route_planner

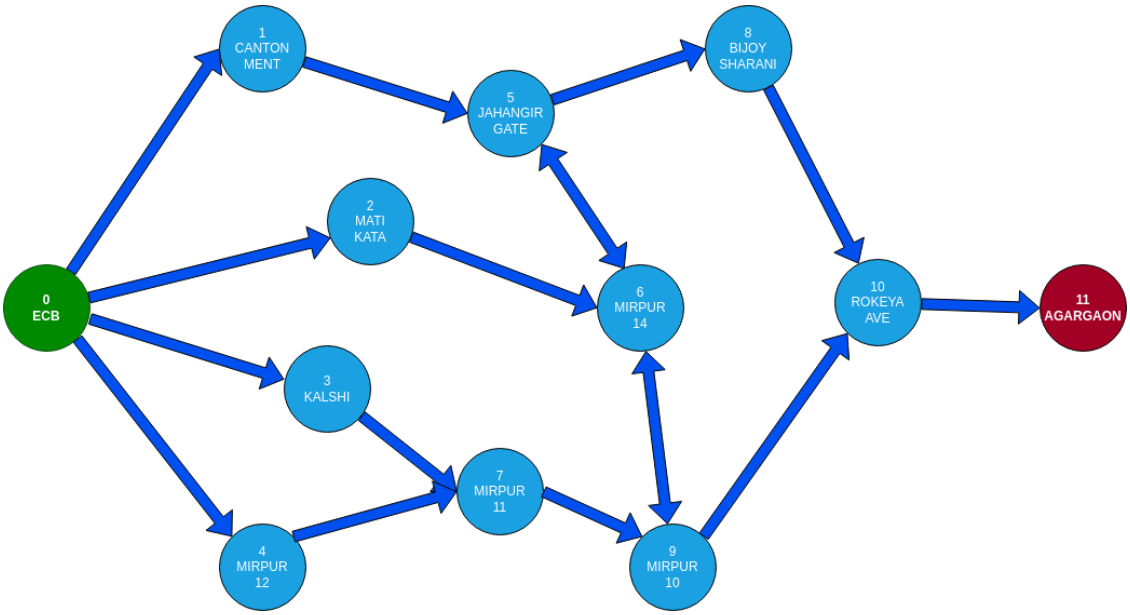
June 24, 2023

1 IMPORT

IMPORTING REQUIRED MODULE

[122]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from  sklearn import preprocessing
import matplotlib.pyplot as plt
import numpy as np
from sklearn.inspection import permutation_importance
from IPython.display import Image, display
from queue import PriorityQueue
import datetime
import random
from sklearn.metrics import r2_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import max_error
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
```

2 MAP



MAP OF OPERATIONAL AREA

3 DATASET

IMPORTING DATASET

[123]:
```
traffic_data = pd.read_csv('/home/rafi/cse404/project/traffic_dataset.csv')
```

[124]:
```
traffic_data
```

[124]:

	location	month	day	date	time_block	weather	traffic_indicator
0	begum_rokeya	apr	fri	1	a	1	3
1	begum_rokeya	apr	fri	1	b	1	3
2	begum_rokeya	apr	fri	1	c	1	3
3	begum_rokeya	apr	fri	1	d	1	3
4	begum_rokeya	apr	fri	1	e	1	3
…	…	…	…	…	…	…	…
87655	mirpur_14	sep	wed	30	b	3	2
87656	mirpur_14	sep	wed	30	c	3	2
87657	mirpur_14	sep	wed	30	d	3	2
87658	mirpur_14	sep	wed	30	e	3	3
87659	mirpur_14	sep	wed	30	f	3	3

[87660 rows x 7 columns]

[125]:
```
colN = ['location', 'month', 'day', 'date', 'time_block', 'weather',␣
↪'traffic_indicator']
```

ASSIGNING FEATURE AND TARGET

[126]:
```
X=traffic_data.drop(columns=['traffic_indicator'])
Y=traffic_data['traffic_indicator']
```

[127]:
```
Y
```

[127]:
```
0        3
1        3
2        3
3        3
4        3
        ..
87655    2
87656    2
87657    2
87658    3
87659    3
Name: traffic_indicator, Length: 87660, dtype: int64
```

[128]:
```
X
```

[128]:

	location	month	day	date	time_block	weather
0	begum_rokeya	apr	fri	1	a	1
1	begum_rokeya	apr	fri	1	b	1
2	begum_rokeya	apr	fri	1	c	1
3	begum_rokeya	apr	fri	1	d	1
4	begum_rokeya	apr	fri	1	e	1
…	…	…	…	…	…	…
87655	mirpur_14	sep	wed	30	b	3
87656	mirpur_14	sep	wed	30	c	3
87657	mirpur_14	sep	wed	30	d	3
87658	mirpur_14	sep	wed	30	e	3
87659	mirpur_14	sep	wed	30	f	3

[87660 rows x 6 columns]

DATA PREPROCESSING

[129]:
```
le=preprocessing.LabelEncoder()
```

[130]:
```
encoded_location = le.fit_transform(traffic_data['location'])
encoded_month = le.fit_transform(traffic_data['month'])
encoded_day = le.fit_transform(traffic_data['day'])
encoded_date = le.fit_transform(traffic_data['date'])
encoded_time_block = le.fit_transform(traffic_data['time_block'])
encoded_weather = le.fit_transform(traffic_data['weather'])
encoded_traffic_indicator = le.fit_transform(traffic_data['traffic_indicator'])
```

[131]:
```
X = np.array(list(zip(encoded_location, encoded_month, encoded_day,␣
↪encoded_date, encoded_time_block, encoded_weather)))
Y = encoded_traffic_indicator
```

SPLITTING INTO TRAIN & TEST

[132]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
```

4 DECISION TREE REGRESSOR

MODEL SELECTION

[133]:
```
model =DecisionTreeRegressor()
```

MODEL TRAIN - GRADIENT BOOSTING REGRESSOR

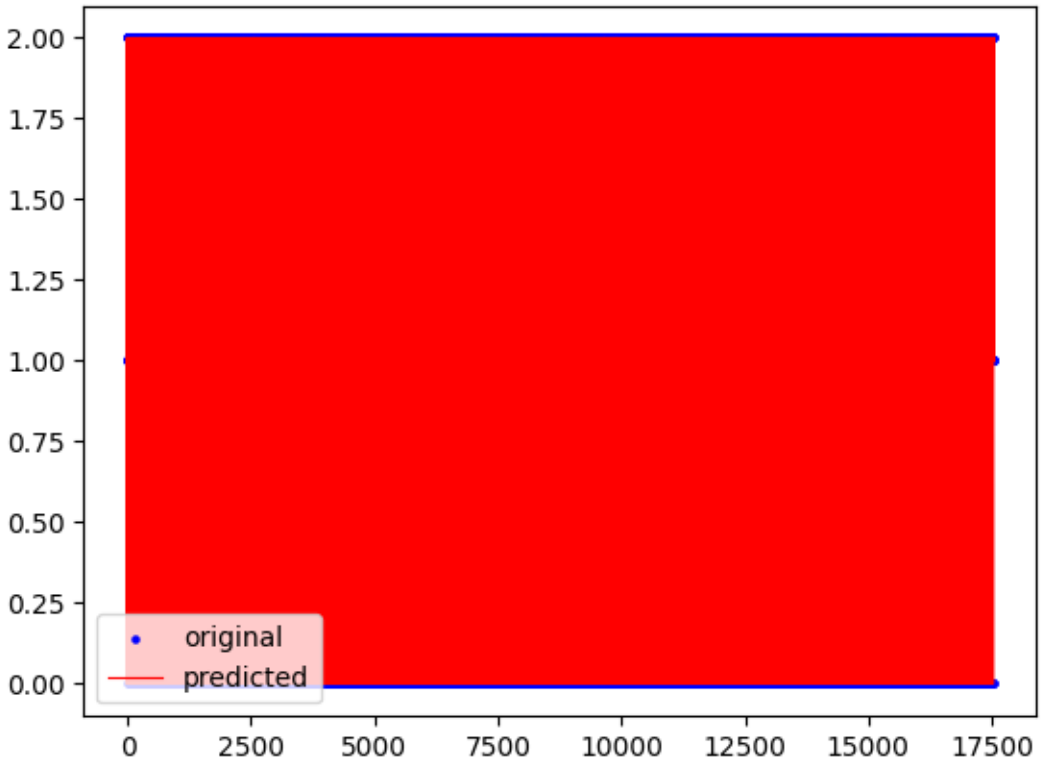
[134]:
```
model.fit(X_train, Y_train)
```

