HW4 part 2 (14 Points): Required Submissions:

1. Submit colab/jupyter notebooks.
2. Pdf version of the notebooks (HWs will not be graded if pdf version is not provided).
3. **The notebooks and pdf files should have the output.**
4. **Name files as follows : FirstNameLastName_HW5_part2**

Question1 (6 Points) : Classification on the 'credit-g' dataset using KNN Classification

Import/Install the packages

```
if 'google.colab' in str(get_ipython()):
    print('Running on Colab')
else:
    print('Not Running on Colab')
```

```
Running on Colab
```

```
if 'google.colab' in str(get_ipython()):
  !pip install --upgrade feature_engine scikit-learn -q
```

Show hidden output

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

```
Mounted at /content/drive
```

```
import feature_engine
```

```
print(feature_engine.__version__)
```

```
1.6.2
```

```
import sklearn
```

```
print(sklearn.__version__)
```

```
1.3.1
```

```
"""Importing the required packages"""

# For DataFrames and manipulations
import pandas as pd
import numpy as np

# For data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
%matplotlib inline

# save and load models
import joblib

# Pathlib to navigate file system
from pathlib import Path
import sys

# For splitting the dataset
from sklearn.model_selection import train_test_split
from feature_engine.selection import DropFeatures
```

```
# For categorical variables
from feature_engine.encoding import OneHotEncoder
from feature_engine.encoding import RareLabelEncoder

# For scaling the data
from sklearn.preprocessing import StandardScaler

# creating pipelines
from sklearn.pipeline import Pipeline

# Hyper parameter tuning
from sklearn.model_selection import GridSearchCV

# Using KNN classification for our data
from sklearn.neighbors import KNeighborsClassifier

# draws a confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay

# We will use this to download the Dataset
from sklearn.datasets import fetch_openml

# feature engine log transformation
from feature_engine.transformation import LogTransformer

# feature engine wrapper
from feature_engine.wrappers import SklearnTransformerWrapper
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import ShuffleSplit
from sklearn.preprocessing import MinMaxScaler
```

Specify Project Folder Location

```
if 'google.colab' in str(get_ipython()):
    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=Tru
```

```
base_folder = Path('/content/drive/MyDrive/Applied_ML/Class_4/Assignment')
```

```
data_folder = base_folder/'Datasets'
save_model_folder = base_folder/'Model'
custom_function_folder = Path('/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Custom_function')
save_model_folder.mkdir(exist_ok=True, parents=True)
```

Import Custom Functions from Python file

```
%load_ext autoreload
%autoreload 2
```

```
sys.path.append(str(custom_function_folder))
```

```
sys.path
```

```
['/content',
 '/env/python',
 '/usr/lib/python310.zip',
 '/usr/lib/python3.10',
 '/usr/lib/python3.10/lib-dynload',
 '',
 '/usr/local/lib/python3.10/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.10/dist-packages/IPython/extensions',
```

```
 '/root/.ipython',
 '/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Custom_function']
```

```
from  plot_learning_curve import plot_learning_curve
```

```
from eda_plots import diagnostic_plots, plot_target_by_category
```

```
from sklearn.datasets import fetch_openml
```

```
import zipfile

with zipfile.ZipFile(data_folder/'seoul+bike+sharing+demand.zip', 'r') as zip_ref:
    zip_ref.extractall(data_folder)
```

Question2 (14 Points) : KNN Regression on Bike Sharing Dataset

- Download the data from following link: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand'

```
data = pd.read_csv('/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Datasets/SeoulBikeData.csv', encoding='latin-1')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object 
 12  Holiday                    8760 non-null   object 
 13  Functioning Day            8760 non-null   object 
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

Task1 (6 Points): Do the EDA and identify the preprocessing steps.

