Configuration Manual

MSc Research Project
Data Analytics

Jatin Rajkumar Singh
Student ID:x20227965

School of Computing
National College of Ireland

Supervisor: Dr Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Jatin Rajkumar Singh
Student ID:	X20227965
Programme:	Data Analytics
Year:	2022-23
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	454
Page Count:	18

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:	Jatin Rajkumar Singh
Date:	31st January 2023

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

Jatin Rajkumar Singh
X20227965

1 Introduction

This document provides all the hardware and software requirements to run the reproducible code for predicting the total power consumption of electric vehicles. Also, it includes descriptions of the columns used in the study.

2 System Configuration

System configuration requirements are divided into 2 subsections for a better understanding of the overall demand of the project.

2.1 Hardware Configuration

The given code and given dataset can be run on Intel's i5 or later or Apple's M1 or later processor. This code will also run on AMD R5 or better versions as well. A minimum of 8 GB of RAM is necessary for the smooth functioning of the code. Code and the dataset require 128 GB or more storage for storing and retrieving the dataset. This code can be run on cloud platforms such as Google colab and kaggle etc.

2.2 Software Requirements

This code can be run on Windows as well as Mac operating systems. For implementing this code on the local machine, Anaconda's Jupyter notebook can be used. In order to run the code on cloud platforms, Google Colab and kaggle etc. Also, this study has used CSV dataset, hence CSV file reader can be used to study the file. This study has used Python programming language along with Pandas, Numpy, Matplotlib, Seaborn, Tensorflow and Keras libraries etc to implement the code, perform the visualisation and train the model.

3 Implementation

This section covers the implementation of the code from data acquisition to results evaluations.

3.1 Data Source

Dataset used in this study '0_VED_Orig_data.csv' has been taken from the public source Github by Mr Linas P. He is an associate professor Vilnius University, Lithuania. This data source is available on the following link.Google Drive Link: `https://drive.google.com/drive/folders/1NxGQzGXARK7qCSMHlsuL-0vrl-0itisL`

3.2 Importing Dataset

Figure 1 represents the code block for importing the dataset for the study. This dataset can be imported locally as well using the pandas library. In the case of google colab, google drive can be mounted in the notebook and then the dataset can be imported from the directories of the drive.

```
#this code block will help in mounting the google drive to this notebook, if the file is stored in the google drive directory,
#then the file can be imported here.

from google.colab import drive
drive.mount('/content/drive')

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

```
df = pd.read_csv('/content/drive/MyDrive/Thesis Dataset/0_VED_orig_data.csv')

#If the file is stored in a local directory, then use the following code.
#df = pd.read_csv('0_VED_orig_data.csv').
```

Figure 1: Importing Dataset

3.3 Data Preprocessing

3.4 Data Transformation

3.5 Data Mining

3.6 Result Evaluation

References

```
] # Setting the seed value to get the reproducible results.

  from numpy.random import seed
  seed(100)
```

```
] #Removing unnamed column from the dataframe

  df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
```

```
] #Checking all the columns of the dataset

  df.columns

  Index(['date', 'datetime', 'daynum', 'vehid', 'trip', 'timestamp_ms', 'lat',
         'lon', 'speed_kmh', 'maf_gsec', 'engine_rpm', 'absoluteload',
         'oat_degc', 'fuelrate_lhr', 'airconditioning_kw', 'airconditioning_w',
         'heaterpower_w', 'hvbattery_a', 'hvbattery_soc_per', 'hvbattery_v',
         'shorttermfueltrimbank1_pct', 'shorttermfueltrimbank2_pct',
         'longtermfueltrimbank1_pct', 'longtermfueltrimbank2_pct',
         'sub_trip_gid', 'sub_trip', 'mm_edge_id', 'mm_direction',
         'mm_edge_source_h', 'mm_edge_target_h', 'mm_edge_km', 'mm_edge_kmh',
         'mm_edge_clazzs', 'mm_edge_frcalong', 'mm_score', 't1', 't2',
         'heavyfog', 'hourlydewpointtemperature', 'hourlydrybulbtemperature',
         'hourlyprecipitation', 'hourlypresentweathertype',
         'hourlyrelativehumidity', 'hourlyskyconditions', 'hourlyvisibility',
         'hourlywetbulbtemperature', 'hourlywinddirection', 'hourlywindspeed',
         'tstorms'],
```

Figure 2: Data Preprocessing

