63K    0.26%
5  Mar 05 2020  1668.0  1638.2  1675.5  1635.60  363.00K    1.52%
6  Mar 04 2020  1643.0  1640.1  1654.3  1632.60  313.34K   -0.09%
7  Mar 03 2020  1644.4  1586.0  1650.5  1585.90  466.53K    3.11%
8  Mar 02 2020  1594.8  1592.8  1612.1  1576.30  443.53K    1.79%
9  Feb 28 2020  1566.7  1646.1  1651.0  1564.00  745.84K   -4.61%
```

The date format is changed & the dataset is under pre-processing. Only date and price is selected for further prediction.

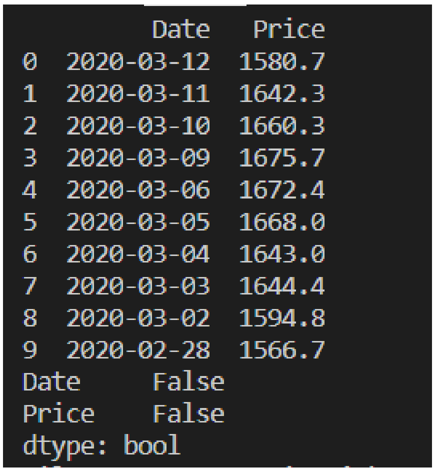
```
# Change the format of "Date"
df["Date"] = pd.to_datetime(df["Date"]).dt.strftime('%Y%m%d')
# Remove the "," in the "Price", "Open', 'Low', and 'High' column and change t
he "string" type to "float"
df['Price']=df['Price'].astype(str).str.replace(',', '').astype(float)
df['Open']=df['High'].astype(str).str.replace(',', '').astype(float)
df['Low']=df['Low'].astype(str).str.replace(',', '').astype(float)
df['High']=df['High'].astype(str).str.replace(',', '').astype(float)
df['Volume']=df['Volume'].replace({'K': '*1e3', '-
': '1'}, regex=True).map(pd.eval)
df['Change %']=df['Change %'].replace({'%': '*1e-2'}, regex=True).map(pd.eval)
df['Volume'] = df['Volume'].replace(1.0,np.NaN)

# Select Date and Price only
```



```
df_price = df[['Date', 'Price']]
df_price.dropna()
print(df_price.head(10))

# Check if there contain null in the attributes
print(df_price.isnull().any())

# Set up the function of selecting range of date and commodity type
def extract_data(start_date,end_date,commodity_type):
  if commodity_type == "Gold":
    return df_price[(df_price.Date>=start_date)&(df_price.Date<=end_date)]
  elif commodity_type == "Silver":
    return df_price[(df_price.Date>=start_date)&(df_price.Date<=end_date)]

# Specify users inputs, you can change the start date and end data to extract
data from G-2019
# In this case, I select the data from 2019-01-01 to 2019-11-
12 and gold as my input
start_date = "20190101"
end_date = "20191112"
commodity_type = "Trends of Gold Price"
df_gold= extract_data(start_date, end_date, commodity_type)

