**Dataset: SuperMarket**

**Description:** The growth of supermarkets in most populated cities are increasing and market competitions are also high. The dataset is one of the historical sales of a supermarket company which has been recorded in 3 different branches for 3 months. Predictive data analytics methods are easy to apply with this dataset.

**Attribute information:**

**Invoice id**: Computer generated sales slip invoice identification number
**Branch**: Branch of supercenter (3 branches are available identified by A, B and C).
**City**: Location of supercenters
**Customer type**: Type of customers, recorded by Members for customers using member card and Normal for without member card.
**Gender**: Gender type of costume
**Product line**: General item categorization groups - Electronic accessories, Fashion accessories, Food and beverages, Health and beauty, Home and lifestyle, Sports and travel
**Unit price**: Price of each product in $
**Quantity**: Number of products purchased by customer
**Tax**: 5% tax fee for customer buying
**Total**: Total price including tax
**Date**: Date of purchase (Record available from January 2019 to March 2019)
**Time**: Purchase time (10am to 9pm)
**Payment**: Payment used by customer for purchase (3 methods are available – Cash, Credit card and Ewallet)
**COGS**: Cost of goods sold
**Gross margin percentage**: Gross margin percentage
**Gross income**: Gross income
**Rating**: Customer stratification rating on their overall shopping experience (On a scale of 1 to 10)

---

**Preprocessing**:

## convert catogorical to numerical, before converting it'll check if is non numerical

```
[ ]
    def convertCatNum(dataset):
      le = preprocessing.LabelEncoder()
      notCol = df._get_numeric_data().columns
      for col in df.columns:
        if col not in notCol:
          dataset[col]=le.fit_transform(dataset[col])
      return dataset
```

## remove null values and replace that with median values

```
[ ] def remNull(dataset):

        imputer = SimpleImputer(missing_values=np.nan,strategy="median")
        imputer.fit(dataset.iloc[:,[4,5,6,10,13]])
        dataset.iloc[:,[4,5,6,10,13]]=imputer.transform(dataset.iloc[:,[4,5,6,10,13]])
        print("Checking null value:\n")
        print(dataset.isnull().sum())
        print("\n\n")
        return dataset
```

## normalise and split the dataset into train and test

## parameters: dataset, features in array

## return: x_train, x_test, y_train, y_test

**remember select the feature**

```
def splitter(dataset, colsx, colsy):
  # X=dataset.iloc[:,2:].values
  # y=dataset.iloc[:,1].values
  X = dataset[[*colsx]].values
  y = dataset[[*colsy]].values
  SD=StandardScaler()
  X=SD.fit_transform(X)
  #y=np.column_stack(SD.fit_transform(y))
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=0)
  return X_train, X_test, y_train, y_test
```

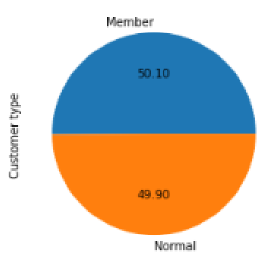
```
[ ] # df = convertCatNum(df.copy(), 'Gender')
    # df = convertCatNum(df.copy(), 'Branch')
    # df = convertCatNum(df.copy(), 'City')
    # df = convertCatNum(df.copy(), 'Customer type')
```

**Visualizing:**

15. PIE CHART

Using pie chart we are able to visualise the percentage of Customer who are Member and Normal customers

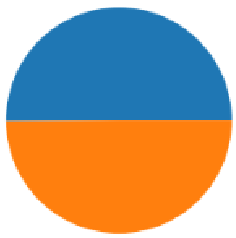
```
ds['Customer type'].value_counts().plot(kind="pie", autopct="%.2f")
plt.show()
```



Using pie chart we are able to visualise the percentage of Gender who are Male and Female customers

```
#ds['Gender'].value_counts().plot(kind="pie", autopct="%.2f")
#plt.show()
plt.pie(ds['Gender'].value_counts())
```

```
([<matplotlib.patches.Wedge at 0x7f93ddf18550>,
  <matplotlib.patches.Wedge at 0x7f93ddf62650>],
 [Text(-0.003455701743252077, 1.099994571861817, ''),
  Text(0.003455701743251942, -1.099994571861817, '')])
```

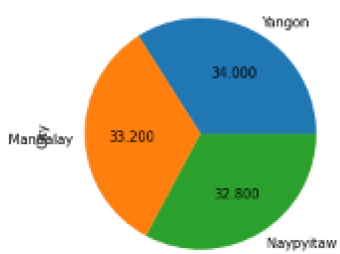


## Using pie chart we are able to visualise the percentage of Product line

```
ds['Product line'].value_counts().plot(kind="pie", autopct="%.3f")
plt.show()
```


```
[ ]
```

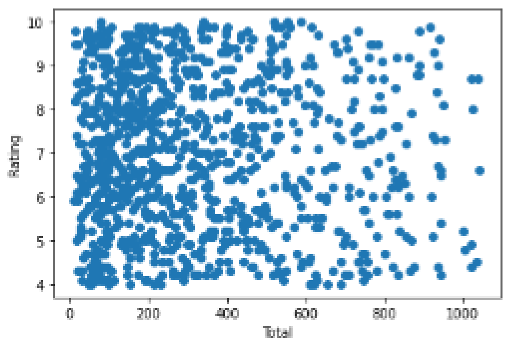
```
[ ] ds['City'].value_counts().plot(kind="pie", autopct="%.3f")
plt.show()
```



## 14. Scatter plot

```
[ ]  # to check whether there is a correlation between Total cost and Rating
```

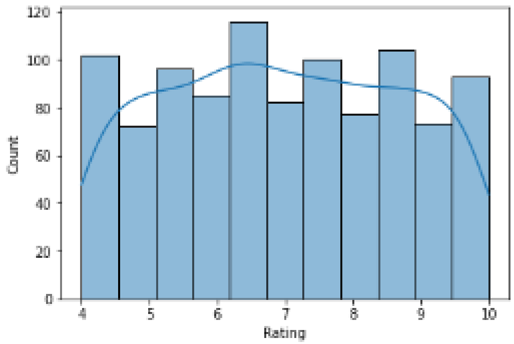
```
#plt.figure(figsize=(10,10))
plt.scatter(x='Total',y='Rating',data=ds)
plt.xlabel('Total')
plt.ylabel('Rating')
plt.show()
```



## 7. Histplot

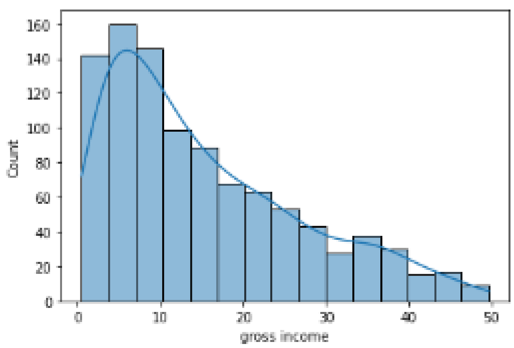
```
[ ] sns.histplot(data = ds['Rating'], kde = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93e36c2d50>
```



