LOVELY PROFESSIONAL UNIVERSITY

MAJOR PROJECT REPORT

On dataset: Wholesale Customers Data Set

Name: Krishna Mohan Mishra

Reg no: 11605503

Roll no: A10

Submitted to: Mr. GokulNath K

OBJECTIVE:

1. Preparing data (Feature Identification) .
2. Choosing a model (Comparative study of minimum 3 models).
3. Training (Reiterating for optimization) .
4. Evaluation (Highest accuracy with confusion matrix)
5. Hyperparameter tuning.

DataSet:

Wholesale Customers Data Set

http://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20data.csv

Data Set Information:
Wholesale Customer dataset refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. Number of Instances: 440, Number of Attributes: 8, Data Set Characteristics: Multivariate, Attribute Characteristics: Integer

Data Set Characteristics:	Multivariate	**Number of Instances:**	440	**Area:**	Business
Attribute Characteristics:	Integer	**Number of Attributes:**	8	**Date Donated**	2014-03-31
Associated Tasks:	Classification, Clustering	**Missing Values?**	N/A	**Number of Web Hits:**	270339

Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_	Delicassen
2	3	12669	9656	7561	214	2674	1338
2	3	7057	9810	9568	1762	3293	1776
2	3	6353	8808	7684	2405	3516	7844
1	3	13265	1196	4221	6404	507	1788
2	3	22615	5410	7198	3915	1777	5185
2	3	9413	8259	5126	666	1795	1451
2	3	12126	3199	6975	480	3140	545
2	3	7579	4956	9426	1669	3321	2566
1	3	5963	3648	6192	425	1716	750
2	3	6006	11093	18881	1159	7425	2098
2	3	3366	5403	12974	4400	5977	1744
2	3	13146	1124	4523	1420	549	497
2	3	31714	12319	11757	287	3881	2931
2	3	21217	6208	14982	3095	6707	602

Attribute Information:

1. FRESH: annual spending (m.u.) on fresh products (Continuous)
2. MILK: annual spending (m.u.) on milk products (Continuous)
3. GROCERY: annual spending (m.u.)on grocery products (Continuous)
4. FROZEN: annual spending (m.u.)on frozen products (Continuous)
5. DETERGENTS_PAPER: annual spending (m.u.) on detergents and paper products (Continuous)
6. DELICATESSEN: annual spending (m.u.)on and delicatessen products (Continuous)
7. CHANNEL: customersale Channel - Horeca (Hotel/Restaurant/Cafe) or Retail channel (Nominal)
8. REGION: customersale Region - Lisnon, Oporto or Other (Nominal)

Objective 1: Preparing data

Machine learning helps us find patterns in data—patterns we then use to make predictions about new data points. To get those predictions right, we must

construct the data set and *transform* the data correctly.

The quality of the data and the amount of useful information that it contains are key factors that determine how well a machine learning algorithm can learn. Therefore, it is absolutely critical that we make sure to examine and preprocess a dataset before we feed it to a learning algorithm.

#import dataset

In this we import libraries and dataset and check if the dataset missing?

```
import pandas as pd
import numpy as np
# Load the wholesale customers dataset
#check whether any dataset missing ?
try:
    data = pd.read_csv("customers.csv")
    data.drop(['Region', 'Channel'], axis = 1, inplace = True)
    print( "Wholesale customers dataset has {} samples with {} features each.".format(*data.shape))
except:
    print ("Dataset could not be loaded. Is the dataset missing?")


stats = data.describe()
print(stats)
# Using data.loc to filter a pandas DataFrame
data.loc[[100, 200, 300],:]
data.columns
# Select three indices of your choice you wish to sample from the dataset
indices = [43, 12, 39]

# Create a DataFrame of the chosen samples
# .reset_index(drop = True) resets the index from 0, 1 and 2 instead of 100, 200 and 300
samples = pd.DataFrame(data.loc[indices], columns = data.columns).reset_index(drop = True)
print ("Chosen samples of wholesale customers dataset:")
```

```
Wholesale customers dataset has 440 samples with 6 features each.
                Fresh          Milk       Grocery        Frozen  \
count      440.000000    440.000000    440.000000    440.000000
mean     12000.297727   5796.265909   7951.277273   3071.931818
std      12647.328865   7380.377175   9503.162829   4854.673333
min          3.000000     55.000000      3.000000     25.000000
25%       3127.750000   1533.000000   2153.000000    742.250000
50%       8504.000000   3627.000000   4755.500000   1526.000000
75%      16933.750000   7190.250000  10655.750000   3554.250000
max     112151.000000  73498.000000  92780.000000  60869.000000

       Detergents_Paper    Delicassen
count        440.000000    440.000000
mean        2881.493182   1524.870455
std         4767.854448   2820.105937
min            3.000000      3.000000
25%          256.750000    408.250000
50%          816.500000    965.500000
75%         3922.000000   1820.250000
max        40827.000000  47943.000000
```

