MMA/GMMA/MMAI 869: Individual Assignment

Version 1: Updated September 27, 2021

\# TODO: fill in the below

- Mody Wang
- 20270310
- Section 1
- Hands-on Machine Learning with Scikit-Learn,Keras & Tensorflow
- 2022-01-06

Assignment Instructions

This assignment contains four questions. The questions are fully contained in this Google Colab Notebook.

You are to make a copy of this Notebook and edit the copy to provide your answers. You are to complete the assignment entirely within Google Colab. Why?

- It gives you practice using cloud-based interactive notebook environments (which is a popular workflow)
- It is easier for you to manage the environment (e.g., installing packages, etc.)
- Google Colab has nice, beefy machines, so you don't have to worry about running out of memory on your local computer.
- It will be easier for the TA to help you debug your code if you need help
- It will be easier for the TA to mark/run your code

Some parts of this assigment require you to write code. Use Python or R. For Python, you may use standard Python libraries, including `scikit-learn`, `pandas`, `numpy`, and `scipy`. For R, you may use `dplyr`, `caret`, `ggplot2`, `rpart` and other standard libraries.

Some parts of this assignment require text responses. In these cases, type your response in the Notebook cell indicated. Use English. Use proper grammar, spelling, and punctuation. Be professional and clear. Be complete, but not overly-verbose. Feel free to use Markdown syntax to format your answer (i.e., add bold, italics, lists, tables).

What to Submit to the Course Portal

- Export your completed Notebook as a PDF file by clicking File->Print->Save as PDF.
- Please do not submit the Notebook file (`.ipynb`) to the course portal.
- Please submit the PDF export of the Notebook.
  - Please name the PDF file `2022_869_FirstnameLastName.pdf`
    - E.g., `2022_869_StephenThomas.pdf`
  - Please make sure you have run all the cells so we can see the output!
  - Best practice: Before exporting to PDF click Runtime->Restart and run all.

Preliminaries: Inspect and Set up environment

No action is required on your part in this section. These cells print out helpful information about the environment, just in case.

In [1]:
```
import datetime
import pandas as pd
import numpy as np
```

In [2]:
```
print(datetime.datetime.now())
```

```
2022-01-01 19:11:51.244086
```

In [3]:
```
!which python
```

```
'which' is not recognized as an internal or external command,
operable program or batch file.
```

In [4]:
```
!python --version
```

```
Python 3.8.5
```

In [5]:
```
!echo $PYTHONPATH
```

```
$PYTHONPATH
```

In [6]:
```
pip install pycaret
```

In [7]:
```
#Loading all the libraries used in the assignment.
from pycaret.clustering import *
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm,skew
import pylab as py
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, cohen_kappa_score, f1_score, log_loss
from sklearn.metrics import confusion_matrix
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import MinMaxScaler
import scipy.cluster
from scipy.spatial import distance
from sklearn.feature_selection import RFECV, RFE
```

Question 1: Uncle Steve's Diamonds

Instructions

You work at a local jewelry store named *Uncle Steve's Diamonds*. You started as a janitor, but you've recently been promoted to senior data analyst! Congratulations.

Uncle Steve, the store's owner, needs to better understand the store's customers. In particular, he wants to know what kind of customers shop at the store. He wants to know the main types of *customer personas*. Once he knows these, he will contemplate ways to better market to each persona, better satisfy each persona, better cater to each persona, increase the loyalty of each persona, etc. But first, he must know the personas.

You want to help Uncle Steve. Using sneaky magic (and the help of Environics), you've collected four useful features for a subset of the customers: age, income, spending score (i.e., a score based on how much they've spent at the store in total), and savings (i.e., how much money they have in their personal bank account).

Your tasks

1. Pick a clustering algorithm (the `sklearn.cluster` module has many good choices, including `KMeans`, `DBSCAN`, and `AgglomerativeClustering` (aka Hierarchical)). (Note that another popular implementation of the hierarchical algorithm can be found in SciPy's `scipy.cluster.hierarchy.linkage`.) Don't spend a lot of time thinking about which algorithm to choose - just pick one. Cluster the customers as best as you can, within reason. That is, try different feature preprocessing steps, hyperparameter values, and/or distance metrics. You don't need to try every posssible combination, but try a few at least. Measure how good each model configuration is by calculating an internal validation metric (e.g., `calinski_harabasz_score` or `silhouette_score`).
2. You have some doubts - you're not sure if the algorithm you chose in part 1 is the best algorithm for this dataset/problem. Neither is Uncle Steve. So, choose a different algorithm (any!) and do it all again.
3. Which clustering algorithm is "better" in this case? Think about charateristics of the algorithm like quality of results, ease of use, speed, interpretability, etc. Choose a "winner" and justify to Uncle Steve.
4. Interpret the clusters of the winning model. That is, describe, in words, a *persona* that accurately depicts each cluster. Use statistics (e.g., cluster means/distributions), examples (e.g., exemplar instances from each cluster), and/or visualizations (e.g., relative importance plots, snakeplots) to get started. Human judgement and creativity will be necessary. This is where it all comes together. Be descripive and *help Uncle Steve understand his customers better*. Please!

Marking

The coding parts (i.e., 1 and 2) will be marked based on:

- *Correctness*. Code clearly and fully performs the task specified.
- *Reproducibility*. Code is fully reproducible. I.e., you (and I) are able to run this Notebook again and again, from top to bottom, and get the same results each time.
- *Style*. Code is organized. All parts commented with clear reasoning and rationale. No old code laying around. Code easy to follow.

Parts 3 and 4 will be marked on:

- *Quality*. Response is well-justified and convincing. Responses uses facts and data where possible.
- *Style*. Response uses proper grammar, spelling, and punctuation. Response is clear and professional. Response is complete, but not overly-verbose. Response follows length guidelines.

Tips

- Since clustering is an unsupervised ML technique, you don't need to split the data into training/validation/test or anything like that. Phew!
- On the flip side, since clustering is unsupervised, you will never know the "true" clusters, and so you will never know if a given algorithm is "correct." There really is no notion of "correctness" - only "usefullness."
- Many online clustering tutorials (including some from Uncle Steve) create flashy visualizations of the clusters by plotting the instances on a 2-D graph and coloring each point by the cluster ID. This is really nice and all, but it can only work if your dataset only has exactly two features - no more, no less. This dataset has more than two features, so you cannot use this technique. (But that's OK - you don't need to use this technique.)
- Must you use all four features in the clustering? Not necessarily, no. But "throwing away" quality data, for no reason, is unlikely to improve a model.
- Some people have success applying a dimensionality reduction technique (like `sklearn.decomposition.PCA`) to the features before clustering. You may do this if you wish, although it may not be as helpful in this case because there are only four features to begin with.
- If you apply a transformation (e.g., `MinMaxScaler` or `StandardScaler`) to the features before clustering, you may have difficulty interpretting the means of the clusters (e.g., what is a mean Age of 0.2234??). There are two options to fix this: first, you can always reverse a transformation with the `inverse_transform` method. Second, you can just use the original dataset (i.e., before any prepropoceesing) during the interpreation step.
- You cannot change the distance metric for K-Means. (This is for theoretical reasons: K-Means only works/makes sense with Euclidean distance.)

1.0: Load data

In [8]:
```
# DO NOT MODIFY THIS CELL
df1 = pd.read_csv("https://drive.google.com/uc?export=download&id=1thHDCwQK3GijytoSSZNekAsItN_FGHtm")
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Age            505 non-null    int64  
 1   Income         505 non-null    int64  
 2   SpendingScore  505 non-null    float64
 3   Savings        505 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 15.9 KB
```

1.1: Clustering Algorithm #1

In [9]:
```
#There are 505 observations and 4 features in the dataset
df1.shape
```

Out[9]:
```
(505, 4)
```

In [10]:
```
#Printing the first few observations.
df1.head()
```

Out[10]:

	Age	Income	SpendingScore	Savings
0	58	77769	0.791329	6559.829923
1	59	81799	0.791082	5417.661426
2	62	74751	0.702657	9258.992965
3	59	74373	0.765680	7346.334504
4	87	17760	0.348778	16869.507130

In [11]:
```
#Checking the fundamental stats for each feature. All the features are numerical.
df1.describe().transpose()
```

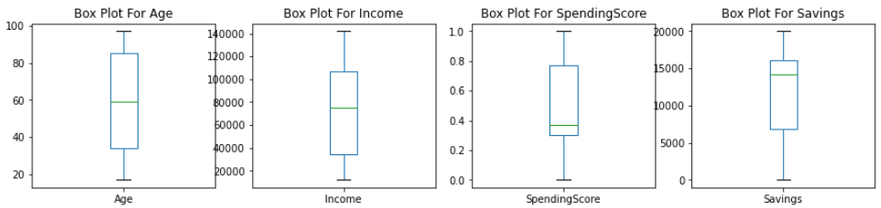
Out[11]:

	count	mean	std	min	25%	50%	75%	max
Age	505.0	59.019802	24.140043	17.0	34.000000	59.000000	85.000000	97.0
Income	505.0	75513.291089	35992.922184	12000.0	34529.000000	75078.000000	107100.000000	142000.0
SpendingScore	505.0	0.505083	0.259634	0.0	0.304792	0.368215	0.768279	1.0
Savings	505.0	11862.455867	4949.229253	0.0	6828.709702	14209.932802	16047.268331	20000.0

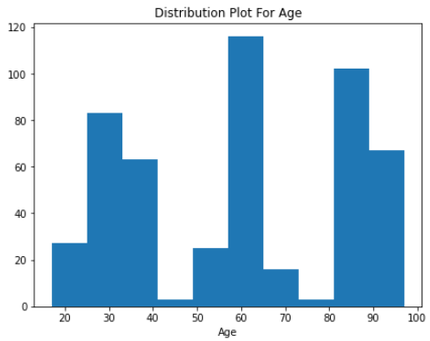
In [12]:
```
#Checking Missing values in the dataset. The data is clean and all the features are easy to understand.
#No categorical features, all the features are numeric.
Missing_Count = df1.isnull().sum()
Total_Count = df1.count()
Missing_Percent = Missing_Count/Total_Count
Missing_Data = pd.concat([Missing_Count,Missing_Percent], axis=1, keys = ["Missing Count","Missing Percent"]).sort_values('Missing Percent', ascending=False)
print(Missing_Data)
```

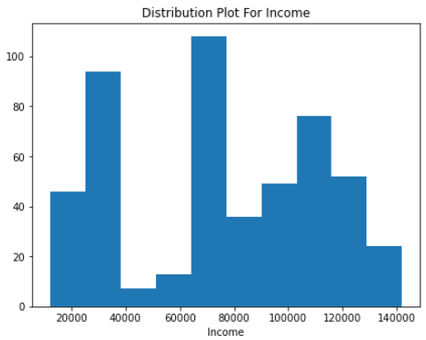
```
              Missing Count  Missing Percent
Age                       0              0.0
Income                    0              0.0
SpendingScore             0              0.0
Savings                   0              0.0
```

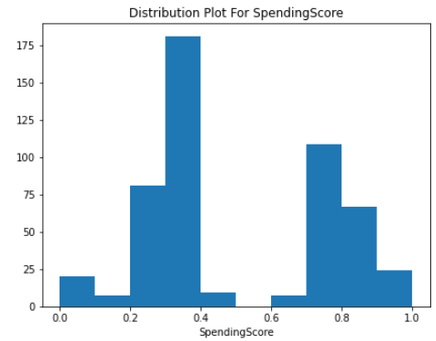
In [13]:
```
#Ploting Box Plot to see the distribution for each feature.
#The box plot gives five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum").
#The box plot will also tell what the outliers are and what their values are.
#In this case, all the features are within reasonable range, indicating that no customers are drastically different from others.
column_box = df1.columns.tolist()
fig = plt.figure(figsize=(15, 53))
for col in column_box:
    ax = fig.add_subplot(15,4, column_box.index(col)+1)
    df1[col].plot.box()
    plt.title('Box Plot For ' + col);
```

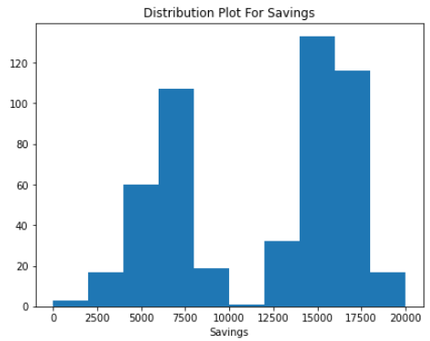


In [14]:
```
#Ploting Distribution Plot to see the distribution for each feature.
#The customers seem grouped in certain degrees.For example, customers age are in three classes.
#Young customers are under 40, middle aged/early senior customers are between 50 - 70, and seniors are more than 80 years old.
#Customers either spend a lot at Uncle Steve's store (high spending score (0.6 to 1.0) vs low spending score (0.0 to 0.4) or save a lot (low vs high savings).
numeric_cols = df1.columns.tolist()
fig = plt.figure(figsize=(40, 30))
for col in numeric_cols:
    ax = fig.add_subplot(5,5, numeric_cols.index(col)+1)
    ax.set_xlabel(col)
    ax.hist(df1[col])
    plt.title('Distribution Plot For ' + col);
```

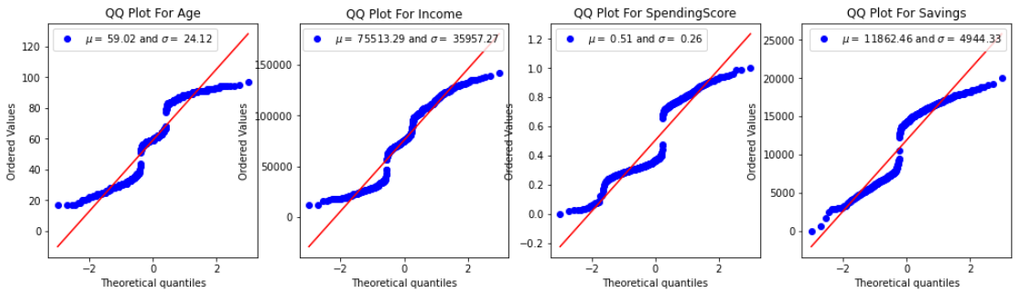






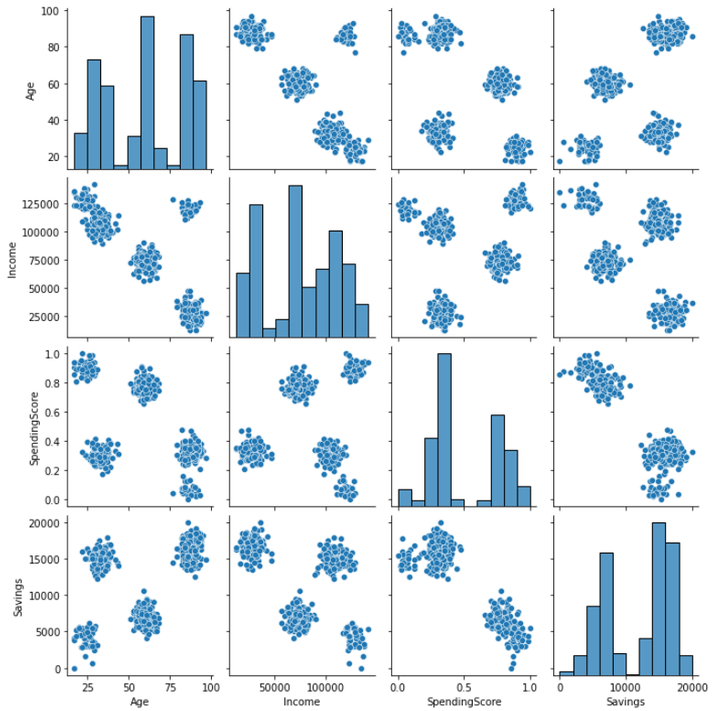


In [15]:
```
#The QQ plot line for each feature shows that points are not close to the best fit line.
#The features are not properly following a normal distribution
#There are a few observations at the tail of the distribution for each feature.
#These observations are at the extreme of the normal distribution.
column_list = df1.columns.tolist()
fig = plt.figure(figsize=(35, 35))
for col in column_list:
    ax = fig.add_subplot(7, 8, column_list.index(col)+1)
    ax.set_xlabel(col)
    stats.probplot(df1[col], dist="norm", plot=py)
    (mean, sd) = norm.fit(df1[col])
    plt.legend(['$\mu=$ {:.2f} and $\sigma=$ {:.2f}'.format(mean, sd)])
    plt.title('QQ Plot For ' + col);
```

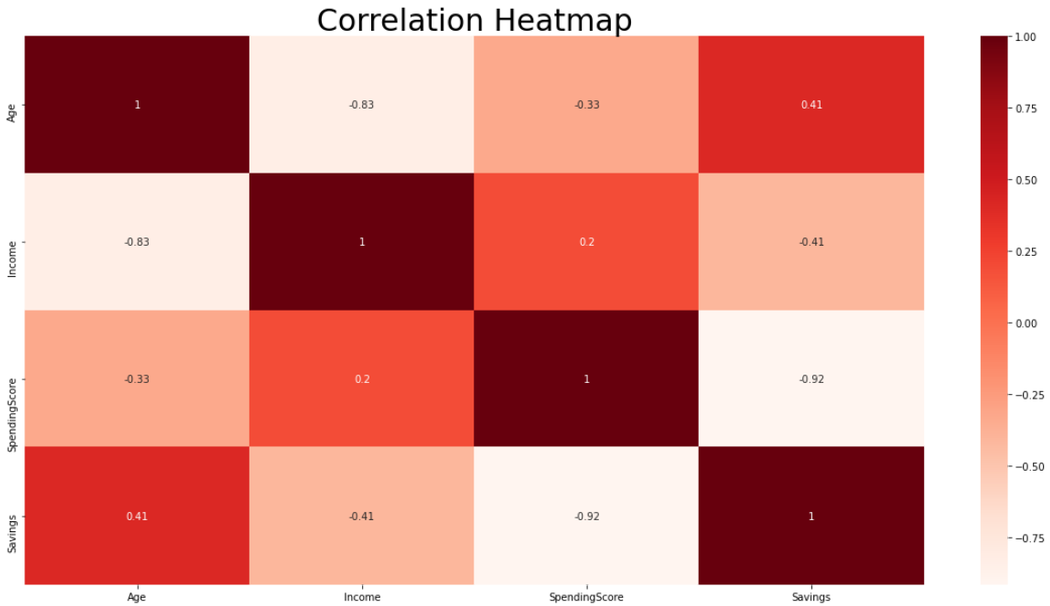


In [16]:
```
#Pair plots between the features.
#The dataset seems grouped properly between the features. The clustered patterns are very intuitive.
#For example, in the age-income chart, the young people (below age 40) tend to have high income.
#Seniors (above age 80) are that some have very low incomes and some have very high incomes.
#Another example, in the savings and spending score chart, people with low spending scores tend to have high savings.
#People with high spending scores tend to have low savings.
sns.pairplot(df1)
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x17fb7527fd0>



In [17]:
```
#Correlation heatmap between the features.
df1_corr = df1.corr()
plt.figure(figsize=(20,10))
sns.heatmap(df1_corr, annot=True, cmap="Reds")
plt.title('Correlation Heatmap',fontsize=30);
plt.show()
```



In [18]:
```
#Running Pycaret to setup the model enviroment. The data has been normalized in the environment.
```

```
#The goal of normalization is to rescale the values of numeric feautre in the dataset without distorting differences
#in the ranges of values or losing information.
#The normalized method is zscore.
#No missing values
#Random_state is set with session ID = 1
cluster1= setup(data = df1, normalize= True,session_id = 1)
```

	Description	Value
0	session_id	1
1	Original Data	(505, 4)
2	Missing Values	False
3	Numeric Features	4
4	Categorical Features	0
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(505, 4)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	cluster-default-name
13	USI	b6a0
14	Imputation Type	simple
15	Iterative Imputation Iteration	None
16	Numeric Imputer	mean
17	Iterative Imputation Numeric Model	None
18	Categorical Imputer	mode
19	Iterative Imputation Categorical Model	None
20	Unknown Categoricals Handling	least_frequent
21	Normalize	True
22	Normalize Method	zscore
23	Transformation	False
24	Transformation Method	None
25	PCA	False
26	PCA Method	None
27	PCA Components	None
28	Ignore Low Variance	False
29	Combine Rare Levels	False
30	Rare Level Threshold	None
31	Numeric Binning	False
32	Remove Outliers	False
33	Outliers Threshold	None
34	Remove Multicollinearity	False
35	Multicollinearity Threshold	None
36	Clustering	False
37	Clustering Iteration	None
38	Polynomial Features	False
39	Polynomial Degree	None
40	Trignometry Features	False
41	Polynomial Threshold	None
42	Group Features	False
43	Feature Selection	False
44	Feature Selection Method	classic
45	Features Selection Threshold	None
46	Feature Interaction	False
47	Feature Ratio	False

	Description	Value
48	Interaction Threshold	None

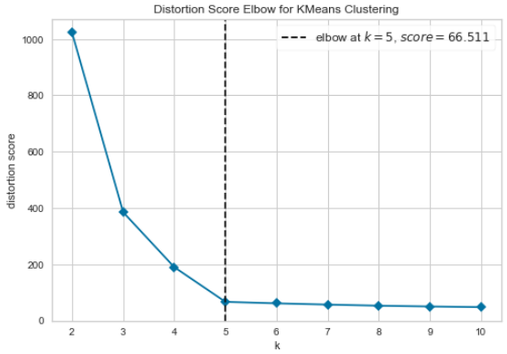
Model K = 4 Clusters (Pycaret Default Setting)

In [19]:
```
#Creating a k-mean model with Pycaret. The default K = 4 clusters, 300 iterations.
#The Silhouette score for K = 4 Kmeans model is 0.758
kmeans4 = create_model('kmeans')
kmeans4
```

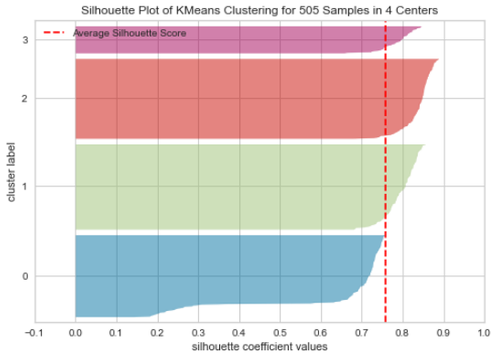
	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.7581	1611.2649	0.3743	0	0	0

Out[19]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=-1, precompute_distances='deprecated',
       random_state=1, tol=0.0001, verbose=0)
```

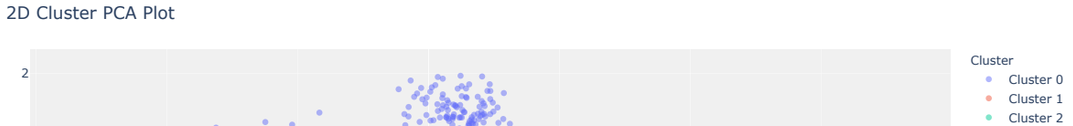
In [20]:
```
#The elbow plot.
#The distortion score is diminishing at 3 - 5 clusters.
#3 -5  clusters should be sufficient to classify customer segments.
plot_model(kmeans4, plot = 'elbow')
```

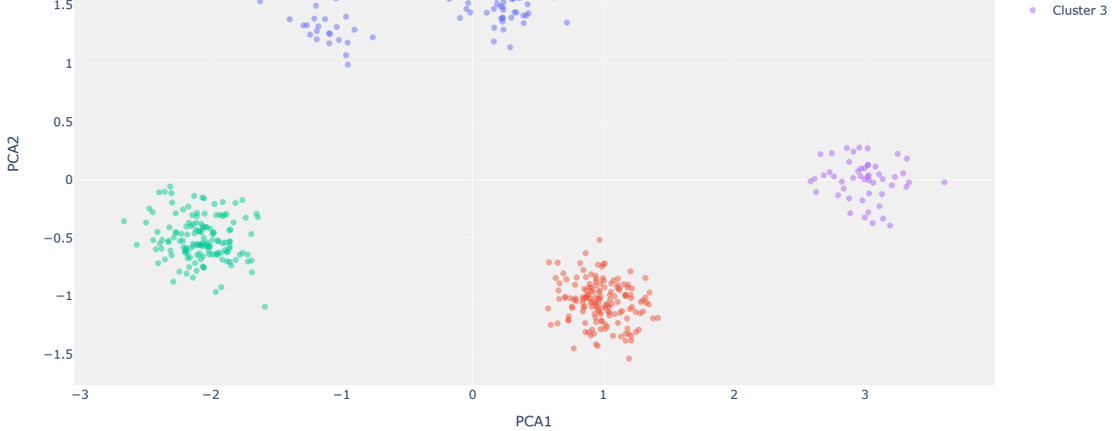


In [21]:
```
#The Silhouette Plot.
#The Silhouette plot shows that each observations from the three clusters are close to the mean score.
#The purple is thin, but is a relatively compact cluster.
#The clusters close to +1 indicating a compact cluster.
plot_model(kmeans4, plot = 'silhouette')
```

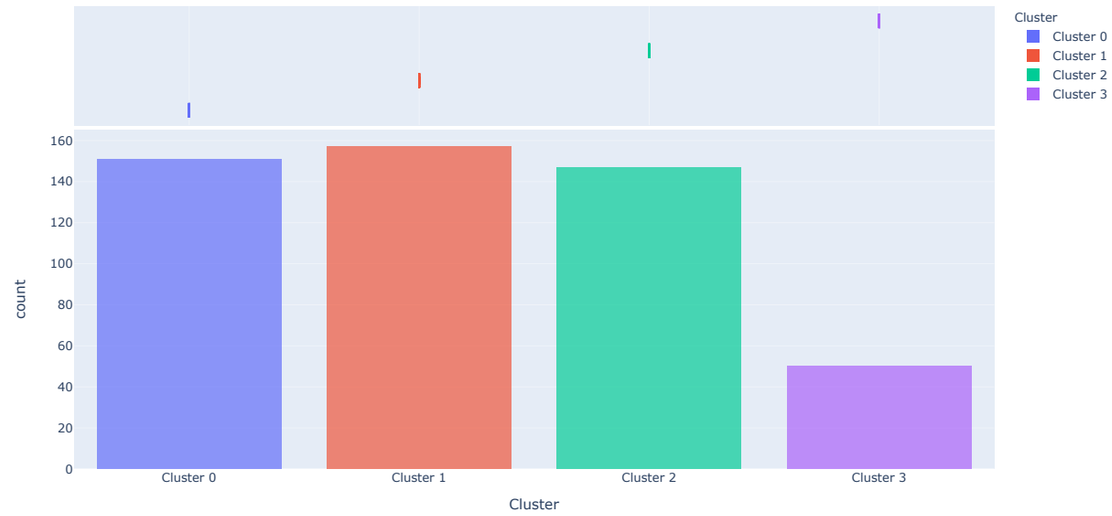


In [22]:
```
#The 2-D PCA Cluster Plot.
#The PCA Cluster plot breaks the four features into two principal components.
#From the PCA plot, each cluster is relatively compact and no overlapping with one another.
#It works by making four clusters and they are independent.
plot_model(kmeans4, plot = 'cluster')
```

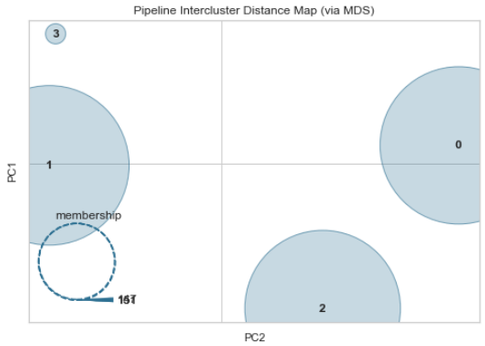




In [23]:
```
#The Distribution Plot
#The distribution plot shows that cluster 0 - 2 have equal amount of customers.
#Cluster 3 has only about half of the other three clusters.
#Maybe 3 clusters are sufficient?
plot_model(kmeans4, plot = 'distribution')
```



In [24]:
```
#The Distance Plot
#The distance plot breaks all four features into two princial components.
#Cluster 0 1 and 2 are about equal size.
#Cluster 3 is very small comparing to others.
plot_model(kmeans4, plot = 'distance')
```



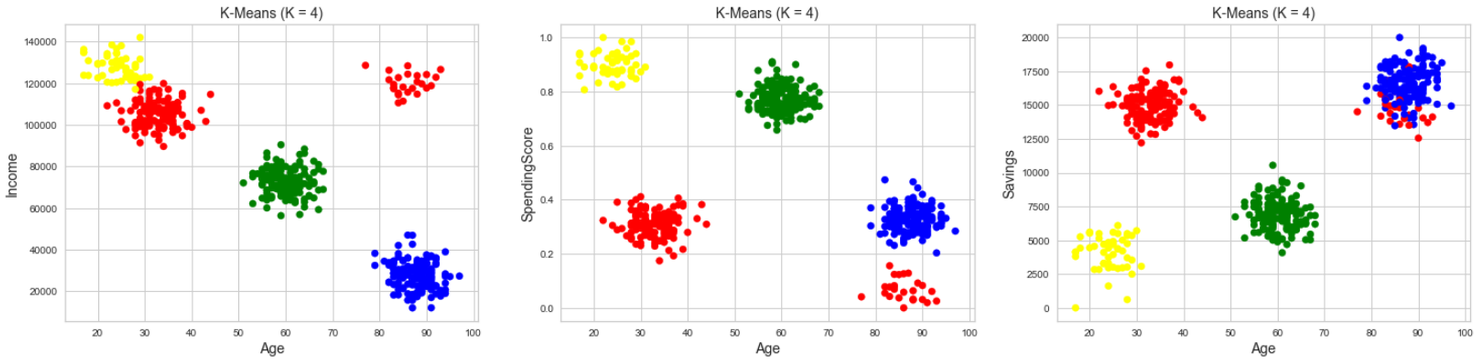
In [25]:
```
#Assigning clusters to each observation.
kmeans4
kmean_results4 = assign_model(kmeans4)
kmean_results4.head()
```

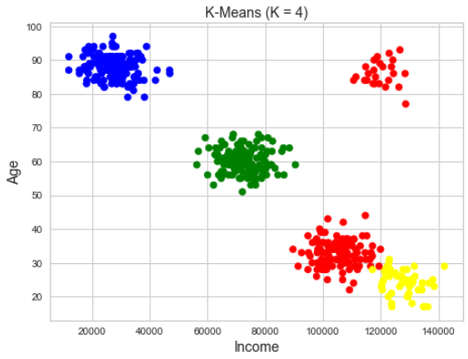
Out[25]:

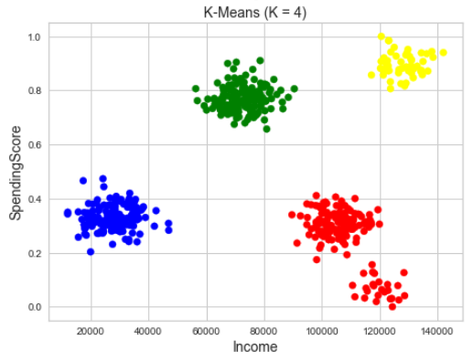
	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.829923	Cluster 1
1	59	81799	0.791082	5417.661426	Cluster 1
2	62	74751	0.702657	9258.992965	Cluster 1
3	59	74373	0.765680	7346.334504	Cluster 1
4	87	17760	0.348778	16869.507130	Cluster 2

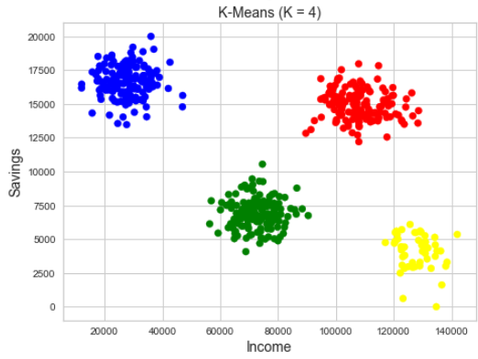
In [26]:
```
#Color code the clusters.
kmean_results4['Color'] = kmean_results4["Cluster"]
kmean_results4['Color'].replace(['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3'], ['red', 'green','blue','yellow'], inplace=True)
kmean_results4['Cluster ID'] = kmean_results4["Cluster"]
kmean_results4['Cluster ID'].replace(['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3'], [0,1,2,3], inplace=True)
```

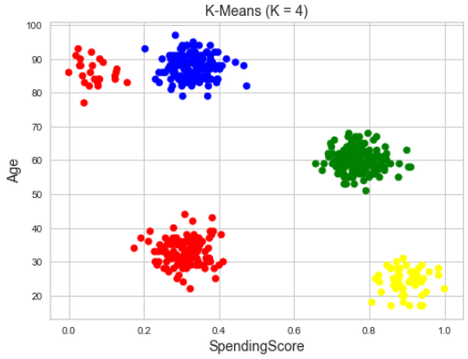
In [27]:
```
#2-D clustering result by features, colored by different clusters. There are four colors (Clusters).
#The 2-D plots show the relationship between each pair of features. Most clusters are independent but a few are overlapped.
#The overlapped clusters can merged by reducing K.
temp_graph = kmean_results4[['Age','Income','SpendingScore','Savings']]
temp_graph2 = kmean_results4[['Age','Income','SpendingScore','Savings']]
column_list = temp_graph.columns.tolist()
column_list2 = temp_graph2.columns.tolist()
j = 1
fig = plt.figure(figsize=(25, 25))
for col in column_list:
  for col2 in column_list2:
    if col == col2 :
      pass
    else:
      plt.subplot(4,3,j)
      plt.scatter(kmean_results4[col],kmean_results4[col2], c=kmean_results4["Color"])
      plt.title("K-Means (K = " + "4" + ")" , fontsize=14);
      plt.xlabel(col, fontsize=14);
      plt.ylabel(col2, fontsize=14);
      j = j + 1
```