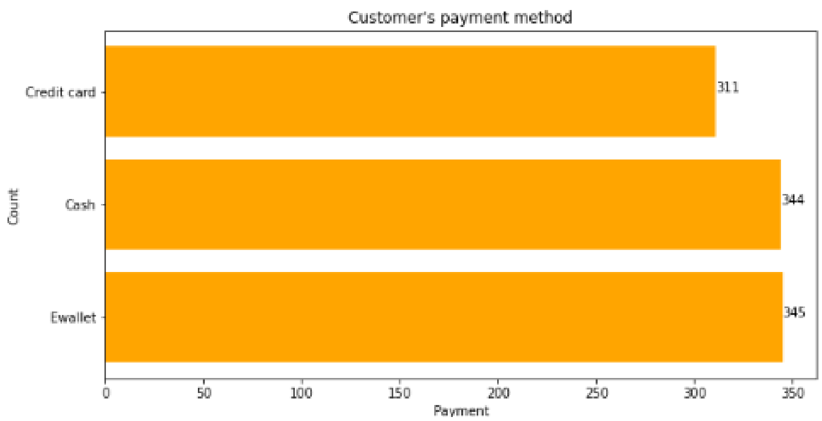

```
[ ] sns.histplot(data = ds['gross income'], kde = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93e3da0ad0>
```



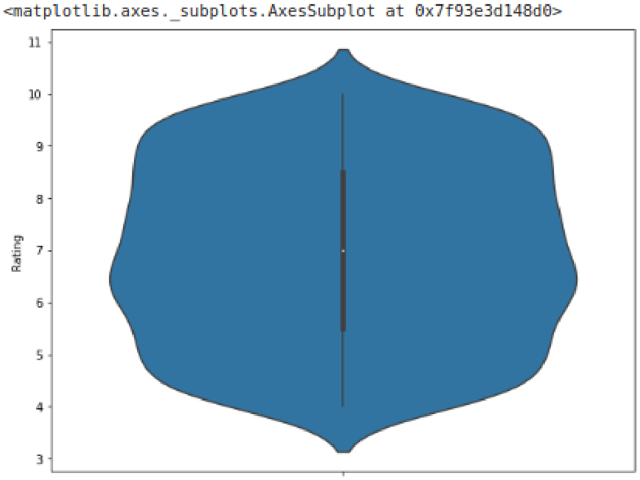
## To visualize the number of customers who use credit card, cash, ewallet

```
Payment = list(ds['Payment'].value_counts().keys())
values = list(ds['Payment'].value_counts())

fig = plt.figure(figsize = (10, 5))

plt.barh(Payment, values, color ='orange')

for index, value in enumerate(values):
    plt.text(value, index,str(value))

plt.xlabel("Payment")
plt.ylabel("Count")
plt.title("Customer's payment method")
plt.show()
```



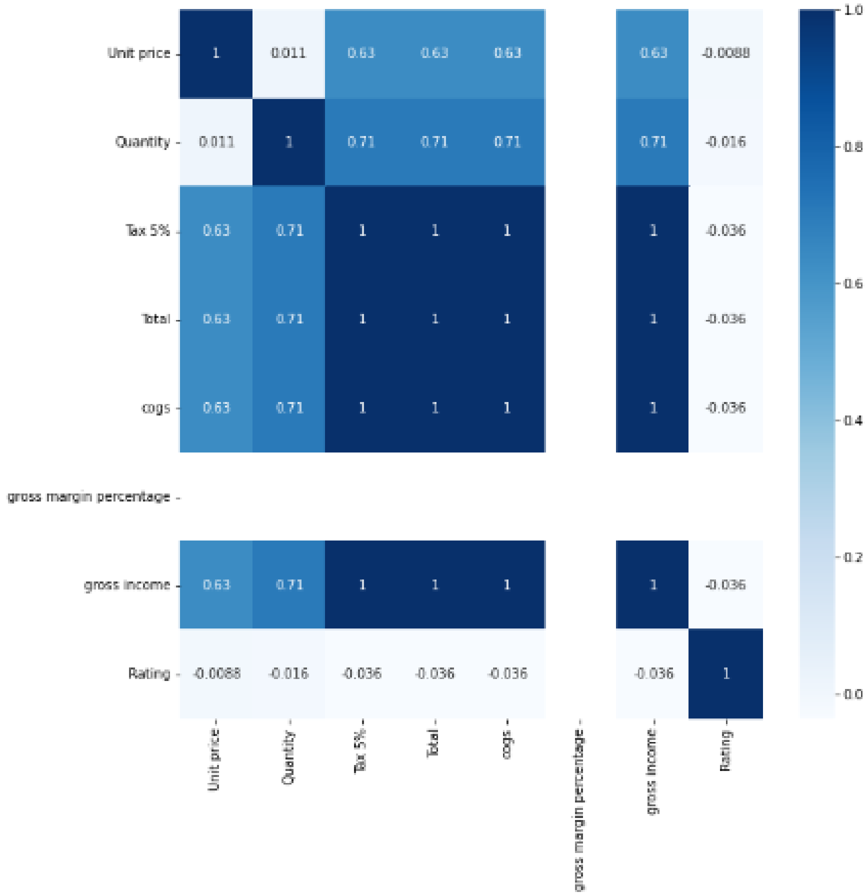

```
fig, ax = plt.subplots(figsize =(9, 7))
sns.violinplot( ax = ax, y = ds["Rating"] )
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f93e3d148d0>



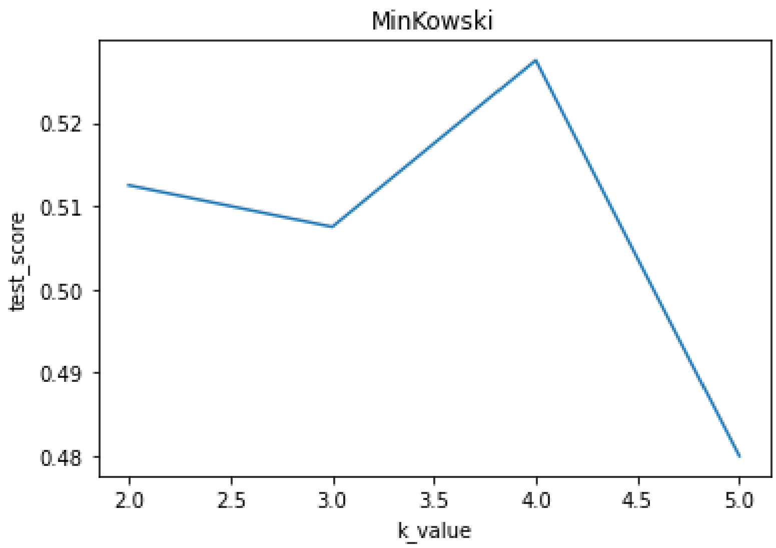

