**SOURCE CODE**

**#Closing Price Of Various Cryptocurrencies**

for coin in crypto\_data:

df = pd.DataFrame(crypto\_data[coin])

df = df[['Date' , 'Close']]

df['Date\_mpl'] = df['Date'].apply(lambda x: mdates.date2num(x)) # making new column 'Date\_mpl' by using date2num lamba function

fig, ax = plt.subplots(figsize=(6,4))

sns.tsplot(df.Close.values, time=df.Date\_mpl.values, alpha=0.8, color=color[3], ax=ax)

ax.xaxis.set\_major\_locator(mdates.AutoDateLocator())

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y.%m.%d'))

fig.autofmt\_xdate()

plt.xlabel('Date', fontsize=12)

plt.ylabel('Price in USD', fontsize=12)

title\_str = "Closing price distribution of " + coin

plt.title(title\_str, fontsize=15)

plt.show()

**#Analysing Of Candlestick Chart For Various Cryptocurrency**

for coin in crypto\_data:

df = pd.DataFrame(crypto\_data[coin])

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

ax1 = plt.subplot2grid((1,1), (0,0))

df['Date\_mpl'] = df['Date'].apply(lambda x: mdates.date2num(x))

temp\_df = df[df['Date']>'2017-05-01']

ohlc = []

for ind, row in temp\_df.iterrows():

ol=[row['Date\_mpl'],row['Open'], row['High'], row['Low'], row['Close'], row['Volume']]

ohlc.append(ol)

candlestick\_ohlc(ax1, ohlc, width=0.4, colorup='#77d879', colordown='#db3f3f')

ax1.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))

ax1.xaxis.set\_major\_locator(mticker.MaxNLocator(10))

plt.xlabel("Date", fontsize=12)

plt.xticks(rotation='vertical')

plt.ylabel("Price in USD", fontsize=12 )

title\_str = "Candlestick chart for " + coin

plt.title(title\_str, fontsize=15)

plt.subplots\_adjust(left=0.09, bottom=0.20, right=0.94, top=0.90, wspace=0.2, hspace=0)

plt.show()

#**Correlation Map For Various Cryptocurrency**

**#Spearman Correlation Map**

df = pd.DataFrame()

currency\_name = []

df['Date'] = crypto\_data['bitcoin'].Date

df = df[df['Date']>'2017-05-01']

for coin in crypto\_data:

currency\_name.append(coin)

temp\_df = crypto\_data[coin]

df[coin] = temp\_df[temp\_df['Date']>'2017-05-01'].Close

temp\_df = df[currency\_name]

corrmat = temp\_df.corr(method='spearman')

fig, ax = plt.subplots(figsize=(5, 5))

sns.heatmap(corrmat, vmax=1., square=True)

plt.title("Spearman correlation map", fontsize=15)

plt.show()

temp\_df.corr(method='spearman')

#**Pearson Correlation Map**

df = pd.DataFrame()

currency\_name = []

df['Date'] = crypto\_data['bitcoin'].Date

df = df[df['Date']>'2017-05-01']

for coin in crypto\_data:

currency\_name.append(coin)

temp\_df = crypto\_data[coin]

df[coin] = temp\_df[temp\_df['Date']>'2017-05-01'].Close

temp\_df = df[currency\_name]

corrmat = temp\_df.corr(method='pearson')

fig, ax = plt.subplots(figsize=(5, 5))

sns.heatmap(corrmat, vmax=1., square=True)

plt.title("Pearson correlation map", fontsize=15)

plt.show()

temp\_df.corr(method='pearson')

#**Kendall Correlation Map**

df = pd.DataFrame()

currency\_name = []

df['Date'] = crypto\_data['bitcoin'].Date

df = df[df['Date']>'2017-05-01']

for coin in crypto\_data:

currency\_name.append(coin)

temp\_df = crypto\_data[coin]

df[coin] = temp\_df[temp\_df['Date']>'2017-05-01'].Close

temp\_df = df[currency\_name]

corrmat = temp\_df.corr(method='kendall')

fig, ax = plt.subplots(figsize=(5, 5))

sns.heatmap(corrmat, vmax=1., square=True)

