Configuration Manual

MSc Research Project
Data Analytics

Ashwini Chaudhari
Student ID: x18129676

School of Computing
National College of Ireland

Supervisor:    Dr. Muhammad Iqbal
<table>
<thead>
<tr>
<th><strong>Student Name:</strong></th>
<th>Ashwini Chaudhari</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Student ID:</strong></td>
<td>x18129676</td>
</tr>
<tr>
<td><strong>Programme:</strong></td>
<td>MSc Data Analytics</td>
</tr>
<tr>
<td><strong>Year:</strong></td>
<td>2019-20</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>MSc Research Project</td>
</tr>
<tr>
<td><strong>Supervisor:</strong></td>
<td>Dr. Muhammad Iqbal</td>
</tr>
<tr>
<td><strong>Submission Due Date:</strong></td>
<td>12/12/2019</td>
</tr>
<tr>
<td><strong>Project Title:</strong></td>
<td>Configuration Manual</td>
</tr>
<tr>
<td><strong>Word Count:</strong></td>
<td>XXX</td>
</tr>
<tr>
<td><strong>Page Count:</strong></td>
<td>11</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th><strong>Signature:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date:</strong></td>
<td>12th December 2019</td>
</tr>
</tbody>
</table>

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

- Attach a completed copy of this sheet to each project (including multiple copies).
- Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).
- 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.

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

<table>
<thead>
<tr>
<th><strong>Office Use Only</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature:</strong></td>
</tr>
<tr>
<td><strong>Date:</strong></td>
</tr>
<tr>
<td><strong>Penalty Applied (if applicable):</strong></td>
</tr>
</tbody>
</table>
1 Introduction

This configuration manual elaborates on the system setup, software hardware specifications and the step carried out for the implementation of the Research Project: Forecasting cryptocurrency prices using Machine Learning.

2 System Configuration

2.1 Hardware

- Model: MacBook Pro, 2018, 256 GB
- Processor: 2.3 GHz Intel Core i5
- RAM: 8 GB 2133 MHz LPDDR3
- Graphics: Intel Iris Plus Graphics 655 1536 MB

2.2 Software

- Access to a Gmail account for access to Google Drive
- Google Colaboratory (Cloud based Jupyter notebook environment)

3 Project Development

This project follows a Cross Industry Standard Process for Data Mining (CRISP-DM) Methodology and the details of the research process such as data collection, processing techniques and algorithms are discussed below.

3.1 Google Colaboratory Environment Setup

The Google Colaboratory environment is used for perform the experiments. Google drive access is required for accessing Google Colaboratory and hence a valid Gmail account is essential. Once the URL is followed, a code is visible using which access will be granted to use the Google Drive for mounting the files and data.

We have used Python 3 notebooks for all Experiments
3.2 Data Preparation

Step 1: Getting Bitcoin Price Data from Quandl (2019)

```python
# Define Quandl Helper Function
def get_quandl_data(quandl_id):
    # Download and cache Quandl datasets
    cache_path = '{}.pkl'.format(quandl_id).replace('/', '_')
    try:
        f = open(cache_path, 'rb')
        df = pickle.load(f)
        print('Loaded {} from cache'.format(quandl_id))
    except (OSError, IOError) as e:
        print('Downloading {} from Quandl'.format(quandl_id))
        df = quandl.get(quandl_id, return_type='pandas')
        df.to_pickle(cache_path)
        print('Cached {} at {}'.format(quandl_id, cache_path))
    return df

# Pull Kraken Exchange Pricing Data

# Full Kraken BTC price exchange data
btc_usd_price_kraken = get_quandl_data('BCHARTS/KRAKENUSD')
```

Download BCHARTS/KRAKENUSD from Quandl
Cached BCHARTS/KRAKENUSD at BCHARTS/KRAKENUSD.pkl
Step 2: Get Bitcoin Pricing Data from Bitstamp, Coinbase and Itbit

```python
# Pull Pricing Data from More BTC Exchanges
# Pull pricing data for 3 more BTC exchanges
exchanges = ["COINBASE", "BITSTAMP", "ITBIT"]
exchange_data = {}

for exchange in exchanges:
    exchange_code = 'CHARTS/{}USD'.format(exchange)
    btc_exchange_df = get_quandl_data(exchange_code)
    exchange_data[exchange] = btc_exchange_df
```

