Configuration Manual

MSc Research Project
Data Analytics

Ashwini Chaudhari
Student ID: x18129676

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

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

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:	12th December 2019

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.

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

Configuration Manual

Ashwini Chaudhari
x18129676

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

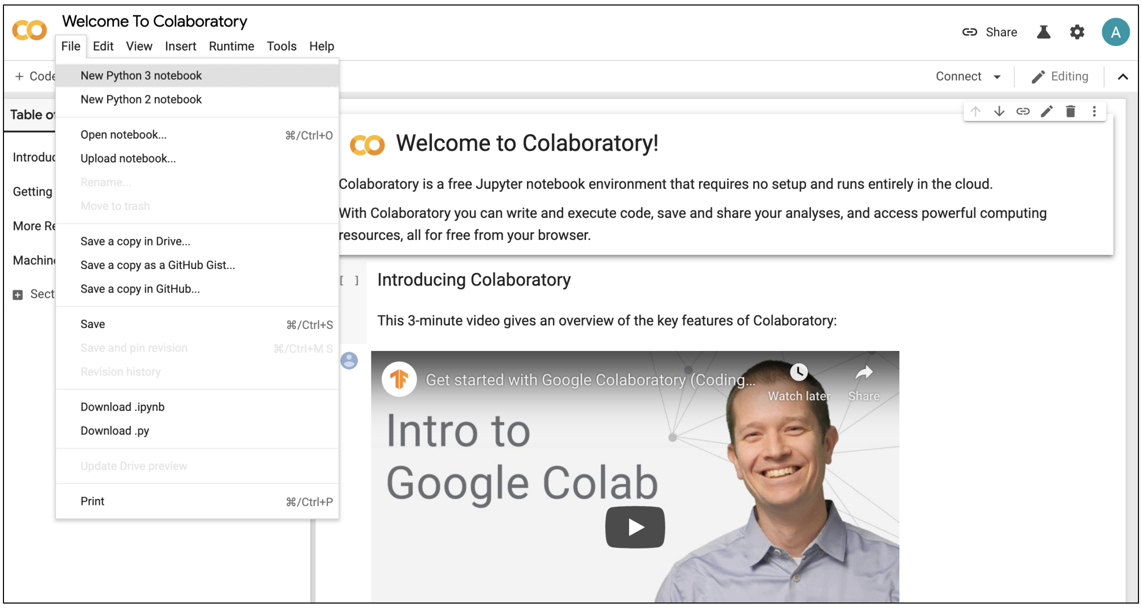


Figure 1: Modified CRISP-DM Process

3.2 Data Preparation

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

```
[ ] #Define Quandl Helper Function
    def get_quandl_data(quandl_id):
        '''Download and cache Quandl dataseries'''
        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, returns="pandas")
            df.to_pickle(cache_path)
            print('Cached {} at {}'.format(quandl_id, cache_path))
        return df
```

```
[ ] #Pull Kraken Exchange Pricing Data

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

```
Downloading BCHARTS/KRAKENUSD from Quandl
Cached BCHARTS/KRAKENUSD at BCHARTS-KRAKENUSD.pkl
```

Step 2: Get Bitcoin Pricing Data from Bitstamp, Coinbase and Itbit

```
# Pull Pricing Data From More BTC Exchanges
# Pull pricing data for 3 more BTC exchanges
exchanges = ['COINBASE','BITSTAMP','ITBIT']

exchange_data = {}

exchange_data['KRAKEN'] = btc_usd_price_kraken

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

```
Downloading BCHARTS/COINBASEUSD from Quandl
Cached BCHARTS/COINBASEUSD at BCHARTS-COINBASEUSD.pkl
Downloading BCHARTS/BITSTAMPUSD from Quandl
Cached BCHARTS/BITSTAMPUSD at BCHARTS-BITSTAMPUSD.pkl
Downloading BCHARTS/ITBITUSD from Quandl
Cached BCHARTS/ITBITUSD at BCHARTS-ITBITUSD.pkl
```

+ Code   + Text

```
# Merge All Of The Pricing Data Into A Single Dataframe
def merge_dfs_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)
```

