In [749]:

```
# Importing required packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# Seed value
seed=200
```

QUESTION A :

Derivation of the given problem

- To derive the result, we take the derivative of the ridge regression objective function with respect to β and set it to zero:

$$\nabla_\beta(\lambda\beta^T\beta + (y - X\beta)^T(y - X\beta)) = 0$$

- After derivating it, we get the following form:

$$\lambda\beta + X^T(y - X\beta) = 0$$

$$\lambda\beta + X^Ty - X^TX\beta = 0$$

$$(X^TX + \lambda I)\beta = X^Ty$$

- Hence we can say that the given form in the question holds (HENCE PROVED)

- Now we can get β in the form of :

$$\lambda\beta = X^Ty - X^TX\beta$$

$$\lambda\beta = X^T(y - X\beta)$$

$$\beta = \frac{1}{\lambda}[X^T(y - X\beta)]$$

- Now, considering $\alpha = \frac{1}{\lambda}(y - X\beta)$. Then, $\beta = \frac{1}{\lambda}X^T(y - X\beta)$ can be written as $\beta = X^T\alpha$. (HENCE PROVED)

- Substituting $\beta = X^T\alpha$ into $(X^TX + \lambda I)\beta = X^Ty$, we get

$$(X^TX + \lambda I)X^T\alpha = X^Ty$$

$$X(X^TX + \lambda I)\alpha = y$$

$$\alpha = (X^TX + \lambda I)^{-1}y$$

Using the kernel function idea, we can replace X^TX by the kernel matrix K. Therefore,
$\alpha = (K + \lambda I)^{-1}y$.

To represent the inference function $\langle\beta, x\rangle$ using α and kernels, we have

$\langle\beta, x\rangle = x^T\beta = x^TX^T\alpha = \sum_{i=1}^n \alpha_i k(x, x_i)$,

where $k(x, x_i)$ is the kernel function representing the similarity between x and x_i. Thus, we can use this kernel representation to make predictions in kernel ridge regression.

QUESTION B:

- Read the data set in Data Q2.csv into a pandas dataframe.

In [750]:

```
df=pd.read_csv("Data_Q2.csv")
df
```

Out[750]:

	Temperature	Humidity	Wind Speed	Flow	Consumption
0	5.578	93.00	0.082	0.185	5935.174070
1	15.510	64.38	0.085	0.133	6044.657863
2	15.730	64.21	0.084	0.152	6061.944778
3	15.620	65.22	0.083	0.145	6108.043217
4	15.450	67.69	0.083	0.189	6119.567827
...	...	...	...	...	...
995	17.330	42.24	4.917	31.540	9443.855422
996	7.010	76.40	4.920	65.890	9449.638554
997	14.810	82.30	4.913	0.159	9449.638554
998	12.090	77.40	0.073	0.104	9449.638554
999	16.680	64.92	0.079	112.400	9449.990000

1000 rows × 5 columns

In [751]:

```
# Check for null values
df.isnull().sum()
```

Out[751]:

```
Temperature    0
Humidity       0
Wind Speed     0
Flow           0
Consumption    0
dtype: int64
```

QUESTION C :

- Perform standardization of each column in the data frame and create a new data frame.

In [752]:

```
df_new=(df-df.mean())/df.std()
df_new.describe()
```

Out[752]:

	Temperature	Humidity	Wind Speed	Flow	Consumption
count	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03
mean	2.131628e-16	1.492140e-16	6.927792e-17	1.421085e-17	-1.477929e-15
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-2.170116e+00	-2.907045e+00	-6.501680e-01	-6.677073e-01	-3.448482e+00
25%	-7.397174e-01	-7.556299e-01	-6.423859e-01	-6.651830e-01	-6.787327e-01
50%	1.896711e-01	2.419695e-01	-6.405548e-01	-5.240777e-01	1.095636e-01
75%	7.782641e-01	8.867375e-01	1.570037e+00	3.081102e-01	7.164880e-01
max	2.467526e+00	1.539466e+00	1.608948e+00	6.404628e+00	1.934698e+00

QUESTION D:

- Split the data into two sets such that 80% of the data is considered as set T1 and 20% of the data is considered as set T2. Justify if the splits T1 and T2 have similar spread in Consumption column.

