Configuration Manual

MSc Research Project
Data Analytics

Rohit Jagadale
Student ID: x18184545

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rafai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rohit Jagadale		
Student ID:	x18184545		
Programme:	MSc Data Analytics	**Year:**	2019-2020
Module:	Research Project		
Supervisor:	Mr. Hicham Rafai		
Submission Due Date:	17/08/2020		
Project Title:	Configuration Manual		
Word Count:	1117		

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:	Rohit Jagadale
Date:	17/08/2020

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):	

Rohit Jagadale
x18184545

1 Introduction

Configuration Manual provides detailed documentation of the implementation of Storm Trajectory Prediction System. The system configurations required for the project are listed as below:

Operating System: Windows 10
Processor: Intel i7 2.8 GHz
RAM: 12 GB
System: 64-bit Operating System

2 Integrated Development Environment

The project is implemented using Python 3.7 version on Google Colab[1]. Google Colab is a free cloud service where the python applications can be developed and can be executed on GPU if required. As this project does not involve large processing, the Google Colab environment is kept in default CPU mode. To save the dataset, Microsoft Excel is used

3 Datasets

The National Hurricane Center (NHC) data present on Kaggle[2] data repository is used for the research. The data is present in CSV file format and contains 49105 records. The data file is uploaded on Google Colab environment

4 Data Preprocessing

The code implemented as part for the research is developed with the reference of GitHub Repository[3] and the LSTM model hyperparameters are based on a previous research carried out by authors (Alemany *et al.*, 2019)

As a first step of the implementation, the HURDAT2 dataset is loaded into Google Colab environment from local system. As a next step, all the required libraries are loaded for the analysis

[1] https://colab.research.google.com/notebooks/intro.ipynb#recent=true
[2] https://www.kaggle.com/noaa/hurricane-database
[3] https://github.com/hammad93/hurricane-net

```python
import pandas as pd
import numpy as np
import tensorflow
import matplotlib.pyplot as plt
import math as Math
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import math, time
from keras.models import model_from_json
from geopy.distance import great_circle as vc
import seaborn as sns
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import Bidirectional

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

The data file is loaded into dataframe and further is processed using different processing techniques.

```python
storm_data = pd.read_csv('atlantic.csv')
```

```python
storm_data.head()
```

	ID	Name	Date	Time	Event	Status	Latitude	Longitude	Maximum Wind	Minimum Pressure	Low Wind NE	Low Wind SE	Low Wind SW	Low Wind NW	Moderate Wind NE	Moderate Wind SE	Moderate Wind SW	Moderate Wind NW	High Wind NE	High Wind SE	High Wind SW	High Wind NW
0	AL011851	UNNAMED	18510625	0		HU	28.0N	94.8W	80	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999
1	AL011851	UNNAMED	18510625	600		HU	28.0N	95.4W	80	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999
2	AL011851	UNNAMED	18510625	1200		HU	28.0N	96.0W	80	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999
3	AL011851	UNNAMED	18510625	1800		HU	28.1N	96.5W	80	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999
4	AL011851	UNNAMED	18510625	2100	L	HU	28.2N	96.8W	80	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999	-999

```python
storm_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49105 entries, 0 to 49104
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                49105 non-null  object
 1   Name              49105 non-null  object
 2   Date              49105 non-null  int64
 3   Time              49105 non-null  int64
 4   Event             49105 non-null  object
 5   Status            49105 non-null  object
 6   Latitude          49105 non-null  object
 7   Longitude         49105 non-null  object
 8   Maximum Wind      49105 non-null  int64
 9   Minimum Pressure  49105 non-null  int64
 10  Low Wind NE       49105 non-null  int64
 11  Low Wind SE       49105 non-null  int64
 12  Low Wind SW       49105 non-null  int64
 13  Low Wind NW       49105 non-null  int64
 14  Moderate Wind NE  49105 non-null  int64
 15  Moderate Wind SE  49105 non-null  int64
 16  Moderate Wind SW  49105 non-null  int64
 17  Moderate Wind NW  49105 non-null  int64
 18  High Wind NE      49105 non-null  int64
 19  High Wind SE      49105 non-null  int64
 20  High Wind SW      49105 non-null  int64
 21  High Wind NW      49105 non-null  int64
dtypes: int64(16), object(6)
memory usage: 8.2+ MB
```

Data preprocessing:

From the dataset, irrelevant columns are removed and the remaining columns are renamed for better understanding. Null values and duplicate records are checked in the data. The Latitude and longitude columns are changed from string type to float for further calculations

```
#Remove columns which are not relevant for the research
storm_data.drop(['Low Wind NE', 'Low Wind SE', 'Low Wind SW',
                 'Low Wind NW','Moderate Wind NE','Moderate Wind SE','Moderate Wind SW','Moderate Wind NW',
                 'High Wind NE','High Wind SE','High Wind SW','High Wind NW'], axis = 1, inplace = True)
```

