**-----------------PAI PROJECT----------------**

**22F-3142**

**22F-3088**

**22F-3101**

**CODE:**

import requests

import pandas as pd

import numpy as np

import json

import matplotlib

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d.axes3d import Axes3D

import mplfinance as mpl

api\_url = "https://api.coincap.io/v2/assets"

#this funtion is for crypto data class (for individual currency)

def fetch\_crypto\_details(crypto\_name):

crypto\_url = f"{api\_url}/{crypto\_name}"

response = requests.get(crypto\_url)

if response.status\_code == 200:

data = response.json()['data']

cryptoDic = {

'id': data['id'],

'symbol': data['symbol'],

'priceUsd': data['priceUsd'],

'volumeUsd24Hr': data['volumeUsd24Hr'],

'marketCapUsd': data['marketCapUsd']

}

cryptoDF = pd.DataFrame([cryptoDic])

return cryptoDF

else:

return None

#this function is to fetch historical data for individual currency

def historical\_data(crypto\_name):

history\_url = f"{api\_url}/{crypto\_name}/history?interval=d1"

history\_response = requests.get(history\_url)

if history\_response.status\_code == 200:

history\_data = history\_response.json()['data']

history\_dic = {

'priceUsd': [item['priceUsd'] for item in history\_data],

'date': [item['date'] for item in history\_data]

}

historyDF = pd.DataFrame(history\_dic)

return historyDF

else:

return None

#this funtion deals with individual currency

class CryptoData:

def \_\_init\_\_(self, crypto\_name):

self.crypto\_name = crypto\_name

self.cryptoDF = fetch\_crypto\_details(crypto\_name)

self.cryptoDF = self.cryptoDF.fillna(self.cryptoDF.mean(numeric\_only=True))

self.historyDF = historical\_data(crypto\_name)

def load\_from\_csv(self):

self.cryptoDF = pd.read\_csv('group9\_crypto')

self.historyDF = pd.read\_csv('group9\_history')

#to output the current price of any given crypto

def current\_crypto\_price(self):

if self.cryptoDF is not None:

crypto\_id = self.cryptoDF['id'].iloc[0]

if crypto\_id == "bitcoin":

crypto\_id = "Bitcoin(BTC)"

elif crypto\_id == "ethereum":

crypto\_id = "Ethereum(ETH)"

elif crypto\_id == "xrp":

crypto\_id = "Ripple(XRP)"

price\_usd = float(self.cryptoDF['priceUsd'].iloc[0])

print(f"{crypto\_id} Current Price: {price\_usd:.2f} with Market Cap: {float(self.cryptoDF['marketCapUsd'].iloc[0]):.2f} and 24HrVolume: {float(self.cryptoDF['volumeUsd24Hr'].iloc[0]):.2f}")

else:

print(f"Could not fetch data for {self.crypto\_name}")

#to calculate log returns

def calculateLog(self):

price = self.historyDF['priceUsd'].apply(float)

log = np.log(price/price.shift(1))

self.historyDF['log\_returns'] = log

self.historyDF['log\_returns'].fillna(0.0,inplace=True)

return self.historyDF['log\_returns']

def stats(self):

if not self.historyDF['log\_returns'].isnull().all():

arr=np.array(self.historyDF['log\_returns'])

print("Mean: ",np.mean(arr))

print("Standard Deviation: ",np.std(arr))

print("Median: ",np.median(arr))

else:

print("No Log Returns to Calculate Stats")

def save\_to\_csv(self):

self.cryptoDF.to\_csv('group9\_crypto')

self.historyDF.to\_csv('group9\_history')

def highest\_price(self):

if not self.historyDF.empty:

self.historyDF['priceUsd'] = pd.to\_numeric(self.historyDF['priceUsd'], errors = 'coerce')

max\_price\_row = self.historyDF.loc[self.historyDF['priceUsd'].idxmax()]

maxCryptoPrice = max\_price\_row['priceUsd']

date\_of\_max\_price = max\_price\_row['date']

date\_of\_max\_price = date\_of\_max\_price.split('T')[0]#to split after T and time

crypto\_name\_display = { "bitcoin": "Bitcoin(BTC)", "ethereum" : "Ethereum(ETH)", "xrp" : "Ripple(XRP)" }.get(self.crypto\_name, self.crypto\_name)

print(f"Highest Price of {crypto\_name\_display} was {maxCryptoPrice:.2f} on {date\_of\_max\_price}")

else:

print(f"Could not fetch data for {self.crypto\_name}")

def correlation\_matrix(self, other\_crypto):