```
# from the above visualisation we can conclude that the more number of cutomer rate around 6 to 7
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f93ddce3c50>



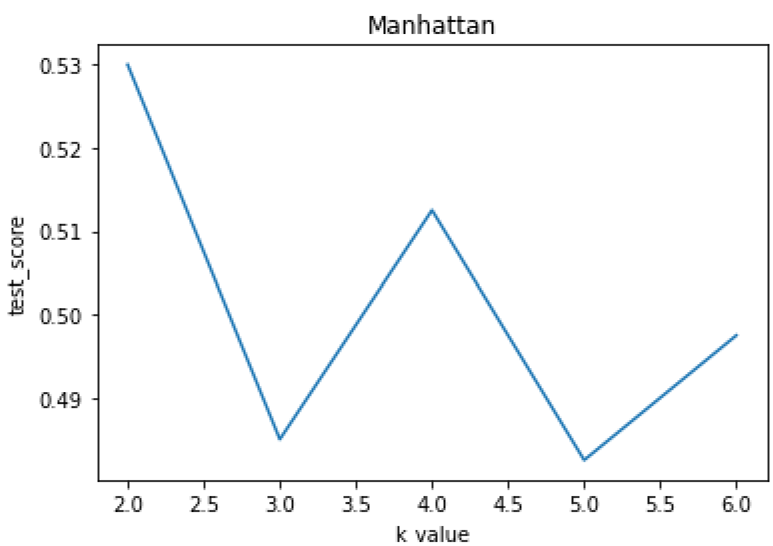

```
# from the above visualisation we can say that there is a correlation between Tax and total, cogs, gross income
```

Algorithm used: **K Nearest Neighbor**
Distance Formula used: **MinKowski**



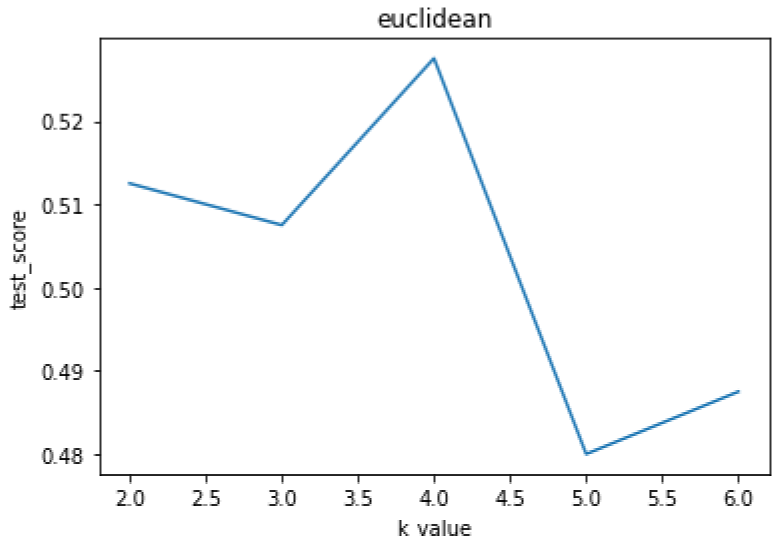

*Inference: Accuracy was increasing with increase in K value till K=4. Accuracy was highest when K=4.*

Distance Formula used: **Manhattan**




*Inference: Accuracy was decreasing with increase in K value. Accuracy was highest when K=2.*

Distance Formula used: **Euclidean**




*Inference: Accuracy was decreasing with increase in K value. Accuracy was highest when K=4.*

```
confusionMatrix(6, 'minkowski')
```

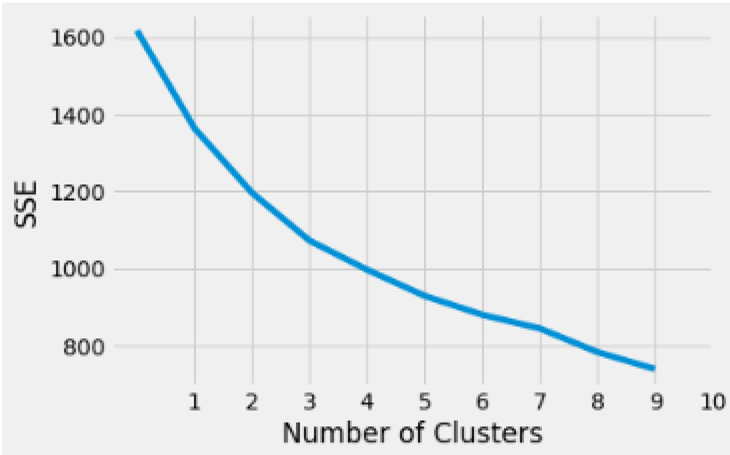
```
              precision    recall  f1-score   support

         0.0       0.53      0.55      0.54       217
         1.0       0.44      0.41      0.42       183

    accuracy                           0.49       400
   macro avg       0.48      0.48      0.48       400
weighted avg       0.49      0.49      0.49       400

[[120  97]
 [108  75]]

________________________________________________________________________

Accuracy score:

0.4875
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.
  return self._fit(X, y)
```

```
[60] confusionMatrix(5, 'minkowski')
```

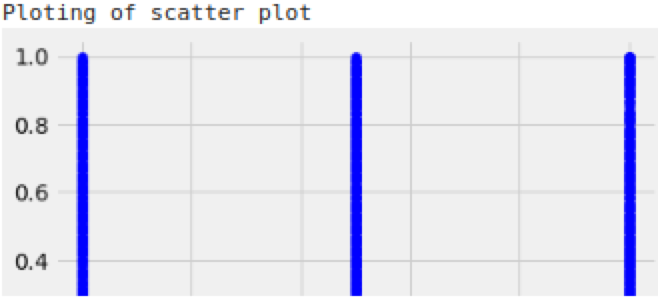
```
              precision    recall  f1-score   support

         0.0       0.53      0.41      0.46       217
         1.0       0.45      0.57      0.50       183

    accuracy                           0.48       400
   macro avg       0.49      0.49      0.48       400
weighted avg       0.49      0.48      0.48       400

[[ 88 129]
 [ 79 104]]

____________________________________________________________________

Accuracy score:

0.48
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classificat
  return self._fit(X, y)
```

```
The Score of euclidean with k value of 2 is:

_________________________________________________________

              precision    recall  f1-score   support

         0.0       0.54      0.68      0.60       217
         1.0       0.45      0.32      0.37       183

    accuracy                           0.51       400
   macro avg       0.50      0.50      0.49       400
weighted avg       0.50      0.51      0.50       400

[[147  70]
 [125  58]]

_________________________________________________________

Accuracy score:

0.5125
```

