CODE FOR COLORED PLOTS

CHAPTER 2: DATA EXPLORATION AND VISUALIZATION

EXERCISE 2.03: VISUALIZING DATA WITH PANDAS

4. Group the **`Revenue`** by **`Order method type`** and create a bar plot:

```
sales.groupby('Order method type').sum()\
     .plot(kind = 'bar', y = 'Revenue')
```

This gives the following output:

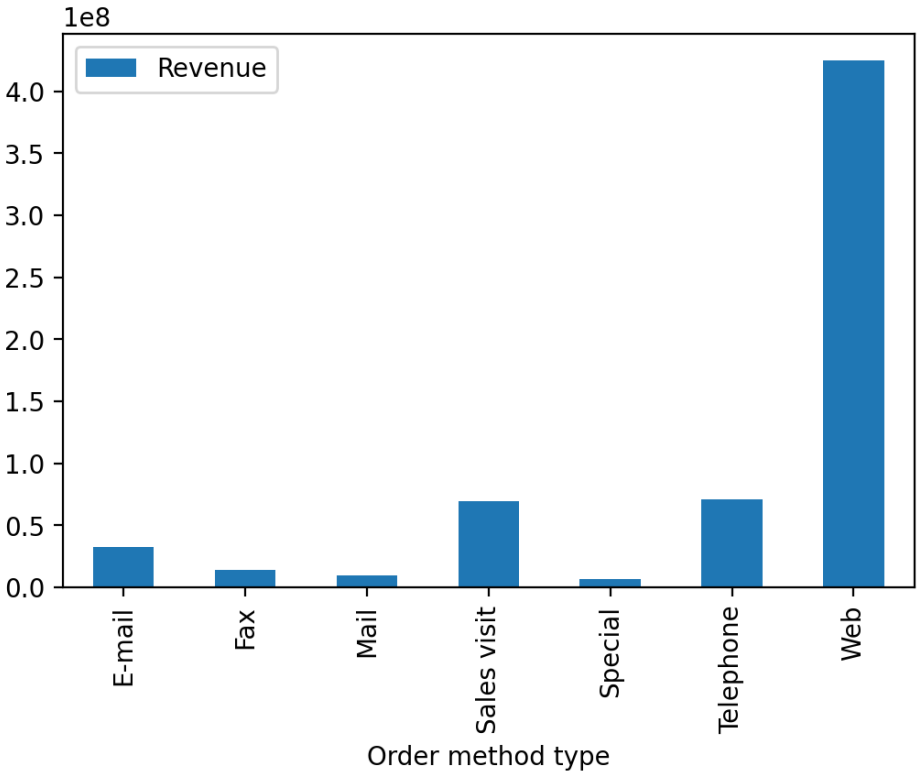

Figure 2.53: Revenue generated through each order method type in sales.csv

From the preceding image, you can infer that web orders generate the maximum revenue.

5. Now group the columns by year and create boxplots to get an idea on a relative scale:

```
plot= sales.groupby('Year')[['Revenue', 'Planned revenue',\
                             'Gross profit']].plot(kind= 'box')
```

You should get the following plots. The first plot represents the year 2004, the second plot represents the year 2005, the third plot represents the year 2006 and the final one represents 2007.

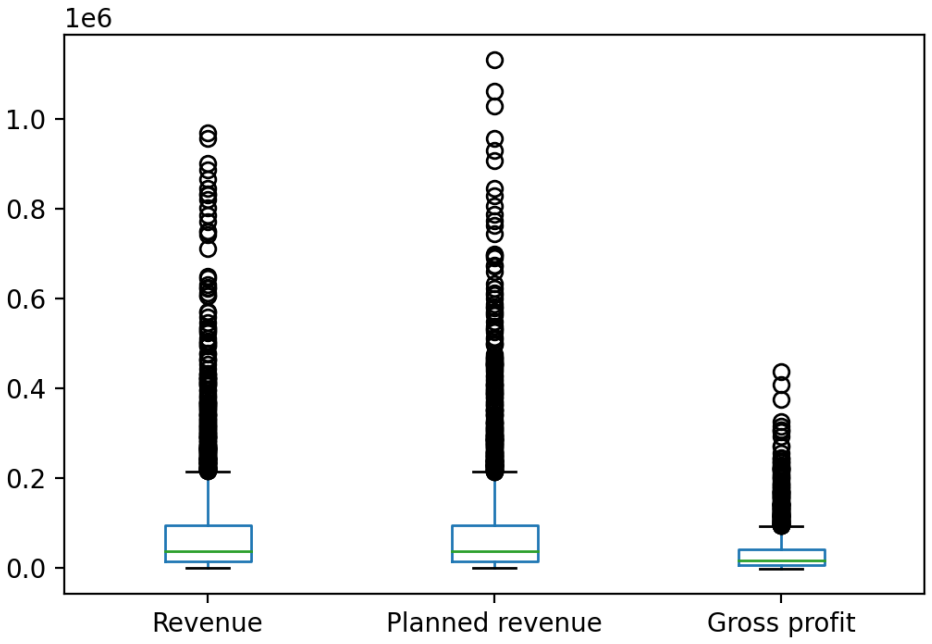

Figure 2.54: Boxplot for Revenue, Planned revenue and Gross profit for year 2004

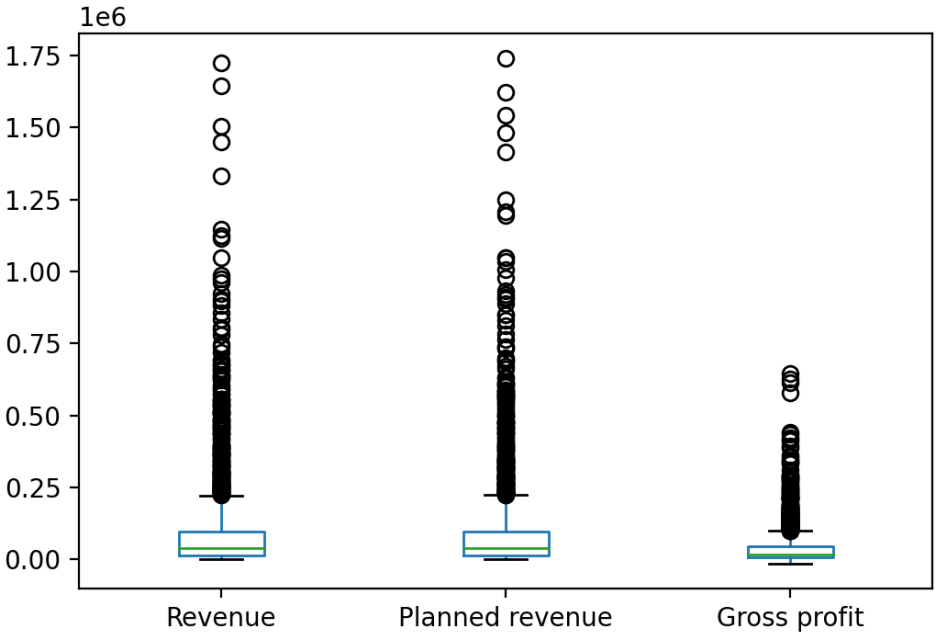

Figure 2.55: Boxplot for Revenue, Planned revenue and Gross profit for year 2005

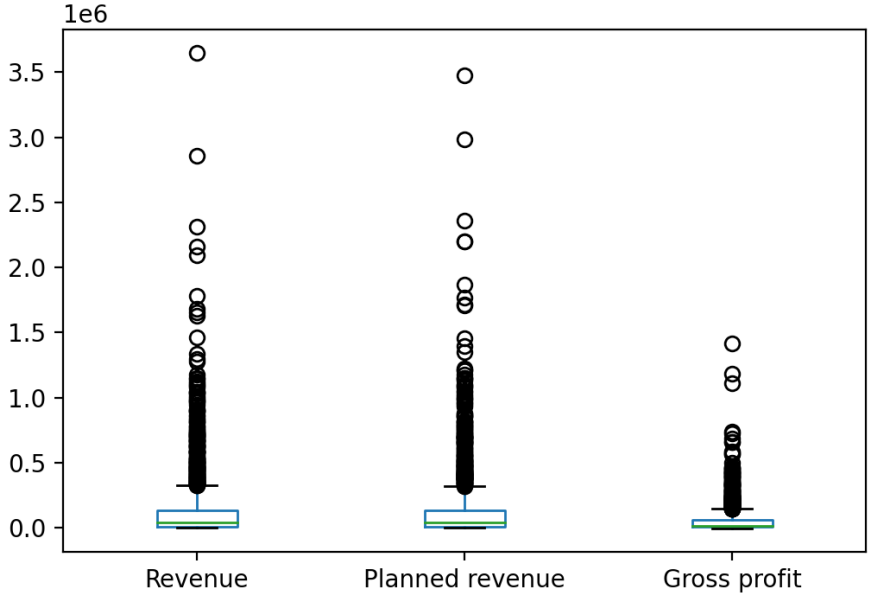

Figure 2.56: Boxplot for Revenue, Planned revenue and Gross profit for year 2006

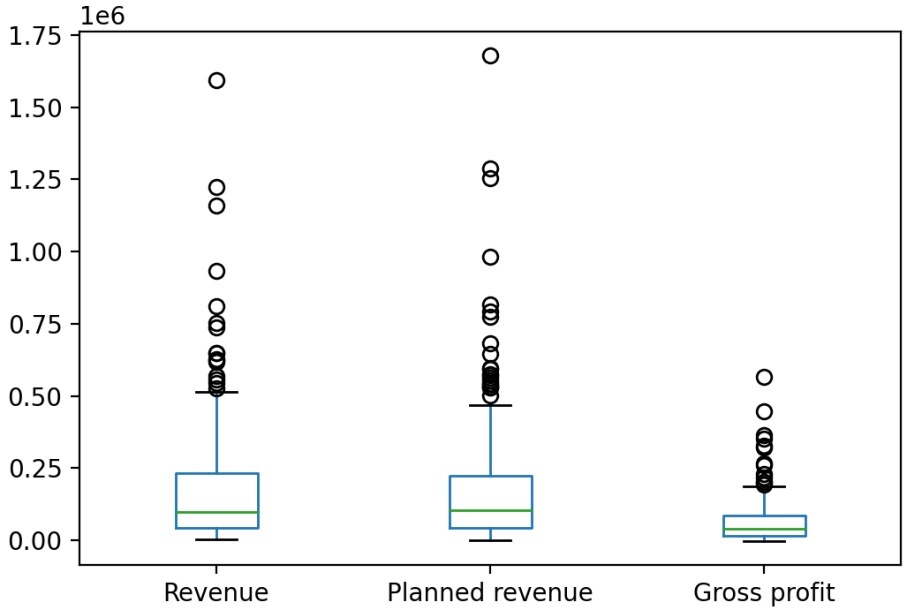

Figure 2.57: Boxplot for Revenue, Planned revenue and Gross profit for year 2007

The bubbles in the plots represent outliers. Outliers are extreme values in the data. They are caused either due to mistakes in measurement or due to the real behavior of the data. Outlier treatment depends entirely on the business use case. In some of the scenarios, outliers are dropped or are capped at a certain value based on the inputs from the business. It is not always advisable to drop the outliers as they can give us a lot of hidden information in the data.

From the above plots, we can infer that **`Revenue`** and **`Planned revenue`** have a higher median than **`Gross profit`** (the median is represented by the line inside the box).

ACTIVITY 2.01: ANALYZING ADVERTISEMENTS

8. To find out which Products have a higher viewership on TV, plot a bar chart between Products and TV.

```
ads.groupby('Products').sum().plot(kind = 'bar', y = 'TV')
```

The output should look like the following:

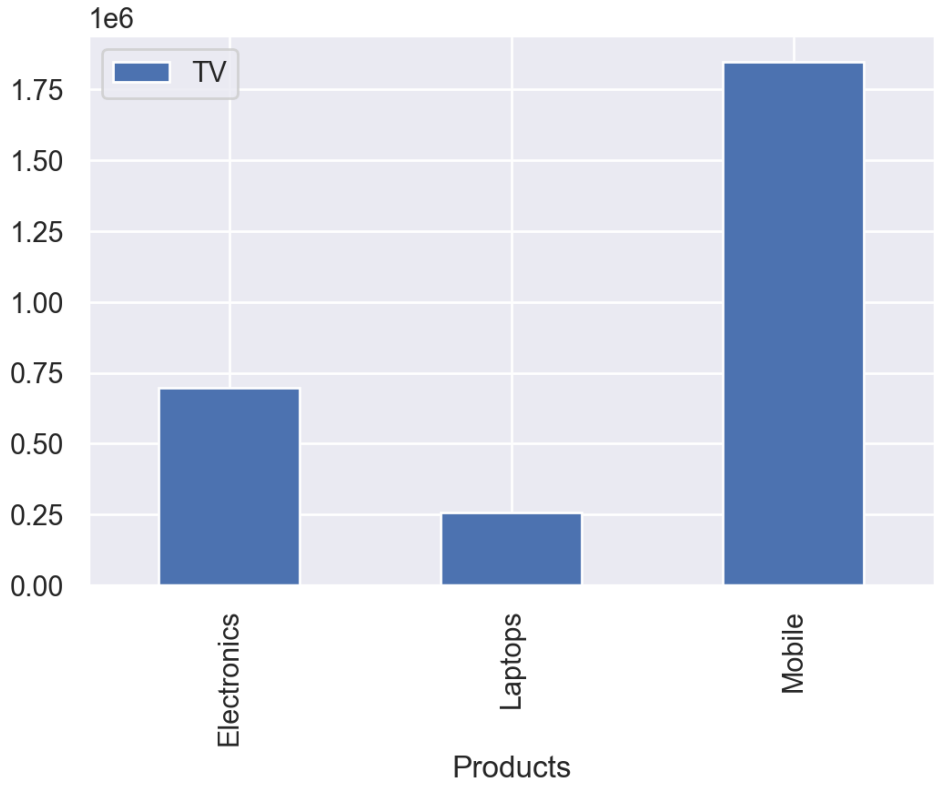

Figure 2.70: Bar Chart between Products and TV

You can observe that the mobile category has the highest viewership on TV followed by Electronics.

9. To find out which products have the lowest viewership on the web, plot a bar chart between **`Products`** and **`Web`** using the following command:

```
ads.groupby('Products').sum().plot(kind = 'bar', y = 'Web')
```

You should get the following output:

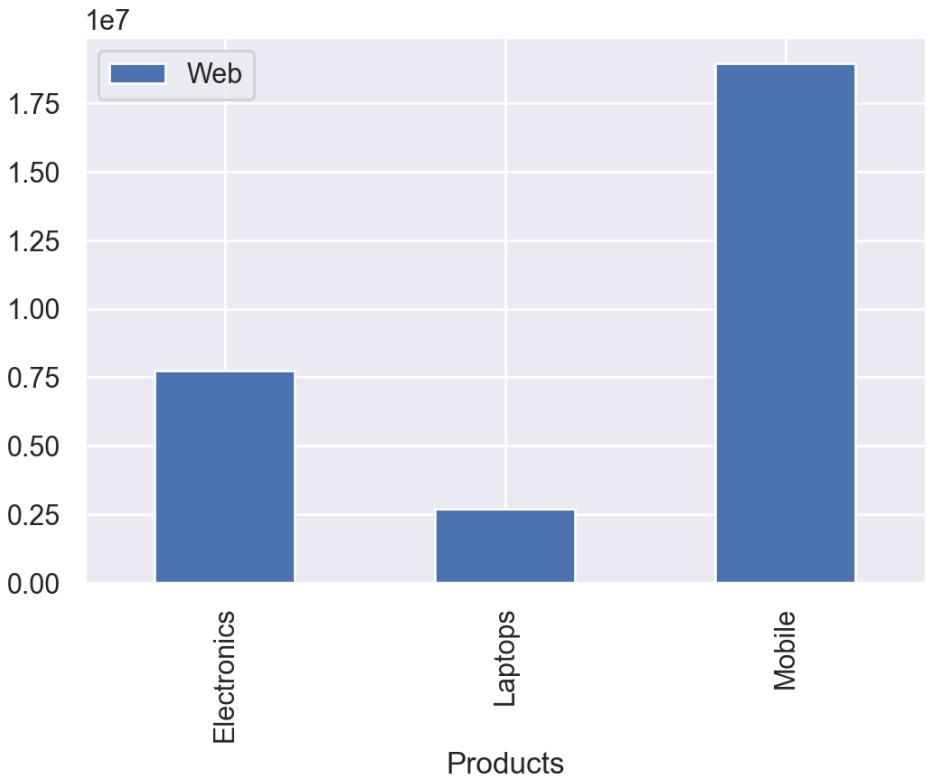

Figure 2.71: Bar chart across products and web

You can see that the **`Laptops`** category has the lowest viewership on the web.

10. Understand the relationships between the various columns in the DataFrame with the help of a pair plot. Use the following command:

```
sns.pairplot(ads, hue='Products')
```

This should give the following output:

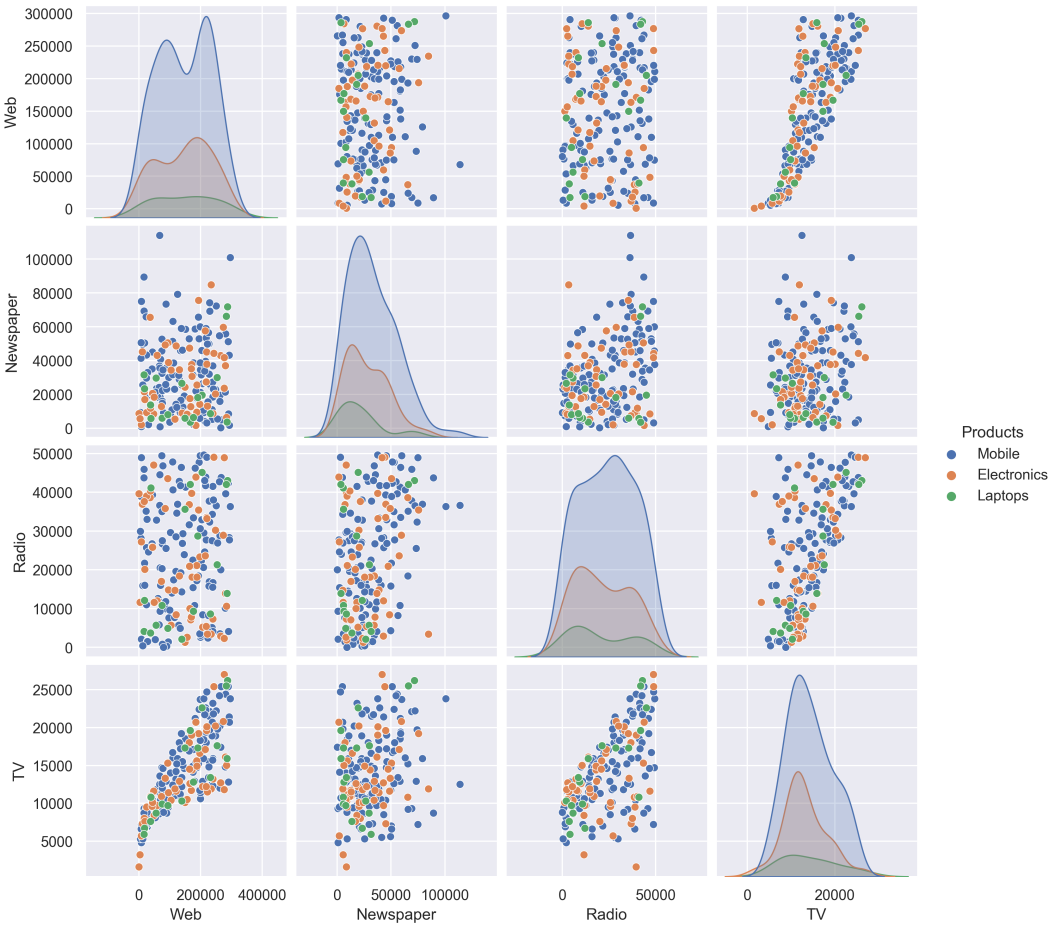

Figure 2.72: Output of pairplot of the ads feature

You can derive the following insights from the data: Looking at the charts in your pair plot you can see that **`TV`** and **`Radio`** have a positive correlation along with **`TV`** and **`Web`**. The correlation between **`TV`** and **`Newspaper`** is random; that is, you can't determine whether there is a positive correlation or a negative correlation.