[134]:
```
DecisionTreeRegressor()
```

5 TESTING & EVALUATION

[135]:
```
prediction = model.predict(X_test)
```

[136]:
```
x_ax = range(len(Y_test))
plt.scatter(x_ax, Y_test, s=5, color="blue", label="original")
plt.plot(x_ax, prediction, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



MEAN SQUARE ERROR

[137]:
```
dt_mse = mean_squared_error(Y_test, prediction)
```

[138]:
```
dt_mse
```

[138]:
```
0.11168149669176364
```

R2 SCORE (COEFFICIENT OF DETERMINATION)

[139]:
```
print('Test R^2     : %.3f'%r2_score(Y_test, prediction))
print('Test R^2     : %.3f'%model.score(X_test, Y_test))
print('Training R^2 : %.3f'%model.score(X_train, Y_train))
dt_r2 = model.score(X_test, Y_test)
```

```
Test R^2     : 0.749
Test R^2     : 0.749
Training R^2 : 1.000
```

MEAN ABSOLUTE ERROR (MAE)

[140]:
```
print('Test MAE  : %.3f'%mean_absolute_error(Y_test, prediction))
print('Train MAE : %.3f'%mean_absolute_error(Y_train, model.predict(X_train)))
```

```
dt_mae = mean_absolute_error(Y_test, prediction)
```

```
Test MAE  : 0.111
Train MAE : 0.000
```

MEAN SQUARED LOG ERROR

[141]:
```
print("Mean Squared Log Error : {:.3f}".format(mean_squared_log_error(Y_test,␣
 ↪prediction)))
dt_msle = mean_squared_log_error(Y_test, prediction)
```

```
Mean Squared Log Error : 0.038
```

MEDIAN ABSOLUTE ERROR

[142]:
```
print('Median Absolute Error : {}'.format(median_absolute_error(Y_test,␣
 ↪prediction)))
dt_mad = median_absolute_error(Y_test, prediction)
```

```
Median Absolute Error : 0.0
```

MAXIMUM RESIDUAL ERROR

[143]:
```
print('Maximum Residual Error : {:.3f}'.format(max_error(Y_test, prediction)))
dt_mre = max_error(Y_test, prediction)
```

```
Maximum Residual Error : 2.000
```

EXPLAINED VARIANCE ERROR

[144]:
```
print('Explained Variance Error : {:.3f}'.
 ↪format(explained_variance_score(Y_test, prediction)))
dt_eve = explained_variance_score(Y_test, prediction)
```

```
Explained Variance Error : 0.749
```

6 RANDOM FOREST REGRESSOR

MODEL SELECTION

[145]:
```
model = RandomForestRegressor()
```

MODEL TRAIN - GRADIENT BOOSTING REGRESSOR

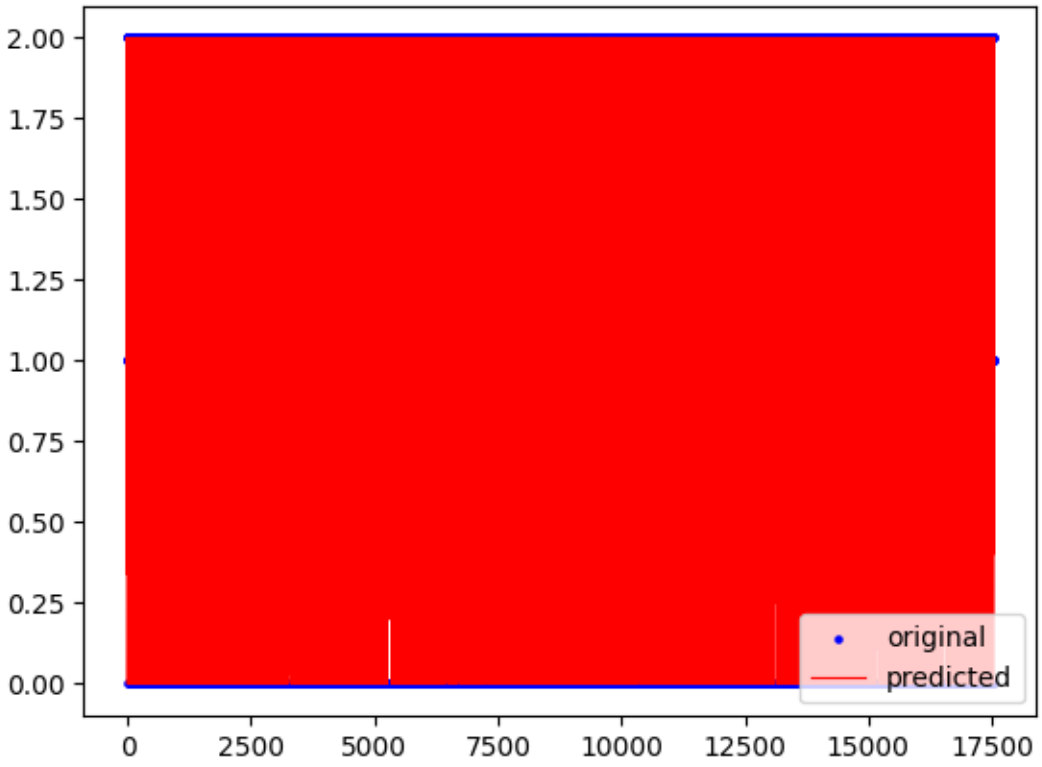
[146]:
```
model.fit(X_train, Y_train)
```

[146]:
```
RandomForestRegressor()
```

7 TESTING & EVALUATION

[147]:
```
prediction = model.predict(X_test)
```

[148]:
```
x_ax = range(len(Y_test))
plt.scatter(x_ax, Y_test, s=5, color="blue", label="original")
plt.plot(x_ax, prediction, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



[149]:
```
contigency = [1, 2, 3]
con = lambda : random.choice(contigency)
```

MEAN SQUARE ERROR

[150]:
```
mse = mean_squared_error(Y_test, prediction)
```

[151]:
```
mse
```

[151]: 0.06577015742642026

R2 SCORE (COEFFICIENT OF DETERMINATION)

[152]:
```
print('Test R^2     : %.3f'%r2_score(Y_test, prediction))
print('Test R^2     : %.3f'%model.score(X_test, Y_test))
print('Training R^2 : %.3f'%model.score(X_train, Y_train))
```

```
Test R^2     : 0.852
Test R^2     : 0.852
Training R^2 : 0.979
```

MEAN ABSOLUTE ERROR (MAE)

[153]:
```
print('Test MAE  : %.3f'%mean_absolute_error(Y_test, prediction))
print('Train MAE : %.3f'%mean_absolute_error(Y_train, model.predict(X_train)))
```

```
Test MAE  : 0.110
Train MAE : 0.041
```

MEAN SQUARED LOG ERROR

[154]:
```
print("Mean Squared Log Error : {:.3f}".format(mean_squared_log_error(Y_test,␣
↪prediction)))
```

```
Mean Squared Log Error : 0.023
```

MEDIAN ABSOLUTE ERROR

[155]:
```
print('Median Absolute Error : {}'.format(median_absolute_error(Y_test,␣
↪prediction)))
```

```
Median Absolute Error : 0.0
```

MAXIMUM RESIDUAL ERROR

[156]:
```
print('Maximum Residual Error : {:.3f}'.format(max_error(Y_test, prediction)))
```

```
Maximum Residual Error : 1.400
```

EXPLAINED VARIANCE ERROR

[157]:
```
print('Explained Variance Error : {:.3f}'.
↪format(explained_variance_score(Y_test, prediction)))
```

```
Explained Variance Error : 0.852
```

[158]:
```
rf_mse = mean_squared_error(Y_test, prediction)
rf_r2 = model.score(X_test, Y_test)
rf_mae = mean_absolute_error(Y_test, prediction)
rf_msle = mean_squared_log_error(Y_test, prediction)
rf_mad = median_absolute_error(Y_test, prediction)
rf_mre = max_error(Y_test, prediction)
rf_eve = explained_variance_score(Y_test, prediction)
```