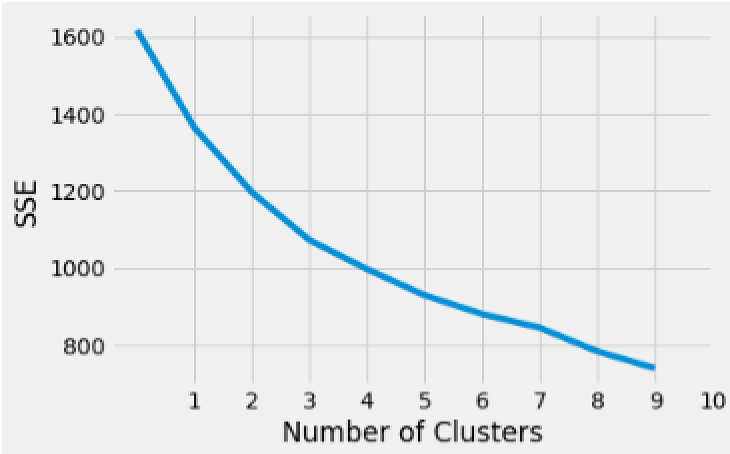
```
Ploting of the kMeans
```



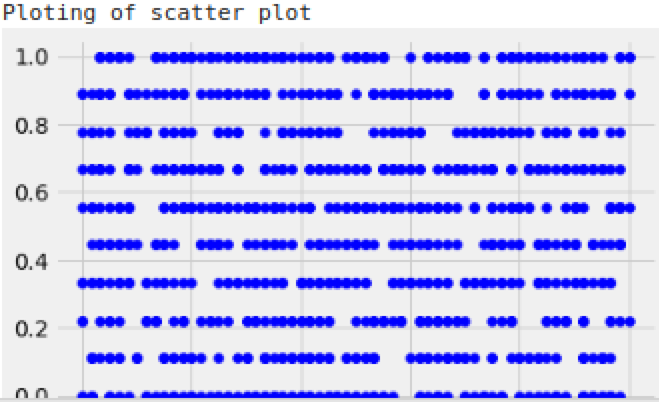

```
--------------------------------------------------------

Ploting of scatter plot
```



```
Ploting of the kMeans
```




```
------------------------------------------------------------

Ploting of scatter plot
```



# Algorithm used: Naive byes

```python
def naiv():

  # instantiate the model
  gnb = GaussianNB()


  numRows = trans_formed_min_max.shape[0]
  trainSize = round(numRows * 0.6)
  # divide train and test dataset
  train = trans_formed_min_max.iloc[:trainSize, : ]
  test = trans_formed_min_max.iloc[trainSize : , :]

  x_cols = ["City", "Branch","Gender" , "Product line", "Unit price", "Quantity", "Tax 5%", "Total", "Payment", 'cogs', 'gross income', 'Rating']
  y_col = ["Customer type"]
    # fit the model
  gnb.fit(train[x_cols], train[y_col])
  y_pred = gnb.predict(test[x_cols])

  print(y_pred)

  print('\n\nModel accuracy score: {0:0.4f}\n\n'. format(accuracy_score(test[y_col], y_pred)))
```

```
[70] naiv()
```

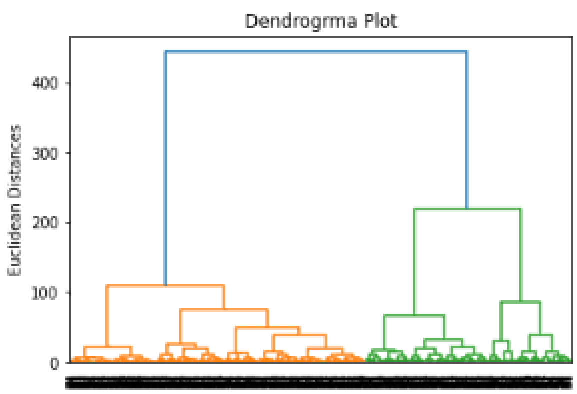
```
[1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0.
 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0.
 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1.
 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1.
 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.
 0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0.]


Model accuracy score: 0.4825


/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
  y = column_or_1d(y, warn=True)
```

# Hierarchical Clustering

```
[ ] x = df.iloc[:, [6, 11]].values
    x
```

```
array([[ 7.    , 26.1415],
       [ 5.    ,  3.82  ],
       [ 7.    , 16.2155],
       ...,
       [ 1.    ,  1.592 ],
       [ 1.    ,  3.291 ],
       [ 7.    , 30.919 ]])
```

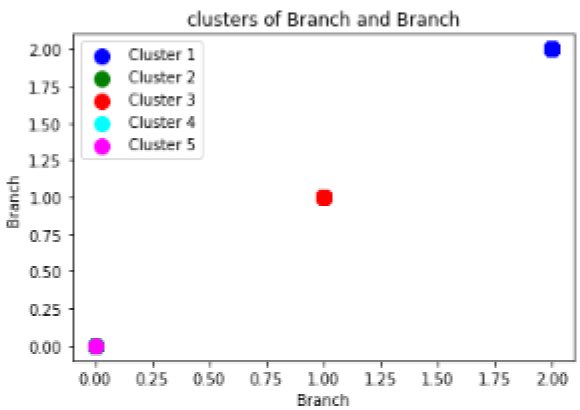
```
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")

mtp.show()
```

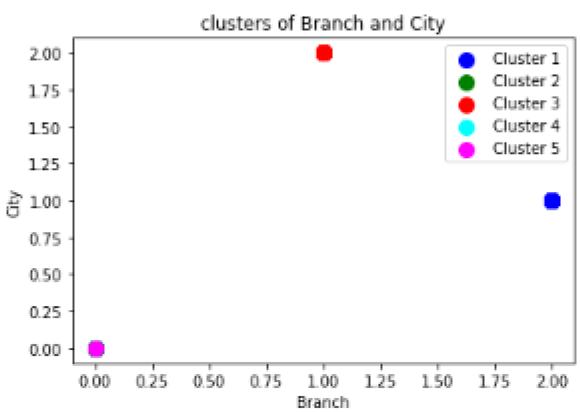


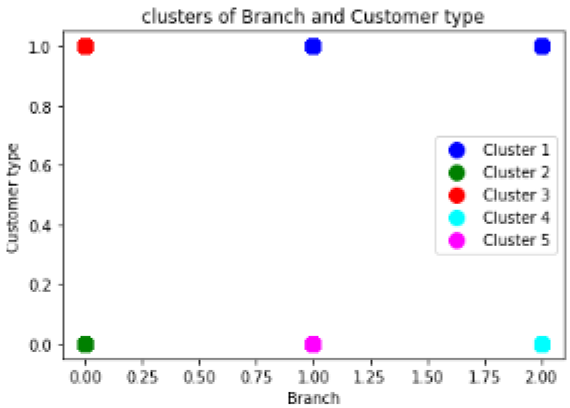
## training the hierarchical model on dataset