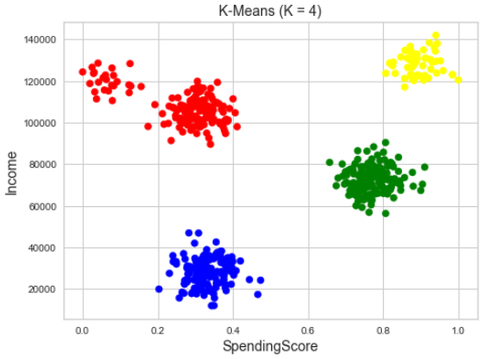


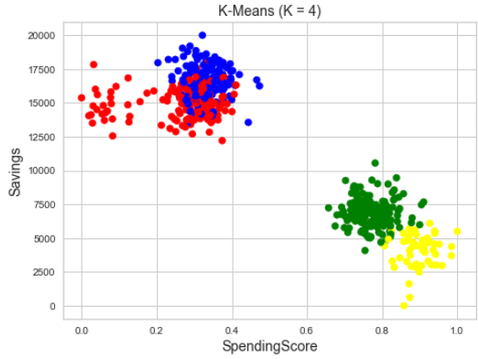


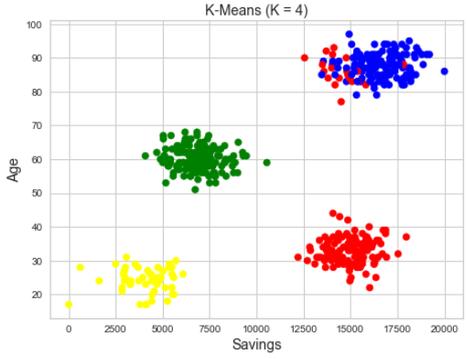


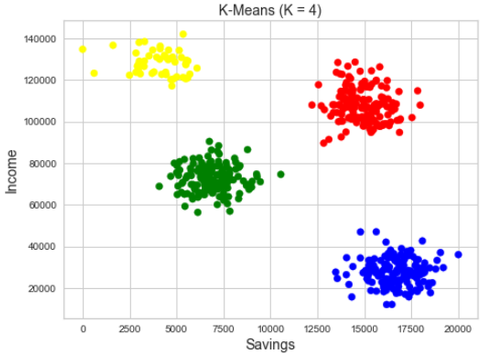


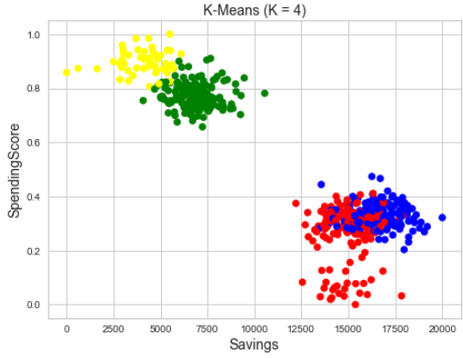












In [28]:
```
#Each Cluster shows the respective feature means.
np.set_printoptions(precision=2)
```

```
np.set_printoptions(suppress=True)
kmean_results4_example = kmean_results4.drop(["Cluster","Color"],axis = 1)
cols = kmean_results4_example.columns.tolist()
labels = kmean_results4_example['Cluster ID']
means = np.zeros((len(np.unique(kmean_results4_example["Cluster ID"])), kmean_results4_example.shape[1]-1))
for i in range(0, len(np.unique(kmean_results4_example["Cluster ID"]))):
  for j in range(0, kmean_results4_example.shape[1]-1):
     means[i,j] = kmean_results4_example[kmean_results4_example['Cluster ID']==i][cols[j]].mean(axis=0)

print("Age  Income  Spending_Score  Savings")
meansK4 = means
means
```

```
Age  Income  Spending_Score  Savings
```

Out[28]:
```
array([[    41.59, 107695.98,      0.27,  14937.27],
       [    59.96,  72448.06,      0.77,   6889.97],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    24.18, 128029.12,      0.9 ,   4087.52]])
```

In [29]:
```
#Exemplars.
#The representative from each cluster.
for i in range(0, len(np.unique(kmean_results4_example["Cluster ID"]))):
    exemplar_idx = distance.cdist([means[i]], kmean_results4_example.iloc[: , :-1]).argmin()
    print('\nCluster {}:'.format(i))
    print("  Exemplar ID: {}".format(exemplar_idx))
    display(kmean_results4_example.iloc[[exemplar_idx]])
```

```
Cluster 0:
  Exemplar ID: 130
```

	Age	Income	SpendingScore	Savings	Cluster ID
130	34	107255	0.328343	15130.595226	0

```
Cluster 1:
  Exemplar ID: 419
```

	Age	Income	SpendingScore	Savings	Cluster ID
419	51	72086	0.791115	6732.096069	1

```
Cluster 2:
  Exemplar ID: 375
```

	Age	Income	SpendingScore	Savings	Cluster ID
375	84	27384	0.313647	16734.672754	2

```
Cluster 3:
  Exemplar ID: 499
```

	Age	Income	SpendingScore	Savings	Cluster ID
499	25	128625	0.816739	4914.117127	3

In [30]:
```
#The fundamental stats for each cluster.
col_names = kmean_results4_example.columns
pd.set_option("display.precision", 2)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
def stats_to_df(d):
    tmp_df = pd.DataFrame(columns=col_names)
    tmp_df.loc[0] = d.minmax[0]
    tmp_df.loc[1] = d.mean
    tmp_df.loc[2] = d.minmax[1]
    tmp_df.loc[3] = d.variance
    tmp_df.loc[4] = d.skewness
    tmp_df.loc[5] = d.kurtosis
    tmp_df.index = ['Min', 'Mean', 'Max', 'Variance', 'Skewness', 'Kurtosis']
    return tmp_df.T
print('All Data:')
print('Number of Instances: {}'.format(kmean_results4_example.shape[0]))
d = stats.describe(kmean_results4_example, axis=0)
display(stats_to_df(d))
for i in range(0, len(np.unique(kmean_results4_example["Cluster ID"]))):
  temp = kmean_results4_example[kmean_results4_example["Cluster ID"] == i]
  d = stats.describe(temp, axis=0)
  print('\nCluster {}:'.format(i))
  print('Number of Instances: {}'.format(d.nobs))
  display(stats_to_df(d))
```

```
All Data:
Number of Instances: 505
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	59.02	97.00	582.74	-0.06	-1.42
Income	12000.00	75513.29	142000.00	1295490447.35	-0.14	-1.25
SpendingScore	0.00	0.51	1.00	0.07	0.25	-1.42
Savings	0.00	11862.46	20000.00	24494870.20	-0.36	-1.45
Cluster ID	0.00	1.19	3.00	0.95	0.25	-1.02

```
Cluster 0:
Number of Instances: 151
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	22.00	41.59	93.00	408.07	1.70	1.14
Income	89598.00	107695.98	128596.00	64681181.27	0.42	-0.21
SpendingScore	0.00	0.27	0.41	0.01	-1.26	0.56
Savings	12207.53	14937.27	17968.55	1157752.50	0.16	-0.10
Cluster ID	0.00	0.00	0.00	0.00	0.00	-3.00

```
Cluster 1:
Number of Instances: 157
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	51.00	59.96	68.00	11.40	0.18	-0.26
Income	56321.00	72448.06	90422.00	38940844.97	0.15	0.04
SpendingScore	0.66	0.77	0.91	0.00	0.40	0.30
Savings	4077.66	6889.97	10547.78	1107285.52	0.27	0.50
Cluster ID	1.00	1.00	1.00	0.00	0.00	-3.00

```
Cluster 2:
Number of Instances: 147
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	79.00	87.78	97.00	12.31	-0.01	-0.47
Income	12000.00	27866.10	46977.00	41587092.13	0.13	0.17
SpendingScore	0.20	0.33	0.47	0.00	0.31	0.48
Savings	13470.97	16659.26	20000.00	1401501.24	-0.15	0.03
Cluster ID	2.00	2.00	2.00	0.00	0.00	-3.00

```
Cluster 3:
Number of Instances: 50
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	24.18	31.00	13.42	-0.41	-0.55
Income	117108.00	128029.12	142000.00	32363636.19	0.31	-0.65
SpendingScore	0.81	0.90	1.00	0.00	0.19	-0.23
Savings	0.00	4087.52	6089.48	1632657.33	-1.02	1.25
Cluster ID	3.00	3.00	3.00	0.00	0.00	-3.00

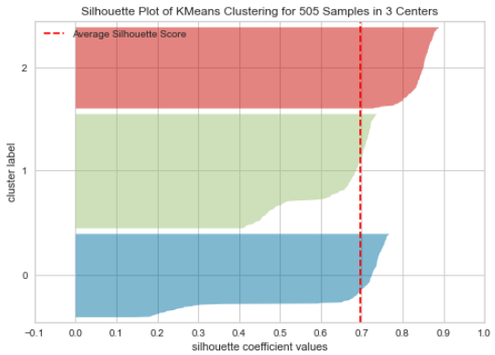
Model K = 3 Clusters

In [31]:
```
#Creat a new Kmeans model with K = 3.
#The Silhouette score for K = 3 Kmeans model is 0.7.
kmeans3 = create_model('kmeans',num_clusters =3 )
kmeans3
```

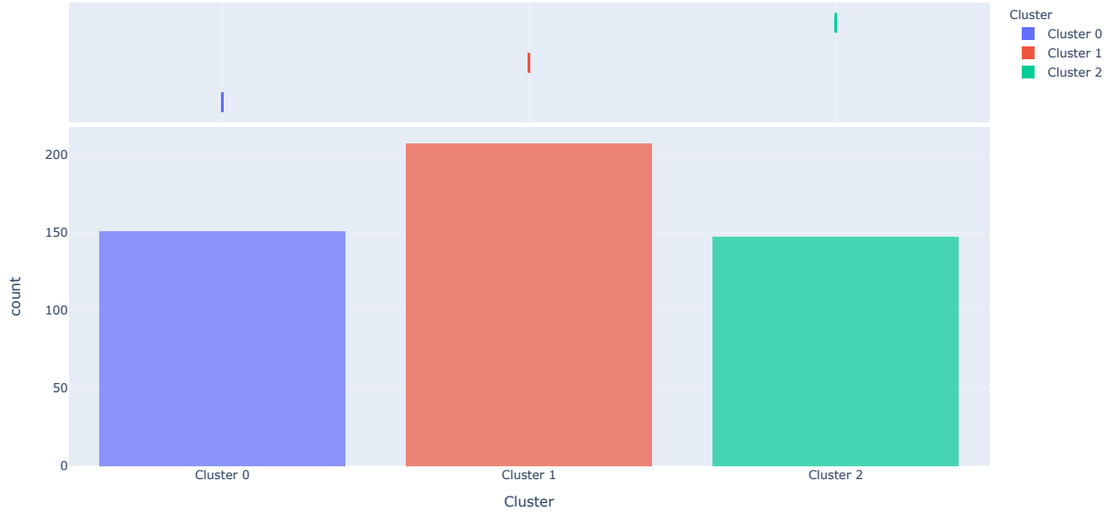
	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.70	1066.58	0.53	0	0	0

Out[31]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=-1, precompute_distances='deprecated',
       random_state=1, tol=0.0001, verbose=0)
```

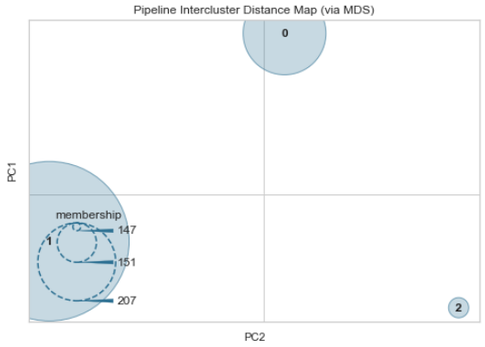
In [32]:
```
#The silhouette Plot
#All three clusters have passed the average score and each forms a reasonable compact cluster.
#All three clusters have a reasonable size.
plot_model(kmeans3, plot = 'silhouette')
```



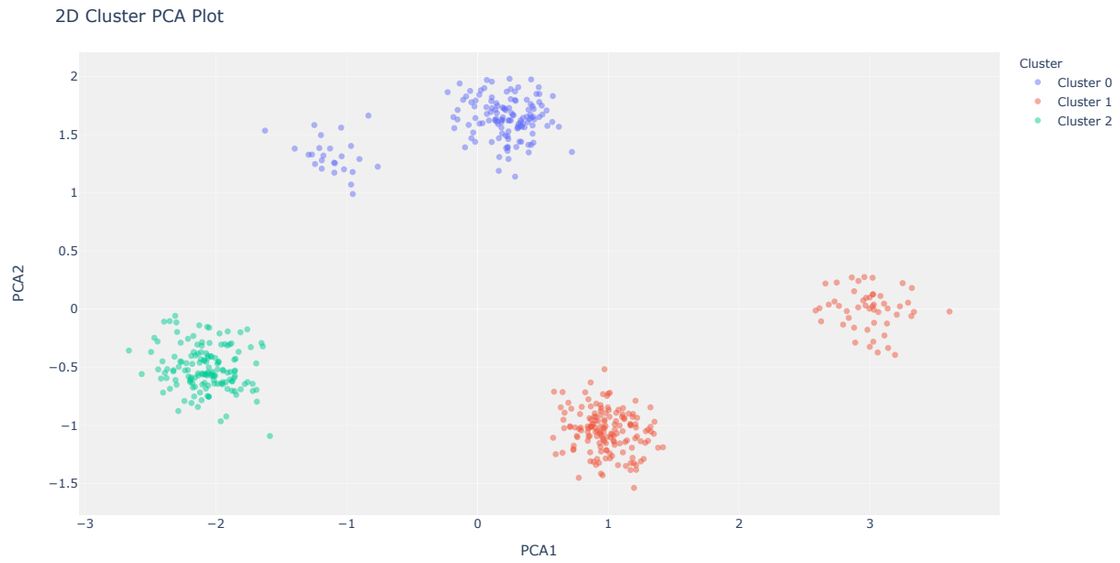
In [33]:
```
#Distribution Plot
#The size for each cluster is very similar to one another.
plot_model(kmeans3, plot = 'distribution')
```



In [34]:
```
#Distance Plot
plot_model(kmeans3, plot = 'distance')
```



In [35]:
```
#The 2-D PCA Cluster Plot.
#The PCA Cluster plot breaks the four features into two principal components.
#From the PCA plot, each cluster is relatively compact and no overlapping with one another.
plot_model(kmeans3, plot = 'cluster')
```



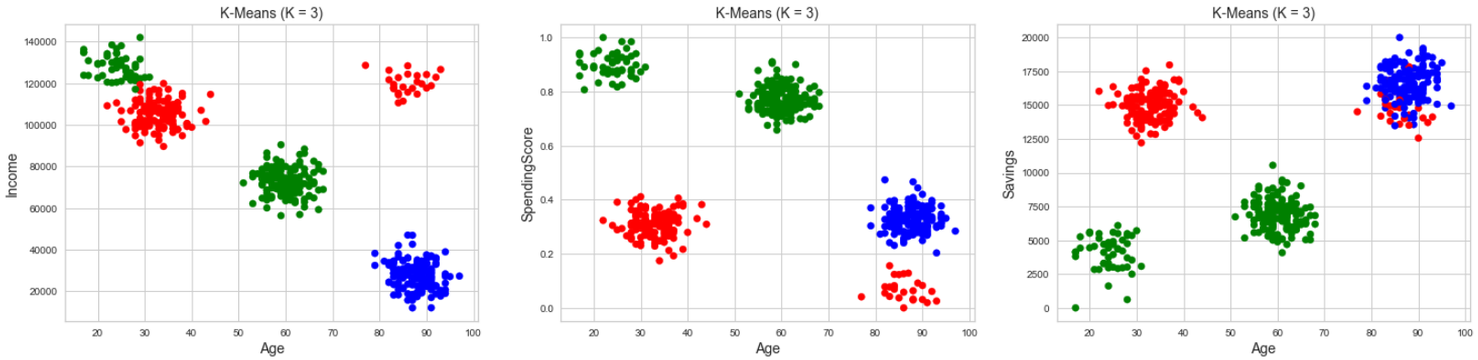
In [36]:
```
#Assigning clusters to each observation.
kmeans3
kmean_results3 = assign_model(kmeans3)
kmean_results3.head()
```

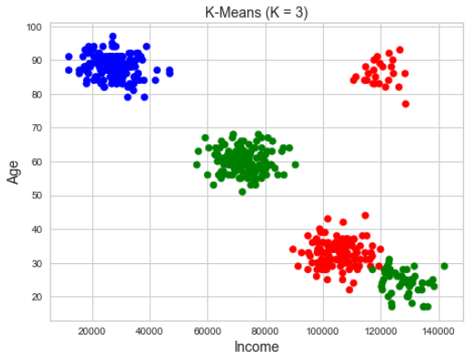
Out[36]:

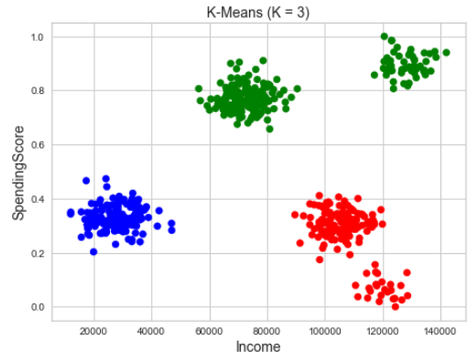
	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.79	6559.83	Cluster 1
1	59	81799	0.79	5417.66	Cluster 1
2	62	74751	0.70	9258.99	Cluster 1
3	59	74373	0.77	7346.33	Cluster 1
4	87	17760	0.35	16869.51	Cluster 2

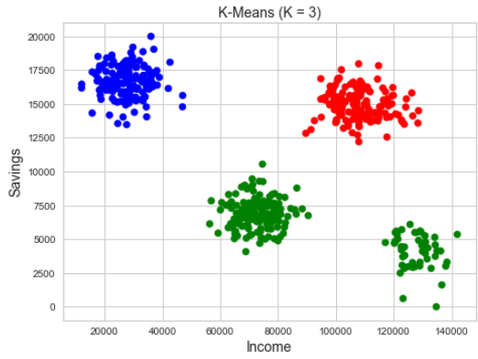
In [37]:
```
#Color code the clusters.
kmean_results3['Color'] = kmean_results3["Cluster"]
kmean_results3['Color'].replace(['Cluster 0', 'Cluster 1','Cluster 2'], ['red', 'green','blue'], inplace=True)
kmean_results3['Cluster ID'] = kmean_results3["Cluster"]
kmean_results3['Cluster ID'].replace(['Cluster 0', 'Cluster 1','Cluster 2'], [0, 1,2], inplace=True)
```

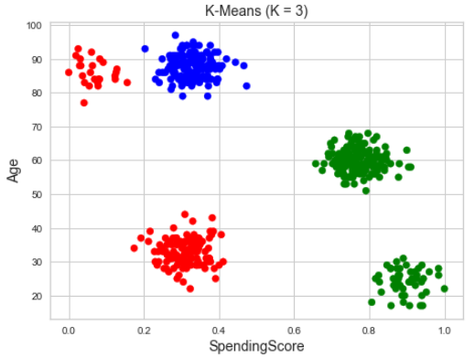
In [38]:
```
#2-D clustering result by features, colored by different clusters. There are three colors (Clusters).
temp_graph = kmean_results3[['Age','Income','SpendingScore','Savings']]
temp_graph2 = kmean_results3[['Age','Income','SpendingScore','Savings']]
column_list = temp_graph.columns.tolist()
column_list2 = temp_graph2.columns.tolist()
j = 1
fig = plt.figure(figsize=(25, 25))
for col in column_list:
  for col2 in column_list2:
    if col == col2 :
      pass
    else:
      plt.subplot(4,3,j)
      plt.scatter(kmean_results3[col],kmean_results3[col2], c=kmean_results3["Color"])
      plt.title("K-Means (K = " + "3" + ")" , fontsize=14);
      plt.xlabel(col, fontsize=14);
      plt.ylabel(col2, fontsize=14);
      j = j + 1
```

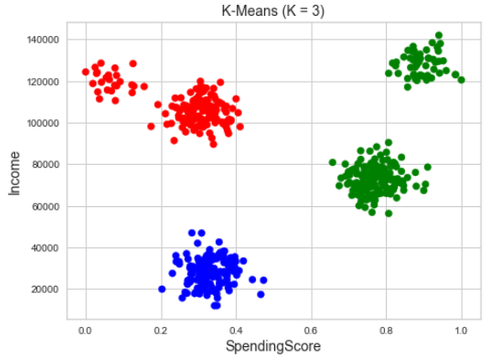


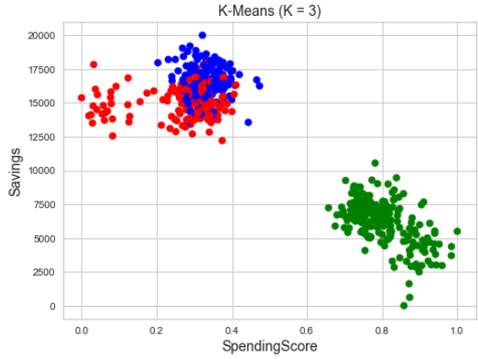


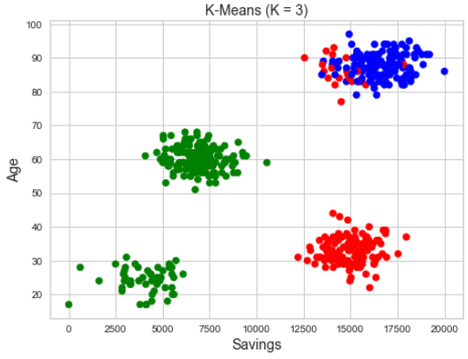


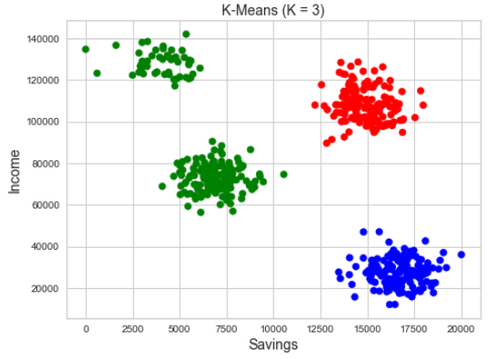


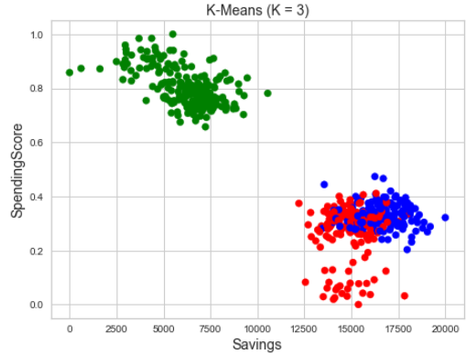












In [39]:
```
#Each Cluster shows the respective feature means.
np.set_printoptions(precision=2)
```

```
np.set_printoptions(suppress=True)
kmean_results3_example = kmean_results3.drop(["Cluster","Color"],axis = 1)
cols = kmean_results3_example.columns.tolist()
labels = kmean_results3_example['Cluster ID']
means = np.zeros((len(np.unique(kmean_results3_example["Cluster ID"])), kmean_results3_example.shape[1]-1))
for i in range(0, len(np.unique(kmean_results3_example["Cluster ID"]))):
  for j in range(0, kmean_results3_example.shape[1]-1):
     means[i,j] = kmean_results3_example[kmean_results3_example['Cluster ID']==i][cols[j]].mean(axis=0)

print("Age  Income  Spending_Score  Savings")
meansK3 = means
means
```

```
Age  Income  Spending_Score  Savings
array([[    41.59, 107695.98,      0.27,  14937.27],
       [    51.31,  85873.44,      0.8 ,   6213.05],
       [    87.78,  27866.1 ,      0.33,  16659.26]])
```

In [40]:

```
#Exemplars.
for i in range(0, len(np.unique(kmean_results3_example["Cluster ID"]))):
    exemplar_idx = distance.cdist([means[i]], kmean_results3_example.iloc[: , :-1]).argmin()
    print('\nCluster {}:'.format(i))
    print("  Exemplar ID: {}".format(exemplar_idx))
    display(kmean_results3_example.iloc[[exemplar_idx]])
```

```
Cluster 0:
  Exemplar ID: 130
```

	Age	Income	SpendingScore	Savings	Cluster ID
130	34	107255	0.33	15130.60	0

```
Cluster 1:
  Exemplar ID: 181
```

	Age	Income	SpendingScore	Savings	Cluster ID
181	64	86283	0.76	6852.55	1

```
Cluster 2:
  Exemplar ID: 375
```

	Age	Income	SpendingScore	Savings	Cluster ID
375	84	27384	0.31	16734.67	2

In [41]:

```
#The fundamental stats for each cluster.
col_names = kmean_results3_example.columns
pd.set_option("display.precision", 2)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
def stats_to_df(d):
    tmp_df = pd.DataFrame(columns=col_names)
    tmp_df.loc[0] = d.minmax[0]
    tmp_df.loc[1] = d.mean
    tmp_df.loc[2] = d.minmax[1]
    tmp_df.loc[3] = d.variance
    tmp_df.loc[4] = d.skewness
    tmp_df.loc[5] = d.kurtosis
    tmp_df.index = ['Min', 'Mean', 'Max', 'Variance', 'Skewness', 'Kurtosis']
    return tmp_df.T
print('All Data:')
print('Number of Instances: {}'.format(kmean_results3_example.shape[0]))
d = stats.describe(kmean_results3_example, axis=0)
display(stats_to_df(d))
for i in range(0, len(np.unique(kmean_results3_example["Cluster ID"]))):
  temp = kmean_results3_example[kmean_results3_example["Cluster ID"] == i]
  d = stats.describe(temp, axis=0)
  print('\nCluster {}:'.format(i))
  print('Number of Instances: {}'.format(d.nobs))
  display(stats_to_df(d))
```

```
All Data:
Number of Instances: 505
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	59.02	97.00	582.74	-0.06	-1.42
Income	12000.00	75513.29	142000.00	1295490447.35	-0.14	-1.25
SpendingScore	0.00	0.51	1.00	0.07	0.25	-1.42
Savings	0.00	11862.46	20000.00	24494870.20	-0.36	-1.45
Cluster ID	0.00	0.99	2.00	0.59	0.01	-1.31

```
Cluster 0:
Number of Instances: 151
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	22.00	41.59	93.00	408.07	1.70	1.14
Income	89598.00	107695.98	128596.00	64681181.27	0.42	-0.21
SpendingScore	0.00	0.27	0.41	0.01	-1.26	0.56

	Min	Mean	Max	Variance	Skewness	Kurtosis
Savings	12207.53	14937.27	17968.55	1157752.50	0.16	-0.10
Cluster ID	0.00	0.00	0.00	0.00	0.00	-3.00

```
Cluster 1:
Number of Instances: 207
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	51.31	68.00	247.44	-1.13	-0.46
Income	56321.00	85873.44	142000.00	605890544.89	1.09	-0.51
SpendingScore	0.66	0.80	1.00	0.00	0.58	-0.34
Savings	0.00	6213.05	10547.78	2672678.10	-0.72	1.04
Cluster ID	1.00	1.00	1.00	0.00	0.00	-3.00

```
Cluster 2:
Number of Instances: 147
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	79.00	87.78	97.00	12.31	-0.01	-0.47
Income	12000.00	27866.10	46977.00	41587092.13	0.13	0.17
SpendingScore	0.20	0.33	0.47	0.00	0.31	0.48
Savings	13470.97	16659.26	20000.00	1401501.24	-0.15	0.03
Cluster ID	2.00	2.00	2.00	0.00	0.00	-3.00

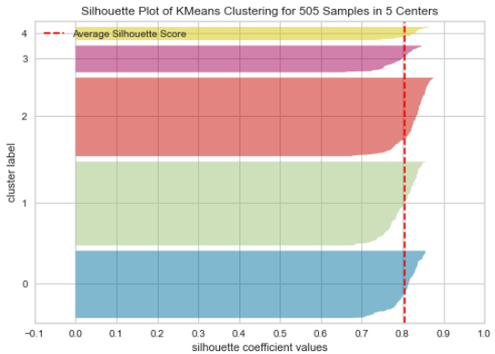
Model K = 5 Clusters

In [42]:
```
#Creating a new Kmeans model with K = 5.
#The Silhouette score for the model is 0.8
kmeans5 = create_model('kmeans',num_clusters = 5 )
kmeans5
```

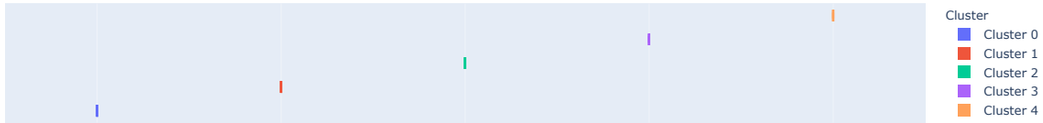
	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.80	3671.36	0.28	0	0	0

