

Configuration Manual

MSc Research Project
Data Analytics

Nisarg Shah
Student ID: x18137415

School of Computing
National College of Ireland

Supervisor : Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Nisarg Shah
Student ID:	x18137415
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2018
Project Title:	Configuration Manual
Word Count:	1288
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	28th January 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	



Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

Submission Author	Nisarg Shah
Turnitin Paper ID (Ref. ID)	1232552843
Submission Title	x18137415_Config_Manual
Assignment Title	Submit Final Version - CONFIGURATION Manual (PDF file)
Submission Date	11/12/19, 20:23

Configuration Manual

Nisarg Shah
x18137415

1 Introduction

This configuration manual will help you understand the software and hardware requirements that will help to re-create the project. This manual will help to implement the research project "Predicting Terrorism Attacks with Bitcoin Prediction using Time Series Analysis."

2 System Requirements

2.1 Hardware

Intel(R) Core(TM) i7 - 4150U CPU @ 2.00 Ghz; AMD Radeon R5 M230 2GB ; 12 GB RAM; 1TB HDD; Windows 8.1 64 bit.

2.2 Software

- Rstudio: Loading the data and feature selection of data and filtering the data and making tree maps for initial data visualization.
- Jupyter: Running various models on Bitcoin (ARIMA, LSTM, RNN) and Terrorism (ARIMA) on a IDE of Python that is accessed using a browser.
- Tableau: For the final visualization of data of Bitcoin and Terrorism.

3 Project Development

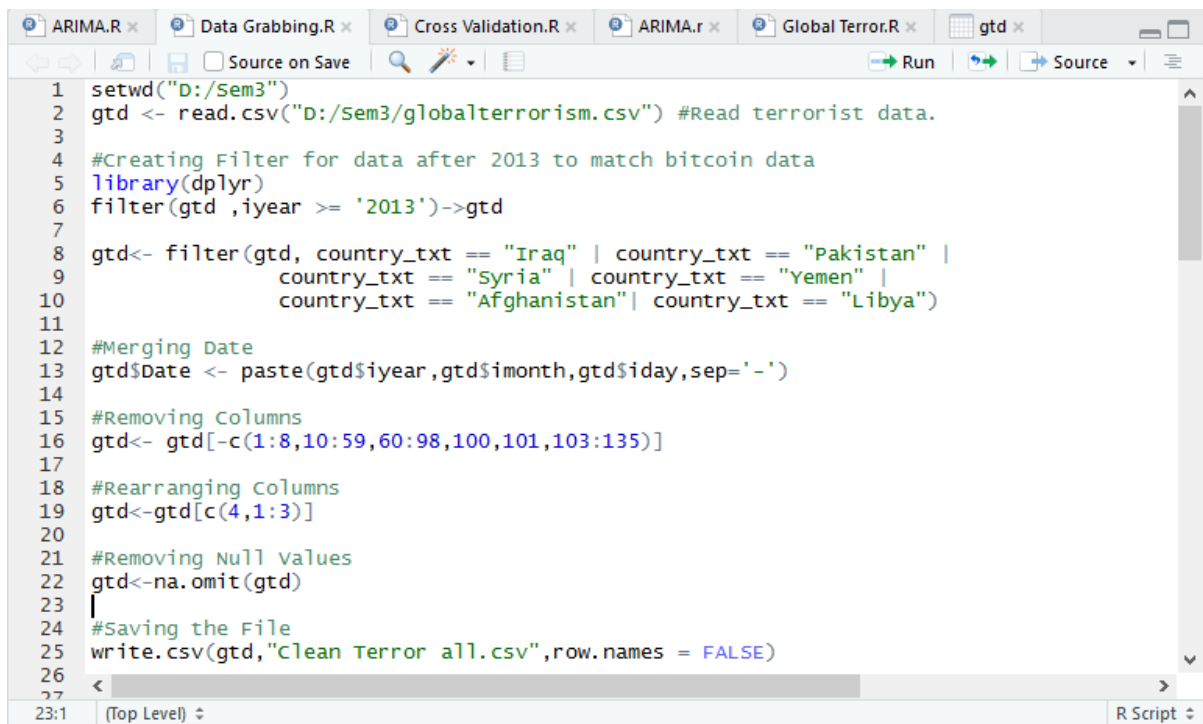
3.1 Dataset

Two different datasets were taken for this research. The first one being Bitcoin Historical data from kaggle Zielak (2019) and Terrorism data is also taken from kaggle START (2017).

Bitcoin data has more than 2,49,000 rows and 8 columns. Terror data has more than 1,80,000 rows and 135 columns.