Step 3: Combine all Bitcoin Pricing Data and calculate the average Bitcoin Price

Step 4: Get BTC-altcoin exchange rate data from Poloniex (2019)

```python
# Merge All Of The Pricing Data Into A Single DataFrame
def merge_df_on_column(dataframes, labels, col):
    """Merge a single column of each dataframe into a new combined dataframe""
    series_dict = {}
    for index in range(len(dataframes)):
        series_dict[labels[index]] = dataframes[index][col]
    return pd.DataFrame(series_dict)
```

```python
def get_json_data(json_url, cache_path):
    """Download and cache JSON data, return as a dataframe.""
    try:
        f = open(cache_path, 'rb')
        df = pickle.load(f)
        print('Loaded () from cache'.format(json_url))
    except (OSError, IOError) as e:
        print('Downloading ()'.format(json_url))
        df = pd.read_json(json_url)
        df.to_pickle(cache_path)
        print('Cached () at {}'.format(json_url, cache_path))
    return df
```

```python
base_pol_uri = 'https://poloniex.com/public?command=returnChartData&currencyPair[][start][end][startStr][endStr][startStr][endStr]'
start_date = datetime.datetime(2015-01-01), 'Y-m-d') # get data from the start of 2015
end_date = datetime.now() # up until today
period = 86400 # pull daily data ($6,400 seconds per day)

def get_crypto_data(poloniex_pair):
    """Retrieve cryptocurrency data from poloniex""
    json_url = base_pol_uri.format(poloniex_pair, start_date, end_date, start_date, end_date, period)
    data_df = get_json_data(json_url, poloniex_pair)
    return data_df
```

```python
altcoins = ["ETH", "LTC", "XRP", "ETC", "STR", "DASH", "SC", "XMR", "XEM"]

altcoin_data = {}
for altcoin in altcoins:
    coinpair = 'BTC_{}'.format(altcoin)
    crypto_price_df = get_crypto_data(coinpair)
    altcoin_data[altcoin] = crypto_price_df
```

[3]
Step 5: Calculate altcoin price using average bitcoin price and BTC-altcoin exchange rate.

Step 6: Combine Bitcoin average price and altcoin prices into a single dataframe.

4 Modelling

4.1 ARIMA Model

Step 1: Import all required libraries

```python
# import libraries
import pandas as pd
from pandas import DataFrame
import numpy as np

import matplotlib.pyplot as plt
plt.rcParams("figure.figsize") = (15,7)

import seaborn as sns
from datetime import datetime, timedelta

from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsa.plot_acf import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

from scipy import stats
import statsmodels.api as sm
from itertools import product

import warnings
warnings.filterwarnings('ignore')

print("all required libraries imported")
```

Step 2: Mount the google drive and read the csv file using pandas

```python
[ ] #Mount Drive
from google.colab import drive

drive.mount('/content/gdrive')

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call .

[ ] #Read data

df = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/altcoin_data.csv')
df.head()
```

Step 3: Set date as index and sort the dataframe

```python
[ ] #setting index as date
df['date'] = pd.to_datetime(df.date, format='Y-m-d')
df.index = df['date']

#sorting
df = df.sort_index(ascending=True, axis=0)

#plot
plt.figure(figsize=(10,5))
plt.plot(df['BTC'], label='Close Price history')
```
Step 4: Set forecast frequency and check the data for stationarity and seasonality.

```python
# Monthly Forecasting
# Resampling to monthly frequency
bmc_month = bmc.resample('M').mean()

# Stationarity check and Seasonal decomposition
seasonal_decompose(bmc_month['close'], freq=12).plot()
seasonal_decompose(bmc_month['BTC', freq = 12]).plot()
print("Dickey–Fuller test: p=\%f" % adfuller(bmc_month['BTC'][1]))
plt.show()
```