```python
#Checking the null values in the dataframe.

df.isna().sum()
```

```
date                              0
datetime                          0
daynum                            0
vehid                             0
trip                              0
timestamp_ms                      0
lat                               0
lon                               0
speed_kmh                         0
maf_gsec                     408058
engine_rpm                   408058
absoluteload                 408058
oat_degc                          0
fuelrate_lhr                 408058
airconditioning_kw           408058
airconditioning_w                 0
heaterpower_w                     0
hvbattery_a                       0
hvbattery_soc_per                 0
hvbattery_v                       0
shorttermfueltrimbank1_pct   408058
shorttermfueltrimbank2_pct   408058
longtermfueltrimbank1_pct    408058
longtermfueltrimbank2_pct    408058
sub_trip_gid                      0
sub_trip                          0
mm_edge_id                        0
mm_direction                   1915
mm_edge_source_h                  0
mm_edge_target_h                  0
mm_edge_km                        0
mm_edge_kmh                       0
mm_edge_clazzs                    0
```

Figure 3: Null Values

```
#counting the total unique values of mm_edge_id

mm = np.unique(df['mm_edge_id'])
len(mm)
```

```
2131
```

```
#counting the total unique values of trip so that we can know how many trips have been captured.

n_trip = np.unique(df['trip'])
len(n_trip)
```

```
482
```

```
#counting the total unique values of sub_trip

df.sub_trip.nunique()
```

```
56
```

```
#grouping only with trip and mm_edge_id to create a new dataframe which has all the trips with their mm_edeg_id.
#This will help in calculating the average speed, travel time and energy cosnumed in a single trip.

grouped_df = df.groupby(["trip", "mm_edge_id"])
vals = grouped_df.first()
vals = vals.reset_index()
vals.shape
```

```
(19348, 49)
```

Figure 4: Data Preprocessing

```
#Checking the new dataframe

vals.head()
```

	trip	mm_edge_id	date	datetime	daynum	vehid	timestamp_ms	lat	lon	speed_kmh	...	hourlydrybulbtemperature	hourlyprecipitation	hourlypresentweathertype	hou
0	554	161632	2017-11-05	2017-11-05 16:45:00	4.698313	455	92900	42.244299	-83.732130	63.349998	...	59.0	0.01	None	
1	554	161633	2017-11-05	2017-11-05 16:45:00	4.698313	455	87900	42.244276	-83.733168	64.809998	...	59.0	0.01	None	
2	554	161636	2017-11-05	2017-11-05 16:45:00	4.698313	455	57900	42.243874	-83.739069	50.369999	...	59.0	0.01	None	
3	554	552278	2017-11-05	2017-11-05 16:45:00	4.698313	455	107900	42.244407	-83.729035	61.669998	...	59.0	0.01	None	
4	554	552279	2017-11-05	2017-11-05 16:45:00	4.698313	455	112900	42.244633	-83.728136	54.059998	...	59.0	0.01	None	

5 rows x 49 columns

Figure 5: Data Preprocessing- Vals

```python
#since there are timestamp values in the column of the dataset, a function is created in order to get the calculate the time (in seconds)
#between the trips. This can help in calculating the readings after each trips.

def find_time(df):

    t_max = np.max(pd.to_datetime(df['timestamp_ms']).astype('int64'))
    t_min = np.min(pd.to_datetime(df['timestamp_ms']).astype('int64'))
    dif = (t_max - t_min) / 1e3

    return(dif)
```

```python
#Battery reading after every timestamp.

df['hvbattery_soc_per']
```

```
0         96.341469
1         96.341469
2         96.341469
3         96.341469
4         96.341469
            ...
408053    59.024395
408054    59.024395
408055    59.024395
408056    59.024395
408057    59.024395
Name: hvbattery_soc_per, Length: 408058, dtype: float64
```

Figure 6: Data Preprocessing Find_time()

```python
] #This function will find the difference between each readings of hvbattery_soc_per so that energy consumed between/in the trip can be calculated.
  #hvbattery_soc_per is showing the readings, this function can let us know the energy consumed after certain time.