# Change "Date" as index and sort the data
df_price['Date'] =pd.to_datetime(df_price.Date)
df_sort = df_price.sort_values('Date')
```

Output:

Only ‘Date’ & ‘Price’ are selected. The data is checked if there contain null in the attributes.



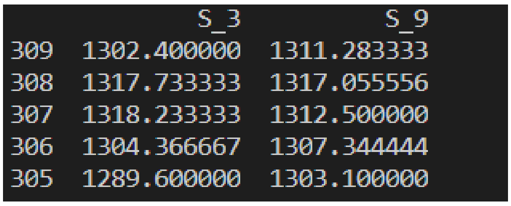
```
         Date   Price
0  2020-03-12  1580.7
1  2020-03-11  1642.3
2  2020-03-10  1660.3
3  2020-03-09  1675.7
4  2020-03-06  1672.4
5  2020-03-05  1668.0
6  2020-03-04  1643.0
7  2020-03-03  1644.4
8  2020-03-02  1594.8
9  2020-02-28  1566.7
Date     False
Price    False
dtype: bool
```

An explanatory variable is a variable that is manipulated to determine the value of the Gold Price the next day. Simply, they are the features which we want to use to predict the Gold Price. The explanatory variables in this strategy are the moving averages for past 3 days and 9 days.

The data is visualized and split into training set & test set.

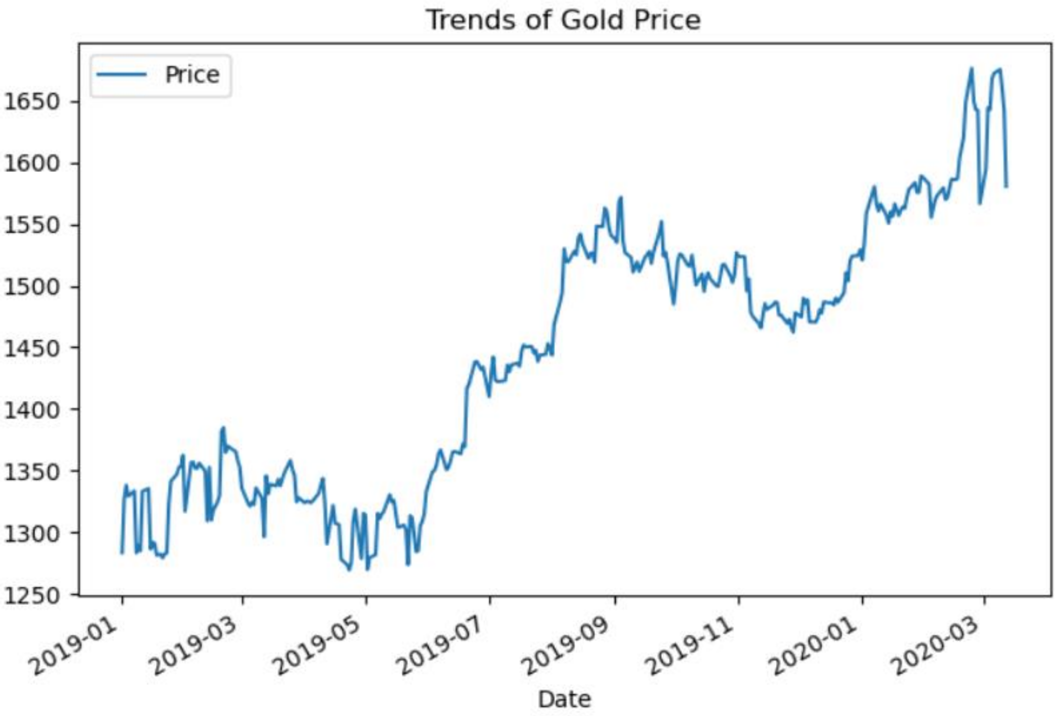
```
# Visualize the Data
df_sort.plot(x='Date',y='Price')
plt.title(commodity_type)
plt.show()

df_sort['S_3'] = df_sort['Price'].shift(1).rolling(window=3).mean()
df_sort['S_9']= df_sort['Price'].shift(1).rolling(window=9).mean()
df_sort= df_sort.dropna()
X = df_sort[['S_3','S_9']]
print(X.head())

y = df_sort['Price']
print(y.head())

t=.8
t = int(t*len(df_sort))
# Train dataset
X_train = X[:t]
y_train = y[:t]
# Test dataset
X_test = X[t:]
y_test = y[t:]
```

Output:



```
             S_3          S_9
309  1302.400000  1311.283333
308  1317.733333  1317.055556
307  1318.233333  1312.500000
306  1304.366667  1307.344444
305  1289.600000  1303.100000
```

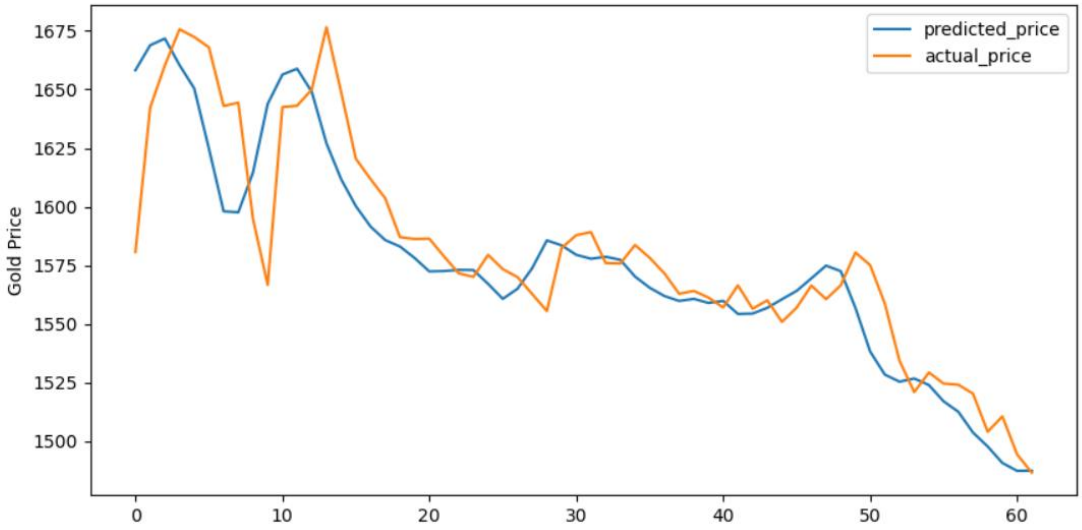
Output:



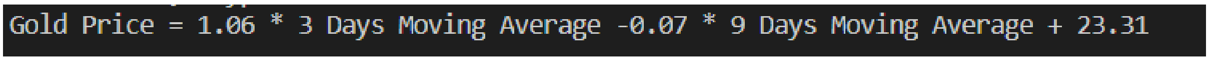
Linear regression algorithm is used to predict the future gold price.

```
linear = LinearRegression().fit(X_train,y_train)
print ("Gold Price =", round(linear.coef_[0],2), "* 3 Days Moving Average", ro
und(linear.coef_[1],2), "* 9 Days Moving Average +", round(linear.intercept_,2
))

predicted_price = linear.predict(X_test)
predicted_price = pd.DataFrame(predicted_price,index=y_test.index,columns = ['
price'])
predicted_price.plot(figsize=(10,5))
y_test.plot()
plt.legend(['predicted_price','actual_price'])
plt.ylabel("Gold ETF Price")
plt.show()

r2_score = linear.score(X[t:],y[t:])*100
float("{0:.2f}".format(r2_score))
```

Output:



The equation for linear regression:

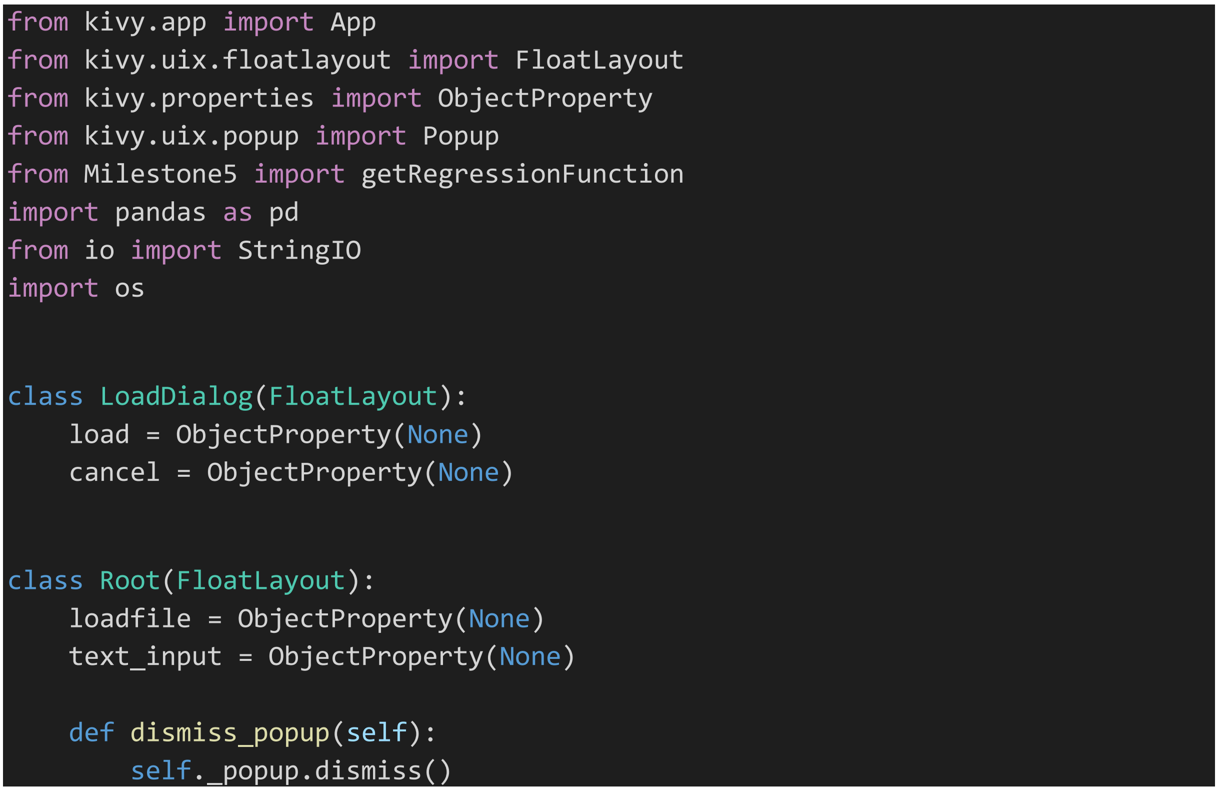