#train_test_split:

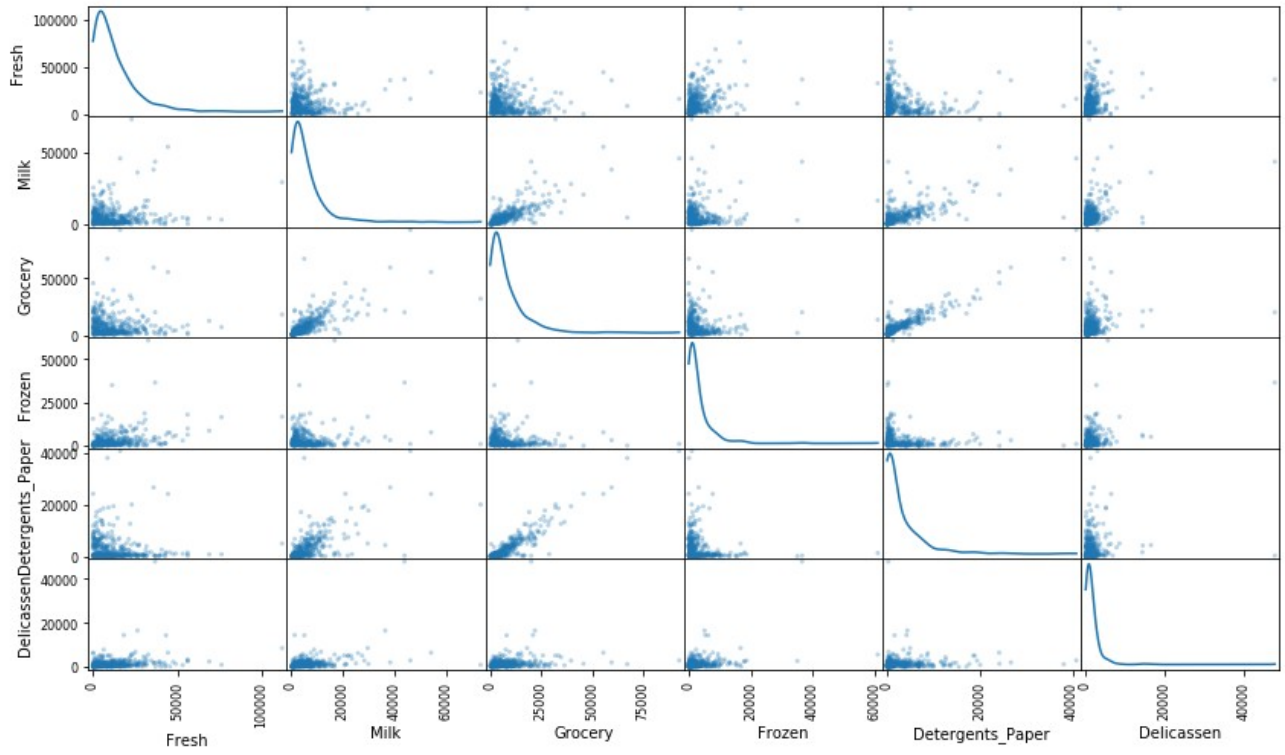
To evaluate how well a trained model performs on unseen data, we have to split the dataset into separate training and test datasets.

```
# Split the data into training and testing sets using the given feature as the target
X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
```

