Regression report

First of all :

a) Preprocessing technique:

1- As I saw the first view of table I have 3 categorical features which are

(season ,holiday ,function day) as I use regression techniques they are

Dealing with numerical data so I had to converting them:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
encoding = LabelEncoder()
bike['Seasons'] = encoding.fit_transform(bike['Seasons'])
bike['Holiday'] = encoding.fit_transform(bike['Holiday'])
bike['Functioning Day'] = encoding.fit_transform(bike['Functioning Day'])
```

in (season) feature I found 3 parameters only(winter ,summer, spring) when they are converted the scene will be (0 ,1 ,2) but 1 year has 4 seasons so, I decided to add 1 empty record all are nan values except season have (autumn) feature, I used label encoder the values after that will be(0, 1, 2, 3).

2- Second scene I saw some features having problems here , I found That (Visibility (10m)) column has high numerical ranges comparable to the other features and column(Temperature(Â°C)) and (Dew point temperature(Â°C)) having negative values so I had to deal with this by scaling them:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MaxAbsScaler
scale = MinMaxScaler(copy=True, feature_range=(0, 1))
x1 = np.array(bike['Temperature(°C)'], dtype=float64)
bike['Temperature(°C)'] = scale.fit_transform(x1.reshape(-1, 1))
x2 = np.array(bike['Dew point temperature(°C)'], dtype=float64)
bike['Dew point temperature(°C)'] = scale.fit_transform(x2.reshape(-1, 1))
x3 = np.array(bike['Solar Radiation (MJ/m2)'], dtype=float64)
bike['Solar Radiation (MJ/m2)'] = scale.fit_transform(x3.reshape(-1, 1))
x4 = np.array(bike['Humidity(%)'], dtype=int64)
bike['Humidity(%)'] = scale.fit_transform(x4.reshape(-1, 1))
x5 = np.array(bike['Wind speed (m/s)'], dtype=float64)
bike['Wind speed (m/s)'] = scale.fit_transform(x5.reshape(-1, 1))
```

```
x6 = np.array(bike['Visibility (10m)'], dtype=int64)
bike['Visibility (10m)'] = scale.fit_transform(x6.reshape(-1, 1))
x7 = np.array(bike['Hour'], dtype=int64)
bike['Hour'] = scale.fit_transform(x7.reshape(-1, 1))
```

I imported all these to techniques from Sklearn And I tried to use everyone of them, it seems that they are very close To each other in converting values into range (0, 1) , to seems more visible to normal data, when I also used these data I found that they are almost giving the same output in predictions , so I used minmax scaling for them Because it seems the easiest to me.

3- The last one the train file seems clean to me but, I add new record with nan values so I had to remove them , or actually replacing them with specific values:

```
from sklearn.impute import SimpleImputer
imputes = SimpleImputer(missing_values=np.nan,
strategy='mean').fit_transform(bike)
```