```
Gold Price = 1.06 * 3 Days Moving Average -0.07 * 9 Days Moving Average + 23.31
```

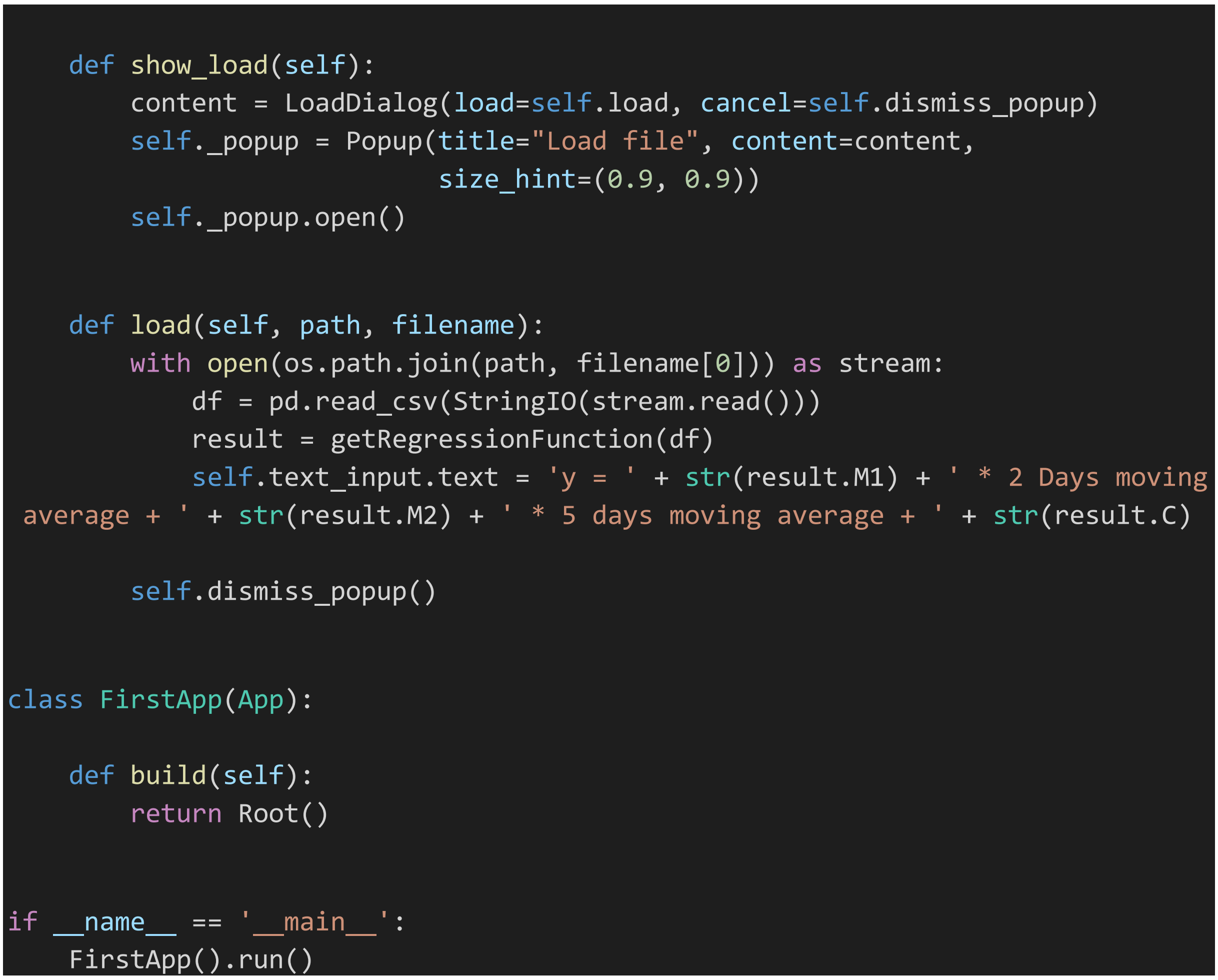
Milestone 5: Kivy mobile apps & Flask Deployment

