School of IT & Business Technologies
Graduate Diploma in Data Analytics (Level 7)
Cover Sheet and Student Declaration

This sheet must be signed by the student and attached to the submitted assessment.

Course Title:	**Machine Learning and AI**	**Course code:**	GDDA708
Student Name:	Mira Torririt	**Student ID:**	764707793
Assessment No & Type:	Summative Assessment 1[Project]	**Cohort:**	GDDA7123C
Due Date:	09/02/2024	**Date Submitted:**	09/02/2024
Tutor's Name:	Harsh Tiwari		
Assessment Weighting	40%		
Total Marks	100		

Student Declaration:

I declare that:

- I have read the New Zealand School of Education Ltd policies and regulations on assessments and understand what plagiarism is.
- I am aware of the penalties for cheating and plagiarism as laid down by the New Zealand School of Education Ltd.
- This is an original assessment and is entirely my own work.
- Where I have quoted or made use of the ideas of other writers, I have acknowledged the source.
- This assessment has been prepared exclusively for this course and has not been or will not be submitted as assessed work in any other course.
- It has been explained to me that this assessment may be used by NZSE Ltd, for internal and/or external moderation.

Student signature:

Date: 09/02/2024

Tutor only to complete			
Assessment results:	**Part A** (max. 25 marks)	**Part B** (max. 25 marks)	**Part C** (max. 50 marks)
	Total Marks: /100		**Grade:**

Graduate Diploma in Data Analytics (Level 7)	
Course and Code	GDDA708 – Machine Learning and AI
Assignment Title	Assessment 1
Assessment Type	Project 1
Student's Name	Mira A. Torririt
Student's ID	764707793
Tutor's Name	Harsh Tiwari

Part A: Supervised Machine Learning (please see Appendix A) - Classification

Task 1: Data Preparation

a. The Python libraries numpy and pandas were used to load the dataset using the following code snippets :
   - bank_churn_df = pd.read_csv('bank_churn_prediction.csv') – loading
     bank_churn_df – display
   - bank_churn_df.head() – displaying the first rows
   - bank_churn_df.tail() – displaying the last rows

b. The missing values were handled using the following techniques:
   - To identify the missing values, the isnull(). sum () function was used, with the code; bank_churn_df.isnull().sum()
   - To remove the missing values, dropna function was used, with the code;
     bank_churn_df =
     bank_churn_df.dropna(subset=['country','products_number','estimated_salary'], axis=0)
     bank_churn_df.isnull().sum()
     Dropna was used to remove incomplete rows instead of fillna, for more accurate analysis and to avoid the manipulation of data.

c. The effective way to get an overview of the dataset is the df.info function. The code bank_churn_df.info() determined the different data types and displayed the non-null counts and total number of entries.
   The one-hot encoding method was used to convert the categorical data (gender and country) to numerical. The function pd.get_dummies converted the gender to the binary data, dropping the first value, using the code drop_first=True, to avoid redundancy.

d. To analyze the data using the visualization techniques, the libraries seaborn and matplotlib.pyplot were imported. For data distribution, a histogram was used for graphical representation. It also identifies the patterns or trends. The given data showed a bell-shaped curve, which indicates a normal distribution of age, in which the mean (average) value is the center of the distribution. The box plot shows the account balances' minimum, maximum, median, interquartile range, and potential outliers. The outliers in the account balances may create anomalies or errors during analysis. It affects the accuracy of the data, which is vital in the decision-making process.

Task 2: Feature Engineering

a. I used the Recursive Feature Elimination (RFE) to automatically select the important features for the feature selection. It is more time-efficient to use than the manual selection. In this method, I removed the churn as the outcome feature, and the customer id as unnecessary in the analysis using the code X = bank_churn_df.drop(['churn','customer_id'],axis=1) y = bank_churn_df['churn']. Then splitting the data for training and testing purposes using the code X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42). The logistic regression model was used, using the code log_reg_model = LogisticRegression(max_iter=120000), because it is well-suited for binary classification, predicting whether the customer will churn. RFE will train the model on the entire set of features using code rfe = RFE(log_reg_model, n_features_to_select=5). And train the model using code rfe.fit(X_train, y_train), selected = X_train.columns[rfe.support_]. The 5 important features selected are: products_number, credit card, active member, gender (male) and country (Germany).

   The second technique is by using the Tree-based models using the random forest model as classifier with the code random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42). The code random_forest_model.fit(X_train, y_train) was used to train the model. The random forest evaluates the contribution of each feature to the overall predictive performance which influence in predicting the bank churn. In getting the important feature the code feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': random_forest_model.feature_importances_}), revealing the 5 most important features namely, age, estimated salary, balance,credit score and product number.

b. The feature scaling method used is the Min-Max. It is the type of feature scaling that transforms the numerical values into binary code ranging from 0 to 1 for normalization to a uniform scale. It affects the data distribution by compressing the numerical values and narrowing the gap between them. It ensures that all features are uniformly scaled, promoting a consistent and standard dataset representation.

Task 3: Model Building and Prediction

a. First, in the Logistic regression model, I copied the bank churn to avoid changing the original data frame using the code log_reg_df = bank_churn_df.copy(). Scale the important features to make sure that the numerical values are uniform using the code features_to_scale = ['age', 'balance', 'credit_score', 'estimated_salary', 'tenure', 'products_number'] Then extract the subset features by subset_features = log_reg_df[features_to_scale].values to enhance model interpretability. Using the min-max scaler to ensure all features are on a similar scale, it prevents certain features from dominating the learning process. The cod for min-max is scaler = MinMaxScaler() scaled_features = scaler.fit_transform(subset_features). Replace the original values with scaled values in data frame by using the code log_reg_df[features_to_scale] = scaled_features.

   On splitting the result for training and testing, the test size of 20% determines the percentage of test data extracted from the X data frame. The remaining 80% will be used for training. The random state dictates the consistency of the randomness process of the model. It ensures that the result is consistent when comparing different models. I used 42 as the random state in this data using the code X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42).

In creating the model using the logistic regression,set multi class to multinomial since the dataset is not just a binary classification (ex, age, balance, credit score), use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm to solve small to medium size data with the code log_reg_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=120000). Now, we can train the model by using the code log_reg_model.fit(X_train, y_train) and make a prediction using the code prediction = log_reg_model.predict(X_test). To compare the prediction results, count the number of the same result between y test and prediction and divide it by the total count of prediction. Example, y_test = [0, 1, 2, 4, 5], prediction = [0, 3, 2, 3, 5], the result of 3 matches over 5 is 0.6 accuracy where in accuracy = accuracy_score(y_test, prediction), classification_report_output = classification_report(y_test, prediction). Using the logistic regression, the accuracy score is 0.81:

```
Using LogisticRegression
Accuracy: 0.81
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      1579
           1       0.61      0.22      0.33       420

    accuracy                           0.81      1999
   macro avg       0.72      0.59      0.61      1999
weighted avg       0.78      0.81      0.77      1999
```

Second, in the Random Forest Classifier, I created a model with estimators equal to 100 and a random state 42. The n_estimator provides the number of decision trees to be used by the forest. The higher the n_ estimator, the higher or better the result of the model. It also increases the processing of time. I used 100 for the n_estimator as the common number provides a good trade-off between the result and the performance. The random states dictate the consistency of the randomness process of the model. It ensures consistency when comparing different models. I used 42 for the random state as it is the common number. The code used is random_forest_model = andomForestClassifier(n_estimators=100, random_state=42). To train the model, the code used is random_forest_model.fit(X_train, y_train). Based on the 100 trees created (using n_estimators), it will check using the random entries and look for pass/ fail results. The code used is prediction = random_forest_model.predict(X_test). Using the random forest classifier, the result of the accuracy is 0.87.

```
Using RandomForestClassifier
Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1579
           1       0.81      0.49      0.61       420

    accuracy                           0.87      1999
   macro avg       0.84      0.73      0.76      1999
weighted avg       0.86      0.87      0.85      1999
```

Third, using the Decision Tree Classifier, I used a maximum depth of 3. The higher the maximum depth, the more patterns to use in the training of the data. But this may lead to overfitting (pickup noise data or unrelated patterns. The code used is decision_tree_model = DecisionTreeClassifier(max_depth = 3). To train the model, the code used is decision_tree_model.fit(X_train, y_train). Based on the 100 trees created (n_estimators), it will predict a pass/fail result using the code prediction = decision_tree_model.predict(X_test). Using the decision tree classifier, the accuracy result is 0.83.

```
Using DecisionTreeClassifier
Accuracy: 0.83
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.99      0.90      1579
           1       0.88      0.23      0.36       420

    accuracy                           0.83      1999
   macro avg       0.85      0.61      0.63      1999
weighted avg       0.84      0.83      0.79      1999
```

b. In supervised machine learning, the 3 important aspects to consider are:
   1. Data preparation and processing. This includes data cleaning and making sure that there is no missing information. Feature scaling and normalization focus on the standardization of the data, ensuring that they are on a similar scale. Converting the categorical variables to numerical ones to perform arithmetic operations as machine learning models do not understand the text. Handling outliers that may affect the model's performance and feature engineering to prepare input data that will best fit the machine learning algorithm.
   2. Model Selection and Training. The right model will give us the right result. It is important to determine the nature of the problem, whether it is a classification or regression, the size of the dataset, and its attributes. Splitting the data to test and training is essential in machine learning as it assesses the performance of the model.
   3. Model evaluation – choosing the appropriate accuracy, precision, recall, and F1 score, ensuring the effectiveness of the model.