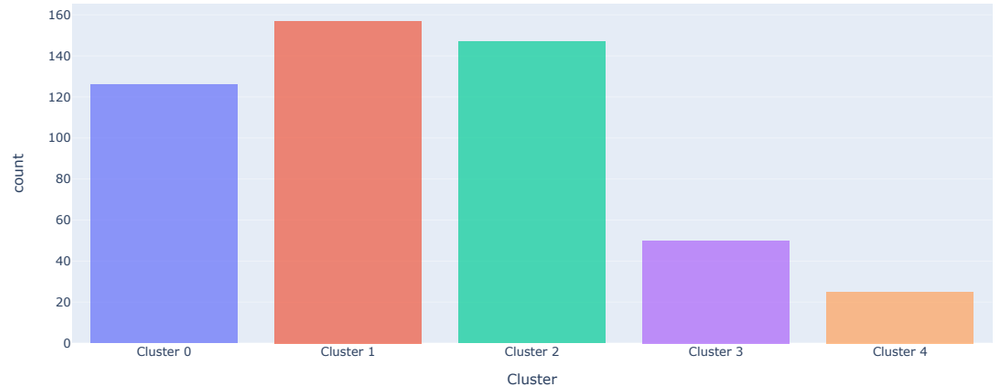
Out[42]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='deprecated',
       random_state=1, tol=0.0001, verbose=0)
```

In [43]:
```
#The Silhouette Plot.
#The plot shows that all five clusters are relatively compact and have passed the average score.
#The drawback is that the yellow and purplse cluster are relatively thin compaing to the other clusters.
#These two clusters might be able to merge with other clusters.
plot_model(kmeans5, plot = 'silhouette')
```

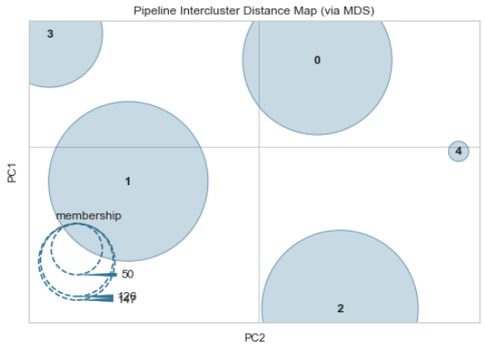


In [44]:
```
#Distribution Plot.
#The distribution plot shows that Cluster 3 and Cluster 4 are relatively small.
plot_model(kmeans5, plot = 'distribution')
```

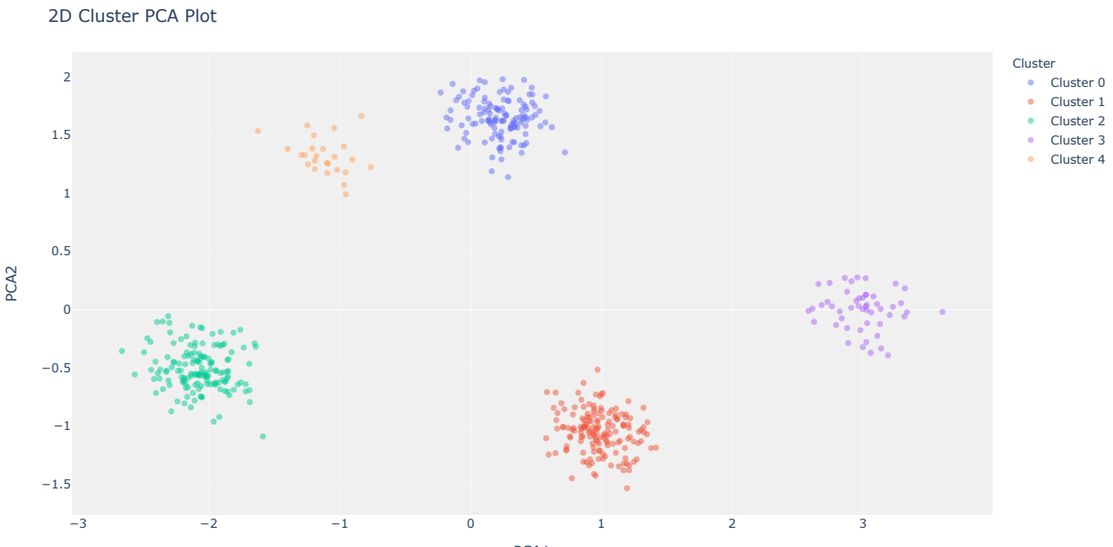




In [45]:
```
#Distance Plot.
plot_model(kmeans5, plot = 'distance')
```



In [46]:
```
#The 2-D PCA Cluster Plot.
#The PCA Cluster plot breaks the four features into two principal components.
#From the PCA plot, each cluster is relatively compact and no overlapping with one another.
plot_model(kmeans5, plot = 'cluster')
```



PCA1

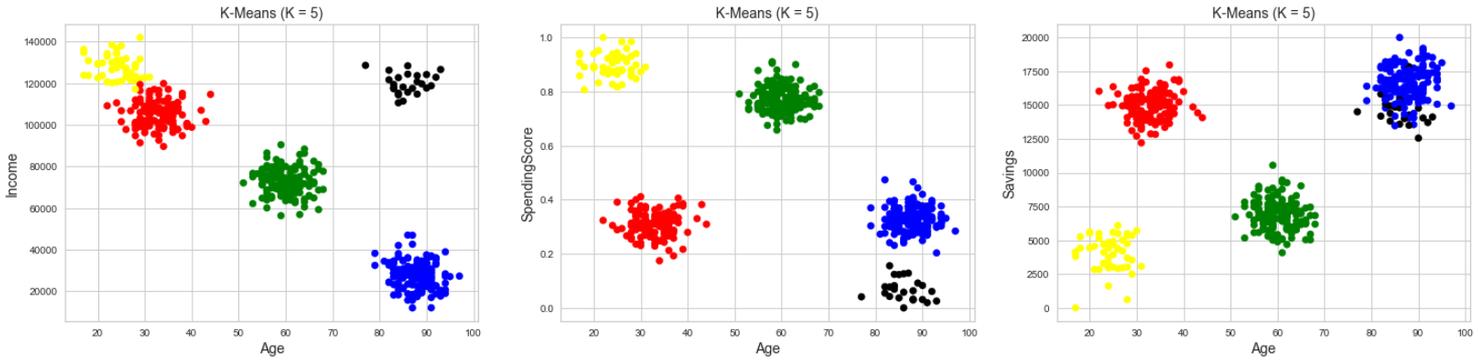
In [47]:
```
#Assigning clusters to each observation.
kmeans5
kmean_results5 = assign_model(kmeans5)
kmean_results5.head()
```

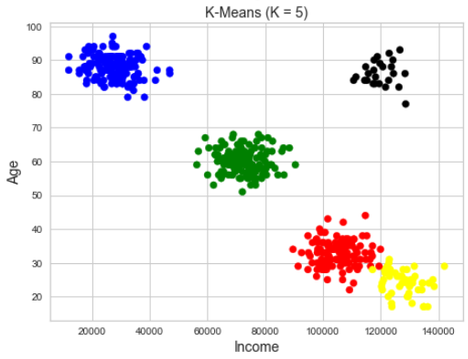
Out[47]:

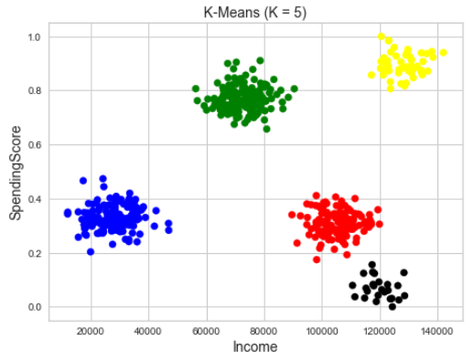
	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.79	6559.83	Cluster 1
1	59	81799	0.79	5417.66	Cluster 1
2	62	74751	0.70	9258.99	Cluster 1
3	59	74373	0.77	7346.33	Cluster 1
4	87	17760	0.35	16869.51	Cluster 2

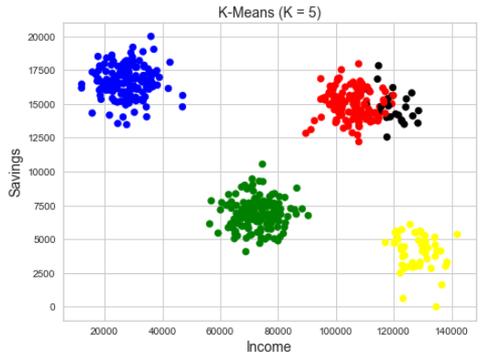
In [48]:
```
#Color code the clusters.
kmean_results5['Color'] = kmean_results5["Cluster"]
kmean_results5['Color'].replace(['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cluster 4'], ['red', 'green','blue','yellow','black'], inplace=True)
kmean_results5['Cluster ID'] = kmean_results5["Cluster"]
kmean_results5['Cluster ID'].replace(['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cluster 4'], [0,1,2,3,4], inplace=True)
```

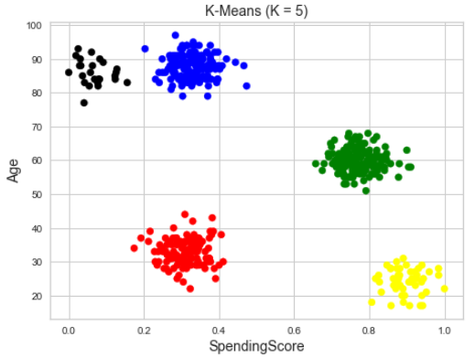
In [49]:
```
#2-D clustering result by features, colored by different clusters. There are five colors (Clusters).
#The plots show more overlapping clusters than the in the K = 4 and K = 3 models.
#More clusters will show that some clusters have similar characteristics.
temp_graph = kmean_results5[['Age','Income','SpendingScore','Savings']]
temp_graph2 = kmean_results5[['Age','Income','SpendingScore','Savings']]
column_list = temp_graph.columns.tolist()
column_list2 = temp_graph2.columns.tolist()
j = 1
fig = plt.figure(figsize=(25, 25))
for col in column_list:
  for col2 in column_list2:
    if col == col2 :
      pass
    else:
      plt.subplot(4,3,j)
      plt.scatter(kmean_results5[col],kmean_results5[col2], c=kmean_results5["Color"])
      plt.title("K-Means (K = " + "5" + ")" , fontsize=14);
      plt.xlabel(col, fontsize=14);
      plt.ylabel(col2, fontsize=14);
      j = j + 1
```

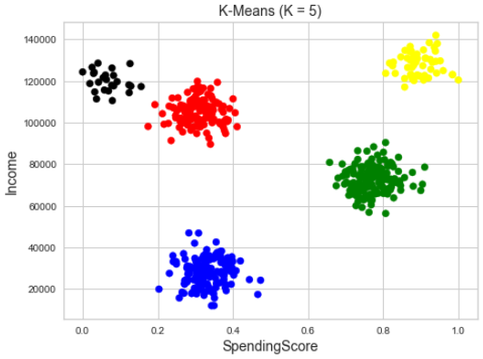


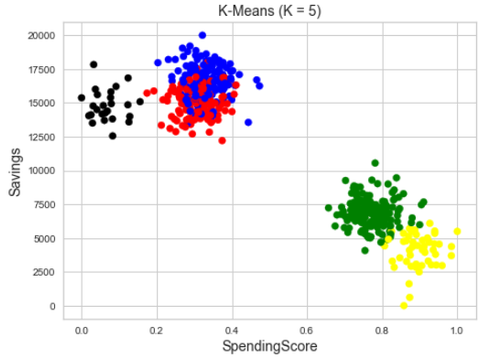


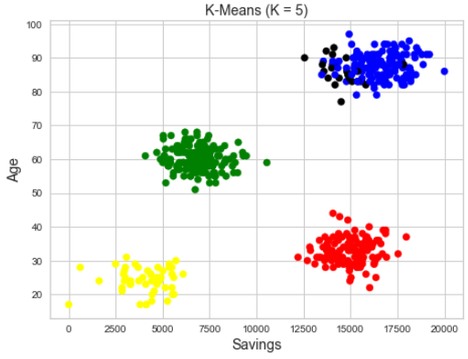


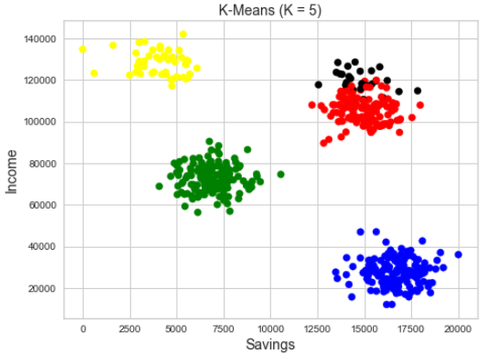


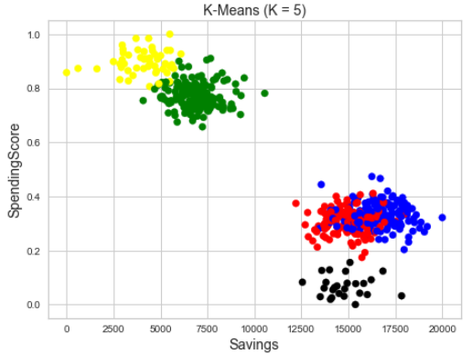












In [50]:
```
#Each Cluster shows the respective feature means.
np.set_printoptions(precision=2)
```

```
np.set_printoptions(suppress=True)
kmean_results5_example = kmean_results5.drop(["Cluster","Color"],axis = 1)
cols = kmean_results5_example.columns.tolist()
labels = kmean_results5_example['Cluster ID']
means = np.zeros((len(np.unique(kmean_results5_example["Cluster ID"])), kmean_results5_example.shape[1]-1))
for i in range(0, len(np.unique(kmean_results5_example["Cluster ID"]))):
  for j in range(0, kmean_results5_example.shape[1]-1):
     means[i,j] = kmean_results5_example[kmean_results5_example['Cluster ID']==i][cols[j]].mean(axis=0)

print("Age  Income  Spending_Score  Savings")
meansK5 = means
means
```

```
Age  Income  Spending_Score  Savings
```

Out[50]:
```
array([[    32.78, 105265.81,      0.31,  14962.78],
       [    59.96,  72448.06,      0.77,   6889.97],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    24.18, 128029.12,      0.9 ,   4087.52],
       [    86.  , 119944.04,      0.07,  14808.68]])
```

In [51]:
```
#Exemplars.
#In the cluster 4, the sample is pulled from cluster 0, meaning that the clusters are overlapping.
#Creating more clusters will expose clusters with similar characteristics.
for i in range(0, len(np.unique(kmean_results5_example["Cluster ID"]))):
    exemplar_idx = distance.cdist([means[i]], kmean_results5_example.iloc[: , :-1]).argmin()
    print('\nCluster {}:'.format(i))
    print("  Exemplar ID: {}".format(exemplar_idx))
    display(kmean_results5_example.iloc[[exemplar_idx]])
```

```
Cluster 0:
  Exemplar ID: 395
```

	Age	Income	SpendingScore	Savings	Cluster ID
395	31	105006	0.37	15419.42	0

```
Cluster 1:
  Exemplar ID: 419
```

	Age	Income	SpendingScore	Savings	Cluster ID
419	51	72086	0.79	6732.10	1

```
Cluster 2:
  Exemplar ID: 375
```

	Age	Income	SpendingScore	Savings	Cluster ID
375	84	27384	0.31	16734.67	2

```
Cluster 3:
  Exemplar ID: 499
```

	Age	Income	SpendingScore	Savings	Cluster ID
499	25	128625	0.82	4914.12	3

```
Cluster 4:
  Exemplar ID: 224
```

	Age	Income	SpendingScore	Savings	Cluster ID
224	29	119366	0.36	15012.85	0

In [52]:
```
#The fundamental stats for each cluster.
col_names = kmean_results5_example.columns
pd.set_option("display.precision", 2)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
def stats_to_df(d):
    tmp_df = pd.DataFrame(columns=col_names)
    tmp_df.loc[0] = d.minmax[0]
    tmp_df.loc[1] = d.mean
    tmp_df.loc[2] = d.minmax[1]
    tmp_df.loc[3] = d.variance
    tmp_df.loc[4] = d.skewness
    tmp_df.loc[5] = d.kurtosis
    tmp_df.index = ['Min', 'Mean', 'Max', 'Variance', 'Skewness', 'Kurtosis']
    return tmp_df.T
print('All Data:')
print('Number of Instances: {}'.format(kmean_results5_example.shape[0]))
d = stats.describe(kmean_results5_example, axis=0)
display(stats_to_df(d))
for i in range(0, len(np.unique(kmean_results5_example["Cluster ID"]))):
  temp = kmean_results5_example[kmean_results5_example["Cluster ID"] == i]
  d = stats.describe(temp, axis=0)
  print('\nCluster {}:'.format(i))
  print('Number of Instances: {}'.format(d.nobs))
  display(stats_to_df(d))
```

```
All Data:
Number of Instances: 505
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	59.02	97.00	582.74	-0.06	-1.42

	Min	Mean	Max	Variance	Skewness	Kurtosis
Income	12000.00	75513.29	142000.00	1295490447.35	-0.14	-1.25
SpendingScore	0.00	0.51	1.00	0.07	0.25	-1.42
Savings	0.00	11862.46	20000.00	24494870.20	-0.36	-1.45
Cluster ID	0.00	1.39	4.00	1.23	0.50	-0.40

```
Cluster 0:
Number of Instances: 126
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	22.00	32.78	44.00	14.38	0.16	0.33
Income	89598.00	105265.81	119877.00	36973960.91	0.03	-0.30
SpendingScore	0.17	0.31	0.41	0.00	-0.23	0.01
Savings	12207.53	14962.78	17968.55	1127279.12	0.06	-0.18
Cluster ID	0.00	0.00	0.00	0.00	0.00	-3.00

```
Cluster 1:
Number of Instances: 157
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	51.00	59.96	68.00	11.40	0.18	-0.26
Income	56321.00	72448.06	90422.00	38940844.97	0.15	0.04
SpendingScore	0.66	0.77	0.91	0.00	0.40	0.30
Savings	4077.66	6889.97	10547.78	1107285.52	0.27	0.50
Cluster ID	1.00	1.00	1.00	0.00	0.00	-3.00

```
Cluster 2:
Number of Instances: 147
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	79.00	87.78	97.00	12.31	-0.01	-0.47
Income	12000.00	27866.10	46977.00	41587092.13	0.13	0.17
SpendingScore	0.20	0.33	0.47	0.00	0.31	0.48
Savings	13470.97	16659.26	20000.00	1401501.24	-0.15	0.03
Cluster ID	2.00	2.00	2.00	0.00	0.00	-3.00

```
Cluster 3:
Number of Instances: 50
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	24.18	31.00	13.42	-0.41	-0.55
Income	117108.00	128029.12	142000.00	32363636.19	0.31	-0.65
SpendingScore	0.81	0.90	1.00	0.00	0.19	-0.23
Savings	0.00	4087.52	6089.48	1632657.33	-1.02	1.25
Cluster ID	3.00	3.00	3.00	0.00	0.00	-3.00

```
Cluster 4:
Number of Instances: 25
```

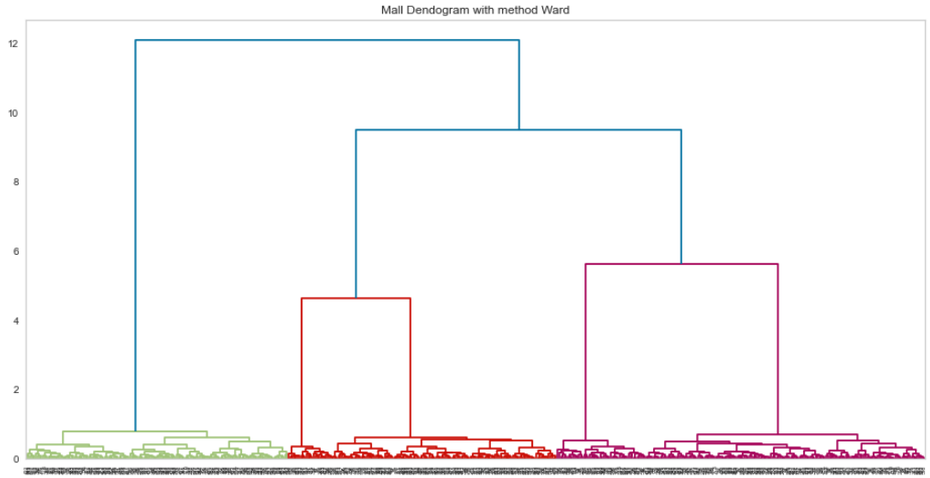
	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	77.00	86.00	93.00	13.42	-0.09	-0.02
Income	110582.00	119944.04	128596.00	24413997.37	0.02	-0.75
SpendingScore	0.00	0.07	0.16	0.00	0.46	-0.56
Savings	12554.69	14808.68	17833.09	1344068.40	0.62	0.44
Cluster ID	4.00	4.00	4.00	0.00	0.00	-3.00

1.2: Clustering Algorithm #2

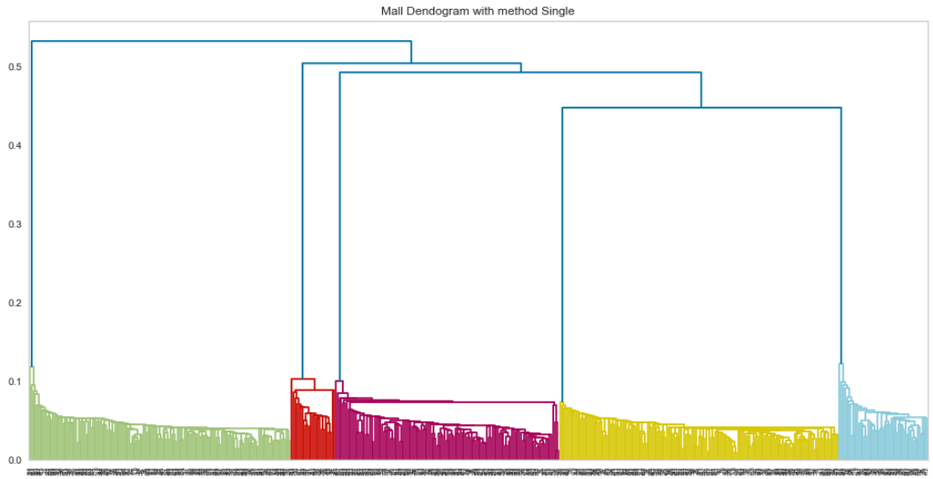
In [53]:
```
#Copying two sets of data from the original dataset.
#cl_data2 is for the final set to add the labels back for exemplars and visuals, and cl_data2_transform is for scaling and modeling.
cl_data2 = df1.copy()
cl_data2_transform = df1.copy()
```

In [54]:
```
#Applying min and max scaler on the features.
#Since the dataset doesn't contain outliers, the min and max scaler can make all the features end up on the same scale.
#All features will be transformed into the range [0,1], meaning that the minimum and maximum value of a feature is between 0 and 1.
features = ["Age","Income","SpendingScore","Savings"]
scaler = MinMaxScaler()
scaler.fit(cl_data2_transform[features])
cl_data2_transform[features] = scaler.transform(cl_data2_transform[features])
```

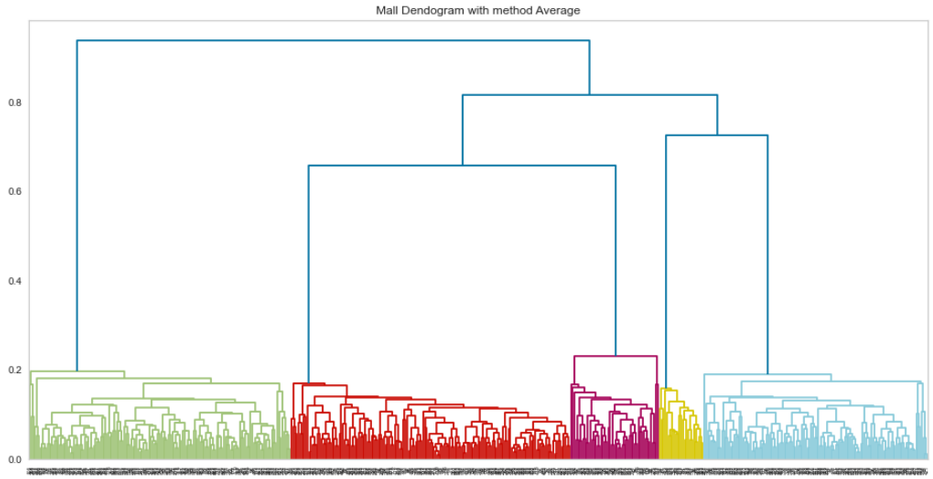
In [55]:
```
#Ploting dendogram with different methods.
#The dendogram shows that it is sufficient to classify all observations with a few clusters.
#This dendogram uses method Ward.
aggl = scipy.cluster.hierarchy.linkage(cl_data2_transform, method='ward', metric='euclidean')
plt.figure(figsize=(16, 8));
plt.grid(False)
plt.title("Mall Dendogram with method Ward");
dend = scipy.cluster.hierarchy.dendrogram(aggl);
```



In [56]:
```
#Ploting dendogram with different methods.
#Again, the dendogram shows that all the observations can be classified with a few clusters.
#The method in this dendogram is Single, not a significant different with the method Ward.
aggl = scipy.cluster.hierarchy.linkage(cl_data2_transform, method='single', metric='euclidean')
plt.figure(figsize=(16, 8));
plt.grid(False)
plt.title("Mall Dendogram with method Single");
dend = scipy.cluster.hierarchy.dendrogram(aggl);
```



In [57]:
```
#Ploting dendogram with different methods.
#Again, a few clusters will do the job.
#The method is Average and provides very similar results as above.
aggl = scipy.cluster.hierarchy.linkage(cl_data2_transform, method='average', metric='euclidean')
plt.figure(figsize=(16, 8));
plt.grid(False)
plt.title("Mall Dendogram with method Average");
dend = scipy.cluster.hierarchy.dendrogram(aggl);
```



In [58]:
```
#Building Agglomerative Clustering model with 5 clusters and linkage ward.
agg = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
agg.fit(cl_data2_transform)
```

Out[58]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=5)
```

In [59]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[59]:
```
0.8156911152768872
```

In [60]:
```
#Building Agglomerative Clustering model with 4 clusters and linkage ward.
agg = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
agg.fit(cl_data2_transform)
```

Out[60]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=4)
```

In [61]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[61]:
```
0.7569364964839269
```

In [62]:
```
#Building Agglomerative Clustering model with 3 clusters and linkage ward.
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
agg.fit(cl_data2_transform)
```

Out[62]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=3)
```

In [63]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[63]:
```
0.6823512495574657
```

In [64]:
```
#Building Agglomerative Clustering model with 5 clusters and linkage Single.
agg = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='single')
agg.fit(cl_data2_transform)
```

Out[64]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='single', memory=None, n_clusters=5)
```

In [65]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[65]:
```
0.8156911152768872
```

In [66]:
```
#Building Agglomerative Clustering model with 4 clusters and linkage Single.
agg = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='single')
agg.fit(cl_data2_transform)
```

Out[66]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='single', memory=None, n_clusters=4)
```

In [67]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[67]:
```
0.7273564342674617
```

In [68]:
```
#Building Agglomerative Clustering model with 4 clusters and linkage average.
agg = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='average')
agg.fit(cl_data2_transform)
```

Out[68]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='average', memory=None, n_clusters=4)
```

In [69]:
```
#The Silhouette score.
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[69]:
```
0.7273564342674617
```

In [70]:
```
#Building Agglomerative Clustering model with 3 clusters and linkage average.
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='average')
agg.fit(cl_data2_transform)
```

Out[70]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='average', memory=None, n_clusters=3)
```

In [71]:
```
#The Silhouette score
silhouette_score(cl_data2_transform, agg.labels_)
```

Out[71]:
```
0.6823512495574657
```

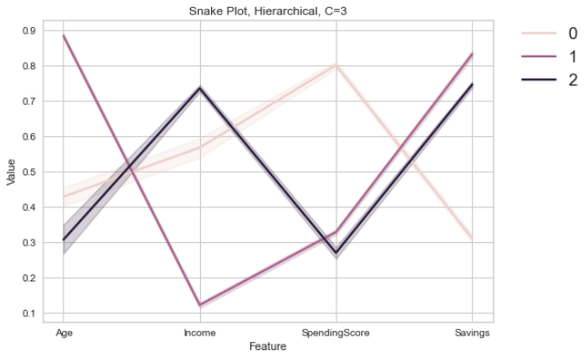
Exemplars with 3 clusters from Agglomerative modeling

In [72]:
```
#Building Agglomerative Clustering model with 3 clusters and linkage Ward.
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
agg.fit(cl_data2_transform)
```

Out[72]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=3)
```

In [73]:
```
#Add the labels from the model back to the dataset for evaluation.
cl_data2["Cluster ID"] = agg.labels_
```

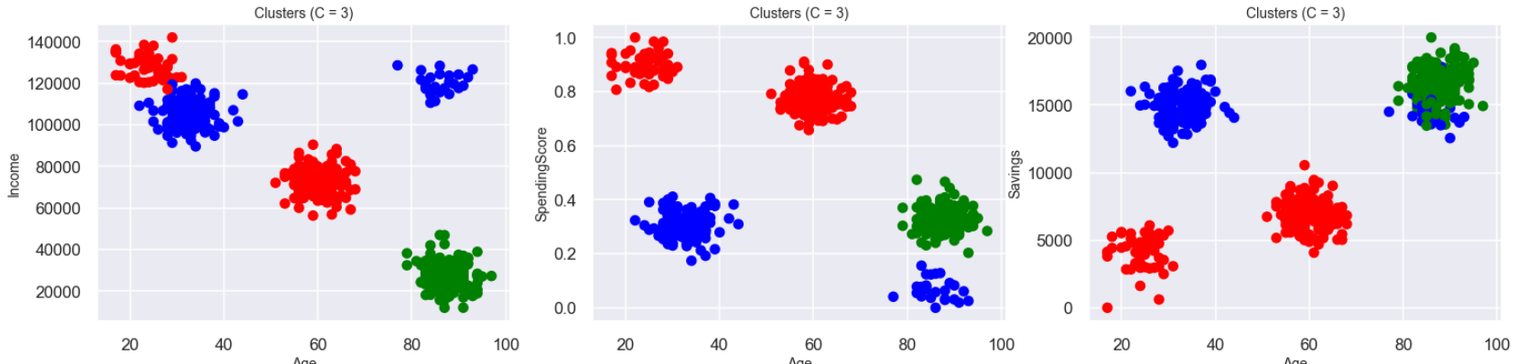
In [74]:
```
#The Snakeplot.
#The plot shows the feature importance for each cluster.
#For cluster 2, the most important feature is SpendingScore.
#For cluster 0, the most important feature is Income.
cl_data2_snake = cl_data2_transform.copy()
cl_data2_snake["Cluster ID"] = agg.labels_
X_df_melt = pd.melt(cl_data2_snake,
    id_vars=["Cluster ID"],
    value_vars=['Age', 'Income', 'SpendingScore', 'Savings'],
    var_name='Feature',
    value_name='Value')

plt.title('Snake Plot, Hierarchical, C=3');
sns.set(style="darkgrid")
sns.set_context("talk")
sns.lineplot(x="Feature", y="Value", hue='Cluster ID', data=X_df_melt, legend="full");
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
```

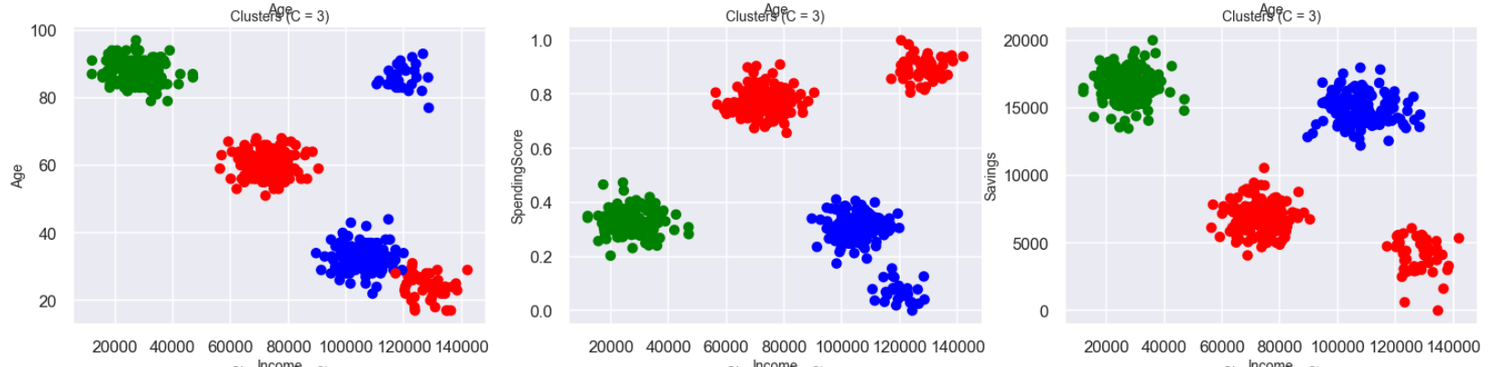


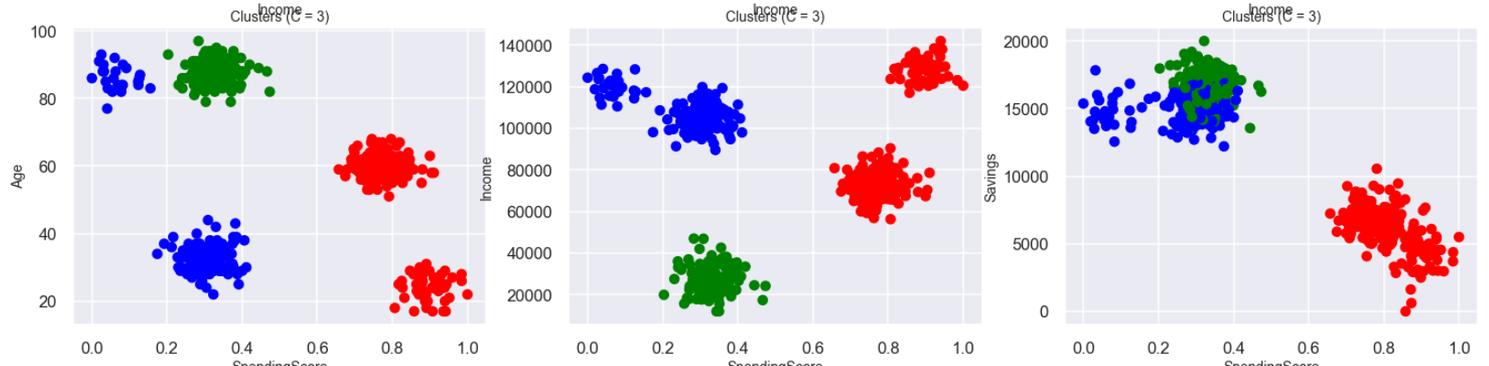
In [75]:
```
#Color code the clusters.
cl_data2['Color'] = cl_data2["Cluster ID"]
cl_data2['Color'].replace([0, 1,2], ['red', 'green','blue'], inplace=True)
```

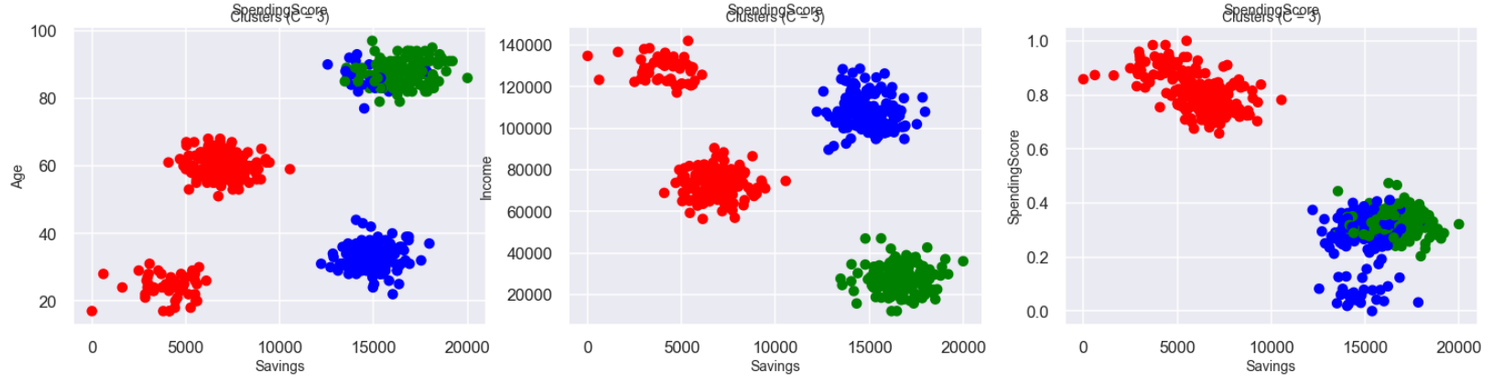
In [76]:
```
#2-D clustering result by features, colored by different clusters. There are three colors (Clusters).
```

```
temp_graph = cl_data2[['Age','Income','SpendingScore','Savings']]
temp_graph2 = cl_data2[['Age','Income','SpendingScore','Savings']]
column_list = temp_graph.columns.tolist()
column_list2 = temp_graph2.columns.tolist()
j = 1
fig = plt.figure(figsize=(25, 25))
for col in column_list:
  for col2 in column_list2:
    if col == col2 :
      pass
    else:
      plt.subplot(4,3,j)
      plt.scatter(cl_data2[col],cl_data2[col2], c=cl_data2["Color"])
      plt.title("Clusters (C = " + "3" + ")" , fontsize=14);
      plt.xlabel(col, fontsize=14);
      plt.ylabel(col2, fontsize=14);
      j = j + 1
```









In [77]:
```
#Checking the first few observations.
```

```
cl_data2.head()
```

Out[77]:

	Age	Income	SpendingScore	Savings	Cluster ID	Color
0	58	77769	0.79	6559.83	0	red
1	59	81799	0.79	5417.66	0	red
2	62	74751	0.70	9258.99	0	red
3	59	74373	0.77	7346.33	0	red
4	87	17760	0.35	16869.51	1	green

In [78]:
```
#Each Cluster shows the respective feature means.
cl_data2_example = cl_data2.drop(["Color"],axis = 1)
cols = cl_data2_example.columns.tolist()
labels = cl_data2_example['Cluster ID']
means = np.zeros((len(np.unique(cl_data2_example["Cluster ID"])), cl_data2_example .shape[1]-1))
for i in range(0, len(np.unique(cl_data2_example["Cluster ID"]))):
  for j in range(0, cl_data2_example.shape[1]-1):
     means[i,j] = cl_data2_example[cl_data2_example['Cluster ID']==i][cols[j]].mean(axis=0)

print("Age  Income  Spending_Score  Savings")
meansH3 = means
means
```

```
Age  Income  Spending_Score  Savings
```

Out[78]:
```
array([[    51.31,  85873.44,      0.8 ,   6213.05],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    41.59, 107695.98,      0.27,  14937.27]])
```

In [79]:
```
#Exemplars.
for i in range(0, len(np.unique(cl_data2_example["Cluster ID"]))):
    exemplar_idx = distance.cdist([means[i]], cl_data2_example.iloc[: , :-1]).argmin()
    print('\nCluster {}:'.format(i))
    print("  Exemplar ID: {}".format(exemplar_idx))
    display(cl_data2_example.iloc[[exemplar_idx]])
```