In [753]:

```
# Split data into T1 and T2
T1,T2 = train_test_split(df_new, test_size=0.2,random_state=seed)

# Check the shape of T1 and T2
print("Shape of T1 (X_train, y_train):",T1.shape)
print("Shape of T2 (X_test, y_test):", T2.shape)


# Calculate mean and standard deviation of Consumption for T1 and T2
consumption_mean_T1 = np.mean(T1['Consumption'])
consumption_std_T1 = np.std(T1['Consumption'])
consumption_mean_T2 = np.mean(T2['Consumption'])
consumption_std_T2 = np.std(T2['Consumption'])

# Print the results
print("\nMean Consumption in T1:", consumption_mean_T1)
print("\nVariance of Consumption in T1:", consumption_std_T1**2)
print("\nMean Consumption in T2:", consumption_mean_T2)
print("\nVariance of Consumption in T2:", consumption_std_T2**2)
```

```
Shape of T1 (X_train, y_train): (800, 5)
Shape of T2 (X_test, y_test): (200, 5)

Mean Consumption in T1: 0.002711724815472447

Variance of Consumption in T1: 1.0078065744582132

Mean Consumption in T2: -0.01084689926189765

Variance of Consumption in T2: 0.9636266331376501
```

COMMENTS :

- To check the spread of the Consumption column we have to check the mean and variance of the split datasets. So I have computed the mean and variance of datasets T1 and T2. As we can see that, the mean and variances are nearly similar to each other so we can say that the Consumption column has similar spread is justified.

QUESTION E :

- Using T1 as training data, train kernel ridge regression model. Use RBF kernel and tune the gamma parameter using 5-fold cross-validation.

In [754]:

```
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Split T1 into features (X) and target (y)
X_train = T1.drop('Consumption', axis=1)
y_train = T1['Consumption']

# Split T2 into features (X) and target (y)
X_test = T2.drop('Consumption', axis=1)
y_test = T2['Consumption']

# Define the kernel ridge regression model with RBF kernel
model = KernelRidge(kernel='rbf')

# Define the range of hyperparameters to tune
param_grid = {'alpha':np.array([0.001,0.01,0.1,1,10,100]) ,'gamma': [0.01, 0.1, 1,5,10,
15,20,50,100]}

# Define the grid search object with 5-fold cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_root_mean_squared_erro
r')

# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding mean squared error
print('Best hyperparameters:', grid_search.best_params_)
print('Root Mean squared error:',-grid_search.best_score_ )
```

```
Best hyperparameters: {'alpha': 1.0, 'gamma': 10}
Root Mean squared error: 0.8890653757120288
```

QUESTION F :

- Compute and display the RMSE and R2 values on the training set T1 and test set T2.

In [755]:

```
from sklearn.metrics import mean_squared_error, r2_score

# make predictions on T1 and T2
y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test)

# calculate RMSE on T1 and T2
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# calculate R2 on T1 and T2
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

print("RMSE on Train Set: {:.2f}".format(rmse_train))
print("RMSE on Test Set: {:.2f}".format(rmse_test))
print("\nR2 on Train Set: {:.2f}".format(r2_train))
print("R2 on Test Set: {:.2f}".format(r2_test))
```

```
RMSE on Train Set: 0.66
RMSE on Test Set: 0.83

R2 on Train Set: 0.57
R2 on Test Set: 0.29
```

QUESTION G:

- Consider the original data in Data Q2.csv and load it into a different pandas dataframe called frame2. Add another column with name Class to the data frame frame2 such that the following hold:

1. samples having Consumption values ≤ 6500 are labeled as class 1
2. samples having Consumption values > 6500 and ≤ 7000 are labeled as class 2
3. samples having Consumption values > 7000 and ≤ 7500 are labeled as class 3
4. samples having Consumption values > 7500 and ≤ 8000 are labeled as class 4
5. samples having Consumption values > 8000 and ≤ 8500 are labeled as class 5
6. samples having Consumption values > 8500 and ≤ 9000 are labeled as class 6
7. samples having Consumption values > 9000 are labeled as class 7

In [756]:

```
# Load data into dataframe
frame2 = pd.read_csv("Data_Q2.csv")

# Add Class column
conditions = [
    frame2['Consumption'] <= 6500,
    (frame2['Consumption'] > 6500) & (frame2['Consumption'] <= 7000),
    (frame2['Consumption'] > 7000) & (frame2['Consumption'] <= 7500),
    (frame2['Consumption'] > 7500) & (frame2['Consumption'] <= 8000),
    (frame2['Consumption'] > 8000) & (frame2['Consumption'] <= 8500),
    (frame2['Consumption'] > 8500) & (frame2['Consumption'] <= 9000),
    frame2['Consumption'] > 9000
]
choices = [1, 2, 3, 4, 5, 6,7]

frame2['Class'] = pd.Series(np.select(conditions, choices))

frame2

print("Number of datapoints in unique classes :", np.unique(frame2['Class'],return_coun
ts=True))
```

```
Number of datapoints in unique classes : (array([1, 2, 3, 4, 5, 6, 7]), ar
ray([ 15,  27, 103, 218, 280, 255, 102], dtype=int64))
```

- Since there is a class imbalance we will have to resample the classes before training for proper classification.