# Calculate correlation matrix between returns of two cryptocurrencies

correlations = np.corrcoef(

np.array(self.historyDF['log\_returns']),

np.array(other\_crypto.calculateLog())

)

print("Correlation Matrix:\n", correlations)

**MAIN**

#main

eth = CryptoData("ethereum")

btc = CryptoData("bitcoin")

xrp = CryptoData("xrp")

flag = bool(True)

while flag:

print("PRESS 1 TO RUN NEW PROGRAM\nPRESS 2 TO LOAD FROM FILE")

main\_menu = int(input("INPUT CHOICE: "))

if main\_menu == 2:

btc.load\_from\_csv()

eth.load\_from\_csv()

xrp.load\_from\_csv()

elif main\_menu == 1 or main\_menu == 2:

print("PRESS 1 TO CALCULATE CURRENT PRICE")

print("PRESS 2 TO CALCULATE HIGHEST PRICE")

print("PRESS 3 TO GET CO-RELATION MATRIX")

print("PRESS 4 TO GET STATS")

print("PRESS 5 TO GET GRAPHS AND CHARTS")

print("PRESS 6 TO EXIT")

sub\_menu = int(input("INPUT CHOICE: "))

if sub\_menu == 1:

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

btc.current\_crypto\_price()

elif crypto == 2:

eth.current\_crypto\_price()

elif crypto == 3:

xrp.current\_crypto\_price()

else:

print("INCORRECT INPUT")

elif sub\_menu == 2:

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

btc.highest\_price()

elif crypto == 2:

eth.highest\_price()

elif crypto == 3:

xrp.highest\_price()

else:

print("INCORRECT INPUT")

elif sub\_menu == 3:

crypto = int(input("INPUT 1 FOR BTC/ETH \nINPUT 2 FOR BTC\XRP nINPUT 3 FOR XRP/ETH\nINPUT CHOICE: "))

if crypto == 1:

btc.correlation\_matrix(eth)

elif crypto == 2:

btc.correlation\_matrix(xrp)

elif crypto == 3:

xrp.correlation\_matrix(eth)

else:

print("INCORRECT INPUT")

elif sub\_menu == 4:

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

btc.stats()

elif crypto == 2:

eth.stats()

elif crypto == 3:

xrp.stats()

else:

print("INCORRECT INPUT")

elif sub\_menu == 5:

E=historical\_data("ethereum")

# convert data type object to is '%Y-%m-%d %H:%M:%S'

E["date"] = pd.to\_datetime(E["date"], format='%Y-%m-%d %H:%M:%S')

# Convert priceusd from object to float64 using astype()

E["priceUsd"] = E["priceUsd"].astype('float64')

B=historical\_data("bitcoin")

# convert data type object to is '%Y-%m-%d %H:%M:%S'

B["date"] = pd.to\_datetime(B["date"], format='%Y-%m-%d %H:%M:%S')

# Convert priceusd from object to float64 using astype()

B["priceUsd"] = B["priceUsd"].astype('float64')

X=historical\_data("xrp")

# convert data type object to is '%Y-%m-%d %H:%M:%S'

X["date"] = pd.to\_datetime(X["date"], format='%Y-%m-%d %H:%M:%S')

# Convert priceusd from object to float64 using astype()

X["priceUsd"] = X["priceUsd"].astype('float64')

print("PRESS 1 FOR LINE CHART\nPRESS 2 FOR BAR GRAPH\nPRESS 3 FOR 3D CHARTS\nPRESS 4 FOR SCATTER AND STEP CHARTS\nPRESS 5 FOR CANDLE CHARTS")

chart = int(input("INPUT CHOICE: "))

if chart == 1:

#line charts

def myLineCharts(E,title,colr):

fig,axes=plt.subplots(1,1,figsize=(13,4))

axes.plot(E['date'], E['priceUsd'],colr)

axes.set\_title(f'Price vs Date ({title})')

axes.set\_xlabel('Date')

axes.set\_ylabel('Price (USD)')

axes.grid(True)

axes.tick\_params(axis='x', rotation=45)

fig.tight\_layout()

plt.show()