Scatter matrix

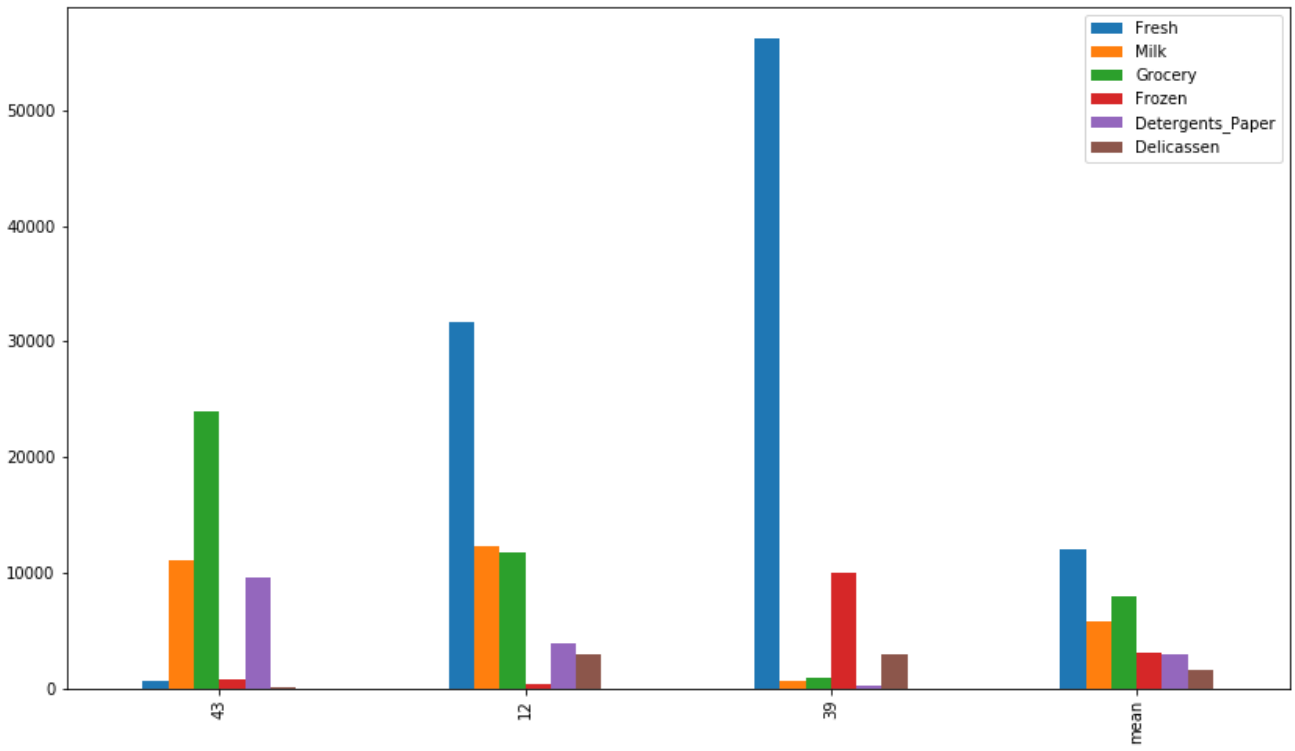
A scatter matrix is a pair-wise scatter plot of several variables presented in a matrix format. It can be used to determine whether the variables are correlated and whether the correlation is positive or negative.

```
70
71 pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde'); #scatter matrix
```