```
storm_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49105 entries, 0 to 49104
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                49105 non-null  object
 1   Name              49105 non-null  object
 2   Date              49105 non-null  int64
 3   Time              49105 non-null  int64
 4   Event             49105 non-null  object
 5   Status            49105 non-null  object
 6   Latitude          49105 non-null  object
 7   Longitude         49105 non-null  object
 8   Maximum Wind      49105 non-null  int64
 9   Minimum Pressure  49105 non-null  int64
dtypes: int64(4), object(6)
memory usage: 3.7+ MB
```

```
#Total Number of records in the dataset
len(storm_data)
```

```
49105
```

```
#Renaming the columns into meaningful information
storm_data.columns = ['Storm_ID','Name','Date','Time','Event','Status','Latitude','Longitude','WindSpeed','Pressure']

#Removing the values with negative pressure
storm_data = storm_data[storm_data['Pressure'] != -999]

#Checking for the NaN values
#c = (storm_data['WindSpeed'] == NULL ).sum()
count = storm_data["WindSpeed"].isna().sum()
count1 = storm_data["Pressure"].isna().sum()
print(count,count1)
```

```
0 0
```

```
#Cheking for the blank values in latitude and longitude
storm_data.loc[storm_data['Latitude'] == ''].count().iloc[0]
```

```
0
```

```
storm_data.loc[storm_data['Longitude'] == ''].count().iloc[0]
```

```
0
```

As seen from the above outputs, there are no null values in the data. The below code describes the number of storms present in the dataset

```
#Total amount of storms we have in our dataset
storm_count = len(pd.unique(storm_data['Storm_ID']))
print(storm_count)
```

```
1030
```

```
#Removing the storms which have only one instance in the dataset
storm_data = storm_data.groupby('Storm_ID').filter(lambda x : len(x)>1)
```

```
#Total amount of storms we have in our dataset
storm_count = len(pd.unique(storm_data['Storm_ID']))
print(storm_count)
```

```
905
```

```
#Remove "N" from latitude and longitude
storm_data['Latitude'] = storm_data['Latitude'].str[:-1]
storm_data['Longitude'] = storm_data['Longitude'].str[:-1]
```

```
#Converting Date, Time into string
storm_data['Date']=storm_data['Date'].astype(str)
storm_data['Time']=storm_data['Time'].astype(str)
#Converting Latitude and Longitude into float
storm_data['Latitude']=storm_data['Latitude'].astype(float)
storm_data['Longitude']=storm_data['Longitude'].astype(float)
```

```
#Lambda function to make all the records in Time column of length 4 by adding preceding zeros
storm_data['Time'] = storm_data['Time'].apply(lambda x: x.zfill(4))
```

Date Extraction: Day, Month and Year is extracted from date column whereas Hour is extracted from time column

```
#Date Extraction : Deriving seperate columns Year, Month, Day and Hour from Date and Time columns
storm_data['Year']=storm_data['Date'].str[0:4]
storm_data['Month']=storm_data['Date'].str[4:6]
storm_data['Day']=storm_data['Date'].str[6:]
storm_data['Hour']=storm_data['Time'].str[0:2]
```

```
#Dropping Date and Time columns as we have seperate columns now
storm_data.drop(['Date','Time'], axis = 1, inplace = True)
```

```
storm_data.head()
```

	Storm_ID	Name	Event	Status	Latitude	Longitude	WindSpeed	Pressure	Year	Month	Day	Hour
346	AL031854	UNNAMED		HU	28.0	78.6	110	938	1854	09	07	12
351	AL031854	UNNAMED		HU	31.6	81.1	100	950	1854	09	08	18
352	AL031854	UNNAMED	L	HU	31.7	81.1	100	950	1854	09	08	20
1039	AL081861	UNNAMED		TS	42.0	71.5	50	1000	1861	11	03	12
1040	AL081861	UNNAMED		TS	44.0	70.0	50	999	1861	11	03	18

```
# Enumerating the objects of dataframe into integers for EDA
keys = list(enumerate(pd.unique(storm_data['Storm_ID'])))

storm_count = len(pd.unique(storm_data['Storm_ID']))
print(storm_count)

y = np.zeros((storm_count))
for x in range(0,storm_count):
    y[x] = len(pd.DataFrame(storm_data[storm_data['Storm_ID'] == keys[x][1]], columns = storm_data.keys()).reset_index(drop = True))