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

myLineCharts(B,"Bitcoin(BTC)",'g')

elif crypto == 2:

myLineCharts(E,"Ethereum(ETH)",'r')

elif crypto == 3:

myLineCharts(X,"Ripple(XRP)",'b')

else:

print("INCORRECT INPUT")

elif chart == 2:

def myBarChart(E,B,X):

mean\_return\_E = E["priceUsd"].mean()

mean\_return\_B = B["priceUsd"].mean()

mean\_return\_X = X["priceUsd"].mean()

mean\_returns = [mean\_return\_E, mean\_return\_B, mean\_return\_X]

cryptocurrencies = ['Etherium', 'Bitcoin', 'XRP']

plt.figure(figsize=(6, 4))

bars = plt.bar(cryptocurrencies, mean\_returns, color=['red', 'green', 'blue'])

# Adding labels to the bars

for bar, ret in zip(bars, mean\_returns): # Changed 'bar' to 'bars' for iteration

plt.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() - 0.05,

f'{ret:.2f}', ha='center', va='bottom', color='black', fontweight='bold')

plt.xlabel('Cryptocurrencies')

plt.ylabel('Mean Returns')

plt.title('Mean Returns of Cryptocurrencies')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

myBarChart(E,B,X)

elif chart == 3:

#3d chart

def my3DCharts(E,B,X):

fig=plt.figure(figsize=(14,4))

ax=fig.add\_subplot(1,3,1,projection='3d')

Z= X["priceUsd"]

Z2=Z.values.reshape(-1, 1)

p=ax.plot\_wireframe(E["priceUsd"],B["priceUsd"],Z2,rstride=4,cstride=4,linewidth=3,alpha=0.5)

ax.set\_xlabel('Etherim')

ax.set\_ylabel('Bitcoin')

ax.set\_zlabel('XRP')

ax.set\_xlim([E["priceUsd"].min(),E["priceUsd"].max()])

ax.set\_ylim([B["priceUsd"].min(),B["priceUsd"].max()])

ax.set\_zlim([X["priceUsd"].min(),X["priceUsd"].max()])

fig.colorbar(p,ax=ax,shrink=0.7,location='left',pad=0.0)

fig.tight\_layout()

ax=fig.add\_subplot(1,3,2,projection='3d')

Z= X["priceUsd"]

Z2=Z.values.reshape(-1, 1)

p=ax.plot\_surface(E["priceUsd"],B["priceUsd"],Z2,rstride=4,cstride=4,cmap=matplotlib.cm.coolwarm,linewidth=3,alpha=0.8)

ax.set\_xlabel('Etherim')

ax.set\_ylabel('Bitcoin')

ax.set\_zlabel('XRP')

ax.set\_xlim([E["priceUsd"].min(),E["priceUsd"].max()])

ax.set\_ylim([B["priceUsd"].min(),B["priceUsd"].max()])

ax.set\_zlim([X["priceUsd"].min(),X["priceUsd"].max()])

fig.colorbar(p,ax=ax,shrink=0.6,location='left',pad=0.0)

fig.tight\_layout()

ax=fig.add\_subplot(1,3,3,projection='3d')

Z= X["priceUsd"]

Z2=Z.values.reshape(-1, 1)

ax.plot\_surface(E["priceUsd"],B["priceUsd"],Z2,rstride=4,cstride=4,cmap=matplotlib.cm.RdBu,linewidth=3,alpha=0.8)

ax.set\_xlabel('Etherim')

ax.set\_ylabel('Bitcoin')

ax.set\_zlabel('XRP')

ax.set\_xlim([E["priceUsd"].min(),E["priceUsd"].max()])

ax.set\_ylim([B["priceUsd"].min(),B["priceUsd"].max()])

ax.set\_zlim([X["priceUsd"].min(),X["priceUsd"].max()])

fig.colorbar(p,ax=ax,shrink=0.5,location='left',pad=0.0)

fig.tight\_layout()

plt.show()

my3DCharts(E,B,X)

elif chart == 4:

def myChartsall(E,title,colr):

fig,ax=plt.subplots(1,3,figsize=(13,4))

ax[0].scatter(E["priceUsd"],E["date"],color=colr,ls="-",lw=1.5)

ax[0].set\_title(f"{title} (Scatter)")

ax[0].set\_ylabel('Date')