CHAPTER 3: UNSUPERVISED LEARNING AND CUSTOMER SEGMENTATION

EXERCISE 3.02: TRADITIONAL SEGMENTATION OF MALL CUSTOMERS

1. Plot a histogram of the **`Income`** column using the DataFrame's **`plot`** method using the following code:

```
data0.Income.plot.hist()
plt.xlabel('Income')
plt.show()
```

You should see the following histogram:

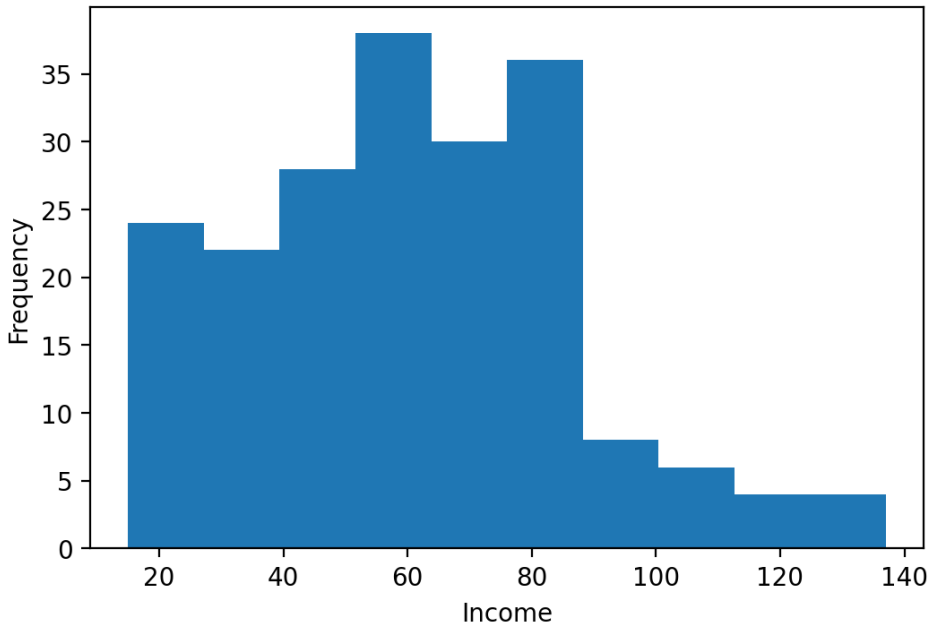

Figure 3.5: Histogram of the Income column

Beyond 90k, the frequency in the bins falls sharply and it seems that these customers can naturally be considered a separate group representing high-income customers. A good proportion of customers seems to lie in the 50k-90k range. These can be considered moderate-income customers. Customers earning less than 40k would be low-income customers. We can use these cutoffs to divide the customers into three groups, as in the following figure. The dotted segments denote the cutoffs/thresholds. The groups can be named **`Low Income`**, **`Moderate Income`**, and **`High Earners`**:

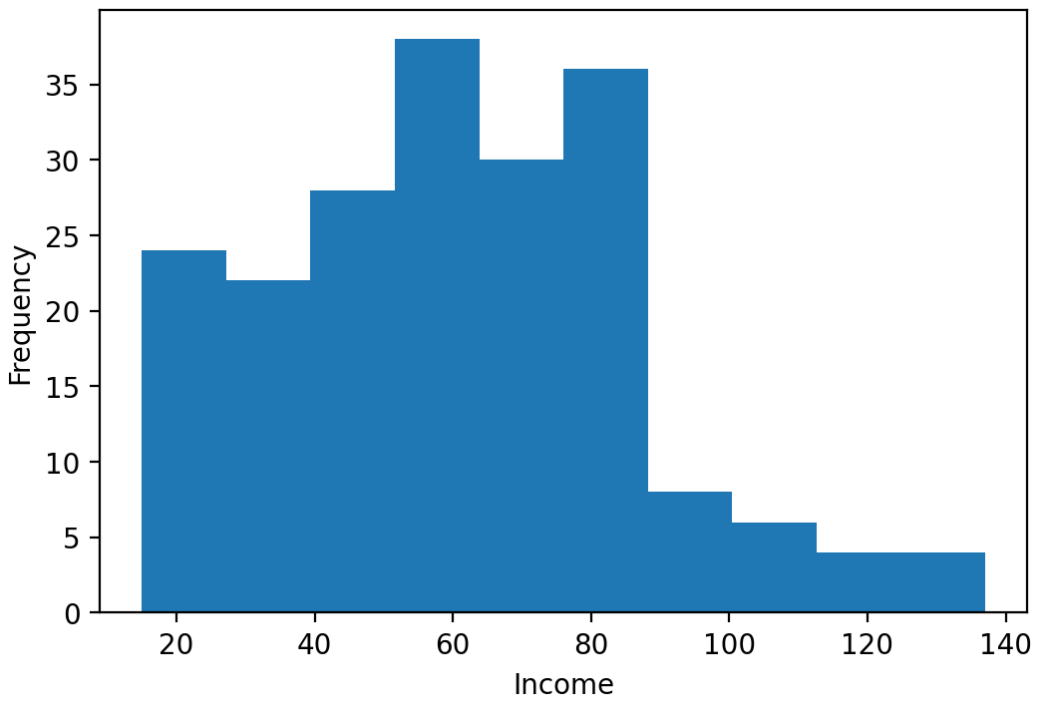

Figure 3.6: Segmentation of the customers based on income

EXERCISE 3.05: K-MEANS CLUSTERING ON MALL CUSTOMERS

2. Visualize the data using a scatter plot with **`Income`** and **`Spend_score`** on the *x* and *y* axes respectively with the following code:

```
data_scaled.plot.scatter(x='Income', y='Spend_score')
plt.show()
```

The resulting plot should look as follows:

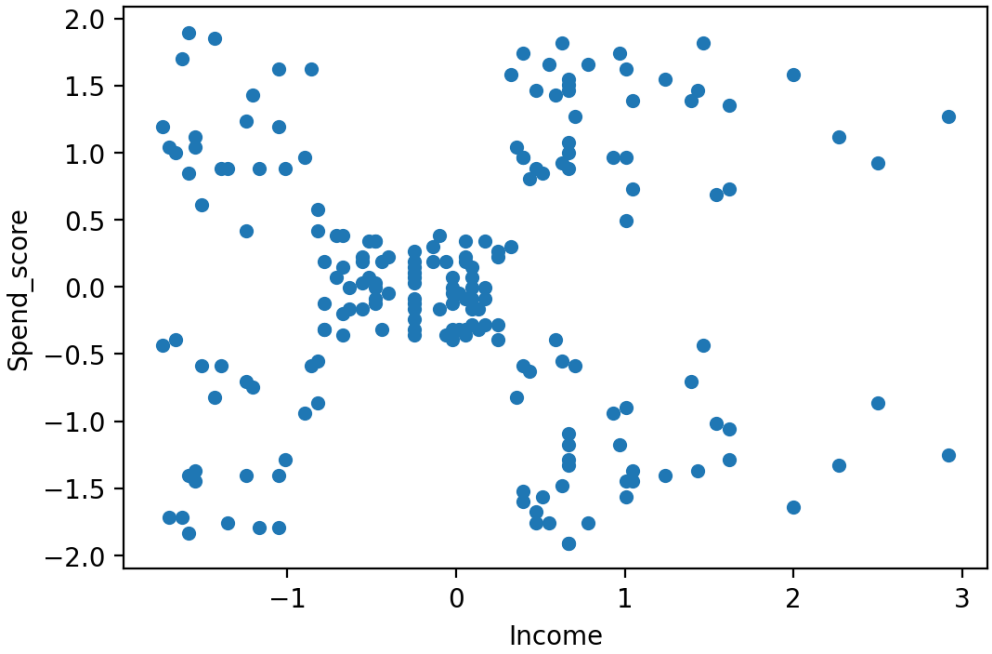

Figure 3.17: A scatter plot of income and spend score

From the plot, there are 5 natural clusters in the data. This tells us that we need to specify **5** as the number of clusters for the k-means algorithm.

5. Now you need to visualize it to see the points assigned to each cluster. Plot each cluster with a marker using the following code. You will subset the dataset for each cluster and use a dictionary to specify the marker for the cluster:

```
markers = ['x', '*', '.', '|', '_']

for clust in range(5):
    temp = data_scaled[data_scaled.Cluster == clust]
    plt.scatter(temp.Income, temp.Spend_score, \
                marker=markers[clust], \
                label="Cluster "+str(clust))
plt.xlabel('Income')
plt.ylabel('Spend_score')
plt.legend()
plt.show()
```

You will get the following output:

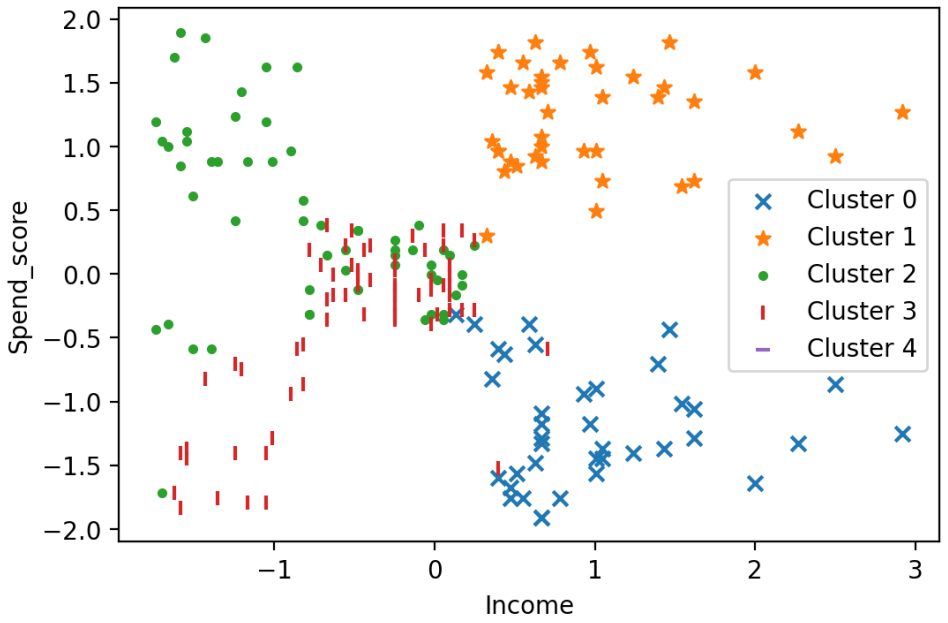

Figure 3.19: A plot of the data with the color/shape indicating which cluster each data point is assigned to

The clusters, represented by the different shapes on the scatter plot, seem to be aligned with the natural groups that we visually identified in *Figure 3.18*. The **`k-means`** algorithm did a good job of identifying the "natural" clusters in the data.

EXERCISE 3.06: DEALING WITH HIGH-DIMENSIONAL DATA

4. Visualize the clusters by using different markers and colors for the clusters on a scatter plot between **`pc1`** and **`pc2`** using the following code:

```
markers = ['x', '*', 'o','|']

for clust in range(4):
    temp = data_scaled[data_scaled.Cluster == clust]
    plt.scatter(temp.pc1, temp.pc2, marker=markers[clust], \
                label="Cluster "+str(clust))
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

The following plot should appear:

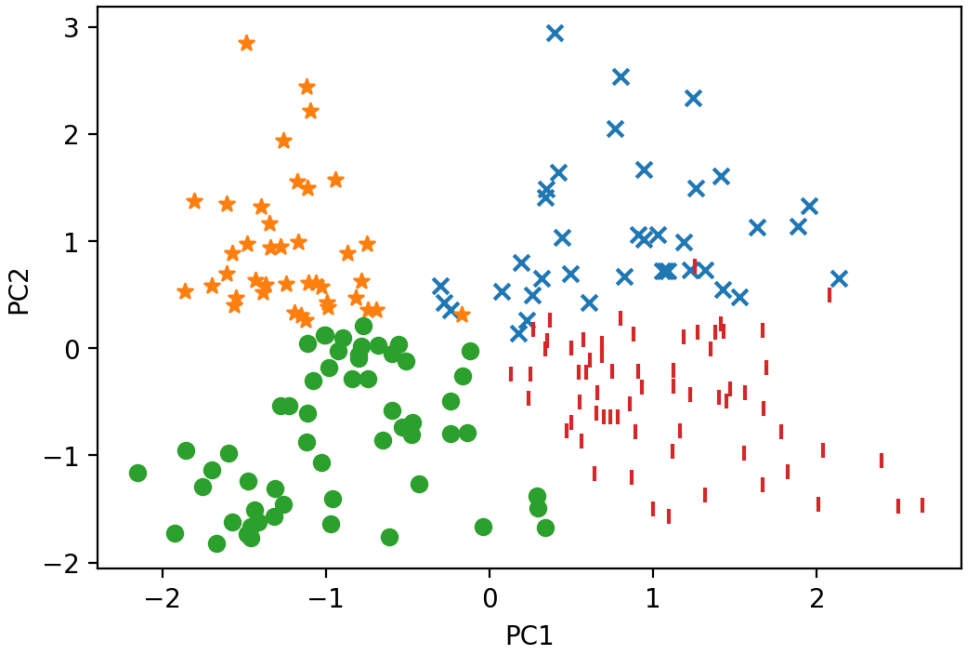

Figure 3.27: A plot of the data reduced to two dimensions denoting the three clusters

Note that the *x* and *y* axes here are the PCs, and therefore are not easily interpretable (as these are derived combinations of the original features). However, by visualizing the clusters, we can get a sense of how good the clusters are based on how much they overlap.

6. Next, visualize this information using bar plots. Check which features are the most differentiated for the clusters using the following code:

```
data0.groupby('Cluster')[['Age', 'Income', \
                          'Spend_score']].mean().plot.bar()
plt.show()
```

The output should be as follows:

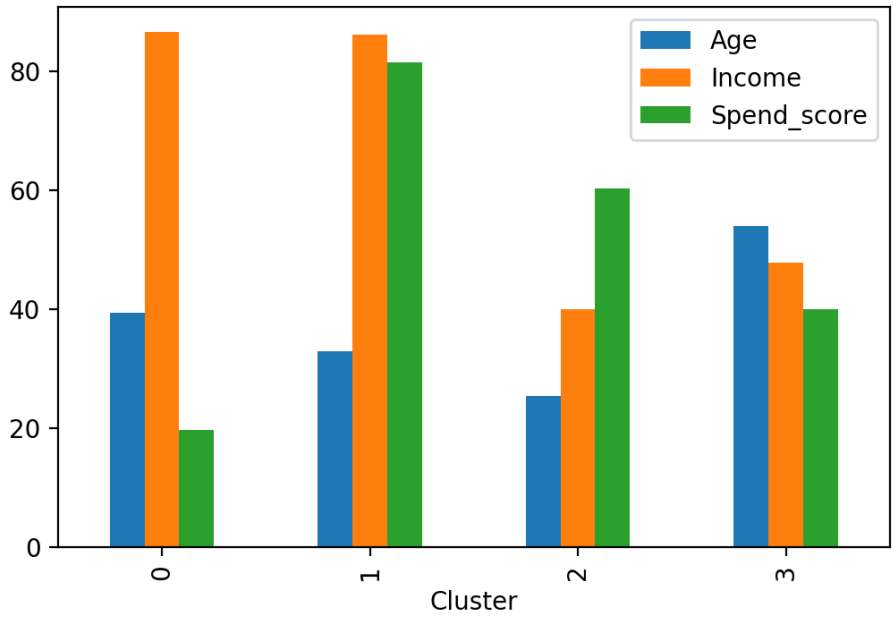

Figure 3.29: Bar plot to check features that are the most differentiated for the clusters

You can see that all three columns vary significantly across the clusters. You have cluster **0** with the highest average income but the lowest spend score and average age. You have cluster **2**, which is average in age, high on income, and the highest on spending. Let's describe these better in the next step.

ACTIVITY 3.01: BANK CUSTOMER SEGMENTATION FOR LOAN CAMPAIGN

6. Visualize the clusters by using different markers and colors for the clusters on a scatter plot between **`Income`** and **`CCAvg`**:

```
markers = ['x', '.', '_']

plt.figure(figsize=[8,5])
for clust in range(3):
    temp = bank0[bank0.Cluster == clust]
    plt.scatter(temp.Income, temp.CCAvg, \
                marker=markers[clust], \
                label="Cluster "+str(clust))

plt.xlabel('Income')
```

```
plt.ylabel('CCAvg')
plt.legend()
plt.show()
```

You should get the following output:

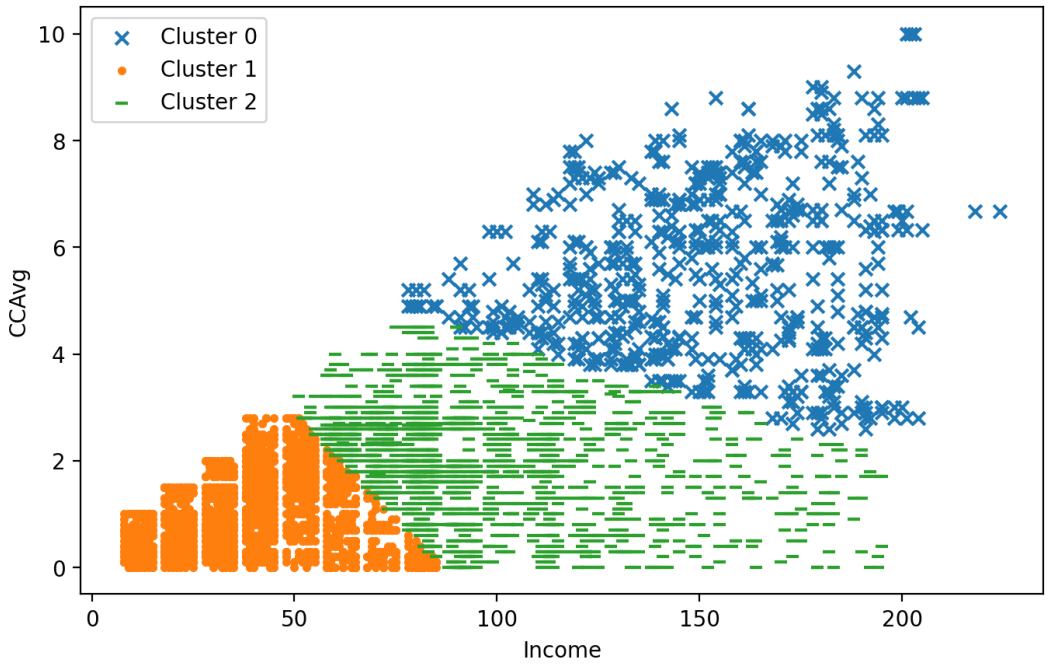

Figure 3.34: Visualizing the clusters using different markers

8. Perform a visual comparison of the clusters using the standardized values for **`Income`** and **`CCAvg`**:

```
bank0.groupby('Cluster')[['Income_scaled', 'CCAvg_scaled']]\
                        .mean().plot.bar()
plt.show()
```

The output should be as follows:

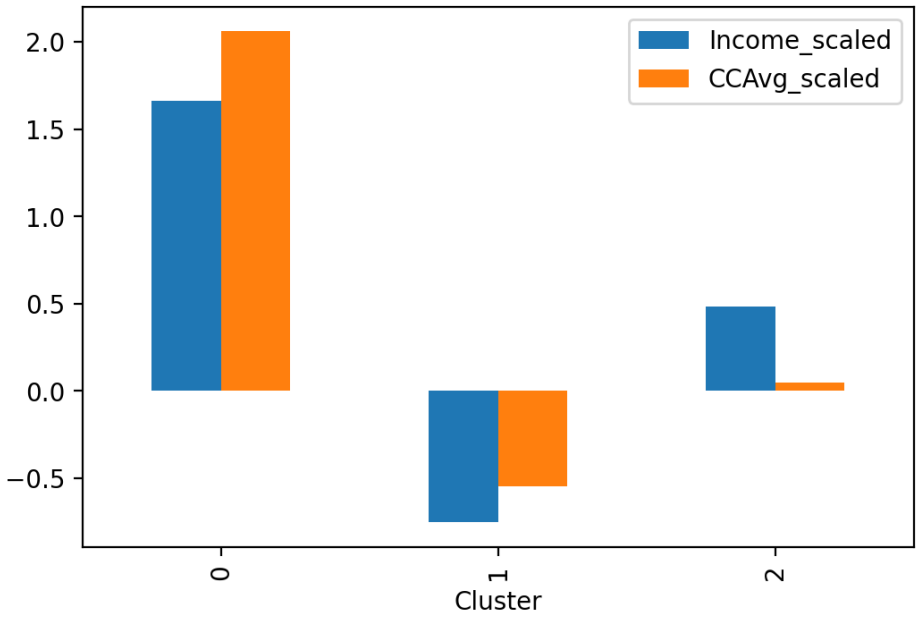

Figure 3.36: Visual comparison of the clusters using the standardized values for Income and CCAvg

In *Figure 3.36*, you can see the average values of the features against each cluster on a "standardized" scale. Notice how **`Cluster 0`** is closer to the average (**0** on the standard scale) for both **`Income`** and **`CCAvg`**. **`Cluster 2`** is much higher on both these variables, while **`Cluster 1`** has lower-than-average values for both variables.

ACTIVITY 3.02: BANK CUSTOMER SEGMENTATION WITH MULTIPLE FEATURES

5. Visualize the clusters by using different markers and colors for the clusters on a scatter plot between **`pc1`** and **`pc2`**:

```
markers = ['x', '.', '_']
plt.figure(figsize=[10,12])
for clust in range(3):
    temp = bank_scaled[bank_scaled.Cluster == clust]
    plt.scatter(temp.pc1, temp.pc2, marker=markers[clust], \
                label="Cluster "+str(clust))

plt.xlabel('PC1')
```

```
plt.ylabel('PC2')
plt.legend()
plt.show()
```

The plot should appear as follows:

Figure 3.39: A plot of the data reduced to two dimensions denoting the three clusters

> **NOTE**
>
> The shapes and colors of the segments you get may vary slightly from the preceding figure. This is because of some additional randomness in the process, introduced by the random cluster center initialization that is done by the k-means algorithm. The cluster number assigned to each group may also diff er. Unfortunately, the randomness remains even after setting the random seed and we can't control it. Nevertheless, while the assigned shapes may differ, the characteristics of the resulting clusters will not diff er.

In the preceding figure, we can see the three clusters that the machine learning algorithm identified. We can see that in this case, the clusters are not very clearly naturally separated, as is the case in many real-world situations.

CHAPTER 4: EVALUATING AND CHOOSING THE BEST SEGMENTATION APPROACH

EXERCISE 4.01: DATA STAGING AND VISUALIZATION

3. Plot a scatterplot of the **`Income`** and **`Spend_score`** fields using the following code. You will be performing clustering later using these two features as the criteria.

```
mall0.plot.scatter(x='Income', y='Spend_score')
plt.show()
```

The plot you get should look like the one shown here:

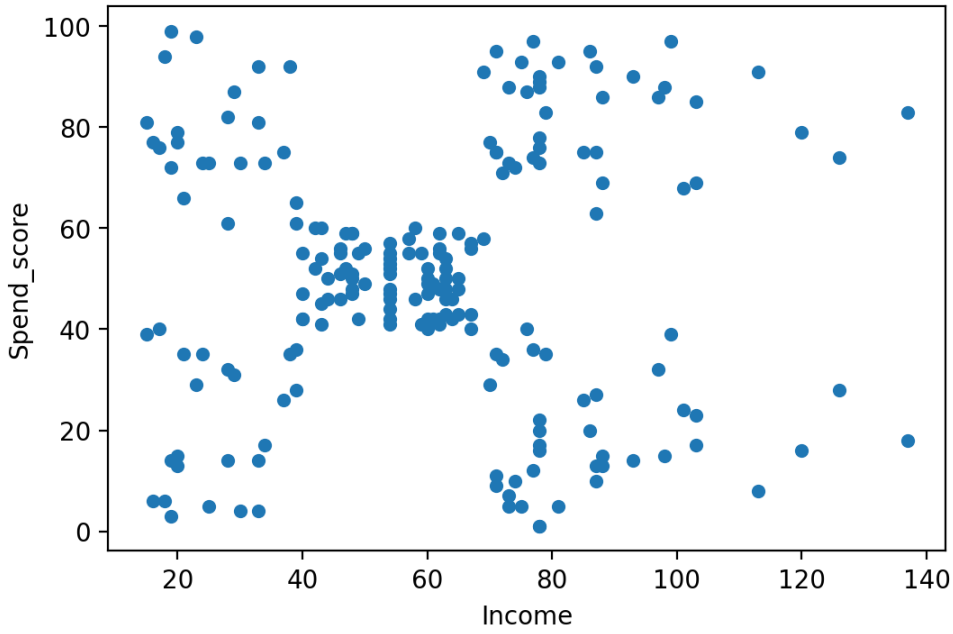

Figure 4.3: Scatterplot of Spend_score vs. Income

Figure 4.3 shows how the customers are distributed on the two attributes. We see that in the middle of the plot we have a bunch of customers with moderate income and moderate spend scores. These customers form a dense group that can be thought of as a natural cluster. Similarly, customers with low income and low spend scores also form a somewhat close group and can be thought of as a natural cluster. Customers with income above 70 and a spend score of less than 40 in the lower right area of the plot are interesting. These customers are thinly spread across and do not form a close group. However, these customers are significantly different from the other customers and can be considered a loose cluster.

EXERCISE 4.02: CHOOSING THE NUMBER OF CLUSTERS BASED ON VISUAL INSPECTION

3. Then, using a **`for`** loop, cluster the data using a different number of clusters, ranging from two to seven, and visualize the resulting plots obtained in a **`subplot`**. Use a separate **`for`** loop to plot each cluster in each **`subplot`**, so we can use different shapes for each cluster. Use the following snippet:

```
plt.figure(figsize=[12,8])

for n in range(2,8):
    model = KMeans(n_clusters=n, random_state=42)
    mall_scaled['Cluster']= model.fit_predict\
                            (mall_scaled[cluster_cols])

    plt.subplot(2,3, n-1)
    for clust in range(n):
        temp = mall_scaled[mall_scaled.Cluster == clust]
        plt.scatter(temp.Income, temp.Spend_score, \
                    marker=markers[clust], \
                    label="Cluster "+str(clust))
        plt.title("N clusters: "+str(n))
        plt.xlabel('Income')
        plt.ylabel('Spend_score')
        plt.legend()

plt.show()
```

> **NOTE**
>
> If the plots you are getting are overlapping or are not clear, you can try changing the dimensions of the plot by modifying them in the emboldened line of code.

You should see the following plots when this is done:

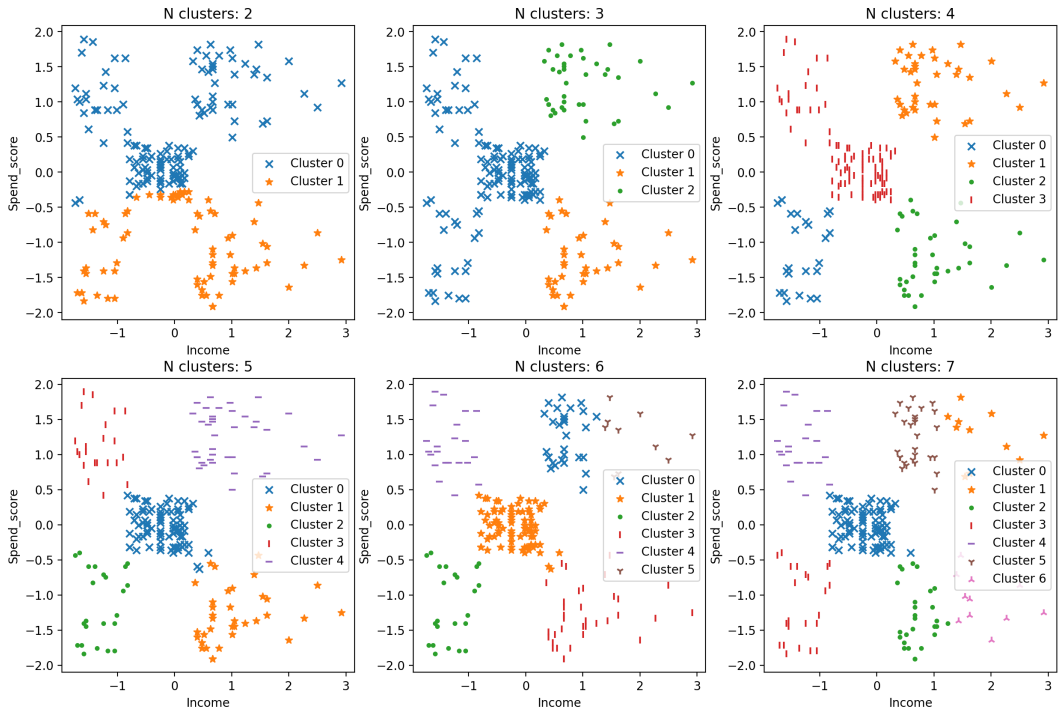

Figure 4.4: Scatterplots of Income and Spend_score with clusters progressing from 2 to 7

By observing the resulting plots, we can see that with too few clusters (**2**, **3** or **4**), we end up with clusters spanning sparse regions in between more densely packed regions. For instance, with **3** clusters, we see that we get one huge cluster for lower-income customers. On the other hand, with too many (**6** or **7**), we end up with clusters that border each other but do not seem separated by a region of sparseness. Five clusters seem to capture things very well with clusters that are non-overlapping and are fairly dense. Five is, therefore, the 'right' number of clusters. This is in line with our expectation that we built in *Exercise 4.01*, *Data Staging and Visualization*. This simple visual method is effective, but note that it is entirely subjective.

EXERCISE 4.03: DETERMINING THE NUMBER OF CLUSTERS USING THE ELBOW METHOD

4. Create the SSE/inertia plot as a line plot with the following code.