8 LINEAR REGRESSOR

MODEL SELECTION

[159]:
```
model = LinearRegression()
```

MODEL TRAIN - GRADIENT BOOSTING REGRESSOR

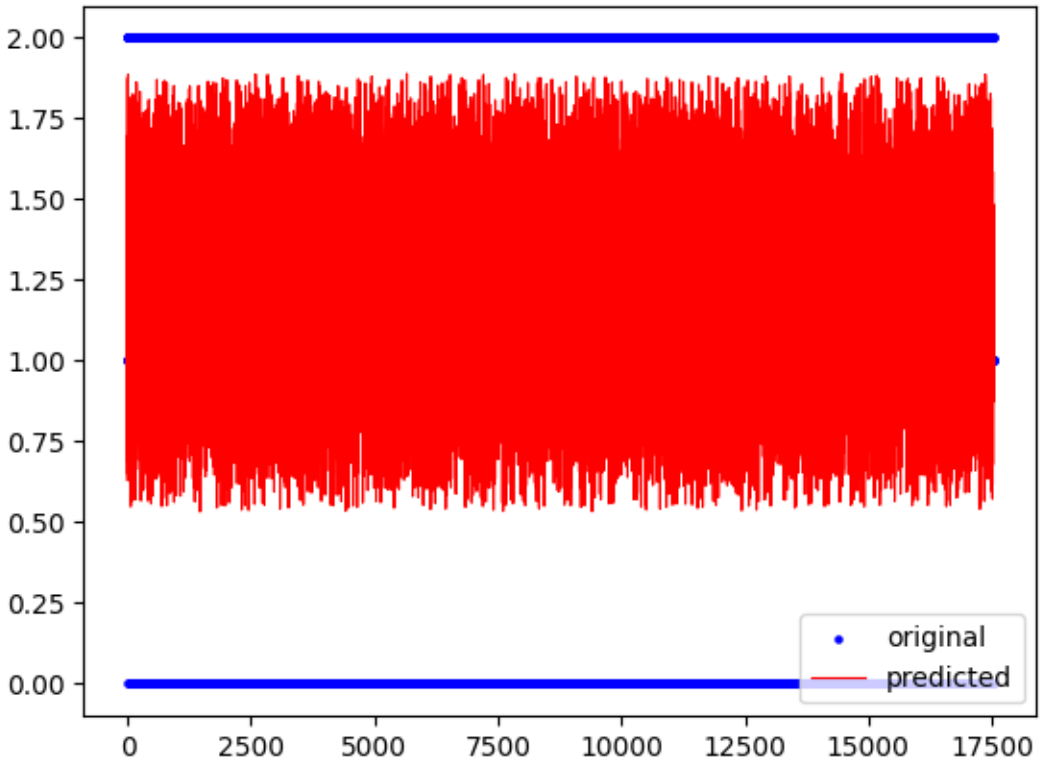
[160]:
```
model.fit(X_train, Y_train)
```

[160]:
```
LinearRegression()
```

9 TESTING & EVALUATION

[161]:
```
prediction = model.predict(X_test)
```

[162]:
```
x_ax = range(len(Y_test))
plt.scatter(x_ax, Y_test, s=5, color="blue", label="original")
plt.plot(x_ax, prediction, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



[163]:
```
contigency = [1, 2, 3]
con = lambda : random.choice(contigency)
```

MEAN SQUARE ERROR

[164]:
```
mse = mean_squared_error(Y_test, prediction)
```

[165]:
```
mse
```

[165]: 0.34328034268170826

R2 SCORE (COEFFICIENT OF DETERMINATION)

[166]:
```
print('Test R^2     : %.3f'%r2_score(Y_test, prediction))
print('Test R^2     : %.3f'%model.score(X_test, Y_test))
print('Training R^2 : %.3f'%model.score(X_train, Y_train))
```

```
Test R^2     : 0.228
Test R^2     : 0.228
Training R^2 : 0.220
```

MEAN ABSOLUTE ERROR (MAE)

[167]:
```
print('Test MAE  : %.3f'%mean_absolute_error(Y_test, prediction))
print('Train MAE : %.3f'%mean_absolute_error(Y_train, model.predict(X_train)))
```

```
Test MAE  : 0.482
Train MAE : 0.484
```

MEAN SQUARED LOG ERROR

[168]:
```
print("Mean Squared Log Error : {:.3f}".format(mean_squared_log_error(Y_test,␣
↪prediction)))
```

```
Mean Squared Log Error : 0.099
```

MEDIAN ABSOLUTE ERROR

[169]:
```
print('Median Absolute Error : {}'.format(median_absolute_error(Y_test,␣
↪prediction)))
```

```
Median Absolute Error : 0.4370936874467828
```

MAXIMUM RESIDUAL ERROR

[170]:
```
print('Maximum Residual Error : {:.3f}'.format(max_error(Y_test, prediction)))
```

```
Maximum Residual Error : 1.467
```

EXPLAINED VARIANCE ERROR

[171]:
```
print('Explained Variance Error : {:.3f}'.
  ↪format(explained_variance_score(Y_test, prediction)))
```

```
Explained Variance Error : 0.228
```

[172]:
```
ln_mse = mean_squared_error(Y_test, prediction)
ln_r2 = model.score(X_test, Y_test)
ln_mae = mean_absolute_error(Y_test, prediction)
ln_msle = mean_squared_log_error(Y_test, prediction)
ln_mad = median_absolute_error(Y_test, prediction)
ln_mre = max_error(Y_test, prediction)
ln_eve = explained_variance_score(Y_test, prediction)
```

10 GRADIENT BOOSTING REGRESSOR

MODEL SELECTION

[173]:
```
model = GradientBoostingRegressor()
```

MODEL TRAIN - GRADIENT BOOSTING REGRESSOR

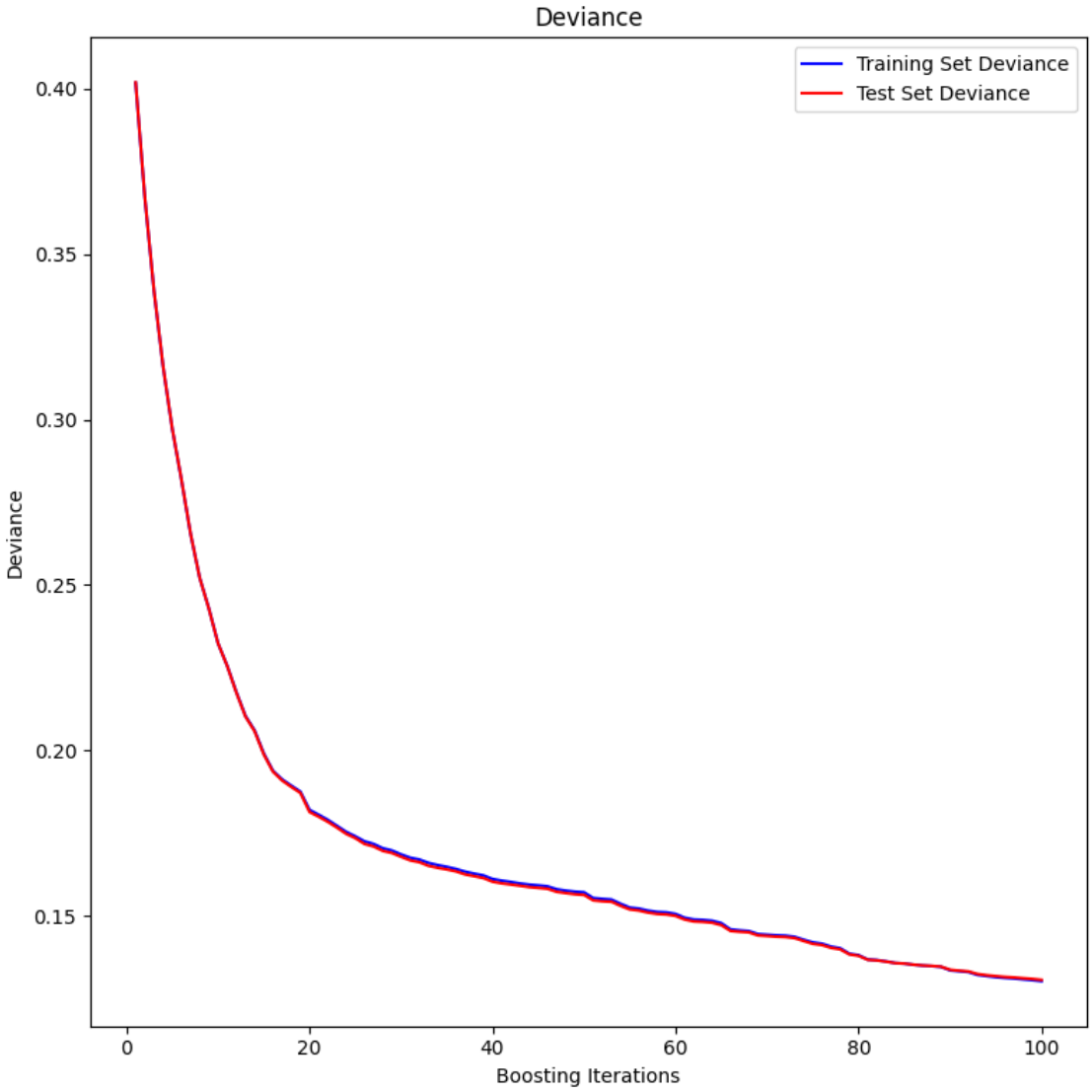
[174]:
```
model.fit(X_train, Y_train)
```

[174]:
```
GradientBoostingRegressor()
```

TRAINING AND TEST DEVIANCE

[175]:
```
model_params = {'n_estimators': 100,
          'max_depth': 3,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
test_score = np.zeros((model_params['n_estimators'],), dtype=np.float64)
for i, y_pred in enumerate(model.staged_predict(X_test)):
    test_score[i] = model.loss_(Y_test, y_pred)