#Deriving number of records for each storm instance
storm_instance = pd.DataFrame(y)
```

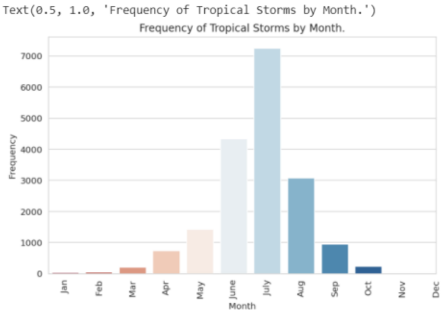
```
905
```

5 Exploratory Data Analysis (EDA)

The Below figure shows the frequency of Storms by month. As per the figure, July month displays the highest tropical storms

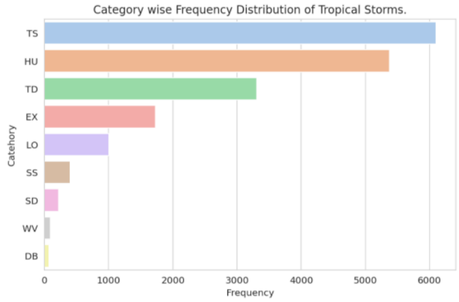
```
plt.figure(figsize = (8,5))
sns.set_style("whitegrid")
temp_bar = storm_data.groupby('Month').count()
mnt = ['Jan' , 'Feb' , 'Mar' , 'Apr' , 'May' , 'June' , 'July' , 'Aug' , 'Sep','Oct' , 'Nov' , 'Dec']
temp_bar = temp_bar.sort_values(by = 'Month' , ascending = False)
ax = sns.barplot(x = temp_bar.index , y = 'Year' , data=temp_bar , palette = 'RdBu' )
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11] , mnt , rotation = 90)
plt.ylabel('Frequency')
plt.title('Frequency of Tropical Storms by Month.')
```

```
Text(0.5, 1.0, 'Frequency of Tropical Storms by Month.')
```



Below figure explains the category wise storm distribution. The categories are described in the comments of the code

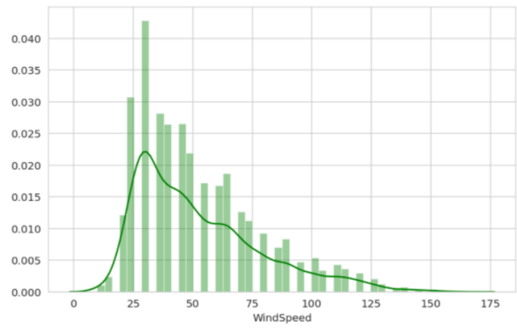
```
## Frequency of Cyclones by Category
# TD – Tropical cyclone of tropical depression intensity (< 34 knots)
# TS – Tropical cyclone of tropical storm intensity (34-63 knots)
# HU – Tropical cyclone of hurricane intensity (> 64 knots)
# EX – Extratropical cyclone (of any intensity)
# SD – Subtropical cyclone of subtropical depression intensity (< 34 knots)
# SS – Subtropical cyclone of subtropical storm intensity (> 34 knots)
# LO – A low that is neither a tropical cyclone, a subtropical cyclone, nor an extratropical cyclone (of any intensity)
# WV – Tropical Wave (of any intensity)
# DB – Disturbance (of any intensity)
temp_event = storm_data.groupby('Status').count().sort_values(by = 'Year' , ascending = False)
fig , ax = plt.subplots()
fig.set_size_inches(8,5)
sns.barplot(y = list(temp_event.index) , x = 'Year' , data = temp_event, palette= 'pastel' )
plt.xlabel('Frequency')
plt.ylabel('Catehory')
plt.title('Category wise Frequency Distribution of Tropical Storms.')
plt.show()
```



The windspeed and pressure distribution is plotted using seaborn library

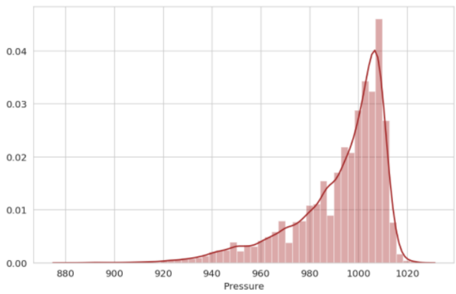
```
plt.figure(figsize = (8,5))
sns.distplot(storm_data['WindSpeed'],color='green')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5333ef550>
```




```
plt.figure(figsize = (8,5))
sns.distplot(storm_data['Pressure'],color='brown')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe53350e898>
```



6 Data Transformation

Data is transformed using multiple techniques like log transform, feature scaling, handling outliers. Distance of the storm is calculated using great-circle distance whereas angle of the storm or direction is calculated using in-build mathematical functions. All the comments in the code explains the transformation steps

```
#Calculating Distance and Direction
y = np.zeros(total_storm_count)
storm_data['distance'] = np.zeros(total_data_count)
storm_data['direction'] = np.zeros(total_data_count)

# For all storms
for x in range(0, total_storm_count):
    stormdf = pd.DataFrame(storm_data[storm_data['Storm_ID'] == keys[x][1]], columns = storm_data.keys()).reset_index(drop = False)
    stormdist = 0
    init = (0,0)

    # For all latitude and longitude points of storms, calculate the angle of travel and distance
    for n in zip(stormdf['Latitude'], stormdf['Longitude']):

        if init == (0,0):
            init = n
            continue
        # Stores the distance into the DataFrame
        storm_data.at[stormdf[(stormdf['Latitude'] == n[0]) & (stormdf['Longitude'] == n[1])]['index'].values[0], 'distance']= vc(init,n).miles

        long_dist = n[1] - init[1];
        temp = float(n[0])
        y_x = Math.sin(long_dist) * Math.cos(temp);

        x_x = Math.cos(n[1]) * Math.sin(temp) - Math.sin(n[1]) * Math.cos(temp) * Math.cos(long_dist);
        s_eq = Math.degrees(Math.atan2(y_x, x_x))
        if (s_eq < 0):
            s_eq+= 360;