```
plt.figure(figsize=[7,5])
plt.plot(range(2,11), inertia_scores)
plt.title("SSE/Inertia vs. number of clusters")
plt.xlabel("Number of clusters: K")
plt.ylabel('SSE/Inertia')
plt.show()
```

You should get the following plot.

Figure 4.7: SSE plot for different values of k, showing an "elbow" (inflection point) at k=5

By observing the preceding plot, you will notice that there is an elbow in the plot at **`K=5`**. So, we take five as the optimal number of clusters, the best value of **`K`** for the **`KMeans`** algorithm. Before that, every additional cluster gives us big gains in reducing the sum of squared errors. Beyond five, we seem to be getting extremely low returns.

EXERCISE 4.04: MEAN-SHIFT CLUSTERING ON MALL CUSTOMERS

3. Visualize the clusters using a scatter plot.

```
markers = ['x', '*', '.', '|', '_', '1', '2']

plt.figure(figsize=[8,6])
for clust in range(mall_scaled.Cluster.nunique()):
    temp = mall_scaled[mall_scaled.Cluster == clust]
    plt.scatter(temp.Income, temp.Spend_score, \
                marker=markers[clust], \
                label="Cluster"+str(clust))

plt.xlabel("Income")
plt.ylabel("Spend_score")
plt.legend()
plt.show()
```

You should get the following plot:

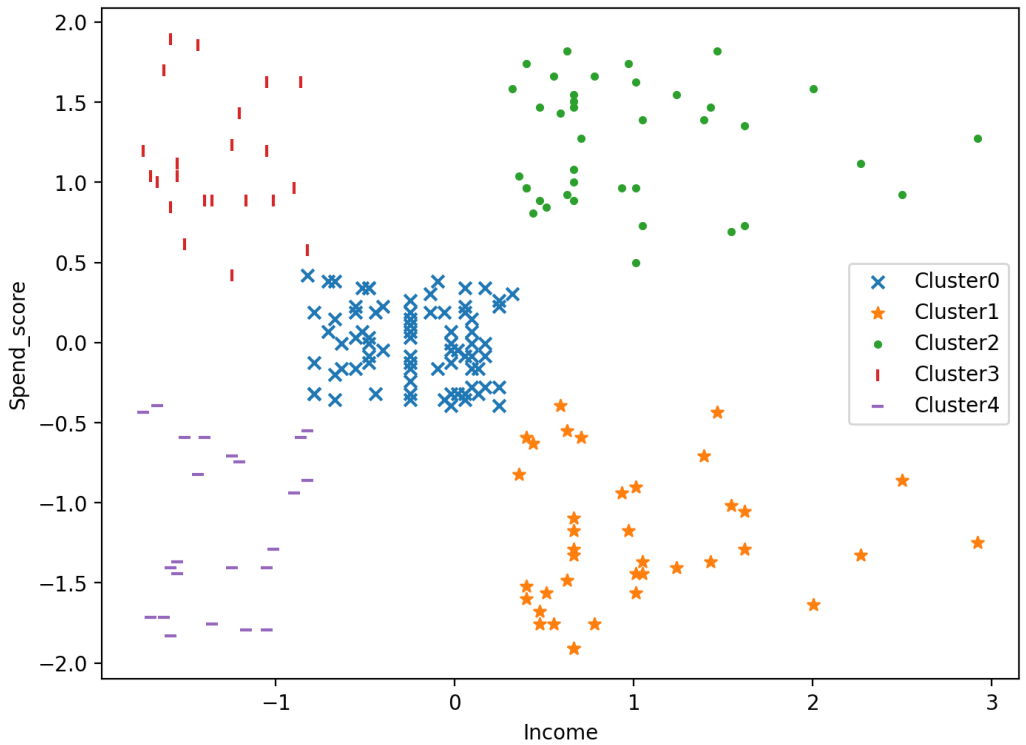

Figure 4.12: Clusters from mean-shift with bandwidth at 0.9

The model has found five unique clusters. They are very much aligned with the clusters you arrived at earlier using **K-means** where you specified **5** clusters. But notice that the clusters on the right have areas of very low density. The choice of bandwidth has led to such loose clusters.

5. Visualize the obtained clusters using a scatter plot.

```
plt.figure(figsize=[8,6])

for clust in range(mall_scaled.Cluster.nunique()):
    temp = mall_scaled[mall_scaled.Cluster == clust]
    plt.scatter(temp.Income, temp.Spend_score, \
                marker=markers[clust], \
                label="Cluster"+str(clust))
plt.xlabel("Income")
plt.ylabel("Spend_score")
plt.legend()
plt.show()
```

The output should look like the following plot:

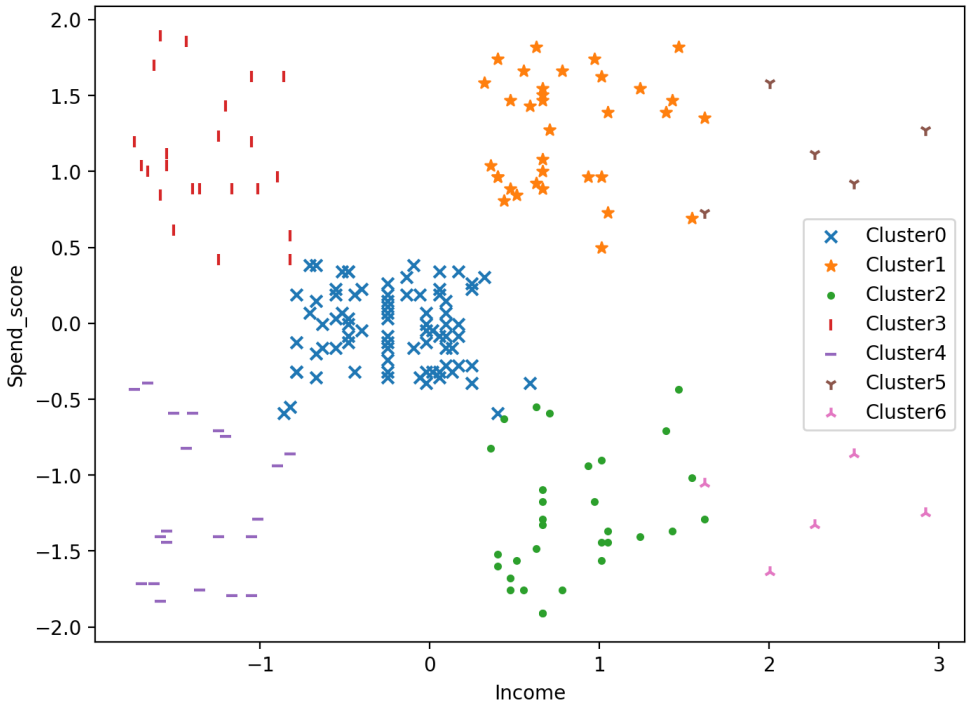

Figure 4.13: mean-shift with estimated bandwidth

8. Visualize the clusters obtained.

```
plt.figure(figsize=[8,6])

for clust in range(mall_scaled.Cluster.nunique()):
    temp = mall_scaled[mall_scaled.Cluster == clust]
    plt.scatter(temp.Income, temp.Spend_score, \
                marker=markers[clust], \
                label="Cluster"+str(clust))

plt.xlabel("Income")
plt.ylabel("Spend_score")
plt.legend()
plt.show()
```

The output should be as follows.

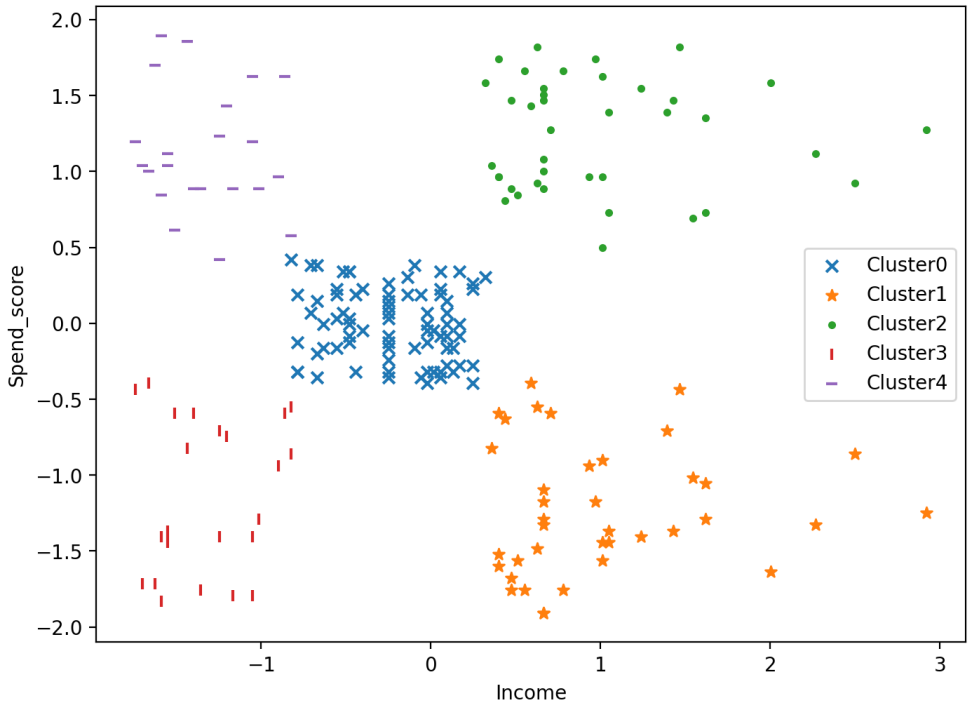

Figure 4.14: The data clustered using mean-shift clustering, quantile value of 0.15

You can see from *Figure 4.14* that you have obtained five clusters. This is the optimal number as you have seen from multiple approaches, including visual inspection.

EXERCISE 4.05: CLUSTERING DATA USING THE K-PROTOTYPES METHOD

4. To understand the obtained clusters, get the proportions of the different education levels in each cluster using the following code.

```
res = bank_scaled.groupby('Cluster')['Education']\
                 .value_counts(normalize=True)

res.unstack().plot.barh(figsize=[9,6])
plt.show()
```

You should get the following plot

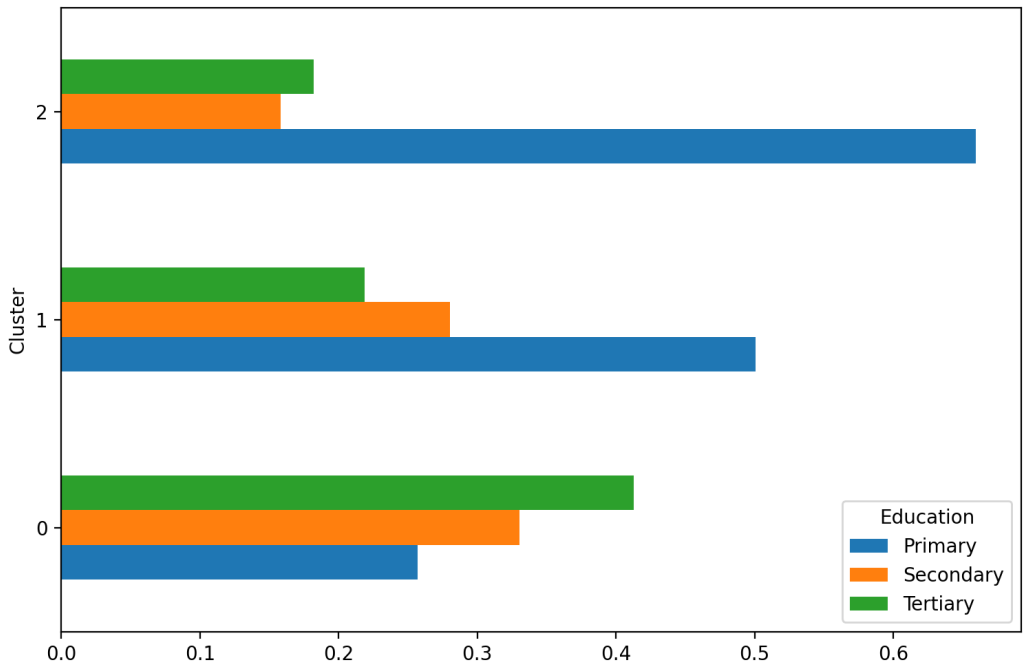

Figure 4.16: The proportions of customers of different educational levels in each cluster

You can see in *Figure 4.16* that **`cluster 2`** is dominated by customers with primary education. In **`cluster 1`**, the number of primary educated customers roughly equals the number of secondary and tertiary educated customers together. In **`cluster 0`**, customers with higher education (secondary or tertiary) significantly outnumber those with primary education.

EXERCISE 4.06: USING SILHOUETTE SCORE TO PICK OPTIMAL NUMBER OF CLUSTERS

4. Make a line plot with the silhouette score for the different values of `K`. Then identify the best value of `K` from the plot.

```
plt.figure(figsize=[7,5])
plt.plot(range(2,11), silhouette_scores)
plt.xlabel("Number of clusters: K")
plt.ylabel('Avg. Silhouette Score')
plt.show()
```

Your plot will look as follows:

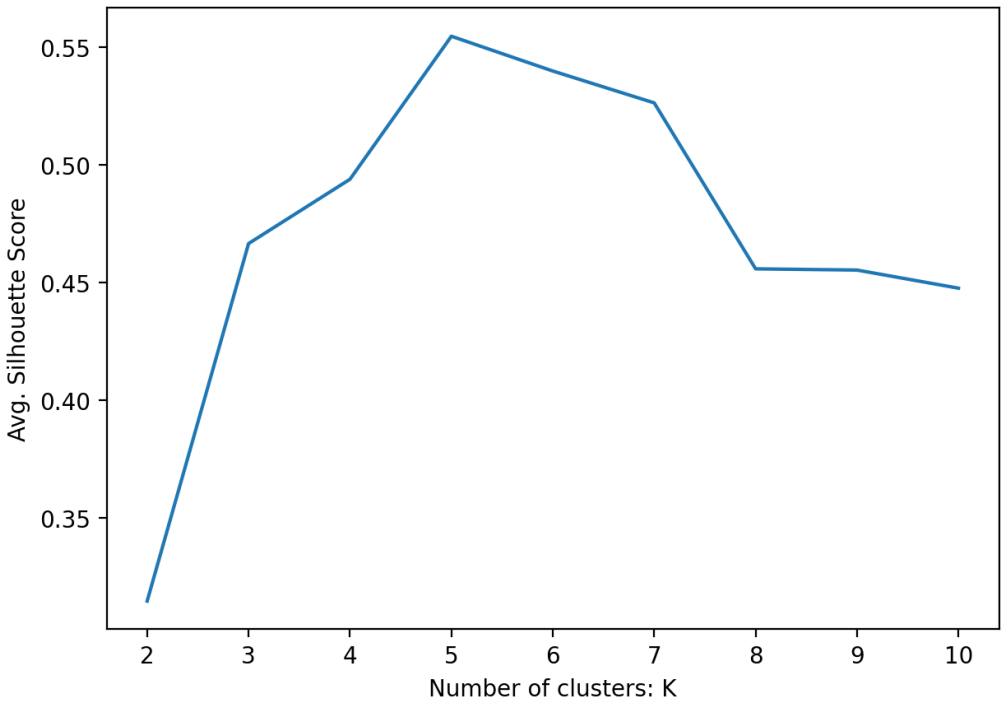

Figure 4.18: Average silhouette score vs. K. K=5 has the best value.

From the preceding plot, you can infer that `K=5` has the best **silhouette score** and is therefore the optimal number of clusters.

EXERCISE 4.07: USING A TRAIN-TEST SPLIT TO EVALUATE CLUSTERING PERFORMANCE

4. Visualize the predicted clusters on the test data using a scatter plot, marking the different clusters.

```
for clust in range(df_test.Cluster.nunique()):
    temp = df_test[df_test.Cluster == clust]
    plt.scatter(temp.Income, temp.Spend_score, \
                marker=markers[clust])
plt.xlabel("Income")
plt.ylabel("Spend_score")
plt.show()
```

You should get the following plot:

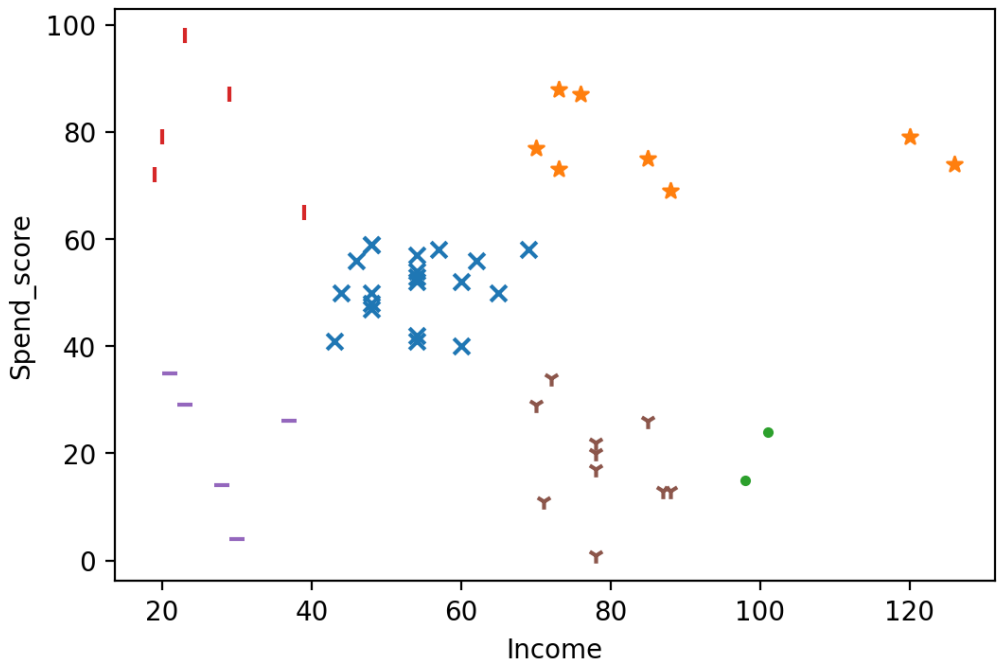

Figure 4.20: The clusters on the test data

ACTIVITY 4.01: OPTIMIZING A LUXURY CLOTHING BRAND'S MARKETING CAMPAIGN USING CLUSTERING

3. Visualize the data to get a good understanding of it. Since you are dealing with four dimensions, use **PCA** to reduce to two dimensions before plotting.