Part B: Supervised Machine Learning (please see Appendix B) – Regression

Task 1 : Data Preparation

a. The Python libraries numpy and pandas were used to load the dataset using the following code snippets :

- advertising_df = pd.read_csv('advertising.csv') – loading
- advertising_df – display
- advertising_df.head() – displaying the first rows
- advertising_df.tail() – displaying the last rows

b. The missing values were handled using the following techniques:

- To identify the missing values, the isnull(). sum () function was used, with the code; advertising_df.isnull().sum()
- To remove the missing values, dropna function was used, with the code; advertising_df = advertising_df.dropna(subset=['Radio','Newspaper'], axis=0) advertising_df.isnull().sum()

  Dropna was used to remove incomplete rows instead of fillna, for more accurate analysis and to avoid the manipulation of data.

c. The effective way to get an overview of the dataset is the df.info function. The code advertising_df.info() determined the different data types and displayed the non-null counts and total number of entries.

   Since all are numerical data types, no need to use conversion.

d. To analyze the data using the visualization techniques, the libraries seaborn and matplotlib.pyplot were imported. I used the line plot to show the trend of marketing expenses versus sales. The codes are as follows:

```
x_trend = list(range(0, len(advertising_df)))

plt.figure(figsize=(10,6))

plt.plot(x_trend, advertising_df['TV'], label='TV', marker='o')
plt.plot(x_trend, advertising_df['Radio'], label='Radio', marker='s')
plt.plot(x_trend, advertising_df['Newspaper'], label='Newspaper', marker='^')
plt.plot(x_trend, advertising_df['Sales'], label='Sales', linestyle='--', marker='x')

plt.title('Advertising Expenses vs Sales')
plt.xlabel('Data Point')
plt.ylabel('Amount')
plt.grid(True)
plt.legend()
plt.show()
```

The result revealed that the marketing expenses are higher than the sales. The biggest expense goes to the television advertisement.

A pair plot was used to identify the patterns and trends. The result showed that the highest sales is the effect of tv ads.

Task 2: Feature Engineering

a. I used the filter-based feature selection to assess each feature independently, making them more scalable, especially when used in large datasets. This simplifies/narrows down the data and avoids unnecessary information, which may lead to overfitting the model. I used the Pearson correlation coefficient on the variables to calculate the correlation. The feature is considered relevant if the correlation between them is high (closest to 1). If the correlation is low(closest to 0), it is considered irrelevant and excluded from the analysis. In this dataset, the TV ad got the highest correlation with sales. The code used is correlation_matrix = advertising_df.corr().
b. On the feature scaling, I used the Min-Max using the code from sklearn.preprocessing import MinMaxScaler. To extract the subset of features, I used the code subset_features = advertising_df[features_to_scale].values, and to normalize the numerical values I used the codes: scaler = MinMaxScaler() scaled_features = scaler.fit_transform(subset_features), then replaced the original values with scaled values in the data frame using the code minmaxscalar_df = pd.DataFrame(scaled_features, columns=features_to_scale). This scaling technique helps in the normalization of data as it narrows down all features within the common range (0,1)

Task 3: Model building and prediction

a. The three models used are linear regression, random forest regressor, and support vector regressor.
    1. Linear regression – used to predict one variable le (dependent or outcome variable) based on the values of one or more independent variables (features): X = linear_df and y = advertising_df['Sales']. In splitting the variables, the test size is 20% and the train size is 80%. I used the random_state as a parameter to ensure the consistency of the results when comparing different models : X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42).

       Assuming the relationships between the advertisement expenses and sales are linear, I made a sales predictions based on advertisement budgets:
       linear_reg_model = LinearRegression(), then train the model:
       linear_reg_model.fit(X_train, y_train);
       prediction = linear_reg_model.predict(X_test).

       The mean squared error (MSE) measured the amount of error in statistical models. It assess the average square difference between the predicted and actual values. A smaller MSE indicates that the model's predictions are closer to the actual values:
       mse = mean_squared_error(y_test, prediction).

       R-squared measures the proportion between the dependent variable and the independent variables. The result ranges from 0 – 1 (0% to 100%). The closer the result to 0, it means it does not correlate to the dependent variable (sales). The closer the result to 1, the higher the correlation. The higher the correlation, the better the model: r2 = r2_score(y_test, prediction), print(f"Mean Squared Error: {mse:.2f}").

```
Using LinearRegression
Mean Squared Error: 2.23
R-squared: 0.91
```

2. Random Forest Regressor – it incorporates multiple decision trees which helps in reducing overfitting of data in the model. To train the model, I used the code random_forest_model.fit(X_train, y_train). Using the n_estimators, it checked random entries which resulted in pass/fail. The code used is prediction = random_forest_model.predict(X_test).

   Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average square difference between the predicted and actual values. A smaller MSE indicates that the model's predictions are closer to the actual values: mse = mean_squared_error(y_test, prediction), print(f"Mean Squared Error: {mse:.2f}")

   R-squared measures the proportion between the dependent and independent variables. The result ranges from 0-1 (0% to 100%). When the value is 0, the model explains none of the variance is important in the target variable. When the value is closer to 1, the model perfectly predicts the target variable. The higher the result, the better the model: r2 = r2_score(y_test, prediction), print(f"R-squared: {r2:.2f}").

```
Using RandomForestRegressor
Mean Squared Error: 1.55
R-squared: 0.94
```

3. Support vector regression (SVR) relies on a subset of training data called support vectors, meaning that the models are determined by the support vectors, not the entire dataset. Using 20% of the data for testing and 80% for training: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42), I used the model, svr_model = SVR(kernel="linear", C=1.0). To train the model, I used the code, svr_model.fit(X_train, y_train), and for prediction, I used the code prediction = svr_model.predict(X_test). I used the mean squared error and r-squared to evaluate the model: mse = mean_squared_error(y_test, prediction), r2 = r2_score(y_test, prediction), print(f"Mean Squared Error: {mse:.2f}").

   R-squared ranges from 0-1. When the value is 0, the model explains none of the variance is important in the target variable. When the value is 1, the model perfectly predicts the target variable.

```
Mean Squared Error: 2.59
R-squared: 0.89
```

b. In regression analysis, it is important to determine the nature of the variables and their relationships to each other. Identifying the dependent (or the outcome) variable and independent (predictors) variables will influence the effectivity of the model. The correlations determine the relationships of the variables. The feature selection helps determine which relationship is relevant in the analysis.

Part C: Time Series Trend Analysis and Forecasting (please see Appendix C)

Task 1: Data Exploration

The initial observations include exploration of the dataset by getting the information; gold_df.info(). It summarizes the data frame for a quick assessment of its structure. The data has two data types, namely object and float. The total rows are 10, 787 with no null values. To understand the basic statistics of the dataset, I used the code, gold_df.describe(). This function is helpful for a more comprehensive understanding of the data frame. Since the date column is an object data type, I converted it to the datetime for an accurate analysis. I used the calendar heatmap to show the patterns across days or months. It is also useful in detecting seasonality in relation to specific days of the week or months. It starts by importing the libraries; seaborn and matplotlib. I also used the line plot to visualize the trend of gold prices over time. It is an important step in the exploratory data analysis process as it helps the analysts better understand the time-series data.

Task 2: Trend Analysis

A line plot was used to provide a visual summary of the data. It connects the data points that show the rend overtime. In our dataset, the line moves upward means there is a positive trend.

Task 3: Seasonality Assessment

Checking the stationarity of time-series by getting the Auto Correlation Function (ACF) and Partial Auto Correlation Function (PACF). In our dataset, the ACF shows the correlation with lags is high and positive with very slow decay, while in PACF, partial correlations have a single spike at lag 1. I also checked the p-value using the fuller, and the result below confirmed that the time series is likely to be stationary, as it is less than the 0.05 significance level. The Autoregressive Integrated, Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving Average (SARIMA) were used for predictions.

```
from statsmodels. tsa.stattools import adfuller
adf_test=adfuller(gold_df_train['Value'])
print (f'p-value:{adf_test[1]}')
```

```
p-value:0.005754726484247235
```

Task 4: Anomaly Detection

The Z-score is important in anomaly detection as it facilitates normalization, identification, and quantification of outliers in data. It is often detected by setting a threshold typically chosen based on the desired level of sensitivity to outliers.

Task 5: Prediction and Recommendation

Based on the data, the gold price will continue to rise over time. Thus, a need for regular comparative analysis and forecasting is highly recommended.