ax[0].set\_xlabel('Price (USD)')

ax[0].grid(True)

ax[0].tick\_params(axis='x', rotation=45)

ax[1].step(E["priceUsd"],E["date"],color=colr,ls="-",lw=1.5)

ax[1].set\_title(f"{title} (Step)")

ax[1].set\_ylabel('Date')

ax[1].set\_xlabel('Price (USD)')

ax[1].grid(True)

ax[1].tick\_params(axis='x', rotation=45)

ax[2].bar(E["priceUsd"],E["date"],color=colr,ls="-",lw=1.5)

ax[2].set\_title(f"{title} (Bar)")

ax[2].set\_ylabel('Date')

ax[2].set\_xlabel('Price (USD)')

ax[2].grid(True)

ax[2].tick\_params(axis='x', rotation=45)

plt.tight\_layout()

plt.show()

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

myChartsall(B,"Bitcoin(BTC)","green")

elif crypto == 2:

myChartsall(E,"Ethereum(ETH)","red")

elif crypto == 3:

myChartsall(X,"Ripple(XRP)","blue")

else:

print("INCORRECT INPUT")

elif chart == 5:

def get\_candles(crypto\_name):

start\_date = "2023-06-06"

end\_date = "2023-10-05"

freq = "1DAY"

coin = crypto\_name

url = f'https://rest.coinapi.io/v1/exchangerate/{coin}/USD/history?period\_id={freq}&time\_start={start\_date}T00:00:00&time\_end={end\_date}T00:00:00'

headers = {'X-CoinAPI-Key' : '88B335C4-422D-44B8-8E6B-F9342E490DD2'}

response = requests.get(url, headers=headers)

content = json.loads(response.text)

df = pd.json\_normalize(content)

df.time\_period\_start = pd.to\_datetime(df.time\_period\_start)

df = df.set\_index("time\_period\_start")

df.drop(['time\_period\_end', "time\_open", "time\_close"], axis=1, inplace=True)

df.rename(columns={"rate\_open": "Open", "rate\_high":"High", "rate\_low":"Low", "rate\_close": "Close"}, inplace=True)

mpl.plot(df,type="candle", mav =(3,6,9),title = f"{coin} Price", style="yahoo",figsize=(13, 4))

plt.tight\_layout()

plt.show()

crypto = int(input("INPUT 1 FOR BTC\nINPUT 2 FOR ETH\nINPUT 3 FOR XRP\nINPUT CHOICE: "))

if crypto == 1:

print(get\_candles('BTC'))

elif crypto == 2:

print(get\_candles('XRP'))

elif crypto == 3:

print(get\_candles('ETH'))

else:

print("INCORRECT INPUT")

else:

print("WRONG INPUT!")

elif sub\_menu == 6:

print("DO YOU WANT TO SAVE YOUR PROGRESS ?\nYES: 1\nNO: 2")

choice = int(input("INPUT CHOICE: "))

if choice == 1:

btc.save\_to\_csv()

eth.save\_to\_csv()

xrp.save\_to\_csv()

elif choice == 1 or choice == 2:

print("THANK YOU FOR USING OUR CRYPTO SYSTEM!!")

print("MADY BY HUMANS ON EARTH :)")

print("EXITING.....")

flag = False

break

else:

print("WRONG INPUT!")

**------------------DOCUMENTATION------------------**

**LIBRARIES**

* **import requests**

this library is used for getting https data

* **import pandas as pd**

this library is used for arranging data in a data frame.

* **import numpy as np**

this library is used to perform mathematical operations on arrays

* **import json**

this library is used to extract/fetch data from json

* **import matplotlib**
* **import matplotlib.pyplot as plt**

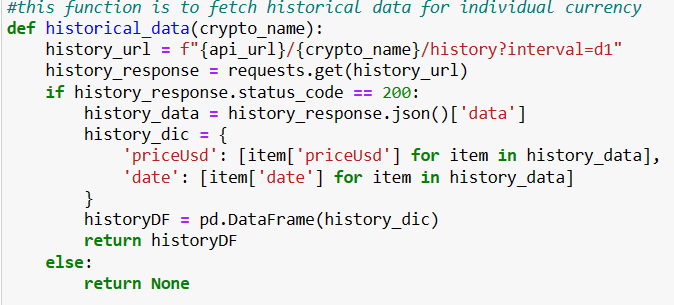
this library is used for visual representation