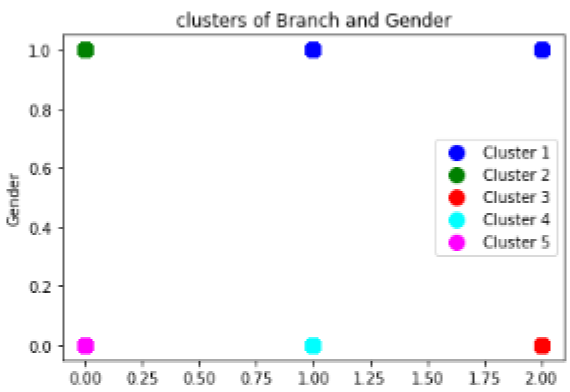
```
[ ] from sklearn.cluster import AgglomerativeClustering
```

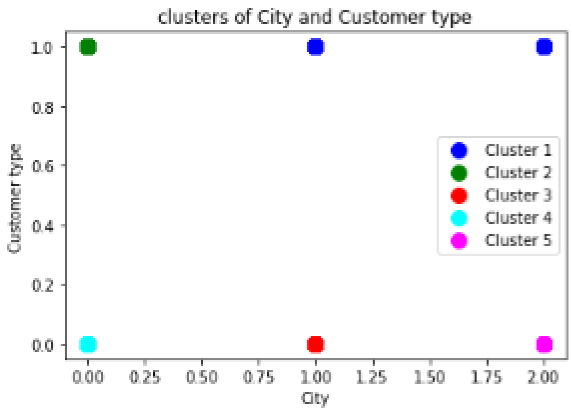
```
#visulaizing the clusters
def clusters(col1, col2):
  x = df.iloc[:, [col1, col2]].values
  hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
  y_pred= hc.fit_predict(x)
  #y_pred
  mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
  mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
  mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
  mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
  mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
  mtp.title('clusters of '+df.columns[col1]+' and '+df.columns[col2])
  mtp.xlabel(df.columns[col1])
  mtp.ylabel(df.columns[col2])
  mtp.legend()
  mtp.show()
```

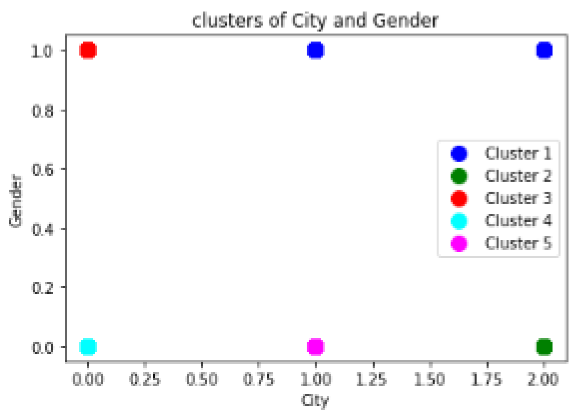



clusters of Branch and Customer type
Customer type
Branch
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


clusters of Branch and Gender
Gender
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



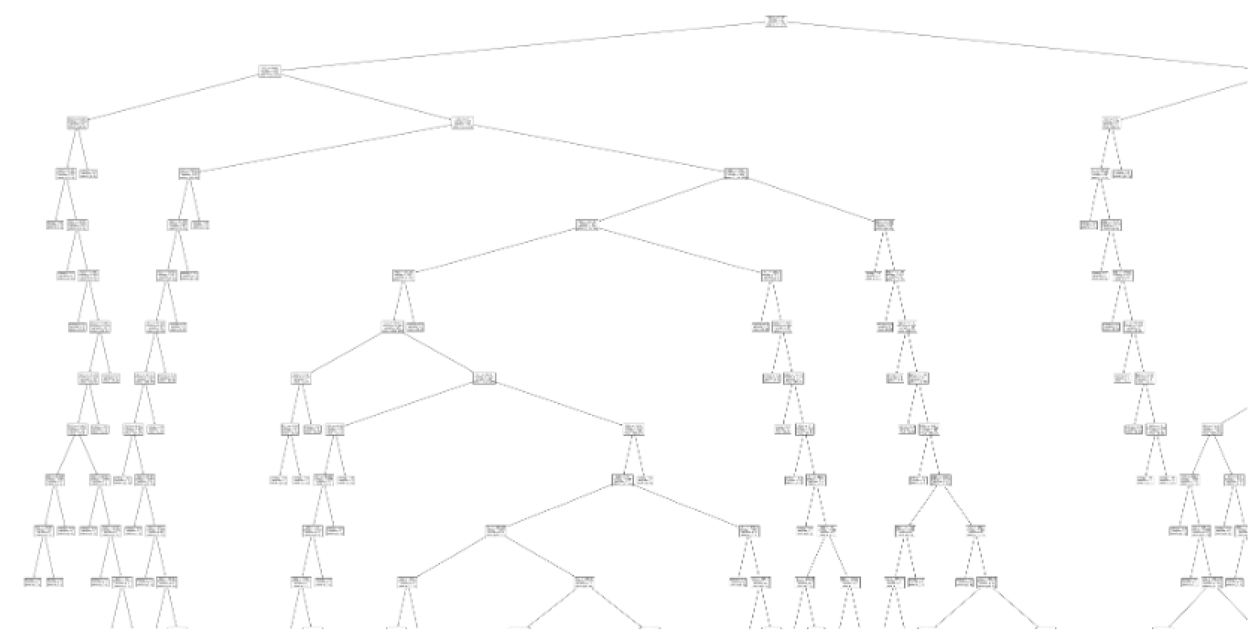




## Decision Tree

```python
def decision(ds, y_val):

  y = ds[y_val].values
  ds.drop(y_val, axis=1, inplace=True)
  #print(ds.head())
  x = ds.iloc[:].values


  # Splitting the dataset into training and test set.

  x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
  classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
  classifier.fit(x_train, y_train)
  # pre-pruning
  param_grid = {
    "max_depth": [3,5,10,15,20,None],
    "min_samples_split": [2,5,7,10],
    "min_samples_leaf": [1,2,5]
  }
  grid_cv = GridSearchCV(classifier, param_grid, scoring="roc_auc", n_jobs=-1, cv=3).fit(x_train, y_train)
  y_pred = classifier.predict(x_test)
  print("\nAccuracy score:\t"+str(accuracy_score(y_test,y_pred)*100))
  print('Model accuracy score with criterion entropy index: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
  print("\n\n")
  plt.figure(figsize=(12,8))

  f = plt.figure()
  f.set_figwidth(100)
  f.set_figheight(100)
  tree.plot_tree(classifier.fit(x_train, y_train))
```

```python
decision(preProcess(df.copy()), 'Gender')
```

```
<Figure size 864x576 with 0 Axes>
```