fig = plt.figure(figsize=(8, 8))
plt.subplot(1, 1, 1)
plt.title('Deviance')
plt.plot(np.arange(model_params['n_estimators']) + 1, model.train_score_, 'b-',
         label='Training Set Deviance')
plt.plot(np.arange(model_params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
fig.tight_layout()
plt.show()
```

```
/home/rafi/.local/lib/python3.10/site-packages/sklearn/utils/deprecation.py:101:
FutureWarning: Attribute `loss_` was deprecated in version 1.1 and will be
removed in 1.3.
  warnings.warn(msg, category=FutureWarning)
```

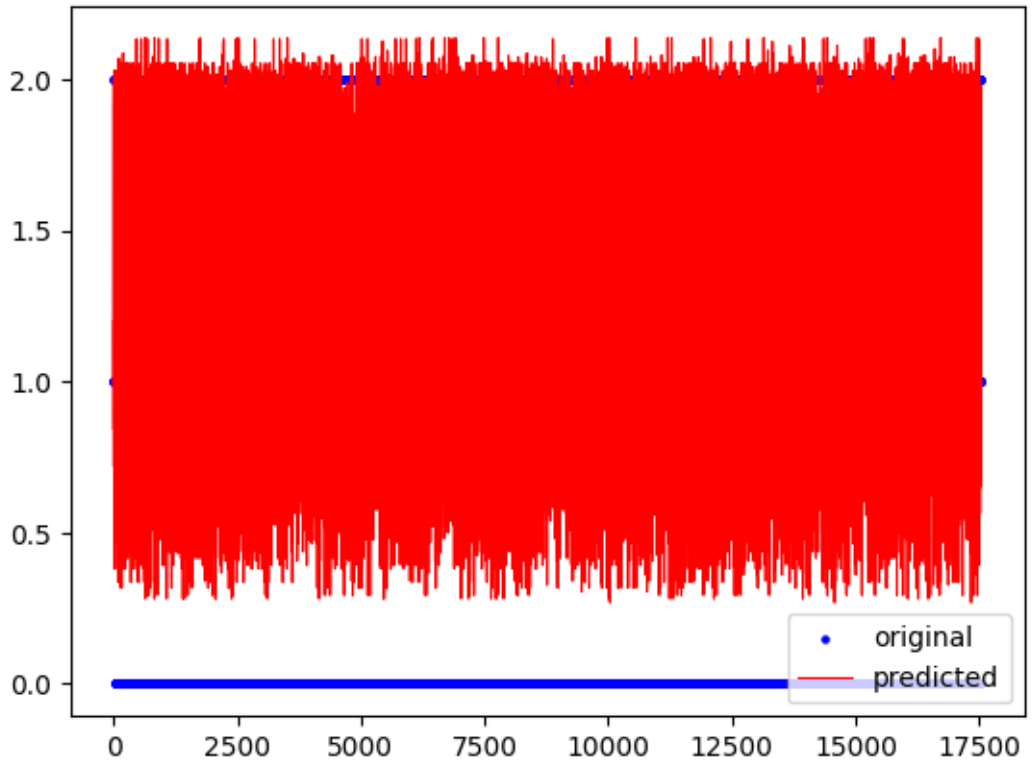


11 TESTING & EVALUATION

[176]:
```
prediction = model.predict(X_test)
```

[177]:
```
x_ax = range(len(Y_test))
plt.scatter(x_ax, Y_test, s=5, color="blue", label="original")
plt.plot(x_ax, prediction, lw=0.8, color="red", label="predicted")
```

```
plt.legend()
plt.show()
```



[178]:
```
contigency = [1, 2, 3]
con = lambda : random.choice(contigency)
```

MEAN SQUARE ERROR

[179]:
```
mse = mean_squared_error(Y_test, prediction)
```

[180]:
```
mse
```

[180]: 0.13067716687140293

R2 SCORE (COEFFICIENT OF DETERMINATION)

[181]:
```
print('Test R^2     : %.3f'%r2_score(Y_test, prediction))
print('Test R^2     : %.3f'%model.score(X_test, Y_test))
print('Training R^2 : %.3f'%model.score(X_train, Y_train))
```

```
Test R^2     : 0.706
Test R^2     : 0.706
Training R^2 : 0.707
```

MEAN ABSOLUTE ERROR (MAE)

[182]:
```
print('Test MAE  : %.3f'%mean_absolute_error(Y_test, prediction))
print('Train MAE : %.3f'%mean_absolute_error(Y_train, model.predict(X_train)))
```

```
Test MAE  : 0.271
Train MAE : 0.271
```

MEAN SQUARED LOG ERROR

[183]:
```
print("Mean Squared Log Error : {:.3f}".format(mean_squared_log_error(Y_test,␣
↪prediction)))
```

```
Mean Squared Log Error : 0.048
```

MEDIAN ABSOLUTE ERROR

[184]:
```
print('Median Absolute Error : {}'.format(median_absolute_error(Y_test,␣
↪prediction)))
```

```
Median Absolute Error : 0.20924330190193396
```

MAXIMUM RESIDUAL ERROR

[185]:
```
print('Maximum Residual Error : {:.3f}'.format(max_error(Y_test, prediction)))
```

```
Maximum Residual Error : 1.090
```

EXPLAINED VARIANCE ERROR

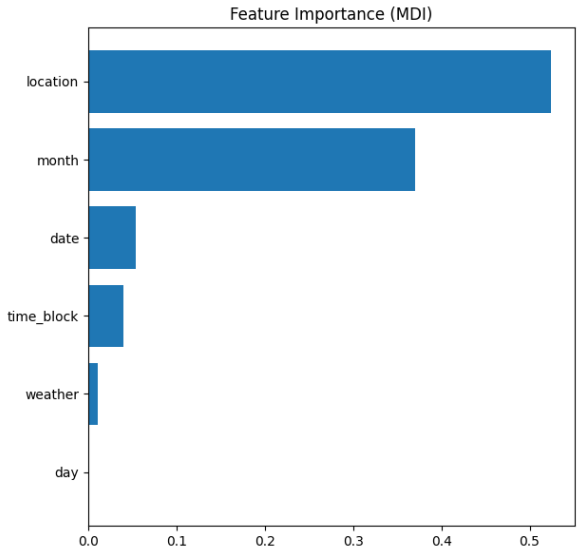
[186]:
```
print('Explained Variance Error : {:.3f}'.
↪format(explained_variance_score(Y_test, prediction)))
```

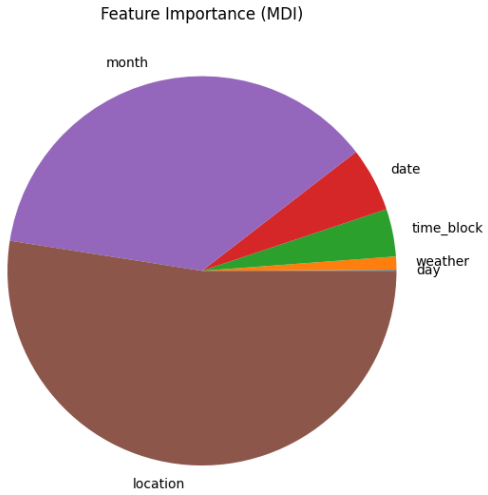
```
Explained Variance Error : 0.706
```

FEATURE IMPORTANCE (MDI) & PERMUTATION IMPORTANCE

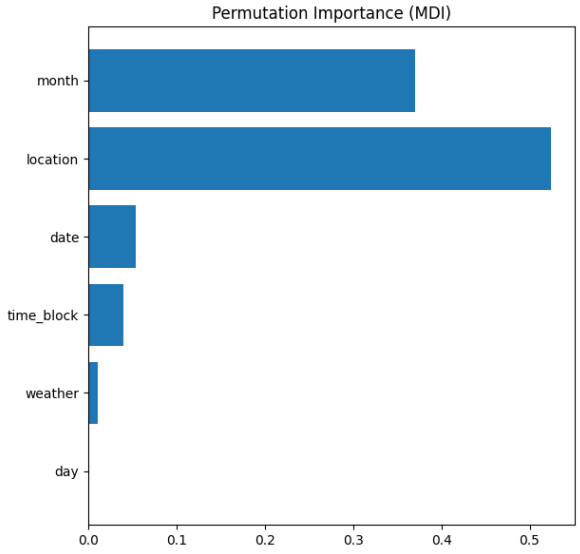
[187]:
```
feature_importance = model.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(colN)[sorted_idx])
plt.title("Feature Importance (MDI)")