* **from mpl\_toolkits.mplot3d.axes3d import Axes3D**
* **import mplfinance as mpl**

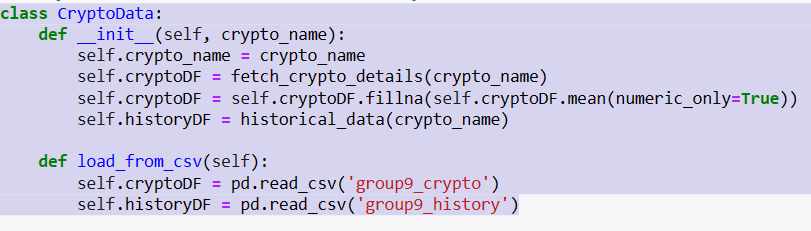
this library to make financial charting of pandas data frames simpler.

****

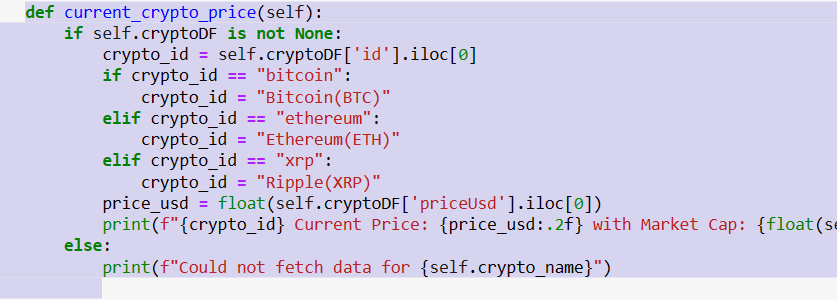
We are given with URL of the site. We have to fetch the data from that URL. We are using method (fetch\_crypto\_details) to fetch the data like id, symbol, price, volume, market capital. If the status code is equal to 200 then we can fetch data from the URL and display the data in a data frame using pandas library.



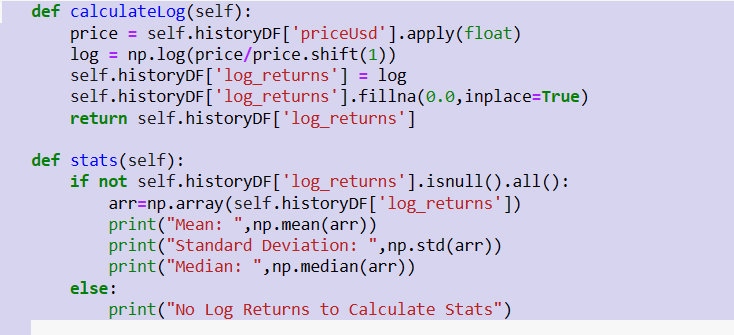
The function(historical\_data) is used to get the past data of any cryptocurrency like the price of bitcoin and the date at which the price was noted. And also display that data in a data frame using pandas library. If the status code of that URL is 200 then we can get the historical data otherwise we can’t get the historical data.



Here the class named “CryptoData” provides the details of crypto data and the historical data of crypto. Class consists of a function named “load\_from\_csv” which can read the read from the comma separated files, one is crypto file and other is history file. Crypto file contains the detail of current crypto details and history file contains the past crypto details.



There is a function named “current\_crypto\_price” which can fetch data of crypto currency with a current price and with current market capital and displays the current details of any given crypto.



This function “calculateLog” is the instance of “cryptoData” class and this function is used to calculate the log of price of the past crypto data. And there is another function named “stats” instance of “cryptoData” class which can calculate the statistics which include **Mean, Median and Standard Deviation** of the past data of the crypto.



After finding statistics of past data of the crypto we can save it to the csv file. After that we find the highest price of the past data of the crypto data. Function named “highest\_price” displays the highest price of the crypto currency and that date when it was noted. And then find correlation matrix between the history data and the log of the crypto currency.

**Main:**

In the main part of the program there is a menu showing various options for the user to choose from. When we press 1 it can display the current crypto price and when we press 2 it can display the highest price of crypto currency. When we press 3 it can display the correlation matrix between the crypto currencies. When we press 4 it can display the statistics of the data. When we press 5 it can display the graph of the different crypto currencies. When we press 6 we can select whether we want to do save or don’t save the data.