**Dickey–Fuller test: p=0.321209**

Step 5: Perform Box-Cox Transformations and Seasonal Differentiation

```python
# Box-Cox Transformations
bmc_month['close_box'], lbda = stats.boxcox(bmc_month['BTC'])
print("Dickey–Fuller test: p=\%f" % adfuller(bmc_month['close_box'][1]))

# Seasonal differentiation (12 months)
bmc_month['box_diff_seasonal_12'] = bmc_month['close_box'] - bmc_month['close_box'].shift(12)
print("Dickey–Fuller test: p=\%f" % adfuller(bmc_month['box_diff_seasonal_12'][12:][1]))

# Seasonal differentiation (3 months)
bmc_month['box_diff_seasonal_3'] = bmc_month['close_box'] - bmc_month['close_box'].shift(3)
print("Dickey–Fuller test: p=\%f" % adfuller(bmc_month['box_diff_seasonal_3'][3:][1]))

# Regular differentiation
bmc_month['box_diff2'] = bmc_month['box_diff_seasonal_12'] - bmc_month['box_diff_seasonal_12'].shift(1)

# STL decomposition
seasonal_decompose(bmc_month['box_diff2'][13:]).plot()
print("Dickey–Fuller test: p=\%f" % adfuller(bmc_month['box_diff2'][13:][1]))
plt.show()
```

Step 6: Perform initial approximation of parameters and model selection.

```python
# Initial approximation of parameters using Autocorrelation and Partial Autocorrelation Plots
ax = plt.subplot(211)
# Plot the autocorrelation function
sm.graphics.tsas.plot_acf(bmc_month['box_diff2'][13:].values.squeeze(), lags=48, ax=ax)
plot_acf(bmc_month['box_diff2'][13:].values.squeeze(), lags=12, ax=ax)
ax = plt.subplot(212)
sm.graphics.tsas.plot_pacf(bmc_month['box_diff2'][13:].values.squeeze(), lags=48, ax=ax)
plot_pacf(bmc_month['box_diff2'][13:].values.squeeze(), lags=12, ax=ax)
plt.tight_layout()
plt.show()
```
Step 7: Get Best Model summary and plot its diagnostics.

```python
# Initial approximation of parameters
qs = range(0, 3)
p = range(0, 3)
d=1
parameters = product(ps, qs)
parameters_list = list(parameters)\nlen(parameters_list)

# Show parameter list
print(parameters_list)

# Model Selection
results = []
best_aic = float("inf")
warnings.filterwarnings('ignore')
for param in parameters_list:
    try:
        model = SARIMAX(btc_month.close_box, order=(param[0], d, param[1])).fit(disp=-1)
        except ValueError:
            print('bad parameter combination:', param)
            continue
        aic = model.aic
        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=False).head())
print(best_model.summary())

print("Dickey-Fuller test:: p%lf" % adfuller(best_model.resid[13:][11])

best_model.plot_diagnostics(figsize=(15, 12))
plt.show()
```

Step 8: Perform STL Decomposition and plot the graph

```python
# STL-decomposition
plt.figure(figsize=(15,7))
plt.subplot(211)
best_model.resid[13:].plot()
plt.ylabel('Residuals')
ax = plt.subplot(212)
#sm.graphics.tsa.plot_acf(best_model.resid[13:].values.squeeze(), lags=48, ax=ax)
plot_acf(best_model.resid[13:].values.squeeze(), lags=12, ax=ax)

print("Dickey-Fuller test:: p%lf" % adfuller(best_model.resid[13:][13])

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

Step 9: Predict bitcoin prices for the following months, plot the graph and evaluate the model results.
The same code with minor modifications was used for implementing price prediction for Ethereum and Litecoin. [Kaggle](https://www.kaggle.com) (2019)

### 4.2 LSTM Model

Step 1: Import the required libraries for LSTM, mount the google drive and load the data.