```
Cluster 0:
  Exemplar ID: 181
```

	Age	Income	SpendingScore	Savings	Cluster ID
181	64	86283	0.76	6852.55	0

```
Cluster 1:
  Exemplar ID: 375
```

	Age	Income	SpendingScore	Savings	Cluster ID
375	84	27384	0.31	16734.67	1

```
Cluster 2:
  Exemplar ID: 130
```

	Age	Income	SpendingScore	Savings	Cluster ID
130	34	107255	0.33	15130.60	2

1.3 Model Comparison

Exploratory Data Analysis:

The dataset contains 505 observations with four numeric features. The features are easy to understand, which indicate a customer's age, income, spending score in the store (0 to 1), and savings. The box plot for each feature shows that the observed values are within a reasonable range and have no outliers, so no customer is drastically different from others. In addition, there are interesting observations from some features. Firstly, the customers' ages fall into three categories: young (under 40 years old), middle to early senior (between 50 to 70 years old), and seniors (more than 80 years old). Secondly, the spending scores are in two categories, representing two extreme spending groups: 0.2 to 0.4 and 0.7 to 0.9.

The patterns are apparent when looking at pair plots between the features. For instance, in the age-income chart, the young people (below age 40) tend to have a high income, and seniors (above age 80) have a meager income, and some have a very high income. Also, in the savings and spending score chart, customers with low spending scores tend to have high savings, and those with high spending scores tend to have low savings. Moreover, the correlation heatmap shows that the features reasonably relate to each other. Savings is -0.92 related to Spending Score. Age is -0.83 related to income. These indicators provide good results as the more likely customers save, the less likely they will spend, and the young customers have relatively more minor income.

Thus, the dataset shows clear groupings to certain degrees, and it would be intuitive to find out the target customers through proper cluster modeling.

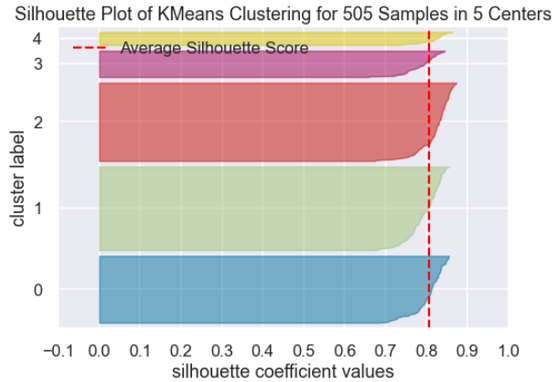
Algorithm #1

The first algorithm used for clustering is K-means. K-means defines K centroids and assigns the observations to the respective closest centroids. The model works well for "well-shaped" clusters and is easy to identify outliers. The K indicates how many clusters the model groups. As the elbow plot shows, the distortion scores diminishe between 3 – 5 clusters. Hence, three models, K=3, K=4, and K=5, have been developed.

There are two indicators for choosing the best model: Silhouette scores and cluster interpretation. The silhouette plot shows how close each observation in one cluster is to other observations in other clusters, and the Silhouette coefficient shows if the clusters are compact or overlapping. A good score closes to +1 means the clusters are very compact. In addition, the objective of cluster interpretation is if the groups make sense to Uncle Steve, and we can extract meaningful customer information from each cluster.

The Silhouette scores increase as K increases, from 0.7 to 0.8 as K increases from 3 to 5. The more clusters modeled, the more specific groups classified. However, in the K = 5 Model, cluster 4 overlaps with cluster 0, representing the customers with similar low spending scores and high savings. The overlapping makes the cluster 4 exemplar pulls the sample from cluster 0. From the Silhouette plot, the cluster 4 and cluster 3 are relatively thin comparing with other clusters. These two groups might be broken down and grouped with other clusters.

In [80]:
```
#The Silhouette Plot for the K = 5 Kmeans model.
plot_model(kmeans5, plot = 'silhouette')
```



Moreover, the clusters are well classified in the K = 4 Model. Each customer group is well presented. The K=4 Model can be improved more in K=3 Model. The difference between the two models is that both cluster 0 and cluster 3 expose similar characteristics, which have high income and high spending scores but just represent different age buckets as about 24 years old and 41 years old, respectively. From Uncle Steve's perspective, these two groups indeed represent the same young and high purchasing power customers. The model can be more simplified and directional. Thus, the K = 3 Model would be best for all three models.

In [81]:
```
print ("K =3 Model Feature Means:")
print("Age  Income  Spending_Score  Savings")
meansK3
```

```
K =3 Model Feature Means:
Age  Income  Spending_Score  Savings
```

Out[81]:
```
array([[    41.59, 107695.98,      0.27,  14937.27],
       [    51.31,  85873.44,      0.8 ,   6213.05],
       [    87.78,  27866.1 ,      0.33,  16659.26]])
```

In [82]:
```
print ("K =4 Model Feature Means:")
print("Age  Income  Spending_Score  Savings")
meansK4
```

```
K =4 Model Feature Means:
Age  Income  Spending_Score  Savings
```

Out[82]:
```
array([[    41.59, 107695.98,      0.27,  14937.27],
       [    59.96,  72448.06,      0.77,   6889.97],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    24.18, 128029.12,      0.9 ,   4087.52]])
```

In [83]:
```
print ("K =5 Model Feature Means:")
print("Age  Income  Spending_Score  Savings")
meansK5
```

```
K =5 Model Feature Means:
Age  Income  Spending_Score  Savings
```

Out[83]:
```
array([[    32.78, 105265.81,      0.31,  14962.78],
       [    59.96,  72448.06,      0.77,   6889.97],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    24.18, 128029.12,      0.9 ,   4087.52],
       [    86.  , 119944.04,      0.07,  14808.68]])
```

Algorithm #2

The second algorithm used for clustering is Agglomerative Clustering. In this algorithm, each observation is a cluster, and then it will merge with the closest clusters at a reasonable distance. The process is demonstrated in a mall dendrogram. During the experiment, various linkage methods have been tested (W ard, Single, Average) along with different cluster numbers (3 to 5, as the mall dendrogram suggests). The dendrograms indicate that these observations indeed can be grouped with a few clusters, about 3 to 6 clusters. The Silhouette scores are close with each other varying linkage methods and cluster numbers = 4 and 5, about 0.72 to 0.8. The three different mall dendrograms suggest that only several noticeable clusters will group all the observations, which is consistent with our findings from EDA that the observations are grouped already to certain degrees. Finally, the 3-cluster Model with linkage method Ward is used to generate outputs. The result is good, but cluster 0 and cluster 2 complement each other. The result indicates the two customer groups who are in the same age bucket, but one is for low spending score and high savings, and one is for high spending score and low savings. The result is good but might not be helpful for Uncle Steve.

In [84]:
```
print ("Number of Clusters = 3 Model Feature Means:")
print("Age  Income  Spending_Score  Savings")
meansH3
```

```
Number of Clusters = 3 Model Feature Means:
Age  Income  Spending_Score  Savings
```

Out[84]:
```
array([[    51.31,  85873.44,      0.8 ,   6213.05],
       [    87.78,  27866.1 ,      0.33,  16659.26],
       [    41.59, 107695.98,      0.27,  14937.27]])
```

In summary, both algorithms have generated good, precise results and are easy to interpret. The first algorithm is built with Pycaret for low code efficiency and automation and is very fast. The second algorithm is developed with Sklearn packages and is relatively slow. I prefer the result from Algorithm K=3 to be the best model, as the experiment output might be more interpretable for Uncle Steve to understand his customers better.

1.4 Personas

Below is the fundamental statistics for each customer group from the K=3 Kmeans model.

The average age for the first group is 41 years old and the average income is about $107K. The spendingscore is from 0.27 to 0.4 and has a lot of saving, more than $15K.

The average age for the second group is 51 years old and the average income is about $85K. The spendingscore is from 0.8 to 1 and has low savings, roughly $6K.

The average age for the second group is 87 years old and the average income is about $27K. The spendingscore is from 0.33 to 0.47 and has a lot of savings, more than $16K.

In [85]:
```
#The fundamental stats for each cluster.
col_names = kmean_results3_example.columns
pd.set_option("display.precision", 2)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
def stats_to_df(d):
    tmp_df = pd.DataFrame(columns=col_names)
    tmp_df.loc[0] = d.minmax[0]
    tmp_df.loc[1] = d.mean
    tmp_df.loc[2] = d.minmax[1]
    tmp_df.loc[3] = d.variance
    tmp_df.loc[4] = d.skewness
    tmp_df.loc[5] = d.kurtosis
    tmp_df.index = ['Min', 'Mean', 'Max', 'Variance', 'Skewness', 'Kurtosis']
    return tmp_df.T
print('All Data:')
print('Number of Instances: {}'.format(kmean_results3_example.shape[0]))
d = stats.describe(kmean_results3_example, axis=0)
display(stats_to_df(d))
for i in range(0, len(np.unique(kmean_results3_example["Cluster ID"]))):
  temp = kmean_results3_example[kmean_results3_example["Cluster ID"] == i]
  d = stats.describe(temp, axis=0)
  print('\nCluster {}:'.format(i))
  print('Number of Instances: {}'.format(d.nobs))
  display(stats_to_df(d))
```

```
All Data:
Number of Instances: 505
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	59.02	97.00	582.74	-0.06	-1.42
Income	12000.00	75513.29	142000.00	1295490447.35	-0.14	-1.25
SpendingScore	0.00	0.51	1.00	0.07	0.25	-1.42
Savings	0.00	11862.46	20000.00	24494870.20	-0.36	-1.45
Cluster ID	0.00	0.99	2.00	0.59	0.01	-1.31

```
Cluster 0:
Number of Instances: 151
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	22.00	41.59	93.00	408.07	1.70	1.14
Income	89598.00	107695.98	128596.00	64681181.27	0.42	-0.21
SpendingScore	0.00	0.27	0.41	0.01	-1.26	0.56
Savings	12207.53	14937.27	17968.55	1157752.50	0.16	-0.10
Cluster ID	0.00	0.00	0.00	0.00	0.00	-3.00

```
Cluster 1:
Number of Instances: 207
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	17.00	51.31	68.00	247.44	-1.13	-0.46
Income	56321.00	85873.44	142000.00	605890544.89	1.09	-0.51
SpendingScore	0.66	0.80	1.00	0.00	0.58	-0.34
Savings	0.00	6213.05	10547.78	2672678.10	-0.72	1.04
Cluster ID	1.00	1.00	1.00	0.00	0.00	-3.00

```
Cluster 2:
Number of Instances: 147
```

	Min	Mean	Max	Variance	Skewness	Kurtosis
Age	79.00	87.78	97.00	12.31	-0.01	-0.47
Income	12000.00	27866.10	46977.00	41587092.13	0.13	0.17
SpendingScore	0.20	0.33	0.47	0.00	0.31	0.48
Savings	13470.97	16659.26	20000.00	1401501.24	-0.15	0.03
Cluster ID	2.00	2.00	2.00	0.00	0.00	-3.00

In [86]:
```
kmean_results3_example.head()
```

Out[86]:

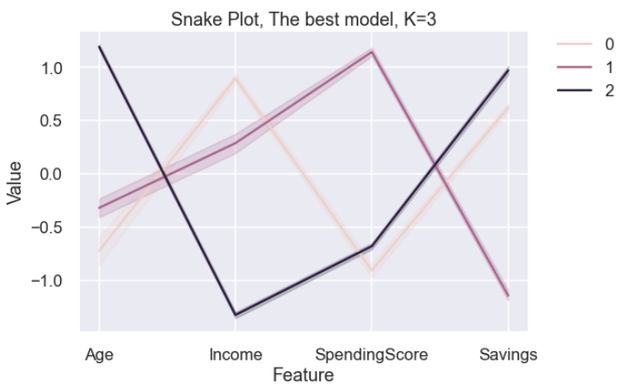
	Age	Income	SpendingScore	Savings	Cluster ID
0	58	77769	0.79	6559.83	1
1	59	81799	0.79	5417.66	1
2	62	74751	0.70	9258.99	1
3	59	74373	0.77	7346.33	1
4	87	17760	0.35	16869.51	2

Below is the SnakePlot from the K = 3 Kmeans model. The plot shows that Cluster 1 (Customer Group 2) has all the loyal customers for Uncle Steve. These customers have high spending scores and relatively high income.

In addition, the cluster 0 (Customer Group 1) has the most income, but they don't spend too much at Uncle Steve's store.

Finally, the cluster 2 (Customer Group 2) are seniros. These customers don't have a lot of income but have a lot of savings.They don't spend too much at Uncle Steve's store either.

In [87]:
```
#The Snakeplot.
#The plot shows the feature importance for each cluster.
kmean3_snake = kmean_results3_example.copy()
scaler = StandardScaler()
scaler.fit(kmean3_snake[['Age']])
kmean3_snake['Age'] = scaler.transform(kmean3_snake[['Age']])

scaler.fit(kmean3_snake[['Income']])
kmean3_snake['Income'] = scaler.transform(kmean3_snake[['Income']])

scaler.fit(kmean3_snake[['SpendingScore']])
kmean3_snake['SpendingScore'] = scaler.transform(kmean3_snake[['SpendingScore']])

scaler.fit(kmean3_snake[['Savings']])
kmean3_snake['Savings'] = scaler.transform(kmean3_snake[['Savings']])

kmean3_snake_df_melt = pd.melt(kmean3_snake,
    id_vars=["Cluster ID"],
    value_vars=['Age', 'Income', 'SpendingScore', 'Savings'],
    var_name='Feature',
    value_name='Value')

plt.title('Snake Plot, The best model, K=3');
sns.set(style="darkgrid")
sns.set_context("talk")
sns.lineplot(x="Feature", y="Value", hue='Cluster ID', data=kmean3_snake_df_melt, legend="full");
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
```



Below is the cluster means and the exemplar from Kmeans model K = 3 from algorithm #1. The result is evident and very easy for Unclue Steve to understand the customer segmentation for his store.

In [88]:
```
print ("K =3 Model Feature Means:")
print("Age  Income  Spending_Score  Savings")
meansK3
```

```
K =3 Model Feature Means:
Age  Income  Spending_Score  Savings
```

Out[88]:
```
array([[    41.59, 107695.98,      0.27,  14937.27],
       [    51.31,  85873.44,      0.8 ,   6213.05],
       [    87.78,  27866.1 ,      0.33,  16659.26]])
```

In [89]:
```
#Exemplars for K = 3 Kmeans model
for i in range(0, len(np.unique(kmean_results3_example["Cluster ID"]))):
    exemplar_idx = distance.cdist([meansK3[i]], kmean_results3_example.iloc[: , :-1]).argmin()
    print('\nCluster {}:'.format(i))
    print("  Exemplar ID: {}".format(exemplar_idx))
    display(kmean_results3_example.iloc[[exemplar_idx]])
```

```
Cluster 0:
  Exemplar ID: 130
```

	Age	Income	SpendingScore	Savings	Cluster ID
130	34	107255	0.33	15130.60	0

```
Cluster 1:
  Exemplar ID: 181
```

	Age	Income	SpendingScore	Savings	Cluster ID
181	64	86283	0.76	6852.55	1

```
Cluster 2:
```

Exemplar ID: 375

	Age	Income	SpendingScore	Savings	Cluster ID
375	84	27384	0.31	16734.67	2

Customer Group 1 (Cluster 0): This group of customers is young (about 40 years old) and has high incomes and savings, but their spending score is deficient. If Uncle Steve wants to target these customers, he should consider selling products more interested to young people.

Customer Group 2 (Cluster 1): This group of customers are middle-aged (about 50 years old) and have high incomes and high spending scores but low savings. What Uncle Steve is selling now is more interested in these customers. These customers are very loyal to Uncle Steve, and he should try keeping them.

Customer Group 3 (Cluster 2): This group of customers are seniors (about 87 years old) and have low incomes, low spending scores but high savings. These customers are probably retirees and don't have much purchasing willingness. Uncle Steve would need to do more market research to see what goods these customers like.

In conclusion, Uncle Steve needs to have more goods that can attract young customers. Since they are below 40 years old, they might interest in more fashion jewelries. These people have incomes and savings, so a big potential to expand customer base. The second group are the loyal customers, they are happy with what Uncle Steve has now, but their money is limited. They have already spent a lot and don't have savings for more purchases. These customers should be maintained properly as foundation. Lastly, the thrid group are seniors. They don't work much but they have savings. Uncle Steve needs to conduct more market research and see what kind of jewelries they like more. For instance, authentic or classic nicklaces and bracelets that could maintain values which they could buy and give to their inheirs in the future.

Question 2: Uncle Steve's Fine Foods

Instructions

Uncle Steve runs a small, local grocery store in Ontario. The store sells all the normal food staples (e.g., bread, milk, cheese, eggs, more cheese, fruits, vegatables, meat, fish, waffles, ice cream, pasta, cereals, drinks), personal care products (e.g., toothpaste, shampoo, hair goo), medicine, and cakes. There's even a little section with flowers and greeting cards! Normal people shop here, and buy normal things in the normal way.

Business is OK but Uncle Steve wants more. He's thus on the hunt for customer insights. Given your success at the jewelry store, he has asked you to help him out.

He has given you a few years' worth of customer transactions, i.e., sets of items that customers have purchased. You have applied an association rules learning algorithm (like Apriori) to the data, and the algorithm has generated a large set of association rules of the form `{X} -> {Y}`, where `{X}` and `{Y}` are item-sets.

Now comes a thought experiment. For each of the following scenarios, state what one of the discovered association rules might be that would meet the stated condition. (Just make up the rule, using your human experience and intuition.) Also, describe whether and why each rule would be considered interesting or uninteresting for Uncle Steve (i.e., is this insight new to him? Would he be able to use it somehow?).

Keep each answer to 600 characters or less (including spaces).

To get those brain juices going, an example condition and answer is provided below:

> Condition: A rule that has high support.
>
> Answer: The rule `{milk} -> {bread}` would have high support, since milk and bread are household staples and a high percentage of transactions would include both `{milk}` and `{bread}`. Uncle Steve would likely not find this rule interesting, because these items are so common, he would have surely already noticed that so many transactions contain them.

Marking

Your responses will be marked as follows:

- *Correctness*. Rule meets the specificed condition, and seems plausible in an Ontario grocery store.
- *Justification of interestness*. Response clearly describes whether and why the rule would be considered interesting to Uncle Steve.

Tips

- There is no actual data for this question. This question is just a thought exercise. You need to use your intuition, creatitivty, and understanding of the real world. I assume you are familiar with what happens inside of normal grocery stores. We are not using actual data and you do not need to create/generate/find any data. I repeat: there is no data for this question.
- The reason this question is having you do a thought experiment, rather than writing and running code to find actual association rules on an actual dataset, is because writing code to find association rules is actually pretty easy. But using your brain to come up with rules that meet certain criteria, on the other hand, is a true test of whether you understand how the algorithm works, what support and confidence mean, and the applicability of rules. The question uses the grocery store context because most, if not all, students should be familiar from personal experience.

2.1: A rule that might have high support and high confidence.

Answer:

Academic Background: A Support in association rule mining indicates how likely a consumer would buy certain products together in total transactions. In a two-product space, it measures as S(X->Y) = # of transactions with X and Y / # of total transactions. A support S% indicates how much S% of transactions include X and Y products. A Confidence in association rule mining indicates how likely a consumer would purchase some products included in some other product transactions. In a two-product space, it measures as S(X->Y) = # of transactions with X and Y / # of transactions with X. A confidence C% indicates how much C% of X transactions that contains Y.

{Breyer/Chapman's Gelato Ice Cream} -> {Waffles}

Breyer and Chapman's gelato ice creams and waffles are very popular food in our daily lives (who doesn't love Breyer and Chapman's ice cream!). Waffles are good for eating alone or with dressing such as chocolate dips. These two products give a high support S% as people's favourite desserts and snacks. In addition, it would also be very easy to find a lot of transactions that contain gelato ice creams that would also contain waffles, supporting high confidence C%. Many creative food artists like to make fancy desserts for the family and kids. In Uncle Steve's store, a customer would very likely find waffles beside the fridges where ice creams keep. For Uncle Steve to increase the sales for the combination, he could put some interesting desert recipe-tip posters on the fridges or the waffle shelf to promote fancy ice cream desserts.

2.2: A rule that might have reasonably high support but low confidence.

Answer:

{Coke/Pepsi} -> {Cetaphil Body Lotion}

Soft drinks like Coke and Pepsi and cleaning products like Cetaphil body lotion are common and popular products in a grocery store. When people go for groceries, they will buy drinks, meat, vegetables, and body lotions for their daily lives. Soft drinks like Pepsi consume very fast, but cleaning products like body lotions are relatively slower. The combination provides a reasonably high support S% because people shop for their daily lives. However, there is no direct relationship between soft drinks and cleaning products. People need these two products but do not necessarily need them simultaneously, resulting in the two products having a very minimal relationship and in a low confidence C%. Uncle Steve may make a sales flyer covering that living products like drinks, food, and body lotions are all on sale this week, and he could promote buying one kind of product (soft drinks like Coke/Pepsi) and get a coupon discount for the other product (body lotions). Bundling them together doesn't make sense to consumers.

2.3: A rule that might have low support and low confidence.

Answer:

{Butter} -> {Soy Sauce}

Butter and soy sauce are very common and popular products in a grocery store. Each product consumes very fast in people's daily lives, but very rare that people buy them together. In this case, butter is usually an essential ingredient in food like sandwiches, bread, stir-fry dishes like butter chicken, etc. Soy Sauce is, on the other extreme that widely consumed in Eastern Asian food like sushi, noodles, tofu, etc. Each is very easy to find in transactions, but together don't make much well-known food. For example, Asian families make a lot of stir-fry food with soy sauce, but they rarely put butter along with it. Butter acts like an emulsifier that gives you a thick and rich taste, but soy sauce adds texture to dishes. Hence, you will find each product included in many transactions, but people rarely buy them together, resulting in a low support S% and confidence C%. Uncle Steve should put butter in sections that make good use, such as bread, sausage, etc. and put soy sauce on shelves that work fine with other similar sauces like oyster sauce, chili sauce, etc. Each product indeed targets different customers.

2.4: A rule that might have low support and high confidence.

Answer:

{Disposable Lighters} -> {Excel/Trident gums}

Disposable lighters and gums are very common products in a grocery store. People who buy disposable lighters are most likely to smoke. Considering that there are only about 12% of smokers in Ontario (tobacco and cigarettes are getting more and more expensive every year, thanks to the government), disposable lighters would not be very hot-selling products in grocery transactions for most people. People usually have a firelighter gun at home for cooking, and if people buy a disposable lighter besides smoking, the lighters would usually last very long. On the other hand, Excel/Trident gums are popular in grocery stores because people consume them very fast. With it being said, the combination provides very low support S% because of the low sales of lighters. However, the confidence C% for the combination is high because smokers who buy lighters would be more likely to purchase gums. They would consume gums to stay away from the cigarette smell for families and children (Yikes! especially wives, mothers, and children who hate the smell!). Hence, people who smoke a lot buy gums along with a lighter. Uncle Steve can even bundle them and sell them together at a discount as a reminder. A lot of smokers would appreciate Uncle Steve very much!

Question 3: Uncle Steve's Credit Union

Instructions

Uncle Steve has recently opened a new credit union in Kingston, named *Uncle Steve's Credit Union*. He plans to disrupt the local market by instaneously providing credit to customers.

The first step in Uncle Steve's master plan is to create a model to predict whether an application has *good risk* or *bad risk*. He has outsourced the creation of this model to you.

You are to create a classification model to predict whether a loan applicant has good risk or bad risk. You will use data that Uncle Steve bought from another credit union (somewhere in Europe, he thinks?) that has around 6000 instances and a number of demographics features (e.g., `Sex`, `DateOfBirth`, `Married`), loan details (e.g., `Amount`, `Purpose`), credit history (e.g., number of loans), as well as an indicator (called `BadCredit` in the dataset) as to whether that person was a bad risk.

Your tasks

To examine the effects of the various ML stages, you are to create the model several times, each time adding more sophistication, and measuring how much the model improved (or not). In particular, you will:

1. Split the data in training and testing. Don't touch the testing data again, for any reason, until step 5. We are pretending that the testing data is "future, unseen data that our model won't see until production." I'm serious, don't touch it. I'm watching you!
2. Build a baseline model - no feature engineering, no feature selection, no hyperparameter tuning (just use the default settings), nothing fancy. (You may need to do some basic feature transformations, e.g., encoding of categorical features, or dropping of features you do not think will help or do not want to deal with yet.) Measure the performance using K-fold cross validation (recommended: `sklearn.model_selection.cross_val_score`) on the training data. Use at least 5 folds, but more are better. Choose a `scoring` parameter (i.e., classification metric) that you feel is appropriate for this task. Don't use accuracy. Print the mean score of your model.
3. Add a bit of feature engineering. The `sklearn.preprocessing` module contains many useful transformations. Engineer at least three new features. They don't need to be especially ground-breaking or complicated. Dimensionality reduction techniques like `sklearn.decomposition.PCA` are fair game but not required. (If you do use dimensionality reduction techniques, it would only count as "one" new feature for the purposes of this assignment, even though I realize that PCA creates many new "features" (i.e., principal componentns).) Re-train your baseline model. Measure performance. Compare to step 1.
4. Add feature selection. The `sklearn.feature_selection` has some algorithms for you to choose from. After selecting features, re-train your model, measure performance, and compare to step 2.
5. Add hyperparameter tuning. Make reasonable choices and try to find the best (or at least, better) hyperparameters for your estimator and/or transformers. It's probably a good idea to stop using `cross_val_score` at this point and start using `sklearn.model_selection.GridSearchCV` as it is specifically built for this purpose and is more convienient to use. Measure performance and compare to step 3.
6. Finally, estimate how well your model will work in production. Use the testing data (our "future, unseen data") from step 0. Transform the data as appropriate (easy if you've built a pipeline, a little more difficult if not), use the model from step 4 to get predictions, and measure the performance. How well did we do?

Marking

Each part will be marked for:

- *Correctness*. Code clearly and fully performs the task specified.
- *Reproducibility*. Code is fully reproducible. I.e., you (and I) should be able to run this Notebook again and again, from top to bottom, and get the same results each and every time.
- *Style*. Code is organized. All parts commented with clear reasoning and rationale. No old code laying around. Code easy to follow.

Tips

- The origins of the dataset are a bit of a mystery. Assume the data set is recent (circa 2021) and up-to-date. Assume that column names are correct and accurate.
- You don't need to experiment with more than one algorithm/estimator. Just choose one (e.g., `sklearn.tree.DecisionTreeClassifier`, `sklearn.ensemble.RandomForestClassifier`, `sklearn.linear_model.LogisticRegression`, `sklearn.svm.LinearSVC`, whatever) and stick with it for this question.
- There is no minimum accuracy/precision/recall for this question. I.e., your mark will not be based on how good your model is. Rather, you mark will be based on good your process is.
- Watch out for data leakage and overfitting. In particular, be sure to `fit()` any estimators and transformers (collectively, *objects*) only to the training data, and then use the objects' `transform()` methods on both the training and testing data. Data School has a helpful video about this. Pipelines are very helpful here and make your code shorter and more robust (at the expense of making it harder to understand), and I recommend using them, but they are not required for this assignment.
- Create as many code cells as you need. In general, each cell should do one "thing."
- Don't print large volumes of output. E.g., don't do: `df.head(100)`

3.0: Load data and split

In [90]:
```
# DO NOT MODIFY THIS CELL

# First, we'll read the provided labeled training data
df3 = pd.read_csv("https://drive.google.com/uc?export=download&id=1wOhyCnvGeY4jplxI8lZ-bbYN3zLtickf")
df3.info()

from sklearn.model_selection import train_test_split

X = df3.drop('BadCredit', axis=1) #.select_dtypes(['number'])
y = df3['BadCredit']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   UserID             6000 non-null   object
 1   Sex                6000 non-null   object
 2   PreviousDefault    6000 non-null   int64 
 3   FirstName          6000 non-null   object
 4   LastName           6000 non-null   object
 5   NumberPets         6000 non-null   int64 
 6   PreviousAccounts   6000 non-null   int64 
 7   ResidenceDuration  6000 non-null   int64 
 8   Street             6000 non-null   object
 9   LicensePlate       6000 non-null   object
 10  BadCredit          6000 non-null   int64 
 11  Amount             6000 non-null   int64 
 12  Married            6000 non-null   int64 
 13  Duration           6000 non-null   int64 
 14  City               6000 non-null   object
 15  Purpose            6000 non-null   object
 16  DateOfBirth        6000 non-null   object
dtypes: int64(8), object(9)
memory usage: 797.0+ KB
```

3.1: Baseline model

The steps taken in the baseline model is:

1. Split the training set into the new training set (X_train_new, y_train_new) and the validation set (X_val,y_val).
2. Check the missing values and unique values for each feature in the training set.
3. Drop the unused features in both the X_train_new and X_val.
4. Split the feature 'DateOfBirth" into three seperate features as "Day","Month","Year" in both X_train_new and X_val.
5. Plot Histograms, QQ plots and Correlation heatmap for each feature for EDA.
6. Apply One Hot Encoding for the features 'Sex','Married','PreviousDefault','City','Purpose' in both X_train_new and X_val.
7. Decision Tree prediction with fit(X_train_new, y_train_new), and predict on (X_val).
8. Model Evaluation (Mean Accuracy, K-fold validation, and accuracy report).