#**Time Series Forecasting using fbprophet()**

for coin in crypto\_data:

df = pd.DataFrame(crypto\_data[coin])

temp\_df = pd.DataFrame()

temp\_df['ds'] = df['Date']

temp\_df['y'] = df['Close']

temp\_df['ds'] = temp\_df['ds'].dt.to\_pydatetime()

model = Prophet()

model.fit(temp\_df)

future = model.make\_future\_dataframe(periods = 60)

forecast = model.predict(future)

title\_str = "predicted value of "+ coin

model.plot(forecast, uncertainty=False)

model.plot\_components(forecast, uncertainty=False)

#**Time series Forecasting using ARIMA model**

# fit model

model = ARIMA(df\_bitcoin, order=(5,1,0))

model\_fit = model.fit(disp=0)

print(model\_fit.summary())

# plot residual errors

residuals = DataFrame(model\_fit.resid)

residuals.plot()

plt.show()

residuals.plot(kind='kde')

plt.show()

print(residuals.describe())

X = df\_bitcoin.values

size = int(len(X) \* 0.80)

train, test = X[0:size], X[size:len(X)]

history = [x for x in train]

predictions = list()

for t in range(len(test)):

model = ARIMA(history, order=(5,1,0))

model\_fit = model.fit(disp=0)

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = test[t]

history.append(obs)

error = mean\_squared\_error(test, predictions)

RMSE=error\*\*0.5

print("Root means sqauare error is ",RMSE)

#calcualting error

ar=test-predictions

np.absolute(ar)

ar=ar/test

ar=ar\*100

forecast\_error=(sum(ar))/(len(ar))

print("forecast error is ")

print (forecast\_error)

arr=[100]

print ("forecast accuracy is")

print(arr-forecast\_error)

#calculation of error done

print('Test MSE: %f' % error)

print('Test RMSE: %f' % RMSE)

# plot

plt.plot(test)

plt.plot(predictions, color='red')

plt.show()

**#Future Graphs Of Fifteen Different Cryptocurrencies**

forecast.to\_csv('numeraire\_forecast.csv')

import matplotlib.dates as mdates

#matplotlib.dates.datestr2num

df = pd.read\_csv('./numeraire\_forecast.csv')

#plt.plot(data.yhat, data.ds)

import matplotlib.dates as mdates

df['Date\_mpl'] = df['ds'].apply(lambda x: mdates.datestr2num(x))

fig, ax = plt.subplots(figsize=(12,8))

ax.xaxis.set\_major\_locator(mdates.AutoDateLocator())

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y.%m.%d'))

fig.autofmt\_xdate()

plt.xlabel('Date', fontsize=12)

plt.ylabel('Price in USD', fontsize=12)

plt.title("Future Closing Price For Numeraire ", fontsize=15)

plt.show()

**#ChatBot Implementation**

# sets up service wrapper, sends initial message, and

# receives response.

import watson\_developer\_cloud

conversation = watson\_developer\_cloud.ConversationV1(

username = '531b64f4-1023-4a46-b81a-f289c6e334b4', # replace with username from service key

password = 'EeL8NB8f2wZR', # replace with password from service key

version = '2017-05-26')

workspace\_id = '3b77dedf-de9f-41b4-a46d-45a9706ed976' # replace with workspace ID

# Start conversation with empty message.

response = conversation.message(

workspace\_id = workspace\_id,

input = {

'text': ''})

# Print the output from dialog, if any.

if response['output']['text']:

print(response['output']['text'][0])

# maintains state.

import watson\_developer\_cloud

# Set up Conversation service.

conversation = watson\_developer\_cloud.ConversationV1(

username = '531b64f4-1023-4a46-b81a-f289c6e334b4',

# replace with username from service key

password = 'EeL8NB8f2wZR', # replace with password from service key

version = '2017-05-26')

workspace\_id = '3b77dedf-de9f-41b4-a46d-45a9706ed976' # replace with workspace ID

# Initialize with empty value to start the conversation.

user\_input = ''

context = {}

# Main input/output loop

while True:

# Send message to Conversation service.

response = conversation.message(

workspace\_id = workspace\_id,

input = {

'text': user\_input},

context = context)

# If an intent was detected, print it to the console.

if response['intents']:

print('Detected intent: #' + response['intents'][0]['intent'])