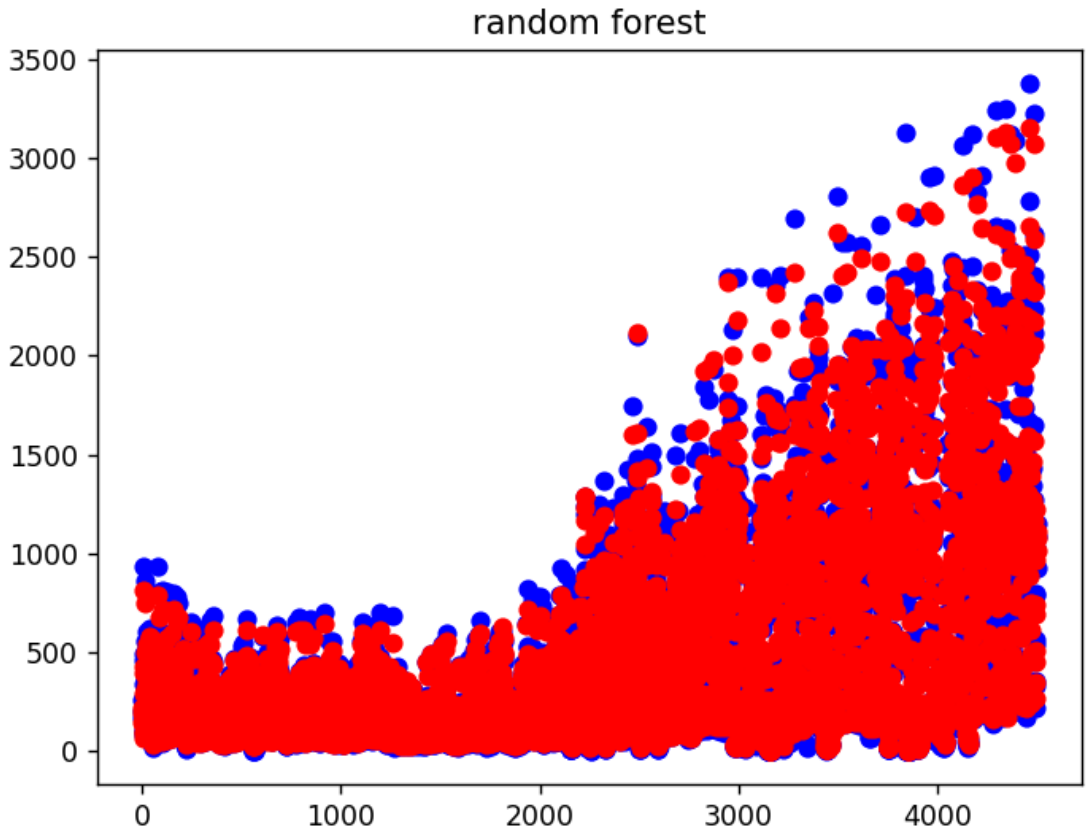
simple imputer is a technique uses some statistical function to apply them on features they are(mean , mode, median ,constant) I picked mean function from them to apply that takes all the values Of columns and getting their average and takes every value in that column and put it in the empty cell.

b) Analysis

1- when I applied the analysis found that the most features affects the model predictions are (Hour, Temperature(Â°C), Solar Radiation (MJ/m2))

```
from sklearn.feature_selection import SelectFromModel
select2 = SelectFromModel(RandomForestRegressor())
Selected = select2.fit_transform(x, y)
print(Selected.shape)
print(select2.get_support())
```

here I got some help from sklearn library by featuring by model here I pick up the best features I use for prediction as you see I used ensemble model that I used it (explained later)
but in my final result I get numbers and I wanted to see and visualize so:



That is my final scatter plot on model but without feature selection I used them all ,because as you see they are three feature selected in the previous frame when I applied the model it seems that the difference between predicted and actual values are very large And they are 12 feature the on the other features they affects slightly low but with numerous they affect a lot.

c) regression techniques:

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression().fit(x_train, y_train)
y_predict = LR.predict(x_test)
print('predict values:', y_predict)
print('actual values:', y_test)
mse = mean_squared_error(y_test, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

1- First linear regression ,all models have mostly the same parameters except for ensemble models (talking later)

Parameters of linear regression are:

a- Fit intercept it gets me a values of Boolean(T or F) According to the interaction of x -axis which is the actual value of model, some people need this for visualizing their Linear model on scattering.
b- Normalize for normalization data scaling them for processing.(Boolean)
c- Copy-x for not deformation data in the file (Boolean)
d- N_jobs of values start from 1 to upper to speed up the model processing (used for hardware issue)

There are some parameter I did not include them actually

But the main parameter I use are:

Fit(x ,y): by splitting csv file in x and y for fitting them in the model

Predict(x): here I used train test split (talk later) for prediction

Score(x, y) they are not metrices ,they define how far of this algorithm I use in effectiveness for prediction

```
ridge = Ridge(alpha=0.1)
ridge.fit(x_train, y_train)
y_predict = ridge.predict(x_test)
print('predict values:', y_predict)
print('actual values:', y_test)
mse = mean_squared_error(y_test, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

2- Ridge regression: almost like normal regression but it seems from it's name means regularize data with an (alpha) parameter Which starts from 0 to upper when alpha increases the rate of regularization increases the more data are being readable.

d) Difference between models:
First I have to show you the results of models:

```
model = LinearRegression().fit(x_train, y_train)
y_predict = model.predict(x_train)
print('predict values:', y_predict)
mse = mean_squared_error(y_train, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
print('accurecy:',accuracy_score(y_train,y_predict))
```

- linear = mean square error: 342.1595997165748

```
lasso = Lasso(alpha=1.0)
lasso.fit(x_train, y_train)
y_predict = lasso.predict(x_train)
print('predict values:', y_predict)
mse = mean_squared_error(y_train, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

- lasso = mean square error: 428.5359741553312

```
ridge = Ridge(alpha=0.1)
ridge.fit(x_train, y_train)
y_predict = ridge.predict(x_train)
print('predict values:', y_predict)
mse = mean_squared_error(y_train, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

- ridge = mean square error: 427.85794935405255

```
elastic_net = ElasticNet(alpha=0.05)
elastic_net.fit(x_train, y_train)
y_predict = elastic_net.predict(x_test)
print('predict values:', y_predict)
print('actual values:', y_test)
mse = mean_squared_error(y_test, y_predict, multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

- elastic net = mean square error: 439.2520698678281

_____Ensemble_____

```
model = RandomForestRegressor()
model.fit(x_train, y_train)
y_predict_train = model.predict(x_train)
print('predict values :', y_predict_train)
mse = mean_squared_error(y_train, y_predict_train,
multioutput='uniform_average')
print('mean square error :', np.sqrt(mse))
```

- Random forest regressor = mean square error : 86.56422476930703

```
model = GradientBoostingRegressor()
model.fit(x_train, y_train)
y_predict_train = model.predict(x_train)
print('predict values:', y_predict_train)
mse = mean_squared_error(y_train, y_predict_train,
multioutput='uniform_average')
print('mean square error:', np.sqrt(mse))
```

- Gradient boosting regressor = mean square error: 223.37402300728553

Every value of mean square errors can be minimized by changing some parameters of every model ,for me I almost used the default one in some models and others I had to use (n_estimation, random_state,learning_rate in ensemble models, n_jobs to speed up algorithms ,for lasso,ridge and elastic net play the rules of changing the alpha parameters ,elastic net model in that show (lasso > elastic net > ridge) .

- Scoring model I had to use the accuracy_score from sklearn

```
- from sklearn.metrics import accuracy_score
- print('accurecy:',accuracy_score(y_train,y_predict))
```

but I had output error:
ValueError: Classification metrics can't handle a mix of multiclass and continuous targets
Led me to use cross validation score with a data frame of x & y and number of k-folds© but I also had the same error there are another of score like(r2_score) also from metrics

too, but the documentation mentioned the accuracy score.

e) Feature selection:

```
f) from sklearn.feature_selection import SelectFromModel
g) select2 = SelectFromModel(RandomForestRegressor())
   Selected = select2.fit_transform(x, y)
   print(Selected.shape)
   print(select2.get_support())
```

I used feature selection that I talked about it in the previous page
And applied it on ensemble model and the output showed me this:
[True True False False False False True False False False False False]

The true values means that it is a good feature which are
(Hour , Temperature(Â°C) , Solar Radiation (MJ/m2)) but when I used these features and dropped the others I had very high mean square error ~ 648.1154421
That means to me the selected feature are few and as we learned
The more features I got the less errors I have including every type of errors, so I kept the others in my model and I worked on scaling them.

f) training ,testing and validation:

```
            from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=33)
```

splitting data to work on it has some kind of shapes

(80 : 20), (90 : 10) {this is the parameter that had the validation set}

(60 : 40) ,(50 : 50) {this is the minimum point that I had to reach

Testing set percentage can not be more that training one.