        # Stores the angle of travel into the DataFrame
        storm_data.at[stormdf[(stormdf['Latitude'] == n[0]) & (stormdf['Longitude'] == n[1])]['index'].values[0], 'direction']= s_eq
        stormdist += vc(init,n).miles
        init = n
    y[x] = stormdist

# Now contains the distance between all given latitude and longitude points
storm_distance = pd.DataFrame(y)
```

Checking the newly added columns distance and direction in the dataframe

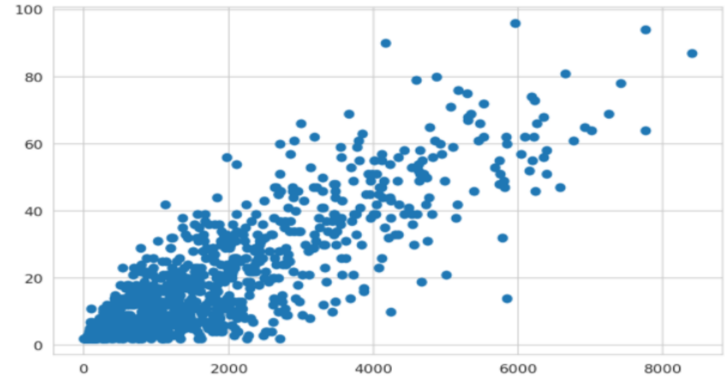
```
storm_data.describe()
```

	Latitude	Longitude	WindSpeed	Pressure	Year	Month	Day	Hour	distance	direction
count	18311.000000	18311.000000	18311.000000	18311.000000	18311.000000	18311.000000	18311.00000	18311.000000	18311.000000	18311.000000
mean	27.134963	63.373906	52.484408	992.326907	1991.852384	8.720059	15.83742	9.286768	91.804588	165.558382
std	10.111113	20.425977	26.398620	19.065069	19.854077	1.308524	8.89313	6.748599	107.572730	112.074834
min	7.200000	0.000000	10.000000	882.000000	1854.000000	1.000000	1.00000	0.000000	0.000000	-0.000000
25%	19.000000	48.400000	30.000000	984.000000	1983.000000	8.000000	8.00000	6.000000	41.855997	62.310919
50%	26.800000	64.800000	45.000000	999.000000	1996.000000	9.000000	16.00000	12.000000	71.363219	172.801334
75%	33.500000	79.600000	65.000000	1006.000000	2005.000000	9.000000	24.00000	18.000000	110.146413	254.800929
max	70.700000	109.300000	165.000000	1024.000000	2015.000000	12.000000	31.00000	23.000000	2704.994140	359.987816

Displaying the distribution of storm instances across the data

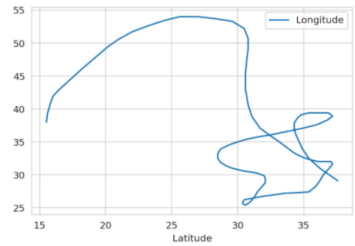
```
# Print the storm key with the amount of data they contain
print ('Top 5 storms with longest travel distance)')
for x in storm_distance.nlargest(5, 0).index:
    print (keys[x][1], "-", storm_distance.loc[x][0], "miles -", storm_instance.loc[x][0])

# Plotted the amount of storm distance traveled vs the amount of data they contain.

plt.figure(figsize = (8,5))
corr = plt.scatter(storm_distance[0], storm_instance[0])
plt.show()
```

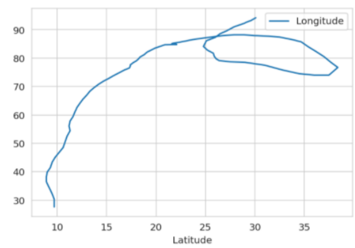
```
Top 5 storms with longest travel distance)
AL032000 - 8400.528797967156 miles - 87.0
AL092004 - 7757.573772788024 miles - 94.0
AL122011 - 7755.775736301039 miles - 64.0
AL131998 - 7421.006070822704 miles - 78.0
AL071995 - 7253.045320612251 miles - 69.0
```

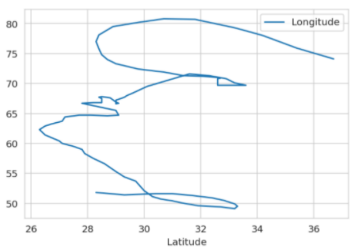


Below graphs shoes the longest travelling storm trajectories

```
# Graph the trajectories of the longest storms (the ones that traveled the most)
for x in storm_instance.nlargest(3, 0).index:
    storm_data[storm_data['Storm_ID'] == keys[x][1]].plot(x='Latitude', y='Longitude')
```







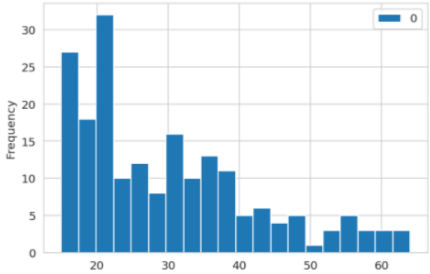
Checking the outliers in the data and removing them