Bar Plot

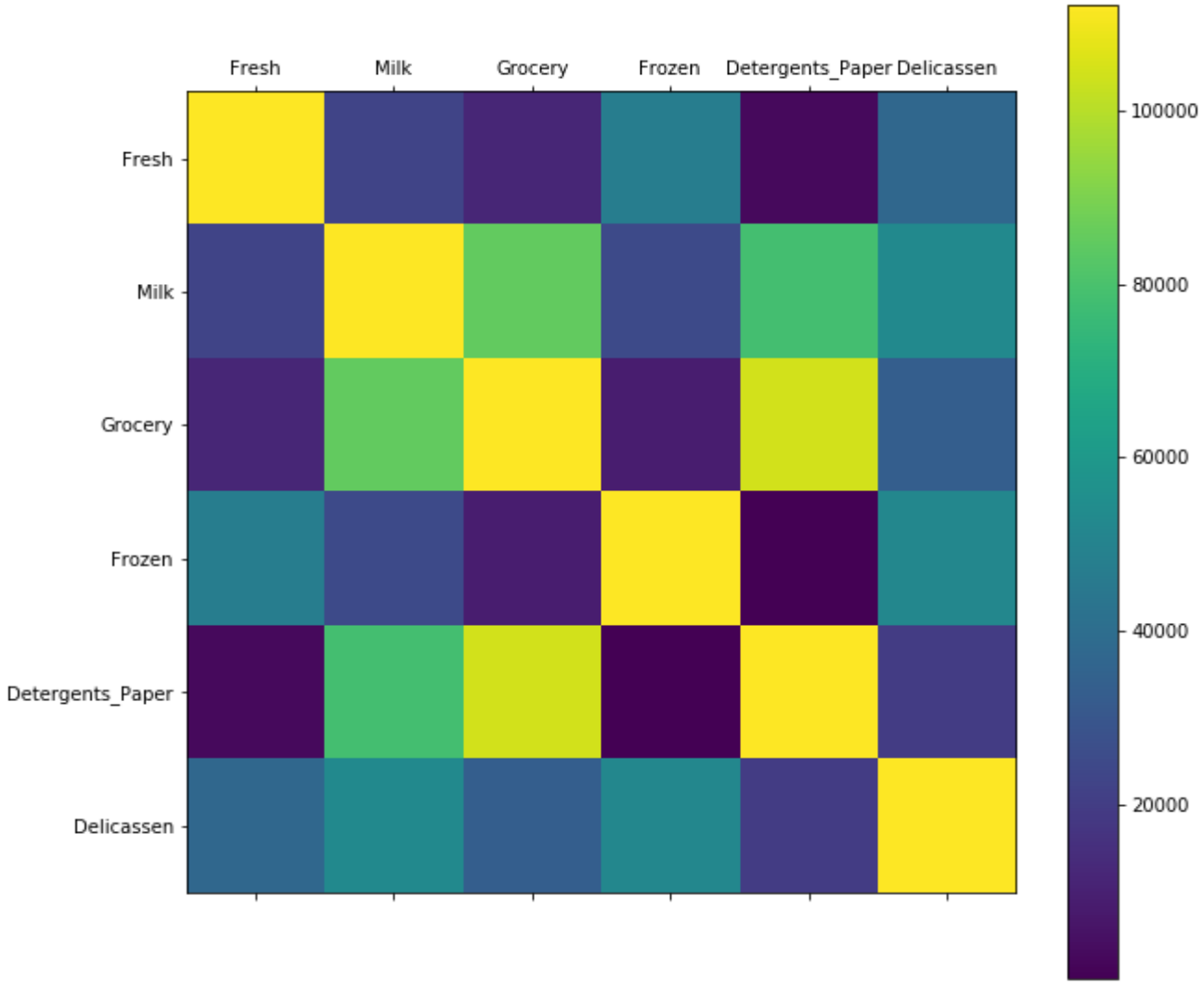
```
# Get the means
mean_data = data.describe().loc['mean', :]
# Append means to the samples' data
samples_bar = samples.append(mean_data)
# Construct indices
samples_bar.index = indices + ['mean']
# Plot bar plot
samples_bar.plot(kind='bar', figsize=(14,8))
# First, calculate the percentile ranks of the whole dataset.
percentiles = data.rank(pct=True)
# Then, round it up, and multiply by 100
percentiles = 100*percentiles.round(decimals=3)
# Select the indices you chose from the percentiles dataframe
percentiles = percentiles.iloc[indices]
data.columns

