Name: Hii Nian Yu

Matric Number: 17007070/1 (WQD180038)

Github link: <https://github.com/hiinianyu/DataMining>

Video link for Milestone 1: <https://www.loom.com/share/9cce920834144fac9162e59de9c9d732>

Video link for Milestone 2: <https://www.loom.com/share/1d115fd969b441c6b4044a24bb5d8c6d>

Video link for Milestone 3: <https://www.loom.com/share/01c7d2995084443cacbf816b8eb7e713>

Video link for Milestone 4 & 5: <https://www.loom.com/share/e39474f8884c4ee79d1f08172546d052>

Title: Gold Price Prediction

**Milestone 1**

I am using selenium to crawl data from a website: <https://www.investing.com/commodities/gold-historical-data>

Below is the code:

Driver = webdriver.Chrome(executable\_path="/home/student/DM\_assignment/chromedriver\_linux64/chromedriver")

driver.set\_page\_load\_timeout(30)

driver.maximize\_window()

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:

time.sleep(3)

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()

time.sleep(3)

tableData = driver.find\_elements\_by\_xpath("//table[@id='curr\_table']/tbody/tr/td")

fileName = r"Result\goldprice\_" + time.strftime("%d-%b-%Y\_%H-%M-%S", time.localtime()) + ".csv"

resultFile = open(fileName, 'w')

resultFile.write("Date\_D,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()

After data saved, I will put the command below to save the file automatically into HDFS system.

def run\_cmd(args\_list):

"""

run linux commands

"""

# import subprocess

print('Running system command: {0}'.format(' '.join(args\_list)))

proc = subprocess.Popen(args\_list, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

s\_output, s\_err = proc.communicate()

s\_return = proc.returncode

return s\_return, s\_output, s\_err

(ret, out, err)= run\_cmd(['hadoop', 'fs', '-put', '/home/student/DM\_assignment/' + fileName, '/user/hdfs/DataMining'])

driver.close()

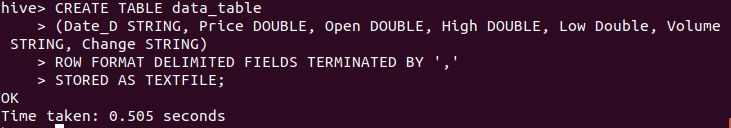
**Milestone 2**

Step to store data into hive:

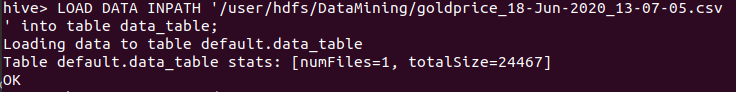
The data file has been saved into HDFS system based on Milestone 1. Double check whether it is stored inside the system using *-ls* command.

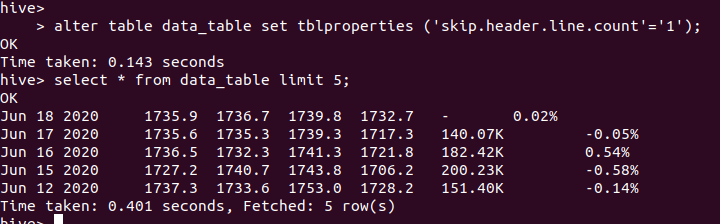


Then create an external table named *data\_table* in hive.



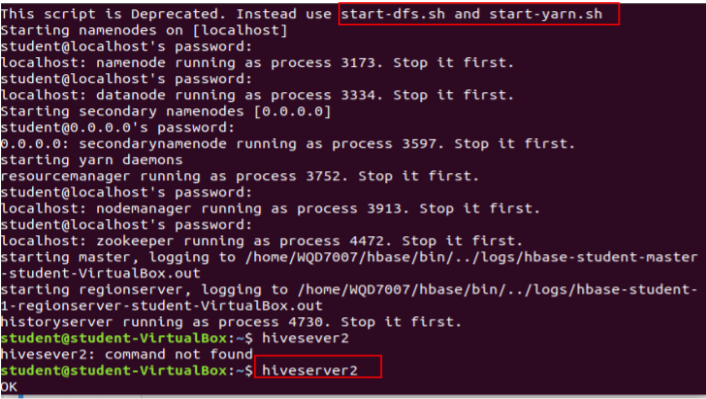
Load the csv file into the table in hive.