3.2 Data Preparation

Initial Preparation is done by both R and Python. Terrorism data is downloaded from Kaggle and then loaded into Rstudio and then data is filtered out to some specific countries where we expect to get some results. Date is merged from day month and year to a single column of date, this is further filtered to match the year of Bitcoin Data i.e 2013. After that we do some initial analysis by printing some tree maps to show number of people killed in various countries and number of people killed year wise. Preda (2019) Data is then saved into CSV format and then Jupyter notebook is used for further analysis. The Figure 1 shows the filtering of data and Figure 2 shows the tree map printing code.

The image shows a screenshot of the RStudio interface. The top pane displays several open R scripts: ARIMA.R, Data Grabbing.R, Cross Validation.R, ARIMA.r, Global Terror.R, and gtd. The main editor pane shows the following R code:

```
1 setwd("D:/Sem3")
2 gtd <- read.csv("D:/Sem3/globalterrorism.csv") #Read terrorist data.
3
4 #Creating Filter for data after 2013 to match bitcoin data
5 library(dplyr)
6 filter(gtd ,iyear >= '2013')->gtd
7
8 gtd<- filter(gtd, country_txt == "Iraq" | country_txt == "Pakistan" |
9             country_txt == "Syria" | country_txt == "Yemen" |
10            country_txt == "Afghanistan"| country_txt == "Libya")
11
12 #Merging Date
13 gtd$Date <- paste(gtd$iyear,gtd$imonth,gtd$iday,sep='-')
14
15 #Removing Columns
16 gtd<- gtd[-c(1:8,10:59,60:98,100,101,103:135)]
17
18 #Rearranging Columns
19 gtd<-gtd[c(4,1:3)]
20
21 #Removing Null values
22 gtd<-na.omit(gtd)
23
24 #Saving the File
25 write.csv(gtd,"Clean Terror all.csv",row.names = FALSE)
26
27
```

The bottom status bar shows the cursor is at line 23, column 1, and the file is named "R Script".

Figure 1: Data Cleaning using RStudio

3.3 Bitcoin ARIMA prediction

Further on Bitcoin coin data is downloaded from kaggle again and loaded onto Jupyter Notebook to pre-process data and do further analysis on it. As seen below in Figure 3 the libraries such as numpy, pandas, scipy, matplotlib and other are used and the code guidance is taken from Aptem (2017).

```

ARIMA.R x Data Grabbing.R x Cross Validation.R x ARIMA.r x Global Terror.R x gtd x
Source on Save Run Source
28 #Printing Tree Maps
29 library(treemap)
30 gtd %>% filter(nkill > 0) -> gtdk
31 treemap(gtdk,
32         index=c("iyear"),
33         vsize = "nkill",
34         palette = "Reds",
35         title="Killings in countries with ISIS",
36         fontsize.title = 14
37 )
38
39 gtd %>% filter(nwound > 0) -> gtdw
40 treemap(gtdw,
41         index=c("iyear"),
42         vsize = "nwound",
43         palette = "Blues",
44         title="wounded in countries with ISIS",
45         fontsize.title = 14
46 )
47
48 treemap(gtdk,
49         index=c("country_txt", "iyear"),
50         type = "value",
51         vsize = "nkill",
52         vcolor="nwound",
53         palette = "Greens",
54         title="Killings in Global terrorism (Countries/Years) - size is proportional with the num",
55         title.legend = "Number of wounded",
56         fontsize.title = 10
57 )
58
59
48:15 (Top Level) R Script

```

Figure 2: Tree Map Printing using RStudio

```

# Import libraries
import numpy as np #array operations and for Linear Algebra
import pandas as pd #used for reading CSV files.
import matplotlib.pyplot as plt # For Data plotting
from scipy import stats #for scientific Computing
import statsmodels.api as sm #for statistical data tests and statistical data exploration
import warnings
from itertools import product #Used for creating nested for loops
from datetime import datetime #for date and time functions
from sklearn import metrics #for mean errors and other error functions
warnings.filterwarnings('ignore')
plt.style.use('seaborn-poster')

# Load data
df = pd.read_csv('Documents\Bitcoin\BTCUSD.csv')
df.head()