Step 3: Combine all Bitcoin Pricing Data and calculate the average Bitcoin Price
Step 4: Get BTC-altcoin exchange rate data from *Poloniex* (2019)

```
[ ] #Define Poloniex API Helper Functions
    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
```

```
[ ] base_polo_url = 'https://poloniex.com/public?command=returnChartData&currencyPair={}&start={}&end={}&period={}'
    start_date = datetime.strptime('2015-01-01', '%Y-%m-%d') # get data from the start of 2015
    end_date = datetime.now() # up until today
    pediod = 86400 # pull daily data (86,400 seconds per day)

    def get_crypto_data(poloniex_pair):
        '''Retrieve cryptocurrency data from poloniex'''
        json_url = base_polo_url.format(poloniex_pair, start_date.timestamp(), end_date.timestamp(), pediod)
        data_df = get_json_data(json_url, poloniex_pair)
        data_df = data_df.set_index('date')
        return data_df
```

```
[ ] 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
```

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. *Data* (2019)

4 Modelling

4.1 ARIMA Model

Step 1: Import all required libraries

```
#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.tsaplots 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")
```

```
all required libraries imported
```

Step 2: Mount the google drive and read the csv file using pandas *Quandl* (2019)

```
[ ] #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

```
[ ] #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.

```
[ ] # Monthly Forecasting
    # Resampling to monthly frequency
    btc_month = btc.resample('M').mean()
```

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

```
Dickey–Fuller test: p=0.321209
```

Step 5: Perform Box-Cox Transformations and Seasonal Differentiation

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

```
Dickey–Fuller test: p=0.475547
```

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

```
Dickey–Fuller test: p=0.470502
```

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

```
Dickey–Fuller test: p=0.089300
```

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

    # STL-decomposition
    seasonal_decompose(btc_month.box_diff2[13:]).plot()
    print("Dickey–Fuller test: p=%f" % adfuller(btc_month.box_diff2[13:])[1])

    plt.show()
```

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

```
# Initial approximation of parameters using Autocorrelation and Partial Autocorrelation Plots
ax = plt.subplot(211)
# Plot the autocorrelation function
#sm.graphics.tsa.plot_acf(btc_month.box_diff2[13:].values.squeeze(), lags=48, ax=ax)
plot_acf(btc_month.box_diff2[13:].values.squeeze(), lags=12, ax=ax)
ax = plt.subplot(212)
#sm.graphics.tsa.plot_pacf(btc_month.box_diff2[13:].values.squeeze(), lags=48, ax=ax)
plot_pacf(btc_month.box_diff2[13:].values.squeeze(), lags=12, ax=ax)
plt.tight_layout()
plt.show()
```

```
[ ] # Initial approximation of parameters
    qs = range(0, 3)
    ps = range(0, 3)
    d=1
    parameters = product(ps, qs)
    parameters_list = list(parameters)
    len(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])
```

Step 7: Get Best Model summary and plot its diagnostics.

```
[ ] # 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())
```

```
[ ] print("Dickey–Fuller test:: p=%f" % adfuller(best_model.resid[13:])[1])
```

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

Step 8: Perform STL Decomposition and plot the graph

```
[ ] # STL-decomposition
    plt.figure(figsize=(15,7))
    plt.subplot(211)
    best_model.resid[13:].plot()
    plt.ylabel(u'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=%f" % adfuller(best_model.resid[13:])[1])