feature_importance = model.feature_importances_
sorted_idx = np.argsort(feature_importance)
plt.subplot(1, 2, 2)
plt.pie(feature_importance[sorted_idx], labels = np.array(colN)[sorted_idx])
plt.title("Feature Importance (MDI)")
fig.tight_layout()
```

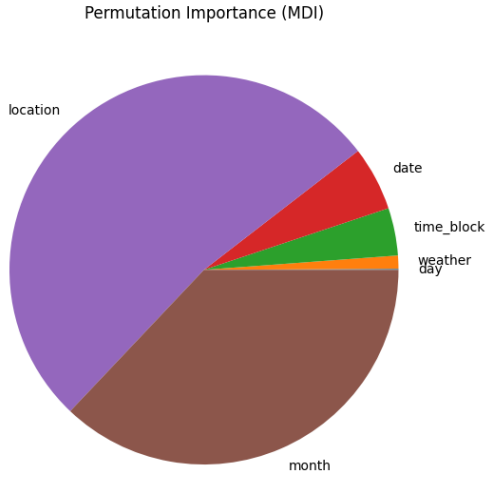
```
plt.show()
```





[188]:
```
result = permutation_importance(
    model, X_test, Y_test, n_repeats=10, random_state=42, n_jobs=2
)
sorted_idx = result.importances_mean.argsort()
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(colN)[sorted_idx])
plt.title("Permutation Importance (MDI)")

result = permutation_importance(
    model, X_test, Y_test, n_repeats=10, random_state=42, n_jobs=2
)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.pie(feature_importance[sorted_idx], labels = np.array(colN)[sorted_idx])
plt.title("Permutation Importance (MDI)")
fig.tight_layout()
plt.show()
```





[189]:
```
gb_mse = mean_squared_error(Y_test, prediction)
gb_r2 = model.score(X_test, Y_test)
gb_mae = mean_absolute_error(Y_test, prediction)
gb_msle = mean_squared_log_error(Y_test, prediction)
gb_mad = median_absolute_error(Y_test, prediction)
gb_mre = max_error(Y_test, prediction)
gb_eve = explained_variance_score(Y_test, prediction)
```

12 HIST GRADIENT BOOSTING REGRESSOR

MODEL SELECTION

[190]:
```
model = HistGradientBoostingRegressor()
```

MODEL TRAIN - GRADIENT BOOSTING REGRESSOR

[191]:
```
model.fit(X_train, Y_train)
```

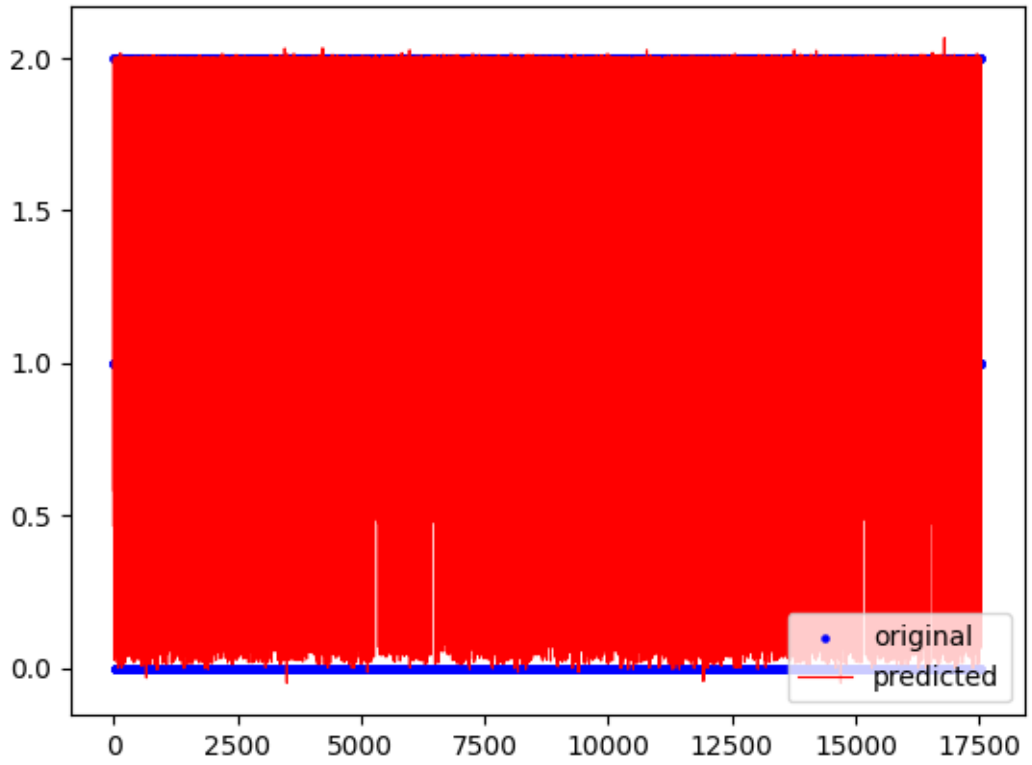
[191]:
```
HistGradientBoostingRegressor()
```

13 TESTING & EVALUATION

[192]:
```
prediction = model.predict(X_test)
```

[193]:
```
x_ax = range(len(Y_test))
plt.scatter(x_ax, Y_test, s=5, color="blue", label="original")
plt.plot(x_ax, prediction, lw=0.8, color="red", label="predicted")
plt.legend()
```

```
plt.show()
```



MEAN SQUARE ERROR

[194]:
```
mse = mean_squared_error(Y_test, prediction)
```

[195]:
```
mse
```

[195]: 0.058196812862329104

R2 SCORE (COEFFICIENT OF DETERMINATION)

[196]:
```
print('Test R^2     : %.3f'%r2_score(Y_test, prediction))
print('Test R^2     : %.3f'%model.score(X_test, Y_test))
print('Training R^2 : %.3f'%model.score(X_train, Y_train))
```

```
Test R^2     : 0.869
Test R^2     : 0.869
Training R^2 : 0.872
```

MEAN ABSOLUTE ERROR (MAE)

```
[197]: print('Test MAE  : %.3f'%mean_absolute_error(Y_test, prediction))
       print('Train MAE : %.3f'%mean_absolute_error(Y_train, model.predict(X_train)))
```

```
Test MAE  : 0.126
Train MAE : 0.125
```

MEDIAN ABSOLUTE ERROR

```
[198]: print('Median Absolute Error : {}'.format(median_absolute_error(Y_test,
        ↪prediction)))
```

```
Median Absolute Error : 0.018046297714618298
```

MAXIMUM RESIDUAL ERROR

```
[199]: print('Maximum Residual Error : {:.3f}'.format(max_error(Y_test, prediction)))
```

```
Maximum Residual Error : 1.400
```

EXPLAINED VARIANCE ERROR

```
[200]: print('Explained Variance Error : {:.3f}'.
        ↪format(explained_variance_score(Y_test, prediction)))
```