```
data.head(10)
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.00	0.0	0.0
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.00	0.0	0.0

```
data.columns
```

```
Index(['Date', 'Rented Bike Count', 'Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons',
       'Holiday', 'Functioning Day'],
      dtype='object')
```

```
data.info('columns')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object 
 12  Holiday                    8760 non-null   object 
 13  Functioning Day            8760 non-null   object 
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

Dropping Date column as it is not needed for prediction

```
data.drop('Date',axis=1, inplace=True)
```

```
data.isna().sum()
```

```
Rented Bike Count            0
Hour                         0
Temperature(°C)              0
Humidity(%)                  0
Wind speed (m/s)             0
Visibility (10m)             0
Dew point temperature(°C)    0
Solar Radiation (MJ/m2)      0
Rainfall(mm)                 0
Snowfall (cm)                0
Seasons                      0
Holiday                      0
Functioning Day              0
dtype: int64
```

```
data.nunique()
```

```
Rented Bike Count            2166
Hour                           24
Temperature(°C)               546
Humidity(%)                    90
Wind speed (m/s)               65
Visibility (10m)             1789
Dew point temperature(°C)     556
Solar Radiation (MJ/m2)       345
Rainfall(mm)                   61
Snowfall (cm)                  51
Seasons                         4
Holiday                         2
```

```
Functioning Day              2
dtype: int64
```

```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Rented Bike Count	8760.0	704.602055	644.997468	0.0	191.00	504.50	1065.25	3556.00
Hour	8760.0	11.500000	6.922582	0.0	5.75	11.50	17.25	23.00
Temperature(°C)	8760.0	12.882922	11.944825	-17.8	3.50	13.70	22.50	39.40
Humidity(%)	8760.0	58.226256	20.362413	0.0	42.00	57.00	74.00	98.00
Wind speed (m/s)	8760.0	1.724909	1.036300	0.0	0.90	1.50	2.30	7.40
Visibility (10m)	8760.0	1436.825799	608.298712	27.0	940.00	1698.00	2000.00	2000.00
Dew point temperature(°C)	8760.0	4.073813	13.060369	-30.6	-4.70	5.10	14.80	27.20
Solar Radiation (MJ/m2)	8760.0	0.569111	0.868746	0.0	0.00	0.01	0.93	3.52
Rainfall(mm)	8760.0	0.148687	1.128193	0.0	0.00	0.00	0.00	35.00
Snowfall (cm)	8760.0	0.075068	0.436746	0.0	0.00	0.00	0.00	8.80

```
data.duplicated().any()
```

```
False
```

We see that Solar radiation, Rainfall and Snowfall have most of data points close to each other with some outliers, so converting those 3 columns into 3 categories

There are no missing data points in the dataframe

```
categorical = [var for var in data.columns if data[var].dtype == 'O'and var not in ['Rented Bike Count']]
discrete = [var for var in data.columns if data[var].dtype != 'O'and len(data[var].unique()) < 20 and var not in ['Rented Bike Cou
continuous = [var for var in data.columns if data[var].dtype != 'O' and var not in discrete and var not in ['Rented Bike Count']]
```

```
categorical
```

```
['Seasons', 'Holiday', 'Functioning Day']
```

```
discrete
```

```
[]
```

```
continuous
```

```
['Hour',
 'Temperature(°C)',
 'Humidity(%)',
 'Wind speed (m/s)',
 'Visibility (10m)',
 'Dew point temperature(°C)',
 'Solar Radiation (MJ/m2)',
 'Rainfall(mm)',
 'Snowfall (cm)']
```

```
categorical_distribution = data[categorical].nunique()
print(categorical_distribution)
```

```
Seasons            4
Holiday            2
Functioning Day    2
dtype: int64
```

```
continuous_distribution = data[continuous].nunique()
print(continuous_distribution)
```

```
Hour                          24
Temperature(°C)              546
Humidity(%)                   90
Wind speed (m/s)              65
Visibility (10m)            1789
Dew point temperature(°C)    556
Solar Radiation (MJ/m2)      345
Rainfall(mm)                  61
Snowfall (cm)                 51
dtype: int64
```

```
for key,values in continuous_distribution.items():
  print(key ,":" ,values)
```

```
Hour : 24
Temperature(°C) : 546
Humidity(%) : 90
Wind speed (m/s) : 65
Visibility (10m) : 1789
Dew point temperature(°C) : 556
Solar Radiation (MJ/m2) : 345
Rainfall(mm) : 61
Snowfall (cm) : 51
```

```
for key,values in categorical_distribution.items():
  print(key,";",values)
```