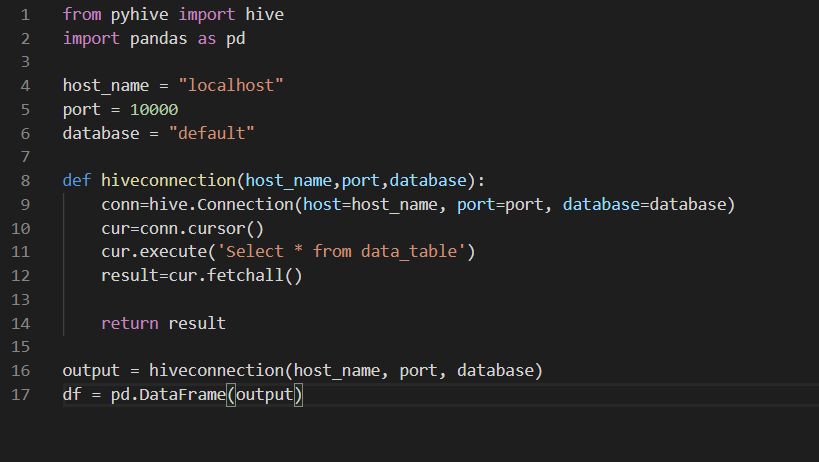
**Milestone3**

In terminal, start Hadoop and hiveserver2



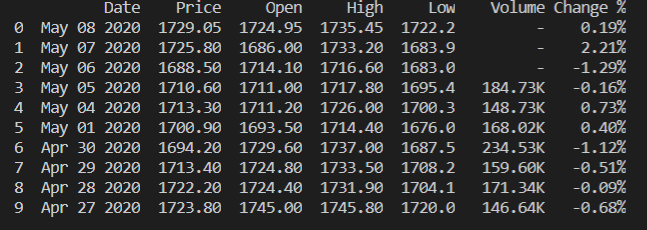
In python, import the pyhive library to connect to hive server

Host = localhost, port = 10000, database = default



Read the top 10 lines of the data





**Milestone4**

Below are the packages that using in this milestone.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.dates import date2num

from datetime import datetime

import subprocess

from sklearn import preprocessing

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from math import ceil

from math import sqrt

First, we load the data.

Df = pd.read\_csv('goldprice\_08-May-2020 19-11-07.csv')

Use df.head() to read first 6 lines of data.

print(Df.head())

Now, doing some preprocessing of data.

print(Df.isnull().any())

# 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 the "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)

print(Df.head())

We sort the data based on ‘Date’ attribute and plot the graph to see the trend.

# Change "Date" as index and sort the data

Df['Date'] =pd.to\_datetime(Df.Date)

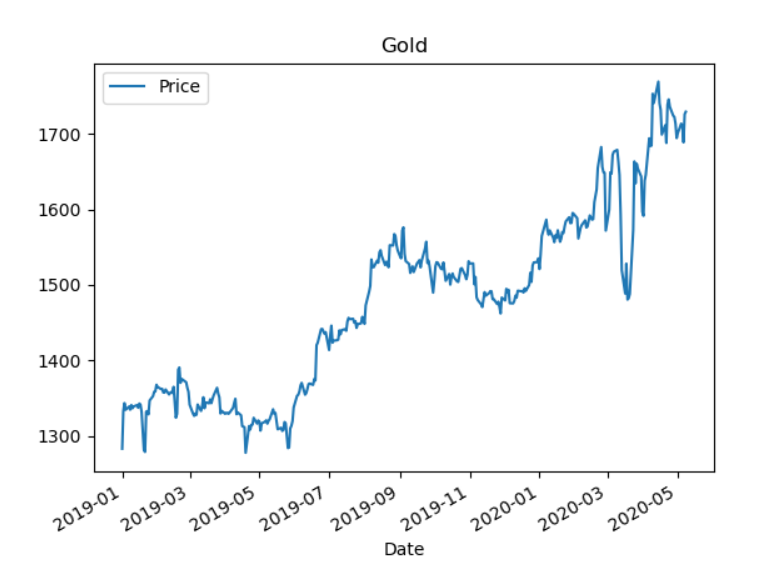
Df.sort\_values('Date')

Df.plot(x='Date',y='Price')

plt.title('Gold')

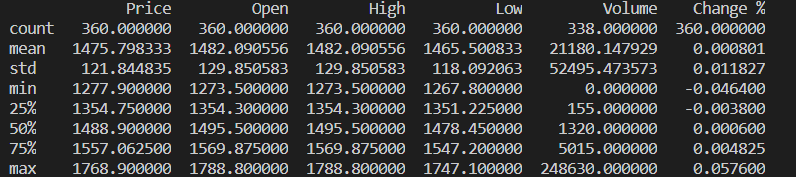
plt.show()

The graph shown as below:



# The statistical properties of data

print(Df.describe())