Let get back to the model in previous I had to use (80 : 20)

But I changed them to the default parameters which are (75 : 25)

I can also use random state with what ever number I want from (1 ,100)

h) here I mostly used a low learning rate for most fitting for data in the model that give me high accuracy with minimum mean square error

```
model = GradientBoostingRegressor(learning_rate=0.05)
```

nan	nan	nan	nan	nan	nan	nan	nan	autmun	nan	nan

Her I inserted one empty record for evaluating label encoder because , clearly the year have 4 seasons and I used simple_imputers for scaling the nan value

It seems to be perfect choice for prediction on Kaggle on testing data.

i) My final conclusion is that I did suffered a lot for the first time In scaling data:

```
scale = MinMaxScaler(copy=True, feature_range=(0, 1))
x1 = np.array(bike['Temperature(°C)'], dtype=float64)
bike['Temperature(°C)'] = scale.fit_transform(x1.reshape(-1, 1))
x2 = np.array(bike['Dew point temperature(°C)'], dtype=float64)
bike['Dew point temperature(°C)'] = scale.fit_transform(x2.reshape(-1, 1))
x3 = np.array(bike['Solar Radiation (MJ/m2)'], dtype=float64)
bike['Solar Radiation (MJ/m2)'] = scale.fit_transform(x3.reshape(-1, 1))
x4 = np.array(bike['Humidity(%)'], dtype=int64)
bike['Humidity(%)'] = scale.fit_transform(x4.reshape(-1, 1))
x5 = np.array(bike['Wind speed (m/s)'], dtype=float64)
bike['Wind speed (m/s)'] = scale.fit_transform(x5.reshape(-1, 1))
x6 = np.array(bike['Visibility (10m)'], dtype=int64)
bike['Visibility (10m)'] = scale.fit_transform(x6.reshape(-1, 1))
x7 = np.array(bike['Hour'], dtype=int64)
bike['Hour'] = scale.fit_transform(x7.reshape(-1, 1))
```

in this view I tried to scale some feature the model selected It every time I removed columns by scaling 3 , 4 ,5 ,6 , etc… feature everyone on its track and I tried to do them altogether in the model prediction

I found nothing difference in prediction ,even mean square error changed very slightly assume from 79.142 to 80.0123 and so on

So , I do see that there is nothing improved ,

```
def plotGraph(y_train, y_pred_train, rand):
    if max(y_train) >= max(y_pred_train):
        my_range = int(max(y_train))
    else:
        my_range = int(max(y_pred_train))
    plt.scatter(range(len(y_train)), y_train, color='blue')
    plt.scatter(range(len(y_pred_train)), y_pred_train, color='red')
    plt.title(rand)
    plt.show()
    return


plotGraph(y_train, y_predict_train, 'random forest')
```

by plotting a graph that show and visualize data for me seems more comfort to see the difference between predicted and actual data so, I decide for every model I use I plot it, that seems to be improvement to me to see what is going on.

```
# preprocessing standardization
# scale = StandardScaler(copy=True, with_mean=True, with_std=True)
# xs = scale.fit_transform(xs.reshape(-1, 1))

# preprocessing MinMaxscaler
scale2 = MinMaxScaler(copy=True, feature_range=(0, 1))
xs = scale2.fit_transform(xs.reshape(-1, 1))

# preprocessing Normalizer
# scale3 = Normalizer(norm='l2', copy=True)
# xs = scale3.fit_transform(xs.reshape(-1, 1))

# preprocessing MaxAbsscaler
# scale3 = MaxAbsScaler(copy=True)
# xs = scale3.fit_transform(xs.reshape(-1, 1))
# create train and test data
```

This part of data scaling on my data seems to be they are close to each other in numeric and categoric data but it also must be and improvement.