In [91]:
```
X_train_new31 = X_train.copy()
```

In [92]:
```
#In this step, the X_train adds back the label "BadCredit" and split again for training set and validation set.
X_train_new31["Label"] = y_train

X_train_new31_Split = X_train_new31.drop('Label', axis=1) #.select_dtypes(['number'])
Y_new_label = X_train_new31['Label']

X_train_new, X_val, y_train_new, y_val = train_test_split(X_train_new31_Split, Y_new_label, test_size=0.2, random_state=42)
```

In [93]:
```
#Checking X_train set features and observations
X_train_new.shape
```

Out[93]:
```
(3840, 16)
```

In [94]:
```
#Checking the first few observations of the dataset.
X_train_new.head()
```

Out[94]:

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	Amount	Married	Duration	City	Purpose	DateOfBirth
3046	842-17-6386	F	0	Katrina	Brooks	1	2	2	519 Brenda Ranch	Y56-03B	4142	0	36	New Roberttown	Repair	1976-03-17
5054	013-75-3847	F	0	Christina	Lewis	1	2	5	7551 Harmon Stravenue Apt. 338	9H 0846P	3493	0	36	Lake Debra	Education	1979-12-21
2065	181-93-2677	F	0	Melissa	Lee	2	0	3	46292 Jacob Estates	HGQ 1385	3590	0	30	Lake Debra	Household	1975-11-19
4389	521-53-3941	F	0	Regina	Beltran	1	1	3	514 Wright Parkway Suite 283	MZM 4709	3446	0	24	West Michael	NewCar	1975-04-27
5292	441-27-0220	F	0	Jacqueline	Wise	0	0	0	020 Hughes Island	6DC P54	3940	1	30	Lake Debra	Household	1971-12-30

In [95]:
```
#Checking the missing values in the training set. The training set is clean.
Missing_Count = X_train_new.isnull().sum()
Total_Count = X_train_new.count()
Missing_Percent = Missing_Count/Total_Count
Missing_Data = pd.concat([Missing_Count,Missing_Percent], axis=1, keys = ["Missing Count","Missing Percent"]).sort_values('Missing Percent', ascending=False)
print(Missing_Data)
```

```
                   Missing Count  Missing Percent
UserID                         0             0.00
Sex                            0             0.00
PreviousDefault                0             0.00
FirstName                      0             0.00
LastName                       0             0.00
NumberPets                     0             0.00
PreviousAccounts               0             0.00
ResidenceDuration              0             0.00
Street                         0             0.00
LicensePlate                   0             0.00
```

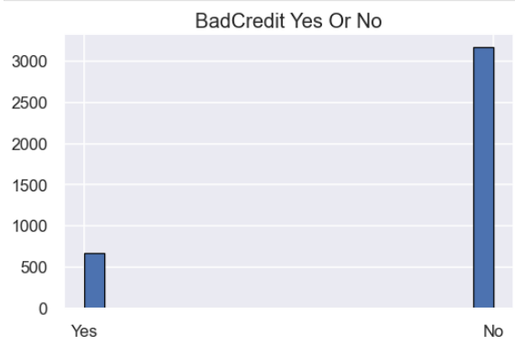
```
Amount                  0            0.00
Married                 0            0.00
Duration                0            0.00
City                    0            0.00
Purpose                 0            0.00
DateOfBirth             0            0.00
```

In [96]:
```
#Checking the unique values in each feature.
X_train_new.nunique()
```

Out[96]:
```
UserID               3840
Sex                     2
PreviousDefault         2
FirstName             536
LastName              863
NumberPets              3
PreviousAccounts        7
ResidenceDuration      10
Street               3840
LicensePlate         3839
Amount               1535
Married                 2
Duration                6
City                   20
Purpose                 8
DateOfBirth          2998
dtype: int64
```

In [97]:
```
#Histogram
def plot_hist(feature, title):
    plt.figure(figsize=(8, 5));
    plt.hist(feature, bins=20, edgecolor='black', linewidth=1.2);
    plt.title(title, fontsize=20);
    #ax.tick_params(axis='both', which='major', labelsize=18);
    plt.grid(True);
```

In [98]:
```
#Histogram for the training output.
#Most clients have BadCredit "No".
y_train_hist = pd.DataFrame(y_train_new)
y_train_hist['Label'].replace([0, 1], ['No', 'Yes'], inplace=True)
plot_hist(y_train_hist['Label'], "BadCredit Yes Or No")
```



In [99]:
```
#In the baseline model, we will drop the following features:
#UserID: This another observaiton identifier as the dataframe index for a customer, so dropping the redundant feature.
#FirstName and LastName: In the baseline model, they have the same meaning as UserID.
#The feature will be dropped for the basedline model, but will consider to use in 3.2 feature engineering section.
#Street: In the training set, there are 4800 unique observations. In the baseline model, we will just use one-hot encoding for some categorical features,
#but this feature has too many observations. will drop here and use again in 3.2 feature engineering section.
#LicensePlate: Same reason as Street, will drop for simplicity in the baseline model and reuse in 3.2 feature engineering section.
X_train_new = X_train_new.drop(['UserID','FirstName','LastName','Street','LicensePlate'], axis=1)
X_val = X_val.drop(['UserID','FirstName','LastName','Street','LicensePlate'], axis=1)
```

In [100…
```
#checking the first few observations.
X_train_new.head()
```

Out[100…

	Sex	PreviousDefault	NumberPets	PreviousAccounts	ResidenceDuration	Amount	Married	Duration	City	Purpose	DateOfBirth
3046	F	0	1	2	2	4142	0	36	New Roberttown	Repair	1976-03-17
5054	F	0	1	2	5	3493	0	36	Lake Debra	Education	1979-12-21
2065	F	0	2	0	3	3590	0	30	Lake Debra	Household	1975-11-19
4389	F	0	1	1	3	3446	0	24	West Michael	NewCar	1975-04-27
5292	F	0	0	0	0	3940	1	30	Lake Debra	Household	1971-12-30

In [101…
```
#Spliting the date of birth into three seperate features for year, month, and day.
X_train_new[["Year", "Month", "Day"]] = X_train_new["DateOfBirth"].str.split("-", expand = True)
X_val[["Year", "Month", "Day"]] = X_val["DateOfBirth"].str.split("-", expand = True)
```

In [102…

```
#Converting the new features to integer variable.
X_train_new["Year"] = X_train_new["Year"].astype(int)
X_train_new["Month"] = X_train_new["Month"].astype(int)
X_train_new["Day"] = X_train_new["Day"].astype(int)

X_val["Year"] = X_val["Year"].astype(int)
X_val["Month"] = X_val["Month"].astype(int)
X_val["Day"] = X_val["Day"].astype(int)
```

In [103…

```
#Dropping the DateOfBirth feature as the original format doesn't support in modeling.
X_train_new = X_train_new.drop(['DateOfBirth'], axis=1)
X_val = X_val.drop(['DateOfBirth'], axis=1)
```

In [104…

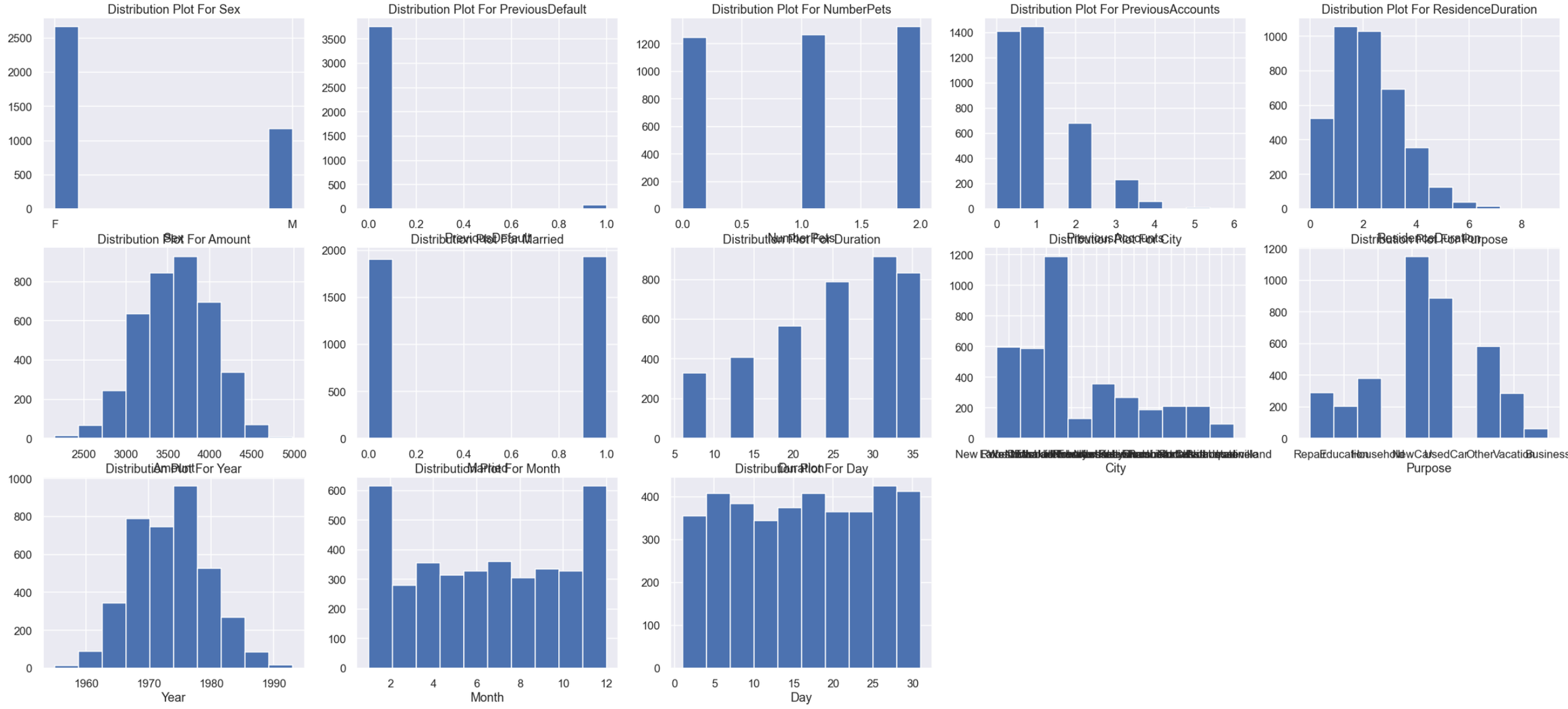
```
#Updated training set observation.
X_train_new.head()
```

Out[104…

	Sex	PreviousDefault	NumberPets	PreviousAccounts	ResidenceDuration	Amount	Married	Duration	City	Purpose	Year	Month	Day
3046	F	0	1	2	2	4142	0	36	New Roberttown	Repair	1976	3	17
5054	F	0	1	2	5	3493	0	36	Lake Debra	Education	1979	12	21
2065	F	0	2	0	3	3590	0	30	Lake Debra	Household	1975	11	19
4389	F	0	1	1	3	3446	0	24	West Michael	NewCar	1975	4	27
5292	F	0	0	0	0	3940	1	30	Lake Debra	Household	1971	12	30

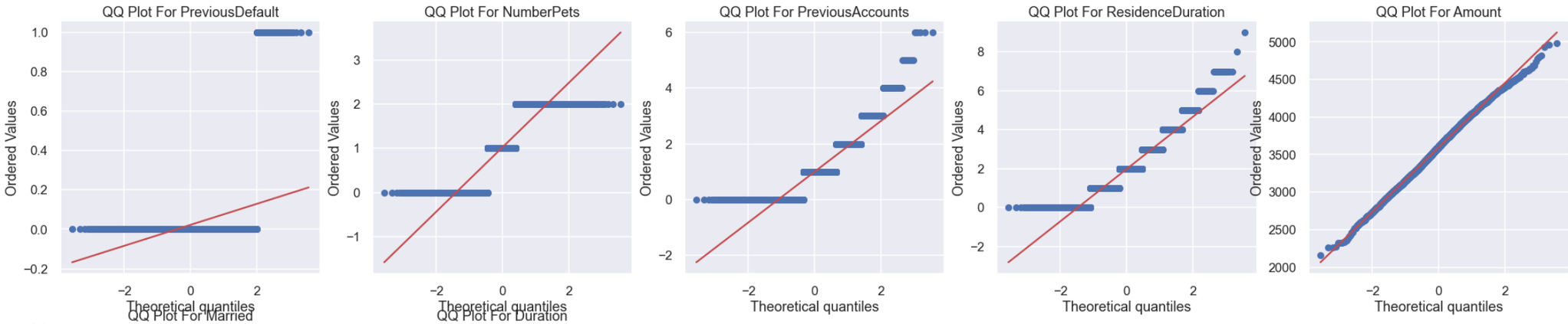
In [105…

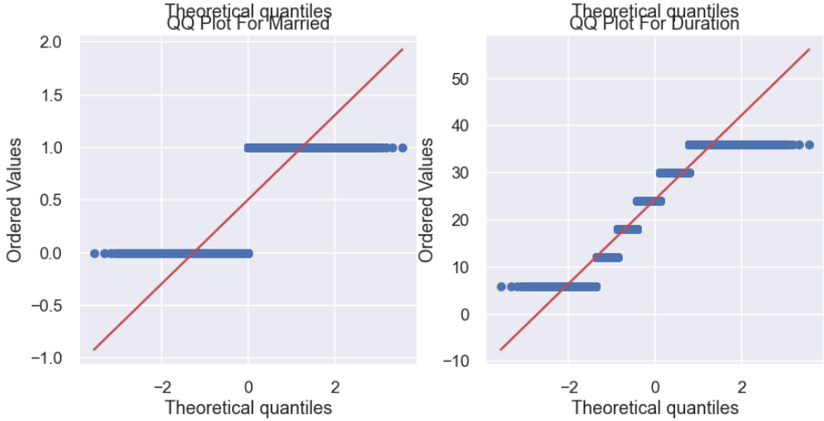
```
#Ploting Distribution Plot to see the distribution for each feature.
numeric_cols = X_train_new.columns.tolist()
fig = plt.figure(figsize=(40, 30))
for col in numeric_cols:
    ax = fig.add_subplot(5,5, numeric_cols.index(col)+1)
    ax.set_xlabel(col)
    ax.hist(X_train_new[col])
    plt.title('Distribution Plot For ' + col);
```



In [106…

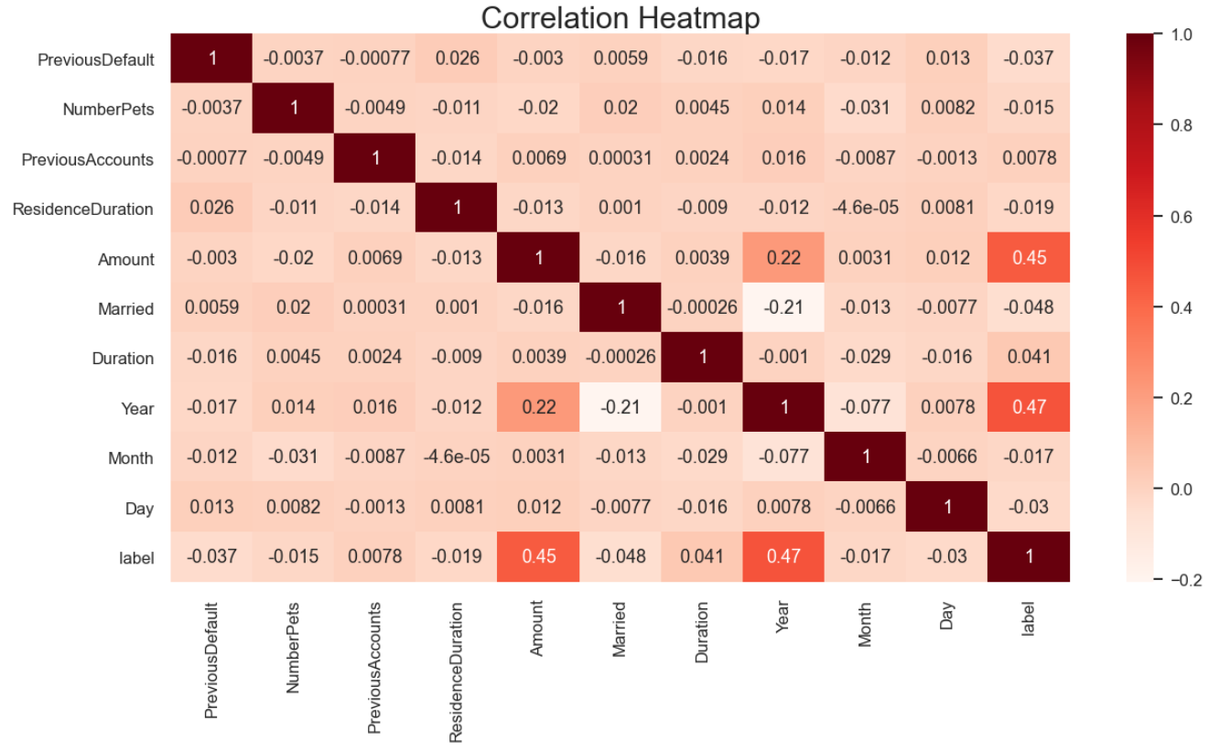
```
#The QQ plot line for each feature.
X_train_new_qq = X_train_new.select_dtypes(['int64'])
column_list = X_train_new_qq.columns.tolist()

fig = plt.figure(figsize=(35, 35))
for col in column_list:
    ax = fig.add_subplot(5, 5, column_list.index(col)+1)
    ax.set_xlabel(col)
    stats.probplot(X_train_new[col], dist="norm", plot=py)
    plt.title('QQ Plot For ' + col);
```





In [107…

```
#Built an extra dataset and added back the training label "BadCredit" back to see the correlations between these features and the label.
#The correlation feature engineering will be used in the 3.2 section.
X_corr = X_train_new.copy()
X_corr['label'] = y_train_new
#Correlation heatmap between the features.
X_combined_corr = X_corr.corr()
plt.figure(figsize=(20,10))
sns.heatmap(X_combined_corr, annot=True, cmap="Reds")
plt.title('Correlation Heatmap',fontsize=30);
plt.show()
```



In [108…
```
#One hot encoder for the binary features Sex, Married, Previous Default.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the training set.
enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
enc = enc.fit(X_train_new[['Sex','Married','PreviousDefault','City','Purpose']])
enc.transform(X_train_new[['Sex','Married','PreviousDefault','City','Purpose']])
```

Out[108…
```
array([[1., 0., 1., ..., 1., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 1., 0.],
       [0., 1., 1., ..., 1., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.]])
```

In [109…
```
#One hot encoder for the binary features Sex, Married, Previous Default.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the validation set.
enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
enc = enc.fit(X_val[['Sex','Married','PreviousDefault','City','Purpose']])
enc.transform(X_val[['Sex','Married','PreviousDefault','City','Purpose']])
```

Out[109…
```
array([[1., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.]])
```

In [110…
```
#One hot encoder for the binary features Sex, Married, Previous Default.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the training set.
_ohe_array = enc.transform(X_train_new[['Sex','Married','PreviousDefault','City','Purpose']])
_ohe_names = enc.get_feature_names()
for i in range(_ohe_array.shape[1]):
  X_train_new[_ohe_names[i]] = _ohe_array[:,i]
```

In [111…
```
#One hot encoder for the binary features Sex, Married, Previous Default.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the validation set.
_ohe_array = enc.transform(X_val[['Sex','Married','PreviousDefault','City','Purpose']])
_ohe_names = enc.get_feature_names()
for i in range(_ohe_array.shape[1]):
  X_val[_ohe_names[i]] = _ohe_array[:,i]
```

In [112…
```
#Dropping the original features as we have already encoded them.
X_train_new = X_train_new.drop(['Sex','Married','PreviousDefault','City','Purpose'], axis=1)
X_val = X_val.drop(['Sex','Married','PreviousDefault','City','Purpose'], axis=1)
```

In [113…
```
#Random Forest modeling and predict with X_validation set.
clf31 = DecisionTreeClassifier(random_state=42)
clf31.fit(X_train_new, y_train_new)
y_pred_dt = clf31.predict(X_val)
```

In [114…
```
#Print out the confusion matrix against the y_validation set.
confusion_matrix(y_val, y_pred_dt)
```

Out[114…
```
array([[727,  76],
       [ 75,  82]], dtype=int64)
```

In [115…
```
#Concatenating X_train and Y_train data for k-fold validation.
X_train_complete = pd.concat([X_train_new, X_val])
y_train_complete = pd.concat([y_train_new, y_val])
```

In [116…
```
#Estimating mean accuracy of the model with K fold = 15.
scores31 = cross_val_score(clf31, X_train_complete, y_train_complete, cv=15, scoring="accuracy")
print("Mean Accuracy: {:.4f}".format(np.mean(scores31)))
```

```
Mean Accuracy: 0.8394
```

In [117…
```
#Printing the accuracy report
class_names31 = [str(x) for x in clf31.classes_]
print(classification_report(y_val, y_pred_dt, target_names=class_names31))
```

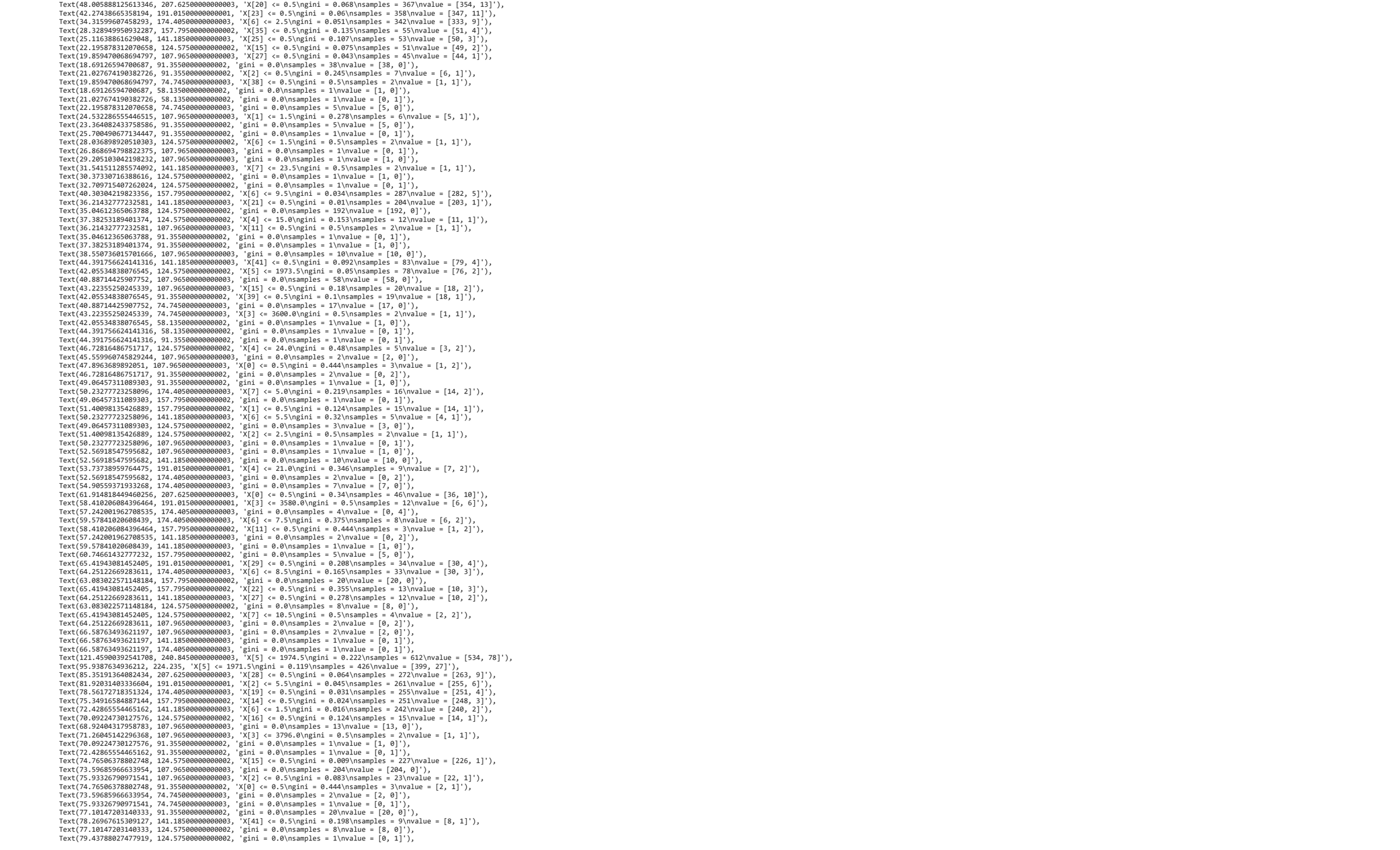
```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       803
           1       0.52      0.52      0.52       157

    accuracy                           0.84       960
   macro avg       0.71      0.71      0.71       960
weighted avg       0.84      0.84      0.84       960
```

In [118…
```
#Measuring the performance of the model prediction against validation result.
print("Accuracy = {:.2f}".format(accuracy_score(y_val, y_pred_dt)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_val, y_pred_dt)))
print("F1 Score = {:.2f}".format(f1_score(y_val, y_pred_dt)))
print("Log Loss = {:.2f}".format(log_loss(y_val, y_pred_dt)))
```

```
Accuracy = 0.84
Kappa = 0.43
F1 Score = 0.52
Log Loss = 5.43
```

In [119…
```
#Checking the selection rules at each node.
from sklearn import tree
tree.plot_tree(clf31)
```

Out[119…
```
[Text(257.28583476447494, 290.675, 'X[5] <= 1978.5\ngini = 0.287\nsamples = 3840\nvalue = [3173, 667]'),
 Text(144.52476079489696, 274.065, 'X[3] <= 3944.5\ngini = 0.165\nsamples = 3115\nvalue = [2832, 283]'),
 Text(77.29883464180568, 257.45500000000004, 'X[3] <= 3693.5\ngini = 0.081\nsamples = 2556\nvalue = [2448, 108]'),
 Text(33.13866535819431, 240.84500000000003, 'X[3] <= 3542.5\ngini = 0.03\nsamples = 1944\nvalue = [1914, 30]'),
 Text(11.316977428851816, 224.235, 'X[3] <= 3368.5\ngini = 0.009\nsamples = 1531\nvalue = [1524, 7]'),
 Text(10.148773307163886, 207.62500000000003, 'gini = 0.0\nsamples = 1069\nvalue = [1069, 0]'),
 Text(12.485181550539744, 207.62500000000003, 'X[3] <= 3369.5\ngini = 0.03\nsamples = 462\nvalue = [455, 7]'),
 Text(8.469479882237488, 191.01500000000001, 'X[0] <= 1.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(7.301275760549558, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(9.637684003925417, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(16.500883218842002, 191.01500000000001, 'X[5] <= 1977.5\ngini = 0.026\nsamples = 460\nvalue = [454, 6]'),
 Text(11.974092247301275, 174.40500000000003, 'X[34] <= 0.5\ngini = 0.018\nsamples = 442\nvalue = [438, 4]'),
 Text(8.76153091265947, 157.79500000000002, 'X[7] <= 30.5\ngini = 0.014\nsamples = 434\nvalue = [431, 3]'),
 Text(5.8410206084396465, 141.18500000000003, 'X[21] <= 0.5\ngini = 0.009\nsamples = 422\nvalue = [420, 2]'),
 Text(3.504612365063788, 124.57500000000002, 'X[6] <= 1.5\ngini = 0.005\nsamples = 398\nvalue = [397, 1]'),
 Text(2.3364082433758586, 107.96500000000003, 'X[15] <= 0.5\ngini = 0.067\nsamples = 29\nvalue = [28, 1]'),
 Text(1.168204121679293, 91.35500000000002, 'gini = 0.0\nsamples = 26\nvalue = [26, 0]'),
 Text(3.504612365063788, 91.35500000000002, 'X[11] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(2.3364082433758586, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(4.672816486751717, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(4.672816486751717, 107.96500000000003, 'gini = 0.0\nsamples = 369\nvalue = [369, 0]'),
 Text(8.177428851815506, 124.57500000000002, 'X[5] <= 1976.5\ngini = 0.08\nsamples = 24\nvalue = [23, 1]'),
 Text(7.009224730127576, 107.96500000000003, 'gini = 0.0\nsamples = 21\nvalue = [21, 0]'),
 Text(9.345632973503434, 107.96500000000003, 'X[0] <= 1.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(8.177428851815506, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(10.513837095191363, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(11.682041216879293, 141.18500000000003, 'X[26] <= 0.5\ngini = 0.153\nsamples = 12\nvalue = [11, 1]'),
 Text(10.513837095191363, 124.57500000000002, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
 Text(12.850245338567223, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(15.18665358194308, 157.79500000000002, 'X[0] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
 Text(14.018449460255152, 141.18500000000003, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(16.354857703631012, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(21.027674190382726, 174.40500000000003, 'X[23] <= 0.5\ngini = 0.198\nsamples = 18\nvalue = [16, 2]'),
 Text(19.859470068694797, 157.79500000000002, 'X[3] <= 3395.5\ngini = 0.111\nsamples = 17\nvalue = [16, 1]'),
 Text(18.69126594700687, 141.18500000000003, 'X[3] <= 3381.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(17.52306182531894, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(19.859470068694797, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(21.027674190382726, 141.18500000000003, 'gini = 0.0\nsamples = 15\nvalue = [15, 0]'),
 Text(22.195878312070658, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(54.9603532875368, 224.235, 'X[5] <= 1976.5\ngini = 0.105\nsamples = 413\nvalue = [390, 23]'),
```


Text(48.005888125613346, 207.62500000000003, 'X[20] <= 0.5\ngini = 0.068\nsamples = 367\nvalue = [354, 13]'),
Text(42.274386655358194, 191.01500000000001, 'X[23] <= 0.5\ngini = 0.06\nsamples = 358\nvalue = [347, 11]'),
Text(34.31599607458293, 174.40500000000003, 'X[6] <= 2.5\ngini = 0.051\nsamples = 342\nvalue = [333, 9]'),
Text(28.328949950932287, 157.79500000000002, 'X[35] <= 0.5\ngini = 0.135\nsamples = 55\nvalue = [51, 4]'),
Text(25.11638861629048, 141.18500000000003, 'X[25] <= 0.5\ngini = 0.107\nsamples = 53\nvalue = [50, 3]'),
Text(22.195878312070658, 124.57500000000002, 'X[15] <= 0.5\ngini = 0.075\nsamples = 51\nvalue = [49, 2]'),
Text(19.859470068694797, 107.96500000000003, 'X[27] <= 0.5\ngini = 0.043\nsamples = 45\nvalue = [44, 1]'),
Text(18.69126594700687, 91.35500000000002, 'gini = 0.0\nsamples = 38\nvalue = [38, 0]'),
Text(21.027674190382726, 91.35500000000002, 'X[2] <= 0.5\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
Text(19.859470068694797, 74.74500000000003, 'X[38] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(18.69126594700687, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(21.027674190382726, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(22.195878312070658, 74.74500000000003, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(24.532286555446515, 107.96500000000003, 'X[1] <= 1.5\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(23.364082433758586, 91.35500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(25.700490677134447, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(28.036898920510303, 124.57500000000002, 'X[6] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(26.868694798822375, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(29.205103042198232, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(31.541511285574092, 141.18500000000003, 'X[7] <= 23.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(30.37330716388616, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(32.709715407262024, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(40.30304219823356, 157.79500000000002, 'X[6] <= 9.5\ngini = 0.034\nsamples = 287\nvalue = [282, 5]'),
Text(36.21432777232581, 141.18500000000003, 'X[21] <= 0.5\ngini = 0.01\nsamples = 204\nvalue = [203, 1]'),
Text(35.04612365063788, 124.57500000000002, 'gini = 0.0\nsamples = 192\nvalue = [192, 0]'),
Text(37.38253189401374, 124.57500000000002, 'X[4] <= 15.0\ngini = 0.153\nsamples = 12\nvalue = [11, 1]'),
Text(36.21432777232581, 107.96500000000003, 'X[11] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(35.04612365063788, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(37.38253189401374, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(38.550736015701666, 107.96500000000003, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
Text(44.391756624141316, 141.18500000000003, 'X[41] <= 0.5\ngini = 0.092\nsamples = 83\nvalue = [79, 4]'),
Text(42.05534838076545, 124.57500000000002, 'X[5] <= 1973.5\ngini = 0.05\nsamples = 78\nvalue = [76, 2]'),
Text(40.88714425907752, 107.96500000000003, 'gini = 0.0\nsamples = 58\nvalue = [58, 0]'),
Text(43.22355250245339, 107.96500000000003, 'X[15] <= 0.5\ngini = 0.18\nsamples = 20\nvalue = [18, 2]'),
Text(42.05534838076545, 91.35500000000002, 'X[39] <= 0.5\ngini = 0.1\nsamples = 19\nvalue = [18, 1]'),
Text(40.88714425907752, 74.74500000000003, 'gini = 0.0\nsamples = 17\nvalue = [17, 0]'),
Text(43.22355250245339, 74.74500000000003, 'X[3] <= 3600.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(42.05534838076545, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(44.391756624141316, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(44.391756624141316, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(46.72816486751717, 124.57500000000002, 'X[4] <= 24.0\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
Text(45.559960745829244, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(47.8963689892051, 107.96500000000003, 'X[0] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(46.72816486751717, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(49.064573111089303, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(50.23277723258096, 174.40500000000003, 'X[7] <= 5.0\ngini = 0.219\nsamples = 16\nvalue = [14, 2]'),
Text(49.06457311089303, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(51.40098135426889, 157.79500000000002, 'X[1] <= 0.5\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
Text(50.23277723258096, 141.18500000000003, 'X[6] <= 5.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(49.06457311089303, 124.57500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(51.40098135426889, 124.57500000000002, 'X[2] <= 2.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(50.23277723258096, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(52.56918547595682, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(52.56918547595682, 141.18500000000003, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
Text(53.73738959764475, 191.01500000000001, 'X[4] <= 21.0\ngini = 0.346\nsamples = 9\nvalue = [7, 2]'),
Text(52.56918547595682, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(54.90559371933268, 174.40500000000003, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(61.914818449460256, 207.62500000000003, 'X[0] <= 0.5\ngini = 0.34\nsamples = 46\nvalue = [36, 10]'),
Text(58.410206084396464, 191.01500000000001, 'X[3] <= 3580.0\ngini = 0.5\nsamples = 12\nvalue = [6, 6]'),
Text(57.242001962708535, 174.40500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(59.57841020608439, 174.40500000000003, 'X[6] <= 7.5\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(58.410206084396464, 157.79500000000002, 'X[11] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(57.242001962708535, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(59.57841020608439, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(60.74661432777232, 157.79500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(65.41943081452405, 191.01500000000001, 'X[29] <= 0.5\ngini = 0.208\nsamples = 34\nvalue = [30, 4]'),
Text(64.25122669283611, 174.40500000000003, 'X[6] <= 8.5\ngini = 0.165\nsamples = 33\nvalue = [30, 3]'),
Text(63.083022571148184, 157.79500000000002, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
Text(65.41943081452405, 157.79500000000002, 'X[22] <= 0.5\ngini = 0.355\nsamples = 13\nvalue = [10, 3]'),
Text(64.25122669283611, 141.18500000000003, 'X[27] <= 0.5\ngini = 0.278\nsamples = 12\nvalue = [10, 2]'),
Text(63.083022571148184, 124.57500000000002, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(65.41943081452405, 124.57500000000002, 'X[7] <= 10.5\ngini = 0.5\nsamples = 4\nvalue = [2, 2]'),
Text(64.25122669283611, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(66.58763493621197, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(66.58763493621197, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(66.58763493621197, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(121.45900392541708, 240.84500000000003, 'X[5] <= 1974.5\ngini = 0.222\nsamples = 612\nvalue = [534, 78]'),
Text(95.9387634936212, 224.235, 'X[5] <= 1971.5\ngini = 0.119\nsamples = 426\nvalue = [399, 27]'),
Text(85.35191364082434, 207.62500000000003, 'X[28] <= 0.5\ngini = 0.064\nsamples = 272\nvalue = [263, 9]'),
Text(81.92031403336604, 191.01500000000001, 'X[2] <= 5.5\ngini = 0.045\nsamples = 261\nvalue = [255, 6]'),
Text(78.56172718351324, 174.40500000000003, 'X[19] <= 0.5\ngini = 0.031\nsamples = 255\nvalue = [251, 4]'),
Text(75.34916584887144, 157.79500000000002, 'X[14] <= 0.5\ngini = 0.024\nsamples = 251\nvalue = [248, 3]'),
Text(72.42865554465162, 141.18500000000003, 'X[6] <= 1.5\ngini = 0.016\nsamples = 242\nvalue = [240, 2]'),
Text(70.09224730127576, 124.57500000000002, 'X[16] <= 0.5\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
Text(68.92404317958783, 107.96500000000003, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(71.26045142296368, 107.96500000000003, 'X[3] <= 3796.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(70.09224730127576, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(72.42865554465162, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(74.76506378802748, 124.57500000000002, 'X[15] <= 0.5\ngini = 0.009\nsamples = 227\nvalue = [226, 1]'),
Text(73.59685966633954, 107.96500000000003, 'gini = 0.0\nsamples = 204\nvalue = [204, 0]'),
Text(75.93326790971541, 107.96500000000003, 'X[2] <= 0.5\ngini = 0.083\nsamples = 23\nvalue = [22, 1]'),
Text(74.76506378802748, 91.35500000000002, 'X[0] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(73.59685966633954, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(75.93326790971541, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(77.10147203140333, 91.35500000000002, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
Text(78.26967615309127, 141.18500000000003, 'X[41] <= 0.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(77.10147203140333, 124.57500000000002, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(79.43788027477919, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),