  def find_energy(df):

      t_max = np.max(df['hvbattery_soc_per'])
      t_min = np.min(df['hvbattery_soc_per'])
      dif = (t_max - t_min)
      return(dif)
```

```python
] #this code cell helps in determining the time consumed in a single trip and in a mm_edge_id using teh find_time() created in the above cell.

  time = df[["trip", "mm_edge_id",'timestamp_ms']].groupby(["trip", "mm_edge_id"]).apply(find_time)
```

Figure 7: Data Preprocessing-find_energy()

```
#this code cell helps in determining the average speed of the vehicle in a single trip and in a mm_edge_id.


speed = df[["speed_kmh", "trip", "mm_edge_id"]].groupby(["trip", "mm_edge_id"]).apply(np.mean)
```

```
#this code cell helps in determining the energy cnsumed in by the vehicle in a single trip and in a mm_edge by subtracting the maximum value to minumum value
#in a single trip and mm_edge_id.


df[["trip", "mm_edge_id",'hvbattery_soc_per']].groupby(["trip", "mm_edge_id"])
energy = df[["trip", "mm_edge_id",'hvbattery_soc_per']].groupby(["trip", "mm_edge_id"]).apply(find_energy)
```

Figure 8: Data Preprocessing-Energy Calculation

```
]
#vals dataset is created because we want to have only the average speed, time and energy consumed in a single trip. hence there is reduction in a shape
#of the original dataset.

vals['speed'] = speed['speed_kmh'].to_numpy()
vals['time'] = time.to_numpy()
vals['ev_kwh'] = energy.to_numpy()
```

```
] #printing the 'speed' column

vals['speed']
```

```
0        63.838749
1        63.888748
2        51.914284
3        59.355712
4        25.929166
           ...
19343    35.572857
19344    17.258799
19345    63.937999
19346    37.288749
19347    58.289522
Name: speed, Length: 19348, dtype: float64
```

Figure 9: Data Preprocessing

```
#importing the library to plot the histograms and other visualisation

import matplotlib.pyplot as plt
```

```
#plotting the histogram for the time column.

plt.hist(vals['time'])
```

```
(array([1.9034e+04, 8.2000e+01, 6.7000e+01, 7.0000e+01, 4.4000e+01,
        3.9000e+01, 1.0000e+01, 1.0000e+00, 0.0000e+00, 1.0000e+00]),
 array([   0.  ,  277.42,  554.84,  832.26, 1109.68, 1387.1 , 1664.52,
        1941.94, 2219.36, 2496.78, 2774.2 ]),
 <a list of 10 Patch objects>)
```

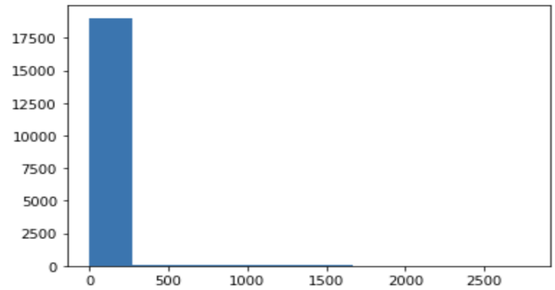

Figure 10: Distribution of Time

```
#creating new dataset without null value columns

vals2 = vals[['trip', 'mm_edge_id', 'lat', 'lon', 'date', 'datetime', 'daynum',
       'vehid', 'timestamp_ms', 'speed_kmh','oat_degc','airconditioning_w', 'heaterpower_w', 'hvbattery_a',
       'hvbattery_soc_per', 'hvbattery_v','sub_trip_gid', 'sub_trip','mm_direction',
       'mm_edge_source_h', 'mm_edge_target_h', 'mm_edge_km', 'mm_edge_kmh',
       'mm_edge_clazzs', 'mm_edge_frcalong', 'mm_score', 't1', 't2','hourlydewpointtemperature','hourlydrybulbtemperature','hourlyprecipitation',
       'hourlyrelativehumidity','hourlyskyconditions','hourlyvisibility','hourlywetbulbtemperature','hourlywinddirection','hourlywindspeed',
       'speed','time','distance','ev_kwh'
       ]]
```

Figure 11: Data Preprocessing- Vals2