```

Figure 3: Bitcoin Arima Libraries and Data Loading using Jupyter

After this the data is re sampled into daily and weekly format for easier analysis as seen in Figure 4.

```

# Unix-time to date format
df.Timestamp = pd.to_datetime(df.Timestamp, unit='s')

# Resampling to daily frequency
df.index = df.Timestamp
df = df.resample('D').mean()

# Resampling to weekly frequency
df_week = df.resample('W').mean()

# Resampling to monthly frequency
df_month = df.resample('M').mean()

# Resampling to annual frequency
df_year = df.resample('A-DEC').mean()

# Resampling to quarterly frequency
df_q = df.resample('Q-DEC').mean()
df_week.head()

fig = plt.figure(figsize=[15, 7])
plt.suptitle('Bitcoin exchanges, mean USD', fontsize=22)

plt.subplot(221)
plt.plot(df.Weighted_Price, '--', label='By Days')
plt.legend()

plt.subplot(221)
plt.plot(df_week.Weighted_Price, '--', label='By Weeks')
plt.legend()

plt.subplot(222)
plt.plot(df_month.Weighted_Price, '--', label='By Months')
plt.legend()

plt.subplot(223)
plt.plot(df_q.Weighted_Price, '--', label='By Quarters')
plt.legend()

plt.subplot(224)
plt.plot(df_year.Weighted_Price, '--', label='By Years')
plt.legend()

plt.show()

```

Figure 4: Bitcoin Data Re sampling and Plotting using Jupyter

This helps us understand how the data looks in different re sampled format. After that we do the seasonality test to check the Observed, Trend, seasonality, Residuals. All these are plotted. The Dickey Fuller test is conducted, If the values are greater than 0. That implies that the data is non stationary. Hence transformations like Box Cox transformations along with Seasonal and Regular Differentiation is done. Auto and Partial correlation is then plotted to see if the seasonality is removed or not. The code for the same is seen in Figure 5

```

plt.figure(figsize=[15,7])
sm.tsa.seasonal_decompose(df_week.Weighted_Price).plot()
print("Dickey-Fuller test: p=%f" % sm.tsa.stattools.adfuller(df_week.Weighted_Price)[1])
plt.show()

Dickey-Fuller test: p=0.821949

# Box-Cox Transformations
df_week['Weighted_Price_box'], lambda = stats.boxcox(df_week.Weighted_Price)
print("Dickey-Fuller test: p=%f" % sm.tsa.stattools.adfuller(df_week.Weighted_Price)[1])

Dickey-Fuller test: p=0.821949

# Seasonal differentiation
df_week['prices_box_diff'] = df_week.Weighted_Price_box - df_week.Weighted_Price_box.shift(12)
print("Dickey-Fuller test: p=%f" % sm.tsa.stattools.adfuller(df_week.prices_box_diff[12:])[1])

Dickey-Fuller test: p=0.010333

# Regular differentiation
df_week['prices_box_diff2'] = df_week.prices_box_diff - df_week.prices_box_diff.shift(1)
plt.figure(figsize=(15,7))

# STL-decomposition
sm.tsa.seasonal_decompose(df_week.prices_box_diff2[13:]).plot()
print("Dickey-Fuller test: p=%f" % sm.tsa.stattools.adfuller(df_week.prices_box_diff2[13:])[1])

plt.show()

Dickey-Fuller test: p=0.000000

```

Figure 5: Transformations and Differentiation using Jupyter

After that Model selection is run, where variations are run by the system in a loop to find the optimal P(Seasonal Autoregressive Order) ,D (Seasonal Difference Order) ,Q(Seasonal Moving Average Order) ,m (The number of times steps for a single seasonal period) values. This helps us to find the best model that would be for the type of data that we are providing it with. The code can be seen below in Figure 6

```

Qs = range(0, 2)
qs = range(0, 3)
Ps = range(0, 3)
ps = range(0, 3)
D=1
d=1
parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
len(parameters_list)

# Model Selection
results = []
best_aic = float("inf")
warnings.filterwarnings('ignore')
for param in parameters_list:
    try:
        model=sm.tsa.statespace.SARIMAX(df_week.Weighted_Price_box, order=(param[0], d, param[1]),
                                        seasonal_order=(param[2], D, param[3], 12)).fit(dispatch=1)

    except ValueError:
        print('wrong parameters:', param)
        continue
    aic = model.aic #Akaike information criterion
    if aic < best_aic:
        best_model = model
        best_aic = aic
        best_param = param
    results.append([param, model.aic])

# Best Models
result_table = pd.DataFrame(results)
result_table.columns = ['parameters', 'aic']
print(result_table.sort_values(by = 'aic', ascending=True).head())
print(best_model.summary())

```

Figure 6: Finding best model to run ARIMA using Jupyter

After that we confirm the residuals plot and auto correlation plot to confirm which should show Dickey Fuller test as $p = 0.00$, indicating that the data is stationary as seen in Figure 7