```
#Removing outliers for data normalization
cond = (storm_instance > 14) & (storm_instance < 65)
keys25 = []

for x in cond.index:
    if cond.loc[x][0]:
        keys25.append(keys[x][1])

word2keys = {}
for x in keys:
    word2keys[x[1]] = x[0]

df = storm_data[storm_data['Storm_ID'].isin(keys25)]
```

Distribution of the data after removing outliers

```
storm_instance.plot.hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe53aceae80>
```



Defining the Grid Model

Below code depict the grid model design. As seen in the code Grid_Point column is created in the dataframe and using the hyperparameters, the grid points are developed.

```
# Assigning each point to a specific location in the grid.
# For example, we will learn how a storm in quadrant 2 with move.
df['Grid_Point'] = np.zeros(df_count)
```

```
# These variable are hyperparameters
lat_interval = round(66 - 7.2)
long_interval = round(13.5 + 109.3)

df['Grid_Point'] = np.floor(df['Latitude'] - 7.200)* long_interval  + np.floor(df['Longitude'] + 109.3)
df['Grid_Point'] = round(df['Grid_Point'])

df.head()
```

```
df.head() # Check loaded data
```

	Storm_ID	Name	Event	Status	Latitude	Longitude	WindSpeed	Pressure	Year	Month	Day	Hour	distance	direction	Grid_Point
20701	AL091945	UNNAMED		HU	25.1	80.0	115	949	1945	9	15	18	0.000000	0.000000	2280.0
20702	AL091945	UNNAMED	L	HU	25.3	80.3	115	949	1945	9	15	19	23.296261	16.943232	2403.0
20703	AL091945	UNNAMED	L	HU	25.4	80.4	115	949	1945	9	15	20	9.312732	5.526718	2403.0
20704	AL091945	UNNAMED		HU	25.9	80.9	100	954	1945	9	16	0	46.511365	20.205036	2404.0
20705	AL091945	UNNAMED		HU	26.6	81.5	85	963	1945	9	16	6	61.004848	3.360923	2527.0

```
df.to_csv('Data_with_Gridpoints.csv') # Save the dataframe to csv for checkpoint
```

7 Implementation and Evaluation of RNN models

For predicting the storm trajectories, LSTM and BiLSTM models are applied. Below code shows the model design

```
# Load the preprocessed data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('Data_with_Gridpoints.csv', index_col=0)

df.head() # Check loaded data
```

	Storm_ID	Name	Event	Status	Latitude	Longitude	WindSpeed	Pressure	Year	Month	Day	Hour	distance	direction	gridID
20701	AL091945	UNNAMED		HU	25.1	80.0	115	949	1945	9	15	18	0.000000	0.000000	2280.0
20702	AL091945	UNNAMED	L	HU	25.3	80.3	115	949	1945	9	15	19	23.296261	16.943232	2403.0
20703	AL091945	UNNAMED	L	HU	25.4	80.4	115	949	1945	9	15	20	9.312732	5.526718	2403.0
20704	AL091945	UNNAMED		HU	25.9	80.9	100	954	1945	9	16	0	46.511365	20.205036	2404.0
20705	AL091945	UNNAMED		HU	26.6	81.5	85	963	1945	9	16	6	61.004848	3.360923	2527.0

Before training the model, the manual feature selection is performed by selecting only the required columns and removing columns like Month, Day, Hour, Event, Latitude, Longitude and Status.

```
df.drop(['Month', 'Day', 'Hour','Event', 'Latitude', 'Longitude', 'Storm_ID','Year','Name','Status'], axis = 1, inplace = True)
temp_df = df

temp_df = temp_df[temp_df['distance'] > 0]
temp_df['distance'] = np.log(temp_df['distance'])

temp_df = temp_df[temp_df['direction'] > 0]
temp_df['direction'] = np.log(temp_df['direction'])

temp_df.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

	WindSpeed	Pressure	distance	direction	Grid_Point
20702	115	949	3.148293	2.829868	2403.0
20703	115	949	2.231383	1.709594	2403.0
20704	100	954	3.839697	3.005932	2404.0
20705	85	963	4.110953	1.212216	2527.0
20706	75	974	4.172329	5.842832	2651.0

Below code shows that the data has 7627 grid points.

```
max(temp_df['Grid_Point']) # Total grid spots
```

```
7627.0
```

The data is scaled for final models using MinMaxScalar functions.

```
# Scaling the data
v_scale = MinMaxScaler(feature_range=(0, 1))
temp_df = pd.DataFrame(v_scale.fit_transform(temp_df), columns=['WindSpeed', 'Pressure', 'Distance', 'Direction', 'Grid_Point'])
temp_df.head()
```