```
#Transforming the 'mm_edge_clazzs' to provide only the type of the class. Strings after 'highway.' represents the type of route or path on which vehicle is moving

vals3['mm_edge_clazzs'] = vals3['mm_edge_clazzs'].str.replace('highway.','')
```

```
<ipython-input-46-9945de8c1ca3>:3: FutureWarning: The default value of regex will change from True to False in a future version.
  vals3['mm_edge_clazzs'] = vals3['mm_edge_clazzs'].str.replace('highway.','')
<ipython-input-46-9945de8c1ca3>:3: 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
  vals3['mm_edge_clazzs'] = vals3['mm_edge_clazzs'].str.replace('highway.','')
```

Figure 12: Data Preprocessing-'mm_edge_clazzs'

```
#calculating distance and ev_kwh for a single trip for the EDA purpose.


vals_grp = vals3.groupby(["trip"])['distance','ev_kwh'].sum()
print(vals_grp)
```

```
        distance     ev_kwh
trip
554     3.745929   0.975605
565     2.966467   0.731720
568    35.261814   3.048790
575   155.432368  12.073158
588    97.200116   9.878036
...          ...        ...
3223   49.382303   4.634144
3229    1.490552   0.731712
3234   29.064058   7.317078
3263   55.462525   4.878048
3271    3.946420   0.243904
```

Figure 13: Distance and Energy Consumption

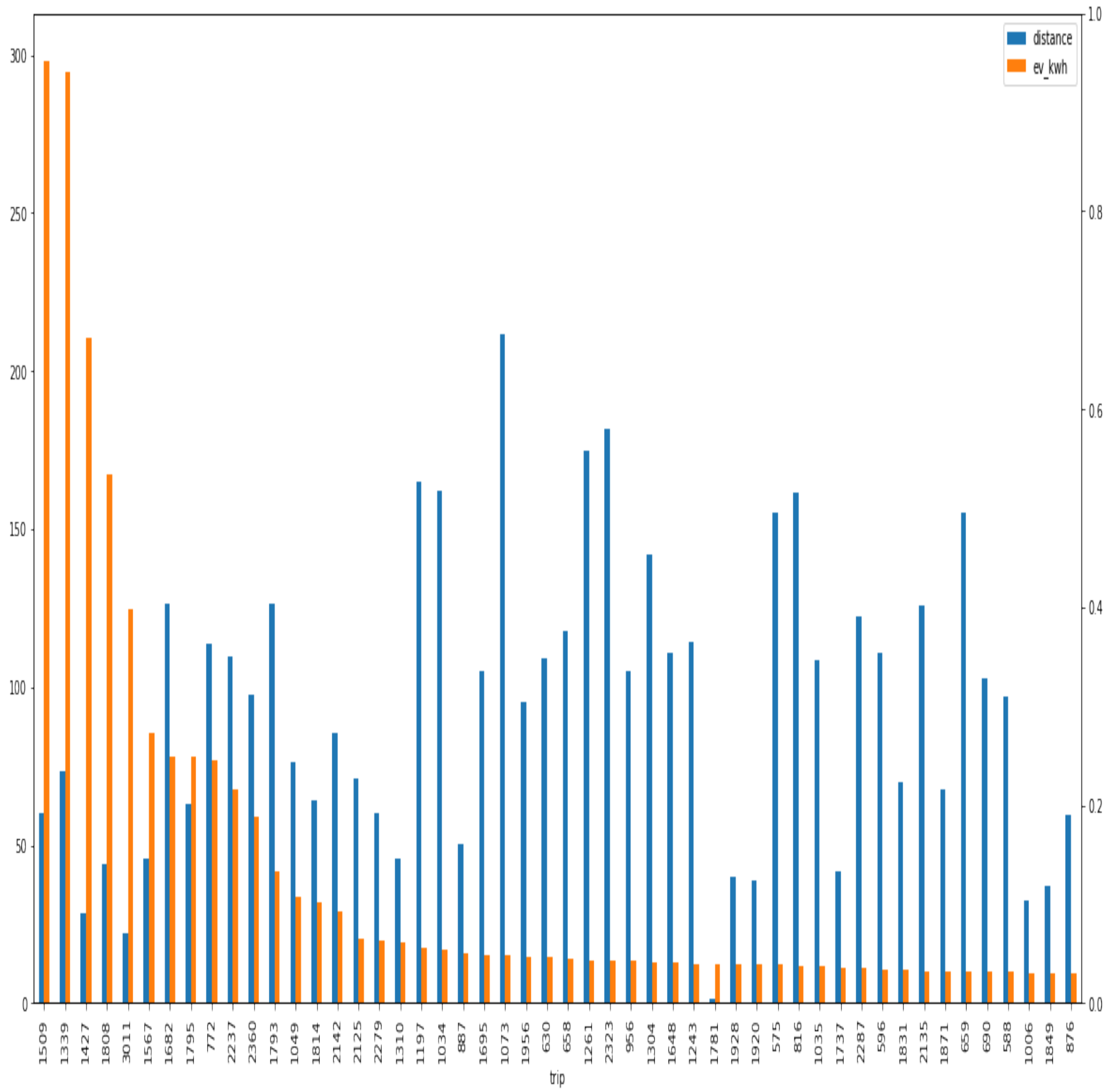

Figure 14: Top 50 Energy consumption trips