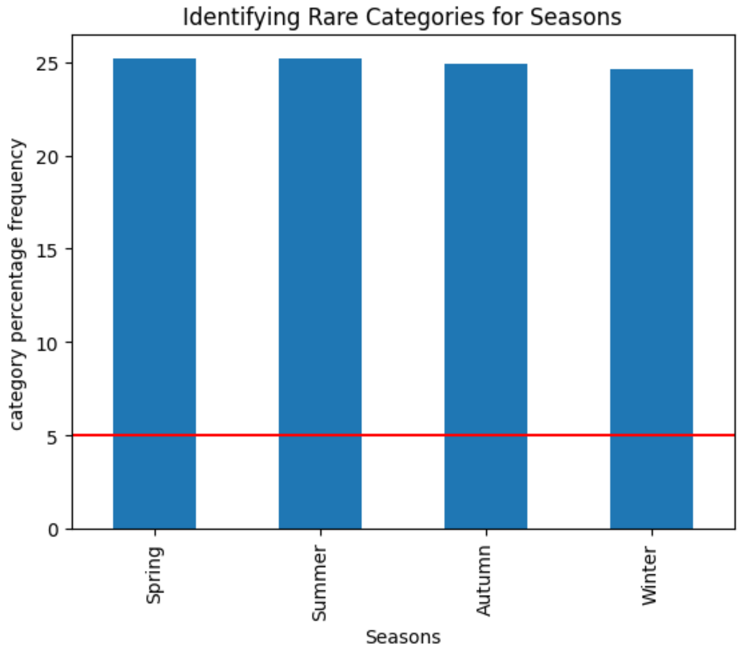
```
Seasons ; 4
Holiday ; 2
Functioning Day ; 2
```