Text(81.77428851815505, 157.79500000000002, 'X[7] <= 20.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(80.60608439646712, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(82.94249263984298, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(85.27890088321884, 174.40500000000003, 'X[27] <= 0.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(84.1106967615309, 157.79500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(86.44710500490677, 157.79500000000002, 'X[1] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(85.27890088321884, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(87.6153091265947, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(88.7835132482863, 191.01500000000001, 'X[8] <= 0.5\ngini = 0.397\nsamples = 11\nvalue = [8, 3]'),
Text(87.6153091265947, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(89.95171736997055, 174.40500000000003, 'X[7] <= 4.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(88.7835132482863, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(91.11992149165849, 157.79500000000002, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(106.52561334641806, 207.62500000000003, 'X[17] <= 0.5\ngini = 0.206\nsamples = 154\nvalue = [136, 18]'),
Text(102.65593719332679, 191.01500000000001, 'X[35] <= 0.5\ngini = 0.183\nsamples = 147\nvalue = [132, 15]'),
Text(98.42119725220805, 174.40500000000003, 'X[4] <= 33.0\ngini = 0.158\nsamples = 139\nvalue = [127, 12]'),
Text(93.45632973503434, 157.79500000000002, 'X[26] <= 0.5\ngini = 0.119\nsamples = 110\nvalue = [103, 7]'),
Text(90.53581943081453, 141.18500000000003, 'X[7] <= 12.5\ngini = 0.077\nsamples = 100\nvalue = [96, 4]'),
Text(89.36761530912659, 124.57500000000002, 'X[30] <= 0.5\ngini = 0.176\nsamples = 41\nvalue = [37, 4]'),
Text(88.19941118743866, 107.96500000000003, 'X[5] <= 1972.5\ngini = 0.139\nsamples = 40\nvalue = [37, 3]'),
Text(85.8630029440628, 91.35500000000002, 'X[1] <= 1.5\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(84.69479882237488, 74.74500000000003, 'X[40] <= 0.5\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
Text(83.52659470068694, 58.13500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(85.8630029440628, 58.13500000000002, 'X[6] <= 3.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(84.69479882237488, 41.52500000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(87.03120706575073, 41.52500000000034, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(87.03120706575073, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(90.53581943081453, 91.35500000000002, 'X[41] <= 0.5\ngini = 0.061\nsamples = 32\nvalue = [31, 1]'),
Text(89.36761530912659, 74.74500000000003, 'gini = 0.0\nsamples = 29\nvalue = [29, 0]'),
Text(91.70402355250245, 74.74500000000003, 'X[4] <= 24.0\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(90.53581943081453, 58.13500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(92.87222767419038, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(90.53581943081453, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(91.70402355250245, 124.57500000000002, 'gini = 0.0\nsamples = 59\nvalue = [59, 0]'),
Text(96.37684003925416, 141.18500000000003, 'X[3] <= 3833.0\ngini = 0.42\nsamples = 10\nvalue = [7, 3]'),
Text(95.20863591756624, 124.57500000000002, 'X[40] <= 0.5\ngini = 0.5\nsamples = 6\nvalue = [3, 3]'),
Text(94.0404317958783, 107.96500000000003, 'X[7] <= 24.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(92.87222767419038, 91.35500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(95.20863591756624, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(96.37684003925416, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(97.5450441609421, 124.57500000000002, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(103.38606476938175, 157.79500000000002, 'X[16] <= 0.5\ngini = 0.285\nsamples = 29\nvalue = [24, 5]'),
Text(102.21786064769381, 141.18500000000003, 'X[6] <= 10.5\ngini = 0.198\nsamples = 27\nvalue = [24, 3]'),
Text(99.88145240431795, 124.57500000000002, 'X[21] <= 0.5\ngini = 0.083\nsamples = 23\nvalue = [22, 1]'),
Text(98.71324828263003, 107.96500000000003, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
Text(101.04965652600589, 107.96500000000003, 'X[5] <= 1972.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(99.88145240431795, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(102.21786064769381, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(104.55426889106967, 124.57500000000002, 'X[0] <= 1.5\ngini = 0.5\nsamples = 4\nvalue = [2, 2]'),
Text(103.38606476938175, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(105.7224730127576, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(104.55426889106967, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(106.89067713444553, 174.40500000000003, 'X[6] <= 5.0\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(105.7224730127576, 157.79500000000002, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(108.05888125613346, 157.79500000000002, 'X[25] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(106.89067713444553, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(109.2270853778214, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(110.39528949950932, 191.01500000000001, 'X[8] <= 0.5\ngini = 0.49\nsamples = 7\nvalue = [4, 3]'),
Text(109.2270853778214, 174.40500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(111.56349362119725, 174.40500000000003, 'X[7] <= 25.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(110.39528949950932, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(112.73169774288517, 157.79500000000002, 'X[11] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(111.56349362119725, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(113.89990186457311, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(146.97924435721296, 224.235, 'X[35] <= 0.5\ngini = 0.398\nsamples = 186\nvalue = [135, 51]'),
Text(139.9928361138371, 207.62500000000003, 'X[7] <= 12.5\ngini = 0.382\nsamples = 179\nvalue = [133, 46]'),
Text(129.52463199214915, 191.01500000000001, 'X[7] <= 11.5\ngini = 0.45\nsamples = 73\nvalue = [48, 25]'),
Text(128.35642787046123, 174.40500000000003, 'X[16] <= 0.5\ngini = 0.423\nsamples = 69\nvalue = [48, 21]'),
Text(120.61707556427871, 157.79500000000002, 'X[5] <= 1976.5\ngini = 0.379\nsamples = 59\nvalue = [44, 15]'),
Text(116.23631010794897, 141.18500000000003, 'X[30] <= 0.5\ngini = 0.251\nsamples = 34\nvalue = [29, 5]'),
Text(115.06810598626103, 124.57500000000002, 'X[22] <= 0.5\ngini = 0.213\nsamples = 33\nvalue = [29, 4]'),
Text(113.89990186457311, 107.96500000000003, 'X[4] <= 21.0\ngini = 0.17\nsamples = 32\nvalue = [29, 3]'),
Text(112.73169774288517, 91.35500000000002, 'X[2] <= 0.5\ngini = 0.375\nsamples = 12\nvalue = [9, 3]'),
Text(110.39528949950932, 74.74500000000003, 'X[4] <= 9.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(109.2270853778214, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(111.56349362119725, 58.13500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(115.06810598626103, 74.74500000000003, 'X[20] <= 0.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(113.89990186457311, 58.13500000000002, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(116.23631010794897, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(115.06810598626103, 91.35500000000002, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
Text(116.23631010794897, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(117.4045142296369, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(124.99784102060843, 141.18500000000003, 'X[10] <= 0.5\ngini = 0.48\nsamples = 25\nvalue = [15, 10]'),
Text(120.90912659470068, 124.57500000000002, 'X[4] <= 21.0\ngini = 0.49\nsamples = 14\nvalue = [6, 8]'),
Text(118.57271835132482, 107.96500000000003, 'X[41] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(117.4045142296369, 91.35500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(119.74092247301276, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(123.24553483807654, 107.96500000000003, 'X[0] <= 0.5\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(122.07733071638862, 91.35500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(124.41373895976447, 91.35500000000002, 'X[4] <= 27.0\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
Text(123.24553483807654, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(125.5819430814524, 74.74500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(129.0865554465162, 124.57500000000002, 'X[3] <= 3742.0\ngini = 0.298\nsamples = 11\nvalue = [9, 2]'),
Text(127.91835132482827, 107.96500000000003, 'X[22] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(126.75014720314033, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(129.0865554465162, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(130.25475956820412, 107.96500000000003, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(136.09578017664376, 157.79500000000002, 'X[7] <= 9.5\ngini = 0.48\nsamples = 10\nvalue = [4, 6]'),

Text(134.92757605495584, 141.18500000000003, 'X[38] <= 0.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(133.75937193326791, 124.57500000000002, 'X[1] <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(132.59116781157996, 107.96500000000003, 'X[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(131.42296368989204, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(133.75937193326791, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(134.92757605495584, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(136.09578017664376, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(137.2639842983317, 141.18500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(130.6928361138371, 174.40500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(150.46104023552502, 191.01500000000001, 'X[6] <= 11.5\ngini = 0.318\nsamples = 106\nvalue = [85, 21]'),
Text(146.06202158979391, 174.40500000000003, 'X[24] <= 0.5\ngini = 0.284\nsamples = 99\nvalue = [82, 17]'),
Text(144.893817468106, 157.79500000000002, 'X[2] <= 0.5\ngini = 0.273\nsamples = 98\nvalue = [82, 16]'),
Text(139.60039254170755, 141.18500000000003, 'X[5] <= 1977.5\ngini = 0.473\nsamples = 13\nvalue = [8, 5]'),
Text(138.43218842001963, 124.57500000000002, 'X[31] <= 0.5\ngini = 0.32\nsamples = 10\nvalue = [8, 2]'),
Text(137.2639842983317, 107.96500000000003, 'X[4] <= 27.0\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(136.09578017664376, 91.35500000000002, 'X[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(134.92757605495584, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(137.2639842983317, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(138.43218842001963, 91.35500000000002, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(139.60039254170755, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(140.76859666339547, 124.57500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(150.1872423945044, 141.18500000000003, 'X[3] <= 3730.5\ngini = 0.225\nsamples = 85\nvalue = [74, 11]'),
Text(143.10500490677134, 124.57500000000002, 'X[7] <= 24.5\ngini = 0.42\nsamples = 20\nvalue = [14, 6]'),
Text(141.93680078508342, 107.96500000000003, 'X[4] <= 33.0\ngini = 0.49\nsamples = 14\nvalue = [8, 6]'),
Text(140.76859666339547, 91.35500000000002, 'X[8] <= 0.5\ngini = 0.444\nsamples = 12\nvalue = [8, 4]'),
Text(139.60039254170755, 74.74500000000003, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(141.93680078508342, 74.74500000000003, 'X[3] <= 3709.0\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(140.76859666339547, 58.13500000000002, 'X[16] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(139.60039254170755, 41.525000000000034, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(141.93680078508342, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(143.10500490677134, 58.13500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(143.10500490677134, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(144.27320902845926, 107.96500000000003, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(157.26947988223748, 124.57500000000002, 'X[23] <= 0.5\ngini = 0.142\nsamples = 65\nvalue = [60, 5]'),
Text(153.91089303238468, 107.96500000000003, 'X[28] <= 0.5\ngini = 0.119\nsamples = 63\nvalue = [59, 4]'),
Text(150.69833169774287, 91.35500000000002, 'X[15] <= 0.5\ngini = 0.094\nsamples = 61\nvalue = [58, 3]'),
Text(147.77782139352306, 74.74500000000003, 'X[7] <= 28.5\ngini = 0.065\nsamples = 59\nvalue = [57, 2]'),
Text(145.4414131501472, 58.13500000000002, 'X[38] <= 0.5\ngini = 0.035\nsamples = 56\nvalue = [55, 1]'),
Text(144.27320902845926, 41.525000000000034, 'gini = 0.0\nsamples = 50\nvalue = [50, 0]'),
Text(146.60961727183513, 41.525000000000034, 'X[7] <= 25.0\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(145.4414131501472, 24.91500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(147.77782139352306, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(150.11422963689893, 58.13500000000002, 'X[16] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(148.94602551521098, 41.525000000000034, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(151.28243375858685, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(153.6188420019627, 74.74500000000003, 'X[36] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(152.45063788027477, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(154.78704612365064, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(157.12345436702648, 91.35500000000002, 'X[2] <= 2.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(155.95525024533856, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(158.29165848871443, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(160.62806673209028, 107.96500000000003, 'X[4] <= 24.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(159.45986261040235, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(161.7962708537782, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(147.23022571148184, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(154.86005888125612, 174.40500000000003, 'X[1] <= 0.5\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(153.6918547595682, 157.79500000000002, 'X[5] <= 1977.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(152.52365063788028, 141.18500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(154.86005888125612, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(156.02826300294407, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(153.9656526005888, 207.62500000000003, 'X[3] <= 3749.5\ngini = 0.408\nsamples = 7\nvalue = [2, 5]'),
Text(152.7974484789009, 191.01500000000001, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(155.13385672227673, 191.01500000000001, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(211.7506869479882, 257.45500000000004, 'X[5] <= 1971.5\ngini = 0.43\nsamples = 559\nvalue = [384, 175]'),
Text(169.9736997055937, 240.84500000000003, 'X[34] <= 0.5\ngini = 0.158\nsamples = 231\nvalue = [211, 20]'),
Text(168.80549558390578, 224.235, 'X[3] <= 3955.0\ngini = 0.152\nsamples = 230\nvalue = [211, 19]'),
Text(159.60588812561335, 207.62500000000003, 'X[7] <= 9.0\ngini = 0.444\nsamples = 9\nvalue = [6, 3]'),
Text(158.43768400392543, 191.01500000000001, 'X[6] <= 4.0\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(157.26947988223748, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(159.60588812561335, 174.40500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(160.77409224730127, 191.01500000000001, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(178.00510304219821, 207.62500000000003, 'X[18] <= 0.5\ngini = 0.134\nsamples = 221\nvalue = [205, 16]'),
Text(169.53562315996075, 191.01500000000001, 'X[7] <= 17.5\ngini = 0.121\nsamples = 216\nvalue = [202, 14]'),
Text(161.9422963689892, 174.40500000000003, 'X[3] <= 4176.0\ngini = 0.044\nsamples = 132\nvalue = [129, 3]'),
Text(160.77409224730127, 157.79500000000002, 'gini = 0.0\nsamples = 88\nvalue = [88, 0]'),
Text(163.11050049067714, 157.79500000000002, 'X[3] <= 4178.5\ngini = 0.127\nsamples = 44\nvalue = [41, 3]'),
Text(160.77409224730127, 141.18500000000003, 'X[0] <= 1.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(159.60588812561335, 124.57500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(161.9422963689892, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(165.44690873405298, 141.18500000000003, 'X[40] <= 0.5\ngini = 0.048\nsamples = 41\nvalue = [40, 1]'),
Text(164.27870461236506, 124.57500000000002, 'gini = 0.0\nsamples = 36\nvalue = [36, 0]'),
Text(166.6151128557409, 124.57500000000002, 'X[27] <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(165.44690873405298, 107.96500000000003, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(167.78331697742885, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(177.1289499509228, 174.40500000000003, 'X[6] <= 9.5\ngini = 0.228\nsamples = 84\nvalue = [73, 11]'),
Text(172.45613346418057, 157.79500000000002, 'X[5] <= 1970.5\ngini = 0.126\nsamples = 59\nvalue = [55, 4]'),
Text(170.1197252208047, 141.18500000000003, 'X[4] <= 33.0\ngini = 0.075\nsamples = 51\nvalue = [49, 2]'),
Text(168.95152109911677, 124.57500000000002, 'gini = 0.0\nsamples = 35\nvalue = [35, 0]'),
Text(171.28792934249265, 124.57500000000002, 'X[16] <= 0.5\ngini = 0.219\nsamples = 16\nvalue = [14, 2]'),
Text(170.1197252208047, 107.96500000000003, 'X[5] <= 1969.5\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
Text(168.95152109911677, 91.35500000000002, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(171.28792934249265, 91.35500000000002, 'X[10] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(170.1197252208047, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(172.45613346418057, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(172.45613346418057, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(174.7925417075564, 141.18500000000003, 'X[6] <= 1.5\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(173.6243375858685, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(175.96074582924436, 124.57500000000002, 'X[4] <= 18.0\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),

Text(174.7925417075564, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(177.12894995093228, 107.96500000000003, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(181.801766437684, 157.79500000000002, 'X[7] <= 19.5\ngini = 0.403\nsamples = 25\nvalue = [18, 7]'),
Text(180.63356231599607, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(182.96997055937192, 141.18500000000003, 'X[3] <= 4174.0\ngini = 0.34\nsamples = 23\nvalue = [18, 5]'),
Text(180.63356231599607, 124.57500000000002, 'X[38] <= 0.5\ngini = 0.494\nsamples = 9\nvalue = [5, 4]'),
Text(179.46535819430815, 107.96500000000003, 'X[1] <= 1.5\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(178.2971540726202, 91.35500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(180.63356231599607, 91.35500000000002, 'X[14] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(179.46535819430815, 74.74500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(181.801766437684, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(181.801766437684, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(185.3063788027478, 124.57500000000002, 'X[7] <= 28.0\ngini = 0.133\nsamples = 14\nvalue = [13, 1]'),
Text(184.13817468105987, 107.96500000000003, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(186.4745829244357, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(186.4745829244357, 191.01500000000001, 'X[7] <= 16.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
Text(185.3063788027478, 174.40500000000003, 'X[2] <= 2.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(184.13817468105987, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(186.4745829244357, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(187.64278704612366, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(171.14190382728165, 224.235, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(253.52767419038273, 240.84500000000003, 'X[5] <= 1975.5\ngini = 0.498\nsamples = 328\nvalue = [173, 155]'),
Text(232.67340529931303, 224.235, 'X[4] <= 9.0\ngini = 0.477\nsamples = 201\nvalue = [122, 79]'),
Text(230.3369970559372, 207.62500000000003, 'X[2] <= 4.5\ngini = 0.117\nsamples = 16\nvalue = [15, 1]'),
Text(229.16879293424927, 191.01500000000001, 'gini = 0.0\nsamples = 15\nvalue = [15, 0]'),
Text(231.5052011776251, 191.01500000000001, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(235.0098135426889, 207.62500000000003, 'X[30] <= 0.5\ngini = 0.488\nsamples = 185\nvalue = [107, 78]'),
Text(233.84160942100098, 191.01500000000001, 'X[7] <= 26.5\ngini = 0.482\nsamples = 180\nvalue = [107, 73]'),
Text(219.29381746810597, 174.40500000000003, 'X[3] <= 4231.0\ngini = 0.494\nsamples = 151\nvalue = [84, 67]'),
Text(203.63258096172717, 157.79500000000002, 'X[1] <= 0.5\ngini = 0.478\nsamples = 109\nvalue = [66, 43]'),
Text(195.82021589793916, 141.18500000000003, 'X[38] <= 0.5\ngini = 0.381\nsamples = 43\nvalue = [32, 11]'),
Text(192.31560353287537, 124.57500000000002, 'X[6] <= 2.5\ngini = 0.326\nsamples = 39\nvalue = [31, 8]'),
Text(188.81099116781158, 107.96500000000003, 'X[7] <= 17.5\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(187.64278704612366, 91.35500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(189.9791952894995, 91.35500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(195.82021589793916, 107.96500000000003, 'X[20] <= 0.5\ngini = 0.219\nsamples = 32\nvalue = [28, 4]'),
Text(192.31560353287537, 91.35500000000002, 'X[6] <= 9.5\ngini = 0.18\nsamples = 30\nvalue = [27, 3]'),
Text(188.81099116781158, 74.74500000000003, 'X[3] <= 4187.5\ngini = 0.091\nsamples = 21\nvalue = [20, 1]'),
Text(187.64278704612366, 58.13500000000002, 'gini = 0.0\nsamples = 18\nvalue = [18, 0]'),
Text(189.9791952894995, 58.13500000000002, 'X[5] <= 1972.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(188.81099116781158, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(191.14739941118742, 41.525000000000034, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(195.82021589793916, 74.74500000000003, 'X[16] <= 0.5\ngini = 0.346\nsamples = 9\nvalue = [7, 2]'),
Text(194.65201177625121, 58.13500000000002, 'X[15] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
Text(193.4838076545633, 41.525000000000034, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(195.82021589793916, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(196.98842001962709, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(199.32482826300293, 91.35500000000002, 'X[0] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(198.156624141315, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(200.49303238469088, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(199.32482826300293, 124.57500000000002, 'X[2] <= 3.0\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(198.156624141315, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(200.49303238469088, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(211.4449460255152, 141.18500000000003, 'X[7] <= 6.5\ngini = 0.5\nsamples = 66\nvalue = [34, 32]'),
Text(203.99764474975464, 124.57500000000002, 'X[3] <= 4119.5\ngini = 0.43\nsamples = 16\nvalue = [5, 11]'),
Text(202.82944062806672, 107.96500000000003, 'X[12] <= 0.5\ngini = 0.26\nsamples = 13\nvalue = [2, 11]'),
Text(201.6612365063788, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(203.99764474975464, 91.35500000000002, 'X[15] <= 0.5\ngini = 0.153\nsamples = 12\nvalue = [1, 11]'),
Text(202.82944062806672, 74.74500000000003, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
Text(205.1658488714426, 74.74500000000003, 'X[3] <= 4023.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(203.99764474975464, 58.13500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(206.3340529931305, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(205.1658488714426, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(218.89224730127575, 124.57500000000002, 'X[27] <= 0.5\ngini = 0.487\nsamples = 50\nvalue = [29, 21]'),
Text(215.09558390579, 107.96500000000003, 'X[21] <= 0.5\ngini = 0.422\nsamples = 33\nvalue = [23, 10]'),
Text(213.92737978410204, 91.35500000000002, 'X[5] <= 1973.5\ngini = 0.383\nsamples = 31\nvalue = [23, 8]'),
Text(209.8386653581943, 74.74500000000003, 'X[7] <= 21.0\ngini = 0.133\nsamples = 14\nvalue = [13, 1]'),
Text(208.67046123650638, 58.13500000000002, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
Text(211.00686947988223, 58.13500000000002, 'X[26] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(209.8386653581943, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(212.17507360157015, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(218.0160942100098, 74.74500000000003, 'X[9] <= 0.5\ngini = 0.484\nsamples = 17\nvalue = [10, 7]'),
Text(216.8478900883219, 58.13500000000002, 'X[7] <= 17.0\ngini = 0.486\nsamples = 12\nvalue = [5, 7]'),
Text(214.51148184494602, 41.525000000000034, 'X[24] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(213.3432777232581, 24.91500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(215.67968596663394, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(219.18429833169773, 41.525000000000034, 'X[0] <= 1.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(218.0160942100098, 24.91500000000002, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(220.35250245338565, 24.91500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(219.18429833169773, 58.13500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(216.26378802747791, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(222.68891069676152, 107.96500000000003, 'X[37] <= 0.5\ngini = 0.457\nsamples = 17\nvalue = [6, 11]'),
Text(221.5207065750736, 91.35500000000002, 'X[2] <= 0.5\ngini = 0.497\nsamples = 13\nvalue = [6, 7]'),
Text(220.35250245338565, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(222.68891069676152, 74.74500000000003, 'X[4] <= 33.0\ngini = 0.463\nsamples = 11\nvalue = [4, 7]'),
Text(221.5207065750736, 58.13500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(223.85711481844945, 58.13500000000002, 'X[0] <= 0.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(222.68891069676152, 41.525000000000034, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(225.0253189401374, 41.525000000000034, 'X[2] <= 2.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(223.85711481844945, 24.91500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(226.19352306182532, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(223.85711481844945, 91.35500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(234.95505397448477, 157.79500000000002, 'X[7] <= 13.5\ngini = 0.49\nsamples = 42\nvalue = [18, 24]'),
Text(230.86633954857703, 141.18500000000003, 'X[7] <= 8.5\ngini = 0.486\nsamples = 24\nvalue = [14, 10]'),
Text(228.52993130520116, 124.57500000000002, 'X[2] <= 1.5\ngini = 0.48\nsamples = 15\nvalue = [6, 9]'),
Text(227.36172718351324, 107.96500000000003, 'X[1] <= 0.5\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(226.19352306182532, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(228.52993130520116, 91.35500000000002, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),

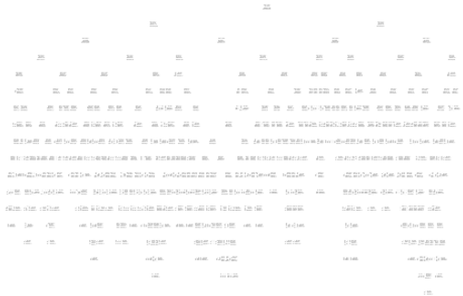
Text(229.6981354268891, 107.96500000000003, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(233.2027477919529, 124.57500000000002, 'X[6] <= 1.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(232.03454367026495, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(234.37095191364082, 107.96500000000003, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(239.04376840039254, 141.18500000000003, 'X[5] <= 1973.5\ngini = 0.346\nsamples = 18\nvalue = [4, 14]'),
Text(237.87556427870462, 124.57500000000002, 'X[3] <= 4361.5\ngini = 0.494\nsamples = 9\nvalue = [4, 5]'),
Text(236.70736015701667, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(239.04376840039254, 107.96500000000003, 'X[6] <= 6.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(237.87556427870462, 91.35500000000002, 'X[3] <= 4450.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(236.70736015701667, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(239.04376840039254, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(240.21197252208046, 91.35500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(240.21197252208046, 124.57500000000002, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
Text(248.38940137389596, 174.40500000000003, 'X[35] <= 0.5\ngini = 0.328\nsamples = 29\nvalue = [23, 6]'),
Text(247.22119725220804, 157.79500000000002, 'X[6] <= 3.5\ngini = 0.293\nsamples = 28\nvalue = [23, 5]'),
Text(243.71658488714425, 141.18500000000003, 'X[3] <= 4059.0\ngini = 0.49\nsamples = 7\nvalue = [4, 3]'),
Text(242.54838076545633, 124.57500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(244.88478900883217, 124.57500000000002, 'X[39] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(243.71658488714425, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(246.05299313052012, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(250.72580961727184, 141.18500000000003, 'X[3] <= 4043.0\ngini = 0.172\nsamples = 21\nvalue = [19, 2]'),
Text(249.55760549558389, 124.57500000000002, 'X[3] <= 4041.5\ngini = 0.346\nsamples = 9\nvalue = [7, 2]'),
Text(248.38940137389596, 107.96500000000003, 'X[16] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
Text(247.22119725220804, 91.35500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(249.55760549558389, 91.35500000000002, 'X[7] <= 28.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(248.38940137389596, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(250.72580961727184, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(250.72580961727184, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(251.89401373895976, 124.57500000000002, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
Text(249.55760549558389, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(236.17801766437682, 191.01500000000001, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(274.3819430814524, 224.235, 'X[4] <= 21.0\ngini = 0.481\nsamples = 127\nvalue = [51, 76]'),
Text(261.2396467124632, 207.62500000000003, 'X[0] <= 0.5\ngini = 0.493\nsamples = 41\nvalue = [23, 18]'),
Text(255.39862610402355, 191.01500000000001, 'X[5] <= 1976.5\ngini = 0.444\nsamples = 15\nvalue = [5, 10]'),
Text(254.23042198233563, 174.40500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(256.5668302257115, 174.40500000000003, 'X[3] <= 4172.5\ngini = 0.496\nsamples = 11\nvalue = [5, 6]'),
Text(254.23042198233563, 157.79500000000002, 'X[7] <= 2.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(253.06221786064768, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(255.39862610402355, 141.18500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(258.90323846908734, 157.79500000000002, 'X[7] <= 13.0\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(257.7350343473994, 141.18500000000003, 'X[38] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(256.5668302257115, 124.57500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(258.90323846908734, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(260.07144259077523, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(267.0806673209028, 191.01500000000001, 'X[7] <= 26.0\ngini = 0.426\nsamples = 26\nvalue = [18, 8]'),
Text(264.744259077527, 174.40500000000003, 'X[35] <= 0.5\ngini = 0.278\nsamples = 18\nvalue = [15, 3]'),
Text(263.576054955839, 157.79500000000002, 'X[23] <= 0.5\ngini = 0.208\nsamples = 17\nvalue = [15, 2]'),
Text(262.40785083415113, 141.18500000000003, 'X[1] <= 2.5\ngini = 0.117\nsamples = 16\nvalue = [15, 1]'),
Text(261.2396467124632, 124.57500000000002, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(263.576054955839, 124.57500000000002, 'X[9] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(262.40785083415113, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(264.744259077527, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(264.744259077527, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(265.9124631992149, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(269.4170755642787, 174.40500000000003, 'X[3] <= 4157.5\ngini = 0.469\nsamples = 8\nvalue = [3, 5]'),
Text(268.24887144259077, 157.79500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(270.5852796859666, 157.79500000000002, 'X[2] <= 5.0\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(269.4170755642787, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(271.75348380765456, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(287.5242394504416, 207.62500000000003, 'X[3] <= 4125.0\ngini = 0.439\nsamples = 86\nvalue = [28, 58]'),
Text(280.515014720314, 191.01500000000001, 'X[1] <= 1.5\ngini = 0.5\nsamples = 39\nvalue = [19, 20]'),
Text(279.3468105986261, 174.40500000000003, 'X[3] <= 3973.0\ngini = 0.49\nsamples = 35\nvalue = [15, 20]'),
Text(275.25809617271835, 157.79500000000002, 'X[3] <= 3951.5\ngini = 0.408\nsamples = 7\nvalue = [5, 2]'),
Text(274.0898920510304, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(276.42630029440625, 141.18500000000003, 'X[29] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(275.25809617271835, 124.57500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(277.5945044160942, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(283.43552502453383, 157.79500000000002, 'X[3] <= 4111.0\ngini = 0.459\nsamples = 28\nvalue = [10, 18]'),
Text(282.2673090284594, 141.18500000000003, 'X[40] <= 0.5\ngini = 0.426\nsamples = 26\nvalue = [8, 18]'),
Text(279.93091265947004, 124.57500000000002, 'X[25] <= 0.5\ngini = 0.278\nsamples = 18\nvalue = [3, 15]'),
Text(278.76270853778215, 107.96500000000003, 'X[3] <= 4053.5\ngini = 0.208\nsamples = 17\nvalue = [2, 15]'),
Text(277.5945044160942, 91.35500000000002, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
Text(279.93091265947004, 91.35500000000002, 'X[3] <= 4075.0\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
Text(278.76270853778215, 74.74500000000003, 'X[41] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(277.5945044160942, 58.13500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(279.93091265947004, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(281.099116781158, 74.74500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(281.099116781158, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(284.6037291462218, 124.57500000000002, 'X[7] <= 18.0\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(283.43552502453383, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(285.77193326790973, 107.96500000000003, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(284.6037291462218, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(281.68321884200196, 174.40500000000003, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(294.5334641805692, 191.01500000000001, 'X[6] <= 11.5\ngini = 0.31\nsamples = 47\nvalue = [9, 38]'),
Text(291.61295387634937, 174.40500000000003, 'X[3] <= 4549.5\ngini = 0.24\nsamples = 43\nvalue = [6, 37]'),
Text(289.2765456329735, 157.79500000000002, 'X[0] <= 1.5\ngini = 0.18\nsamples = 40\nvalue = [4, 36]'),
Text(288.1083415112856, 141.18500000000003, 'gini = 0.0\nsamples = 24\nvalue = [0, 24]'),
Text(290.4447497546614, 141.18500000000003, 'X[21] <= 0.5\ngini = 0.375\nsamples = 16\nvalue = [4, 12]'),
Text(289.2765456329735, 124.57500000000002, 'X[2] <= 4.5\ngini = 0.32\nsamples = 15\nvalue = [3, 12]'),
Text(288.1083415112856, 107.96500000000003, 'X[41] <= 0.5\ngini = 0.245\nsamples = 14\nvalue = [2, 12]'),
Text(285.77193326790973, 91.35500000000002, 'X[9] <= 0.5\ngini = 0.153\nsamples = 12\nvalue = [1, 11]'),
Text(284.6037291462218, 74.74500000000003, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
Text(286.9401373895976, 74.74500000000003, 'X[5] <= 1977.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(285.77193326790973, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(288.1083415112856, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(290.4447497546614, 91.35500000000002, 'X[8] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(289.2765456329735, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(291.61295387634937, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),