	WindSpeed	Pressure	Distance	Direction	Grid_Point
0	0.677419	0.471831	0.262629	0.647543	0.302816
1	0.677419	0.471831	0.098857	0.518344	0.302816
2	0.580645	0.507042	0.386122	0.667848	0.302949
3	0.483871	0.570423	0.434572	0.460983	0.319365
4	0.419355	0.647887	0.445534	0.995021	0.335914

Splitting the data into training and testing sets:

Simple random sampling is used for splitting the data into training and testing set. The Training set contains 80% of the data. The model is trained for sequence data which means if the sequence of features is 1 to 15 in training set, the predicted feature is 16th value and so on.

```
#Training and testing set for the models
#X_train : all the input features for the data with 15 sequences
#y_train : predictor based on the previous set of sequences
#For example : if sequence of the features is X_train 1 to 15 then y_train for the sequence is 16th value. For the next sequence of X_train 2 to 16 y_train is 17th value and so on...

def load_data(stock, seq_len):
    storm_features = len(stock.columns)
    data1 = stock.to_numpy()
    sequence_length = seq_len + 1 # Since index value starts at 0
    result = []

    for index in range(len(data1) - sequence_length):
        result.append(data1[index: index + sequence_length])

    result = np.array(result)
    row = len(result) * 0.80 # The train and test data is divided to 80:20 ratio
    train = result[:int(row), :]
    X_train = train[:, :-1]
    y_train = train[:, -1][:,-1]
    X_test = result[int(row):, :-1]
    y_test = result[int(row):, -1][:,-1]

    #Reshaping the X_train and y_train for model input
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], storm_features))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], storm_features))

    return [X_train, y_train, X_test, y_test]
```

The X_train and X_test has 10068 and 2518 records respectively with 5 features and sequence window of 15

```
num_seq = 15 # Another hyperparameter for window of sequences
X_train, y_train, X_test, y_test = load_data(temp_df[::-1], num_seq)
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train (10068, 15, 5)
y_train (10068,)
X_test (2518, 15, 5)
y_test (2518,)
```

Model 1: LSTM

Long Short-Term Model is applied with 28 neurons, dropout value of 0.1and two hidden layers. Activation function ‘tanh’ is used with loss function of mean squared error and ‘adam’ optimizer.

```
#Model 1 : LSTM model
# LSTM model is optimized for faster computational speed

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import math, time
np.random.seed(1337)

#Defining all the layers of the model
def build_model_lstm(layers):
    model_lstm = Sequential()

    for x in range(0,2):
        model_lstm.add(LSTM(units=28,input_shape = (X_train.shape[1],X_train.shape[2]), return_sequences=True))
        model_lstm.add(Dropout(0.1))

    model_lstm.add(LSTM(layers[2], return_sequences=False))
    model_lstm.add(Dropout(0.1))
    model_lstm.add(Dense(1,activation='tanh'))   #Using Hyperbolic Tangent activation function which will provide results in both the directions
    model_lstm.add(Dropout(0.1))

    start = time.time()
    model_lstm.compile(loss="mse", optimizer="adam")
    print("Compilation Time : ", time.time() - start)
    return model_lstm
```

```
#building the model
model_lstm = build_model_lstm([5, num_seq, 1])
```

```
Compilation Time :  0.0070552825927734375
```

```
model_lstm.summary() #Layer of the LSTM model
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_32 (LSTM)               (None, 15, 28)            3808
_________________________________________________________________
dropout_38 (Dropout)         (None, 15, 28)            0
_________________________________________________________________
lstm_33 (LSTM)               (None, 15, 28)            6384
_________________________________________________________________
dropout_39 (Dropout)         (None, 15, 28)            0
_________________________________________________________________
lstm_34 (LSTM)               (None, 1)                 120
_________________________________________________________________
dropout_40 (Dropout)         (None, 1)                 0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 2
_________________________________________________________________
dropout_41 (Dropout)         (None, 1)                 0
=================================================================
Total params: 10,314
Trainable params: 10,314
Non-trainable params: 0
_________________________________________________________________
```

```
#training the model
history_lstm = model_lstm.fit(X_train, y_train, batch_size=512, epochs=60, validation_split=0.1, verbose=1)
```

```
Epoch 1/60
18/18 [==============================] - 2s 112ms/step - loss: 0.0783 - val_loss: 0.0222
Epoch 2/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0489 - val_loss: 0.0208
Epoch 3/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0474 - val_loss: 0.0218
Epoch 4/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0450 - val_loss: 0.0195
Epoch 5/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0431 - val_loss: 0.0187
Epoch 6/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0413 - val_loss: 0.0181
Epoch 7/60
18/18 [==============================] - 1s 57ms/step - loss: 0.0400 - val_loss: 0.0178
Epoch 8/60
18/18 [==============================] - 1s 57ms/step - loss: 0.0390 - val_loss: 0.0164
Epoch 9/60
18/18 [==============================] - 1s 57ms/step - loss: 0.0370 - val_loss: 0.0149
Epoch 10/60
18/18 [==============================] - 1s 58ms/step - loss: 0.0359 - val_loss: 0.0149
Epoch 11/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0341 - val_loss: 0.0121
Epoch 12/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0338 - val_loss: 0.0142
Epoch 13/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0322 - val_loss: 0.0116
Epoch 14/60
18/18 [==============================] - 1s 56ms/step - loss: 0.0308 - val_loss: 0.0151
Epoch 15/60
```