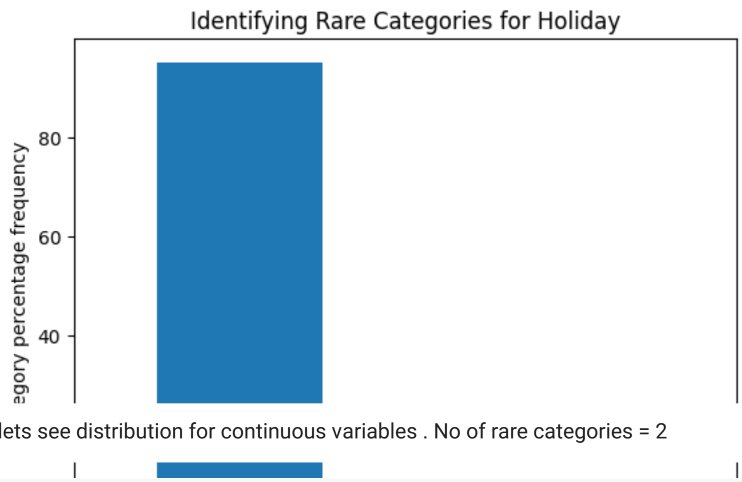
```
def rare_cat(df,var):
  cat_freq = 100* data[var].value_counts(normalize = True)
  fig = cat_freq.sort_values(ascending = False).plot.bar()
  fig.axhline(y=5, color='red')
  fig.set_ylabel('category percentage frequency')
  fig.set_xlabel(var)
  fig.set_title(f'Identifying Rare Categories for {var}')
  plt.show()
```

```
data[continuous].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Hour	8760.0	11.500000	6.922582	0.0	5.75	11.50	17.25	23.00
Temperature(°C)	8760.0	12.882922	11.944825	-17.8	3.50	13.70	22.50	39.40
Humidity(%)	8760.0	58.226256	20.362413	0.0	42.00	57.00	74.00	98.00
Wind speed (m/s)	8760.0	1.724909	1.036300	0.0	0.90	1.50	2.30	7.40
Visibility (10m)	8760.0	1436.825799	608.298712	27.0	940.00	1698.00	2000.00	2000.00
Dew point temperature(°C)	8760.0	4.073813	13.060369	-30.6	-4.70	5.10	14.80	27.20
Solar Radiation (MJ/m2)	8760.0	0.569111	0.868746	0.0	0.00	0.01	0.93	3.52
Rainfall(mm)	8760.0	0.148687	1.128193	0.0	0.00	0.00	0.00	35.00
Snowfall (cm)	8760.0	0.075068	0.436746	0.0	0.00	0.00	0.00	8.80

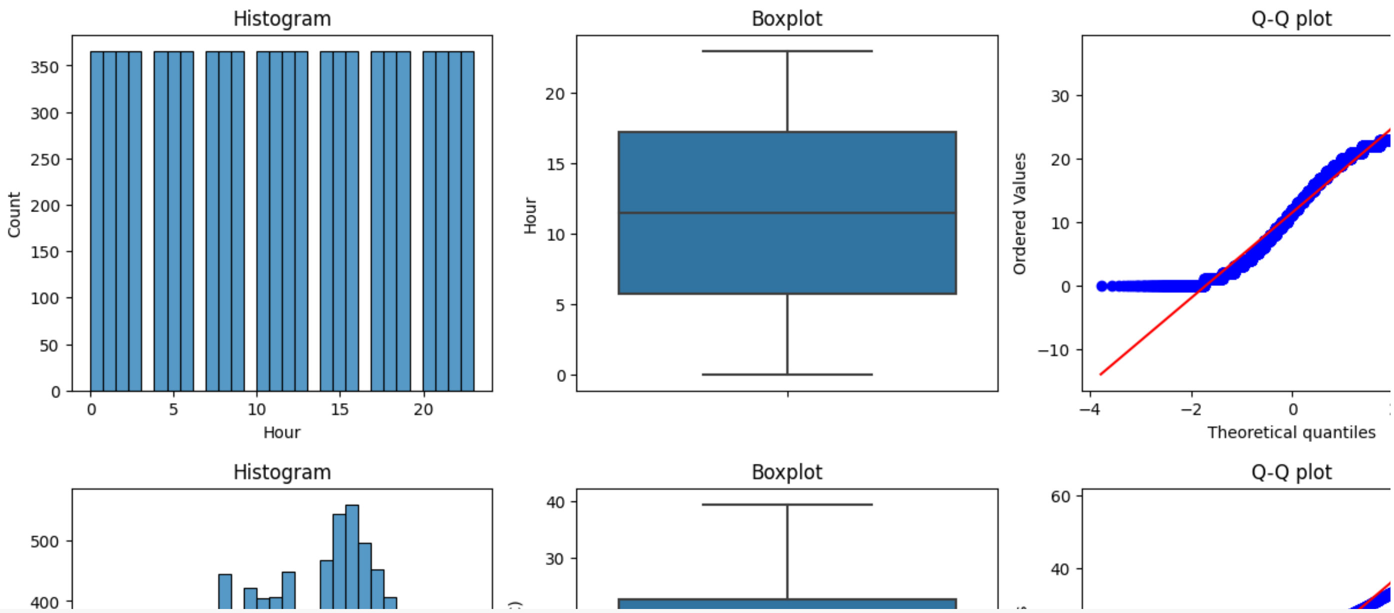
```
for var in categorical:
  rare_cat(data,var)
```





Now lets see distribution for continuous variables . No of rare categories = 2