Text(290.4447497546614, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(291.61295387634937, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(293.9493621197252, 157.79500000000002, 'X[5] <= 1977.0\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(292.78115799803726, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(295.11756624141316, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(297.4539744484789, 174.40500000000003, 'X[3] <= 4314.0\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(296.28577036310105, 157.79500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(298.62217860647695, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(370.046908734053, 274.065, 'X[3] <= 3740.5\ngini = 0.498\nsamples = 725\nvalue = [341, 384]'),
Text(324.83375858684985, 257.45500000000004, 'X[3] <= 3518.5\ngini = 0.302\nsamples = 286\nvalue = [233, 53]'),
Text(309.13601570166827, 240.84500000000003, 'X[3] <= 3397.5\ngini = 0.118\nsamples = 174\nvalue = [163, 11]'),
Text(304.4631992149166, 224.235, 'X[22] <= 0.5\ngini = 0.031\nsamples = 126\nvalue = [124, 2]'),
Text(302.12679097154074, 207.62500000000003, 'X[5] <= 1984.5\ngini = 0.016\nsamples = 123\nvalue = [122, 1]'),
Text(300.9585868498528, 191.01500000000001, 'gini = 0.0\nsamples = 114\nvalue = [114, 0]'),
Text(303.29499509322864, 191.01500000000001, 'X[10] <= 0.5\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(302.12679097154074, 174.40500000000003, 'X[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(300.9585868498528, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(303.29499509322864, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(304.4631992149166, 174.40500000000003, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(306.7996074582924, 207.62500000000003, 'X[40] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(305.6314033366045, 191.01500000000001, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(307.9678115799804, 191.01500000000001, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(313.80883218842, 224.235, 'X[3] <= 3442.5\ngini = 0.305\nsamples = 48\nvalue = [39, 9]'),
Text(311.47242394504417, 207.62500000000003, 'X[6] <= 1.5\ngini = 0.465\nsamples = 19\nvalue = [12, 7]'),
Text(310.3042198233562, 191.01500000000001, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(312.64062806673206, 191.01500000000001, 'X[20] <= 0.5\ngini = 0.375\nsamples = 16\nvalue = [12, 4]'),
Text(311.47242394504417, 174.40500000000003, 'X[25] <= 0.5\ngini = 0.32\nsamples = 15\nvalue = [12, 3]'),
Text(310.3042198233562, 157.79500000000002, 'X[3] <= 3437.5\ngini = 0.245\nsamples = 14\nvalue = [12, 2]'),
Text(309.13601570166827, 141.18500000000003, 'X[1] <= 1.5\ngini = 0.142\nsamples = 13\nvalue = [12, 1]'),
Text(307.9678115799804, 124.57500000000002, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
Text(310.3042198233562, 124.57500000000002, 'X[10] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(309.13601570166827, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(311.47242394504417, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(311.47242394504417, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(312.64062806673206, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(313.80883218842, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(316.14524043179586, 207.62500000000003, 'X[16] <= 0.5\ngini = 0.128\nsamples = 29\nvalue = [27, 2]'),
Text(314.97703631010796, 191.01500000000001, 'gini = 0.0\nsamples = 23\nvalue = [23, 0]'),
Text(317.3134445534838, 191.01500000000001, 'X[5] <= 1983.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(316.14524043179586, 174.40500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(318.48164867517175, 174.40500000000003, 'X[6] <= 6.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(317.3134445534838, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(319.64985279685965, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(340.5315014720314, 240.84500000000003, 'X[5] <= 1984.5\ngini = 0.469\nsamples = 112\nvalue = [70, 42]'),
Text(332.2080471050049, 224.235, 'X[2] <= 1.5\ngini = 0.422\nsamples = 96\nvalue = [67, 29]'),
Text(326.65907752698723, 207.62500000000003, 'X[7] <= 9.5\ngini = 0.295\nsamples = 39\nvalue = [32, 7]'),
Text(324.3226692836114, 191.01500000000001, 'X[2] <= 0.5\ngini = 0.473\nsamples = 13\nvalue = [8, 5]'),
Text(323.1544651692344, 174.40500000000003, 'X[18] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(321.9862610402355, 157.79500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(324.3226692836114, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(325.4908734052993, 174.40500000000003, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(328.9954857703631, 191.01500000000001, 'X[5] <= 1983.5\ngini = 0.142\nsamples = 26\nvalue = [24, 2]'),
Text(327.8272816486752, 174.40500000000003, 'X[41] <= 0.5\ngini = 0.077\nsamples = 25\nvalue = [24, 1]'),
Text(326.65907752698723, 157.79500000000002, 'gini = 0.0\nsamples = 22\nvalue = [22, 0]'),
Text(328.9954857703631, 157.79500000000002, 'X[9] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(327.8272816486752, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(330.163689892051, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(330.163689892051, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(337.7570166830226, 207.62500000000003, 'X[5] <= 1979.5\ngini = 0.474\nsamples = 57\nvalue = [35, 22]'),
Text(333.6683022571148, 191.01500000000001, 'X[36] <= 0.5\ngini = 0.26\nsamples = 13\nvalue = [11, 2]'),
Text(332.50009813542687, 174.40500000000003, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
Text(334.8365063788027, 174.40500000000003, 'X[7] <= 22.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(333.6683022571148, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(336.00471050049066, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(341.8457311089303, 191.01500000000001, 'X[3] <= 3726.5\ngini = 0.496\nsamples = 44\nvalue = [24, 20]'),
Text(340.6775269872424, 174.40500000000003, 'X[3] <= 3705.0\ngini = 0.499\nsamples = 42\nvalue = [22, 20]'),
Text(338.3411187438665, 157.79500000000002, 'X[22] <= 0.5\ngini = 0.491\nsamples = 37\nvalue = [21, 16]'),
Text(337.1729146221786, 141.18500000000003, 'X[4] <= 9.0\ngini = 0.48\nsamples = 35\nvalue = [21, 14]'),
Text(336.00471050049066, 124.57500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(338.3411187438665, 124.57500000000002, 'X[4] <= 21.0\ngini = 0.492\nsamples = 32\nvalue = [18, 14]'),
Text(335.4206084396467, 107.96500000000003, 'X[40] <= 0.5\ngini = 0.375\nsamples = 8\nvalue = [2, 6]'),
Text(334.2524043179588, 91.35500000000002, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
Text(336.5888125613463, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(341.2616290480864, 107.96500000000003, 'X[7] <= 21.5\ngini = 0.444\nsamples = 24\nvalue = [16, 8]'),
Text(338.9252208047105, 91.35500000000002, 'X[6] <= 3.5\ngini = 0.32\nsamples = 15\nvalue = [12, 3]'),
Text(337.7570166830226, 74.74500000000003, 'X[5] <= 1982.0\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
Text(336.5888125613463, 58.13500000000002, 'X[27] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(335.4206084396467, 41.525000000000034, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(337.7570166830226, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(338.9252208047105, 58.13500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(340.0934249263984, 74.74500000000003, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
Text(343.5980372914622, 91.35500000000002, 'X[1] <= 0.5\ngini = 0.494\nsamples = 9\nvalue = [4, 5]'),
Text(342.42983316977427, 74.74500000000003, 'X[7] <= 26.5\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(341.2616290480864, 58.13500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(343.5980372914622, 58.13500000000002, 'X[0] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(342.42983316977427, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(344.7662414131501, 41.525000000000034, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(344.7662414131501, 74.74500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(339.50932286555445, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(343.01393523061824, 157.79500000000002, 'X[10] <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(341.8457311089303, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(344.1821393523062, 141.18500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(343.01393523061824, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(348.8549558390579, 224.235, 'X[32] <= 0.5\ngini = 0.305\nsamples = 16\nvalue = [3, 13]'),
Text(347.68675171737, 207.62500000000003, 'X[1] <= 3.5\ngini = 0.231\nsamples = 15\nvalue = [2, 13]'),
Text(346.51854759568204, 191.01500000000001, 'X[35] <= 0.5\ngini = 0.133\nsamples = 14\nvalue = [1, 13]'),
Text(345.3503434739941, 174.40500000000003, 'gini = 0.0\nsamples = 13\nvalue = [0, 13]'),
Text(347.68675171737, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),

Text(348.8549558390579, 191.01500000000001, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(350.02315996074583, 207.62500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(415.2600588812561, 257.45500000000004, 'X[5] <= 1982.5\ngini = 0.371\nsamples = 439\nvalue = [108, 331]'),
Text(389.74210009813544, 240.84500000000003, 'X[3] <= 4003.5\ngini = 0.454\nsamples = 247\nvalue = [86, 161]'),
Text(372.36506378802744, 224.235, 'X[7] <= 18.5\ngini = 0.498\nsamples = 107\nvalue = [50, 57]'),
Text(362.4353287536801, 207.62500000000003, 'X[2] <= 3.5\ngini = 0.461\nsamples = 61\nvalue = [22, 39]'),
Text(355.5721295387635, 191.01500000000001, 'X[3] <= 3871.0\ngini = 0.415\nsamples = 51\nvalue = [15, 36]'),
Text(350.6072620215898, 174.40500000000003, 'X[22] <= 0.5\ngini = 0.278\nsamples = 24\nvalue = [4, 20]'),
Text(349.43905789990185, 157.79500000000002, 'X[4] <= 21.0\ngini = 0.227\nsamples = 23\nvalue = [3, 20]'),
Text(348.2708537782139, 141.18500000000003, 'X[5] <= 1980.5\ngini = 0.42\nsamples = 10\nvalue = [3, 7]'),
Text(345.93444553483806, 124.57500000000002, 'X[7] <= 11.5\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
Text(344.7662414131501, 107.96500000000003, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(347.102649656526, 107.96500000000003, 'X[11] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(345.93444553483806, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(348.2708537782139, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(350.6072620215898, 124.57500000000002, 'X[8] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(349.43905789990185, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(351.7754661432777, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(350.6072620215898, 141.18500000000003, 'gini = 0.0\nsamples = 13\nvalue = [0, 13]'),
Text(351.7754661432777, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(360.5369970559372, 174.40500000000003, 'X[3] <= 3934.0\ngini = 0.483\nsamples = 27\nvalue = [11, 16]'),
Text(356.44828263002944, 157.79500000000002, 'X[6] <= 9.5\ngini = 0.42\nsamples = 10\nvalue = [7, 3]'),
Text(354.1118743866536, 141.18500000000003, 'X[27] <= 0.5\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
Text(352.94367026496565, 124.57500000000002, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(355.2800785083415, 124.57500000000002, 'X[5] <= 1980.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(354.1118743866536, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(356.44828263002944, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(358.7846908734053, 141.18500000000003, 'X[4] <= 21.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(357.61648675171733, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(359.9528949509323, 124.57500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(364.6257114818449, 157.79500000000002, 'X[37] <= 0.5\ngini = 0.36\nsamples = 17\nvalue = [4, 13]'),
Text(363.457507360157, 141.18500000000003, 'X[7] <= 12.0\ngini = 0.48\nsamples = 10\nvalue = [4, 6]'),
Text(362.2893032384691, 124.57500000000002, 'X[3] <= 3980.0\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(361.1210991167811, 107.96500000000003, 'X[40] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(359.9528949509323, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(362.2893032384691, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(363.457507360157, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(364.6257114818449, 124.57500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(365.79391560353287, 141.18500000000003, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(369.29852796859666, 191.01500000000001, 'X[7] <= 15.5\ngini = 0.42\nsamples = 10\nvalue = [7, 3]'),
Text(368.1303238469087, 174.40500000000003, 'X[41] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
Text(366.9621197252208, 157.79500000000002, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(369.29852796859666, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(370.4667320902846, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(382.29479882237484, 207.62500000000003, 'X[4] <= 33.0\ngini = 0.476\nsamples = 46\nvalue = [28, 18]'),
Text(377.7680078508341, 191.01500000000001, 'X[15] <= 0.5\ngini = 0.401\nsamples = 36\nvalue = [26, 10]'),
Text(374.5554465161923, 174.40500000000003, 'X[2] <= 0.5\ngini = 0.342\nsamples = 32\nvalue = [25, 7]'),
Text(371.6349362119725, 157.79500000000002, 'X[41] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(370.4667320902846, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(372.80314033366045, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(377.47595682041214, 157.79500000000002, 'X[27] <= 0.5\ngini = 0.285\nsamples = 29\nvalue = [24, 5]'),
Text(375.1395485770363, 141.18500000000003, 'X[2] <= 1.5\ngini = 0.172\nsamples = 21\nvalue = [19, 2]'),
Text(373.97134445534834, 124.57500000000002, 'X[6] <= 7.0\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(372.80314033366045, 107.96500000000003, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(375.1395485770363, 107.96500000000003, 'X[41] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(373.97134445534834, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(376.30775269872424, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(376.30775269872424, 124.57500000000002, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(379.81236506378804, 141.18500000000003, 'X[2] <= 3.5\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(378.6441609421001, 124.57500000000002, 'X[37] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(377.47595682041214, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(379.81236506378804, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(380.98056918547593, 124.57500000000002, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(380.98056918547593, 174.40500000000003, 'X[11] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(379.81236506378804, 157.79500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(382.1487733071639, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(386.8215897939156, 191.01500000000001, 'X[38] <= 0.5\ngini = 0.32\nsamples = 10\nvalue = [2, 8]'),
Text(385.65338567222767, 174.40500000000003, 'X[3] <= 3834.0\ngini = 0.198\nsamples = 9\nvalue = [1, 8]'),
Text(384.4851815505397, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(386.8215897939156, 157.79500000000002, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
Text(387.9897939156035, 174.40500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(407.1191364082434, 224.235, 'X[7] <= 27.5\ngini = 0.382\nsamples = 140\nvalue = [36, 104]'),
Text(400.54798822374875, 207.62500000000003, 'X[7] <= 20.5\ngini = 0.347\nsamples = 121\nvalue = [27, 94]'),
Text(396.75132482826297, 191.01500000000001, 'X[1] <= 3.5\ngini = 0.405\nsamples = 92\nvalue = [26, 66]'),
Text(395.5831207065751, 174.40500000000003, 'X[7] <= 2.5\ngini = 0.391\nsamples = 90\nvalue = [24, 66]'),
Text(389.15799803729146, 157.79500000000002, 'X[1] <= 0.5\ngini = 0.494\nsamples = 9\nvalue = [5, 4]'),
Text(387.9897939156035, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(390.32620215897936, 141.18500000000003, 'X[27] <= 0.5\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
Text(389.15799803729146, 124.57500000000002, 'X[2] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(387.9897939156035, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(390.32620215897936, 107.96500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(391.4944062806673, 124.57500000000002, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(402.0082433758587, 157.79500000000002, 'X[27] <= 0.5\ngini = 0.359\nsamples = 81\nvalue = [19, 62]'),
Text(396.16722276741905, 141.18500000000003, 'X[40] <= 0.5\ngini = 0.27\nsamples = 56\nvalue = [9, 47]'),
Text(393.83081452404315, 124.57500000000002, 'X[11] <= 0.5\ngini = 0.157\nsamples = 35\nvalue = [3, 32]'),
Text(392.66261040235526, 107.96500000000003, 'X[6] <= 2.5\ngini = 0.337\nsamples = 14\nvalue = [3, 11]'),
Text(391.4944062806673, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(393.83081452404315, 91.35500000000002, 'X[5] <= 1979.5\ngini = 0.26\nsamples = 13\nvalue = [2, 11]'),
Text(392.66261040235526, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(394.9990186457311, 74.74500000000003, 'X[17] <= 0.5\ngini = 0.153\nsamples = 12\nvalue = [1, 11]'),
Text(393.83081452404315, 58.13500000000002, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
Text(396.16722276741905, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(394.9990186457311, 107.96500000000003, 'gini = 0.0\nsamples = 21\nvalue = [0, 21]'),
Text(398.5036310107949, 124.57500000000002, 'X[6] <= 4.5\ngini = 0.408\nsamples = 21\nvalue = [6, 15]'),
Text(397.33542688910694, 107.96500000000003, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(399.67183513248284, 107.96500000000003, 'X[5] <= 1979.5\ngini = 0.49\nsamples = 14\nvalue = [6, 8]'),
Text(398.5036310107949, 91.35500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(400.84003925417073, 91.35500000000002, 'X[3] <= 4063.5\ngini = 0.48\nsamples = 10\nvalue = [6, 4]'),

Text(399.67183513248284, 74.74500000000003, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(402.0082433758587, 74.74500000000003, 'X[3] <= 4501.0\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
Text(400.84003925417073, 58.13500000000002, 'X[7] <= 19.0\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
Text(399.67183513248284, 41.525000000000034, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(402.0082433758587, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(403.1764474975466, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(407.8492639842983, 141.18500000000003, 'X[4] <= 21.0\ngini = 0.48\nsamples = 25\nvalue = [10, 15]'),
Text(406.68105986261037, 124.57500000000002, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(409.01746810598627, 124.57500000000002, 'X[7] <= 17.5\ngini = 0.494\nsamples = 18\nvalue = [10, 8]'),
Text(407.8492639842983, 107.96500000000003, 'X[6] <= 6.5\ngini = 0.498\nsamples = 15\nvalue = [7, 8]'),
Text(405.5128557409225, 91.35500000000002, 'X[0] <= 1.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(404.3446516192345, 74.74500000000003, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(406.68105986261037, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(410.18567222767416, 91.35500000000002, 'X[3] <= 4019.0\ngini = 0.42\nsamples = 10\nvalue = [3, 7]'),
Text(409.01746810598627, 74.74500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(411.3538763493621, 74.74500000000003, 'X[35] <= 0.5\ngini = 0.346\nsamples = 9\nvalue = [2, 7]'),
Text(410.18567222767416, 58.13500000000002, 'X[39] <= 0.5\ngini = 0.219\nsamples = 8\nvalue = [1, 7]'),
Text(409.01746810598627, 41.525000000000034, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
Text(411.3538763493621, 41.525000000000034, 'X[0] <= 1.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(410.18567222767416, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(412.52208047105006, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(412.52208047105006, 58.13500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(410.18567222767416, 107.96500000000003, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(397.9195289499509, 174.40500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(404.3446516192345, 191.01500000000001, 'X[17] <= 0.5\ngini = 0.067\nsamples = 29\nvalue = [1, 28]'),
Text(403.1764474975466, 174.40500000000003, 'gini = 0.0\nsamples = 26\nvalue = [0, 26]'),
Text(405.5128557409225, 174.40500000000003, 'X[40] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(404.3446516192345, 157.79500000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(406.68105986261037, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(413.69028459273795, 207.62500000000003, 'X[5] <= 1981.5\ngini = 0.499\nsamples = 19\nvalue = [9, 10]'),
Text(412.52208047105006, 191.01500000000001, 'X[27] <= 0.5\ngini = 0.492\nsamples = 16\nvalue = [9, 7]'),
Text(410.18567222767416, 174.40500000000003, 'X[7] <= 29.5\ngini = 0.397\nsamples = 11\nvalue = [8, 3]'),
Text(409.01746810598627, 157.79500000000002, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(411.3538763493621, 157.79500000000002, 'X[4] <= 21.0\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
Text(410.18567222767416, 141.18500000000003, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(412.52208047105006, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(414.8584887144259, 174.40500000000003, 'X[39] <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(413.69028459273795, 157.79500000000002, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(416.02669283611385, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(414.8584887144259, 191.01500000000001, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(440.7780176643768, 240.84500000000003, 'X[2] <= 6.5\ngini = 0.203\nsamples = 192\nvalue = [22, 170]'),
Text(439.6098135426889, 224.235, 'X[18] <= 0.5\ngini = 0.188\nsamples = 190\nvalue = [20, 170]'),
Text(435.15603532875366, 207.62500000000003, 'X[7] <= 29.5\ngini = 0.175\nsamples = 186\nvalue = [18, 168]'),
Text(429.7530912659047, 191.01500000000001, 'X[3] <= 4092.5\ngini = 0.154\nsamples = 178\nvalue = [15, 163]'),
Text(425.9564278704612, 174.40500000000003, 'X[3] <= 4090.5\ngini = 0.217\nsamples = 105\nvalue = [13, 92]'),
Text(424.7882237487733, 157.79500000000002, 'X[2] <= 2.5\ngini = 0.204\nsamples = 104\nvalue = [12, 92]'),
Text(421.8677134445535, 141.18500000000003, 'X[7] <= 5.5\ngini = 0.265\nsamples = 70\nvalue = [11, 59]'),
Text(420.69950932286554, 124.57500000000002, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]'),
Text(423.0359175662414, 124.57500000000002, 'X[7] <= 7.5\ngini = 0.32\nsamples = 55\nvalue = [11, 44]'),
Text(420.69950932286554, 107.96500000000003, 'X[3] <= 3944.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(419.5313052011776, 91.35500000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(421.8677134445535, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(425.3723258096173, 107.96500000000003, 'X[34] <= 0.5\ngini = 0.286\nsamples = 52\nvalue = [9, 43]'),
Text(424.20412168792933, 91.35500000000002, 'X[3] <= 3985.5\ngini = 0.265\nsamples = 51\nvalue = [8, 43]'),
Text(419.5313052011776, 74.74500000000003, 'X[24] <= 0.5\ngini = 0.161\nsamples = 34\nvalue = [3, 31]'),
Text(417.19489695780175, 58.13500000000002, 'X[0] <= 0.5\ngini = 0.117\nsamples = 32\nvalue = [2, 30]'),
Text(416.02669283611385, 41.525000000000034, 'X[4] <= 33.0\ngini = 0.375\nsamples = 8\nvalue = [2, 6]'),
Text(414.8584887144259, 24.91500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(417.19489695780175, 24.91500000000002, 'X[39] <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(416.02669283611385, 8.305000000000007, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(418.3631010794897, 8.305000000000007, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(418.3631010794897, 41.525000000000034, 'gini = 0.0\nsamples = 24\nvalue = [0, 24]'),
Text(421.8677134445535, 58.13500000000002, 'X[6] <= 6.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(420.69950932286554, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(423.0359175662414, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(428.8769381746811, 74.74500000000003, 'X[37] <= 0.5\ngini = 0.415\nsamples = 17\nvalue = [5, 12]'),
Text(426.5405299313052, 58.13500000000002, 'X[5] <= 1983.5\ngini = 0.278\nsamples = 12\nvalue = [2, 10]'),
Text(425.3723258096173, 41.525000000000034, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(427.7087340529931, 41.525000000000034, 'X[5] <= 1989.5\ngini = 0.165\nsamples = 11\nvalue = [1, 10]'),
Text(426.5405299313052, 24.91500000000002, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
Text(428.8769381746811, 24.91500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(431.2133464180569, 58.13500000000002, 'X[2] <= 0.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
Text(430.04514229636897, 41.525000000000034, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(432.3815505397448, 41.525000000000034, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(426.5405299313052, 91.35500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(427.7087340529931, 141.18500000000003, 'X[26] <= 0.5\ngini = 0.057\nsamples = 34\nvalue = [1, 33]'),
Text(426.5405299313052, 124.57500000000002, 'gini = 0.0\nsamples = 28\nvalue = [0, 28]'),
Text(428.8769381746811, 124.57500000000002, 'X[40] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(427.7087340529931, 107.96500000000003, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(430.04514229636897, 107.96500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(427.12463199214915, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(433.54975466143276, 174.40500000000003, 'X[7] <= 27.5\ngini = 0.053\nsamples = 73\nvalue = [2, 71]'),
Text(431.2133464180569, 157.79500000000002, 'X[41] <= 0.5\ngini = 0.029\nsamples = 69\nvalue = [1, 68]'),
Text(430.04514229636897, 141.18500000000003, 'gini = 0.0\nsamples = 63\nvalue = [0, 63]'),
Text(432.3815505397448, 141.18500000000003, 'X[21] <= 0.5\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(431.2133464180569, 124.57500000000002, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(433.54975466143276, 124.57500000000002, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(435.8861629048086, 157.79500000000002, 'X[4] <= 21.0\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(434.7179587831207, 141.18500000000003, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(437.05436702649655, 141.18500000000003, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(440.55897939156034, 191.01500000000001, 'X[3] <= 3949.5\ngini = 0.469\nsamples = 8\nvalue = [3, 5]'),
Text(439.3907752698724, 174.40500000000003, 'X[26] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(438.2225711481845, 157.79500000000002, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(440.55897939156034, 157.79500000000002, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(441.7271835132483, 174.40500000000003, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(444.06359175662413, 207.62500000000003, 'X[9] <= 0.5\ngini = 0.5\nsamples = 4\nvalue = [2, 2]'),
Text(442.8953876349362, 191.01500000000001, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),

```
Text(445.2317958783121, 191.01500000000001, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(441.94622178606477, 224.235, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```



3.2: Feature engineering

1. Split the training set into the new training set (X_train_featured, y_train_featured) and the validation set (X_val2,y_val2).
2. Split the feature 'DateOfBirth" into three seperate features as "Day","Month","Year" in both X_train_featured and X_val2.
3. Create new features with "FirstName", "LastName","Street","City" in both X_train_featured and X_val2.
4. Apply Ordinal Encoding for categorical features such as "LicensePlate","FullName","Location"in both X_train_featured and X_val2.
5. Log Transformation for some features in both X_train_featured and X_val2.
6. Visuaize the correlations between the label "BadCredit" with the features, remove the outliers in X_train_featured and y_train_featured (Index removal).
7. Drop features that have very minimal correlation with the label.
8. Normalize features in both X_train_featured and X_val2.
9. Apply Yeo-Johnson transformation on certain features in both X_train_featured and X_val2.
10. Apply One Hot Encoding for the features 'Sex','Married','City','Purpose' in both X_train_featured and X_val2.
11. Decision Tree prediction with fit(X_train_featured, y_train_featured), and predict on (X_val2).
12. Model Evaluation (Mean Accuracy, K-fold validation, and accuracy report).

In [120…
```
X_train_new32 = X_train.copy()
```

In [121…
```
#In this step, the transformed X training set X_train_new31 will add back the label and split again for training set and validation set.
####This step is crucial as each of the four splitted sets  will be used in the section 3.3 and 3.4 again.
X_train_new32["Label"] = y_train
X_train_new32_Split = X_train_new32.drop('Label', axis=1) #.select_dtypes(['number'])
Y_new_label32 = X_train_new32['Label']
X_train_featured, X_val2, y_train_featured, y_val2 = train_test_split(X_train_new32_Split, Y_new_label32, test_size=0.2, random_state=42)
```

In [122…
```
#Checking the first few observations
X_train_featured.head()
```

Out[122…

	UserID	Sex	PreviousDefault	FirstName	LastName	NumberPets	PreviousAccounts	ResidenceDuration	Street	LicensePlate	Amount	Married	Duration	City	Purpose	DateOfBirth
3046	842-17-6386	F	0	Katrina	Brooks	1	2	2	519 Brenda Ranch	Y56-03B	4142	0	36	New Roberttown	Repair	1976-03-17
5054	013-75-3847	F	0	Christina	Lewis	1	2	5	7551 Harmon Stravenue Apt. 338	9H 0846P	3493	0	36	Lake Debra	Education	1979-12-21
2065	181-93-2677	F	0	Melissa	Lee	2	0	3	46292 Jacob Estates	HGQ 1385	3590	0	30	Lake Debra	Household	1975-11-19
4389	521-53-3941	F	0	Regina	Beltran	1	1	3	514 Wright Parkway Suite 283	MZM 4709	3446	0	24	West Michael	NewCar	1975-04-27
5292	441-27-0220	F	0	Jacqueline	Wise	0	0	0	020 Hughes Island	6DC P54	3940	1	30	Lake Debra	Household	1971-12-30

In [123…
```
#Splitting the DateOfBirth feature into seperate features.
X_train_featured[["Year", "Month", "Day"]] = X_train_featured["DateOfBirth"].str.split("-", expand = True)
X_val2[["Year", "Month", "Day"]] = X_val2["DateOfBirth"].str.split("-", expand = True)
```

In [124…
```
#Convert the new features into numeric features.
X_train_featured["Year"] = X_train_featured["Year"].astype(int)
X_train_featured["Month"] = X_train_featured["Month"].astype(int)
X_train_featured["Day"] = X_train_featured["Day"].astype(int)
X_val2["Year"] = X_val2["Year"].astype(int)
X_val2["Month"] = X_val2["Month"].astype(int)
X_val2["Day"] = X_val2["Day"].astype(int)
```

In [125…
```
#Making a new feature as each customer has their own names.
X_train_featured["FullName"] = X_train_featured["FirstName"] + X_train_featured["LastName"]
X_val2["FullName"] = X_val2["FirstName"] + X_val2["LastName"]
```

In [126…
```
#Making a new feature as the unique Location for each credit
X_train_featured["Location"] = X_train_featured["Street"] + "-" + X_train_featured["City"]
X_val2["Location"] = X_val2["Street"] + "-" + X_val2["City"]
```

In [127…
```
#Ordinal Encoding Location feature. Location feature is currently a categorical variable.
enc = OrdinalEncoder()
```

```
enc = enc.fit(X_train_featured[['Location']])
X_train_featured['Location'] = enc.transform(X_train_featured[['Location']])
```

In [128…
```
#Ordinal Encoding Location feature. Location feature is currently a categorical variable.
enc = OrdinalEncoder()
enc = enc.fit(X_val2[['Location']])
X_val2['Location'] = enc.transform(X_val2[['Location']])
```

In [129…
```
#Ordinal Encoding FullName feature. FullName feature is currently a categorical variable.
enc2 = OrdinalEncoder()
enc2 = enc2.fit(X_train_featured[['FullName']])
X_train_featured['FullName'] = enc2.transform(X_train_featured[['FullName']])
```

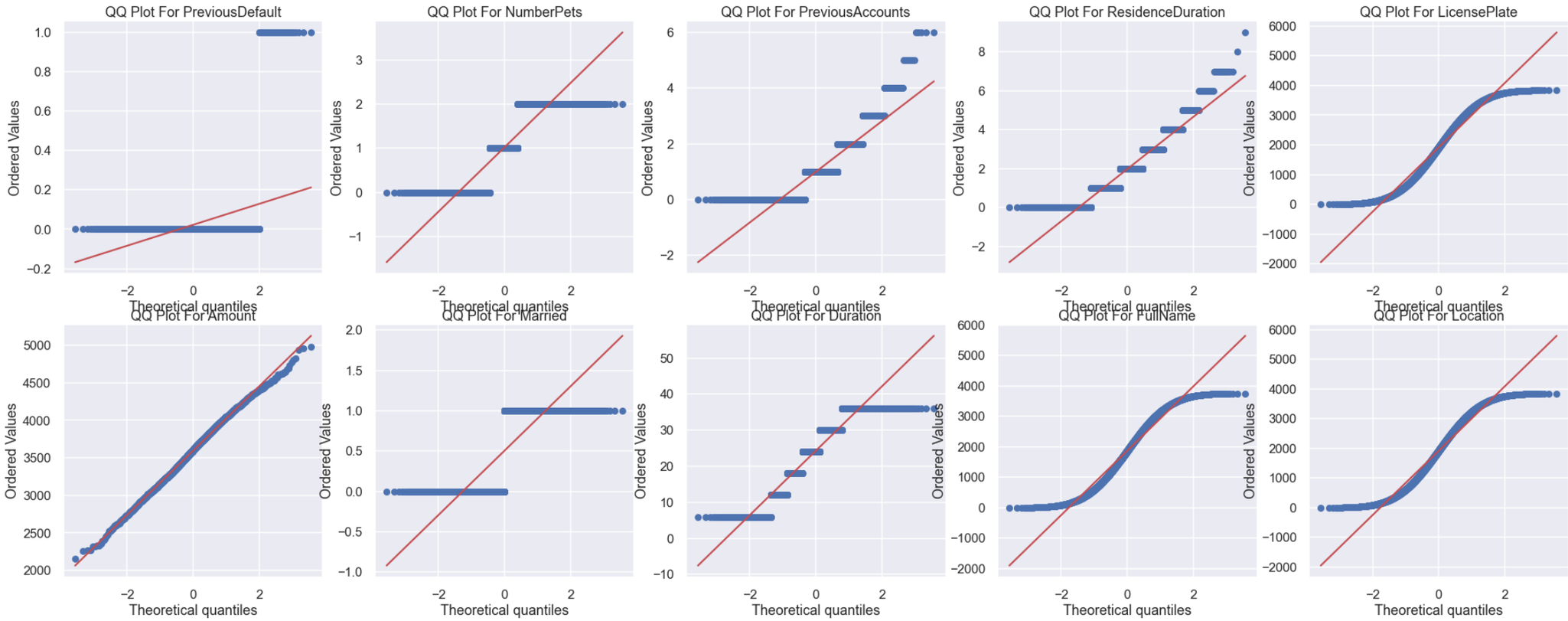
In [130…
```
#Ordinal Encoding FullName feature. FullName feature is currently a categorical variable.
enc2 = OrdinalEncoder()
enc2 = enc2.fit(X_val2[['FullName']])
X_val2['FullName'] = enc2.transform(X_val2[['FullName']])
```

In [131…
```
#Ordinal Encoding LicensePlate feature. LicensePlate feature is currently a categorical variable.
enc3 = OrdinalEncoder()
enc3 = enc3.fit(X_train_featured[['LicensePlate']])
X_train_featured['LicensePlate'] = enc3.transform(X_train_featured[['LicensePlate']])
```

In [132…
```
#Ordinal Encoding LicensePlate feature. LicensePlate feature is currently a categorical variable.
enc3 = OrdinalEncoder()
enc3 = enc3.fit(X_val2[['LicensePlate']])
X_val2['LicensePlate'] = enc3.transform(X_val2[['LicensePlate']])
```

In [133…
```
#Dropping these features as they have already been created in new features.
X_train_featured = X_train_featured.drop(['UserID','FirstName','LastName','Street','DateOfBirth'], axis=1)
X_val2 = X_val2.drop(['UserID','FirstName','LastName','Street','DateOfBirth'], axis=1)
```

In [134…
```
#The QQ plot and line of best fit for each numeric feature.
X_train_featured_qq = X_train_featured.select_dtypes(['int64','float64'])
column_list = X_train_featured_qq.columns.tolist()

fig = plt.figure(figsize=(35, 35))
for col in column_list:
    ax = fig.add_subplot(5,5, column_list.index(col)+1)
    ax.set_xlabel(col)
    stats.probplot(X_train_featured[col], dist="norm", plot=py)
    plt.title('QQ Plot For ' + col);
```



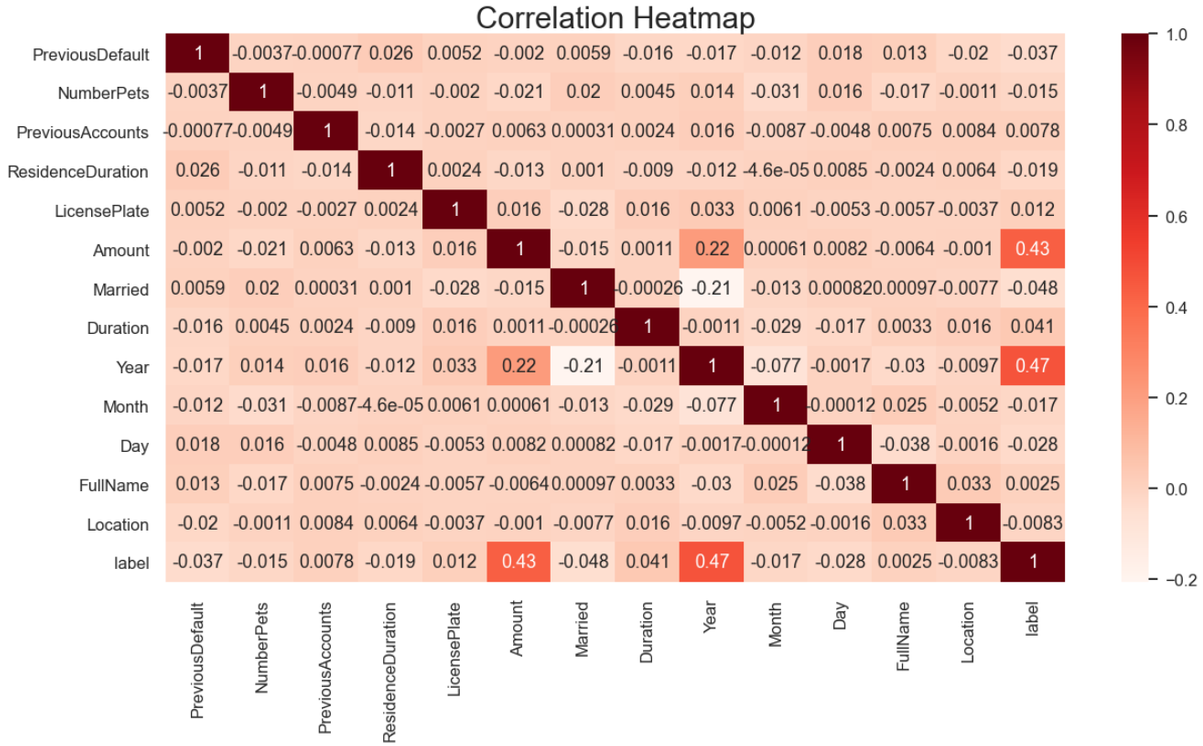
In [135…
```
#Log Transform some features as these features don't close to the line of best fit.
#Some features such as FullName, LicensePlate couldn't transform properly as the ordinal encoding contains 0.
#These features deviate from the line of best fit very much.
X_train_featured["Amount"] = np.log(X_train_featured["Amount"])
X_train_featured["Day"] = np.log(X_train_featured["Day"])
X_train_featured["Year"] = np.log(X_train_featured["Year"])
X_val2["Amount"] = np.log(X_val2["Amount"])
X_val2["Day"] = np.log(X_val2["Day"])
X_val2["Year"] = np.log(X_val2["Year"])
```

In [136…
```
#Checking the dataset again
X_train_featured.head()
```

Out[136…

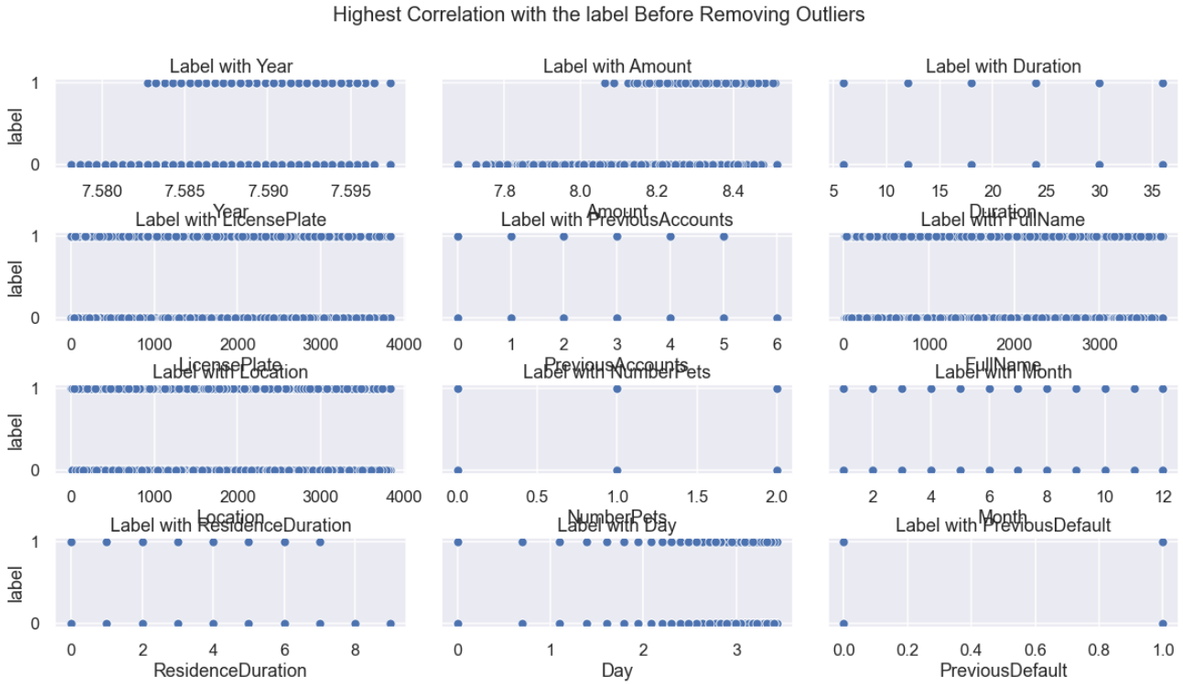
	Sex	PreviousDefault	NumberPets	PreviousAccounts	ResidenceDuration	LicensePlate	Amount	Married	Duration	City	Purpose	Year	Month	Day	FullName	Location
3046	F	0	1	2	2	3691.00	8.33	0	36	New Roberttown	Repair	7.59	3	2.83	1952.00	1977.00
5054	F	0	1	2	5	2348.00	8.16	0	36	Lake Debra	Education	7.59	12	3.04	744.00	2909.00
2065	F	0	2	0	3	2778.00	8.19	0	30	Lake Debra	Household	7.59	11	2.94	2533.00	1745.00
4389	F	0	1	1	3	3154.00	8.14	0	24	West Michael	NewCar	7.59	4	3.30	2961.00	1958.00
5292	F	0	0	0	0	1627.00	8.28	1	30	Lake Debra	Household	7.59	12	3.40	1403.00	78.00

In [137…
```
#Adding the label and training set to a temporary dataset X_combined_corr2 to check the correlations for the features agaist the label.
X_corr2 = X_train_featured.copy()
X_corr2['label'] = y_train_featured
#Correlation heatmap between the features.
X_combined_corr2 = X_corr2.corr()
plt.figure(figsize=(20,10))
sns.heatmap(X_combined_corr2, annot=True, cmap="Reds")
plt.title('Correlation Heatmap',fontsize=30);
plt.show()
```



In [138…