```
] #Dropping all other columns which have lower correlational values with ev_kwh and creating a new dataframe vals4

vals4 = vals3.drop(['lat', 'lon', 'date', 'datetime', 'daynum','hourlydewpointtemperature',
        'vehid', 'timestamp_ms', 'speed_kmh'],axis =1)
```

Figure 15: Data Transformation

```
#converting the values from watt to kwatt

vals4[['airconditioning_w','heaterpower_w']] = vals4[['airconditioning_w','heaterpower_w']]/1000
```

Figure 16: Data Transformation Heater and Air Conditioner

```python
#plotting histogram for understanding the distribution of different class of the mm_edge_id.

plt.figure(figsize=(15,5))
plt.hist(vals4['mm_edge_clazzs'])
```

```
(array([7.559e+03, 7.194e+03, 1.249e+03, 9.000e+01, 4.000e+01, 1.180e+02,
        6.960e+02, 1.070e+02, 1.200e+01, 2.000e+00]),
 array([0. , 0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2, 8.1, 9. ]),
 <a list of 10 Patch objects>)
```

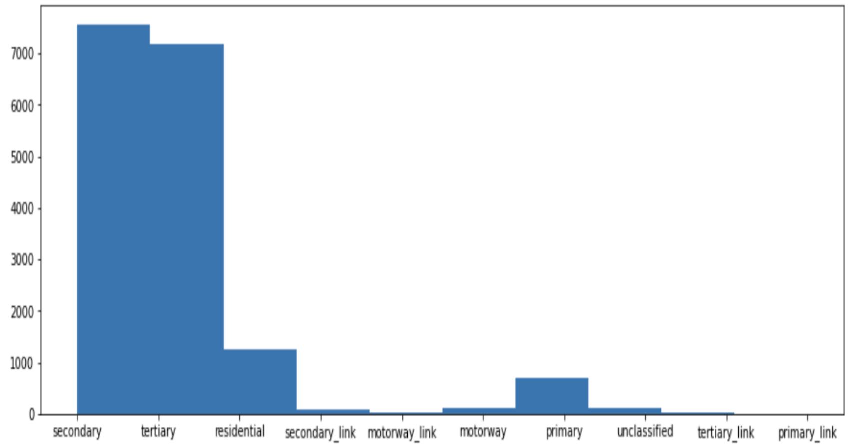

Figure 17: Histograme for Class of the Path

```python
#importing the OneHotEncoder for transforming the categorical columns.

from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder()
```

```python
#transforming the categorical column mm_edge_clazzs into categorical columns.

encoder_mm_edge_clazzs = pd.DataFrame(onehotencoder.fit_transform(vals4[['mm_edge_clazzs']]).toarray())
```

```python
#printing the transformed dataframe.

encoder_mm_edge_clazzs
```

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
17062	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
17063	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
17064	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
17065	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
17066	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

17067 rows × 10 columns

Figure 18: One Hot Encoding on mm_edge_clazzs