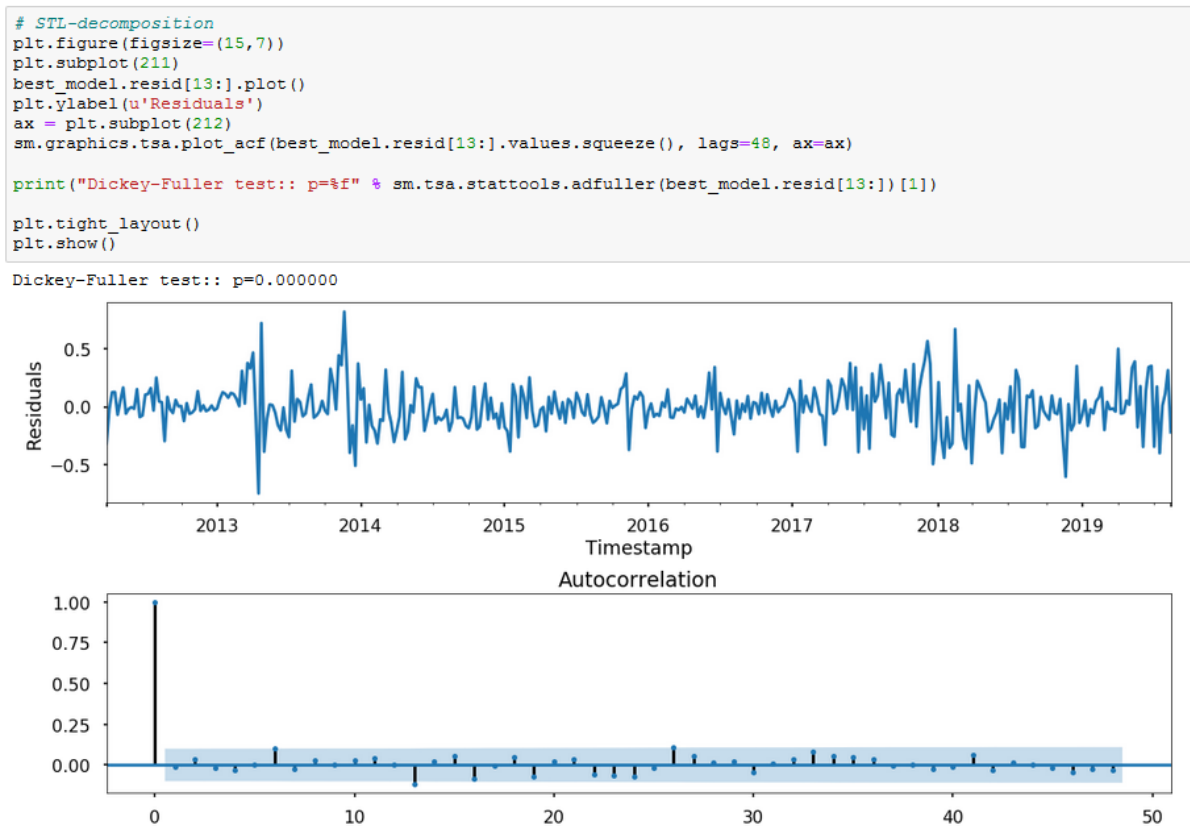


Figure 7: Checking Autocorrelation plot for seasonality using Jupyter

After that we run the prediction model with the 'datetime' function and input the date values we are looking for a prediction of. The whole data is put to find the prediction and another line is plotted along side the original values to show the predicted values. This is plotted below in the Figure 8 After that we find the error percentage to find the actual difference between the original values and the predicted values to show how best our model works.

```

# Prediction
df_week2 = df_week[['Weighted_Price']]
date_list = [datetime(2019, 8, 25), datetime(2019, 9,1), datetime(2019, 9,8), datetime(2019, 9, 15),
             datetime(2019, 9, 22),datetime(2019, 9, 29), datetime(2019, 10, 6), datetime(2019, 10, 13),
             datetime(2019, 10, 20), datetime(2019,10,27), datetime(2019,11,3), datetime(2019,11,10),
             datetime(2019,11,17), datetime(2019,11,24), datetime(2019,12,1), datetime(2019,12,8),
             datetime(2019,12,15), datetime(2019,12,22), datetime(2019,12,29),datetime(2020,1,5)]
future = pd.DataFrame(index=date_list, columns= df_week.columns)
df_week2 = pd.concat([df_week2, future])
df_week2['forecast'] = invboxcox(best_model.predict(start=0, end=500), lmbda)
df_week.to_csv('Documents\Bitcoin\Original Week.csv')
df_week2.to_csv('Documents\Bitcoin\Prediction Week.csv')
plt.figure(figsize=(15,7))
df_week2.Weighted_Price.plot()
df_week2.forecast.plot(color='r', ls='--', label='Predicted Weighted_Price')
plt.legend()
plt.title('Bitcoin Prices, by Weeks')
plt.ylabel('mean USD')
plt.show()