```
Explained Variance Error : 0.869
```

```
[201]: hgb_mse = mean_squared_error(Y_test, prediction)
       hgb_r2 = model.score(X_test, Y_test)
       hgb_mae = mean_absolute_error(Y_test, prediction)
       # hgb_msle = mean_squared_log_error(Y_test, prediction)
       hgb_mad = median_absolute_error(Y_test, prediction)
       hgb_mre = max_error(Y_test, prediction)
       hgb_eve = explained_variance_score(Y_test, prediction)
```

14 ANALYSIS AND COMPARISON AMONG MODELS

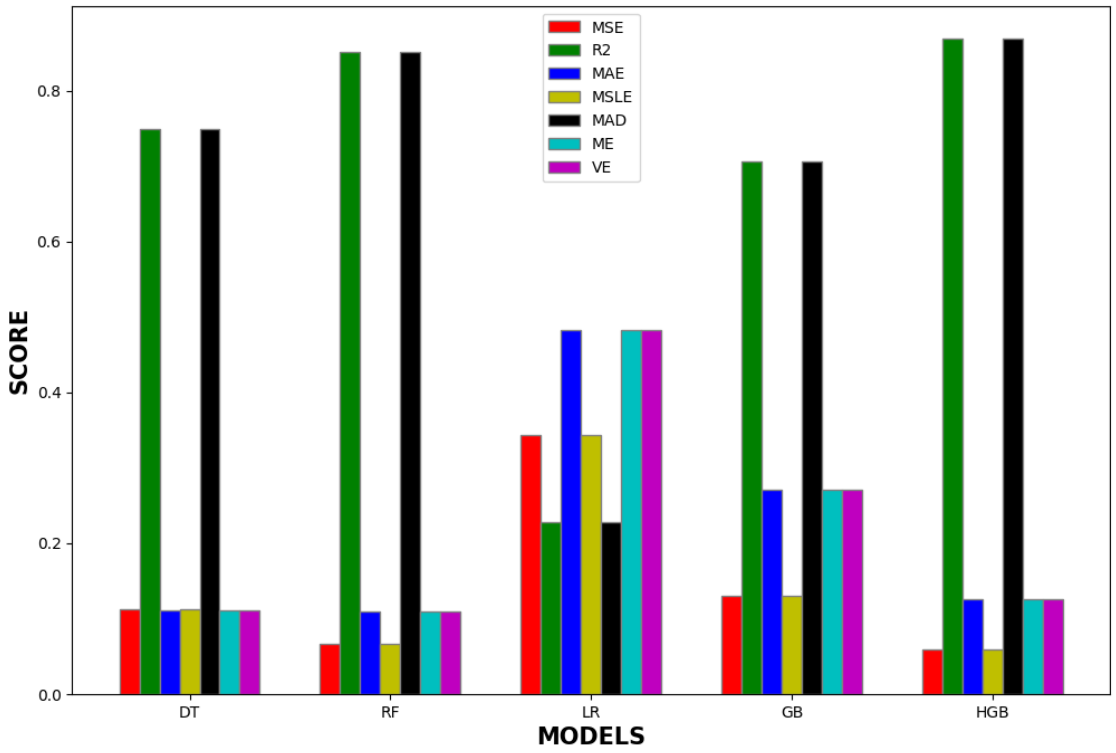
MODEL	MEAN SQUARE ERROR	R2 SCORE	MEAN ARITH-MATIC ERROR	MEAN SQUARED LOG ERROR	MEDIAN ABSO-LUTE ERROR	MAX ERROR	VARIANCE ERROR
DECISION TREE	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2.0	0.7427561190182572
RANDOM FOREST	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.54	0.8492838258776654
LINEAR REGRESSION	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GRADIENT BOOSTING	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

MODEL	MEAN SQUARE ERROR	R2 SCORE	MEAN ARITHMATIC ERROR	MEAN SQUARED LOG ERROR	MEDIAN ABSOLUTE ERROR	MAX ERROR	VARIANCE ERROR
HISTOGRAM GRADIENT BOOSTING	0.0583903038[illegible]	0.5678149534[illegible]	0.1[illegible]	[illegible]23479197	0.020501777[illegible]	1.3[illegible]	0.[illegible]39631584778

[202]:

```python
# set width of bar
barWidth = 0.10
fig = plt.subplots(figsize =(12, 8))

# set height of bar
MSE = [dt_mse, rf_mse, ln_mse, gb_mse, hgb_mse]
R2 = [dt_r2, rf_r2, ln_r2, gb_r2, hgb_r2]
MAE = [dt_mae, rf_mae, ln_mae, gb_mae, hgb_mae]
MSLE = [dt_msle, rf_msle, ln_msle, gb_msle, 0]
MAD = [dt_mad, rf_mad, ln_mad, gb_mad, hgb_mad]
ME = [dt_mre, rf_mre, ln_mre, gb_mre, hgb_mre]
VE = [dt_eve, rf_eve, ln_eve, gb_eve, hgb_eve]


# Set position of bar on X axis
br1 = np.arange(len(MSE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
br6 = [x + barWidth for x in br5]
br7 = [x + barWidth for x in br6]

# Make the plot
plt.bar(br1, MSE, color ='r', width = barWidth,
        edgecolor ='grey', label ='MSE')
plt.bar(br2, R2, color ='g', width = barWidth,
        edgecolor ='grey', label ='R2')
plt.bar(br3, MAE, color ='b', width = barWidth,
        edgecolor ='grey', label ='MAE')
plt.bar(br4, MSE, color ='y', width = barWidth,
        edgecolor ='grey', label ='MSLE')
plt.bar(br5, R2, color ='k', width = barWidth,
        edgecolor ='grey', label ='MAD')
plt.bar(br6, MAE, color ='c', width = barWidth,
        edgecolor ='grey', label ='ME')
plt.bar(br7, MAE, color ='m', width = barWidth,
```

```
        edgecolor ='grey', label ='VE')

# Adding Xticks
plt.xlabel('MODELS', fontweight ='bold', fontsize = 15)
plt.ylabel('SCORE', fontweight ='bold', fontsize = 15)
plt.xticks([r + 3*barWidth for r in range(len(MSE))],
        ['DT', 'RF', 'LR', 'GB', 'HGB'])

plt.legend()
plt.show()
```



15 OPERATIONAL AREA

CUSTOM DATA INPUT

[203]:
```
custom_data = pd.read_csv('/home/rafi/cse404/project/custom_input.csv')
```

[204]:
```
custom_date = input('Enter Date ex: 1,2,3... : ')
custom_month = input('Enter Month ex jan, feb, mar... : ')
custom_time_zone = input('Enter Time in 24H : ')
```

CONVERTING HOUR TO TIME BLOCK

[205]:
```
if '0000' <= custom_time_zone < '0400' :
    custom_time_block = 'a'
elif '0400' <= custom_time_zone < '0800' :
    custom_time_block = 'b'
elif '0800' <= custom_time_zone < '1200' :
    custom_time_block = 'c'
elif '1200' <= custom_time_zone < '1600' :
    custom_time_block = 'd'
elif '1600' <= custom_time_zone < '1800' :
    custom_time_block = 'e'
elif '1800' <= custom_time_zone < '2400' :
    custom_time_block = 'f'
```

CALCULATING DAY FROM DATE AND MONTH

[206]:
```
custom_day_find = custom_date+' '+custom_month+' '+'2023'
day_name= ['mon', 'tue', 'wed', 'thu', 'fri', 'sat','sun']
custom_day = datetime.datetime.strptime(custom_day_find, '%d %b %Y').weekday()
custom_day = day_name[custom_day]
```

FETCHING WEATHER FORECAST FOR THE SPECIFIC INPUT

[207]:
```
if custom_month in ['nov', 'dec', 'jan', 'feb'] :
    custom_weather = 2
elif custom_month in ['jul', 'aug', 'sep', 'oct'] :
    custom_weather = 3
else :
    custom_weather = 1
```

SAVING DATA TO CSV FILE

[208]:
```
custom_data_input = pd.DataFrame([['mirpur_10', custom_month, custom_day,␣
 ↪custom_date, custom_time_block, custom_weather ],
                      ['mirpur_11', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['mirpur_12', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['mirpur_14', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['bijoy_shoroni', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['cantonment', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['jahangir_gate', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                      ['kalshi', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
```