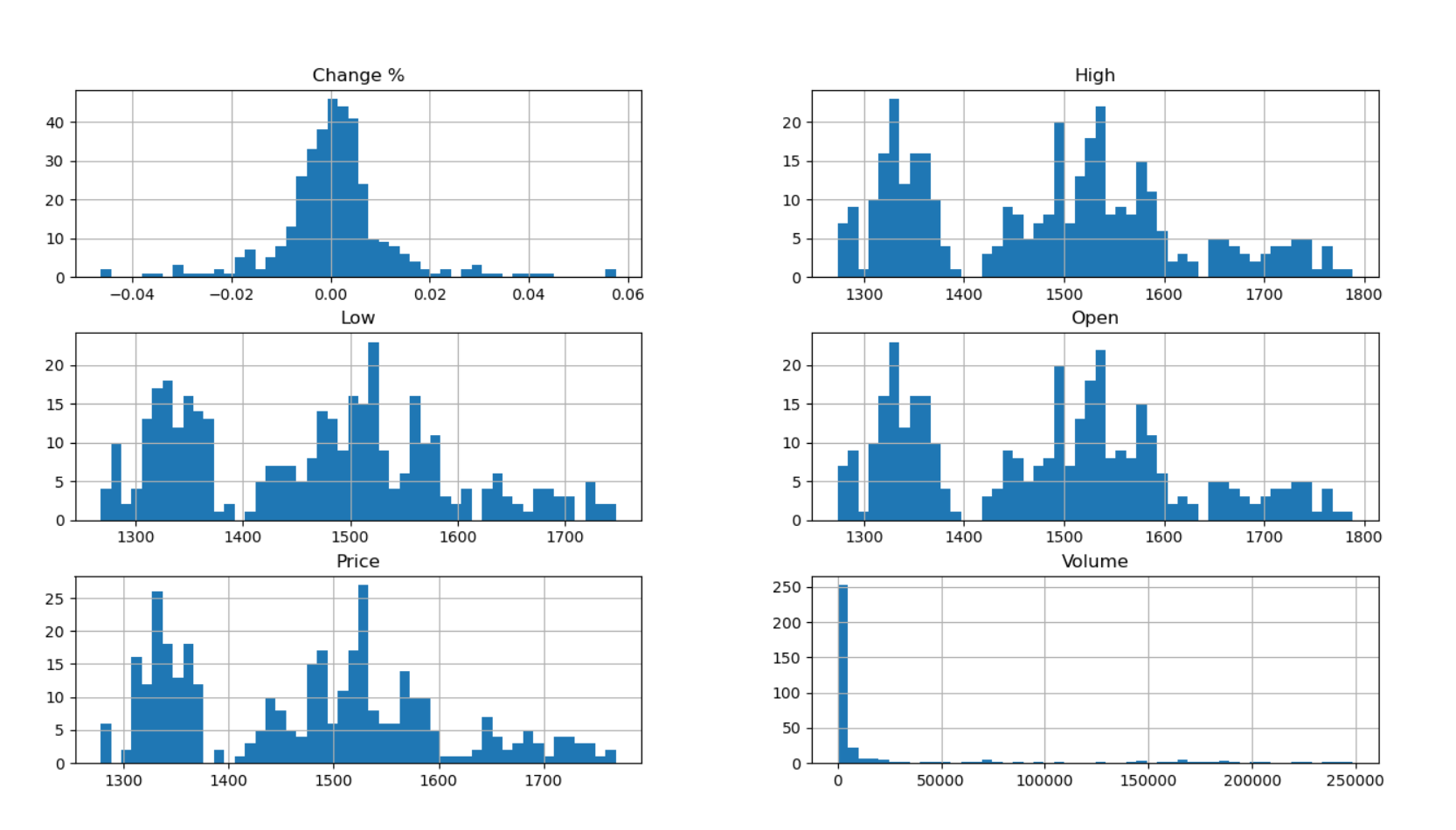


# Darw histgram of each column

plt.rcParams["figure.figsize"] = (20,8)

hist = Df[["Price","High","Low","Open", 'Volume', 'Change %']].hist(bins=50)

plt.show()



# calculate the correlation matrix

corr = Df[["Price","High","Low","Open",'Volume', 'Change %']].corr()