To deploy a **kivy mobile apps**, it is going to have .kv file and a .py file.

This is the python file that I import and run the kivy apps.

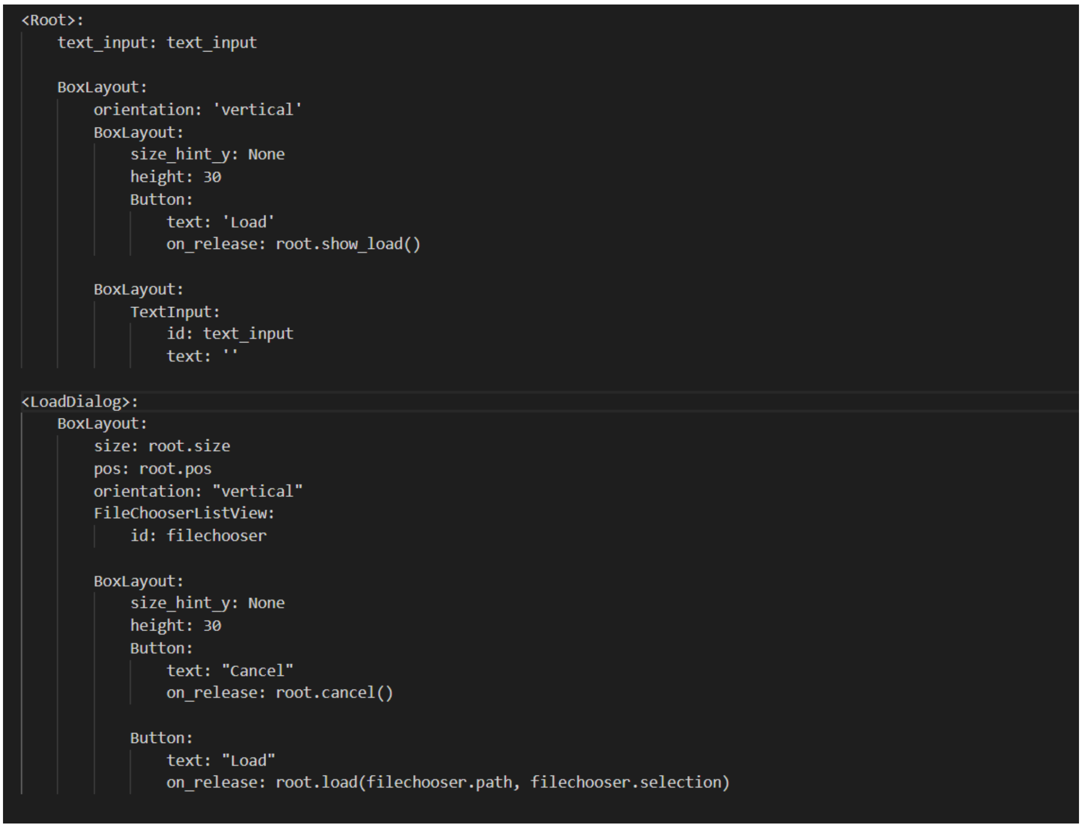