```
from sklearn import decomposition

pca = decomposition.PCA(n_components=2)
pca_res = pca.fit_transform(data_scaled[cluster_cols])

data_scaled['pc1'] = pca_res[:,0]
data_scaled['pc2'] = pca_res[:,1]

plt.figure(figsize=[8,5])
plt.scatter(data_scaled.pc1, data_scaled.pc2)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

The resulting plot should be as follows:

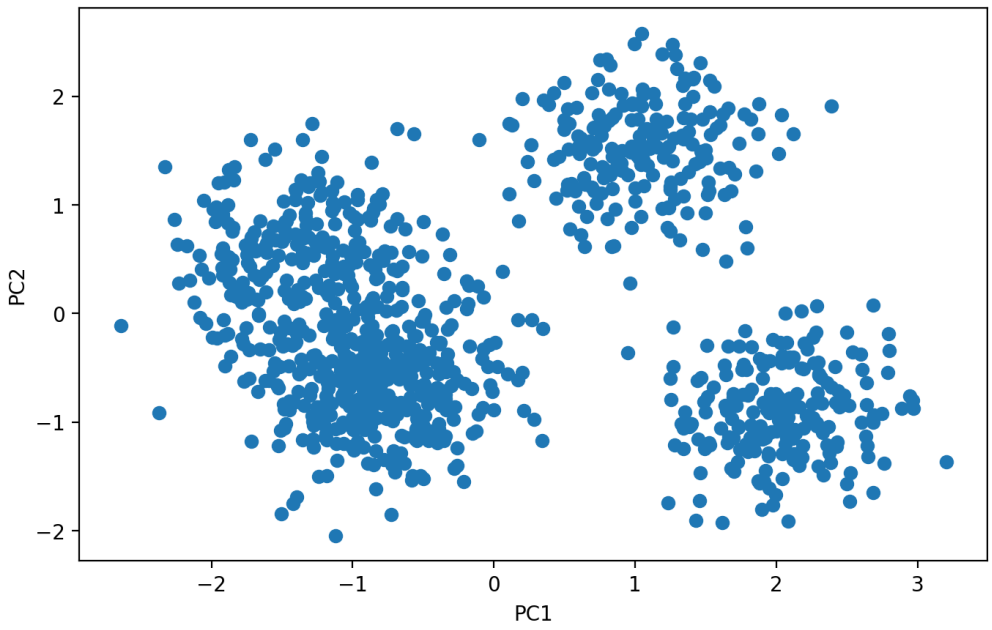

Figure 4.25: Scatterplot of the dimensionality reduced data

4. Visualize clustering for two through seven clusters:

```
from sklearn.cluster import KMeans

markers = ['x', '*', '.', '|', '_', '1', '2']

plt.figure(figsize=[15,10])
for n in range(2,8):
    model = KMeans(n_clusters=n, random_state=42)
    data_scaled['Cluster']= model.fit_predict\
                            (data_scaled[cluster_cols])

    plt.subplot(2,3, n-1)
    for clust in range(n):
        temp = data_scaled[data_scaled.Cluster == clust]
        plt.scatter(temp.pc1, temp.pc2, \
                    marker=markers[clust], \
                    label="Cluster "+str(clust))
        plt.xlabel("PC1")
        plt.ylabel("PC2")
        plt.legend()
        plt.title("N clusters: "+str(n))

plt.show()
```

This should result in the following plot:

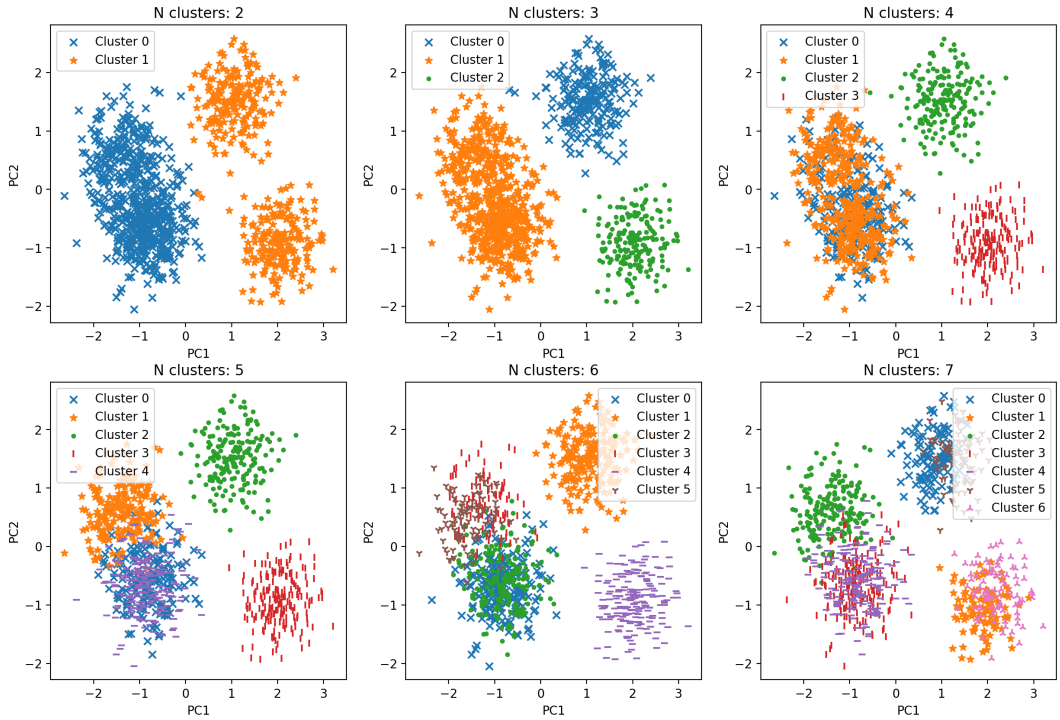

Figure 4.26: Resulting clusters for different number of clusters

From visual inspection, **3** seems to be the right number of clusters. Beyond **3**, the clusters overlap heavily.

5. Choosing clusters using **`elbow`** method - create a plot of the sum of squared errors and look for an elbow. Vary the number of clusters from **2** to **11**.

```
inertia_scores = []

for K in range(2,11):
    inertia = KMeans(n_clusters=K, random_state=42)\
              .fit(data_scaled).inertia_
    inertia_scores.append(inertia)

plt.figure(figsize=[7,5])
plt.plot(range(2,11), inertia_scores)
plt.xlabel("Number of clusters: K")
plt.ylabel('SSE/Inertia')
plt.show()
```

You should get the plot as below -

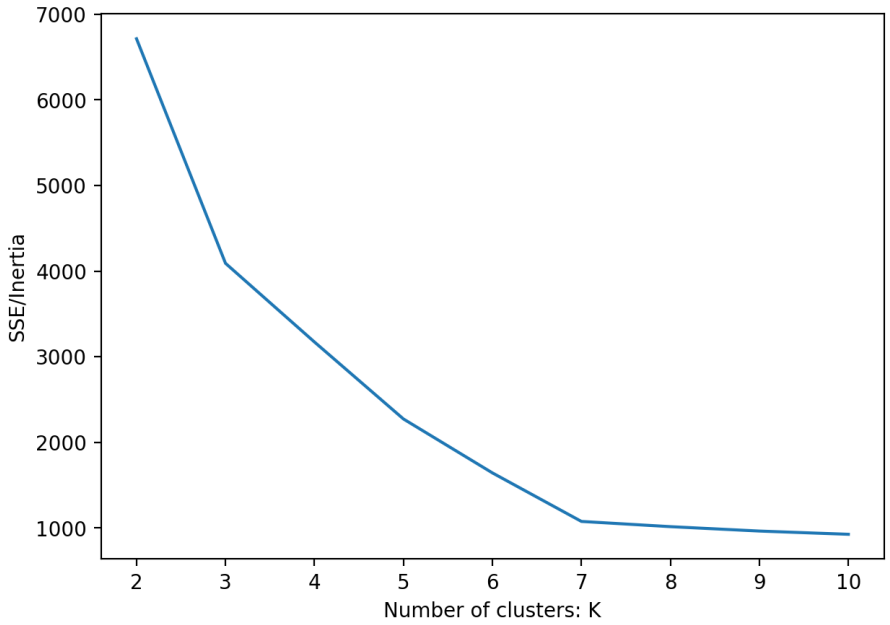

Figure 4.27: SSE/Inertia at different number of clusters

Notice that the elbow is at **`K = 5`**. The **`elbow`** method suggests using that five is the optimal number of clusters.

ACTIVITY 4.02: EVALUATING CLUSTERING ON CUSTOMER DATA

3. Perform k-means on the data. Identify the optimal number of clusters by using the silhouette score approach on the train data by plotting the score for different number of clusters, varying from **2** through **10**.

```
krange = list(range(2,11))
avg_silhouettes = []

for n in krange:
    model = cluster.KMeans(n_clusters=n, random_state=100)
    model.fit(X_train)
    cluster_assignments = model.predict(X_train)

    silhouette_avg = metrics.silhouette_score(X_train, \
                                              cluster_assignments)
```

```
        avg_silhouettes.append(silhouette_avg)

    plt.plot(krange, avg_silhouettes)
    plt.xlabel("Number of clusters")
    plt.ylabel("Average Silhouette Score")
    plt.show()
```

You should obtain the following plot:

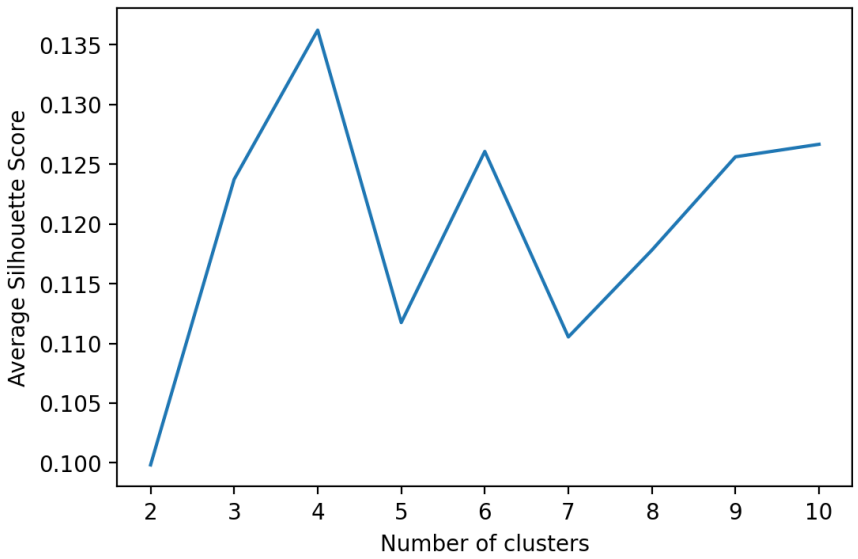

Figure 4.29: Silhouette scores for different number of clusters

From the plot for the Silhouette scores, you will observe that the maximum silhouette score is obtained at **k=4** and is around **0.135**.

6. Perform **k-modes** on the data. Identify the optimal number of clusters by using the silhouette score approach on the train data by plotting the score for different number of clusters, varying from **3** through **10**.

```
    from kmodes.kmodes import KModes

    krange = list(range(3,11))
    avg_silhouettes = []

    for n in krange:
        km = KModes(n_clusters=n, random_state=100)
        km.fit(X_train)
```

```
        kmode_labels = km.predict(X_train)
        kmode_silhouette = metrics.silhouette_score\
                           (X_train, kmode_labels)
        avg_silhouettes.append(kmode_silhouette)

plt.plot(krange, avg_silhouettes)
plt.xlabel("Number of clusters")
plt.ylabel("Average Silhouette Score")
plt.show()
```

The silhouette score plot is as following:

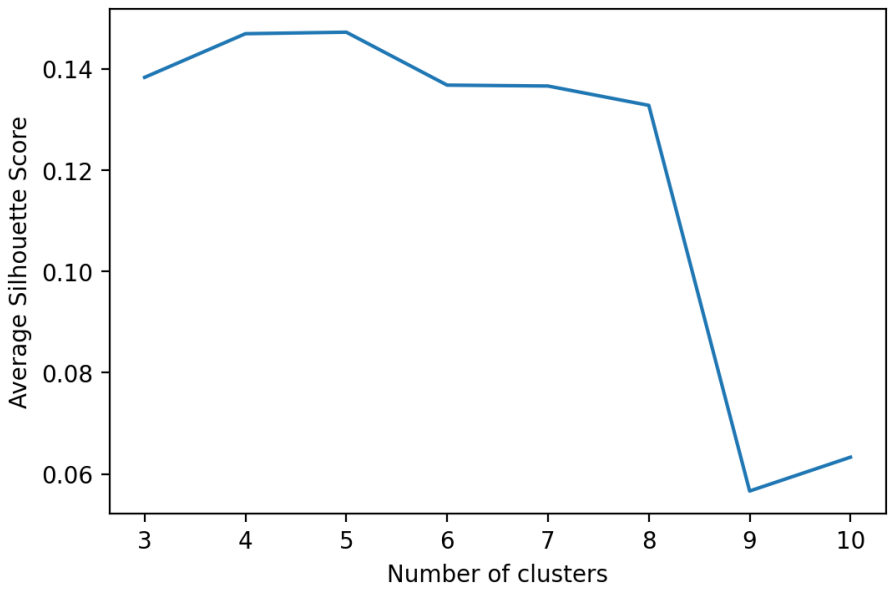

Figure 4.30: Silhouette scores for different K for K-modes

The silhouette score is practically the same for **4** and **5** clusters. We prefer **K=4** as we get the same silhouette score with a lower number of clusters.

CHAPTER 5: PREDICTING CUSTOMER REVENUE USING LINEAR REGRESSION

EXERCISE 5.01: PREDICTING SALES FROM ADVERTISING SPEND USING LINEAR REGRESSION

2. Visualize the association between **`TV`** and **`Sales`** through a scatter plot using the following code:

```
plt.scatter(advertising.TV, advertising.Sales, marker="+")
plt.xlabel("TV")
plt.ylabel("Sales")
plt.show()
```

You should get the following plot:

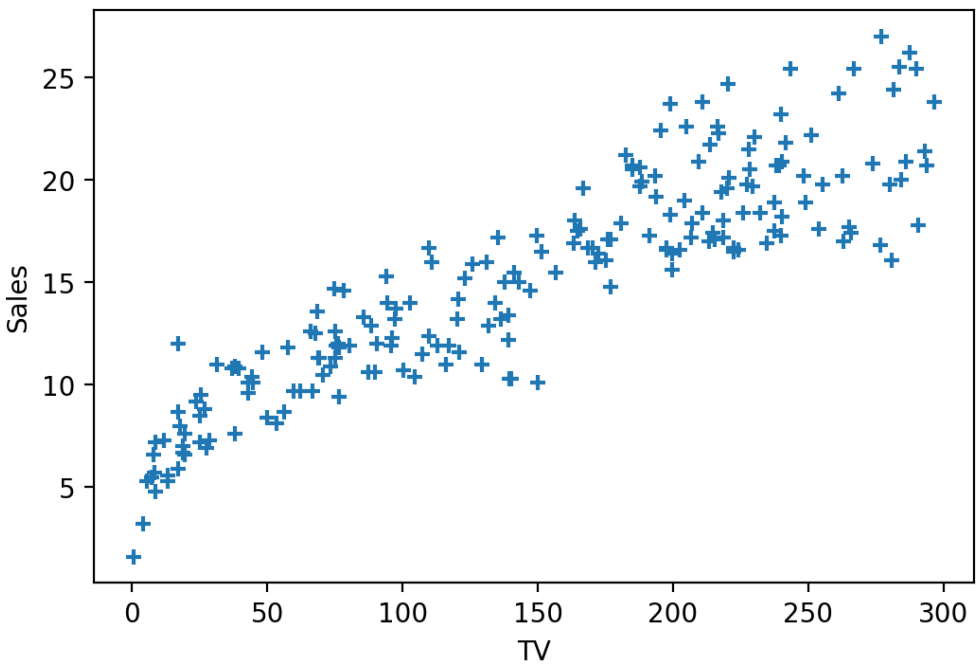

Figure 5.6: Sales versus TV spend.

As you can see from the preceding figure, **`TV`** spend clearly has a close association with **`Sales`** – sales seem to increase as TV spend increases. Let us use **`Sales`** to build the linear regression model.

6. Plot the predicted sales as a line over the scatter plot of **`Sales`** versus **`TV`** (using the simple line plot). This should help you assess how well the line fits the data and if it indeed is a good representation of the relationship:

```
plt.plot(advertising.TV, sales_pred,"k--")
plt.scatter(advertising.TV, advertising.Sales, marker='+')
plt.xlabel("TV")
plt.ylabel('Sales')
plt.show()
```

You should get the following output:

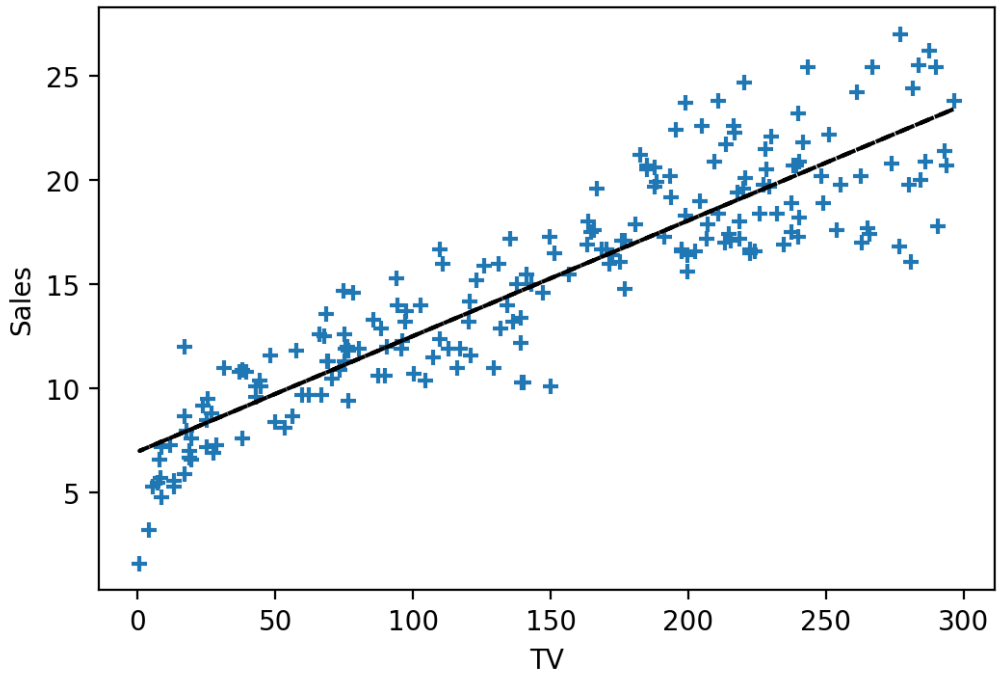

Figure 5.7: Regression line fitted on the data

As you can see, the regression line does a decent job of describing the relationship between sales and advertising spend on TV as a channel. From the line, you see that the sales increase very well together with the spend on TV advertising and that TV advertising can be a decent predictor of the expected sales. This tells HINA Inc. that this is a great channel for marketing investment.

EXERCISE 5.03: EXAMINING RELATIONSHIPS BETWEEN PREDICTORS AND THE OUTCOME

2. Using the **`plot`** method of the pandas DataFrame, make a scatter plot with **`days_since_first_purchase`** on the *x* axis and **`revenue_2020`** on the *y* axis to examine the relationship between them:

```
df.plot.scatter(x="days_since_first_purchase", \
                y="revenue_2020", \
                figsize=[6,6])
plt.show()
```

The output should be as follows:

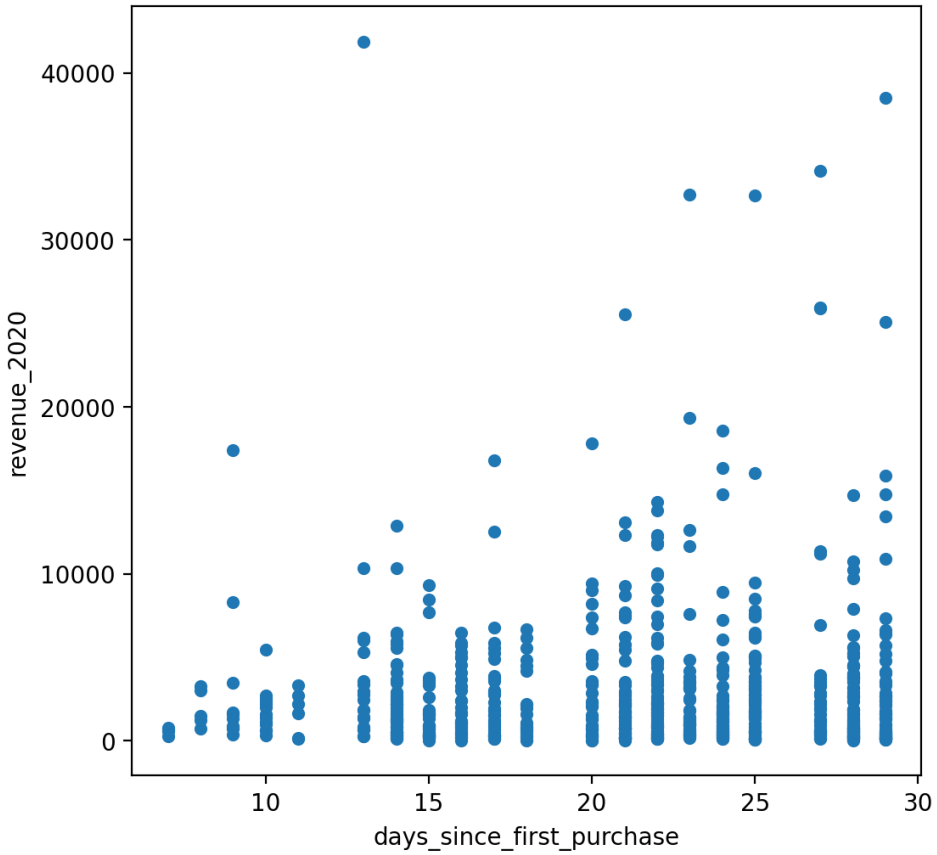

Figure 5.20: revenue_2020 versus days_since_first_purchase in a scatter plot

You can see that while there is not a very strong correlation between them, in general, we see higher instances of high revenue for customers with higher tenure (a higher value for **`days_since_first_purchase`**).

3. Using the **`pairplot`** function of the **`seaborn`** library, create pairwise scatter plots of all the features. Use the following code:

```
import seaborn as sns
sns.set_palette('Blues_r')
sns.pairplot(df)
plt.show()
```

You will get the following plot:

Figure 5.21: The seaborn pairplot of the entire dataset

In *Figure 5.21*, the diagonal shows a histogram for each variable, whereas each row shows the scatter plot between one variable and each other variable. The bottom row of the figures shows the scatter plots of the 2020 revenue (our outcome of interest) against each of the other variables. Because the data points are overlapping and there is a fair amount of variance, the relationships do not look very clear in the visualizations. Spend some time on this image and understand the associations in the data.

4. Using the **`pairplot`** function and the **`y_vars`** parameter, limit the view to the row for your target variable, that is, **`revenue_2020`**:

```
sns.pairplot(df, x_vars=df.columns, y_vars="revenue_2020")
plt.show()
```

You should get the following output:

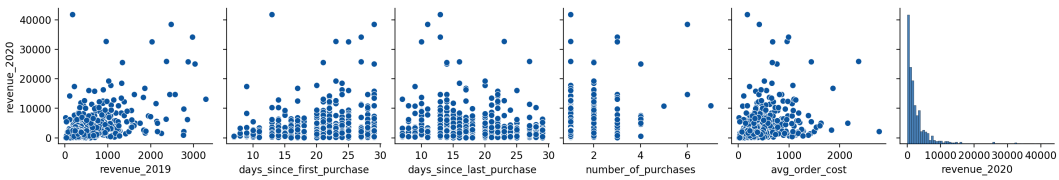

Figure 5.22: Pairplot limited to include the target variable

From this view focused on the associations between **`revenue_2020`** and the independent variables, you visually assess that none of the predictors have a strong relationship with the customer revenue for 2020. The strongest relationship seems to be with **`revenue_2019`** and it is a positive relationship.