```
for var in continuous:
    diagnostic_plots(data,var)
```




```
corrmat = data[continuous].corr().round(2)
top_corr_features = corrmat.index
plt.figure(figsize=(7, 7))
sns.heatmap(data[top_corr_features].corr(),annot=True, square=True, fmt='.2f',
            cbar_kws={"shrink": .80}, linewidths=.5, cmap='RdYlGn');
```

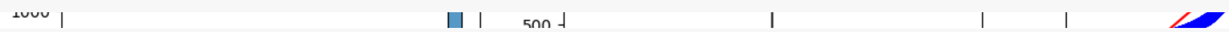
As we see 91% correlation between drew point temp and Temp & 54% drew point temp and Humidity.

It would be better if we drop drew point temp

Wind speed (m/s) Theoretical quantiles

```
#col_removal = 'Dew point temperature(°C)'
#continuous = [var for var in continuous if var != col_removal]
#continuous
```

```
#corrmat = data[continuous].corr().round(2)
#top_corr_features = corrmat.index
#plt.figure(figsize=(7, 7))
#sns.heatmap(data[top_corr_features].corr(),annot=True, square=True, fmt='.2f',
            #cbar_kws={"shrink": .80}, linewidths=.5, cmap='RdYlGn');
```



```
for category in categorical :
  plot_target_by_category(data, 'Rented Bike Count',category,'Bike Count')
```

Colncusions

1. We do not have any missing values or single value columns
2. Drew point Temparature is highly correlated with Temparature and Humidity. Removing Dew point to reduce Endogenity problems
3. Windspeed, Rainfall, Snowfall and Rainfall are left skewed and have a lot of outliers. Log Transformation or Seggregating them into Categories would be a better option.
4. There are 2 columns with Rare categories. Will use rate Categories encoder. 'Functioning Day' & 'Hour'
5. We have categorical variables in the data frame that should be converted into numerical before we run our model. Hence we need to do encoding of categorical variables.

6. Finally we will need to make sure that the continuos variables have same scale. We will need to do feature scaling for continuos variables and discrete variables.

Task2:(8 Points) : Create a pipeline of regressor and preprocessing steps

In this HW you will use KNNRegression. Use gridserach to fine tune your pipeline. The aim of the piepline is to predict the rented bike count.

bin_edges = [-0.1, 0.5, 2.5, 40.0] bin_labels = ['Low', 'Medium', 'High']

data['Solar Radiation_categorized'] = pd.cut(data['Solar Radiation (MJ/m2)'], bins=bin_edges, labels=bin_labels) data['Rainfall_categorized'] = pd.cut(data['Rainfall(mm)'], bins=bin_edges, labels=bin_labels) data['Snowfall_categorized'] = pd.cut(data['Snowfall (cm)'], bins=bin_edges, labels=bin_labels)

data.info()

for var in ['Solar Radiation_categorized','Rainfall_categorized','Snowfall_categorized']: data[var] = data[var].astype('object')#

data.info()

data = data.drop(columns = ['Solar Radiation (MJ/m2)','Rainfall(mm)','Snowfall (cm)'])

```
X = data.drop(['Rented Bike Count'], axis=1)
y = data['Rented Bike Count']
X_train,X_test, y_train, y_test = train_test_split(X,y,random_state=0,test_size=0.33)
```

```
y.head()
```

```
0    254
1    204
2    173
3    107
4     78
Name: Rented Bike Count, dtype: int64
```

```
print(X_train.count(), y_train.count())
```

```
Hour                        5869
Temperature(°C)             5869
Humidity(%)                 5869
Wind speed (m/s)            5869
Visibility (10m)            5869
Dew point temperature(°C)   5869
Solar Radiation (MJ/m2)     5869
Rainfall(mm)                5869
Snowfall (cm)               5869
Seasons                     5869
Holiday                     5869
Functioning Day             5869
dtype: int64 5869
```

As Hour has value 0 in it, one hot encoder wll only run positive values, so adding 1 to every hour

```
data.head()
```

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	Seasons	H
0	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	
1	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	
2	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	
3	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	
4	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	