```python
#Joining original vals4 with new categorical dataframe and creating a new vals5 for better processing.

vals5 = vals4.join(encoder_mm_edge_clazzs)
```

```python
#Printing vals5 dataframe.

vals5.head()
```

	trip	mm_edge_id	oat_degc	airconditioning_w	heaterpower_w	hvbattery_a	hvbattery_soc_per	hvbattery_v	sub_trip_gid	sub_trip	...	0	1	2	3	4	5	6	7	8	9
1	554	161633	11.5	0.0	0.0	-67.0	48.414635	365.0	1428	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	554	161636	11.5	0.0	0.0	-59.5	48.414635	367.5	1428	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	554	552278	11.5	0.4	0.0	30.5	47.682930	376.5	1428	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	554	552279	11.5	0.4	0.0	30.0	47.682930	373.0	1428	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
5	554	583267	11.5	0.0	0.0	-25.0	48.414635	370.0	1428	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows x 42 columns

Figure 19: vals 5

```python
#dropping the columns whihc will not have contribution in the prediction operation

vals6 = vals5.drop(['trip','mm_edge_id','hvbattery_a',
            'hvbattery_soc_per','hvbattery_v','sub_trip_gid','sub_trip','mm_edge_source_h','mm_edge_target_h'
            , 'mm_edge_frcalong','mm_score','t1','t2','hourlyskyconditions'],axis = 1)
```

Figure 20: Data Transformation

```
#dropping rest of the columns

vals7 = vals6.drop(['mm_edge_kmh','mm_edge_clazzs','hourlywinddirection'],axis =1)
```

```
#converting the T (trace values) to zero.

vals7['hourlyprecipitation'] = vals7['hourlyprecipitation'].replace('T',0.0)
vals7['hourlyprecipitation'] = vals7['hourlyprecipitation'].astype('float')
```

```
#Converting the datatype of the columns

vals7['hourlyvisibility'] = vals7['hourlyvisibility'].astype(float)
print(vals7['hourlyvisibility'])
```

```
1        10.0
2        10.0
3        10.0
4        10.0
5        10.0
         ...
19343    10.0
19344    10.0
19345    10.0
19346    10.0
19347    10.0
Name: hourlyvisibility, Length: 17067, dtype: float64
```

```
#creating a new dataframe 'dataset' which is a copy of vals8

dataset = vals7.copy()
```

Figure 21: Data Transformation for Hourly Precipitation

```
] #removing all the 0 valued rows from the time columns so that we have rows which have some travelling time.

  dataset = dataset[dataset['time'] !=0]
```

```
] #checking the new shape of the dataset

  dataset.shape
```

```
(17032, 25)
```

Figure 22: Removing Zero-Valued rows

```python
#dropping all the null values.

dataset = dataset.dropna()
```

```python
#printing the description of the dataset.

dataset[['speed','time','distance','ev_kwh','oat_degc','airconditioning_w','heaterpower_w','mm_direction','mm_edge_km', 'hourlydrybulbtemperature',
         'hourlyprecipitation',   'hourlyrelativehumidity']].describe()
```

	speed	time	distance	ev_kwh	oat_degc	airconditioning_w	heaterpower_w	mm_direction	mm_edge_km	hourlydrybulbtemperature	hourlyprecipitation	h
count	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	15337.000000	
mean	46.047164	26.286477	0.904910	0.197474	10.178294	0.353179	0.404626	0.210928	0.104006	48.529634	0.002774	
std	18.417525	127.185682	1.323194	1.021234	11.370769	0.363775	0.846311	0.977534	0.111399	19.457025	0.016034	
min	0.000000	0.200000	0.009514	0.000000	-15.500000	0.000000	0.000000	-1.000000	0.003448	-12.000000	0.000000	
25%	33.640908	2.900000	0.094774	0.000000	2.000000	0.000000	0.000000	-1.000000	0.048378	34.000000	0.000000	
50%	47.792855	5.800000	0.318273	0.000000	8.000000	0.300000	0.000000	1.000000	0.072998	45.000000	0.000000	
75%	59.893748	11.900000	1.157310	0.000000	20.500000	0.500000	0.500000	1.000000	0.116420	65.000000	0.000000	
max	122.076663	1876.200000	10.508275	26.463417	34.000000	2.300000	4.250000	1.000000	1.951085	92.000000	0.220000	

Figure 23: Cleaned Dataset