```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
from kivy.uix.popup import Popup
from Milestone5 import getRegressionFunction
import pandas as pd
from io import StringIO
import os


class LoadDialog(FloatLayout):
    load = ObjectProperty(None)
    cancel = ObjectProperty(None)


class Root(FloatLayout):
    loadfile = ObjectProperty(None)
    text_input = ObjectProperty(None)

    def dismiss_popup(self):
        self._popup.dismiss()
```



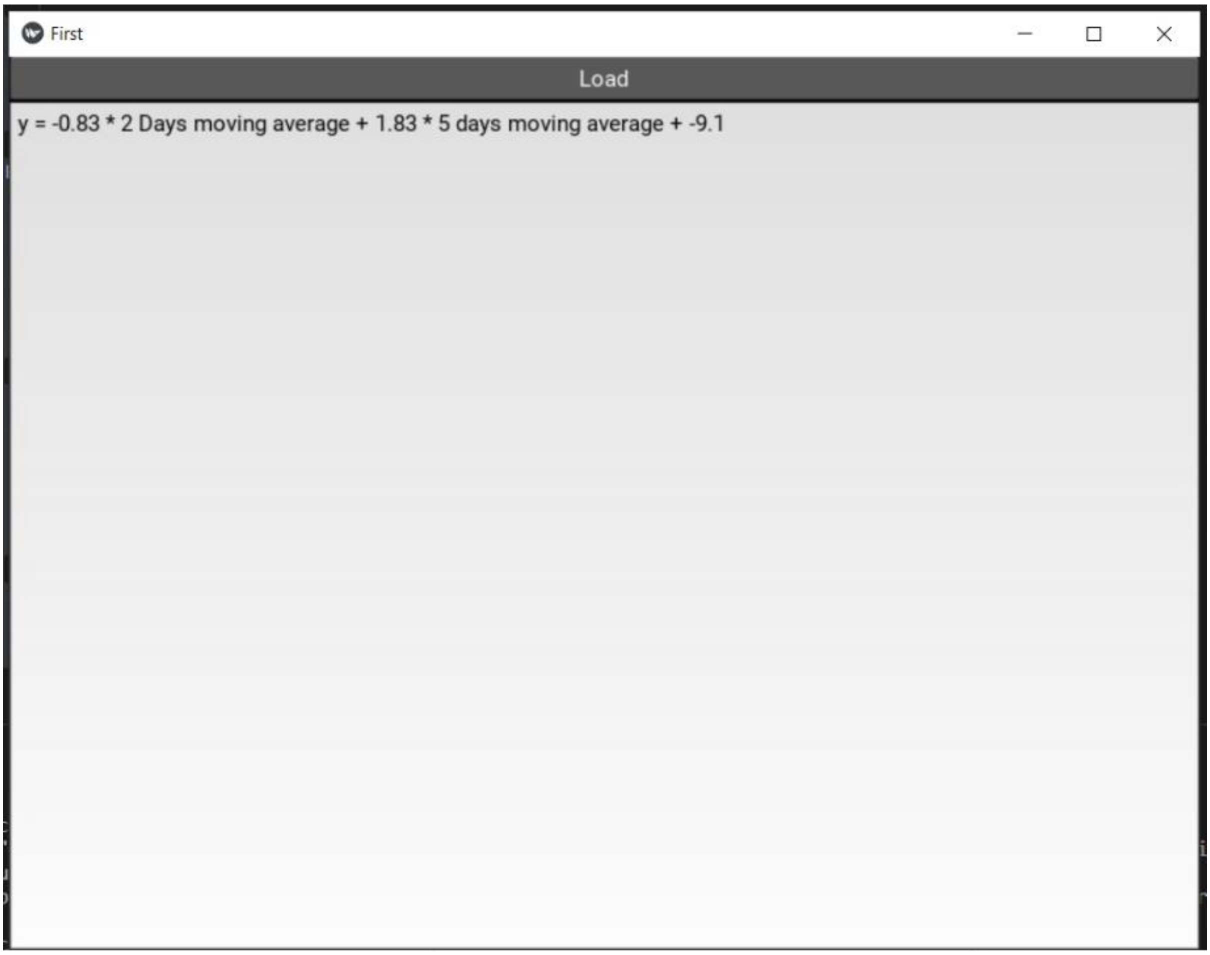
```
    def show_load(self):
        content = LoadDialog(load=self.load, cancel=self.dismiss_popup)
        self._popup = Popup(title="Load file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()