```



Correlation Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used as a way to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

```
73
74 def plot_corr(df,size=10):
75     corr = df.corr()
76     fig, ax = plt.subplots(figsize=(size, size))
77     cax = ax.matshow(df, interpolation='nearest')
78     ax.matshow(corr)
79     fig.colorbar(cax)
80     plt.xticks(range(len(corr.columns)), corr.columns);
81     plt.yticks(range(len(corr.columns)), corr.columns);
82
83 plot_corr(data)
```



Dimensionality reduction using pca

Principal component analysis (PCA) is an unsupervised linear transformation technique that is widely used across different fields, most prominently for dimensionality reduction.

```
from sklearn.decomposition import PCA

#Apply PCA by fitting the good data with the same number of dimensions as features
pca = PCA(n_components=6)
# Fit
pca.fit(good_data)

# Transform the sample log-data using the PCA fit above
pca_samples = pca.transform(log_samples)

# Generate PCA results
#Apply PCA by fitting the good data with only two dimensions

pca = PCA(n_components=2)
pca.fit(good_data)
#Transform the good data using the PCA fit above
reduced_data = pca.transform(good_data)
#Transform the sample log-data using the PCA fit above
pca_samples = pca.transform(log_samples)
# Create a DataFrame for the reduced data
reduced_data = pd.DataFrame(reduced_data, columns = ['Dimension 1', 'Dimension 2'])

```

Objective 2: Choosing a model

I have taken three models and using these three models we find out the accuracy.

Support vector machine (SVM):

Support Vector Machine (**SVM**) is a supervised binary classification algorithm. Given a set of points of two types in N dimensional place SVM generates a (N−1) dimensional hyperplane to separate those points into two groups.

```
for var in dep_vars:

    #Make a copy of the DataFrame, using the 'drop' function to drop the given feature
    new_data = data.drop([var], axis = 1)

    # Create feature Series (Vector)
    new_feature = pd.DataFrame(data.loc[:, var])

    # Split the data into training and testing sets using the given feature as the target
    X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
    dtr = SVC(kernel='linear', C=1,random_state = 42)
    dtr.fit(X_train, y_train)
    y_pred = dtr.predict(X_test)
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    print("confusion matrix of SVM\n",cm)

```

DecisionTreeRegressor:

```
# Importing the libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Create list to loop through
dep_vars = list(data.columns)


# Create loop to test each feature as a dependent variable
for var in dep_vars:
    #Make a copy of the DataFrame, using the 'drop' function to drop the given feature
    new_data = data.drop([var], axis = 1)
    new_feature = pd.DataFrame(data.loc[:, var]) # feature series
    # Split the data into training and testing sets using the given feature as the target
    X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
    # Creating a decision tree regressor and fitting it to the training set
    dtr = DecisionTreeRegressor(random_state=42)
    dtr.fit(X_train, y_train) # fitting the data
    y_pred = dtr.predict(X_test)
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    print("confusion matrix of decision regression\n",cm)

```

KNeighborsClassifier:

```
# Importing libraries

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Create range of clusters
range_n_clusters = list(range(2,11))
print(range_n_clusters)
#KNN Implementation
for n_clusters in range_n_clusters:
    # Apply your clustering algorithm of choice to the reduced data
    clusterer = KMeans(n_clusters=n_clusters).fit(reduced_data)

    # Predict the cluster for each data point
    preds = clusterer.predict(reduced_data)

    #Find the cluster centers
    centers = clusterer.cluster_centers_

    #Predict the cluster for each transformed sample data point
    sample_preds = clusterer.predict(pca_samples)

    #Calculate the mean silhouette coefficient for the number of clusters chosen
    score = silhouette_score(reduced_data, preds, metric='euclidean')

    print ("For n_clusters = {}. The average silhouette_score is : {}".format(n_clusters, score))

```

Objective 3: Training

In this objective given to us we have to reiterate the models to get an optimum or optimized model.

```
# Importing the libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Create list to loop through
dep_vars = list(data.columns)


# Create loop to test each feature as a dependent variable
for var in dep_vars:
    #Make a copy of the DataFrame, using the 'drop' function to drop the given feature
    new_data = data.drop([var], axis = 1)
    new_feature = pd.DataFrame(data.loc[:, var]) # feature series
    # Split the data into training and testing sets using the given feature as the target
    X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
    # Creating a decision tree regressor and fitting it to the training set
    dtr = DecisionTreeRegressor(random_state=42)
    dtr.fit(X_train, y_train) # fitting the data
    y_pred = dtr.predict(X_test)
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    print("confusion matrix of decision regression\n",cm)