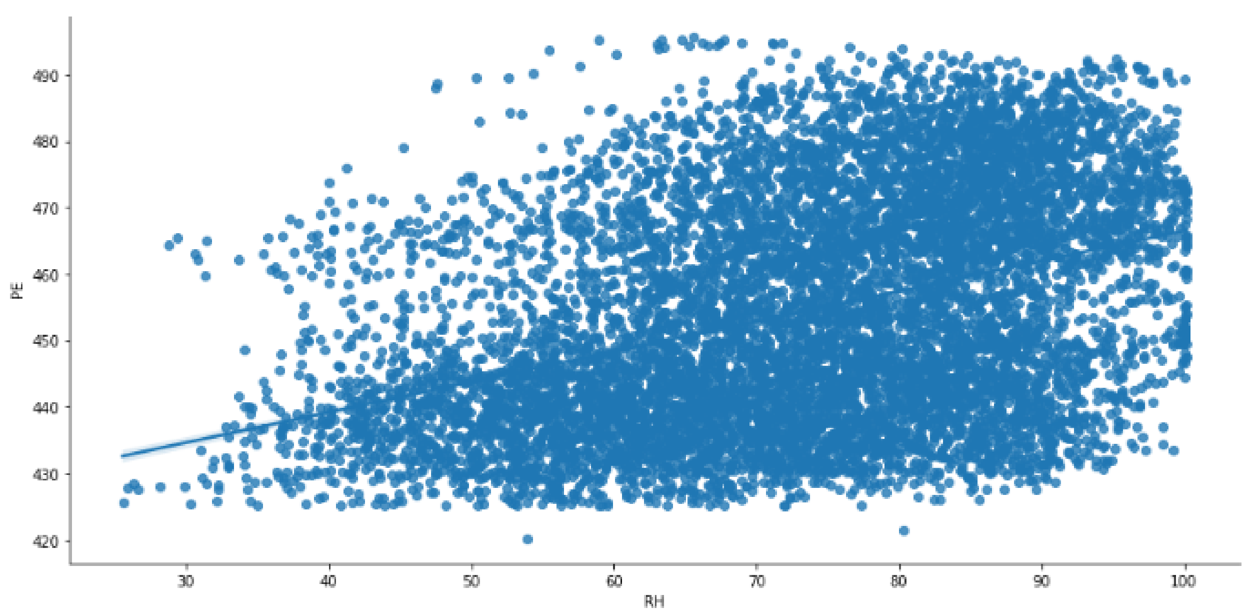


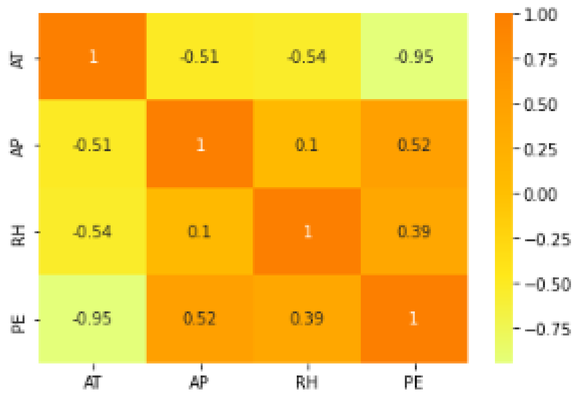
**Linear Regression:**

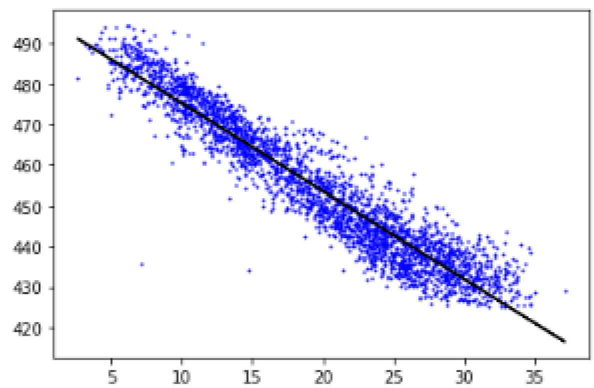
```
#model.fit(x, y)
lin(df.copy(), 'V')
```

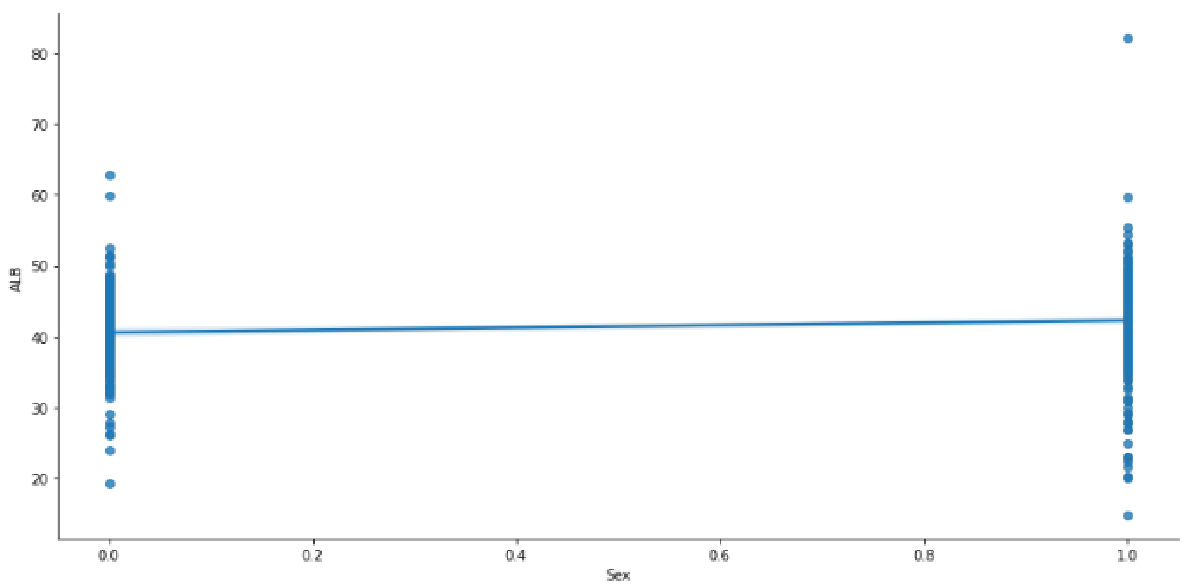
```
[69.42840455 53.12830849 51.96908514 ... 42.06936609 41.33498564
 63.49589119]

 Mean Squared error:    37.33487054761878
```

```
[47.32344468 68.90498626 52.49994471 ... 51.44804726 48.93305719
 69.3166825 ]
[ 0.729788    0.17532246  0.11125979 -0.41543592]
42.97785919016527
37.33487054761878
0.704092463762724
0.7051895503285368
```