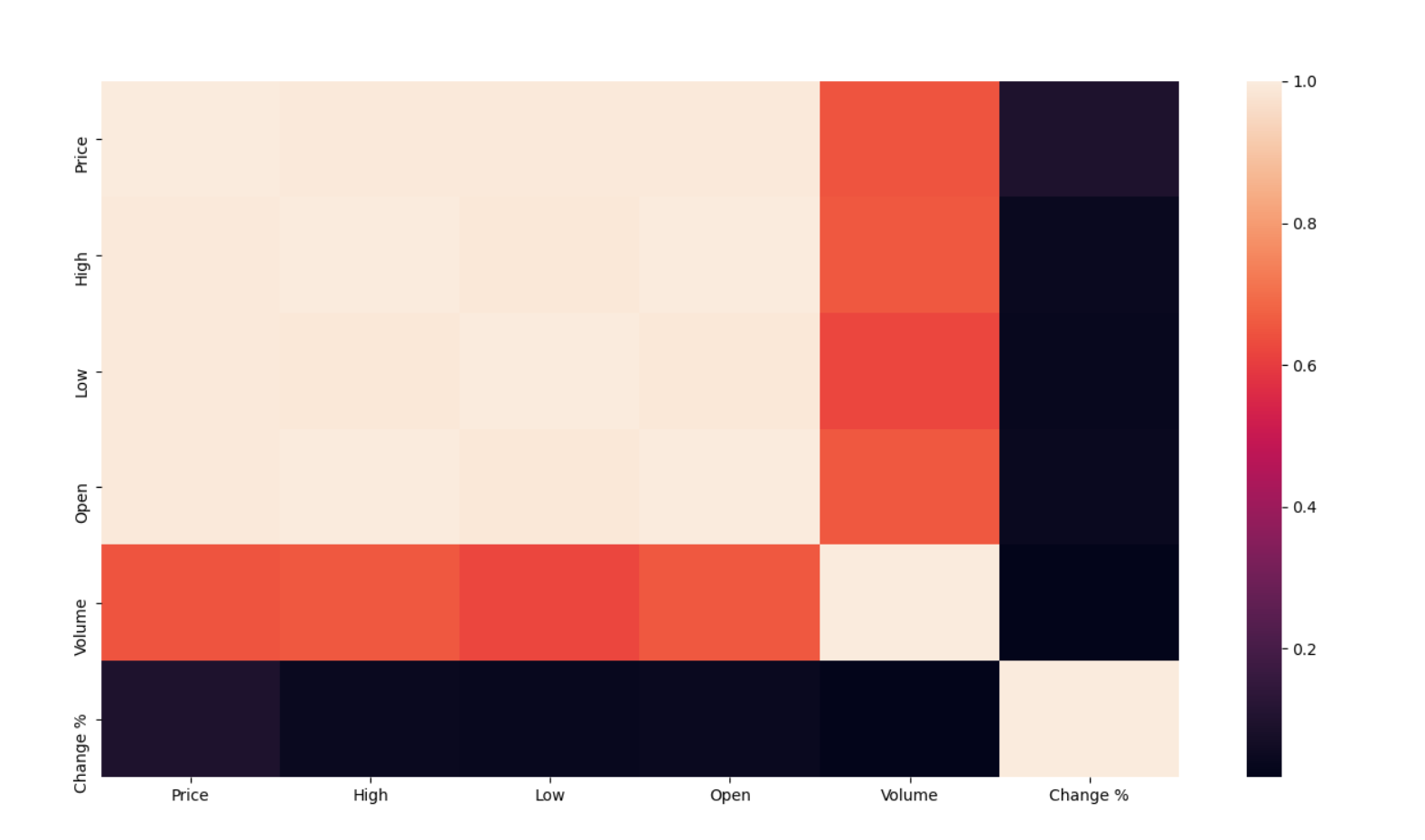
# plot the heatmap to show the correlations among different price columns

sns.heatmap(corr,

xticklabels=corr.columns,

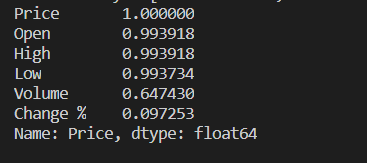
yticklabels=corr.columns)

plt.show()



# Check the correlation score

print (corr['Price'].sort\_values(ascending=False), '\n')



# create two new columns with lagged terms with different moving window size

Df['S\_1'] = Df["Price"].shift(1).rolling(window=2, center = True).mean()

Df['S\_2'] = Df["Price"].shift(1).rolling(window=5, center = True).mean()

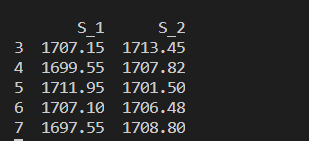
# Define exploratory variables

# Finding moving average of past 2 day and 5 days

Df = Df.dropna()

X = Df[['S\_1','S\_2']]

print(X.head())