QUESTION H:

- Perform standardization of samples in frame2 belonging to each class separately. Ignore Class column during standardization procedure.

In [757]:

```
# Group samples by class label
groups = frame2.groupby('Class')
print(groups)
# Standardize each group separately
for name, group in groups:
    # Get the indices of columns to standardize (exclude 'Class')
    col_indices = [i for i, col in enumerate(group.columns) if col != 'Class']
    # Standardize the columns using mean and std of the group
    group.iloc[:, col_indices] = (group.iloc[:, col_indices] - group.iloc[:, col_indice
s].mean()) / group.iloc[:, col_indices].std()

    # Replace the group in the original dataframe with standardized values
    frame2.loc[group.index] = group
#frame2[frame2['Class']==5].describe()
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001EED1B868B0
>
```

In [758]:

```
frame2.head()
```

Out[758]:

	Temperature	Humidity	Wind Speed	Flow	Consumption	Class
0	-1.633251	1.510316	-0.491477	-0.353228	-1.860482	1
1	1.046563	-1.473479	0.561688	-0.398291	-1.259987	1
2	1.105922	-1.491202	0.210633	-0.381826	-1.165172	1
3	1.076242	-1.385904	-0.140422	-0.387892	-0.912332	1
4	1.030374	-1.128393	-0.140422	-0.349762	-0.849122	1

QUESTION I:

- Split frame2 into train and test splits T3 and T4, such that the samples in T3 are the same as in T1. Consider T3 as training set, ignore the Consumption column and considering Class as labels, train a kernel SVM model with RBF kernel. Tune gamma parameter using 5 fold cross-validation. Take care of class imbalance issues if they exist.

In [759]:

```
# Taking care of class imbalance
from imblearn.combine import SMOTETomek,SMOTEENN
from imblearn.over_sampling import SMOTE
X=frame2.drop(columns=['Class'])
y=frame2['Class']

# Applying hybrid resampling. The strategy can be changed as required.
smt = SMOTETomek(random_state=seed,sampling_strategy='all')
#smt=SMOTE(random_state=seed)

# Fit the model to generate the data.
X_res, y_res = smt.fit_resample(X, y)  # Resmapling done to generate nearabout equal cl
asses

frame2_new=pd.DataFrame(X_res)
frame2_new['Class']=y_res
print("Number of datapoints in unique classes :", np.unique(frame2_new['Class'],return_
counts=True))
```

```
Number of datapoints in unique classes : (array([1, 2, 3, 4, 5, 6, 7]), ar
ray([279, 275, 269, 238, 236, 246, 263], dtype=int64))
```

In [760]:

```
frame2_new.head()
```

Out[760]:

	Temperature	Humidity	Wind Speed	Flow	Consumption	Class
0	-1.633251	1.510316	-0.491477	-0.353228	-1.860482	1
1	1.046563	-1.473479	0.561688	-0.398291	-1.259987	1
2	1.105922	-1.491202	0.210633	-0.381826	-1.165172	1
3	1.076242	-1.385904	-0.140422	-0.387892	-0.912332	1
4	1.030374	-1.128393	-0.140422	-0.349762	-0.849122	1

- After applying resampling technique we can see that although we cannot achieve exactly equal number of classes, I have achieved a near similar class distribution.
- The reason we cannot achieve the equal number of classes is because the number of samples in the majority class may not be divisible by the number of classes and also by the number of samples.