```python
# Import required dependencies for LSTM Model
import pandas as pd
import numpy as np
import chart_studio.plotly as py
import plotly.graph_objs as go
from sklearn.model_selection import train_test_split
from numpy import import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Bidirectional

# Mount Drive
from google.colab import drive
drive.mount('/content/gdrive')

drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount('/content/gdrive', force_remount=True).

# Load data
data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/altcoins_data.csv')
data.head()
```

Step 2: Convert date to datetime format as required and plot the graph

```python
# Preparing the data
data['date'] = pd.to_datetime(data['date'])
grouped_data = data.groupby('date')
real_price = grouped_data['BTC'].mean()

data = go.Scatter(x=real_price.index, y=real_price.values)

plotly.offline.iplot(data)
```
Step 3: Build the Model using `split_sequence` function and Split the data into training and testing samples

```python
# Building the model

def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence)-1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# choose a number of time steps
n_steps = 5

# split into samples
X, y = split_sequence(real_price.values, n_steps)

# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

# split into train - test set
X_train = X[:365]
y_train = y[:365]
X_test = X[365:]
y_test = y[365:]

# store the test dates
test_dates = real_price.index[365:]
```

Step 4: Fit the Model and Train the data

```python
[39] def build_bidirectional_model(X, y):
    # define model
    model = Sequential()
    model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(n_steps, n_features)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    # fit model
    model.fit(X, y, epochs=200, verbose=1)
    return model

# Training the Data
bidirectional_model = build_bidirectional_model(X, y)
```
Step 5: Predict the Price and plot the Graph

```python
# Price Prediction
yhat = bidirectional_model.predict(X_test, verbose=0)

# Plotting the Prediction
trace_high = go.Scatter(
    x=test_dates,
    y=y_test,
    name='Original',
    line=dict(color='#0000ff'),
    opacity=0.4)

trace_low = go.Scatter(
    x=test_dates,
    y=pd.DataFrame(yhat)[0].values,
    name='Forecasted',
    line=dict(color='#ff0000'),
    opacity=0.4)

data = [trace_high, trace_low]
layout = go.Layout(
    title=go.layout.Title(
        text='Bitcoin Price Forecast',
        xref='paper',
        x=0
    ),
    xaxis=go.layout.XAxis(
        title=go.layout.xaxis.Title(
            text='Dates',
            font=dict(
                family='Courier New, monospace',
                size=12,
                color='#7f7f7f'
            )
        ),
    ),
    yaxis=go.layout.YAxis(
        title=go.layout.yaxis.Title(
            text='Bitcoin Price in USD',
            font=dict(
                family='Courier New, monospace',
                size=12,
                color='#7f7f7f'
            )
        ),
    )
)
fig = go.Figure(data=data, layout=layout)
fig = dict(data=data)
plotly.offline.iplot(fig)
```
4.3 Prophet Model

The Prophet Model developed by Facebook is used for forecasting and the steps for the same are as below:

   Step 1: Import the required libraries

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import fbprophet
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import plotly.graph_objs as go
import plotly as py
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

   Step 2: Load the data, select required columns in the dataframe. Then rename the date column to ‘ds’ and Bitcoin Price column to ‘y’.

```python
# Load data
data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/altcoin_data.csv')
data.head()

data = data[365:]
data = data[['date','BTC']]
data.head()

data.columns = ['ds', 'y']
data.head()

data['y']=data['y'].astype(float)
data.tail()
```

   Step 3: Fit the dataframe to the prophet model and forecast prices for the next 4 months.

```python
# Fit prophet model
df_prophet = fbprophet.Prophet(changepoint_prior_scale=0.15, daily_seasonality=True)
df_prophet.fit(data)

# Forecast for 4 months
forecast_time=123 # 4 months
df_forecast = df_prophet.make_future_dataframe(periods= forecast_time, freq='D')

# Do forecasting
df_forecast = df_prophet.predict(df_forecast)
df_forecast.head()
```
Step 4: Plot the forecasted values and evaluate the model results

![Plot of forecasted values](image)

**References**