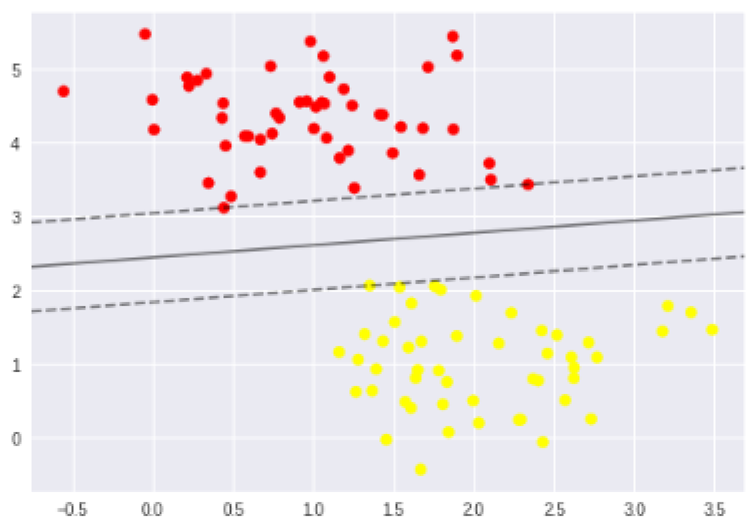

AT
AP
RH
PE
1
-0.51
-0.54
-0.95
0.1
0.52
0.39
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

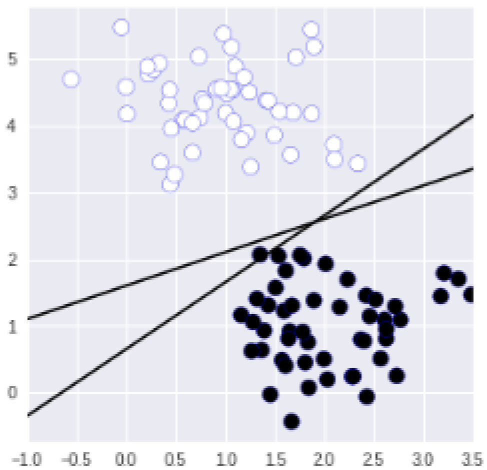

490
480
470
460
450
440
430
420
5
10
15
20
25
30
35


ALB
Sex
80
70
60
50
40
30
20
0.0
0.2
0.4
0.6
0.8
1.0

# SVM:

```
def plot_svc_decision_function(model, ax= None, plot_support=True):


  if ax is None:
    ax = plt.gca()
  xlim = ax.get_xlim()
  ylim = ax.get_ylim()

  x = np.linspace(xlim[0], xlim[1], 30)
  y = np.linspace(ylim[0], ylim[1], 30)
  Y, X = np.meshgrid(y, x)
  xy = np.vstack([X.ravel(), Y.ravel()]).T
  P = model.decision_function(xy).reshape(X.shape)

  ax.contour(X, Y, P, colors="k", levels = [-1, 0, 1], alpha = .5, linestyles=['--','-', '--'])

  if plot_support:
    ax.scatter(model.support_vectors_[:,0],
    model.support_vectors_[:,1],
    s=300, linewidth=1, facecolors='none');

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(x[:,0], x[:,1], c=y, s=50, cmap='autumn')
```

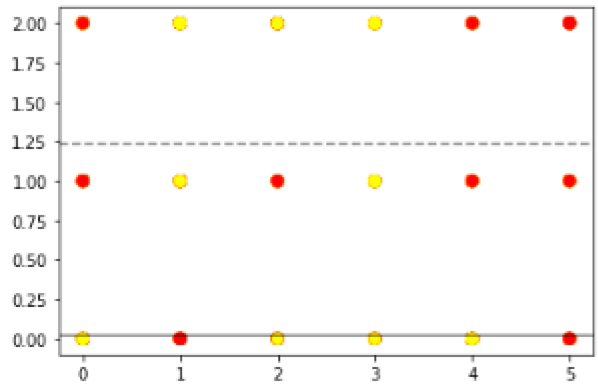





```
def plot_svc_decision_function_(model, ax= None, plot_support=True):


  if ax is None:
    ax = plt.gca()
  xlim = ax.get_xlim()
  ylim = ax.get_ylim()

  x = np.linspace(xlim[0], xlim[1], 30)
  y = np.linspace(ylim[0], ylim[1], 30)
  Y, X = np.meshgrid(y, x)
  xy = np.vstack([X.ravel(), Y.ravel()]).T
  P = model.decision_function(xy).reshape(X.shape)

  ax.contour(X, Y, P, colors="k", levels = [-1, 0, 1], alpha = .5, linestyles=['--','-', '--'])

  if plot_support:
    ax.scatter(model.support_vectors_[:,0],
    model.support_vectors_[:,1],
    s=300, linewidth=1, facecolors='none');

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

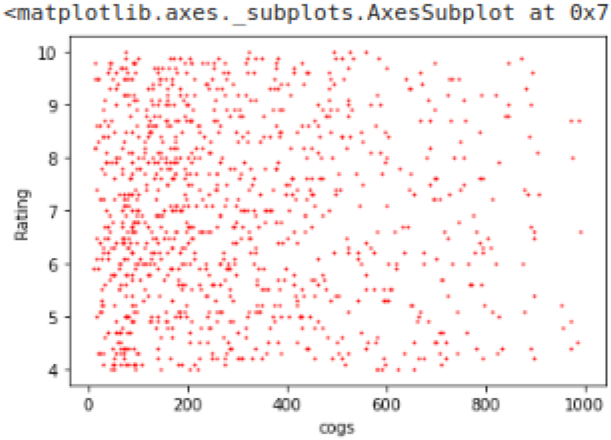
```
x = df[['Product line', 'Branch']].values
y = df[['Gender']].values
plt.scatter(x[:,0], x[:,1], c=y, s=50, cmap='autumn
plot_svc_decision_function(clf)
print(x)
```