EXERCISE 5.04: BUILDING A LINEAR MODEL PREDICTING CUSTOMER SPEND

11. You can plot the model's predictions on the test set against the true value. First, import **`matplotlib`**, and make a scatter plot of the model predictions on **`X_test`** against **`y_test`**. Limit the *x* and *y* axes to a maximum value of **`10,000`** so that we get a better view of where most of the data points lie. Finally, add a line with slope **`1`**, which will serve as your reference—if all the points lie on this line, it means you have a perfect relationship between your predictions and the true answer:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(model.predict(X_test), y_test)
plt.xlim(0,10000)
plt.ylim(0,10000)
plt.plot([0, 10000], [0, 10000], 'r-')
plt.xlabel('Model Predictions')
plt.ylabel('True Value')
plt.show()
```

Your plot will look as follows:

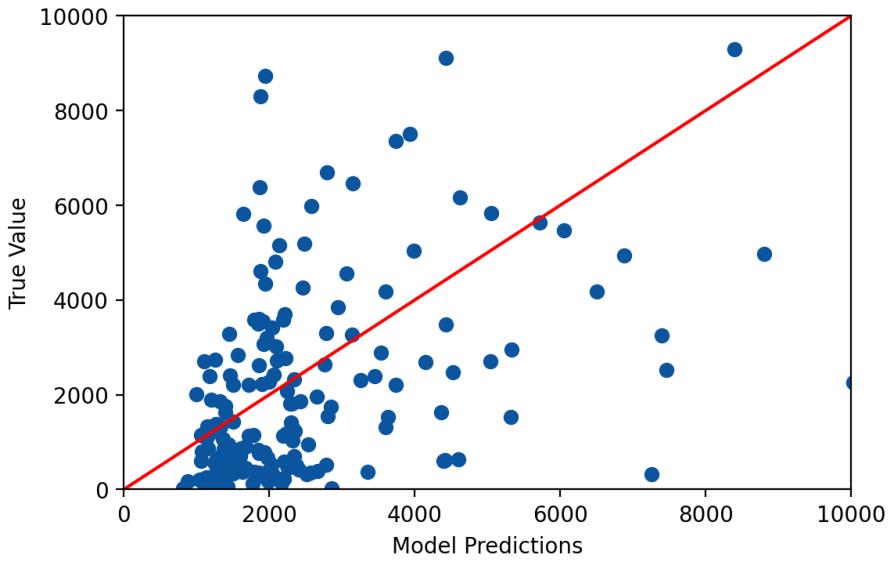

Figure 5.31: The model predictions plotted against the true values

In the preceding plot, the line indicates where points would lie if the prediction was the same as the true value. Since many of your points are quite far from the line, this indicates that the model is not completely accurate. However, there does seem to be some relationship, with higher model predictions having higher true values.

ACTIVITY 5.01: EXAMINING THE RELATIONSHIP BETWEEN STORE LOCATION AND REVENUE

2. Create a scatter plot between **`median_income`** and the **`revenue`** of the store using the **`plot`** method:

```
df.plot.scatter("median_income", 'revenue', figsize=[5,5])
plt.show()
```

The output should look as follows:

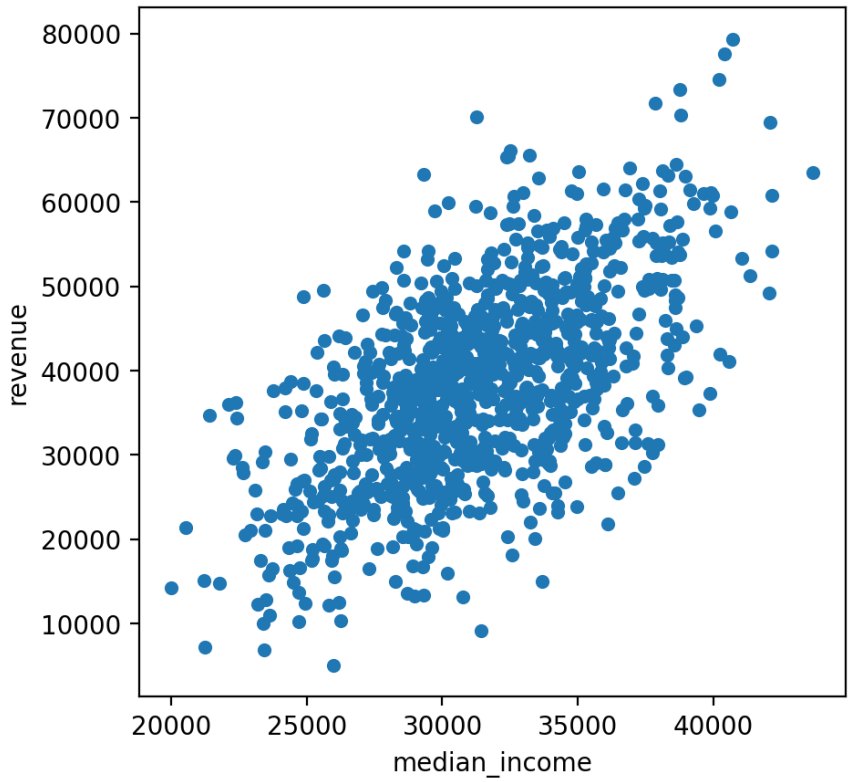

Figure 5.36: Scatter plot of median_income and revenue

There seems to be a decent association between the two variables. Let us examine the other associations in the data.

3. Use seaborn's **`pairplot`** function to visualize the data and its relationships:

```
sns.set_palette('Blues_r')
sns.pairplot(df)
plt.show()
```

The plots should appear as shown here:

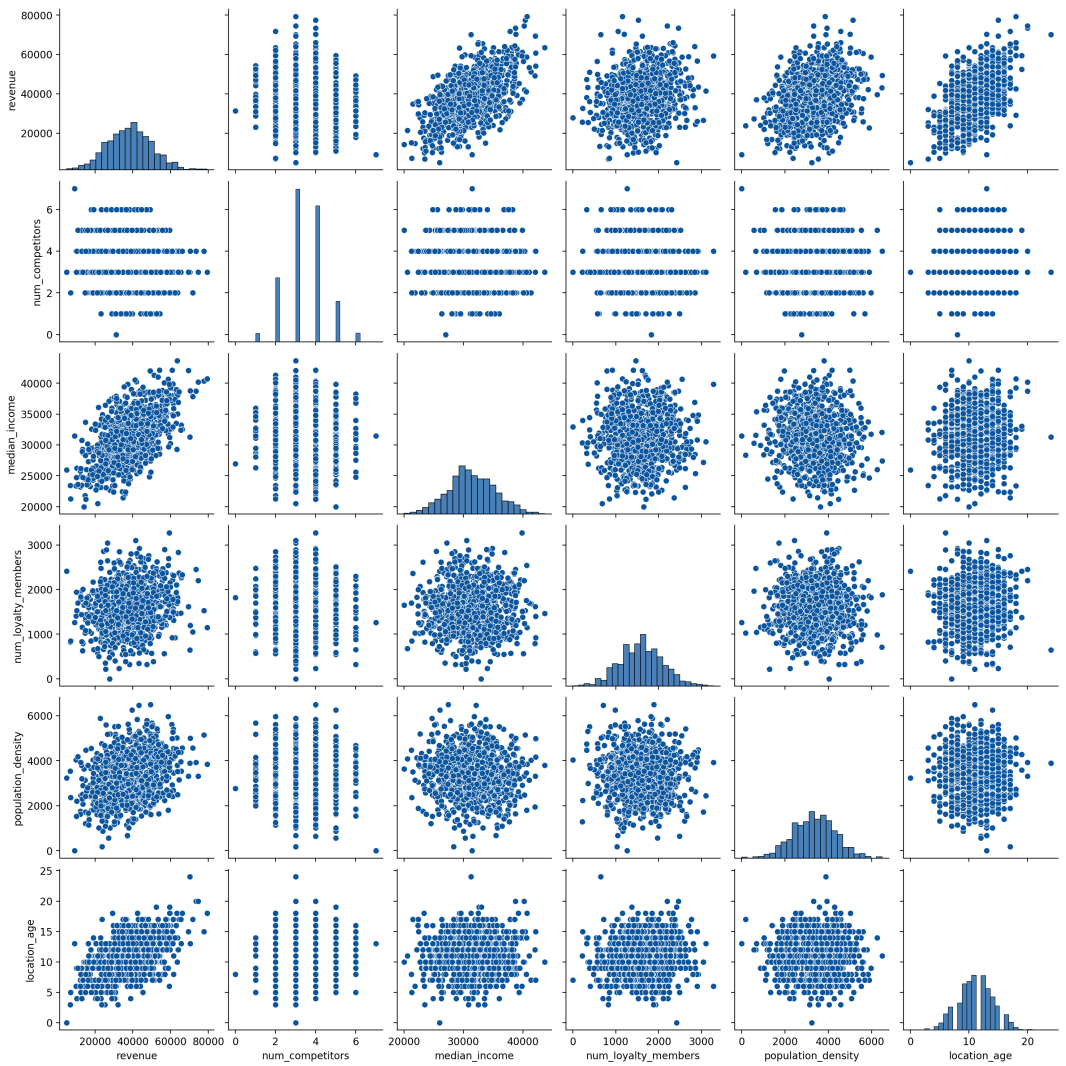

Figure 5.37: The seaborn pairplot of the entire dataset

The pairplot produces scatter plots for all possible combinations of variables. Notice that there are only a few cases where there is any decent association between any pairs of variables. The strongest relations seem to be between **`revenue`** and **`location_age`**, and **`revenue`** and **`median_income`**.

4. Using the **`y_vars`** parameter, plot the row for associations with the **`revenue`** variable:

```
sns.pairplot(df,x_vars=df.columns, y_vars="revenue")
plt.show()
```

You should get the following plot:

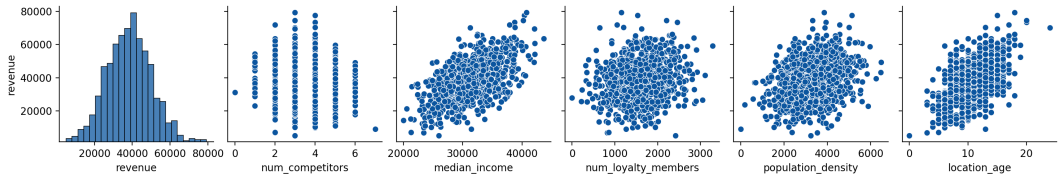

Figure 5.38: Associations with revenue

From this view focused on revenue, you see that **`revenue`** has decent associations with the **`location_age`** and **`median_income`** variables. A numeric value will help identify which of them is stronger.

ACTIVITY 5.02: PREDICTING STORE REVENUE USING LINEAR REGRESSION

8. Plot the model predictions versus the true values on the test data:

```
plt.scatter(model.predict(X_test),y_test)
plt.xlabel('Model Predictions')
plt.ylabel('True Value')
plt.plot([0, 100000], [0, 100000], 'k-', color = 'red')
plt.show()
```

The result should be like the following:

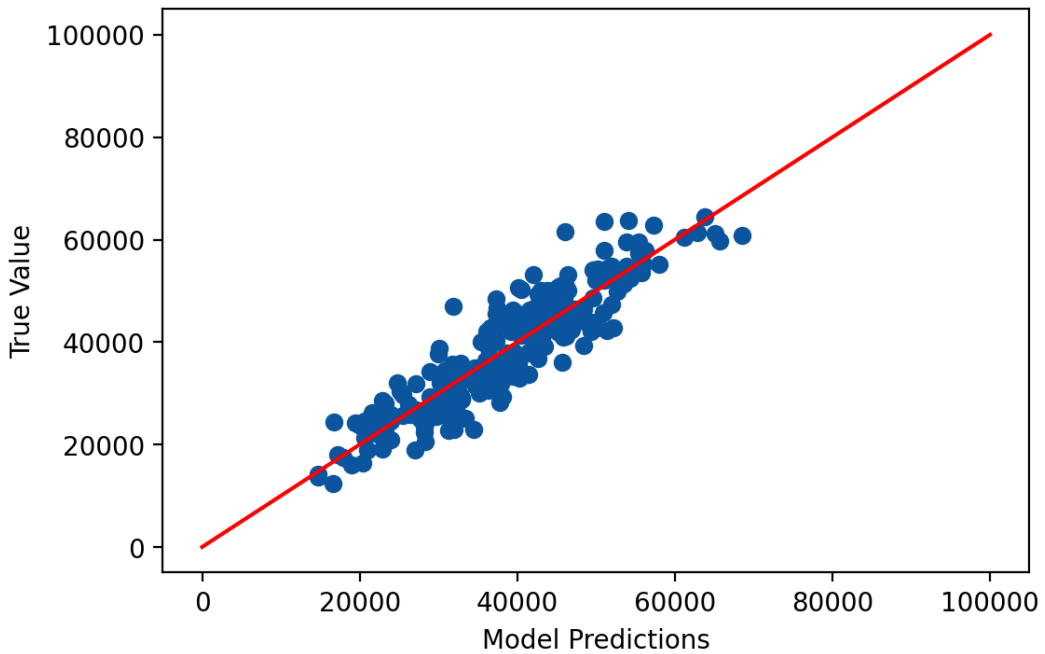

Figure 5.41: True values versus predictions

From the plot, it is evident that there is a very strong association between the predicted and actual values. This tells us that the model is doing a good job at predicting the revenue values for the stores. A correlation value will quantify this association.

CHAPTER 6: MORE TOOLS AND TECHNIQUES FOR EVALUATING REGRESSION MODELS

EXERCISE 6.03: USING TREE-BASED REGRESSION MODELS TO CAPTURE NON-LINEAR TRENDS

8. Create a scatter plot with the test data and, on top of it, plot the predictions from the linear regression model for the range of ages. Plot with **`color='r'`** and **`linewidth=5`** to make it easier to see:

```
plt.scatter(X_test.age.tolist(), y_test.tolist(),color='blue')
plt.plot(ages,model.predict(ages), color='r', linewidth=5, \
         label="Linear Regression")
plt.xlabel("age")
plt.ylabel("spend")
plt.show()
```

The following plot shows the predictions of the linear regression model across the **`age`** range plotted on top of the actual data points:

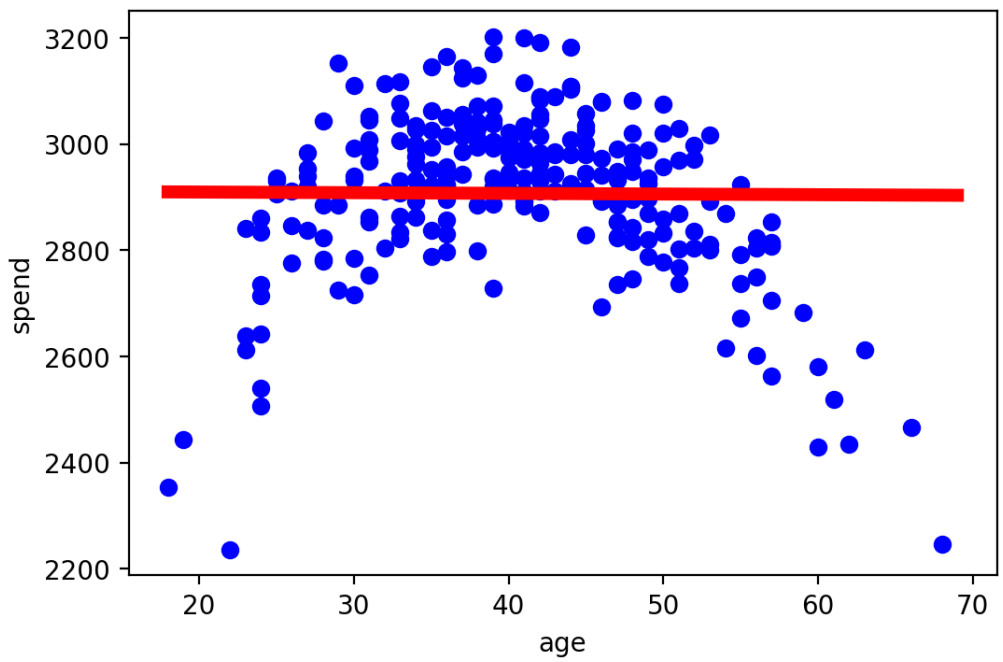

Figure 6.14: The predictions of the linear regression model

You can see that the linear regression model just shows a flat line across the ages; it is unable to capture the fact that people aged around 40 spend more, while people younger and older than 40 spend less.

9. Create another scatter plot with the test data, this time plotting the predictions of the **max2_tree** model on top with **color='r'** and **linewidth=5**:

```
plt.scatter(X_test.age.tolist(), y_test.tolist(),color='blue')
plt.plot(ages,max2_tree_model.predict(ages), \
         color='r',linewidth=5,label="Tree with max depth 2")
plt.xlabel("age")
plt.ylabel("spend")
plt.show()
```

The following plot shows the predictions of the regression tree model with **max_depth** of **2** across the **age** range plotted on top of the actual data points:

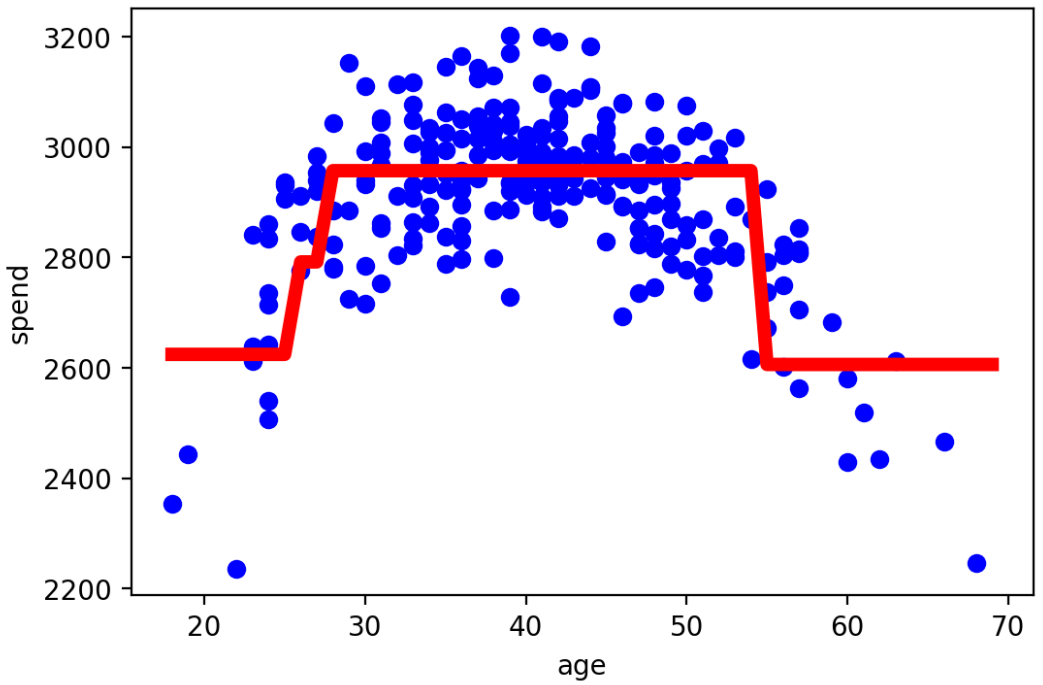

Figure 6.15: The predictions of the regression tree model with max_depth of 2

This model does a better job of capturing the relationship, though it does not capture the sharp decline in the oldest or youngest populations.

10. Create one more scatter plot with the test data, this time plotting the predictions of the **max5_tree** model on top with **color='r'** and **linewidth=5**:

```
plt.scatter(X_test.age.tolist(), y_test.tolist(),color='blue')
plt.plot(ages,max5_tree_model.predict(ages), color='r',\
         linewidth=5, label="Tree with max depth 5")
plt.xlabel("age")
plt.ylabel("spend")
plt.savefig("B16922_06_16.png", dpi=200, bbox_inches = "tight")
plt.show()
```

The following plot shows the predictions of the regression tree model with **max_depth** of **5** across the **age** range plotted on top of the actual data points:

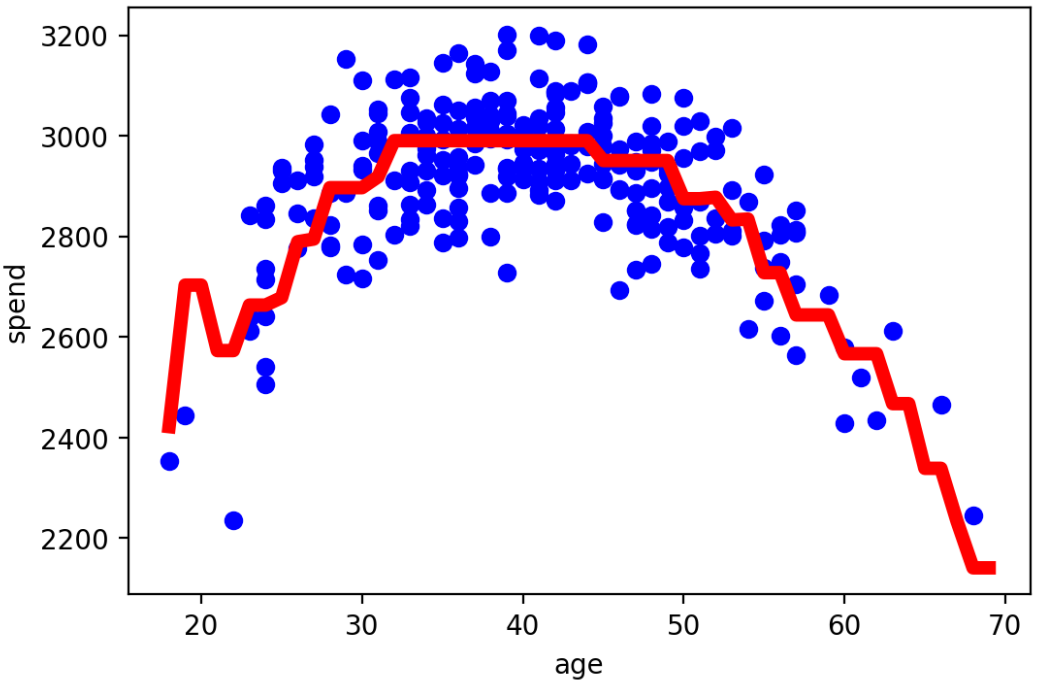

Figure 6.16: The predictions of the regression tree model with max_depth of 5

This model does an even better job of capturing the relationship, properly capturing a sharp decline in the oldest or the youngest population.

13. Create another scatter plot with the test data, this time plotting the predictions of the **max2_forest_model** model on top with **color='r'** and **linewidth=5**:

```
plt.scatter(X_test.age.tolist(), y_test.tolist(),color='blue')
plt.plot(ages,max2_forest_model.predict(ages), color='r',\
         linewidth=5, label="Forest with max depth 2")
plt.xlabel("age")
plt.ylabel("spend")
plt.show()
```

The following plot shows the predictions of the random forest model with **max_depth** of **2** across the **age** range plotted on top of the actual data points:

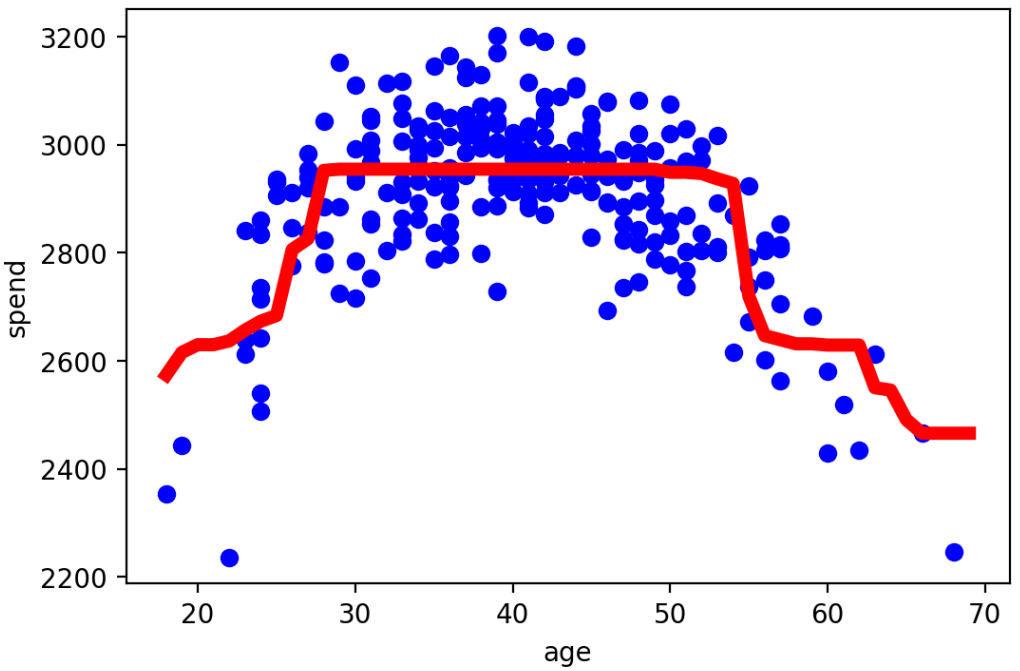

Figure 6.17: The predictions of the random forest model with max_depth of 2

We can see that this model captures the data trend better than the decision tree, but still doesn't quite capture the trend at the very high or low ends of our range.

14. Create another scatter plot with the test data, this time plotting the predictions of the **max2_forest_model** model on top with **color='r'** and **linewidth=5**:

```
plt.scatter(X_test.age.tolist(), y_test.tolist(),color='blue')
plt.plot(ages,max5_forest_model.predict(ages), color='r',\
         linewidth=5, label="Forest with max depth 5")
plt.xlabel("age")
plt.ylabel("spend")
plt.show()
```

The following plot shows the predictions of the random forest model with **max_depth** of **5** across the **age** range plotted on top of the actual data points:

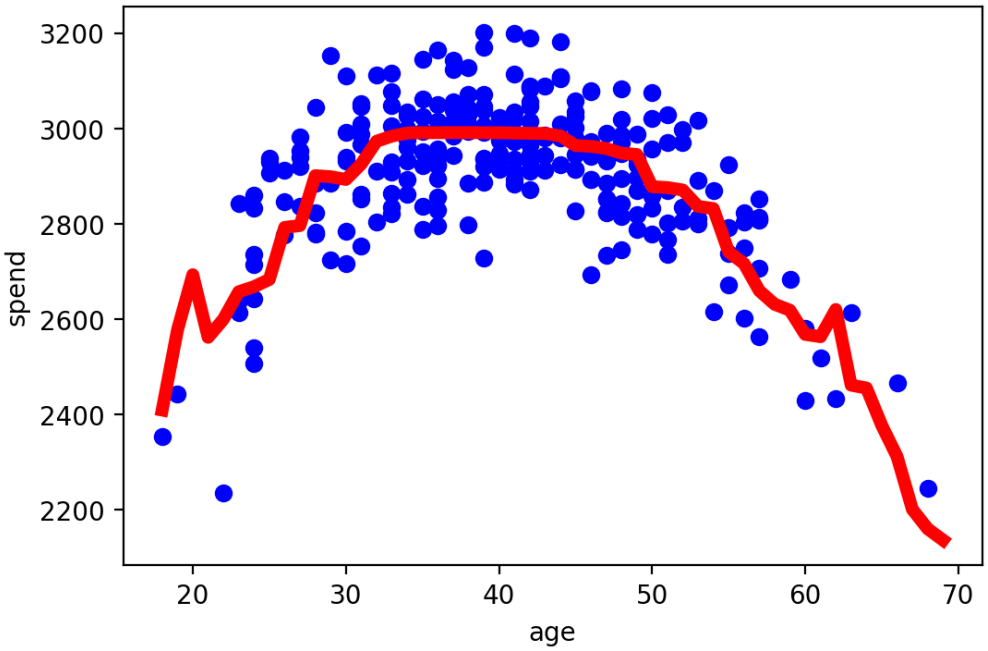

Figure 6.18: The predictions of the random forest model with max_depth of 5

Again, in the model, the greater maximum depth does an even better job of capturing the relationship, properly capturing the sharp decline in the oldest and youngest population groups.

The preceding results can easily be clubbed together to create the plot shown here, which presents a nice comparison of using different **`max_depth`** attributes while training the random forest model.

The code used to generate the plot is given here:

```
plt.figure(figsize=(12,8))
plt.scatter(X_test.age.tolist(), y_test.tolist())
plt.plot(ages,model.predict(ages), color='r', linewidth=5, \
         label="Linear Regression")
plt.plot(ages,max2_tree_model.pr=edict(ages), color='g',\
         linewidth=5,label="Tree with max depth 2")
plt.plot(ages,max5_tree_model.predict(ages), color='k',\
         linewidth=5, label="Tree with max depth 5")
plt.plot(ages,max2_forest_model.predict(ages), color='c',\
         linewidth=5, label="Forest with max depth 2")
plt.plot(ages,max5_forest_model.predict(ages), color='m',\
         linewidth=5, label="Forest with max depth 5")
plt.legend()
plt.xlabel("age")
plt.ylabel("spend")
plt.show()
```

You should get the following output:

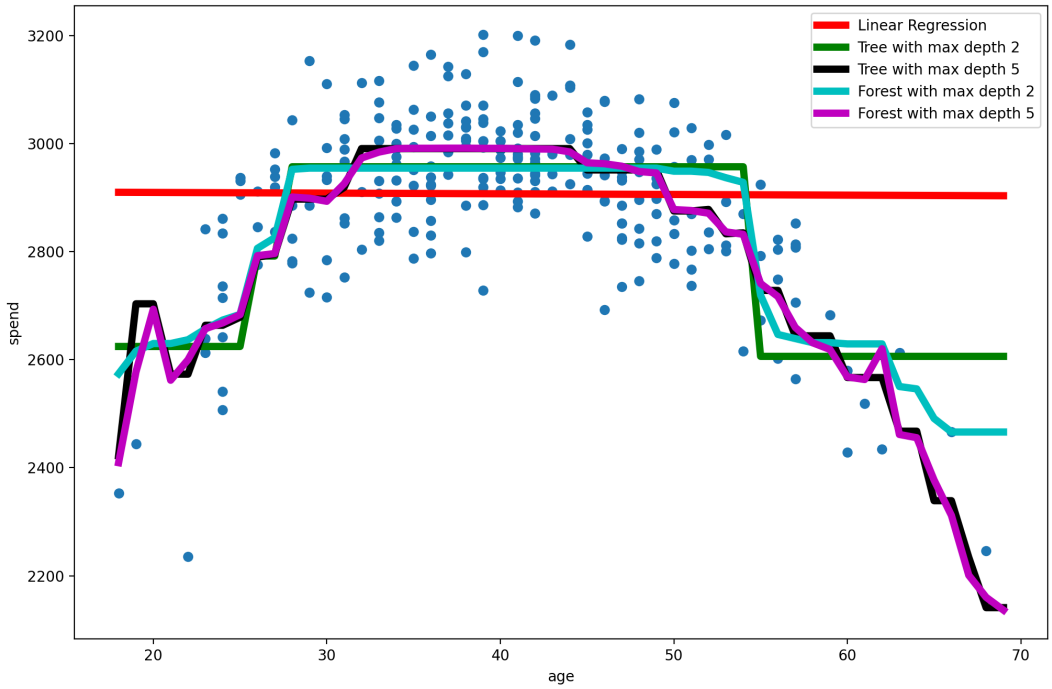

Figure 6.19: Comparison of using different max_depth values while training the random forest model

CHAPTER 7: SUPERVISED LEARNING: PREDICTING CUSTOMER CHURN

EXERCISE 7.05: OBTAINING THE STATISTICAL OVERVIEW AND CORRELATION PLOT

7. Now use the **`seaborn`** library to plot the correlation plot using the following code:

```
corr = data.corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr, \
            xticklabels=corr.columns.values,\
            yticklabels=corr.columns.values,\
            annot=True)
corr
```

The correlation plot should look like the following:

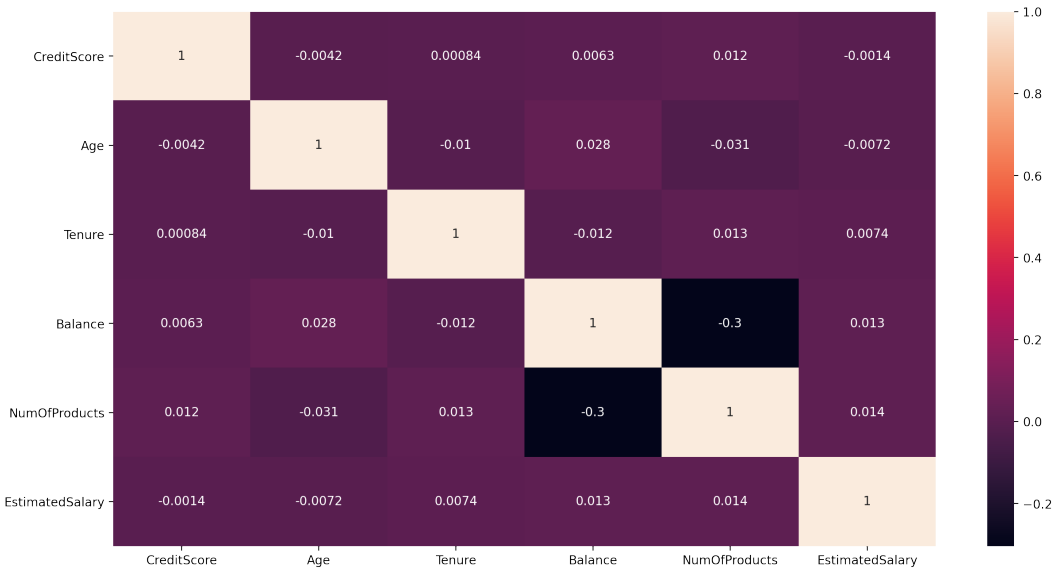

Figure 7.33: Correlation plot of different features

In the correlation plot, it appears that there is a negative (-0.3%) relationship between the number of products purchased and the balance.

EXERCISE 7.06: PERFORMING EXPLORATORY DATA ANALYSIS (EDA)

1. Start with univariate analysis. Plot the distribution graph of the customers for the **`EstimatedSalary`**, **`Age`**, and **`Balance`** variables using the following code:

```
f, axes = plt.subplots(ncols=3, figsize=(15, 6))

sns.distplot(data.EstimatedSalary, kde=True, \
             ax=axes[0]).set_title('EstimatedSalary')
axes[0].set_ylabel('No of Customers')

sns.distplot(data.Age, kde=True, \
             ax=axes[1]).set_title('Age')
axes[1].set_ylabel('No of Customers')

sns.distplot(data.Balance, kde=True, \
             ax=axes[2]).set_title('Balance')
axes[2].set_ylabel('No of Customers')
```

Your output should look as follows:

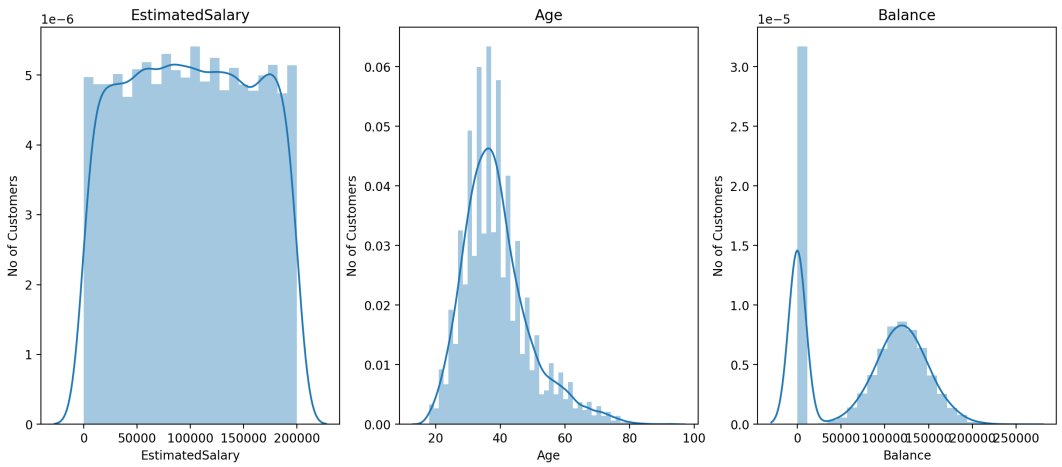

Figure 7.35: Univariate analysis

The following are the observations from the univariate analysis:

`EstimatedSalary`: The distribution of the estimated salary seems to be a plateau distribution – meaning that for a significant range of estimated salary, the number of customers is more or less constant.

Age: This has a normal distribution that is right-skewed. Most customers lie in the range of 30-45 years of age.

Balance: This has a bimodal distribution – this means that there exist two values of balance for which the number of customers is unusually high. A considerable number of customers with a low balance are there, which seems to be an outlier.

2. Now, move on to bivariate analysis. Inspect whether there is a difference in churn for **Gender** using bivariate analysis. Use the following code:

```
plt.figure(figsize=(15,4))
p=sns.countplot(y="Gender", hue='Churn', data=data)
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Churn Distribution by Gender')
```

Your output should look as follows:

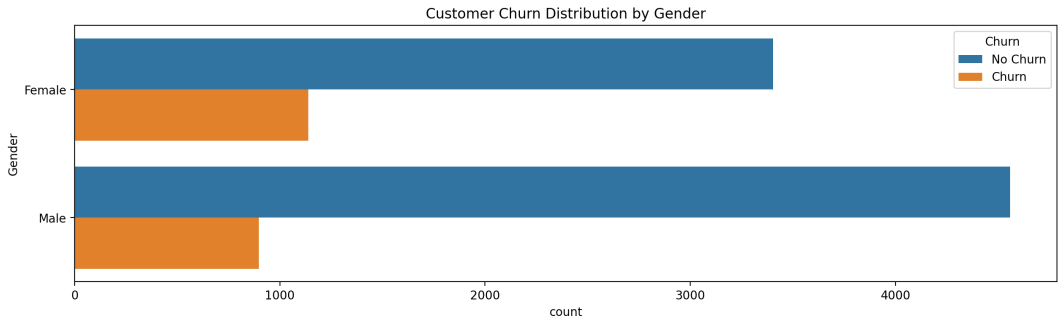

Figure 7.36: Number of customers churned, plotted by gender

You will observe that comparatively, more female customers have churned.

3. Plot **Geography** versus **Churn**:

```
plt.figure(figsize=(15,4))
p=sns.countplot(x='Geography', hue='Churn', data=data)
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Geography Distribution')
```

You should get the following output:

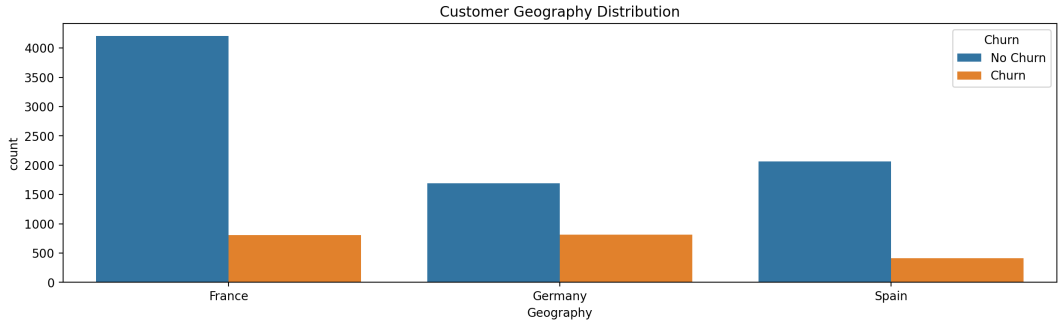

Figure 7.37: Number of customers churned, plotted by geography

Note that the difference between the number of customers that churned and those that did not churn is less for Germany and Spain in comparison with France. France has the highest number of customers compared to the other countries.

4. Plot **`NumOfProducts`** versus **`Churn`**:

```
plt.figure(figsize=(15,4))
p=sns.countplot(x='NumOfProducts', hue='Churn', data=data)
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Distribution by Product')
```

You should get the following output. Note that the largest proportion of churned customers were found where the number of products was one:

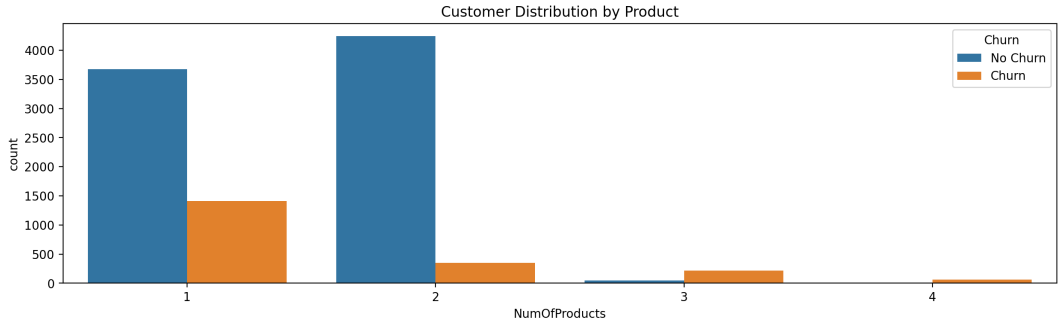

Figure 7.38: Number of customers churned, plotted by product

5. Inspect **Churn** versus **Age**:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Churn'] == 0),'Age'] , \
               color=sns.color_palette("Blues_r")[0],\
               shade=True,label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Churn'] == 1),'Age'] , \
               color=sns.color_palette("Blues_r")[1],\
               shade=True, label='churn')
ax.set(xlabel='Customer Age', ylabel='Frequency')
plt.title('Customer Age - churn vs no churn')
plt.legend()
```

You should get the following output. The peak on the left represents the distribution plot for the **no churn** category, whereas the peak on the right (around the age of 45) represents the **churn** category distribution.

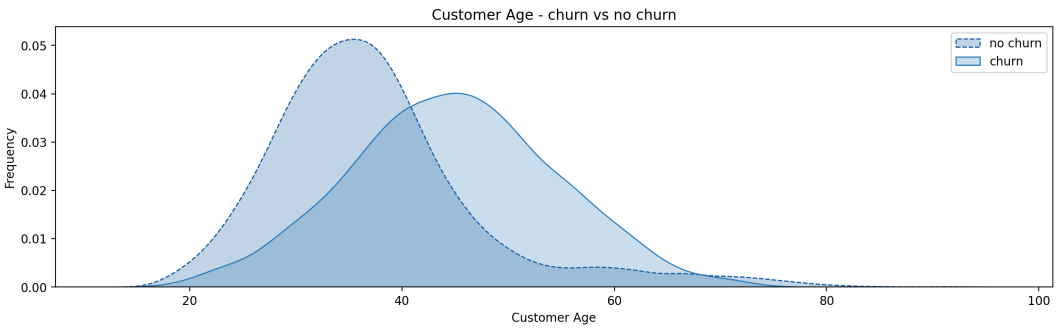

Figure 7.39: Distribution of customer age (churn versus no churn)

Customers in the 35 to 45 age group seem to churn more. As the age of customers increases, they usually churn more.

6. Plot **Balance** versus **Churn**:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Churn'] == 0),'Balance'] , \
               color=sns.color_palette("Blues_r")[0],\
               shade=True,label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Churn'] == 1),'Balance'] , \
               color=sns.color_palette("Blues_r")[1],\
               shade=True, label='churn')
```

```
ax.set(xlabel='Customer Balance', ylabel='Frequency')
plt.title('Customer Balance - churn vs no churn')
plt.legend()
```

You should get the following output:

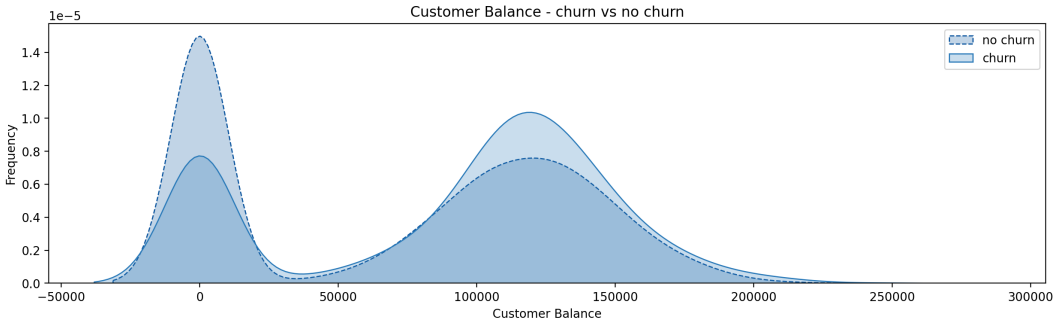

Figure 7.40: Distribution of customer balance (churn versus no churn)

Customers with a negative to low balance churn less than customers with a balance between 75,000 and 150,000.

7. Plot **`CreditScore`** versus **`Churn`**:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Churn'] == 0),'CreditScore'] , \
               color=sns.color_palette("Blues_r")[0],\
               shade=True,label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Churn'] == 1),'CreditScore'] , \
               color=sns.color_palette("Blues_r")[1],\
               shade=True, label='churn')
ax.set(xlabel='CreditScore', ylabel='Frequency')
plt.title('Customer CreditScore - churn vs no churn')
plt.legend()
```

You should get the following output. Notice that the largest proportion of customers who churned have a credit score around 600, whereas those who didn't have a credit score around 650:

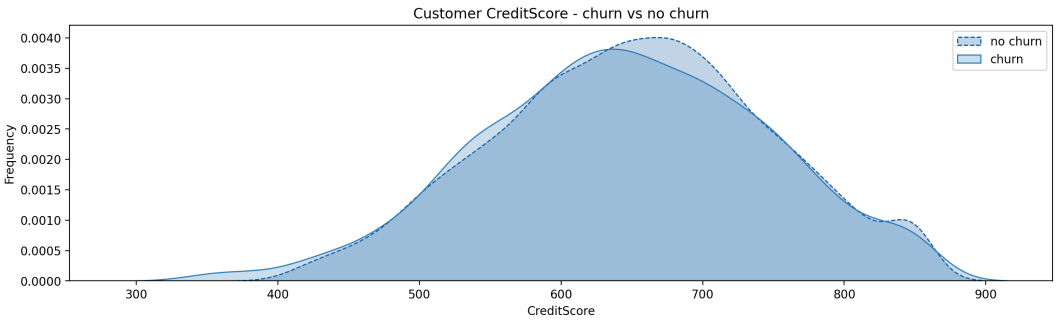

Figure 7.41: Distribution of customer credit score (churn versus no churn)

8. So far you have been analyzing two variables at a time. Compare three variables by plotting **`Balance`** versus **`NumOfProducts`** by **`Churn`**:

```
plt.figure(figsize=(16,4))
p=sns.barplot(x='NumOfProducts',y='Balance',hue='Churn',\
              data=data)
p.legend(loc='upper right')
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Number of Product VS Balance')
```

You should get the following output:

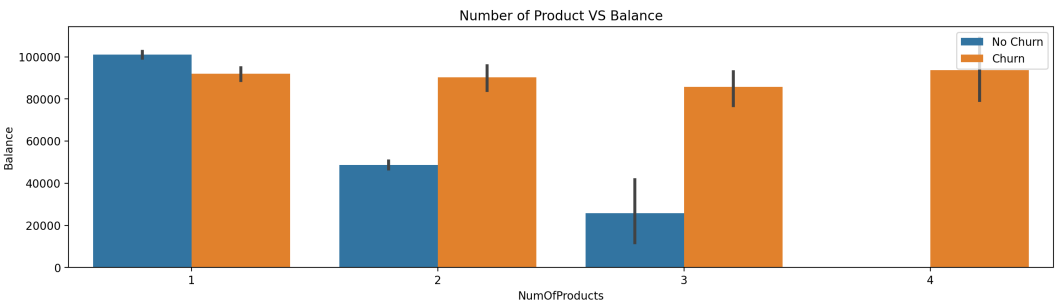

Figure 7.42: Number of products versus balance by churn

In the preceding figure, it appears that as the number of products increases, the balance for churned customers remains very high.