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

Step 9: Predict bitcoin prices for the following months, plot the graph and evaluate the model results.

```
[ ] # Prediction
    btc_month2 = btc_month[['BTC']]
    #date_list = [datetime(2018, 3, 31), datetime(2018, 4, 30), datetime(2018, 5, 31), datetime(2018, 6, 30)]
    date_list = [datetime(2019, 12, 31), datetime(2020, 1, 31), datetime(2020, 2, 29), datetime(2020, 3, 31),
                 datetime(2020, 4, 30), datetime(2020, 5, 31), datetime(2020, 6, 30), datetime(2020, 7, 31),
                 datetime(2020, 8, 31), datetime(2020, 9, 30), datetime(2020, 10, 31), datetime(2020, 11, 30)]

    future = pd.DataFrame(index=date_list, columns= btc_month.columns)
    btc_month2 = pd.concat([btc_month2, future])

    btc_month2['forecast'] = invboxcox(best_model.predict(start=0, end=120), lmbda)

    plt.figure(figsize=(15,7))
    btc_month2.BTC.plot(label = 'Observed')
    btc_month2.forecast.plot(color='r', ls='--', label='Forecasted')
    plt.legend()
    plt.title('Bitcoin Price Close')
    plt.ylabel('Dates')
    plt.ylabel('Bitcoin Price in USD')
    plt.savefig('bitcoin_monthly_forecast.png')
    plt.show()
```

The same code with minor modifications was used for implementing price prediction for Ethereum and Litecoin. *Kaggle* (2019)

4.2 LSTM Model

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

```
# 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 array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Bidirectional
```

```
[31] #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).
```

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

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

```
[35] #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

```
#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)
```

```
[38]
# 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[0:-365]
y_train = y[0:-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

```
[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

```
[41] #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)
```

DeepLearning (2019) *LSTM.* (2019)

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

```
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'.

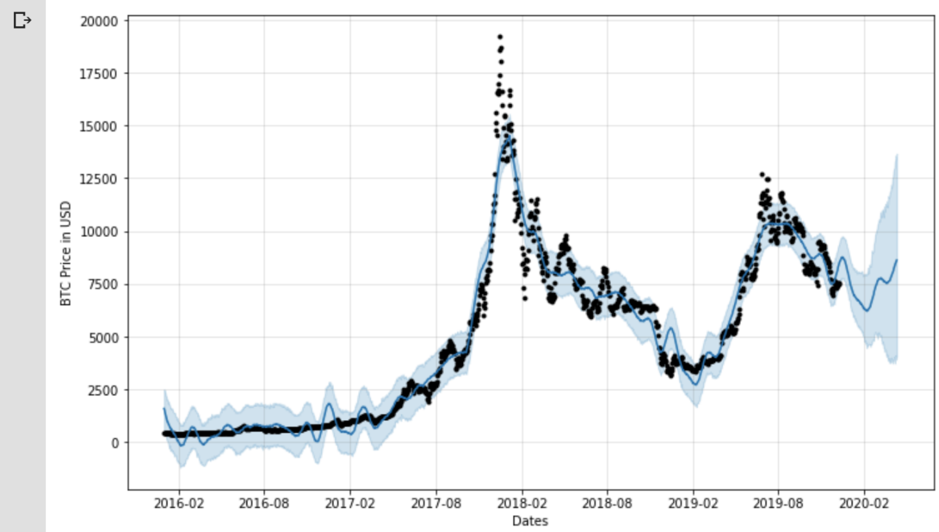
```
[ ] # Load data
    data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/altcoin_data.csv')
    data.head()

[ ] 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.

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

    # Forecast for 4 months
    fcast_time=123  # 4 months
    df_forecast = df_prophet.make_future_dataframe(periods= fcast_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

```
[ ] df_prophet.plot(df_forecast, xlabel = 'Dates', ylabel = 'BTC Price in USD')
```



Prophet (2019)

References

Data (2019). `https://blog.patricktriest.com/analyzing-cryptocurrencies-python/`. [Online].

DeepLearning (2019). `https://towardsdatascience.com/predicting-bitcoin-prices-with-deep-learning-438bc3cf9a6f`. [Online].

Kaggle (2019). `https://www.kaggle.com/jessevent/all-crypto-currencies`. [Online].

LSTM. (2019). `https://sergioskar.github.io/Bitcon_prediction_LSTM/`. [Online].

Poloniex (2019). `https://www.poloniex.com/`. [Online].

Prophet (2019). `https://towardsdatascience.com/apple-stock-and-bitcoin-price-predictions-using-fbs-prophet-for-beginners-python-9`
[Online].

Quandl (2019). `https://www.quandl.com/`. [Online].