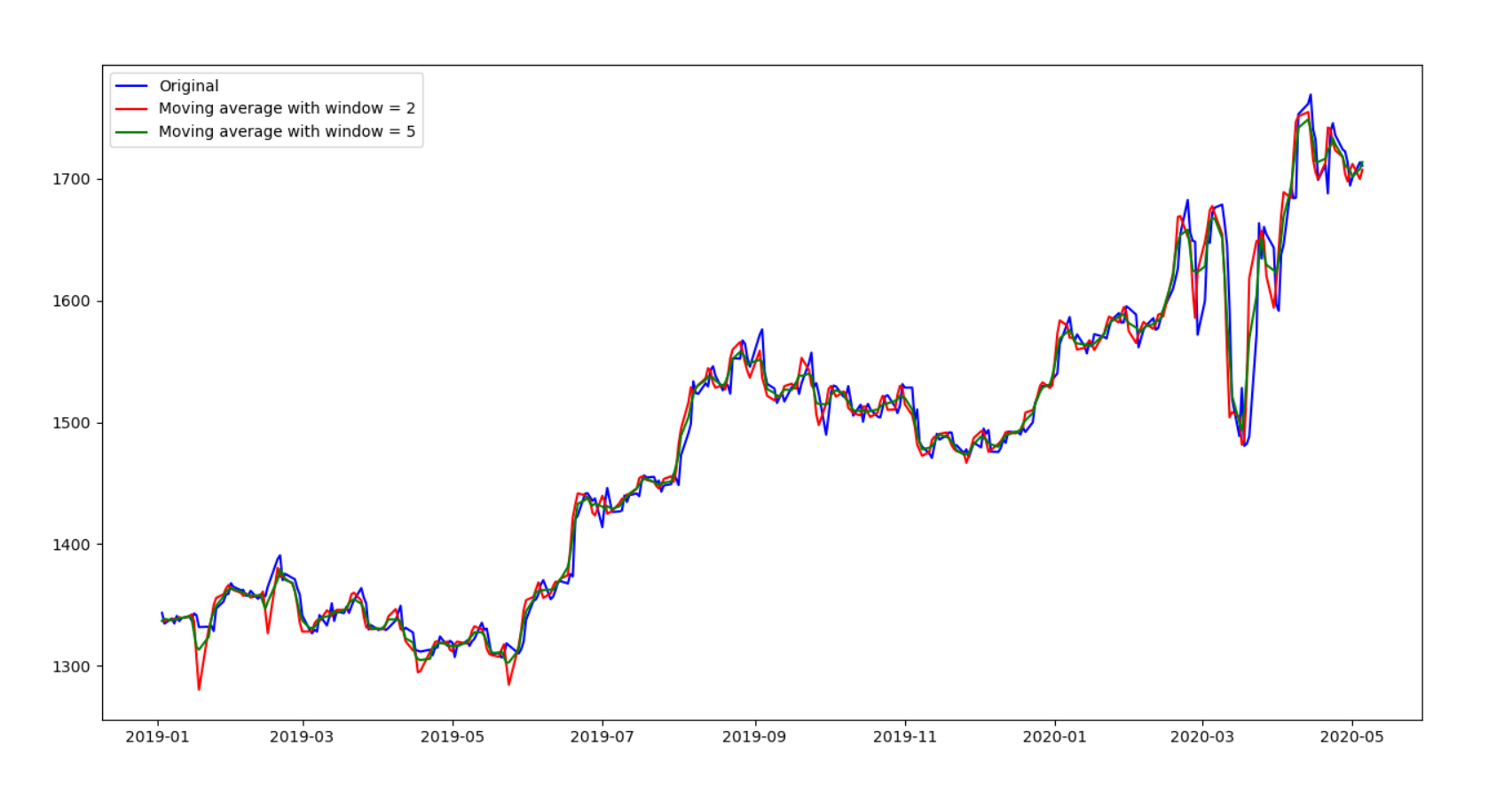
plt.plot(Df['Date'], Df['Price'], color="blue", label = "Original")

plt.plot(Df['Date'], Df['S\_1'], color="red", label= "Moving average with window = 2")

plt.plot(Df['Date'], Df["S\_2"],color="green", label= "Moving average with window = 5")

plt.legend(loc='best')

plt.show()



# dependent variable

y = Df["Price"]

y.head()

print(Df.shape)

# Split into train and test

# training size

t = 0.9

count = int(ceil(X.shape[0]\*t))

X = X.iloc[::-1]

y = y.iloc[::-1]

X\_train = X[:count]

X\_test = X[count:]

y\_train = y[:count]

y\_test = y[count:]

# Performing linear regression

linear = LinearRegression().fit(X\_train, y\_train)

print("Gold Price =", round(linear.coef\_[0], 2), "\* 2 Days Moving Average",

round(linear.coef\_[1], 2), "\* 5 Days Moving Average +",

round(linear.intercept\_, 2))

The equation is:

# Predict prices

predicted\_price = linear.predict(X\_test)

predicted\_price = pd.DataFrame(

predicted\_price, index=y\_test.index, columns=['price'])

predicted\_price = pd.DataFrame(predicted\_price,index=y\_test.index,columns = ['price'])