```
columns_to_drop = ['Dew point temperature(°C)']
```

```
columns_to_transform = ['Wind speed (m/s)','Rainfall(mm)','Snowfall (cm)','Hour','Solar Radiation (MJ/m2)']
```

```
columns_to_scale = ['Hour',
 'Temperature(°C)',
 'Humidity(%)',
 'Wind speed (m/s)',
 'Visibility (10m)',
 'Dew point temperature(°C)',
 'Solar Radiation (MJ/m2)',
 'Rainfall(mm)',
 'Snowfall (cm)']
```

```
rare_labels =['Functioning Day']
```

```
from sklearn.base import BaseEstimator, TransformerMixin
class ConvertToNumpyArray(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return np.array(X)
```

```
from feature_engine.transformation import YeoJohnsonTransformer
from feature_engine.selection.drop_correlated_features import Variables


processing_steps = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('array_conversion',ConvertToNumpyArray()),
    ('knn',KNeighborsRegressor())
    ])
```

```
param_grid = {'knn__n_neighbors':np.arange(6,21,1)}
grid_knn = GridSearchCV(processing_steps,param_grid=param_grid, cv= 5 , return_train_score=True)
```

```
grid_knn.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn.best_params_}')
print(f'Best cross vallidation score is {grid_knn.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 8}
Best cross vallidation score is 0.48871784333597235
```

```
print(f'Test score is {grid_knn.score(X_test,y_test)}')
```

```
Test score is 0.47065176187629376
```

Now lets try with different parameters with the range of 1-20 with step of 2

```
param_grid_2 = {'knn__n_neighbors':np.arange(1,51,6)}
grid_knn_2 = GridSearchCV(processing_steps,param_grid=param_grid_2, cv= 5 , return_train_score=True)
```

```
grid_knn_2.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_2.best_params_}')
print(f'Best cross vallidation score is {grid_knn_2.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 7}
Best cross vallidation score is 0.4878761519949233
```

```
print(f'Test score is {grid_knn_2.score(X_test,y_test)}')
```

```
Test score is 0.47173795980209543
```

Lets use shuffle & split cross validation

```
param_grid_3 = {'knn__n_neighbors':np.arange(1,11,1)}
kfold = ShuffleSplit(n_splits=5,test_size=0.2,random_state=0)
grid_knn_3 = GridSearchCV(processing_steps,param_grid=param_grid_3, cv= kfold , return_train_score=True)
```

```
grid_knn_3.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_3.best_params_}')
print(f'Best cross vallidation score is {grid_knn_3.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 9}
Best cross vallidation score is 0.49116674370665203
```

```
print(f'Test score is {grid_knn_3.score(X_test,y_test)}')
```

```
Test score is 0.46923793040531203
```

```
param_grid_4 = {'knn__n_neighbors':np.arange(1,11,1)}
grid_knn_4 = GridSearchCV(processing_steps,param_grid=param_grid_4, cv= 5 , return_train_score=True)
```

```
grid_knn_4.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_4.best_params_}')
print(f'Best cross vallidation score is {grid_knn_4.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 8}
Best cross vallidation score is 0.48871784333597235
```

```
print(f'Test score is {grid_knn_4.score(X_test,y_test)}')
```

```
Test score is 0.47065176187629376
```

Lets try without Rare Label Encoder

```
processing_steps_2 = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('array_conversion',ConvertToNumpyArray()),
    ('knn',KNeighborsRegressor())
    ])
```

```
param_grid_5 = {'knn__n_neighbors':np.arange(1,11,1)}
grid_knn_5 = GridSearchCV(processing_steps_2,param_grid=param_grid_5, cv= 5 , return_train_score=True)
```

```
grid_knn_5.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_5.best_params_}')
print(f'Best cross vallidation score is {grid_knn_5.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 8}
Best cross vallidation score is 0.48871784333597235
```