    def load(self, path, filename):
        with open(os.path.join(path, filename[0])) as stream:
            df = pd.read_csv(StringIO(stream.read()))
            result = getRegressionFunction(df)
            self.text_input.text = 'y = ' + str(result.M1) + ' * 2 Days moving
 average + ' + str(result.M2) + ' * 5 days moving average + ' + str(result.C)

        self.dismiss_popup()


class FirstApp(App):

    def build(self):
        return Root()


if __name__ == '__main__':
    FirstApp().run()
```

The following is the kv file that I used.



```
<Root>:
    text_input: text_input

    BoxLayout:
        orientation: 'vertical'
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: 'Load'
                on_release: root.show_load()

        BoxLayout:
            TextInput:
                id: text_input
                text: ''

<LoadDialog>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        FileChooserListView:
            id: filechooser

        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                on_release: root.cancel()

            Button:
                text: "Load"
                on_release: root.load(filechooser.path, filechooser.selection)
```

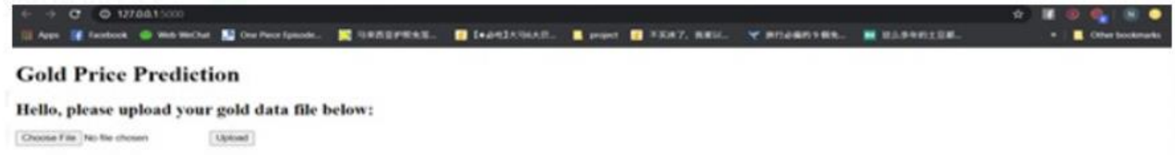
This is the layout of my kivy mobile apps.



y = -0.83 * 2 Days moving average + 1.83 * 5 days moving average + -9.1

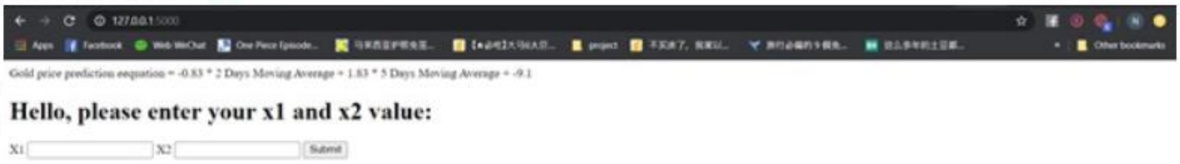
This is the **flask website deployment.** I create __init__.py, home.html, input.html, result.html.

The following is my output layout of the website.



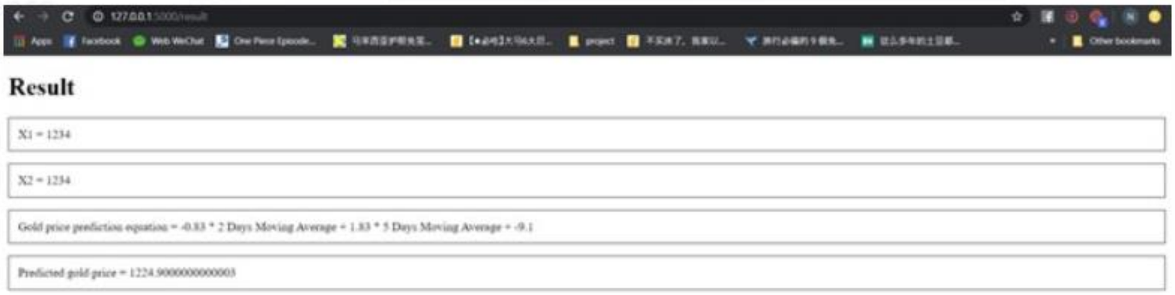
Gold Price Prediction

Hello, please upload your gold data file below:



Gold price prediction eequation = -0.83 * 2 Days Moving Average + 1.83 * 5 Days Moving Average + -9.1

Hello, please enter your x1 and x2 value:



Result

X1 = 1234

X2 = 1234

Gold price prediction equation = -0.83 * 2 Days Moving Average + 1.83 * 5 Days Moving Average + -9.1

Predicted gold price = 1224.9000000000003