Part D: Clustering (please see Appendix D)

Task 1: Data Preparation

a. To ensure the accuracy of the data, it underwent data cleaning techniques. I started by looking for null values: mall_customers_df.isnull().sum().

a.1) The simple imputer was used to replace the missing values in numerical columns such as age, annual income, and spending score. It was replaced using the mean value. Dropping the rows is not an option because it will reduce the size of the data for analysis.

```
from sklearn.impute import SimpleImputer

feature_to_select = ['Age','Annual Income (k$)', 'Spending Score (1-100)']

# Create an instance of SimpleImputer
simp_imputer = SimpleImputer(missing_values=np.nan, strategy="mean")

simp_imputer.fit(mall_customers_df[feature_to_select])

X_imputed = simp_imputer.transform(mall_customers_df[feature_to_select])

mall_customers_df[feature_to_select] = X_imputed

mall_customers_df
```

a.2) There was one missing value in gender which I decided to drop. Since gender has only 2 categories (male and female). The fillna function is not an option due to bias that will affect the accuracy of the analysis.

```
mall_customers_df = mall_customers_df.dropna(subset=['Gender'], axis=0)
mall_customers_df
```

I also dropped the customer ID as it was not needed in the analysis.

```
mall_customers_df = mall_customers_df.drop('CustomerID', axis=1)
mall_customers_df
```

a.3) The box plot was used to detect the outliers. On the given dataset, an outlier appeared in annual income in the male category. Using the score function, the outliers were determined and removed for data accuracy.

```
from scipy.stats import zscore
mall_customers_df['Annual_Z_Score'] = zscore(mall_customers_df['Annual Income (k$)'])
threshold = 2
zscore_annual_outliers = ((mall_customers_df['Annual_Z_Score'] < (-1*threshold)) | (mall_customers_df['Annual_Z_Score'] > threshc
print(f'Annual Outliers using Z Score: {zscore_annual_outliers.sum()}')
mall_customers_df = mall_customers_df[~zscore_annual_outliers]
mall_customers_df
```

```
Annual Outliers using Z Score: 8
```

b. Data Exploration

b. 1) Getting the correlations of age and annual to spending score. It helps in determining the relationships of the variables.

```
correlation_matrix = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].corr()
print(correlation_matrix)
```

c. 2) For data visualization, I used the histogram. It displays the counts of the data points within specific bins. It also includes the means and median values. The peaks of the histogram indicate the concentration of the most values.

```
sns.histplot(data=mall_customers_df, x='Age', kde=True, color='blue', ax=axs[0])
sns.histplot(data=mall_customers_df, x='Annual Income (k$)', kde=True, color='green', ax=axs[1])
sns.histplot(data=mall_customers_df, x='Spending Score (1-100)', kde=True, color='orange', ax=axs[2])
```

b.3) To explore the characteristics of the data, I used the function df.describe . The data shows that the average age of customers is 39 years old. Younger people at the minimum age of 18 go to the mall. With an average income of 60k$, depending on the cost of living, most probably are working with a higher salary and an average of 50 spending score, which means they have a moderate spending behavior. Most customers may be engaged but not consistent.

```
spending_stat = mall_customers_df.describe()
print('Summary Statistics')
spending_stat
```

Task 2: Unsupervised Algorithm Implementation

a.1) K-means - was used for segmentation to make the data more manageable for the analysis. It is a powerful tool to understand customer behavior. The value of K (based on the elbow method) is 5, which means 5 clusters. Based on the customer's income and spending habits, it was categorized into:

1) low-income earners with low spending score
2) high-income earners with low spending score
3) medium-income earners with medium spending score
4) low-income earners with high spending score
5) high-income earners with high spending score

```
#1) # using K-Means
from sklearn.cluster import KMeans

features = mall_customers_df[['Annual Income (k$)', 'Spending Score (1-100)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
# Choose the number of clusters (K)
k = 5

# Initialize KMeans
kmeans = KMeans(n_clusters=k, random_state=42)

# Fit the model to the data
kmeans.fit(scaled_features)

# Get cluster assignments for each data point
cluster_labels = kmeans.labels_

plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=cluster_labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='X', color='red', s=200)
plt.xlabel('Annual Income (scaled)')
plt.ylabel('Spending Score (scaled)')
plt.title('K-means Clustering')
plt.show()
```

a.2) Density-Based Spatial Clustering of Applications with Noise (DBSCAN) – This is a popular choice for clustering of datasets. Unlike K-means, DBSCAN automatically determines the number of clusters and identifies outliers and exclude them from clustering.

```
#2) Using Density Based Spatial Clustering of Applications with Noise (DBSCAN)
# Popular choice for clustering of datasets
# Unlike K Means, DBSCAN automatically determines the number of clusters
# DBSCAN can automatically identifies outliers and exclude them from the clustering
from sklearn.cluster import DBSCAN

# Assume you're interested in two features: 'Annual Income' and 'Spending Score'
# Both are crucial features for understanding the customer buying behavior
# Both are Linear such as Customers with higher-income may spend more.
X = mall_customers_df[["Annual Income (k$)", "Spending Score (1-100)"]]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

# Initialize DBSCAN
# eps=5 maximum distance between data points that may considered them within the same cluster.
dbscan = DBSCAN(eps=5)  # Adjust parameters as needed

# Fit the model
dbscan.fit(scaled_features)

# Get cluster labels (-1 indicates noise/outliers)
cluster_labels = dbscan.labels_

# Visualize the clusters
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=cluster_labels, cmap="viridis")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("DBSCAN Clustering")
plt.show()


# Number of clusters (excluding noise points)
num_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
print(f"Estimated number of clusters: {num_clusters}")

# Identify noise points (outliers)
num_noise = list(cluster_labels).count(-1)
print(f"Estimated number of noise points: {num_noise}")
```

a.3) Gaussian Mixture Model – this assigns each data point to a single cluster.

```
#3) Using Gaussian Mixture Model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import seaborn as sns

# Assume you're interested in two features: 'Annual Income' and 'Spending Score'
X = mall_customers_df[["Annual Income (k$)", "Spending Score (1-100)"]]

# Initialize Gaussian Mixture Model
gmm = GaussianMixture(n_components=4, random_state=2021)  # Specify the number of clusters

# Fit the model
gmm.fit(X)

# Predict cluster labels
cluster_labels = gmm.predict(X)

# Add cluster labels to the original dataframe
mall_customers_df["Cluster"] = cluster_labels

# Visualize the clusters
plt.figure(figsize=(9, 7))
sns.scatterplot(data=mall_customers_df, x="Annual Income (k$)", y="Spending Score (1-100)", hue="Cluster", palette=["red", "blu
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("Customer Segmentation using Gaussian Mixture Model")
plt.savefig("Customer_Segmentation_GMM_Python.png", format="png", dpi=150)
plt.show()
```

b.PCA is a technique for reducing the dimensionality of the dataset while ensuring the preservation of most variance. PCA will create a new set of features that will capture most of the important data points from the selected features.

Steps involved in PCA:

1. Identify the features to combine and reduce to a new set of features.
2. Normalize the features by using StandardScaler. This will give you the scaled values from the feature
3. Use the PCA module and identify the number of components/features to create.
4. Fit and transform the scaled values. This will create new features that contains most of the datapoints from the selected features.
5. Use scatter plot to visualize the new features.

```
# Normalize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)
```

```
from sklearn.decomposition import PCA

# Initialize PCA up to 2 components
# This will create 2 columns/components
pca = PCA(n_components=2)

# Fit and transform the scaled features
pca_result = pca.fit_transform(scaled_features)

# Create a DataFrame with the PCA results
plt.scatter(pca_result[:,0], pca_result[:,1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Principal Component Analysis (PCA) Result with Age')
plt.show()
```

c. LDA reduction for classification and dimensionality is particularly useful when you have a labelled dataset and want to have linear features that separate the classes.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Select features for LDA (including Age, Annual Income, and Spending Score)
X = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values

# Use Gender as the target feature. LDA works better with Categorical variables
# Replacing 1 and 0 to Categorical value which is Male and Female.
y = mall_customers_df['Gender'].values

# Standardize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

# Apply LDA with n_components=1
lda = LinearDiscriminantAnalysis(n_components=1)

# lda_result is the reduced-dimensional space
lda_result = lda.fit_transform(scaled_features, y)

lda_coefficients = lda.coef_
print("LDA Coefficients:", lda_coefficients)

# Create a DataFrame with the LDA results
#lda_df = pd.DataFrame(lda_result, columns=['Gender'])

# Visualize LDA results
plt.scatter(lda_result[:,0], y)
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Gender')
plt.title('LDA Results')
plt.show()
```

d. Cluster plot – this method helps to visually inspect the data points assigned in different clusters. It also helps in the assessment of the clustering algorithm, which separates the groups in identifying distinct patterns.

```
# Using Cluster Plot
x = "Annual Income (k$)"
y = "Spending Score (1-100)"
hue = "Gender"

mall_customer_gender_df = mall_customers_df.copy()

# Create the scatter plot
plt.figure(figsize=(10, 8))

# The mall_customer_gender_df is a segmented data based on the Gender (Male/Female) variable.
sns.scatterplot(data=mall_customer_gender_df, x=x, y=y, hue=hue)
plt.xlabel(x)
plt.ylabel(y)
plt.title("Mall Customers Segmentation")
plt.legend(loc="upper right")
plt.grid()
plt.show()
```

Box plot – it provides insights into the distribution of each feature within the cluster.