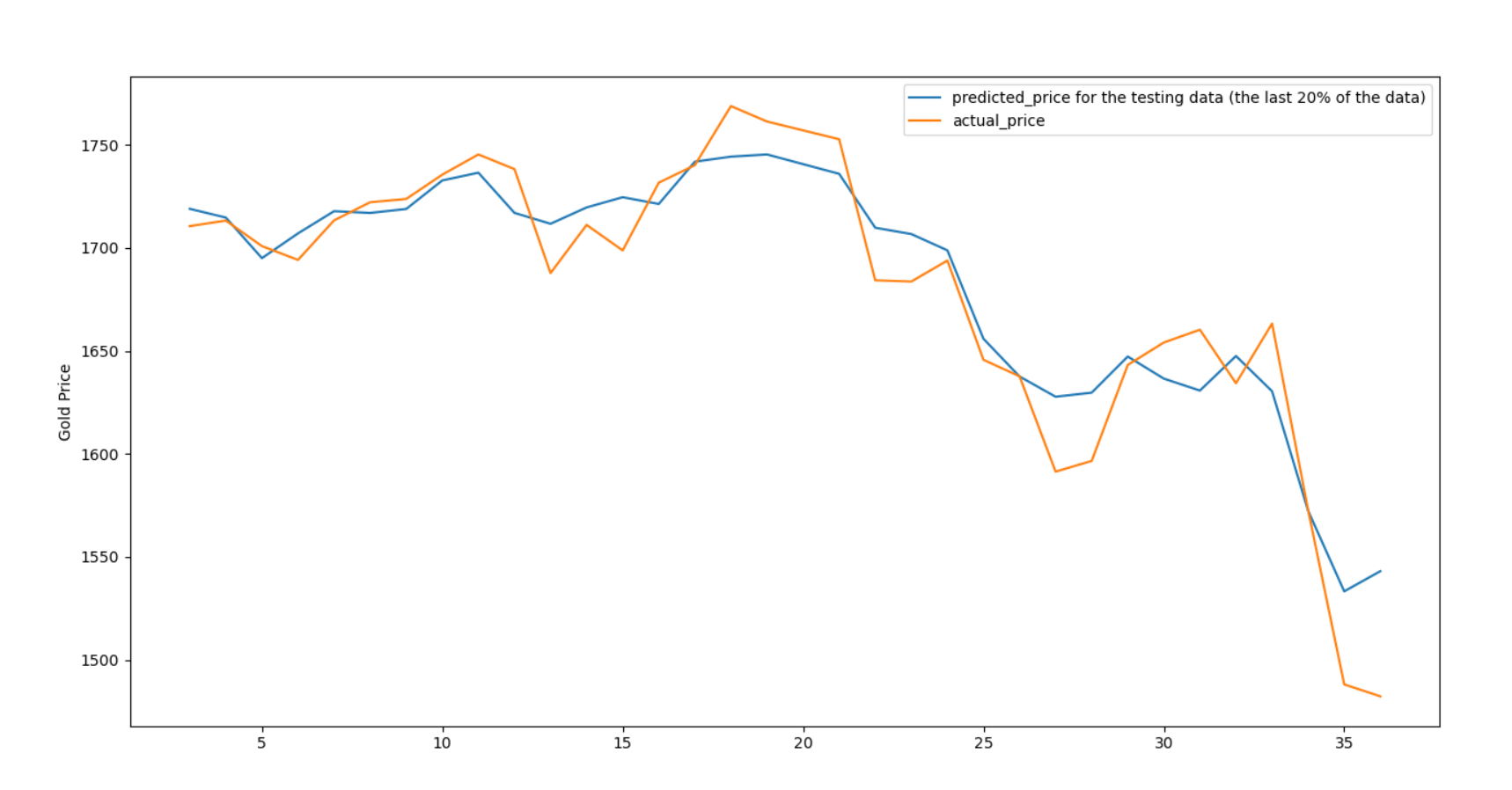
predicted\_price.plot(figsize=(26,8))

y\_test.plot()

plt.legend(['predicted\_price for the testing data (the last 20% of the data)','actual\_price'])

plt.ylabel("Gold Price")

plt.show()



# Calculate R square and rmse to check goodness of fit

r2\_score = linear.score(X\_test, y\_test)\*100

print("R square for regression", float("{0:.2f}".format(r2\_score)))

print("RMSE: ",sqrt(mean\_squared\_error(y\_test,predicted\_price)))



# Mean Absolute Percentage Error

MAPE = np.mean(np.abs((y\_test - predicted\_price['price']) / y\_test))\* 100

print('The Mean Absolute Percentage Error for the forecast is {:.2f}%'.format(MAPE))



**Milestone 5**

In this milestone, I am choosing to deploy in website (flask).

Create a \_\_init\_\_.py file first.

from flask import Flask, render\_template, request, redirect

import pandas as pd

from Milestone5 import getRegressionFunction

ALLOWED\_EXTENSIONS = {'csv'}

app = Flask(\_\_name\_\_)

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

@app.route("/", methods=['GET', 'POST'])

def home():

if request.method == 'GET':

return render\_template("home.html")

elif request.method == 'POST':

if 'file' not in request.files:

print('failed, no file in request')

return redirect(request.url)

file = request.files['file']

if file.filename == '':

print('failed, no file name')

return redirect(request.url)

if file and allowed\_file(file.filename):

print('file read')

dataFrame = pd.read\_csv(file)

newFunction = getRegressionFunction(dataFrame)

return render\_template("input.html", data = newFunction)

@app.route("/result", methods=['POST'])

def result():

if request.method == 'POST':

result = float(request.form.get('M1')) \* float(request.form.get('X1')) + float(request.form.get('M2')) \* float(request.form.get('X2')) + float(request.form.get('C'))

return render\_template("result.html", X1=request.form.get('X1'), X2=request.form.get('X2'), M1=request.form.get('M1') , M2=request.form.get('M2'), C=request.form.get('C'), result = result)

class DependentVariable:

def \_\_init\_\_(self, X1, X2):

self.X1 = X1

self.X2 = X2

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

So I created three template files (html), home.html, input.html, result.html.