```

```

# Prediction
EPSILON = 1e-10
df_week2 = df_week[['Weighted_Price']]
date_list = []
future = pd.DataFrame(index=date_list, columns= df_week.columns)
df_week2 = pd.concat([df_week2, future])
df_week2['forecast'] = invboxcox(best_model.predict(start=0, end=450), lmbda)
true_price = df_week.Weighted_Price
predicted_price = df_week2.forecast

def _error(true_price, predicted_price):
    """ Simple error """
    return true_price - predicted_price

def _percentage_error(true_price, predicted_price):
    """
    Percentage error
    Note: result is NOT multiplied by 100
    """
    return _error(true_price, predicted_price) / (true_price + EPSILON)

n_mse = np.mean(np.square(_error(true_price, predicted_price)))
print('New Mean Squared Error: {}'.format(round(n_mse, 2)))

mae = np.mean(np.abs(_error(true_price, predicted_price)))
print('Mean Absolute Error: {}'.format(round(mae, 2)))

print('RMSE:', np.sqrt(metrics.mean_squared_error(true_price, predicted_price)))

mape = np.mean(np.abs(_percentage_error(true_price, predicted_price)))
print('Mean Absolute Percentage Error: {}'.format(round(mape, 2)))

```

Figure 8: Predicting values and checking Accuracy using Jupyter

3.4 Terrorism ARIMA prediction

Terrorism Data is also loaded onto Jupyter Notebook as seen in Figure 9 for further processing and implementing of model on it. It is mostly similar to the Bitcoin ARIMA prediction but here the data needs to be pre processed and then converted to a time series to implement time series prediction and the code guidance is taken from Aptem (2017).

```
# Import libraries
import numpy as np #array operations and for Linear Algebra
import pandas as pd #used for reading CSV files.
import matplotlib.pyplot as plt # For Data plotting
from scipy import stats #for scientific Computing
import statsmodels.api as sm #for statistical data tests and statistical data exploration
from statsmodels.tsa.stattools import adfuller#Agumented Dickey Fuller Test
from statsmodels.tsa.arima_model import ARIMA #For importing ARIMA model
import warnings
from itertools import product #Used for creating nested for loops
from datetime import datetime #for date and time functions
from sklearn import metrics #for mean errors and other error functions
warnings.filterwarnings('ignore')
plt.style.use('seaborn-poster')

#Loading the file from the local system
df= pd.read_csv('Documents\Terror\Clean Terror.csv',encoding = 'ANSI',parse_dates = ['Date'])
#removing the country columns since not required.
df.drop(('country_txt'), axis =1 , inplace = True)
#Index set as the date column
df.index = df.Date
#Removing Date column as already set as Index.
df.drop(('Date'), axis =1 , inplace = True)
df.head()
```

Figure 9: Library initialization and Data Loaded using Jupyter

Initially the data is loaded and parsed along the date column and the multiple country of the 'country_txt' is dropped. The date column is set as the index for further processing as seen in Figure 10

```
df = df.groupby(['Date'],as_index=True).agg({'nkill':'sum','nwound': 'sum'})
df.head()
```

Date	nkill	nwound
2013-01-01	48	77
2013-01-02	15	36
2013-01-03	78	152
2013-01-04	8	25
2013-01-05	40	63

```
idx = pd.date_range('2013-01-01', '2017-12-31')
df.index = pd.DatetimeIndex(df.index)
df = df.reindex(idx, fill_value=0)
df.head()
```

```
df.replace(0, np.nan, inplace=True)
df.head()
```

```
df.fillna(method = 'ffill', inplace =True)
df = df.resample('W').mean()
df.head(10)
```