Once the model is trained, the prediction is done using X_test. The test data is evaluated using MSE, MAE, RMSE and R-squared values which are displayed in below figure

```
#applying the trained model on test data
pred_lstm = model_lstm.predict(X_test)
```

```
#Evaluating the model
import sklearn.metrics as sm
from math import sqrt
print("Mean absolute error (MAE) =", round(sm.mean_absolute_error(y_test, pred_lstm), 4))
print("Mean squared error (MSE) =", round(sm.mean_squared_error(y_test, pred_lstm), 4))
print("Root Mean squared error (RMSE) =", round(sqrt(sm.mean_squared_error(y_test, pred_lstm)),4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred_lstm), 4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred_lstm), 4))
print("R2 score =", round(sm.r2_score(y_test, pred_lstm), 4))
```

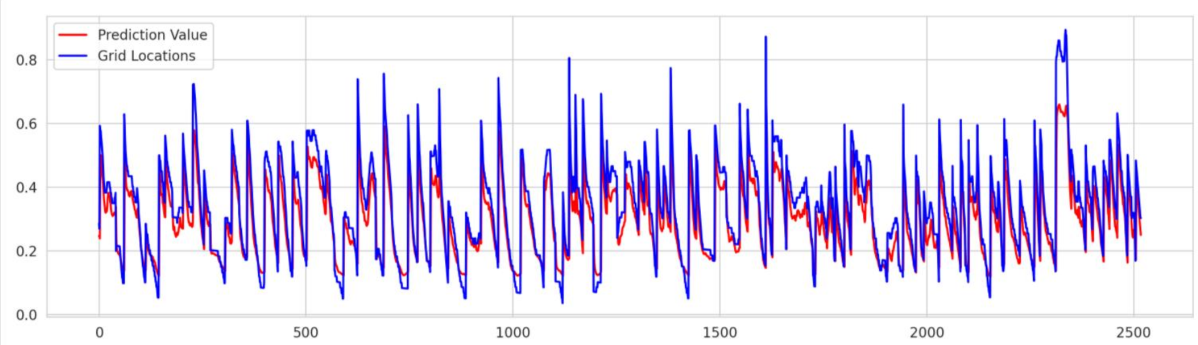
```
Mean absolute error (MAE) = 0.0568
Mean squared error (MSE) = 0.0076
Root Mean squared error (RMSE) = 0.0871
Median absolute error = 0.0453
Explain variance score = 0.7615
R2 score = 0.6599
```

The model loss diagram is plotted for checking the training progress of LSTM model

```
#Value loss curve
plt.figure(2)
plt.plot(history_lstm.history['loss'])
plt.plot(history_lstm.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Actual vs Predicted grip points are for LSTM model

```
#Actual vs Predicted grid points on Grid model
plt.figure(figsize=(15, 4), dpi=100)
plt.plot(pred_lstm, color='red', label='Prediction Value')
plt.plot(y_test, color='blue', label='Grid Locations')
plt.legend(loc='upper left')

plt.show()
```



Model 2: Bidirectional LSTM

The bidirectional LSTM model is applied with hyperparameters such as dropout value with 0.05, 64 neurons and single hidden layer. Activation function 'tanh' is used along with loss function 'mean squared error'. 'Adam' optimizer is used in the model

```
#Model 2 : Bidirectional LSTM : Bidirectional LSTM is extention of LSTM model with forward and backward layers of processing

#Defining the model parameters
def build_model(layers):
    model_biLSTM = Sequential()
    model_biLSTM.add(Bidirectional(LSTM(units = 64, return_sequences = True, dropout = 0.05),
                                   input_shape = (X_train.shape[1],X_train.shape[2])))
    model_biLSTM.add(Dropout(0.05))
    model_biLSTM.add((LSTM(1, return_sequences=False)))

    model_biLSTM.add(Dropout(0.05))

    model_biLSTM.add(Dense(1,activation='tanh'))

    start = time.time()
    model_biLSTM.compile(loss="mse", optimizer="adam")
    print("Compilation Time : ", time.time() - start)
    return model_biLSTM
```

```
#Building the model
model_biLSTM = build_model([5, num_seq, 1])
```

```
Compilation Time :  0.007235288619995117
```

```
model_biLSTM.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_7 (Bidirection (None, 15, 128)           35840
_________________________________________________________________
dropout_42 (Dropout)         (None, 15, 128)           0
_________________________________________________________________
lstm_36 (LSTM)               (None, 1)                 520
_________________________________________________________________
dropout_43 (Dropout)         (None, 1)                 0
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 2
=================================================================
Total params: 36,362
Trainable params: 36,362
Non-trainable params: 0
_________________________________________________________________
```

Model is trained with 60 epochs and batch size of 512

```
#Training the model with 60 epochs
history_biLSTM = model_biLSTM.fit(X_train, y_train, batch_size=512, epochs=60, validation_split=0.1, verbose=1)
```