```
                        ['begum_rokeya', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ],
                        ['matikata', custom_month, custom_day, custom_date,␣
 ↪custom_time_block, custom_weather ]],
                       columns=['location', 'month', 'day', 'date',␣
 ↪'time_block', 'weather'])
custom_data_input.to_csv('custom_input.csv', index=False)
```

CUSTOM DATA PREPROCESSING

[209]:
```
custom_encoded_location = le.fit_transform(custom_data['location'])
custom_encoded_month = le.fit_transform(custom_data['month'])
custom_encoded_day = le.fit_transform(custom_data['day'])
custom_encoded_date = le.fit_transform(custom_data['date'])
custom_encoded_time_block = le.fit_transform(custom_data['time_block'])
custom_encoded_weather = le.fit_transform(custom_data['weather'])
```

[210]:
```
C = np.array(list(zip(custom_encoded_location, custom_encoded_month,␣
 ↪custom_encoded_day, custom_encoded_date, custom_encoded_time_block,␣
 ↪custom_encoded_weather)))
```

PREDICT CUSTOM DATA

[211]:
```
prediction = model.predict(C)
```

MAP LOGIC

[212]:
```
def show_map() :
    if route == route_A :
        display(Image(filename='a.png'))
        best_route_cost =␣
 ↪prediction[5]+prediction[6]+prediction[4]+prediction[8]
        print('The best route for you is as followed :')
        print('ECB -> CANTONMENT -> JAHANGIR GATE -> BIJOY SHARANI -> BEGUM␣
 ↪ROKEYA AVE -> AGARGAON')
        print('''The route is selected using
              1. ARTIFICIAL INTELLIGENCE
              2. MACHINE LEARNING
              3. UNIFORMED COST SEARCH
              having the minimum cost of ''', best_route_cost)
    elif route == route_B :
        display(Image(filename='b.png'))
        best_route_cost =␣
 ↪prediction[9]+prediction[3]+prediction[6]+prediction[4]+prediction[8]
        print('The best route for you is as followed :')
        print('ECB -> MATIKATA -> MIRPUR 14 -> JAHANGIR GATE -> BIJOY SHARANI␣
 ↪-> BEGUM ROKEYA AVE -> AGARGAON')
        print('''The route is selected using
```

```
            1. ARTIFICIAL INTELLIGENCE
            2. MACHINE LEARNING
            3. UNIFORMED COST SEARCH
            having the minimum cost of ''', best_route_cost)
  elif route == route_C :
      display(Image(filename='c.png'))
      best_route_cost =␣
↪prediction[9]+prediction[3]+prediction[0]+prediction[8]
      print('The best route for you is as followed :')
      print('ECB -> MATIKATA -> MIRPUR 14 -> MIRPUR 10 -> BEGUM ROKEYA AVE ->␣
↪AGARGAON')
      print('''The route is selected using
            1. ARTIFICIAL INTELLIGENCE
            2. MACHINE LEARNING
            3. UNIFORMED COST SEARCH
            having the minimum cost of ''', best_route_cost)
  elif route == route_D :
      display(Image(filename='d.png'))
      best_route_cost =␣
↪prediction[7]+prediction[1]+prediction[0]+prediction[8]+prediction[3]+prediction[6]+predict
      print('The best route for you is as followed :')
      print('ECB -> KALSHI -> MIRPUR 11 -> MIRPUR 10 -> MIRPUR 14 -> JAHANGIR␣
↪GATE -> BIJOY SHARANI -> BEGUM ROKEYA AVE -> AGARGAON')
      print('''The route is selected using
            1. ARTIFICIAL INTELLIGENCE
            2. MACHINE LEARNING
            3. UNIFORMED COST SEARCH
            having the minimum cost of ''', best_route_cost)
  elif route == route_E :
      display(Image(filename='e.png'))
      best_route_cost =␣
↪prediction[7]+prediction[1]+prediction[0]+prediction[8]
      print('The best route for you is as followed :')
      print('ECB -> KALSHI -> MIRPUR 11 -> MIRPUR 10 -> BEGUM ROKEYA AVE ->␣
↪AGARGAON')
      print('''The route is selected using
            1. ARTIFICIAL INTELLIGENCE
            2. MACHINE LEARNING
            3. UNIFORMED COST SEARCH
            having the minimum cost of ''', best_route_cost)
  elif route == route_F :
      display(Image(filename='f.png'))
      best_route_cost =␣
↪prediction[7]+prediction[2]+prediction[0]+prediction[8]
      print('The best route for you is as followed :')
```

```
        print('ECB -> MIRPUR 12 -> MIRPUR 11 -> MIRPUR 10 -> BEGUM ROKEYA AVE␣
↪-> AGARGAON')
        print('''The route is selected using
              1. ARTIFICIAL INTELLIGENCE
              2. MACHINE LEARNING
              3. UNIFORMED COST SEARCH
              having the minimum cost of ''', best_route_cost)
    else :
        print('No suitable route available! Try Again')
```

NODE BASED PREDICTION

[213]:
```
print('Matikata         :', prediction[9])
print('Cantonment       :', prediction[5])
print('Mirpur 10        :', prediction[0])
print('Mirpur 11        :', prediction[1])
print('Mirpur 12        :', prediction[2])
print('Mirpur 14        :', prediction[3])
print('Begum Rokeya     :', prediction[8])
print('Bijoy Shoroni    :', prediction[4])
print('Jahangit Gate    :', prediction[6])
print('Kalshi           :', prediction[7])
```

```
Matikata         : 1.007032229954942
Cantonment       : 1.0071101921842218
Mirpur 10        : 2.0062645475118157
Mirpur 11        : 2.0069764320879133
Mirpur 12        : 1.0056409158800987
Mirpur 14        : 2.0078062789844946
Begum Rokeya     : 1.990762195927357
Bijoy Shoroni    : 1.988619938534107
Jahangit Gate    : 1.9894763082487088
Kalshi           : 1.007032229954942
```

16 SHORTEST PATH SEARCH ALGORITHM

DEIFINING UNIFORM COST SEARCH

[214]:
```
class Node(object):
    """This class represents a node in a graph."""

    def __init__(self, label: str=None):
        """
        Initialize a new node.

        Args:
            label: the string identifier for the node
        """
```

```
        self.label = label
        self.children = []

    def __lt__(self,other):
        """
        Perform the less than operation (self < other).

        Args:
            other: the other Node to compare to
        """
        return (self.label < other.label)

    def __gt__(self,other):
        """
        Perform the greater than operation (self > other).

        Args:
            other: the other Node to compare to
        """
        return (self.label > other.label)

    def __repr__(self):
        """Return a string form of this node."""
        return '{} -> {}'.format(self.label, self.children)

    def add_child(self, node, cost=1):
        """
        Add a child node to this node.

        Args:
            node: the node to add to the children
            cost: the cost of the edge (default 1)
        """
        edge = Edge(self, node, cost)
        self.children.append(edge)


class Edge(object):
    """This class represents an edge in a graph."""

    def __init__(self, source: Node, destination: Node, cost: int=1):
        """
        Initialize a new edge.

        Args:
            source: the source of the edge
            destination: the destination of the edge
```

```
            cost: the cost of the edge (default 1)
        """
        self.source = source
        self.destination = destination
        self.cost = cost

    def __repr__(self):
        """Return a string form of this edge."""
        return '{}: {}'.format(self.cost, self.destination.label)

def ucs(root, goal):
    """
    Return the uniform cost search path from root to gaol.