```python
#creating jointplot for understanding the distribution of speed and energy.

sns.jointplot(x ='speed', y ='ev_kwh', data = dataset)
```

```
<seaborn.axisgrid.JointGrid at 0x7fbd92c947f0>
```

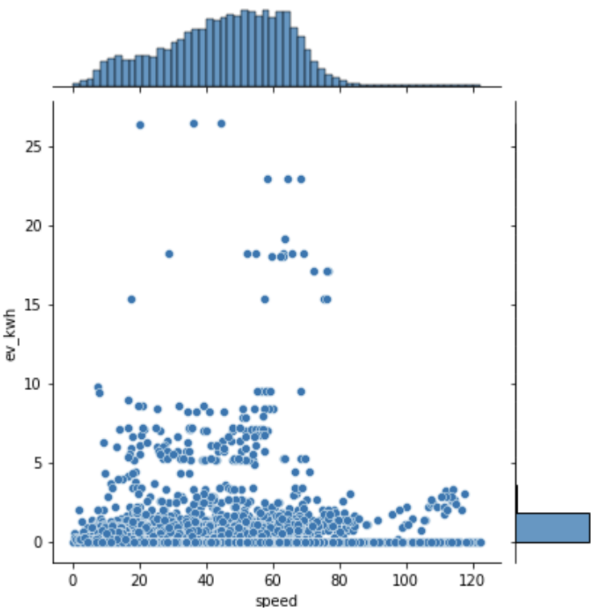

Figure 24: Speed Vs Energy Consumption

```
#creating jointplot for understanding the distribution of distance and energy.

sns.jointplot(x ='distance', y ='ev_kwh', data = dataset)
```

```
<seaborn.axisgrid.JointGrid at 0x7fbd92bef580>
```

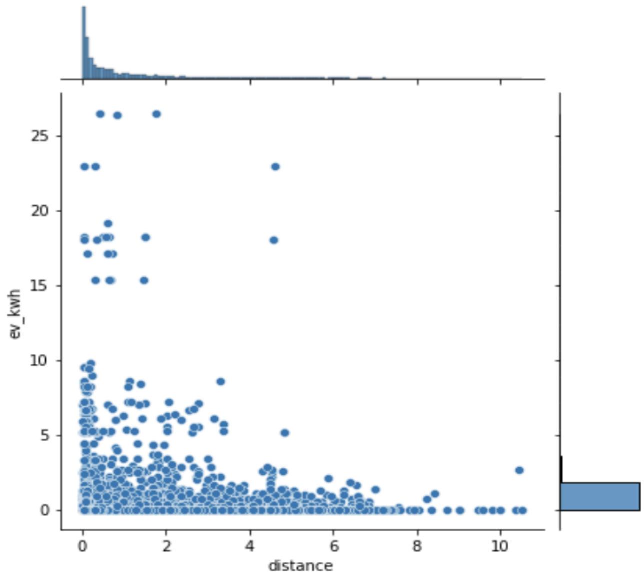

Figure 25: Distance Vs Energy Consumption

```
#Storing all the independent columns in X.

X = dataset.drop(['ev_kwh'],axis =1 )
```

```
#Storing the target variable ev_kwh in y.

y = dataset['ev_kwh'].values
```

```
#Importing the train_test_split from sklearns library.

from sklearn.model_selection import train_test_split
```

```
#splitting the dataset in train, validation and test set for measuring the performance of the model.

X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8)
X_val, X_test, y_val, y_test = train_test_split(X_rem,y_rem, test_size=0.5)
```

Figure 26: X and y Datasets

```
#Printing subset of all the sub sets.

print(X_train.shape), print(y_train.shape)
print(X_val.shape), print(y_val.shape)
print(X_test.shape), print(y_test.shape)
```

```
(12269, 24)
(12269,)
(1534, 24)
(1534,)
(1534, 24)
(1534,)
(None, None)
```

```
#standardising all the subsets for the model training.