In [761]:

```
# Split data into T3 and T4
'''
Since in the question it is explicitly mentioned that we must keep the samples in T1 an
d T3 same. However it is also written that any class
imbalance is to be taken care of. So I will be using startify function in this case.
'''
T3,T4 = train_test_split(frame2_new, test_size=0.2,stratify=frame2_new['Class'],random_
state=seed)  # The seed value replicates the samples in T1 in T3
T3.drop(columns=['Consumption'],inplace=True)
T4.drop(columns=['Consumption'],inplace=True)
print(T3.shape,T4.shape)
print("Number of datapoints in unique classes :", np.unique(T3['Class'],return_counts=T
rue))
```

```
(1444, 5) (362, 5)
Number of datapoints in unique classes : (array([1, 2, 3, 4, 5, 6, 7]), ar
ray([223, 220, 215, 190, 189, 197, 210], dtype=int64))
```

In [762]:

```
T3.head()
```

Out[762]:

	Temperature	Humidity	Wind Speed	Flow	Class
580	0.289994	1.051699	1.686249	-0.217260	6
421	-1.104974	-0.053003	1.333921	-0.234959	5
1267	0.858229	-0.487858	-0.349970	-0.353097	2
506	0.696646	-2.409560	-0.751485	-0.729339	5
1466	-0.423593	0.807291	-0.541347	-0.311039	3

- Here we can observe that we get near similar class distribution with slight deviation which will not affect much. This is within the capabilties of the model.
- Also perfect class balance of 1:1 is not possible due to odd number of samples in majority class and the problem mentioned above.

In [763]:

```
# train a kernel SVM model with RBF kernel
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# define the parameter grid for hyperparameter tuning
param_grid = {'C':np.array([0.001,0.01,0.1,1,10,50,100]) ,'gamma': np.array([0.001,0.0
1,0.1,1,10,100,1000])}
cv = StratifiedKFold(n_splits=5)
# define the kernel SVM model
svm = SVC(kernel='rbf', class_weight='balanced')  # We have used class weight balanced
to take care of class balance issues in training

# perform hyperparameter tuning using 5-fold cross-validation
best_classification_model = GridSearchCV(svm, param_grid,cv=cv,scoring='accuracy')
best_classification_model .fit(T3.iloc[:, :-1], T3['Class'])

# get the best hyperparameters and the corresponding model
best_params = best_classification_model .best_params_
best_svm = best_classification_model .best_estimator_

print("Best hyperparameters:", best_params)
print("Training accuracy:", best_svm.score(T3.iloc[:, :-1], T3['Class']))
print("Test accuracy:", best_svm.score(T4.iloc[:, :-1], T4['Class']))
```

```
Best hyperparameters: {'C': 100.0, 'gamma': 1.0}
Training accuracy: 0.9889196675900277
Test accuracy: 0.9088397790055248
```

QUESTION J:

- Now consider samples belonging to a particular class i in T3: build a kernel ridge regression model with RBF kernel (ignore the Class column for this task). Tune gamma parameter using 5 fold cross-validation restricted to samples belonging to only class i. Repeat this for each class. Thus, at the end, for each class i, you would now have a kernel ridge regression model Mi.

In [764]:

```
T3,T4 = train_test_split(frame2_new, test_size=0.2,stratify=frame2_new['Class'],random_
state=seed)  # The seed value replicates the samples in T1 in T3
#T3.drop(columns=['Consumption'],inplace=True)
#T4.drop(columns=['Consumption'],inplace=True)
print(T3.shape,T4.shape)
print("Number of datapoints in unique classes :", np.unique(T3['Class'],return_counts=T
rue))
```

```
(1444, 6) (362, 6)
Number of datapoints in unique classes : (array([1, 2, 3, 4, 5, 6, 7]), ar
ray([223, 220, 215, 190, 189, 197, 210], dtype=int64))
```

In [765]:

```
T3
```

Out[765]:

	Temperature	Humidity	Wind Speed	Flow	Consumption	Class
580	0.289994	1.051699	1.686249	-0.217260	-1.052016	6
421	-1.104974	-0.053003	1.333921	-0.234959	-0.231781	5
1267	0.858229	-0.487858	-0.349970	-0.353097	-1.001939	2
506	0.696646	-2.409560	-0.751485	-0.729339	1.074272	5
1466	-0.423593	0.807291	-0.541347	-0.311039	1.290980	3
...	...	...	...	...	...	...
424	-1.756365	0.707177	-0.748033	1.176771	-0.158708	5
1102	1.042218	-1.185274	-0.085383	-0.354789	-0.898672	1
188	0.359424	0.843192	1.500722	-0.725206	-0.719937	4
11	-0.755809	0.790952	-0.842531	3.256180	1.047180	1
425	-0.396025	-0.186230	-0.749327	-0.024521	-0.158708	5