```
# Using Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x="Cluster", y="Annual Income (k$)", data=mall_customers_df)
plt.xlabel("Cluster")
plt.ylabel("Annual Income")
plt.title("Box Plot of Annual Income by Cluster")
plt.show()
```

Task 3: Conclusion

Segmentation is an important tool in data categorization. One aspect to consider is the degree of heterogeneity within the dataset about population diversity, data variability, and economic and social heterogeneity. This helps businesses tailor their products and services to meet the needs of the customers, which is the second aspect of segmentation. The right market segmentation is a tool for the right marketing strategy in targeting the right customers. Its primary focus is to study consumer behavior and product or service engagement. To achieve this goal, it is important to know the key features, which include consumer preferences and spending ability.

Appendix A : Classification

Task 1 Data Preparation

In [38]:
```python
import numpy as np
import pandas as pd
```

In [39]:
```python
#a) Dataset and code snipet
bank_churn_df = pd.read_csv('bank_churn_prediction.csv')
bank_churn_df
```

Out[39]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card
0	15634602	619	France	Female	42	2	0.00	1.0	1
1	15647311	608	Spain	Female	41	1	83807.86	1.0	0
2	15619304	502	France	Female	42	8	159660.80	3.0	1
3	15701354	699	France	Female	39	1	0.00	2.0	0
4	15737888	850	Spain	Female	43	2	125510.82	1.0	1
...	...	...	...	...	...	...	...	...	...
9995	15606229	771	France	Male	39	5	0.00	2.0	1
9996	15569892	516	France	Male	35	10	57369.61	1.0	1
9997	15584532	709	France	Female	36	7	0.00	1.0	0
9998	15682355	772	Germany	Male	42	3	75075.31	2.0	1
9999	15628319	792	France	Female	28	4	130142.79	1.0	1

10000 rows × 12 columns

In [40]:
```python
bank_churn_df.shape
```

Out[40]:
```
(10000, 12)
```

In [41]:
```python
bank_churn_df.head()
```

Out[41]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	act
0	15634602	619	France	Female	42	2	0.00	1.0	1	
1	15647311	608	Spain	Female	41	1	83807.86	1.0	0	
2	15619304	502	France	Female	42	8	159660.80	3.0	1	
3	15701354	699	France	Female	39	1	0.00	2.0	0	
4	15737888	850	Spain	Female	43	2	125510.82	1.0	1	

In [42]:
```
bank_churn_df.tail()
```

Out[42]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card
9995	15606229	771	France	Male	39	5	0.00	2.0	1
9996	15569892	516	France	Male	35	10	57369.61	1.0	1
9997	15584532	709	France	Female	36	7	0.00	1.0	0
9998	15682355	772	Germany	Male	42	3	75075.31	2.0	1
9999	15628319	792	France	Female	28	4	130142.79	1.0	1

In [43]:
```
#b) Two techniques or strategies to handle missing values effectively in the dataset.
#1 Identify missing values using isnull().sum() function
bank_churn_df.isnull().sum()
```

Out[43]:
```
customer_id         0
credit_score        0
country             2
gender              0
age                 0
tenure              0
balance             0
products_number     2
credit_card         0
active_member       0
estimated_salary    3
churn               0
dtype: int64
```

In [44]:
```
#2) using dropna to columns country, products_number and estimated_salary
bank_churn_df = bank_churn_df.dropna(subset=['country','products_number','estimated_s
bank_churn_df.isnull().sum()
```

Out[44]:
```
customer_id         0
credit_score        0
country             0
gender              0
age                 0
tenure              0
balance             0
products_number     0
credit_card         0
active_member       0
estimated_salary    0
churn               0
dtype: int64
```

In [45]:
```
bank_churn_df.shape
```

Out[45]:
```
(9993, 12)
```

In [46]:
```
#c) Handling different data types
bank_churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9993 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customer_id       9993 non-null   int64  
 1   credit_score      9993 non-null   int64  
 2   country           9993 non-null   object 
 3   gender            9993 non-null   object 
 4   age               9993 non-null   int64  
 5   tenure            9993 non-null   int64  
 6   balance           9993 non-null   float64
 7   products_number   9993 non-null   float64
 8   credit_card       9993 non-null   int64  
 9   active_member     9993 non-null   int64  
 10  estimated_salary  9993 non-null   float64
 11  churn             9993 non-null   int64  
dtypes: float64(3), int64(7), object(2)
memory usage: 1014.9+ KB
```

In [47]:
```
bank_churn_df.dtypes
```

Out[47]:
```
customer_id           int64
credit_score          int64
country              object
gender               object
age                   int64
tenure                int64
balance             float64
products_number     float64
credit_card           int64
active_member         int64
estimated_salary    float64
churn                 int64
dtype: object
```

In [48]:
```python
#using one-hot encoding.
bank_churn_df = pd.get_dummies(bank_churn_df, columns=['gender', 'country'], drop_fir
bank_churn_df.head(11000)
```

Out[48]:

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_member	est
0	15634602	619	42	2	0	1	1	1	
1	15647311	608	41	1	83807	1	0	1	
2	15619304	502	42	8	159660	3	1	0	
3	15701354	699	39	1	0	2	0	0	
4	15737888	850	43	2	125510	1	1	1	
...	...	...	...	...	...	...	...	...	
9995	15606229	771	39	5	0	2	1	0	
9996	15569892	516	35	10	57369	1	1	1	
9997	15584532	709	36	7	0	1	0	1	
9998	15682355	772	42	3	75075	2	1	0	
9999	15628319	792	28	4	130142	1	1	0	

9993 rows × 13 columns

In [49]:
```python
bank_churn_df.describe()
```

Out[49]:

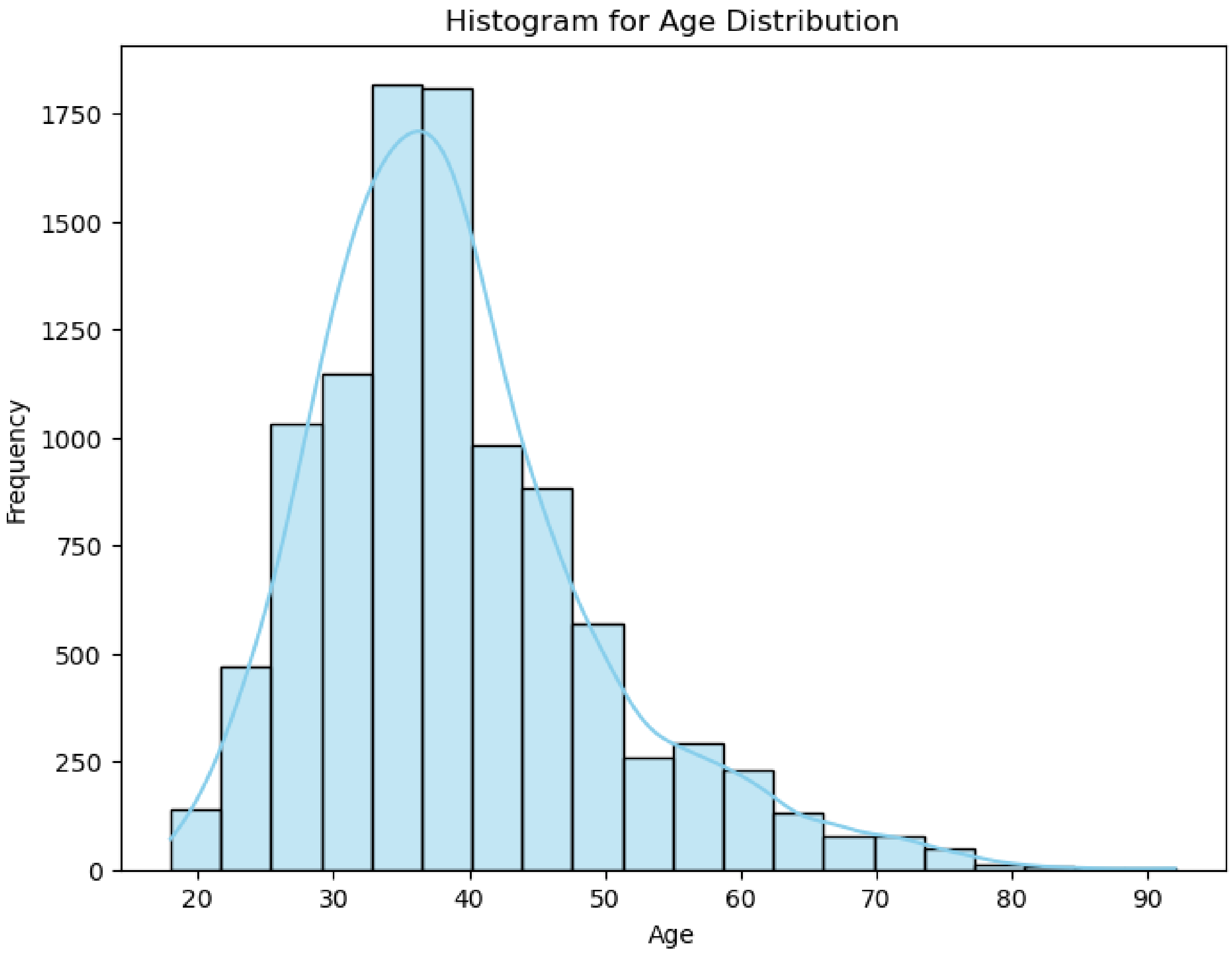
	customer_id	credit_score	age	tenure	balance	products_number	credit_ca
count	9.993000e+03	9993.000000	9993.000000	9993.000000	9993.000000	9993.000000	9993.00000
mean	1.569096e+07	650.573201	38.922646	5.011308	76488.567397	1.530271	0.70559
std	7.194598e+04	96.648965	10.488991	2.890961	62404.962061	0.581704	0.45579
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000
25%	1.562852e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000
50%	1.569074e+07	652.000000	37.000000	5.000000	97234.000000	1.000000	1.00000
75%	1.575333e+07	718.000000	44.000000	7.000000	127649.000000	2.000000	1.00000
max	1.581569e+07	850.000000	92.000000	10.000000	250898.000000	4.000000	1.00000

In [50]:
```python
#No more categorical variables
bank_churn_df.dtypes
```

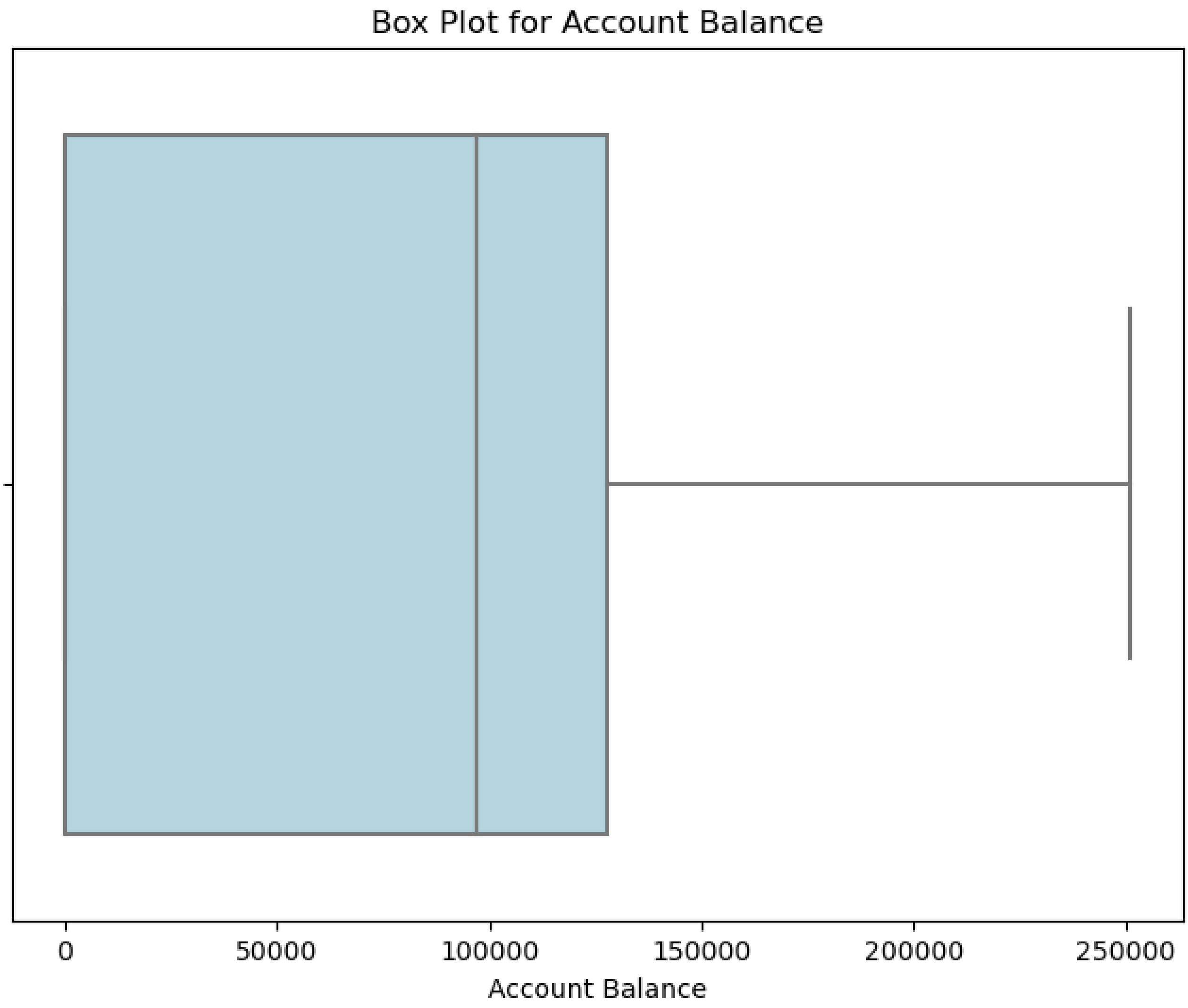
Out[50]:
```
customer_id         int32
credit_score        int32
age                 int32
tenure              int32
balance             int32
products_number     int32
credit_card         int32
active_member       int32
estimated_salary    int32
churn               int32
Gender_Male         int32
Country_Germany     int32
Country_Spain       int32
dtype: object
```

In [51]:
```
#d) Apply three specific data visualizations techniques to analyze the data distribut
import seaborn as sns
import matplotlib.pyplot as plt
```

In [52]:
```
# Histogram for Age distribution
plt.figure(figsize=(8, 6))
sns.histplot(bank_churn_df['age'], bins=20, kde=True, color='skyblue')
plt.title('Histogram for Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