```
#Visualizing the features which have correlaiton with the label before removing outliers.
high_corr = X_combined_corr2['label'].sort_values(ascending=False)[1:][:13].index.tolist()
fig, axes = plt.subplots(4,3, figsize=(20, 10), sharey=True);
plt.subplots_adjust(hspace = 0.7, wspace=0.1)
fig.suptitle('Highest Correlation with the label Before Removing Outliers', fontsize=20);
for i,col in zip(range(12),high_corr):
    sns.scatterplot(y=X_corr2['label'], x=X_corr2[col], ax=axes[i//3][i%3])
    axes[i//3][i%3].set_title('Label with '+col)
```



In [139…
```
#Removing Outliers according to the visuals
#Removing the outliers in our original training set X_train_featured. Now we have finished using X_combined_corr2 for checking correlations.
#X_val2 is not touched to avoid data leakage.
#X_val2 is only used for inputing valdation set.
drop_index = X_train_featured[((X_train_featured['Day']<1))|
                 ((X_train_featured['PreviousAccounts']>=6))].index

X_train_featured = X_train_featured.drop(drop_index)
y_train_featured = y_train_featured.drop(drop_index)
```

In [140…
```
#Dropping NumberPets and Location features as these two features have very minimal correlation with the target variable.
X_train_featured = X_train_featured.drop(['NumberPets','Location'], axis=1)
X_val2 = X_val2.drop(['NumberPets','Location'], axis=1)
```

In [141…
```
#Normalizing the features in X_train_featured
scaler = StandardScaler()
scaler.fit(X_train_featured[['Amount']])
X_train_featured['Amount'] = scaler.transform(X_train_featured[['Amount']])

scaler.fit(X_train_featured[['Year']])
X_train_featured['Year'] = scaler.transform(X_train_featured[['Year']])

scaler.fit(X_train_featured[['Month']])
X_train_featured['Month'] = scaler.transform(X_train_featured[['Month']])

scaler.fit(X_train_featured[['Duration']])
X_train_featured['Duration'] = scaler.transform(X_train_featured[['Duration']])
```

In [142…
```
#Normalizing the features in X_val2
scaler = StandardScaler()
scaler.fit(X_val2[['Amount']])
X_val2['Amount'] = scaler.transform(X_val2[['Amount']])

scaler.fit(X_val2[['Year']])
X_val2['Year'] = scaler.transform(X_val2[['Year']])

scaler.fit(X_val2[['Month']])
X_val2['Month'] = scaler.transform(X_val2[['Month']])

scaler.fit(X_val2[['Duration']])
X_val2['Duration'] = scaler.transform(X_val2[['Duration']])
```

In [143…
```
#Checking the dataset again.
X_train_featured.head()
```

Out[143…

	Sex	PreviousDefault	PreviousAccounts	ResidenceDuration	LicensePlate	Amount	Married	Duration	City	Purpose	Year	Month	Day	FullName

	Sex	PreviousDefault	PreviousAccounts	ResidenceDuration	LicensePlate	Amount	Married	Duration	City	Purpose	Year	Month	Day	FullName
3046	F	0	2	2	3691.00	1.21	0	1.25	New Roberttown	Repair	0.44	-1.04	2.83	1952.00
5054	F	0	2	5	2348.00	-0.18	0	1.25	Lake Debra	Education	0.95	1.61	3.04	744.00
2065	F	0	0	3	2778.00	0.04	0	0.61	Lake Debra	Household	0.27	1.31	2.94	2533.00
4389	F	0	1	3	3154.00	-0.29	0	-0.03	West Michael	NewCar	0.27	-0.75	3.30	2961.00
5292	F	0	0	0	1627.00	0.80	1	0.61	Lake Debra	Household	-0.41	1.61	3.40	1403.00

In [144…
```
#Peform Yeo-Johnson transformation for the features.
#The Yeo-Johnson transformation is similar to Box-Cox transformation in which it transfers a skewed feature to have a normal distribution.
#Yeo-Johnson can consider negative values.

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['PreviousAccounts']])
X_train_featured['PreviousAccounts'] = scaler.transform(X_train_featured[['PreviousAccounts']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['LicensePlate']])
X_train_featured['LicensePlate'] = scaler.transform(X_train_featured[['LicensePlate']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['ResidenceDuration']])
X_train_featured['ResidenceDuration'] = scaler.transform(X_train_featured[['ResidenceDuration']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_val2[['PreviousAccounts']])
X_val2['PreviousAccounts'] = scaler.transform(X_val2[['PreviousAccounts']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_val2[['LicensePlate']])
X_val2['LicensePlate'] = scaler.transform(X_val2[['LicensePlate']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_val2[['ResidenceDuration']])
X_val2['ResidenceDuration'] = scaler.transform(X_val2[['ResidenceDuration']])
```

In [145…
```
#One hot encoder for the binary features Sex, Married, PreviousDefault.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the training set.
enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
enc = enc.fit(X_train_featured[['Sex','Married','City','Purpose','PreviousDefault']])
enc.transform(X_train_featured[['Sex','Married','City','Purpose','PreviousDefault']])
```

Out[145…
```
array([[1., 0., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 1., 0.],
       [0., 1., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.]])
```

In [146…
```
#One hot encoder for the binary features Sex, Married,PreviousDefault.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the validation set.
enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
enc = enc.fit(X_val2[['Sex','Married','City','Purpose','PreviousDefault']])
enc.transform(X_val2[['Sex','Married','City','Purpose','PreviousDefault']])
```

Out[146…
```
array([[1., 0., 1., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       ...,
       [0., 1., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 1., 0.],
       [0., 1., 1., ..., 0., 1., 0.]])
```

In [147…
```
#One hot encoder for the binary features Sex, Married, PreviousDefault.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the training set.
_ohe_array = enc.transform(X_train_featured[['Sex','Married','City','Purpose','PreviousDefault']])
_ohe_names = enc.get_feature_names()
for i in range(_ohe_array.shape[1]):
  X_train_featured[_ohe_names[i]] = _ohe_array[:,i]
```

In [148…
```
#One hot encoder for the binary features Sex, Married,PreviousDefault.
#One hot encoder for the City and Purpose features.
#Apply the transformation on the validation set.
_ohe_array = enc.transform(X_val2[['Sex','Married','City','Purpose','PreviousDefault']])
_ohe_names = enc.get_feature_names()
for i in range(_ohe_array.shape[1]):
  X_val2[_ohe_names[i]] = _ohe_array[:,i]
```

In [149…
```
#Dropping the original features.
X_train_featured = X_train_featured.drop(['Sex','Married','City','Purpose','PreviousDefault'], axis=1)
X_val2 = X_val2.drop(['Sex','Married','City','Purpose','PreviousDefault'], axis=1)
```

In [150…
```
#Using decision tree classifer again for modeling.
clf2 = DecisionTreeClassifier(random_state=43)
clf2.fit(X_train_featured, y_train_featured)
y_pred_dt2 = clf2.predict(X_val2)
```

In [151…
```
#Confusion matrix
confusion_matrix(y_val2, y_pred_dt2)
```

Out[151…
```
array([[702, 101],
       [ 64,  93]], dtype=int64)
```

In [152…
```
#Concatenating X_train and Y_train data for k-fold validation.
X_train_complete2 = pd.concat([X_train_featured, X_val2])
y_train_complete2 = pd.concat([y_train_featured, y_val2])
```

In [153…
```
#Estimating mean accruacy of the model with K fold = 15.
scores2 = cross_val_score(clf2, X_train_complete2,y_train_complete2, cv=15, scoring="accuracy")
print("Mean Accuracy: {:.4f}".format(np.mean(scores2)))
```

```
Mean Accuracy: 0.8353
```

In [154…
```
#Printing the accuracy report.
class_names = [str(x) for x in clf2.classes_]
print(classification_report(y_val2, y_pred_dt2, target_names=class_names))
```

```
              precision    recall  f1-score   support

           0       0.92      0.87      0.89       803
           1       0.48      0.59      0.53       157

    accuracy                           0.83       960
   macro avg       0.70      0.73      0.71       960
weighted avg       0.84      0.83      0.84       960
```

In [155…
```
#Measuring the performance of the model prediction against validation result.
print("Accuracy = {:.2f}".format(accuracy_score(y_val2, y_pred_dt2)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_val2, y_pred_dt2)))
print("F1 Score = {:.2f}".format(f1_score(y_val2, y_pred_dt2)))
print("Log Loss = {:.2f}".format(log_loss(y_val2, y_pred_dt2)))
```

```
Accuracy = 0.83
Kappa = 0.43
F1 Score = 0.53
Log Loss = 5.94
```

Compare with the baseline model: In the baseline model, the mean score with K=15 fold cross validations is 0.8353. The Accuracy and F1 score is 0.84 and 0.52 respectively. In the feature engineering model, the mean score with K=15 fold cross validations is 0.8353.The accuracy and F1 Score is 0.83 and 0.53 respectively. Both results are very close to each other. This is because the feature engineering provide very similar effects as the baseline model. The engineered features remove some outliers which might be insignificant to the modeling. The newly created features have some impacts for modeling. The value gained from this step is that the features are re-organized and the noises are removed.

3.3: Feature selection

The steps taken in the feature selection model is:

1. Take the model trained from 3.2 section (clf2).
2. Apply wrapper method on the model clf2.
3. Identify that there are 15 features important.
4. Build a new model with the transformed training set and validation set.
5. Model Evaluation (Mean Accuracy, K-fold validation, and accuracy report).

In [156…
```
#Taking the clf2 from the section 3.2 and applying Recursive Feature Elimination (Wrapper Method).
sel = RFE(estimator=clf2, n_features_to_select = 15)
sel = sel.fit(X_train_featured, y_train_featured)
```

In [157…
```
#Checking the ranking for each feature in the RFE model.
sel.ranking_
```

Out[157…
```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  3, 13,  1, 14,  9,  4,  7, 12,
        5, 10, 24, 19, 18, 27, 22, 28,  1,  1, 17, 16,  1,  8, 21, 23, 15,
        2,  1,  1, 20, 25, 11,  6, 26, 29])
```

In [158…
```
#Transform the data and build a new decision tree model.
X_train_new = sel.transform(X_train_featured)
X_val2_new = sel.transform(X_val2)
clf3 = DecisionTreeClassifier(random_state=43)
clf3.fit(X_train_new, y_train_featured)
y_pred3 = clf3.predict(X_val2_new)
```

In [159…
```
#Checking the important features.
for i in range(X_train_new.shape[1]):
        print('Column: %d, Selected %s, Rank: %.3f' % (i, sel.support_[i], sel.ranking_[i]))
```

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected True, Rank: 1.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected True, Rank: 1.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected True, Rank: 1.000
Column: 6, Selected True, Rank: 1.000
Column: 7, Selected True, Rank: 1.000
Column: 8, Selected True, Rank: 1.000
```

```
Column: 9, Selected False, Rank: 3.000
Column: 10, Selected False, Rank: 13.000
Column: 11, Selected True, Rank: 1.000
Column: 12, Selected False, Rank: 14.000
Column: 13, Selected False, Rank: 9.000
Column: 14, Selected False, Rank: 4.000
```

In [160…
```
#Printing the accuracy report.
print(classification_report(y_val2, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       803
           1       0.51      0.57      0.54       157

    accuracy                           0.84       960
   macro avg       0.71      0.73      0.72       960
weighted avg       0.85      0.84      0.84       960
```

In [161…
```
#Estimating mean accruacy of the model with K fold = 15.
scores = cross_val_score(clf3, X_train_complete2, y_train_complete2, cv=15, scoring="accuracy")
print("Mean Accuracy: {:.4f}".format(np.mean(scores)))
```

```
Mean Accuracy: 0.8353
```

In [162…
```
#Measuring the performance of the model prediction against validation result.
print("Accuracy = {:.2f}".format(accuracy_score(y_val2, y_pred3)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_val2, y_pred3)))
print("F1 Score = {:.2f}".format(f1_score(y_val2, y_pred3)))
print("Log Loss = {:.2f}".format(log_loss(y_val2, y_pred3)))
```

```
Accuracy = 0.84
Kappa = 0.44
F1 Score = 0.54
Log Loss = 5.58
```

Compare with the feature engineering model: In the featured selection model, the mean score with K=15 fold cross validations is 0.8353.The accuracy and F1 Score is 0.84 and 0.54 respectively, a slight improvement comparing to the feature engineering model.

3.4: Hyperparameter tuning

The steps taken in the hyperparameter model is:

1. Take the model trained from 3.3 section (clf3) and apply random search CV to find the best paramters. The best parameters are set with a ranged random integers and uniform numbers.
2. Retrain the model with the best parameters
3. Check the best parameters and estimators.
4. Make predicitions on the validation set with the new parameters.
5. Model Evaluation (Mean Accuracy, and accuracy report).

In [163…
```
#Taking the clf3 from the section 3.3 and use random search cv to find the best parameters for the model.
#Retrain the model with the best parameters.
#In the random search function, the parameters are with the ranged integers and let the machine to find the optimal parameters.
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

params = {"criterion": ["gini", "entropy"],
          "splitter": ["best", "random"],
          "class_weight": ['balanced', None],
          "max_depth": randint(2, 21),
          "min_samples_leaf": randint(5, 15),
          "max_features": uniform(0.0, 1.0)}

Best_model = RandomizedSearchCV(clf3, params,n_iter=1000, scoring='f1_macro',cv=10, verbose=1,random_state = 42)
Best_model = Best_model.fit(X_train_featured, y_train_featured)
```

```
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:  3.4min finished
```

In [164…
```
#The best parameters
Best_model.best_params_
```

Out[164…
```
{'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 2,
 'max_features': 0.8698963620621283,
 'min_samples_leaf': 11,
 'splitter': 'best'}
```

In [165…
```
#The best estimators
Best_model.best_estimator_
```

Out[165…
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=2, max_features=0.8698963620621283,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=11,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=43, splitter='best')
```

In [166…
```
#Best mean accuracy
Best_model.best_score_
```

Out[166…
```
0.7860931859454057
```

In [167…
```
#Make the prediction with the new model on the validation set.
y_pred4 = Best_model.predict(X_val2)
print(classification_report(y_val2, y_pred4))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       803
           1       0.61      0.64      0.62       157

    accuracy                           0.87       960
   macro avg       0.77      0.78      0.77       960
weighted avg       0.88      0.87      0.87       960
```

In [168…
```
#Parameters result comparsion
def cv_results_to_df(cv_results):
    results = pd.DataFrame(list(cv_results['params']))
    results['mean_val_score'] = cv_results['mean_test_score']
    results['rank_val_score'] = cv_results['rank_test_score']

    results = results.sort_values(['mean_val_score'], ascending=False)
    return results
```

In [169…
```
#The parameters result comparsion.
cv_results_to_df(Best_model.cv_results_)
```

Out[169…

	class_weight	criterion	max_depth	max_features	min_samples_leaf	splitter	mean_val_score	rank_val_score
698	None	entropy	3	0.78	13	best	0.79	1
344	None	entropy	3	1.00	10	best	0.79	1
553	None	entropy	2	0.46	6	best	0.79	1
541	None	entropy	2	0.45	5	best	0.79	1
283	None	entropy	3	1.00	5	best	0.79	1
…	…	…	…	…	…	…	…	…
401	balanced	gini	3	0.07	6	random	0.26	996
301	balanced	entropy	2	0.09	5	best	0.23	997
460	balanced	gini	2	0.09	8	best	0.23	997
945	balanced	entropy	3	0.17	13	random	0.21	999
680	balanced	entropy	2	0.18	6	random	0.20	1000

1000 rows × 8 columns

In [170…
```
#Confusion matrix
confusion_matrix(y_val2, y_pred4)
```

Out[170…
```
array([[738,  65],
       [ 57, 100]], dtype=int64)
```

In [171…
```
#Measuring the performance of the model prediction against validation result.
print("Accuracy = {:.2f}".format(accuracy_score(y_val2, y_pred4)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_val2, y_pred4)))
print("F1 Score = {:.2f}".format(f1_score(y_val2, y_pred4)))
print("Log Loss = {:.2f}".format(log_loss(y_val2, y_pred4)))
```

```
Accuracy = 0.87
Kappa = 0.54
F1 Score = 0.62
Log Loss = 4.39
```

Compare with the feature selection model: In the hyperparamter tuning model, we have obtained the best parameters for the selective features, which are criterion = "entropy", max_depth = 2, max_features = 0.87, min_samples_leaf = 11. The mean score for the model is 0.788. The accruacy and F1 Score is 0.87 and 0.62, respectively. The model has been siginificantly improved by tuning hyperparameters.

3.5: Performance estimation

The steps taken in the Performance estimation model is:

1. X_test and y_test are the unseen data which will be used in production.
2. Replicate the feature engineering in the section 3.2 for X_test, so all the respective features are organized and encoded.
3. Use the best model (Best_model) from the section 3.4 and retrain the model with the full training set and training label.
4. Make the prediction on the testing set (featured X_test)
5. Model Evaluation (Mean Accuracy, K-fold validation, and accuracy report).

In [172…
```
#Splitting the DateOfBirth feature into seperate features.
X_test[["Year", "Month", "Day"]] = X_test["DateOfBirth"].str.split("-", expand = True)
```

In [173…
```
#Convert the new features into numeric features.
X_test["Year"] = X_test["Year"].astype(int)
X_test["Month"] = X_test["Month"].astype(int)
X_test["Day"] = X_test["Day"].astype(int)
```

In [174…
```
#Making a new feature as each customer has their own names.
X_test["FullName"] = X_test["FirstName"] + X_test["LastName"]
```

In [175…
```
#Making a new feature as the unique Location for each loan
X_test["Location"] = X_test["Street"] + "-" + X_test["City"]
```

In [176…
```
#Ordinal Encoding Location feature. Location feature is currently a categorical variable.

enc = OrdinalEncoder()
enc = enc.fit(X_test[['Location']])

X_test['Location'] = enc.transform(X_test[['Location']])
```

In [177…
```
#Ordinal Encoding FullName feature. FullName feature is currently a categorical variable.
enc2 = OrdinalEncoder()
enc2 = enc2.fit(X_test[['FullName']])
X_test['FullName'] = enc2.transform(X_test[['FullName']])
```

In [178…
```
#Ordinal Encoding LicensePlate feature. LicensePlate feature is currently a categorical variable.
enc3 = OrdinalEncoder()
enc3 = enc3.fit(X_test[['LicensePlate']])
X_test['LicensePlate'] = enc3.transform(X_test[['LicensePlate']])
```

In [179…
```
#Dropping these features as they have already been created in new features.
X_test = X_test.drop(['UserID','FirstName','LastName','Street','DateOfBirth'], axis=1)
```

In [180…
```
#Log transformation on the features
X_test["Amount"] = np.log(X_test["Amount"])
X_test["Day"] = np.log(X_test["Day"])
X_test["Year"] = np.log(X_test["Year"])
```

In [181…
```
#Drop the unused features as these features are not used in the model.
X_test = X_test.drop(['NumberPets','Location'], axis=1)
```

In [182…
```
#Normalizing the data for the features.
scaler = StandardScaler()
scaler.fit(X_test[['Amount']])
X_test['Amount'] = scaler.transform(X_test[['Amount']])

scaler.fit(X_test[['Year']])
X_test['Year'] = scaler.transform(X_test[['Year']])

scaler.fit(X_test[['Month']])
X_test['Month'] = scaler.transform(X_test[['Month']])

scaler.fit(X_test[['Duration']])
X_test['Duration'] = scaler.transform(X_test[['Duration']])
```

In [183…
```
#Peform Yeo-Johnson transformation for the features

#The Yeo-Johnson transformation is similar to Box-Cox transformation in which it transfers a skewed feature to have a normal distribution.
#Yeo-Johnson can consider negative values.

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['PreviousAccounts']])
X_test['PreviousAccounts'] = scaler.transform(X_test[['PreviousAccounts']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['LicensePlate']])
X_test['LicensePlate'] = scaler.transform(X_test[['LicensePlate']])

scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(X_train_featured[['ResidenceDuration']])
X_test['ResidenceDuration'] = scaler.transform(X_test[['ResidenceDuration']])
```

In [184…
```
#One hot encoder for the binary features Sex, Married, PreviousDefault
#One hot encoder for the City and Purpose features.

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
enc = enc.fit(X_test[['Sex','Married','City','Purpose','PreviousDefault']])

enc.transform(X_test[['Sex','Married','City','Purpose','PreviousDefault']])
```

Out[184…
```
array([[1., 0., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       ...,
       [0., 1., 0., ..., 0., 1., 0.],
       [0., 1., 1., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.]])
```

In [185…
```
#One hot encoder for the binary features Sex, Married, PreviousDefault
#One hot encoder for the City and Purpose features.
_ohe_array = enc.transform(X_test[['Sex','Married','City','Purpose','PreviousDefault']])
_ohe_names = enc.get_feature_names()
for i in range(_ohe_array.shape[1]):
  X_test[_ohe_names[i]] = _ohe_array[:,i]
```

In [186…
```
#Dropping the original features.
X_test = X_test.drop(['Sex','Married','City','Purpose','PreviousDefault'], axis=1)
```

In [187…
```
#In this step, we are fitting the best model on the complete training set and training label.
#The best model will predict the unseen data
#X_train_complete2 and y_train_complete2 are the full featured training set and training label.
Best_model.fit(X_train_complete2, y_train_complete2)
y_pred_best = Best_model.predict(X_test)
```

```
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:  4.1min finished
```

In [188…
```
#Confusion matrix
confusion_matrix(y_test, y_pred_best)
```

Out[188…
```
array([[854, 137],
       [ 29, 180]], dtype=int64)
```

In [189…
```
#Make the prediction with the new model on the validation set.
print(classification_report(y_test, y_pred_best))
```

```
              precision    recall  f1-score   support

           0       0.97      0.86      0.91       991
           1       0.57      0.86      0.68       209

    accuracy                           0.86      1200
   macro avg       0.77      0.86      0.80      1200
weighted avg       0.90      0.86      0.87      1200
```

In [190…
```
#Measuring the performance of the model prediction against the test result.
from sklearn.metrics import accuracy_score, cohen_kappa_score, f1_score, log_loss
print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_best)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_best)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_best)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_best)))
```

```
Accuracy = 0.86
Kappa = 0.60
F1 Score = 0.68
Log Loss = 4.78
```

Final Evaluation: The Best_model performs relatively good in the production. The model has been re-trained with the full training set and training label. The F1 Score is 0.68, Precision Score is 0.77 and Recall Score is 0.86.The result is among the best in all the sections. The original baseline model F1 Score is 0.52, showing that the feature engineering, feature selection and hyperparamter tuning add values to the model.

Question 4: Uncle Steve's Wind Farm

Instructions

Uncle Steve has invested in wind. He's built a BIG wind farm with a total of 700 turbines. He's been running the farm for a couple of years now and things are going well. He sells the power generated by the farm to the Kingston government and makes a tidy profit. And, of course, he has been gathering data about the turbines' operations.

One area of concern, however, is the cost of maintenece. While the turbines are fairly robust, it seems like one breaks/fails every couple of days. When a turbine fails, it usually costs around $20,000 to repair it. Yikes!

Currently, Uncle Steve is not doing any preventative maintenance. He just waits until a turbine fails, and then he fixes it. But Uncle Steve has recently learned that if he services a turbine *before* it fails, it will only cost around $2,000.

Obviously, there is a potential to save a lot of money here. But first, Uncle Steve would need to figure out *which* turbines are about to fail. Uncle Steve being Uncle Steve, he wants to use ML to build a predictive maintenance model. The model will alert Uncle Steve to potential turbine failures before they happen, giving Uncle Steve a chance to perform an inspection on the turbine and then fix the turbine before it fails. Uncle Steve plans to run the model every morning. For all the turbines that the model predicts will fail, Uncle Steve will order an inspection (which cost a flat $500, no matter if the turbine was in good health or not; the $500 would not be part of the $2,000 service cost). For the rest of the turbines, Uncle Steve will do nothing.

Uncle Steve has used the last few year's worth of operation data to build and assess a model to predict which turbines will fail on any given day. (The data includes useful features like sensor readings, power output, weather, and many more, but those are not important for now.) In fact, he didn't stop there: he built and assessed two models. One model uses using deep learning (in this case, RNNs), and the other uses random forests.

He's tuned the bejeebers out of each model and is comfortable that he has found the best-performing version of each. Both models seem really good: both have accuracy scores > 99%. The RNN has better recall, but Uncle Steve is convinced that the random forest model will be better for him since it has better precision. Just to be sure, he has hired you to double check his calculations.

Your task

Which model will save Uncle Steve more money? Justify.

In addition to the details above, here is the assessment of each model:

- Confusion matrix for the random forest:

	Predicted Fail	Predicted No Fail
Actual Fail	201	55
Actual No Fail	50	255195

- Confusion matrix for the RNN:

	Predicted Fail	Predicted No Fail
Actual Fail	226	30
Actual No Fail	1200	254045

Marking

- *Quality*. Response is well-justified and convincing.
- *Style*. Response uses proper grammar, spelling, and punctuation. Response is clear and professional. Response is complete, but not overly-verbose. Response follows length guidelines.

Tips

- Figure out how much Uncle Steve is currently (i.e., without any predictive maintinance models) paying in maintenance costs.
- Use the information provided above to create a cost matrix.
- Use the cost matrix and the confusion matrices to determine the costs of each model.
- The cost of an inspection is the same, no matter if the turbine is in good condition or is about to fail.
- If the inspection determines that a turbine is about to fail, then it will be fixed right then and there for the additional fee.
- For simplicity, assume the inspections are perfect: i.e., that inspecting a turbine will definitely catch any problems that might exist, and won't accidentally flag an otherwise-healthy turbine.

Answer:

According to the outputs from each model, there is a total of 255,501 observations. The observations are from the 700 turbines operation in the last few years. In these observations, 256 are failures, and 255,245 are not failures. Currently, Uncle Steve is not doing any preventative maintenance, he fixes turbines when broken. Uncle Steve has paid $20,000 * 256 = $5,120,000 in the past few years to maintain turbine operation without any modeling prediction, preventive maintenance, and inspection.

When Uncle Steven decides to use ML models to predict failures and conduct preventive maintenance and inspection, the cost matrix is derived below.

Cost Matrix

	Predicted Fail	Predicted No Fail
Actual Fail	$2,500	$20,000
Actual No Fail	$2,500	$0

In the cost matrix "Predicted Fail" column, Uncle Steve will pay $500 for inspection for all turbines predicted to fail. The task assumes that the inspection will certainly catch any problem in the turbine, and then the preventive maintenance cost of $2,000 will be applied. Thus, when a model predicts failure and whether in actual that is a failure or not, the total cost is $2,500. In contrast, in the "Predicted No Fail" column, Uncle Steve couldn't do anything when a model didn't predict any failures. In this case, when there is an actual turbine broken, Uncle Steve must pay the entire $20,000 for the regular maintenance. Besides, when the model predicts no failure and, there is no failure in actuality, Uncle Steve pays nothing, and turbines operate normally.

To calculate the total cost for each model, take the confusion matrix from each model and multiply the cost matrix. Random Forest (RF) will cost $1,727,500 and Uncle Steve saves $5,120,000 - $1,727,500 = $3,392,500. Recurrent Neural Network (RNN) will cost $4,165,000 and Uncle Steve saves $5,120,000 - $4,165,000 = $955,000. However, the models require assessment and accuracy evaluation.

Random Forest

	Predicted Fail	Predicted No Fail
Actual Fail	201	55
Actual No Fail	50	255,195

X

Cost Matrix

	Predicted Fail	Predicted No Fail
Actual Fail	$2,500	$20,000
Actual No Fail	$2,500	$0

=

Total Cost

	Predicted Fail	Predicted No Fail
Actual Fail	$502,500	$1,100,000
Actual No Fail	$125,000	$0

Recurrent Neural Network

	Predicted Fail	Predicted No Fail
Actual Fail	226	30
Actual No Fail	1,200	254,045

X

Cost Matrix

	Predicted Fail	Predicted No Fail
Actual Fail	$2,500	$20,000
Actual No Fail	$2,500	$0

=

Total Cost

	Predicted Fail	Predicted No Fail
Actual Fail	$565,000	$600,000
Actual No Fail	$3,000,000	$0

From the measures below, we can see that both models have more than 99% accuracy. However, accuracy is usually not a good measurement because the dataset contains imbalance. The dataset in this task is imbalanced because more than 90% of the dataset indicates no failures for turbines. Uncle Steve needs a model which can accurately predict the number of failures so he can purchase a reasonable amount of inspection. We would have to check the F1 score, Precision, and Recall. For Precision measurement, RF has scored 80%, and RNN only has 16%. Precision measures how many actual positives (failures) have been identified in both predicted and actual positives. RNN makes too many false positive predictions which would waste Uncle Steve's money. RNN has made too much Type 1 error. In addition, RF has scored 78%, and RNN has scored 88% in Recall. Recall measures the % of predicted positives in total actual positives. RNN has done slightly better than RF, but both results are relatively close. In determining the trade-off of the 10% difference, (226 – 201) *$20,000 = $50,000 < (1200 – 50)* $2,500 = $2,875,000. Uncle Steve pays way too much for inspection than missing the failures. Finally, RF has scored 79%, and RNN has scored 27% in the F1 measurement. F1 score is the harmonic mean of the model's precision and Recall. It measures the overall model performance, especially when the dataset has a large class imbalance. RF does better than RNN in this case.

	Random Forest	**Recurrent Neural Network**
Accuracy	99.96%	99.52%
Precision	80.08%	15.85%
Recall	78.52%	88.28%
F1	79.29%	26.87%

Therefore, Uncle Steve should keep using the Random Forest model, and the model will save him $3,392,500 with doing nothing and save him $3,392,500 - $955,000 = $2,437,500 more with the Recurrent Neural Network model.