Home.html as below:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Home</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/home.css') }}">

</head>

<body>

<h1>Hello, please upload your gold data file below:</h1>

<form method=post enctype=multipart/form-data>

<input type=file name=file>

<input type=submit value=Upload>

</form>

</body>

</html>

Input.html as below:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Input</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/input.css') }}">

</head>

<body>

<div>Gold price = {{data.M1}} \* 2 Days Moving Average + {{data.M2}} \* 5 Days Moving Average + {{data.C}} </div>

<h1>Hello, please enter your x1 and x2 value:</h1>

<form action="{{ url\_for('result') }}" method="POST">

<input value={{data.M1}} name="M1" hidden>

<input value={{data.M2}} name="M2" hidden>

<input value={{data.C}} name="C" hidden>

<label for="text1">X1</label>

<input type="text" name="X1" id="text1">

<label for="text2">X2</label>

<input type="text" name="X2" id="text2">

<input type="submit">

</form>

</body>

</html>

Result.html as below:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>Result</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/result.css') }}">

</head>

<body>

<h1>Result</h1>

<p class="paragraph">X1 = {{X1}}</p>

<p class="paragraph">X2 = {{X2}}</p>

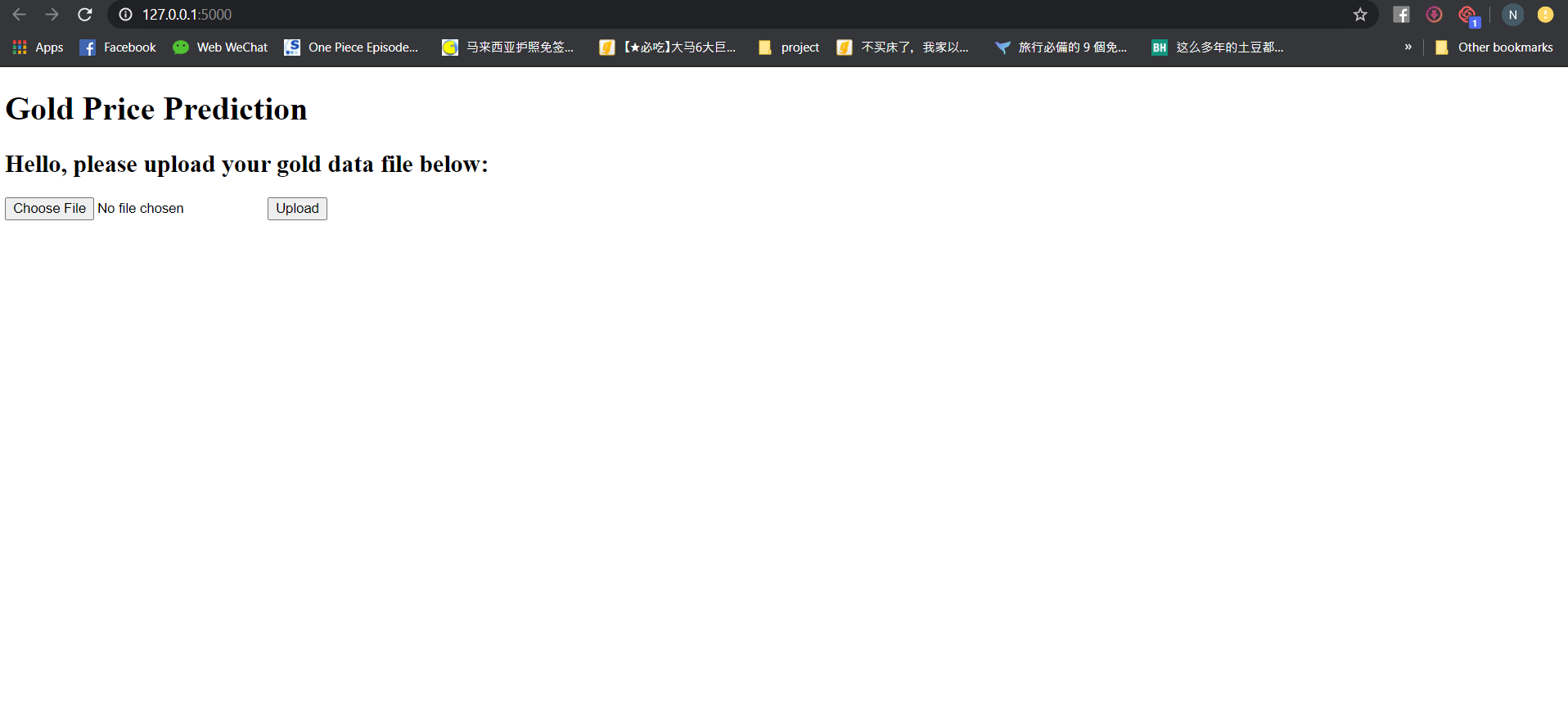
<p class="paragraph">Gold price = {{M1}} \* 2 Days Moving Average + {{M2}} \* 5 Days Moving Average + {{C}}</p>