1444 rows × 6 columns

In [766]:

```
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, mean_squared_error

# create a dictionary to store the models for each class
all_models = {}

# loop over the classes
for i in choices:
    # filter out samples belonging to class i from T3
    Ti = T3[T3['Class'] == i].drop(columns=['Class'],axis=1)
    # split Ti into X and y
    X = Ti.drop(columns=['Consumption']).to_numpy()
    y = Ti['Consumption'].to_numpy()

    # tune gamma using 5-fold cross-validation
    # Define the range of hyperparameters to tune
    param_grid = {'alpha':np.array([0.001,0.01,0.1,0.5,0.25,1,10]) ,'gamma': [0.01, 0.
1, 0.5,0.25,1, 2, 3, 5,10,15,18,20]}

    # Define the kernel ridge regression model with RBF kernel
    model = KernelRidge(kernel='rbf')
    # Define the grid search object with 5-fold cross-validation
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_root_mean_squared_
error')#neg_mean_squared_error

    # Fit the grid search object to the training data
    grid_search.fit(X, y)

    best_model=KernelRidge(kernel='rbf',alpha=grid_search.best_params_['alpha'],gamma=g
rid_search.best_params_['gamma'])

    best_model.fit(X, y)

    print(i,grid_search.best_params_,-grid_search.best_score_ )

    # store the model for class i
    all_models[i] = best_model
```

```
1 {'alpha': 0.01, 'gamma': 2} 0.11348093114024409
2 {'alpha': 0.1, 'gamma': 2} 0.41678802993354785
3 {'alpha': 0.25, 'gamma': 3} 0.714454325989627
4 {'alpha': 0.5, 'gamma': 10} 0.8410808729285927
5 {'alpha': 0.5, 'gamma': 15} 0.9314543121498076
6 {'alpha': 0.5, 'gamma': 2} 0.939312423038235
7 {'alpha': 0.1, 'gamma': 10} 0.6547994274855914
```

In [767]:

```
print(all_models)
```

```
{1: KernelRidge(alpha=0.01, gamma=2, kernel='rbf'), 2: KernelRidge(alpha=
0.1, gamma=2, kernel='rbf'), 3: KernelRidge(alpha=0.25, gamma=3, kernel='r
bf'), 4: KernelRidge(alpha=0.5, gamma=10, kernel='rbf'), 5: KernelRidge(al
pha=0.5, gamma=15, kernel='rbf'), 6: KernelRidge(alpha=0.5, gamma=2, kerne
l='rbf'), 7: KernelRidge(alpha=0.1, gamma=10, kernel='rbf')}
```

COMMENTS :

- Here I have tuned the hyperparameter alpha as well to get better accuracy. Also I have tuned alpha for initial case also.

QUESTION K:

- For testing (or) inference, implement the following procedure: for any sample, first predict the class label as j and then based on the class label j, use model Mj to predict the Consumption value. Using this procedure, find the RMSE values for T3 and T4.

In [768]:

```
'''
T3,T4 = train_test_split(frame2, test_size=0.2,stratify=frame2['Class'],random_state=se
ed)  # The seed value replicates the samples in T1 in T3
#T3.drop(columns=['Consumption'],inplace=True)
#T4.drop(columns=['Consumption'],inplace=True)
print(T3.shape,T4.shape)
print("Number of datapoints in unique classes :", np.unique(T3['Class'],return_counts=T
rue))
'''
```

Out[768]:

```
'\nT3,T4 = train_test_split(frame2, test_size=0.2,stratify=frame2[\'Class
\'],random_state=seed)  # The seed value replicates the samples in T1 in T
3\n#T3.drop(columns=[\'Consumption\'],inplace=True)\n#T4.drop(columns=[\'C
onsumption\'],inplace=True)\nprint(T3.shape,T4.shape)\nprint("Number of da
tapoints in unique classes :", np.unique(T3[\'Class\'],return_counts=Tru
e))\n'
```