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.fit_transform(X_val)
X_test = scaler.fit_transform(X_test)
```

Figure 27: Data Transformation using StandardScalar Technique

```
#setting the seed for reproducible results

import tensorflow as tf
tf.random.set_seed(10)
```

```
#importing the keras libraries for model training.

from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

Figure 28: Importing Libraries

```
#initialising the sequential class for the training.

model = Sequential()
```

```
#adding the dense layers in the model. activation set to relu.

model.add(Dense(X_train.shape[1],activation='relu'))
model.add(Dense(64,activation='relu'))
```

```
model.add(Dense(64,activation='relu'))

model.add(Dense(128,activation='relu'))

model.add(Dense(1))
```

Figure 29: Adding layers to the Model

```
#Fitting the model and training it with 1500 epochs

r = model.fit(X_train, y_train,
              validation_data=(X_val,y_val),
              batch_size=64,
              epochs=1500)
```

```
Epoch 1452/1500
192/192 [==============================] - 1s 5ms/step - loss: 0.0791 - val_loss: 0.1294
Epoch 1453/1500
192/192 [==============================] - 1s 5ms/step - loss: 0.0608 - val_loss: 0.1432
Epoch 1454/1500
192/192 [==============================] - 1s 7ms/step - loss: 0.0597 - val_loss: 0.1469
Epoch 1455/1500
192/192 [==============================] - 1s 6ms/step - loss: 0.0596 - val_loss: 0.1556
Epoch 1456/1500
192/192 [==============================] - 1s 5ms/step - loss: 0.0634 - val_loss: 0.1964
Epoch 1457/1500
192/192 [==============================] - 1s 6ms/step - loss: 0.0704 - val_loss: 0.2269
Epoch 1458/1500
[redacted]s: 0.0697 - val_loss: 0.1397
```

Figure 30: Model Training Process

```
#Visualising training and validation loss

plt.figure(figsize=(10, 6))

plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8834dbac10>
```

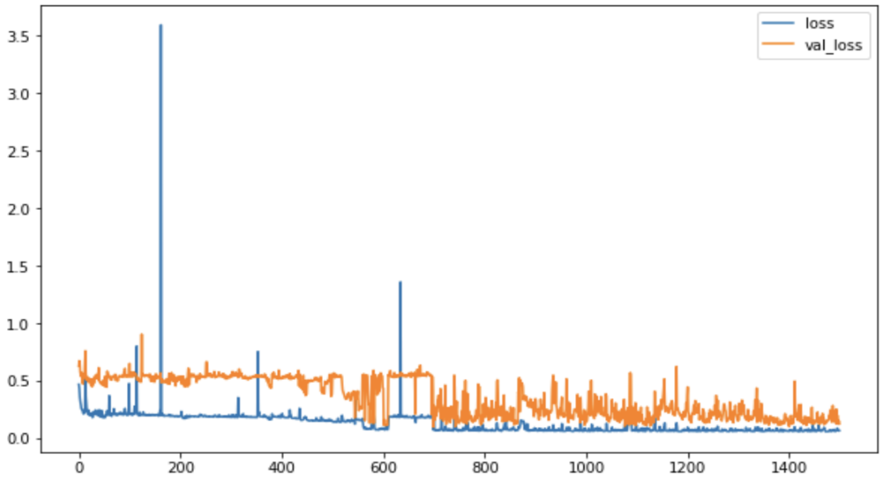

Figure 31: Training and Validation loss

```
print_evaluate(y_train, y_train_pred, 'train')
print_evaluate(y_val, y_val_pred, 'validation')
print_evaluate(y_test, y_test_pred, 'test')
```

```
========Training Result=======
MAE:  0.12969676279441741
MSE:  0.05896567838612335
RMSE:  0.2428284958280707
R2 Square:  0.9433114353373161
=========Validation Result=======
MAE:  0.16670457885253023
MSE:  0.12524125166334157
RMSE:  0.35389440750503753
R2 Square:  0.8961316324498286
========Testing Result=======
MAE:  0.16399552538073225
MSE:  0.09942433507042563
RMSE:  0.31531624612510156
R2 Square:  0.8896393243397365
```

Figure 32: Final Results