Figure 10: Data processing using Jupyter

The date is then grouped into 1 as there are multiple attacks in a day and need to

be summed into 1 to create a time series as seen in Figure 11 A new index is created to complete the time series. Since not all days there is a terror attack and hence we put in a new index and the values are set to 0, which are further on changed to 'NaN' and then they are filled in using ffill method since in box cox transformations all values should be positive and above 0. The data is also resampled into weekly format so as to reduce the error percentage. Below the data is plotted showing the number of people killed and wounded over the years in Iraq, Afghanistan, Pakistan, Yemen, Syria and Libya where ISIS is active.

```
fig = plt.figure(figsize=[15, 7])
plt.suptitle('People Killed or Wounded in Terror Attacks, in ISIS affected countries', fontsize=22)

plt.subplot(221)
plt.plot(df.nkill, '-', label='People Killed')
plt.legend()

plt.subplot(222)
plt.plot(df.nwound, '-', label='People Wounded')
plt.legend()

#plt.tight_layout()
plt.show()
```

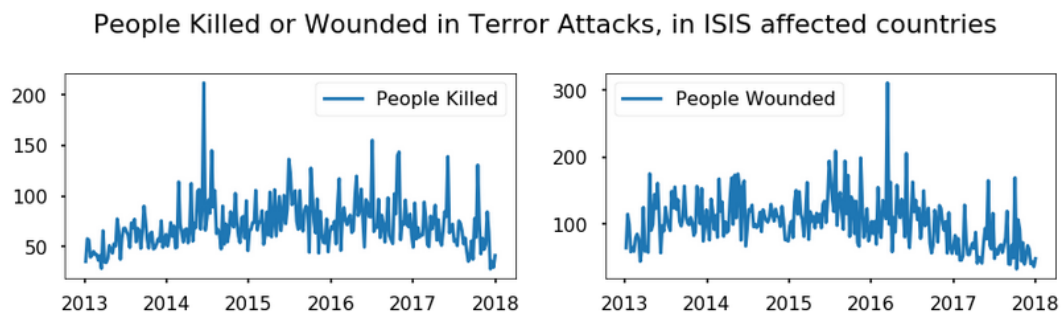


Figure 11: Data Plotted using Jupyter

Then the same process of ARIMA as in bitcoin prediction is repeated of finding auto correlation and removing stationary data. Further we predict the values in a time series based on the input of week dates that we provide and then check for the error percentage between the original values and the predicted values as seen in Figure 12

```

# Prediction
df2 = df[['nkill']]
date_list = [datetime(2018, 1, 1), datetime(2018, 1, 7), datetime(2018, 1, 14), datetime(2018, 1, 21),
             datetime(2018, 1, 28), datetime(2018, 2, 4), datetime(2018, 2, 11), datetime(2018, 2, 18),
             datetime(2018, 2, 25), datetime(2018, 3, 4), datetime(2018, 3, 11), datetime(2018, 3, 18),
             datetime(2018, 3, 25), datetime(2018, 4, 1), datetime(2018, 4, 15), datetime(2018, 4, 22),
             datetime(2018, 4, 29), datetime(2018, 5, 6), datetime(2018, 5, 13), datetime(2018, 5, 20),
             datetime(2018, 5, 27), datetime(2018, 6, 3), datetime(2018, 6, 10), datetime(2018, 6, 17),
             datetime(2018, 6, 24), datetime(2018, 7, 1), datetime(2018, 7, 8), datetime(2018, 7, 15),
             datetime(2018, 7, 22), datetime(2018, 7, 29)]
future = pd.DataFrame(index=date_list, columns= df.columns)
df2 = pd.concat([df2, future])
df2['forecast'] = invboxcox(best_model.predict(start=0, end=3000), lambda)
df2.to_csv('Documents\Terror\ARIMA\All Countries Original.csv')
df2.to_csv('Documents\Terror\ARIMA\All Countries Prediction.csv')
plt.figure(figsize=(15,7))
df2.nkill.plot()
df2.forecast.plot(color='r', ls='--', label='Predicted People Killed')
plt.legend()
plt.title('People Killed in Iraq, by Weeks')
plt.ylabel('Number of People')
plt.show()