In [30]:
```
# Box Plot for Balance Distribution
plt.figure(figsize=(8, 6))
sns.boxplot(x='balance', data=bank_churn_df, color='lightblue')
plt.title('Box Plot for Account Balance')
plt.xlabel('Account Balance')
plt.show()
```

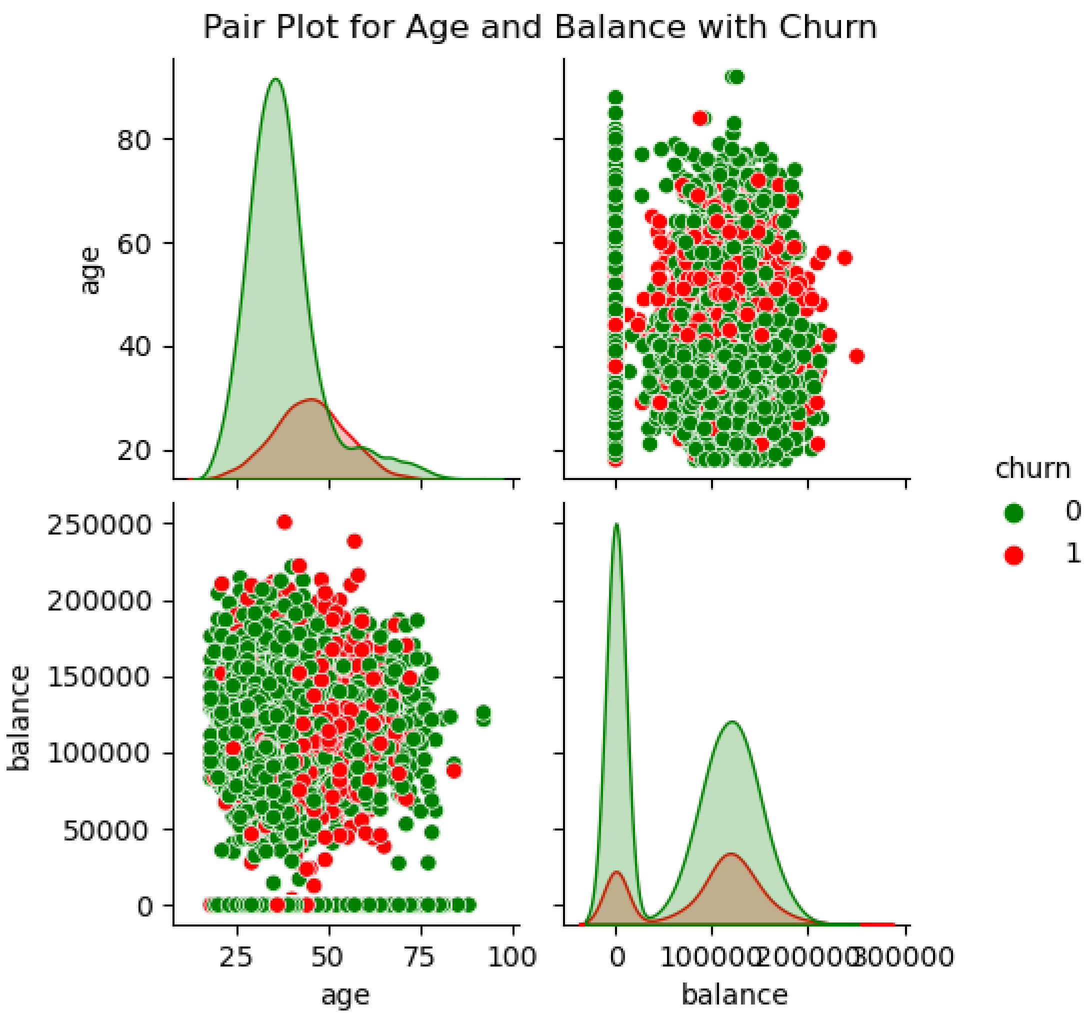


In [ ]:

In [31]:
```
# Pair Plot for Age and Balance with Churn as hue. This is to explore the relationshi
plt.figure(figsize=(10, 8))
sns.pairplot(bank_churn_df[['age', 'balance','churn']], hue='churn', palette={0: 'gre
plt.suptitle('Pair Plot for Age and Balance with Churn', y=1.02)
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

```
<Figure size 1000x800 with 0 Axes>
```



Task 2 Feature Engineering

In [53]:
```
bank_churn_df
```

Out[53]:

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_member	est
0	15634602	619	42	2	0	1	1	1	
1	15647311	608	41	1	83807	1	0	1	
2	15619304	502	42	8	159660	3	1	0	
3	15701354	699	39	1	0	2	0	0	
4	15737888	850	43	2	125510	1	1	1	
...	...	...	...	...	...	...	...	...	
9995	15606229	771	39	5	0	2	1	0	
9996	15569892	516	35	10	57369	1	1	1	
9997	15584532	709	36	7	0	1	0	1	
9998	15682355	772	42	3	75075	2	1	0	
9999	15628319	792	28	4	130142	1	1	0	

9993 rows × 13 columns

In [54]:
```
#Implement 2 most relevant feature selection techniques.
#a) using Recursive Feature Elimination (RFE)
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# excluding churn as it is the target feature and customer_id as it is not needed.
X = bank_churn_df.drop(['churn','customer_id'],axis=1)
y = bank_churn_df['churn']

# splitting result for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# create model using LogisticRegression
log_reg_model = LogisticRegression(max_iter=120000)

# Using RFE to identify the most important features for a given predictive model.
# Importance will be based on the coefficients or feature importances.
# RFE will train the log_reg_model on the entire set of features.
# We only use 5 features for this training. 5 is just arbitrary.
rfe = RFE(log_reg_model, n_features_to_select=5)
# Train the model
rfe.fit(X_train, y_train)
selected = X_train.columns[rfe.support_]

print("Using Recursive Feature Elimination (RFE)")
print("Feature Selected")
print(selected)
```

```
Using Recursive Feature Elimination (RFE)
Feature Selected
Index(['products_number', 'credit_card', 'active_member', 'Gender_Male',
       'Country_Germany'],
      dtype='object')
```

In [55]:
```
#a) using Feature Importance from Tree-based Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# excluding churn as it is the target feature and customer_id as it is not needed.
X = bank_churn_df.drop(['churn','customer_id'], axis=1)
y = bank_churn_df['churn']