```

```
#KNN Implementation
for n_clusters in range_n_clusters:
    # Apply your clustering algorithm of choice to the reduced data
    clusterer = KMeans(n_clusters=n_clusters).fit(reduced_data)

    # Predict the cluster for each data point
    preds = clusterer.predict(reduced_data)

    #Find the cluster centers
    centers = clusterer.cluster_centers_

    #Predict the cluster for each transformed sample data point
    sample_preds = clusterer.predict(pca_samples)

    #Calculate the mean silhouette coefficient for the number of clusters chosen
    score = silhouette_score(reduced_data, preds, metric='euclidean')

    print ("For n_clusters = {}. The average silhouette_score is : {}".format(n_clusters, score))

```

```
for var in dep_vars:

    #Make a copy of the DataFrame, using the 'drop' function to drop the given feature
    new_data = data.drop([var], axis = 1)

    # Create feature Series (Vector)
    new_feature = pd.DataFrame(data.loc[:, var])

    # Split the data into training and testing sets using the given feature as the target
    X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
    dtr = SVC(kernel='linear', C=1,random_state = 42)
    dtr.fit(X_train, y_train)
    y_pred = dtr.predict(X_test)
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    print("confusion matrix of SVM\n",cm)

```

Objective 4: Evaluation

Confusion matrix: a matrix that lays out the performance of a learning algorithm. The confusion matrix is simply a square matrix that reports the counts of the true positive, true negative, false positive, and false negative predictions of a classifier.
Silhouette Score – It is measure of how similar an object to its own cluster compared to other cluster.

Confusion Matrix For SVM:

```
confusion matrix of SVM
 [[0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of SVM
 [[0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of SVM
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of SVM
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Confusion Matrix For Decision Tree Regression:

```
confusion matrix of decision regression
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of decision regression
 [[0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of decision regression
 [[0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
confusion matrix of decision regression
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Silhouette Score for KneighborsClassifier:

```
For n_clusters = 2. The average silhouette_score is : 0.42628101546910835
For n_clusters = 3. The average silhouette_score is : 0.39418735404519234
For n_clusters = 4. The average silhouette_score is : 0.3329304196814193
For n_clusters = 5. The average silhouette_score is : 0.3496074606159664
For n_clusters = 6. The average silhouette_score is : 0.3650885924121462
For n_clusters = 7. The average silhouette_score is : 0.36490353248489016
For n_clusters = 8. The average silhouette_score is : 0.36405497488401767
For n_clusters = 9. The average silhouette_score is : 0.3572844712605567
For n_clusters = 10. The average silhouette_score is : 0.3542011421758249
```

Objective 5: Hyperparameter tuning

The approach of grid search is quite simple, it's a brute-force exhaustive search paradigm where we specify a list of values for different hyperparameters, and the computer evaluates the model performance for each combination of those to obtain the optimal set.

Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

```
212 from sklearn.svm import SVC
213 from sklearn.model_selection import GridSearchCV
214 from sklearn.pipeline import Pipeline
215 from sklearn.preprocessing import StandardScaler
216 X_train, X_test, y_train, y_test = train_test_split(new_data, new_feature, test_size=0.25, random_state=42)
217 pipe_svc = Pipeline([('scl',StandardScaler()),('clf',SVC(random_state=1))])
218 param_range =[0.0001,0.001,0.01,0.1,1.0,10.0,100.0,1000.0]
219 param_grid =[{'clf_C': param_range,'clf_kernel':['linear']},{'clf_C':param_range,'clf_gamma':param_range,'clf_kernel':['rbf']}]
220 gs=GridSearchCV(estimator=pipe_svc,param_grid=param_grid,scoring='accuracy',cv=3,n_jobs=-1)
221 gs=gs.fit(X_train,y_train)
222 print(gs.best_score_)
223 print(gs.best_params_)
224
```