```

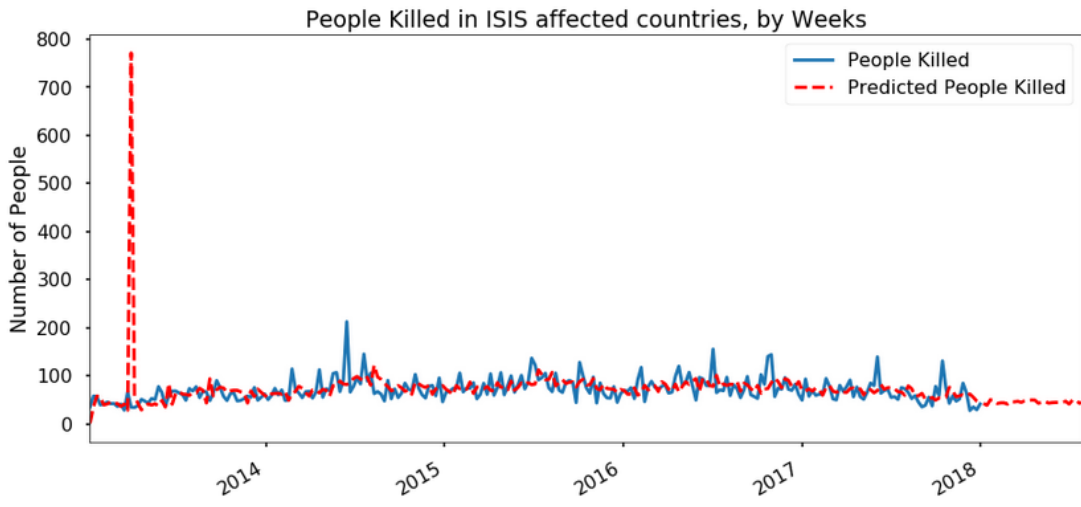


Figure 12: Data Plotted using Jupyter

3.5 Bitcoin RNN prediction

Bitcoin Data is loaded onto the Jupyter Notebook. Data is parsed based on the time stamp which was in seconds format and converted to normal date as seen in Figure 13 and the code guidance is taken from Tatbak (2019).

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # For Plotting graphs
from sklearn import metrics #For checking error
import warnings
warnings.filterwarnings("ignore")

bit_data=pd.read_csv("Documents/Bitcoin/BTCUSD.csv")
bit_data["date"]=pd.to_datetime(bit_data["Timestamp"],unit="s").dt.date
group=bit_data.groupby("date")
data=group["Close"].mean()
```

Figure 13: Data Loaded and Parsed using Jupyter

Next the data is split into training and testing for further prediction. It is saved into array format. After that the array is scaled by MinMaxScaler function and then transformed as seen in Figure 14

```
close_train=data.iloc[:len(data)-50] #setting the training data
close_test=data.iloc[len(close_train):]#setting the testing data

#feature scaling (set values between 0-1)
close_train=np.array(close_train) #create array
close_train=close_train.reshape(close_train.shape[0],1)#reshaping the array
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1)) #minimum value of the feature is subtracted by the range
close_scaled=scaler.fit_transform(close_train)

timestep=50
x_train=[]
y_train=[]

for i in range(timestep,close_scaled.shape[0]):
    x_train.append(close_scaled[i-timestep:i,0])
    y_train.append(close_scaled[i,0])

x_train,y_train=np.array(x_train),np.array(y_train)
x_train=x_train.reshape(x_train.shape[0],x_train.shape[1],1) #reshaped for RNN
print("x_train shape= ",x_train.shape)
print("y_train shape= ",y_train.shape)
```

Figure 14: Training and Testing data created using Jupyter

Next the RNN model is run using TensorFlow with 100 epoch size. There are five rnn layers deployed to improve the accuracy and working of the system, as seen in Figure 15

```

from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout, Flatten

regressor=Sequential()
#first RNN layer
regressor.add(SimpleRNN(128,activation="relu",return_sequences=True,input_shape=(x_train.shape[1],1)))
regressor.add(Dropout(0.25))
#second RNN layer
regressor.add(SimpleRNN(256,activation="relu",return_sequences=True))
regressor.add(Dropout(0.25))
#third RNN layer
regressor.add(SimpleRNN(512,activation="relu",return_sequences=True))
regressor.add(Dropout(0.35))
#fourth RNN layer
regressor.add(SimpleRNN(256,activation="relu",return_sequences=True))
regressor.add(Dropout(0.25))
#fifth RNN layer
regressor.add(SimpleRNN(128,activation="relu",return_sequences=True))
regressor.add(Dropout(0.25))
#convert the matrix to 1-line
regressor.add(Flatten())
#output layer
regressor.add(Dense(1))

regressor.compile(optimizer="adam",loss="mean_squared_error")
regressor.fit(x_train,y_train,epochs=100,batch_size=64)

```

Figure 15: RNN is run using Tensor Flow using Jupyter

```

inputs=data[len(data)-len(close_test)-timestep:]
inputs=inputs.values.reshape(-1,1)
inputs=scaler.transform(inputs)

x_test=[]
for i in range(timestep,inputs.shape[0]):
    x_test.append(inputs[i-timestep:i,0])
x_test=np.array(x_test)
x_test=x_test.reshape(x_test.shape[0],x_test.shape[1],1)

predicted_data=regressor.predict(x_test)
predicted_data=scaler.inverse_transform(predicted_data)
pd.DataFrame(predicted_data).to_csv('Documents\Bitcoin\Predicted Price RNN.csv')

data_test=np.array(close_test)
data_test=data_test.reshape(len(data_test),1)
pd.DataFrame(data_test).to_csv('Documents\Bitcoin\Original Price RNN.csv')