```
from feature_engine.transformation import YeoJohnsonTransformer
from feature_engine.selection.drop_correlated_features import Variables


processing_steps_3 = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('scaler',StandardScaler()),
```

```
    ('array_conversion',ConvertToNumpyArray()),
    ('knn',KNeighborsRegressor())
    ])
```

```
param_grid_6 = {'knn__n_neighbors':np.arange(1,11,1)}
grid_knn_6 = GridSearchCV(processing_steps_3,param_grid=param_grid_6, cv= 5 , return_train_score=True)
```

```
grid_knn_6.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ DropFeatures
▸ RareLabelEncoder
▸ OneHotEncoder
▸ YeoJohnsonTransformer
▸ StandardScaler
▸ ConvertToNumpyArray
▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_6.best_params_}')
print(f'Best cross vallidation score is {grid_knn_6.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 5}
Best cross vallidation score is 0.7852097387187529
```

When using Standard scaler, we get good CV score.Lets explore with different parameters

```
param_grid_7 = {'knn__n_neighbors':np.arange(1,21,5)}
grid_knn_7 = GridSearchCV(processing_steps_3,param_grid=param_grid_7, cv= 5 , return_train_score=True)
```

```
grid_knn_7.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ DropFeatures
▸ RareLabelEncoder
▸ OneHotEncoder
▸ YeoJohnsonTransformer
▸ StandardScaler
▸ ConvertToNumpyArray
▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_7.best_params_}')
print(f'Best cross vallidation score is {grid_knn_7.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 6}
Best cross vallidation score is 0.7832774230744043
```

```
param_grid_8 = {'knn__n_neighbors':np.arange(4,8,1)}
grid_knn_8 = GridSearchCV(processing_steps_3,param_grid=param_grid_8, cv= 5 , return_train_score=True)
```

```
grid_knn_8.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ StandardScaler
```

```
print(f'Best parameter is {grid_knn_8.best_params_}')
print(f'Best cross vallidation score is {grid_knn_8.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 5}
Best cross vallidation score is 0.7852097387187529
```

Lets try with MinMax scaler

```
from feature_engine.transformation import YeoJohnsonTransformer
from feature_engine.selection.drop_correlated_features import Variables


processing_steps_4 = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('scaler',MinMaxScaler()),
    ('array_conversion',ConvertToNumpyArray()),
    ('knn',KNeighborsRegressor())
    ])
```

```
param_grid_9 = {'knn__n_neighbors':np.arange(1,11,1)}
grid_knn_9 = GridSearchCV(processing_steps_4,param_grid=param_grid_9, cv= 5 , return_train_score=True)
```

```
grid_knn_9.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ MinMaxScaler
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_9.best_params_}')
print(f'Best cross vallidation score is {grid_knn_9.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 5}
Best cross vallidation score is 0.7918688450682064
```

As this is a big data set, ets try with shuffle split cross validation

```
param_grid_10 = {'knn__n_neighbors':np.arange(1,11,1)}
kfold2 = ShuffleSplit(n_splits = 5, test_size=0.25,random_state=0)
grid_knn_10 = GridSearchCV(processing_steps_4,param_grid=param_grid_10,cv=kfold2,return_train_score=True)
```

```
grid_knn_10.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ DropFeatures
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ YeoJohnsonTransformer
  ▸ MinMaxScaler
  ▸ ConvertToNumpyArray
  ▸ KNeighborsRegressor
```

```
print(f'Best parameter is {grid_knn_10.best_params_}')
print(f'Best cross vallidation score is {grid_knn_10.best_score_}')
```

```
Best parameter is {'knn__n_neighbors': 4}
Best cross vallidation score is 0.7786887001857846
```

We see that grid_knn_9 is better with mean CV dcore of .7918 and knn=5. Lets test the model with that

```
print(f'The test score of the grid_9 odel is {grid_knn_9.score(X_test,y_test)}')
```

```
The test score of the grid_9 odel is 0.8034474017869272
```

I would consider my Model 9 as the better model.