In [769]:

```
from sklearn.metrics import mean_squared_error,r2_score

# First predict the class label for each sample in T3 and T4
y_pred_T3_class = best_classification_model.predict(T3.drop(['Class','Consumption'], ax
is=1))#'Class',
y_pred_T4_class = best_classification_model.predict(T4.drop(['Class','Consumption'], ax
is=1))#'Class',

# Drop the Class labels from T3 and add new predicted values
T3.drop(['Class'], axis=1,inplace=True)
T4.drop(['Class'], axis=1,inplace=True)

# Add the predicted class labels to it
T3['pred_class']=y_pred_T3_class
T4['pred_class']=y_pred_T4_class

rmse_T3=0
rmse_T4=0

for i in choices:
    # Filter samples belonging to class i
    T3_i = T3[T3['pred_class'] == i].drop(['pred_class', 'Consumption'],axis=1).to_nump
y()
    T4_i = T4[T4['pred_class'] == i].drop(['pred_class', 'Consumption'],axis=1).to_nump
y()

    # Predict Consumption using the corresponding kernel ridge regression model
    y_pred_T3_i = all_models[i].predict(T3_i)
    y_pred_T4_i = all_models[i].predict(T4_i)

    # Actual Values
    y3=T3[T3['pred_class'] == i]['Consumption'].to_numpy()
    y4=T4[T4['pred_class'] == i]['Consumption'].to_numpy()

    # Compute the RMSE values for T3 and T4
    rmse_T3 = np.sqrt(rmse_T3**2+mean_squared_error(y_pred_T3_i,y3))
    rmse_T4 = np.sqrt(rmse_T4**2+mean_squared_error(y_pred_T4_i,y4))

    print(f"\nR2 score of {i} th class in Train set :",r2_score(y_pred_T3_i,y3))
    print(f"R2 score of {i} th class in Test set :",r2_score(y_pred_T4_i,y4))
    print(f"RMSE score of {i} th class in Train set :",np.sqrt(mean_squared_error(y3,y_
pred_T3_i)))
    print(f"RMSE score of {i} th class in Test set :",np.sqrt(mean_squared_error(y4,y_p
red_T4_i)))

print("\nRMSE for T3:", rmse_T3)
print("RMSE for T4:", rmse_T4)
```

```
R2 score of 1 th class in Train set : 0.9774560390885241
R2 score of 1 th class in Test set : 0.9476283320570961
RMSE score of 1 th class in Train set : 0.11648141760952051
RMSE score of 1 th class in Test set : 0.16950677801845276

R2 score of 2 th class in Train set : 0.7706746730427432
R2 score of 2 th class in Test set : 0.7323234567552668
RMSE score of 2 th class in Train set : 0.3747413400122353
RMSE score of 2 th class in Test set : 0.40938317428544435

R2 score of 3 th class in Train set : 0.6600692675833802
R2 score of 3 th class in Test set : 0.3210089729713316
RMSE score of 3 th class in Train set : 0.4357073548314661
RMSE score of 3 th class in Test set : 0.5373502626075388

R2 score of 4 th class in Train set : 0.680912365292819
R2 score of 4 th class in Test set : -1.4210645599286735
RMSE score of 4 th class in Train set : 0.4016102473534042
RMSE score of 4 th class in Test set : 0.8424410802920868

R2 score of 5 th class in Train set : 0.3937769793451853
R2 score of 5 th class in Test set : -3.548585440533498
RMSE score of 5 th class in Train set : 0.49172683748056023
RMSE score of 5 th class in Test set : 1.0209804878623017

R2 score of 6 th class in Train set : 0.14073456412956242
R2 score of 6 th class in Test set : -4.248200868795875
RMSE score of 6 th class in Train set : 0.5623184050419975
RMSE score of 6 th class in Test set : 1.0494573144246653

R2 score of 7 th class in Train set : 0.9603738578208446
R2 score of 7 th class in Test set : 0.5879576271057683
RMSE score of 7 th class in Train set : 0.17275336629598495
RMSE score of 7 th class in Test set : 0.4978967275591821

RMSE for T3: 1.0454528584829583
RMSE for T4: 1.8937904774233683
```

COMMENTS :

- From the above results we can see that some classes have a higher r2 score and lower RMSE than some other classes. This can be attributed to the fact that first we have to classify the dataset into correct class and then predict the correct consumption value. So we have to factor in the classification results and the wrong model being applied.

QUESTION L:

- Compare and contrast the RMSE values obtained in part (f) and part (k). Using your observations, suggest when the two-stage approach of classification-followed-by-regression would be useful when compared to the simple regression approach on the full data set.

COMMENTS :

- From the RMSE results of (f) and (k) we get that the RMSE of the k part is greater than f.
- This can be due to the fact that proper and equal resampling did not take place in this case as there was a huge disparity in initial the number of classes given. So it was not possible to exactly resample each class to the desired values.
- These resampling techniques may not be always correct as the data is synthetically generated and may include noise.
- However, we are getting very good results in some classes whereas very bad data in others.
- Another fact to keep in mind is that the standardization is different for different classes whereas in simple regression it is a uniform standardization. So, if a class is wrongly classified is judged with another model then the error becomes significant.
- The two step approach can be used when we have balanced classes and the accuracy is above 95%. Here that is not the case.
- Also we should ensure there is very less noise in the dataset. Standardization should be uniform.