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(data_test,color="r",label="true result")
plt.plot(predicted_data,color="b",label="predicted result")
plt.legend()
plt.xlabel("Time(50 days)")
plt.ylabel("Close Values")
plt.grid(True)
plt.show()

```

Figure 16: RNN model is run and results displayed using Jupyter

The accuracy of the model is then checked. We use similar parameters as we did for ARIMA model. i.e New Mean Squared Error, Mean Absolute Error, Root Mean Squared Error and Mean Absolute Percentage Error as shown in Figure 17

```

# Prediction
EPSILON = 1e-10
true_price = data_test
predicted_price = predicted_data

def _error(true_price, predicted_price):
    """ Simple error """
    return true_price - predicted_price

def _percentage_error(true_price, predicted_price):
    """
    Percentage error
    Note: result is NOT multiplied by 100
    """
    return _error(true_price, predicted_price) / (true_price + EPSILON)

n_mse = np.mean(np.square(_error(true_price, predicted_price)))
print('New Mean Squared Error: {}'.format(round(n_mse, 2)))

mae = np.mean(np.abs(_error(true_price, predicted_price)))
print('Mean Absolute Error: {}'.format(round(mae, 2)))

print('RMSE:', np.sqrt(metrics.mean_squared_error(true_price, predicted_price)))

mape = np.mean(np.abs(_percentage_error(true_price, predicted_price)))
print('Mean Absolute Percentage Error: {}'.format(round(mape, 2)))

```

Figure 17: Model's Accuracy is Displayed using Jupyter

3.6 Bitcoin LSTM Prediction

Long Short term memory model has one of the best accuracy there is. It continues the work of RNN. It uses Tensor flow to calculate the values. It is much faster than RNN. This is because we reduce the batch size to 32 than the 64 in RNN. The code can be seen in Figure 18

```

from sklearn.metrics import mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Flatten

model=Sequential()

model.add(LSTM(10,input_shape=(None,1),activation="relu"))

model.add(Dense(1))

model.compile(loss="mean_squared_error",optimizer="adam")

model.fit(x_train,y_train,epochs=100,batch_size=32)

```

Figure 18: LSTM model is run using Tensor Flow using Jupyter

After that the accuracy is checked similar to the RNN model. We use similar parameters as we did for ARIMA model. i.e New Mean Squared Error, Mean Absolute Error, Root Mean Squared Error and Mean Absolute Percentage Error as shown in Figure 19


```

inputs=data[len(data)-len(close_test)-timestep:]
inputs=inputs.values.reshape(-1,1)
inputs=scaler.transform(inputs)

x_test=[]
for i in range(timestep,inputs.shape[0]):
    x_test.append(inputs[i-timestep:i,0])
x_test=np.array(x_test)
x_test=x_test.reshape(x_test.shape[0],x_test.shape[1],1)

predicted_data=model.predict(x_test)
predicted_data=scaler.inverse_transform(predicted_data)
pd.DataFrame(predicted_data).to_csv('Documents\Bitcoin\Predicted Price LSTM.csv')

data_test=np.array(close_test)
data_test=data_test.reshape(len(data_test),1)
pd.DataFrame(data_test).to_csv('Documents\Bitcoin\Original Price LSTM.csv')

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(data_test,color="r",label="true result")
plt.plot(predicted_data,color="b",label="predicted result")
plt.legend()
plt.xlabel("Time(50 days)")
plt.ylabel("Close Values")
plt.grid(True)
plt.show()

```

Figure 19: LSTM model's is plotted using Jupyter

References

- Aptem (2017). Bitcoin price prediction by arima.
URL: <https://www.kaggle.com/myonin/bitcoin-price-prediction-by-arima>
- Preda, G. (2019). Global terrorist attacks.
URL: <https://www.kaggle.com/gpreda/global-terrorist-attacks>
- START (2017). Global terrorism data.
URL: <https://www.kaggle.com/START-UMD/gtd>
- Tatbak, E. (2019). Rnn vs lstm on bitcoin dataset.
URL: <https://www.kaggle.com/etatbak/rnn-vs-lstm-on-bitcoin-dataset>
- Zielak (2019). Historical bitcoin data.
URL: <https://www.kaggle.com/mczielinski/bitcoin-historical-data>