<p class="paragraph">Gold price = {{result}}</p>

</body>

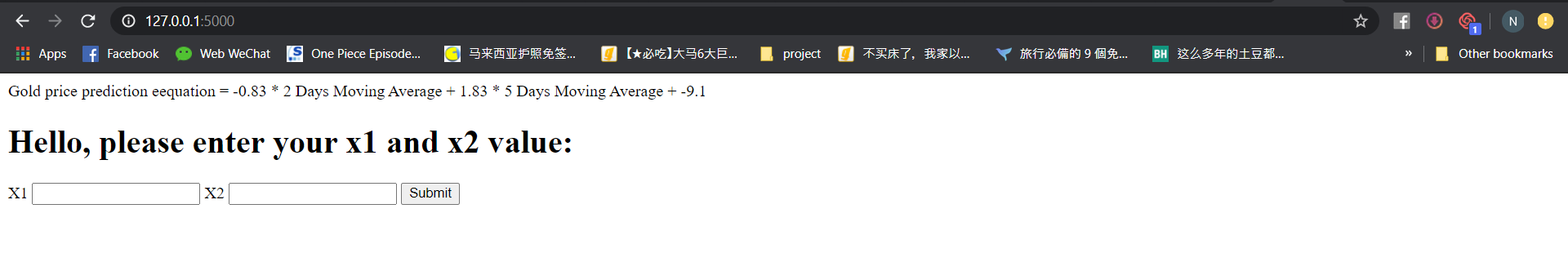
</html>

I try run it in local, it will show the home page like this:



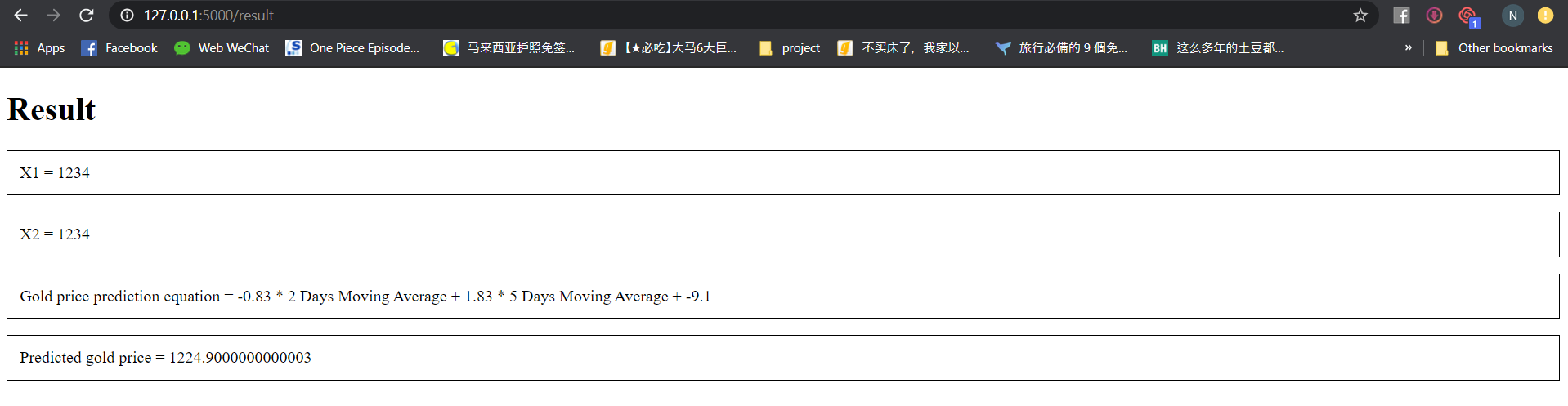
User can upload their own csv file to create their own gold price prediction equation.

After they upload, it will redirect to the input page as shown below:



User can key in the X1 and X2 value based on the day they want to know the gold price.

After they key in the X1 and X2 value, it will redirect to result page which showing the predicted gold price.



Another one deploys in app (Kivy).

First, we create trial.py file first.

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()

We must have *.kv* file too.

<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)

The gold price prediction equation will show up in the app, which called FirstApp.