EXERCISE 7.07: PERFORMING FEATURE SELECTION

7. Plot the important features obtained from the random forest using Matplotlib's **`plt`** attribute:

```
plt.figure(figsize=(15,4))
plt.title("Feature importances using Random Forest")
plt.bar(range(X_train.shape[1]), importances[indices],\
        align="center")
plt.xticks(range(X_train.shape[1]), features[indices], \
           rotation='vertical',fontsize=15)
plt.xlim([-1, X_train.shape[1]])

plt.show()
```

You should get the following plot:

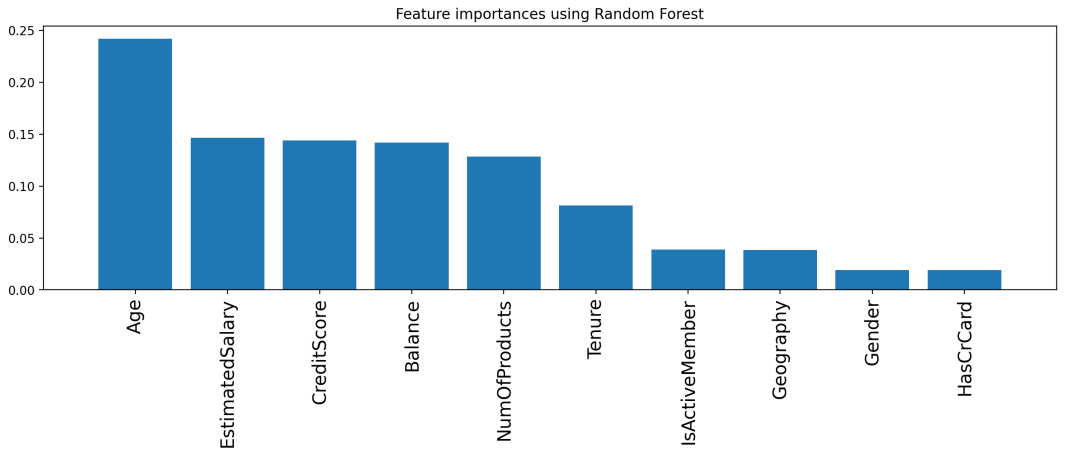

Figure 7.56: Feature importance using random forest

From the preceding figure, you can see that the five most important features selected from tree-based feature selection are **`Age`**, **`EstimatedSalary`**, **`CreditScore`**, **`Balance`**, and **`NumOfProducts`**.

ACTIVITY 7.01: PERFORMING THE OSE TECHNIQUE FROM OSEMN

20. Find the correlation among different variables:

```
corr = data.corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr, \
            xticklabels=corr.columns.values, \
            yticklabels=corr.columns.values,annot=True)

corr
```

This gives the following result:

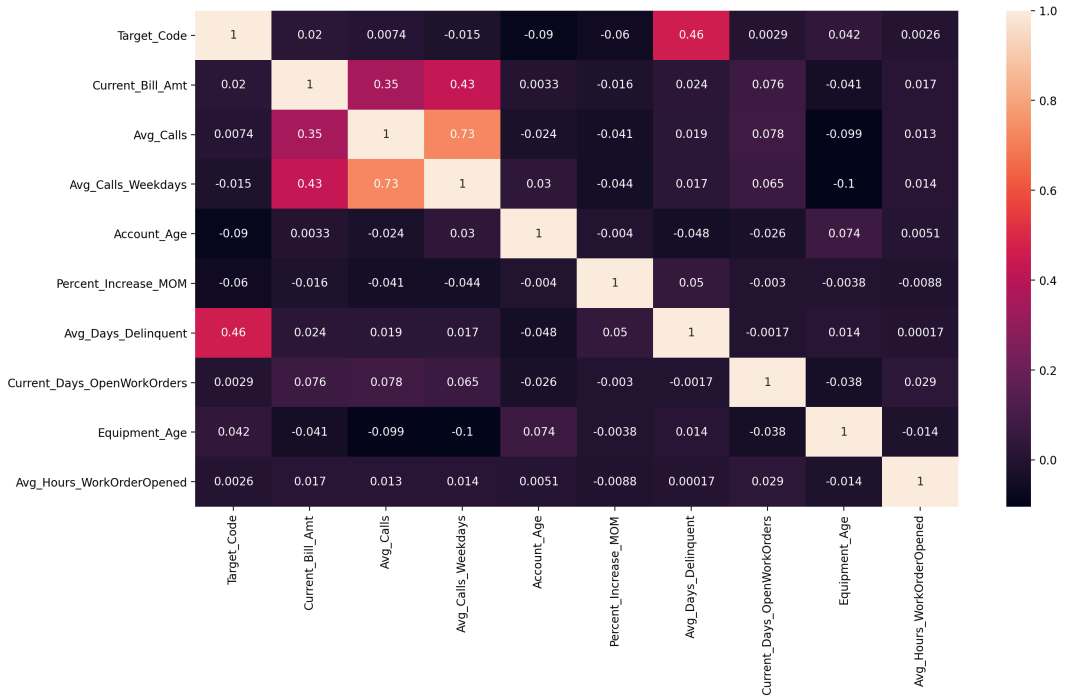

Figure 7.73: Correlation among different variables

From the plot, you will observe that **`Avg_Calls_Weekdays`** and **`Avg_Calls`** are highly correlated, as they both represent the same thing – average calls. **`Current_Bill_Amt`** seems to be correlated with both variables, which is as expected, since the more calls you make, the higher your bill will be.

21. Next, perform univariate and bivariate analyses. Here's the univariate analysis:

```
f, axes = plt.subplots(ncols=3, figsize=(15, 6))
sns.distplot(data.Avg_Calls_Weekdays, kde=True, \
             ax=axes[0]).set_title('Avg_Calls_Weekdays')
axes[0].set_ylabel('No of Customers')
sns.distplot(data.Avg_Calls, kde=True,\
             ax=axes[1]).set_title('Avg_Calls')
axes[1].set_ylabel('No of Customers')
sns.distplot(data.Current_Bill_Amt, kde=True,\
             ax=axes[2]).set_title('Current_Bill_Amt')
axes[2].set_ylabel('No of Customers')
```

You should get the following output. Please ignore any warnings:

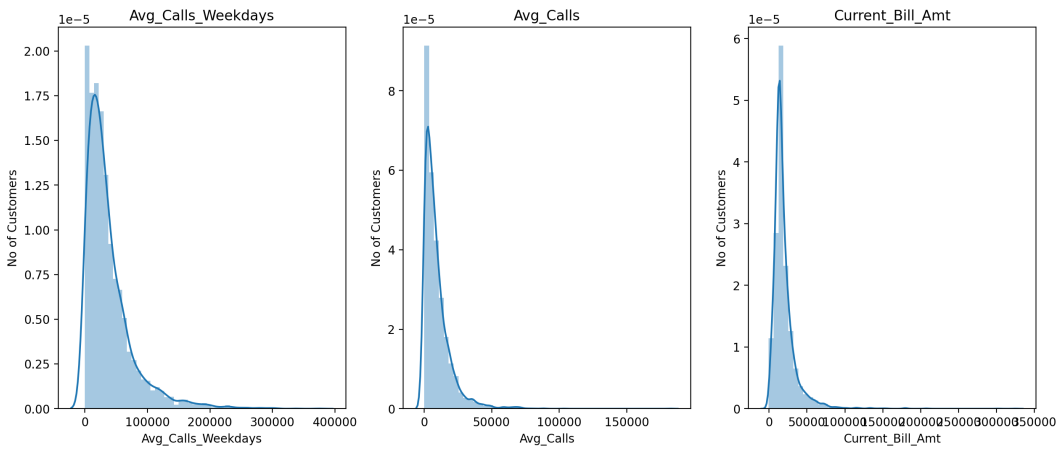

Figure 7.74: Univariate analysis

22. Now, perform the bivariate analysis.

The code for the plot of **`Complaint_Code`** versus **`Target_Churn`** is as follows:

```
plt.figure(figsize=(17,10))
p=sns.countplot(y="Complaint_Code", hue='Target_Churn', \
                data=data,palette="Set2")
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Complaint Code Distribution')
```

You should get an output like the following:

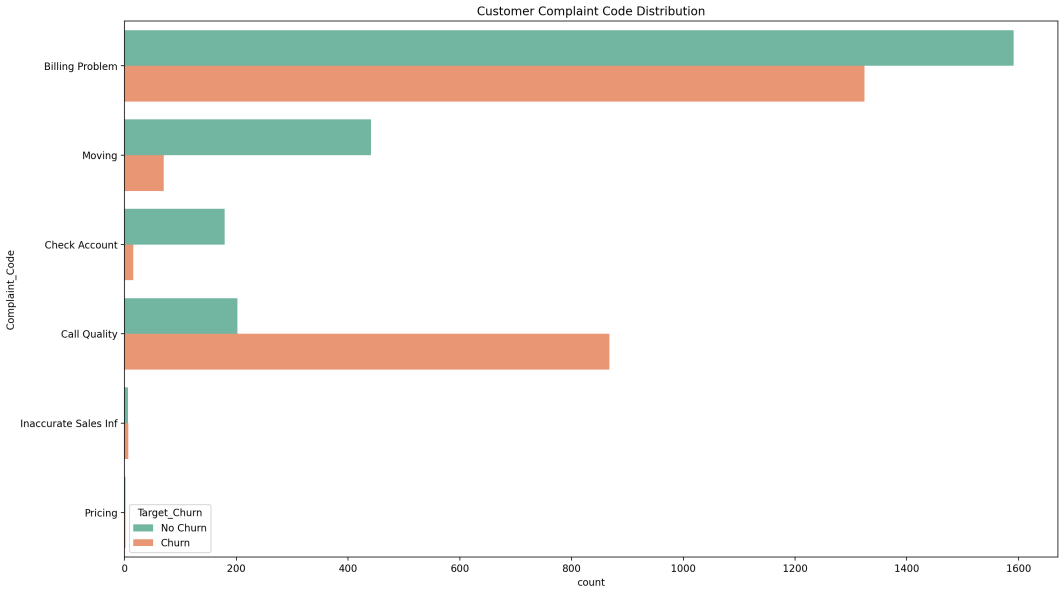

Figure 7.75: Bivariate analysis

From this plot, you'll observe that call quality and billing problems are the two main reasons for customer churn.

The code for the plot of **`Acct_Plan_Subtype`** versus **`Target_Churn`** is as follows:

```
plt.figure(figsize=(15,4))
p=sns.countplot(y="Acct_Plan_Subtype", hue='Target_Churn', \
                data=data,palette="Set2")
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Acct_Plan_Subtype Distribution')
```

You should get the following result:

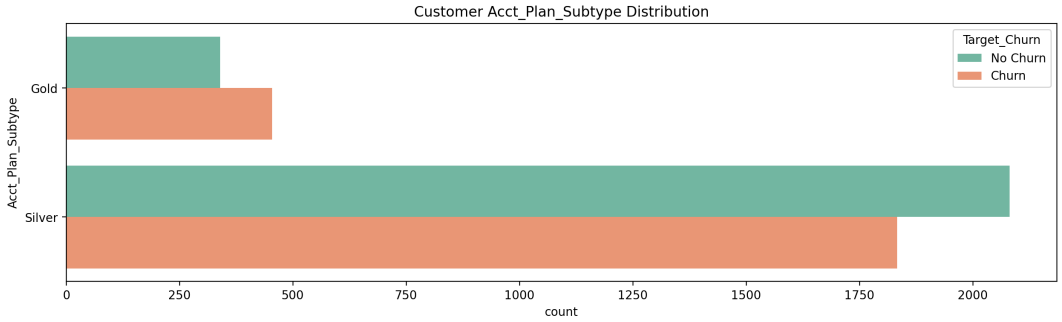

Figure 7.76: Plot of Acct_Plan_Subtype versus Target_Churn

From the preceding plot, we can infer that a larger percentage of customers from the Gold plan have churned. This can be an indicator that the Gold plan needs to be reviewed to understand why the customers are churning at a higher rate with that plan.

The code for the plot of **`Current_TechSupComplaints`** versus **`Target_Churn`** is as follows:

```
plt.figure(figsize=(15,4))
p=sns.countplot(y="Current_TechSupComplaints", hue='Target_Churn', \
                data=data,palette="Set2")
legend = p.get_legend()
legend_txt = legend.texts
legend_txt[0].set_text("No Churn")
legend_txt[1].set_text("Churn")
p.set_title('Customer Current_TechSupComplaints Distribution')
```

You should get the following result:

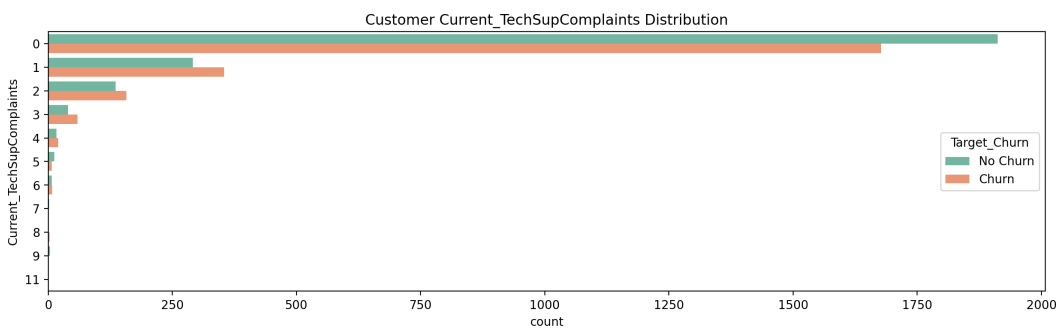

Figure 7.77: Plot of Current_TechSupComplaints versus Target_Churn

From the preceding plot, as expected, we can see that the **No Churn** count is highest for zero tech complaints. Moreover, for all non-zero tech complaints, the number of churned customers is higher than non-churned customers.

The code for the plot of **Avg_Days_Delinquent** versus **Target_Code** is as follows:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 0), \
               'Avg_Days_Delinquent'] , \
               color=sns.color_palette("Set2")[0],\
               shade=True,label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 1),\
               'Avg_Days_Delinquent'] , \
               color=sns.color_palette("Set2")[1],\
               shade=True, label='churn')
ax.set(xlabel='Average No of Days Deliquent/Defaulted \
       from paying', ylabel='Frequency')
plt.title('Average No of Days Deliquent/Defaulted from \
          paying - churn vs no churn')
plt.legend()
```

You should get the following output:

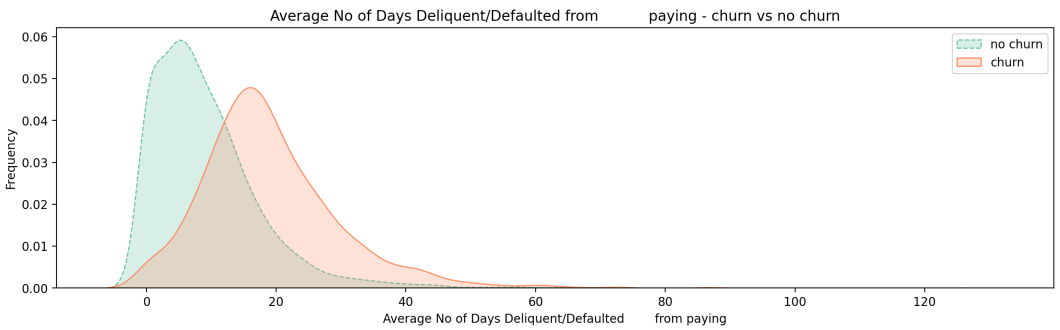

Figure 7.78: Plot of Avg_Days_Delinquent versus Target_Code

From this plot, you'll observe that if the average number of days delinquent is more than 16 days, customers start to churn.

The code for the plot of **`Account_Age`** versus **`Target_Code`** is as follows:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 0), \
                        'Account_Age'], \
               color=sns.color_palette("Set2")[0], \
               shade=True,label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 1), \
                        'Account_Age'], \
               color=sns.color_palette("Set2")[1] ,\
               shade=True, label='churn')
ax.set(xlabel='Account_Age', ylabel='Frequency')
plt.title('Account_Age - churn vs no churn')
plt.legend()
```

You should get the following output:

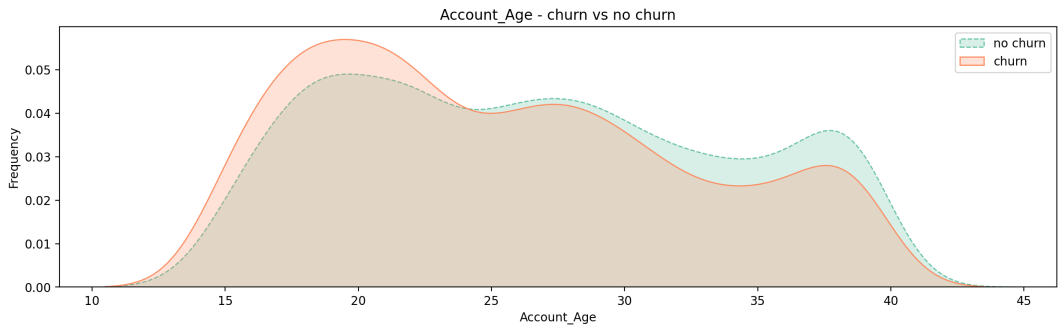

Figure 7.79: Plot of Account_Age versus Target_Code

From this plot, you'll observe that during the initial 15-20 days of opening an account, the amount of customer churn increases; however, after 20 days, the churn rate declines.

The code for the plot of **Percent_Increase_MOM** versus **Target_Code** is as follows:

```
plt.figure(figsize=(15,4))
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 0), \
                        'Percent_Increase_MOM'], \
               color=sns.color_palette("Set2")[0], \
               shade=True, label='no churn',\
               linestyle='--')
ax=sns.kdeplot(data.loc[(data['Target_Code'] == 1), \
                        'Percent_Increase_MOM'], \
               color=sns.color_palette("Set2")[1], \
               shade=True, label='churn')
ax.set(xlabel='Percent_Increase_MOM', ylabel='Frequency')
plt.title('Percent_Increase_MOM- churn vs no churn')
plt.legend()
```

You should get the following output:

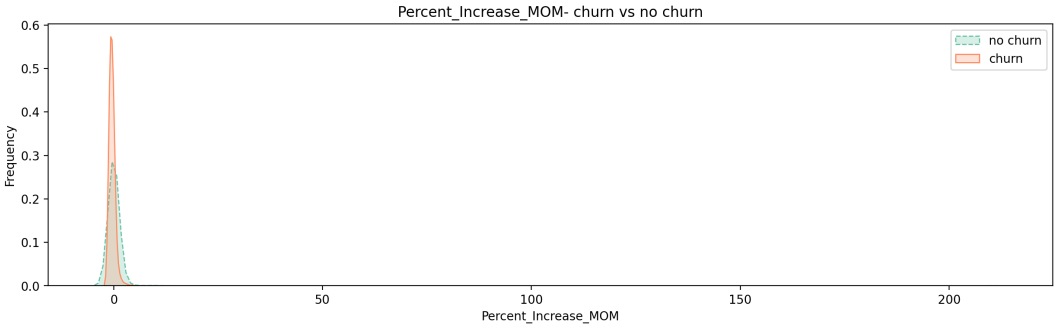

Figure 7.80: Plot of Percent_Increase_MOM versus Target_Code

From this plot, you can see that customers who have a **Percent_Increase_MOM** value within a range of 0% to 10% are more likely to churn.

ACTIVITY 7.02: PERFORMING THE MN TECHNIQUE FROM OSEMN

4. Perform feature selection using the random forest classifier. To carry this out, first fit a random forest classifier on the dataset, and then obtain the feature importance. Plot the feature importance with the help of a bar graph:

```
forest=RandomForestClassifier(n_estimators=500,random_state=1)
forest.fit(X_train,y_train)

importances=forest.feature_importances_
features = data.drop(['Target_Code','Target_Churn'],axis=1)\
               .columns
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(15,4))
plt.title("Feature importances using Random Forest")
plt.bar(range(X_train.shape[1]), importances[indices],\
        align="center")
plt.xticks(range(X_train.shape[1]), features[indices], \
           rotation='vertical', fontsize=15)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

The preceding code will give the following bar graph:

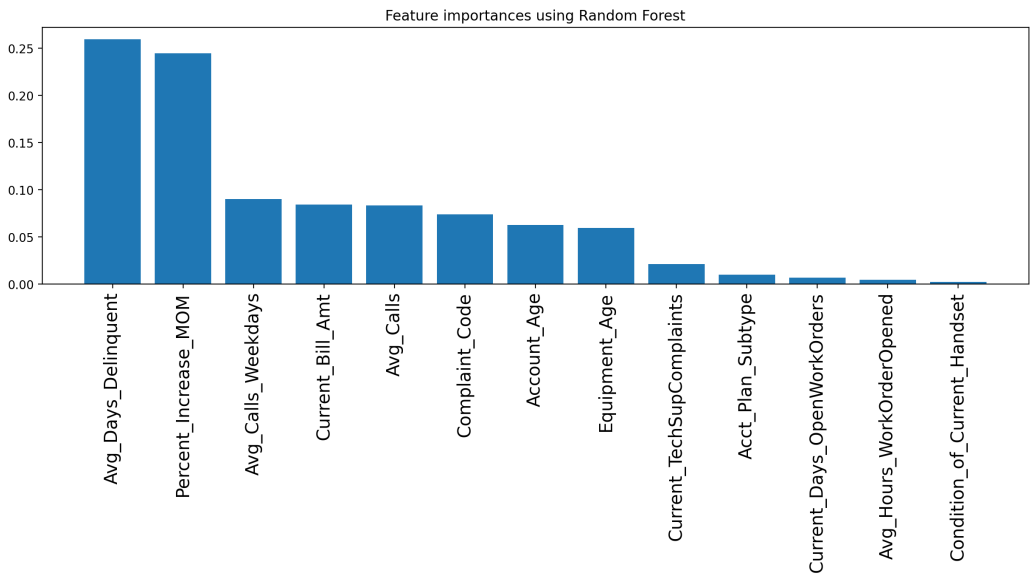

Figure 7.82: Feature importance using random forest

CHAPTER 8: FINE-TUNING CLASSIFICATION ALGORITHMS

EXERCISE 8.09: EVALUATING THE PERFORMANCE METRICS FOR A MODEL

6. Plot the confusion matrix using the following code:

```
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g)
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

You should get the following output:

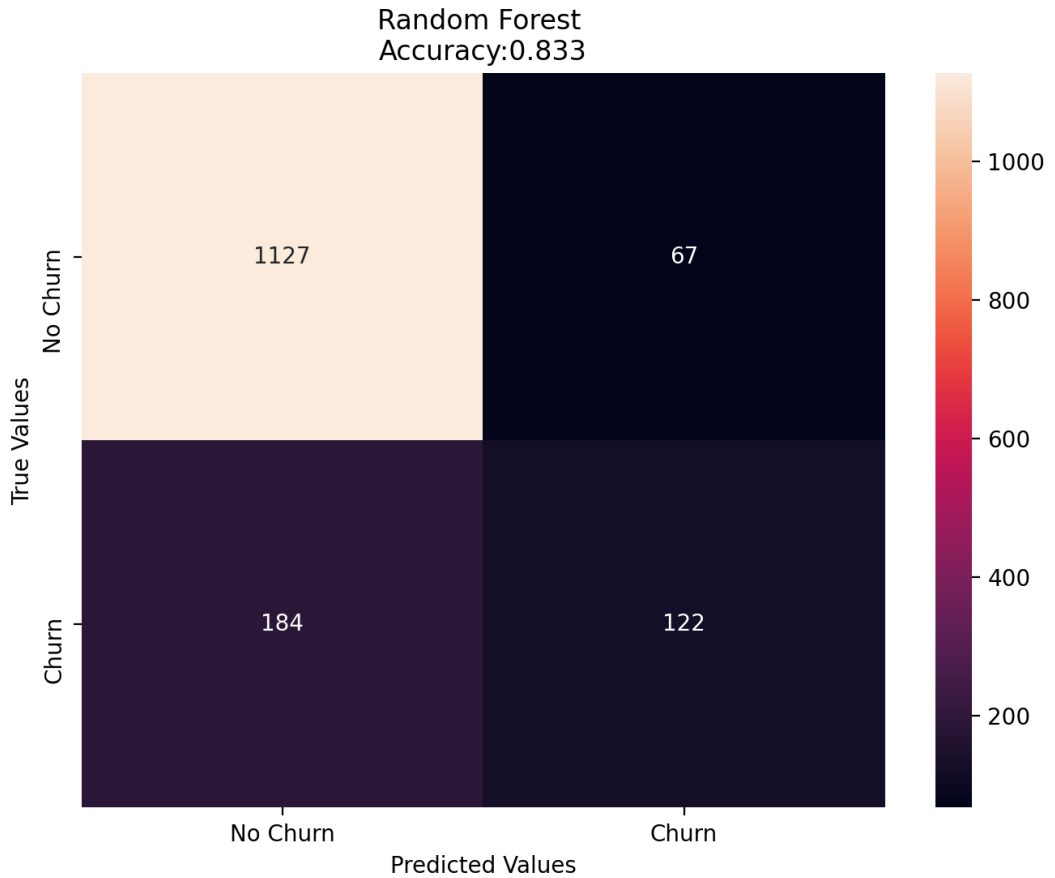

Figure 8.51: Confusion matrix

EXERCISE 8.10: PLOTTING THE ROC CURVE

3. Plot the ROC curve using the following code:

```
plt.figure()
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='%s AUC = %0.2f' % \
         ('Random Forest', roc_auc, color = 'red'))
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.ylabel('Sensitivity(True Positive Rate)')
plt.xlabel('1-Specificity(False Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

Your plot should appear as follows:

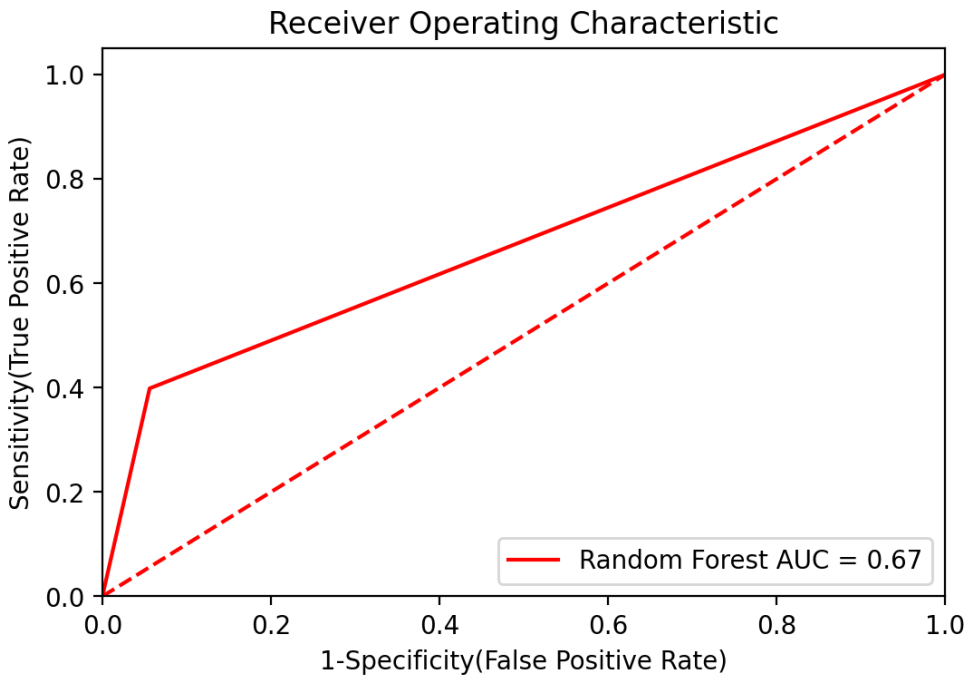

Figure 8.55: ROC curve

ACTIVITY 8.03: COMPARISON OF THE MODELS

5. Plot the confusion matrix:

```
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm,\
                     index = ['No Churn','Churn'], \
                     columns = ['No Churn','Churn'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

The preceding code will plot the following confusion matrix. You can also try manually calculating the precision and recall values using the following diagram and compare it with the values obtained in the previous step:

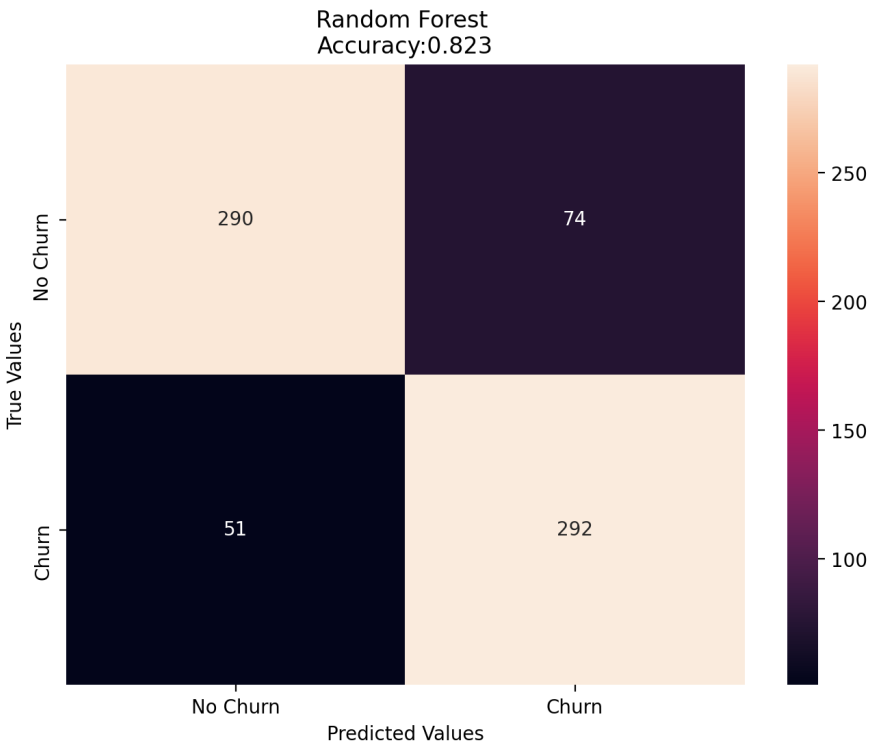

Figure 8.62: Confusion matrix

8. Plot the ROC curve:

```
for m in models:
    model = m['model']
    model.fit(X_train_scalar_combined, y_train)
    y_pred=model.predict(X_test_scalar_combined)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred, pos_label=1)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, label='%s AUC = %0.2f' \
             % (m['label'], roc_auc))
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.ylabel('Sensitivity(True Positive Rate)')
plt.xlabel('1-Specificity(False Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

The preceding code will generate the following ROC curve. Recall that the more a curve lies toward the top-left corner, the higher the performance of the model. Based on this, we can conclude that the random forest (grid search) is the best model out of the four models we have discussed so far in this particular case:

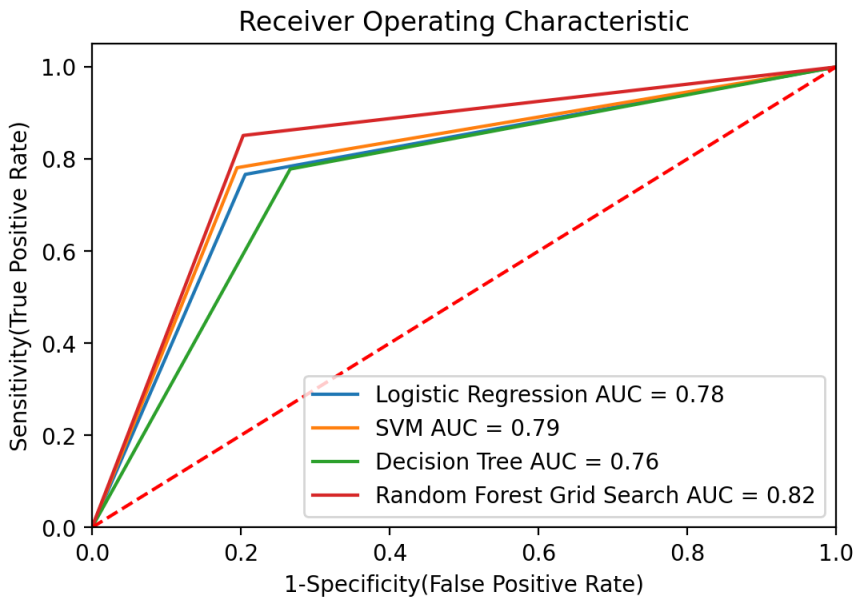

Figure 8.63: ROC curve

Comparing the AUC results of the different algorithms (logistic regression: **0.78**; SVM: **0.79**, decision tree: **0.77**, and random forest: **0.82**), we can conclude that random forest is the best performing model with an AUC score of **0.82** and can be chosen for the marketing team to predict customer churn.

CHAPTER 9: MULTICLASS CLASSIFICATION ALGORITHM

EXERCISE 9.03: PERFORMING CLASSIFICATION ON IMBALANCED DATA

14. Finally, plot the confusion matrix:

```
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,\
                     index = ['<=50K', '>50K'], \
                     columns = ['<=50K', '>50K'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g',cmap='Greys_r')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

Your output should appear as follows:

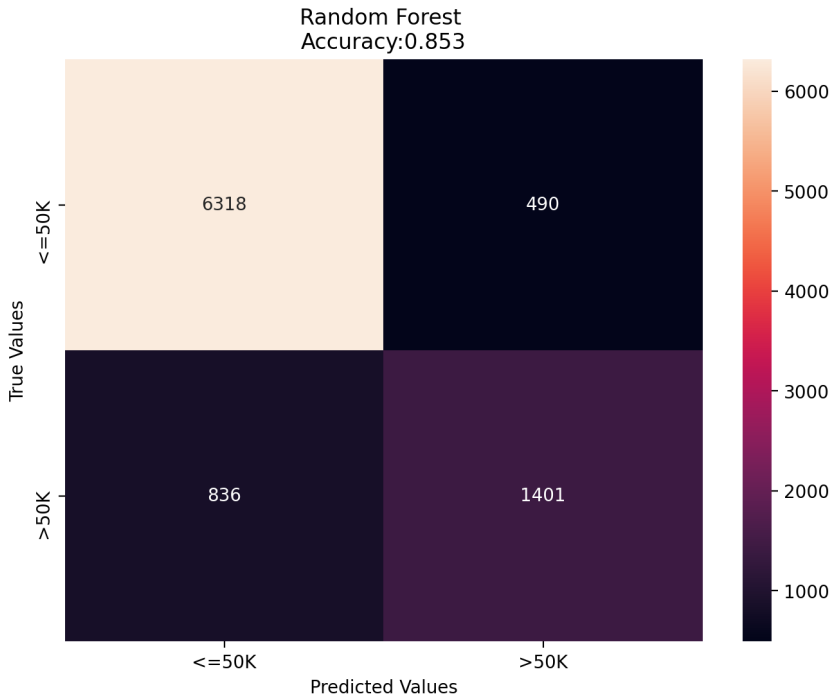

Figure 9.19: Confusion matrix for the random forest classifier

From the preceding confusion matrix, you can say that the model classified **836** people as earning less than or equal to 50,000 USD; however, they were actually earning more than 50,000 USD. Similarly, the model classified **490** people as earning more than 50,000 USD when they were actually earning less than or equal to 50,000 USD.

EXERCISE 9.04: FIXING THE IMBALANCE OF A DATASET USING SMOTE

6. Plot the confusion matrix using the following code:

```
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,\
                     index = ['<=50K', '>50K'],\
                     columns = ['<=50K', '>50K'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

It will appear as follows:

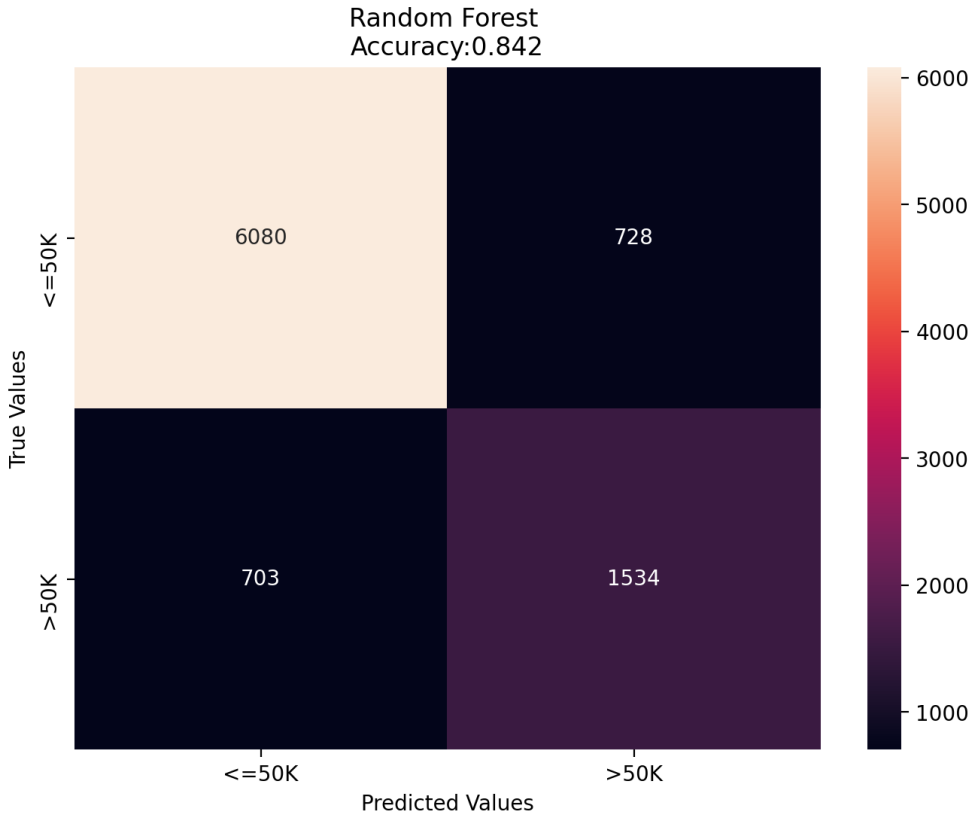

Figure 9.22: Confusion matrix

In *Figure 9.19*, you saw that without using class imbalance, your classifier was able to identify only **1401** people who were earning more than 50,000 USD, whereas by using sampling techniques (**SMOTE**), the classifier identified **1528** people who were earning more than 50,000 USD.

ACTIVITY 9.01: PERFORMING MULTICLASS CLASSIFICATION AND EVALUATING PERFORMANCE

16. Once you have the confusion matrix, plot it using the following code:

```
cm_df = pd.DataFrame(cm,\
                     index = target_names, \
                     columns = target_names)
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

It should appear as follows. You can see that many predictions match the true class, which shows the high accuracy of the classifier:

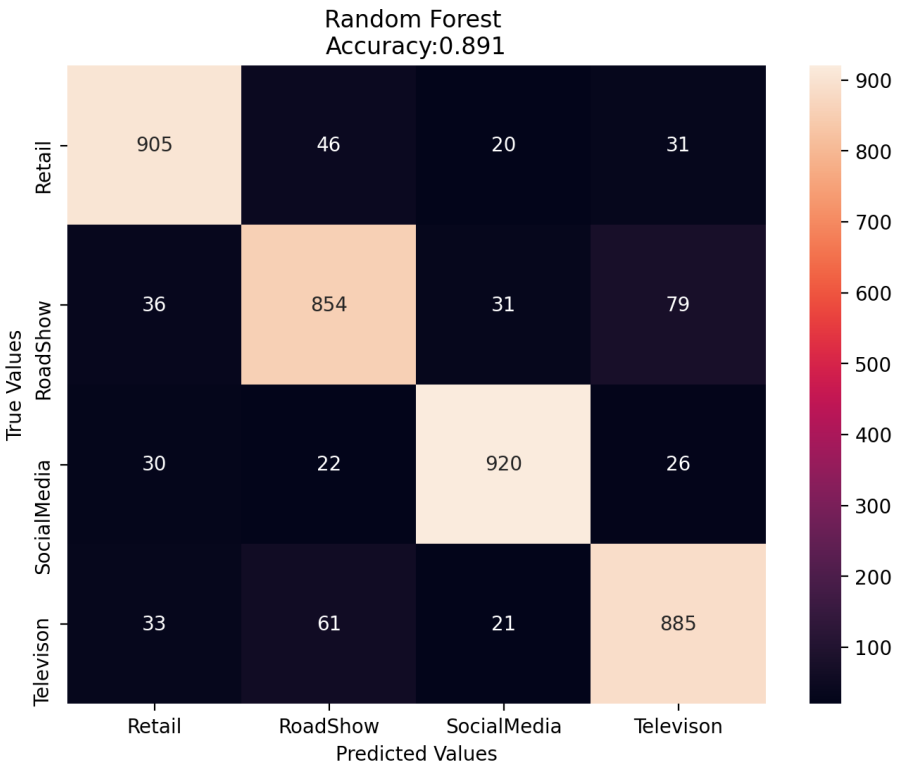

Figure 9.38: Confusion matrix for the random forest classifier

ACTIVITY 9.02: DEALING WITH IMBALANCED DATA USING SCIKIT-LEARN

19. You can also see the impact of class imbalance using the confusion matrix. Enter the following code:

```
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,\
                     index = ['No', 'Yes'], \
                     columns = ['No', 'Yes'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g',cmap='Greys_r')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

You will get the following confusion matrix, which shows that only 24 **`Yes`** values were correctly classified:

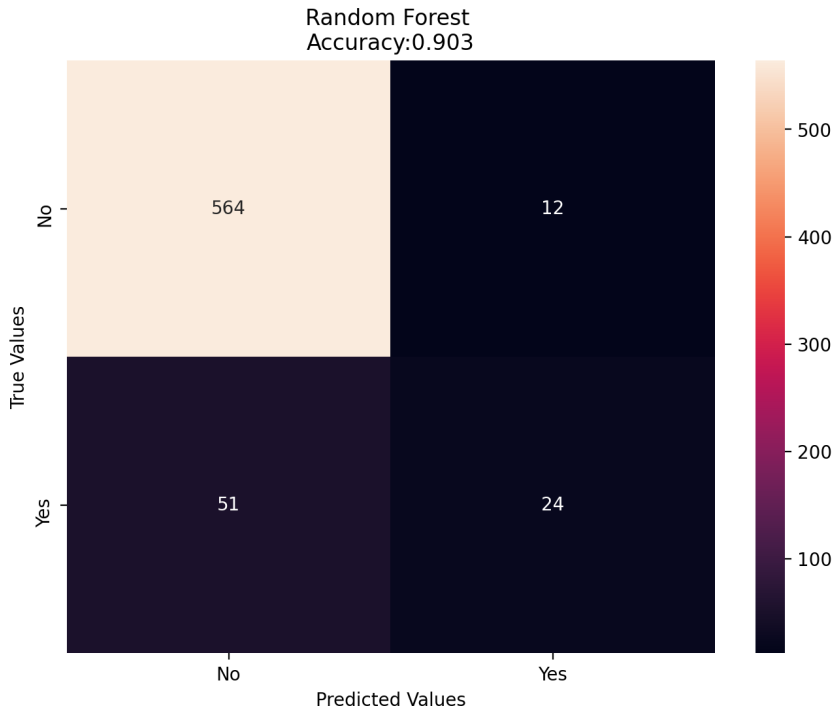

Figure 9.47: Confusion matrix

24. Plot the confusion matrix using the following code:

```
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,\
                     index = ['No', 'Yes'], \
                     columns = ['No', 'Yes'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt='g')
plt.title('Random Forest \nAccuracy:{0:.3f}'\
          .format(accuracy_score(y_test, y_pred)))
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.show()
```

The output should appear as follows:

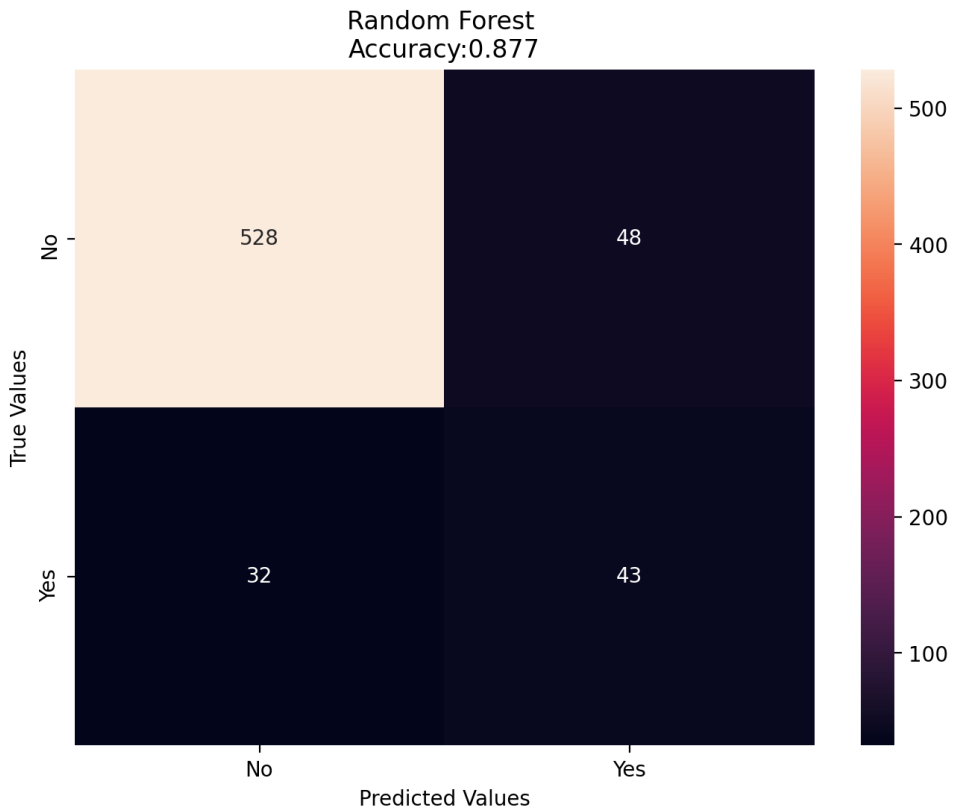

Figure 9.49: Confusion matrix of the random forest classifier

Notice that the number of correctly classified points in the **`Yes`** class has increased to **`46`**, which further shows the importance of balancing an imbalanced dataset. This also matches with what you had set out to do at the beginning of the activity – you first increased the number of sample points in the minority class using the SMOTE technique, then you used the revised dataset to train a machine learning model. This model showed comparatively lower performance; however, the performance was similar for both the classes, which was not the case last time. This was also seen by the improvement in the number of correctly classified points in the **`Yes`** class.