```
18/18 [==============================] - 2s 112ms/step - loss: 0.0327 - val_loss: 0.0171
Epoch 3/60
18/18 [==============================] - 2s 114ms/step - loss: 0.0276 - val_loss: 0.0128
Epoch 4/60
18/18 [==============================] - 2s 113ms/step - loss: 0.0204 - val_loss: 0.0089
Epoch 5/60
18/18 [==============================] - 2s 114ms/step - loss: 0.0180 - val_loss: 0.0080
Epoch 6/60
18/18 [==============================] - 2s 115ms/step - loss: 0.0171 - val_loss: 0.0076
Epoch 7/60
18/18 [==============================] - 2s 115ms/step - loss: 0.0159 - val_loss: 0.0070
Epoch 8/60
18/18 [==============================] - 2s 115ms/step - loss: 0.0145 - val_loss: 0.0068
Epoch 9/60
18/18 [==============================] - 2s 113ms/step - loss: 0.0138 - val_loss: 0.0062
Epoch 10/60
18/18 [==============================] - 2s 113ms/step - loss: 0.0139 - val_loss: 0.0058
Epoch 11/60
18/18 [==============================] - 2s 116ms/step - loss: 0.0135 - val_loss: 0.0056
Epoch 12/60
18/18 [==============================] - 2s 114ms/step - loss: 0.0130 - val_loss: 0.0054
Epoch 13/60
18/18 [==============================] - 2s 115ms/step - loss: 0.0131 - val_loss: 0.0055
Epoch 14/60
18/18 [==============================] - 2s 114ms/step - loss: 0.0121 - val_loss: 0.0054
```

The model is predicted using test data and evaluated using multiple evaluation techniques mentioned in below figure

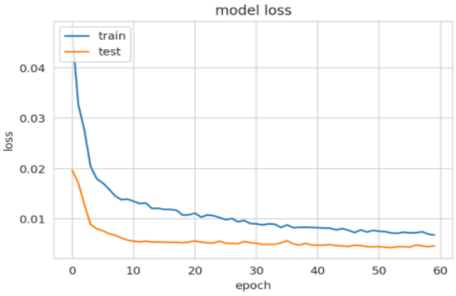
```
#Applying the model on test data
pred = model_biLSTM.predict(X_test)
```

```
#Evaluating the model
import sklearn.metrics as sm

print("Mean absolute error (MAE)=", round(sm.mean_absolute_error(y_test, pred),4))
print("Mean squared error (MSE) =", round(sm.mean_squared_error(y_test, pred), 4))
print("Root Mean squared error (RMSE) =", round(sqrt(sm.mean_squared_error(y_test, pred)),4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 4))
print("R2 score =", round(sm.r2_score(y_test, pred), 4))
```

```
Mean absolute error (MAE)= 0.0404
Mean squared error (MSE) = 0.0057
Root Mean squared error (RMSE) = 0.0755
Median absolute error = 0.0261
Explain variance score = 0.7694
R2 score = 0.7446
```

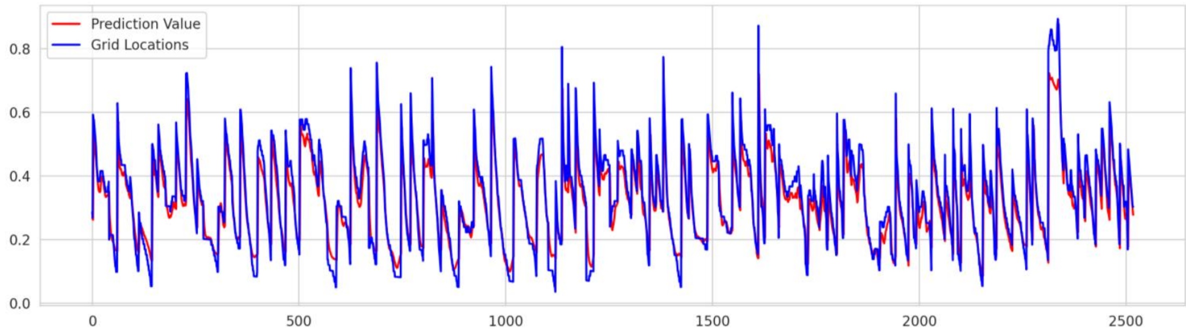
BiLSTM training curve is depicted in below figure

```
#Model Loss curve
plt.figure(2)
plt.plot(history_biLSTM.history['loss'])
plt.plot(history_biLSTM.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Similar to LSTM, the actual vs predicted grid points are plotted in below figure

```
#Actual vs Predicted grid points for grid-based model
plt.figure(figsize=(15, 4), dpi=100)
plt.plot(pred, color='red', label='Prediction Value')
plt.plot(y_test, color='blue', label='Grid Locations')
plt.legend(loc='upper left')

plt.show()
```



References:

Alemany, S. *et al.* (2019) 'Predicting Hurricane Trajectories Using a Recurrent Neural Network', *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, pp. 468–475. doi: 10.1609/aaai.v33i01.3301468.