    Args:
        root: the starting node for the search
        goal: the goal node for the search

    Returns: a list with the path from root to goal

    Raises: ValueError if goal isn't in the graph
    """
    # create a priority queue of paths
    queue = PriorityQueue()
    queue.put((0, [root]))
    # iterate over the items in the queue
    while not queue.empty():
        # get the highest priority item
        pair = queue.get()
        current = pair[1][-1]
        # if it's the goal, return
        if current.label == goal:
            return pair[1]
        # add all the edges to the priority queue
        for edge in current.children:
            # create a new path with the node from the edge
            new_path = list(pair[1])
            new_path.append(edge.destination)
            # append the new path to the queue with the edges priority
            queue.put((pair[0] + edge.cost, new_path))
```

DEFINING NODES

[215]:
```
ECB = Node('ECB')
CANTONMENT = Node('CANTONMENT')
MATIKATA = Node('MATIKATA')
KALSHI = Node('KALSHI')
```

```
MIRPUR_12 = Node('MIRPUR_12')
JAHANGIR_GATE = Node('JAHANGIR_GATE')
MIRPUR_14 = Node('MIRPUR_14')
MIRPUR_10 = Node('MIRPUR_10')
MIRPUR_11 = Node('MIRPUR_11')
BIJOY_SHARANI = Node('BIJOY_SHARANI')
BEGUM_ROKEYA = Node('BEGUM_ROKEYA')
AGARGAON = Node('AGARGAON')
```

[216]:
```
route_A = ['ECB', 'CANTONMENT', 'JAHANGIR_GATE', 'BIJOY_SHARANI',␣
 ↪'BEGUM_ROKEYA', 'AGARGAON']
route_B = ['ECB', 'MATIKATA', 'MIRPUR_14', 'JAHANGIR_GATE', 'BIJOY_SHARANI',␣
 ↪'BEGUM_ROKEYA', 'AGARGAON']
route_C = ['ECB', 'MATIKATA', 'MIRPUR_14', 'MIRPUR_10', 'BEGUM_ROKEYA',␣
 ↪'AGARGAON']
route_D = ['ECB', 'KALSHI', 'MIRPUR_11', 'MIRPUR_10', 'MIRPUR_14',␣
 ↪'JAHANGIR_GATE', 'BIJOY_SHARANI', 'AGARGAON']
route_E = ['ECB', 'KALSHI', 'MIRPUR_11', 'MIRPUR_10', 'BEGUM_ROKEYA',␣
 ↪'AGARGAON']
route_F = ['ECB', 'MIRPUR_12', 'MIRPUR_11', 'MIRPUR_10', 'BEGUM_ROKEYA',␣
 ↪'AGARGAON']
```

ASSIGNING BRANCHES WITH PREDICTED VALUES

[217]:
```
ECB.add_child(CANTONMENT, prediction[5]+0)
ECB.add_child(MATIKATA, prediction[9]+0)
ECB.add_child(KALSHI, prediction[7]+0)
ECB.add_child(MIRPUR_12, prediction[2]+0)

CANTONMENT.add_child(JAHANGIR_GATE, con()*prediction[6]+con()*prediction[5])

MATIKATA.add_child(MIRPUR_14, con()*prediction[3]+con()*prediction[9])

KALSHI.add_child(MIRPUR_11, con()*prediction[1]+con()*prediction[7])

MIRPUR_12.add_child(MIRPUR_11, con()*prediction[1]+con()*prediction[2])

JAHANGIR_GATE.add_child(BIJOY_SHARANI, con()*prediction[4]+con()*prediction[6])
JAHANGIR_GATE.add_child(MIRPUR_14, con()*prediction[3]+con()*prediction[6])

MIRPUR_11.add_child(MIRPUR_10, con()*prediction[0]+con()*prediction[1])

MIRPUR_10.add_child(MIRPUR_14, con()*prediction[3]+con()*prediction[0])

MIRPUR_14.add_child(MIRPUR_10, con()*prediction[0]+con()*prediction[3])
MIRPUR_14.add_child(JAHANGIR_GATE, con()*prediction[6]+con()*prediction[3])
```

```
BIJOY_SHARANI.add_child(BEGUM_ROKEYA, con()*prediction[8]+con()*prediction[4])

MIRPUR_10.add_child(BEGUM_ROKEYA, con()*prediction[8]+con()*prediction[0])

BEGUM_ROKEYA.add_child(AGARGAON, 0+con()*prediction[8])
```

OVERVIEW OF PREDICTED VALUES OF ALL NODES

[218]:
```
_ = [print('*', node) for node in [ECB, CANTONMENT, MATIKATA, KALSHI,␣
↪MIRPUR_12, JAHANGIR_GATE, MIRPUR_11, MIRPUR_14, MIRPUR_10, BIJOY_SHARANI,␣
↪BEGUM_ROKEYA, AGARGAON]]
```

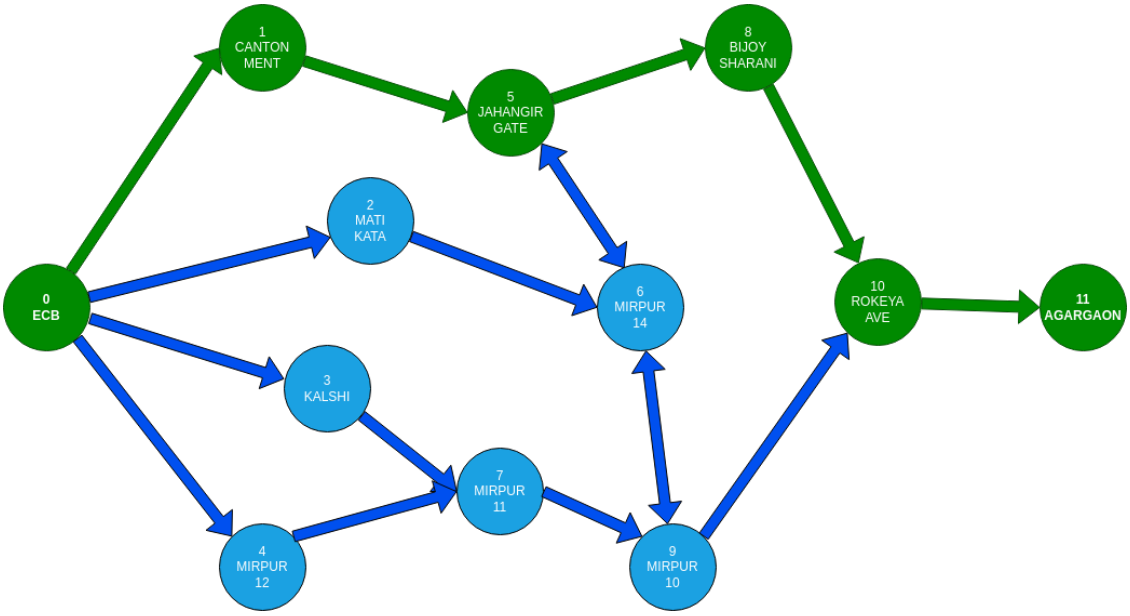
```
* ECB -> [1.0071101921842218: CANTONMENT, 1.007032229954942: MATIKATA,
1.007032229954942: KALSHI, 1.0056409158800987: MIRPUR_12]
* CANTONMENT -> [5.993173000865861: JAHANGIR_GATE]
* MATIKATA -> [9.044515526818309: MIRPUR_14]
* KALSHI -> [5.0209850941307685: MIRPUR_11]
* MIRPUR_12 -> [6.025234695936025: MIRPUR_11]
* JAHANGIR_GATE -> [3.978096246782816: BIJOY_SHARANI, 7.994565174466407:
MIRPUR_14]
* MIRPUR_11 -> [8.025770074623361: MIRPUR_10]
* MIRPUR_14 -> [10.035947931977116: MIRPUR_10, 6.005088866217698: JAHANGIR_GATE]
* MIRPUR_10 -> [6.020335374008126: MIRPUR_14, 8.009555838462804: BEGUM_ROKEYA]
* BIJOY_SHARANI -> [7.960906526316178: BEGUM_ROKEYA]
* BEGUM_ROKEYA -> [1.990762195927357: AGARGAON]
* AGARGAON -> []
```

[219]:
```
route = []
for x in ucs(ECB, 'AGARGAON') :
    route_temp = str(x).split(' ')
    route.append(route_temp[0])
```

[220]:
```
ucs(ECB, 'AGARGAON')
```

[220]:
```
[ECB -> [1.0071101921842218: CANTONMENT, 1.007032229954942: MATIKATA,
1.007032229954942: KALSHI, 1.0056409158800987: MIRPUR_12],
 CANTONMENT -> [5.993173000865861: JAHANGIR_GATE],
 JAHANGIR_GATE -> [3.978096246782816: BIJOY_SHARANI, 7.994565174466407:
MIRPUR_14],
 BIJOY_SHARANI -> [7.960906526316178: BEGUM_ROKEYA],
 BEGUM_ROKEYA -> [1.990762195927357: AGARGAON],
 AGARGAON -> []]
```

[221]:
```
show_map()
```




```
The best route for you is as followed :
ECB -> CANTONMENT -> JAHANGIR GATE -> BIJOY SHARANI -> BEGUM ROKEYA AVE ->
AGARGAON
The route is selected using
              1. ARTIFICIAL INTELLIGENCE
              2. MACHINE LEARNING
              3. UNIFORMED COST SEARCH
              having the minimum cost of  6.975968634894395
```