```
[[3 0]
 [0 2]
 [4 0]
 ...
 [2 0]
 [4 0]
 [1 0]]
```

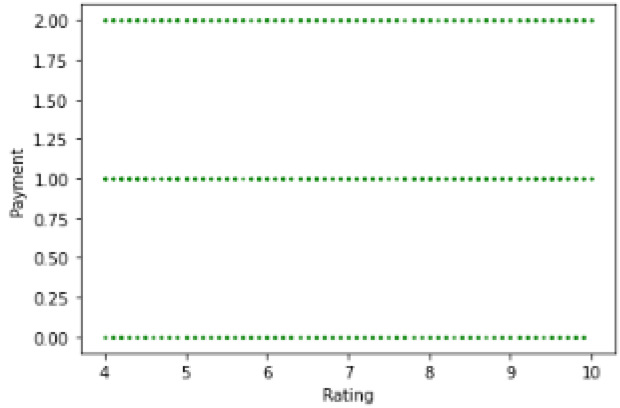



# MLP

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f65641c2c90>
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f656191c790>
```




```
activationList = ["relu", "identity", "logistic", "tanh"]
for i in range(0,4):
  clf = MLPClassifier(activation = activationList[i]);
  clf.fit(x_train, y_train);
  tempscore = clf.score(x_train, y_train)
  print("Activation function -",activationList[i],"- Accuracy : ",tempscore)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptror
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptror
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptror
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptror
  y = column_or_1d(y, warn=True)
Activation function - relu - Accuracy :  0.6125
Activation function - identity - Accuracy :  0.55125
Activation function - logistic - Accuracy :  0.54625
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptror
  y = column_or_1d(y, warn=True)
Activation function - tanh - Accuracy :  0.54875
```

**Inference:** *We can see that the Activation function: relu has more Accuracy Score compared to others*

```
y_pred = clf.predict(testX_scaled)
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, y_pred)
```

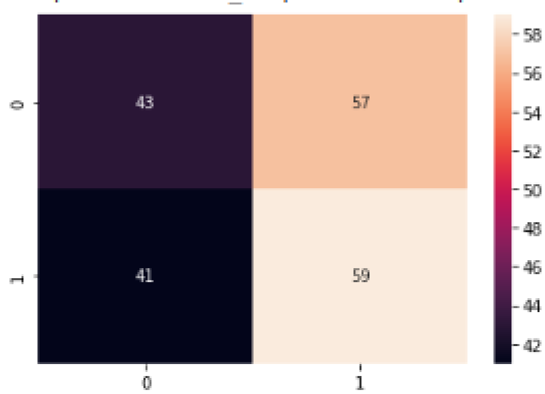
```
Accuracy: 0.51
```

```
#Get the confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
```

```
[[43 57]
 [41 59]]
```

```
sns.heatmap(cf_matrix, annot=True)
```

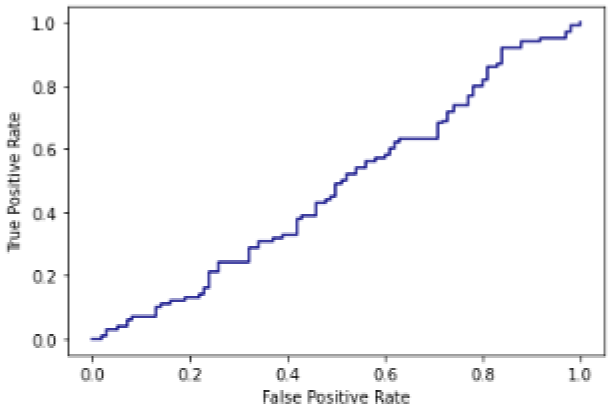
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f656188dcd0>
```



## ploting the TP and FP of x_train and y_train

```
mlp = MLPClassifier()
mlp.fit(x_train,y_train)
y_pred_proba = mlp.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba, pos_label=0)
plt.plot(fpr,tpr ,color="navy")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

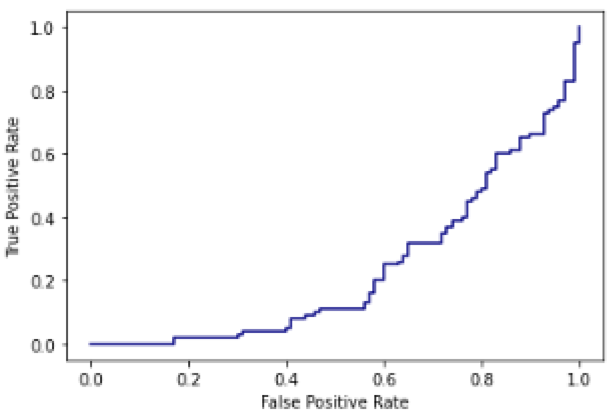
```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multi
  ConvergenceWarning,
```



## ploting the TP and FP of x_test and y_test

```
[ ]
    mlp = MLPClassifier()
    mlp.fit(x_test,y_test)
    y_pred_proba = mlp.predict_proba(x_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba, pos_label=0)
    plt.plot(fpr,tpr ,color="navy")
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multi
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multi
  ConvergenceWarning,
```



```
y_pred = mlp.predict(testX_scaled)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.70      0.65      0.67       100
         1.0       0.67      0.72      0.70       100

    accuracy                           0.69       200
   macro avg       0.69      0.69      0.68       200
weighted avg       0.69      0.69      0.68       200
```

```
[ ] activationList = ["relu", "identity", "logistic", "tanh"]
    for i in range(0,4):
      clf = MLPClassifier(activation = activationList[i], hidden_layer_s
      clf.fit(x_train, y_train);
      tempscore = clf.score(x_train, y_train)
      print("Activation function -",activationList[i],"- Accuracy : ",te
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multil
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multil
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multil
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multil
  y = column_or_1d(y, warn=True)
Activation function - relu - Accuracy :  0.5125
Activation function - identity - Accuracy :  0.53875
Activation function - logistic - Accuracy :  0.51
Activation function - tanh - Accuracy :  0.525
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multil
  y = column_or_1d(y, warn=True)
```

```
activationList = ["relu", "identity", "logistic", "tanh"]
for i in range(0,4):
  clf = MLPClassifier(activation = activationList[i], hidden_layer_sizes
  clf.fit(x_train, y_train);
  tempscore = clf.score(x_train, y_train)
  print("Activation function -",activationList[i],"- Accuracy : ",tempsc
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilaye
  y = column_or_1d(y, warn=True)
Activation function - relu - Accuracy :  0.52875
Activation function - identity - Accuracy :  0.54625
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilaye
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilaye
  y = column_or_1d(y, warn=True)
Activation function - logistic - Accuracy :  0.52
Activation function - tanh - Accuracy :  0.53875
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilaye
  y = column_or_1d(y, warn=True)
```