# splitting result for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Using Random Forest model to set importance
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
random_forest_model.fit(X_train, y_train)

# Set the feature importance in a dataframe
feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': random_forest_
# Sort features by importance
feature_importances = feature_importances.sort_values(by='Importance', ascending=Fals

print("Using Feature Importance from Tree-based Models")
# Getting only the top 5 most important features
print(feature_importances.head(5))
```

```
Using Feature Importance from Tree-based Models
            Feature  Importance
1               age    0.238673
7  estimated_salary    0.147322
3           balance    0.143126
0      credit_score    0.141037
4   products_number    0.132226
```

In [56]:
```
#b) Feature scaling method on a subset of features
from sklearn.preprocessing import MinMaxScaler

# Create new DataFrame to avoid changing the original dataframe
minmaxscalar_df = bank_churn_df.copy()

# Scaling these features to normalize numerical values.
features_to_scale = ['age', 'balance', 'credit_score', 'estimated_salary', 'tenure',

# Extract the subset of features
subset_features = minmaxscalar_df[features_to_scale].values

# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to ho
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)

# Replace the original values with scaled values in the DataFrame
minmaxscalar_df[features_to_scale] = scaled_features

# Display the DataFrame after scaling
minmaxscalar_df

# Q. How MinMaxScaler affects the Data Distribution
# A. It compacts the numerical values and lessen the difference between them. It ensu

# Q. Impact on model training
# A. It helps the model to train with normalized values. It helps model from convergi
```

Out[56]:

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_membe
0	15634602	0.538	0.324324	0.2	0.000000	0.000000	1	
1	15647311	0.516	0.310811	0.1	0.334028	0.000000	0	
2	15619304	0.304	0.324324	0.8	0.636354	0.666667	1	
3	15701354	0.698	0.283784	0.1	0.000000	0.333333	0	
4	15737888	1.000	0.337838	0.2	0.500243	0.000000	1	
...	...	...	...	...	...	...	...	.
9995	15606229	0.842	0.283784	0.5	0.000000	0.333333	1	
9996	15569892	0.332	0.229730	1.0	0.228655	0.000000	1	
9997	15584532	0.718	0.243243	0.7	0.000000	0.000000	0	
9998	15682355	0.844	0.324324	0.3	0.299225	0.333333	1	
9999	15628319	0.884	0.135135	0.4	0.518705	0.000000	1	

9993 rows × 13 columns

Task 3 Model Building and Prediction

In [57]:

```
#a.1) using Logistic Regression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Copy bank_churn_df DataFrame to avoid changing the original dataframe
log_reg_df = bank_churn_df.copy()

# Scale the selected feature to make sure numerical values are in uniform or close sc
features_to_scale = ['age', 'balance', 'credit_score', 'estimated_salary', 'tenure',

# Extract the subset of features
subset_features = log_reg_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to h
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)

# Replace the original values with scaled values in the DataFrame
log_reg_df[features_to_scale] = scaled_features

# excluding churn as it is the target feature and customer_id as it is not needed.
X = log_reg_df.drop(['churn','customer_id'],axis=1)
y = log_reg_df['churn']

# splitting result for training and testing
# test_size determines the percentage of test data to extract from the X dataframe. T
# test_size used is 0.2 or 20%
# random_state dictates the consistency of the randomness process of the model. It en
# I used 42 for random_state as it is just the common number used.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# create model using LogisticRegression
# set multi_class = multinomial since the dataset is not just binary classification (
# use lbfgs to solve small to medium size data
log_reg_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_ite

# Train the model
log_reg_model.fit(X_train, y_train)

prediction = log_reg_model.predict(X_test)

# Comparing the prediction result vs the y_test data (churn).
# counts the number of the same result between y_test and prediction and divide it by
# ex. y_test = [0, 1, 2, 4, 5], prediction = [0, 3, 2, 3, 5]
# results to 3 matches over 5 items result to 0.6 accuracy
accuracy = accuracy_score(y_test, prediction)
classification_report_output = classification_report(y_test, prediction)

print("Using LogisticRegression")
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_output)
```

```
Using LogisticRegression
Accuracy: 0.81
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      1579
           1       0.61      0.22      0.33       420

    accuracy                           0.81      1999
   macro avg       0.72      0.59      0.61      1999
weighted avg       0.78      0.81      0.77      1999
```

In [60]:
```
#a.2) using RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Copy bank_churn_df DataFrame to avoid changing the original dataframe
random_forest_df = bank_churn_df.copy()
# Scale the selected feature to make sure numerical values are in uniform or close sca
features_to_scale = ['age', 'balance', 'credit_score', 'estimated_salary', 'tenure',

# Extract the subset of features
subset_features = random_forest_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to ha
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)

# Replace the original values with scaled values in the DataFrame
random_forest_df[features_to_scale] = scaled_features

# excluding churn as it is the target feature and customer_id as it is not needed.
X = random_forest_df.drop(['customer_id','churn'], axis=1)
y = random_forest_df['churn']

# splitting result for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Creating model with n_estimators=100 and random_state=42.
# n_estimator provides the number of decision trees to be used by the forest.
# the highter the n_estimator, the higher/better result of the model but it also incre
# I used 100 for the n_estimators as it is the common number which provides a good tra
# random_state dictates the consistency of the randomness process of the model. It ens
# I used 42 for random_state as it is just the common number used.
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
random_forest_model.fit(X_train, y_train)


# based from the 100 trees created (n_estimators), it will then checks using random en
prediction = random_forest_model.predict(X_test)

# Compares the prediction result vs the y_test data (churn).
# counts the number of the same result between y_test and prediction and divide it by
# ex. y_test = [0, 1, 2, 4, 5], prediction = [0, 3, 2, 3, 5]
# results to 3 matches over 5 items result to 0.6 accuracy
accuracy = accuracy_score(y_test, prediction)

classification_report_output = classification_report(y_test, prediction)

print("Using RandomForestClassifier")
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_output)
```

```
Using RandomForestClassifier
Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1579
           1       0.81      0.49      0.61       420

    accuracy                           0.87      1999
   macro avg       0.84      0.73      0.76      1999
weighted avg       0.86      0.87      0.85      1999
```

In [59]:

```python
#a.3) using DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Copy bank_churn_df DataFrame to avoid changing the original dataframe
tree_df = bank_churn_df.copy()
# Scale the selected feature to make sure numerical values are in uniform or close sc
features_to_scale = ['age', 'balance', 'credit_score', 'estimated_salary', 'tenure',

# Extract the subset of features
subset_features = tree_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to h
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)

# Replace the original values with scaled values in the DataFrame
tree_df[features_to_scale] = scaled_features

# excluding churn as it is the target feature and customer_id as it is not needed.
X = tree_df.drop(['customer_id','churn'], axis=1)
y = tree_df['churn']

# splitting result for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Creating model with max_depth=3.
# max_depth sets the limit of tree to create
# the highter the max_depth, the more patterns to use in the training data but may le
decision_tree_model = DecisionTreeClassifier(max_depth = 3)

# Train the model
decision_tree_model.fit(X_train, y_train)


# based from the 100 trees created (n_estimators), it will then checks using random e
prediction = decision_tree_model.predict(X_test)

# Compares the prediction result vs the y_test data (churn).
# counts the number of the same result between y_test and prediction and divide it by
# ex. y_test = [0, 1, 2, 4, 5], prediction = [0, 3, 2, 3, 5]
# results to 3 matches over 5 items result to 0.6 accuracy
accuracy = accuracy_score(y_test, prediction)

classification_report_output = classification_report(y_test, prediction)

print("Using DecisionTreeClassifier")
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_output)
```

```
Using DecisionTreeClassifier
Accuracy: 0.83
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.99      0.90      1579
           1       0.88      0.23      0.36       420

    accuracy                           0.83      1999
   macro avg       0.85      0.61      0.63      1999
weighted avg       0.84      0.83      0.79      1999
```

#b)

In [ ]:

```

```

In [ ]:

```

```

Appendix B : Regression

Task 1 Data Preparation

In [3]:
```
import numpy as np
import pandas as pd
```

In [10]:
```
#a) Load dataset
advertising_df = pd.read_csv('advertising.csv')
advertising_df
```

Out[10]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

200 rows × 4 columns

In [11]:
```
advertising_df.shape
```

Out[11]:
```
(200, 4)
```

In [12]:
```
advertising_df.head()
```

Out[12]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

In [13]:
```
advertising_df.tail()
```

Out[13]:

	TV	Radio	Newspaper	Sales
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

In [6]:
```
#b) Two techniques or strategies to handle missing values effectively in the dataset.
#1 Identify missing values using isnull().sum() function
advertising_df.isnull().sum()
```

Out[6]:
```
TV           0
Radio        6
Newspaper    2
Sales        0
dtype: int64
```

In [7]:
```
#2) using dropna to columns country, products_number and estimated_salary to remove NA
advertising_df = advertising_df.dropna(subset=['Radio','Newspaper'], axis=0)
advertising_df.isnull().sum()
```

Out[7]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

In [8]:
```
advertising_df.shape
```

Out[8]:
```
(194, 4)
```

In [10]:
```
#c) Using .info to check for inconsistencies like wrong value type per each Numerical
advertising_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 194 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         194 non-null    float64
 1   Radio      194 non-null    float64
 2   Newspaper  194 non-null    float64
 3   Sales      194 non-null    float64
dtypes: float64(4)
memory usage: 7.6 KB
```

In [11]:
```
advertising_df
```

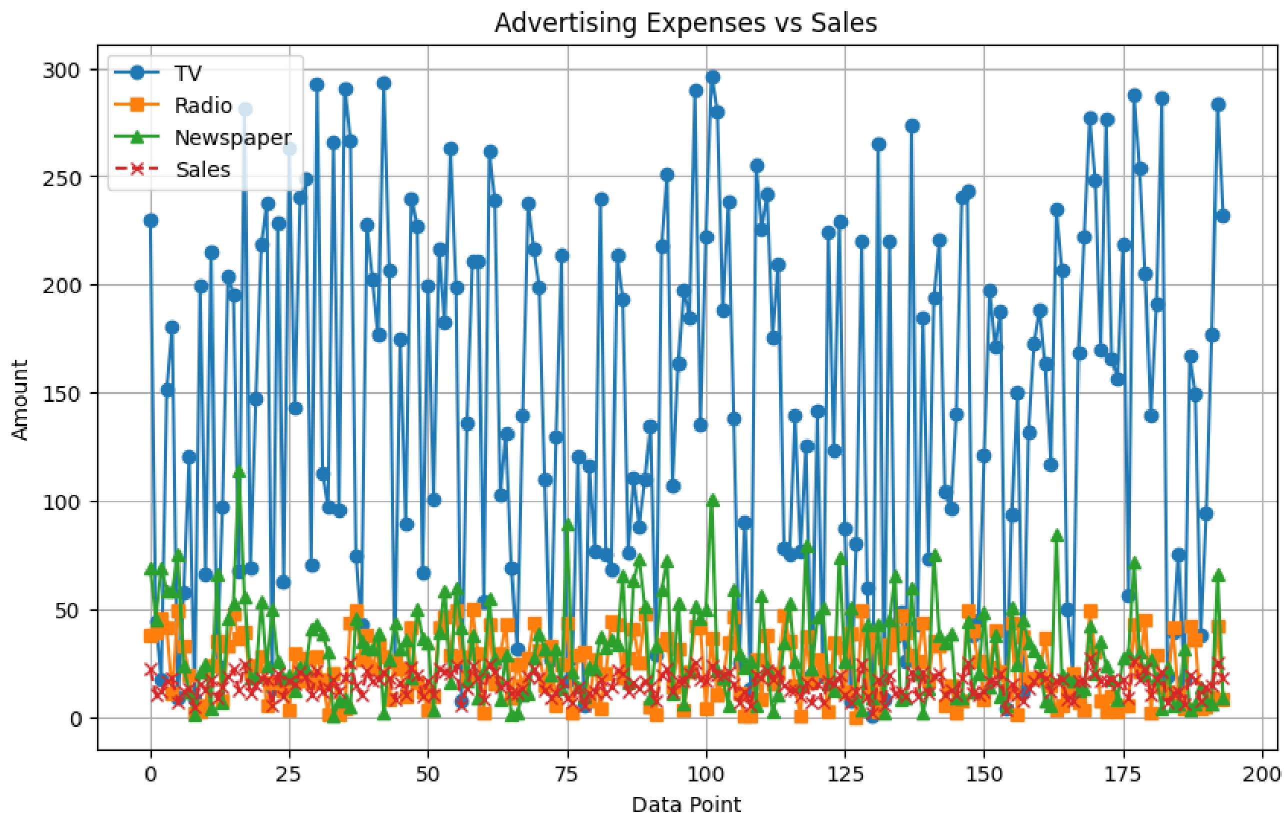
Out[11]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

194 rows × 4 columns

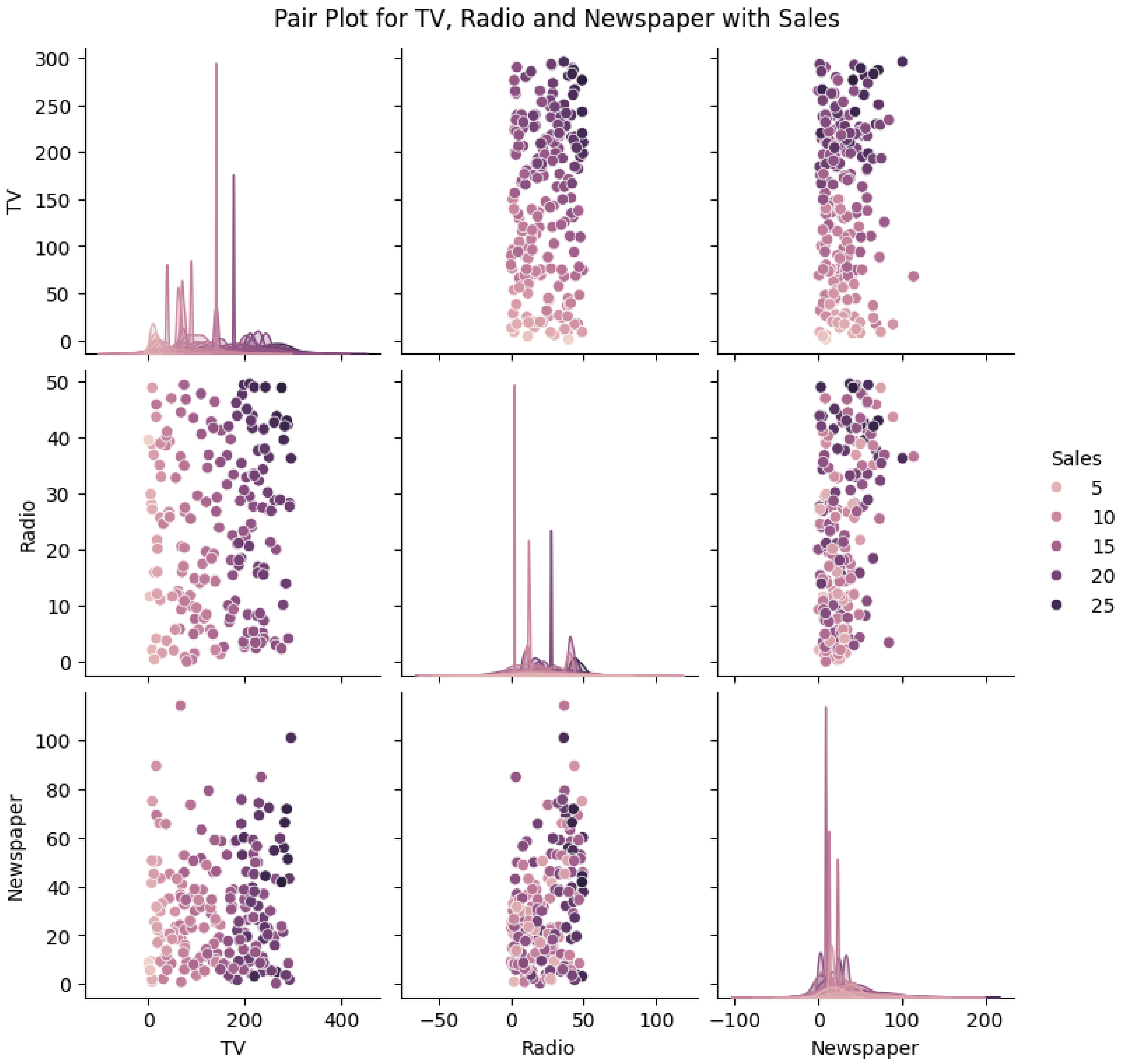
In [12]:
```
#d) Apply three specific data visualizations techniques to analyze the data distribut
import seaborn as sns
import matplotlib.pyplot as plt
```

In [21]:
```
# Using Line plot to see the trend of marketing expenses such as TV, Radio, and Newspa
x_trend = list(range(0, len(advertising_df)))

plt.figure(figsize=(10,6))

plt.plot(x_trend, advertising_df['TV'], label='TV', marker='o')
plt.plot(x_trend, advertising_df['Radio'], label='Radio', marker='s')
plt.plot(x_trend, advertising_df['Newspaper'], label='Newspaper', marker='^')
plt.plot(x_trend, advertising_df['Sales'], label='Sales', linestyle='--', marker='x')

plt.title('Advertising Expenses vs Sales')
plt.xlabel('Data Point')
plt.ylabel('Amount')
plt.grid(True)
plt.legend()
plt.show()
```



In [22]:
```
# Using Pair Plot for TV, Radio and Newspaper on Sales.
plt.figure(figsize=(10, 8))
sns.pairplot(advertising_df[['TV', 'Radio', 'Newspaper', 'Sales']], hue='Sales', diag_
plt.suptitle('Pair Plot for TV, Radio and Newspaper with Sales', y=1.02)
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



Task 2 Feature Engineering

In [12]:
```
advertising_df
```

Out[12]:

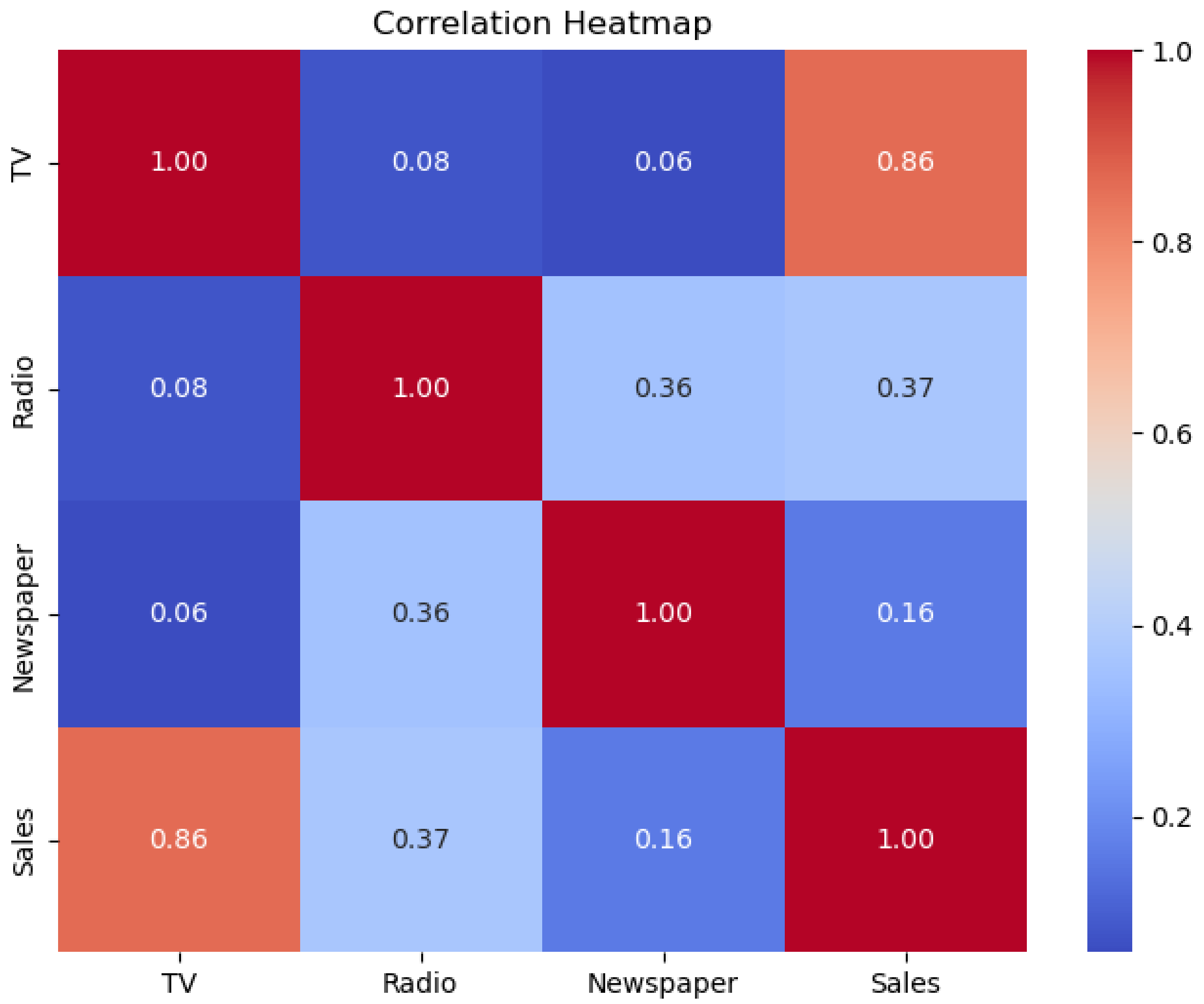
	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

194 rows × 4 columns

In [17]:
```
#a Implement one feature selection technique - Using Filter-based feature selection.
#  Calculate correlation using the Pearson correlation coefficient with TV, Radio and
#  If the correlation between them is high (closest to 1), then we consider the featu
#  If the correlation between them is low (closest to 0), then we exclude the feature
correlation_matrix = advertising_df.corr()
# excluding Sales in the result, and considering only correlation above 0.4, We can o
correlation_matrix['Sales']
```

Out[17]:
```
TV           0.860370
Radio        0.368008
Newspaper    0.159587
Sales        1.000000
Name: Sales, dtype: float64
```

In [16]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))  # Optional: Set the figure size
sns.heatmap(correlation_matrix, cmap="coolwarm", annot=True, fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



In [31]:

```python
#b) Feature scaling method on a subset of features - Using Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler

# Scaling these features to normalize numerical values.
# We didn't include Sales as we want to predict the Sales outcome
features_to_scale = ['TV', 'Radio', 'Newspaper']

# Extract the subset of features
subset_features = advertising_df[features_to_scale].values

# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to h
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)

# Replace the original values with scaled values in the DataFrame
minmaxscalar_df = pd.DataFrame(scaled_features, columns=features_to_scale)

print('Before Min-Max Scaler')
print(advertising_df)
print('After Min-Max Scaler')
print(minmaxscalar_df)
```

```
Before Min-Max Scaler
        TV  Radio  Newspaper  Sales
0    230.1   37.8       69.2   22.1
1     44.5   39.3       45.1   10.4
2     17.2   45.9       69.3   12.0
3    151.5   41.3       58.5   16.5
4    180.8   10.8       58.4   17.9
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[194 rows x 4 columns]
After Min-Max Scaler
           TV     Radio  Newspaper
0    0.775786  0.762097   0.605981
1    0.148123  0.792339   0.394019
2    0.055800  0.925403   0.606860
3    0.509976  0.832661   0.511873
4    0.609063  0.217742   0.510994
..        ...       ...        ...
189  0.126818  0.074597   0.118734
190  0.316199  0.098790   0.068602
191  0.596212  0.187500   0.053650
192  0.956713  0.846774   0.579595
193  0.782550  0.173387   0.073879

[194 rows x 3 columns]
```

Task 3 Model Building and Prediction

In [32]:
```
#a.1) using Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Scale the selected feature to make sure numerical values are in uniform or close sc
features_to_scale = ['TV', 'Radio', 'Newspaper']
# Extract the subset of features
subset_features = advertising_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to h
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)
# Replace the original values with scaled values in the DataFrame
linear_df = pd.DataFrame(scaled_features, columns=features_to_scale)

# use the scaled dataframe for the X axis
X = linear_df
# use the Sales column in original Advertising dataframe
y = advertising_df['Sales']

# splitting result for training and testing
# test_size determines the percentage of test data to extract from the X dataframe. T
# test_size used is 0.2 or 20%
# random_state dictates the consistency of the randomness process of the model. It en
# I used 42 for random_state as it is just the common number used.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# create model using LinearRegression
# We assume that the relationship between the Advertisement Expenses (TV, Radio, and 
# We can make Sales predictions based on Advertisement Budgets.
linear_reg_model = LinearRegression()

# Train the model
linear_reg_model.fit(X_train, y_train)

prediction = linear_reg_model.predict(X_test)

print("Using LinearRegression")
# Evaluate the model using mean squared error and R-squared

# Mean Squared Error (MSE) measures the amount of error in statistical models.
# It assess the average square difference between the predicted and actual values
# A smaller MSE indicates that the model's predictions are closer to the actual value
mse = mean_squared_error(y_test, prediction)

# R-Squared measures the proportion between the dependent variable (Sales) and the in
# The result ranges from 0 - 1 (0% to 100%)
# The closer the result to 0, it means it doesnt correlate to the dependent variable
# The closer the result to 1, it means it gives higher correlation to the dependent v
# The higher the result, the better the regression model we can use for observations
r2 = r2_score(y_test, prediction)
print(f"Mean Squared Error: {mse:.2f}")

#R-Squared ranges from 0 to 1
#When value is 0, the model explains none of the variance in the target variable
#When value is 1, the model perfectly predicts the target variable
print(f"R-squared: {r2:.2f}")
```

```
Using LinearRegression
Mean Squared Error: 2.23
R-squared: 0.91
```

In [39]:

```
#a.2) using RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Scale the selected feature to make sure numerical values are in uniform or close sca
features_to_scale = ['TV', 'Radio', 'Newspaper']
# Extract the subset of features
subset_features = advertising_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to ha
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)
# Replace the original values with scaled values in the DataFrame
randomforest_df = pd.DataFrame(scaled_features, columns=features_to_scale)

# use the scaled dataframe for the X axis
X = randomforest_df
# use the Sales column in original Advertising dataframe
y = advertising_df['Sales']

# splitting result for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Creating model with n_estimators=100 and random_state=42.
# n_estimator provides the number of decision tress to be used by the forest.
# the highter the n_estimator, the higher/better result of the model but it also incre
# I used 100 for the n_estimators as it is the common number used that provides a good
# random_state dictates the consistency of the randomness process of the model. It ens
# I used 42 for random_state as it is just the common number used.
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
random_forest_model.fit(X_train, y_train)


# based from the 100 trees created (n_estimators), it will then checks using random en
prediction = random_forest_model.predict(X_test)

print("Using RandomForestRegressor")
# Evaluate the model using mean squared error and R-squared

# Mean Squared Error (MSE) measures the amount of error in statistical models.
# It assess the average square difference between the predicted and actual values
# A smaller MSE indicates that the model's predictions are closer to the actual values
mse = mean_squared_error(y_test, prediction)
print(f"Mean Squared Error: {mse:.2f}")

# R-Squared measures the proportion between the dependent variable (Sales) that explai
# The result ranges from 0 - 1 (0% to 100%)
# When value is 0, the model explains none of the variance is important in the target
# When value is closer to 1, the model perfectly predicts the target variable
# The higher the result, the better the regression model we can use for our observatio
r2 = r2_score(y_test, prediction)
print(f"R-squared: {r2:.2f}")
```

```
Using RandomForestRegressor
Mean Squared Error: 1.55
R-squared: 0.94
```

In [40]:

```
#a.3) using Support Vector Regression
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Scale the selected feature to make sure numerical values are in uniform or close sc
features_to_scale = ['TV', 'Radio', 'Newspaper']
# Extract the subset of features
subset_features = advertising_df[features_to_scale].values
# using Min-Max Scaling to normalize numerical values to a uniform scale. We aim to h
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(subset_features)
# Replace the original values with scaled values in the DataFrame
svr_df = pd.DataFrame(scaled_features, columns=features_to_scale)

# use the scaled dataframe for the X axis
X = svr_df
# use the Sales column in original Advertising dataframe
y = advertising_df['Sales']

# splitting result for training and testing
# test_size determines the percentage of test data to extract from the X dataframe. T
# test_size used is 0.2 or 20%
# random_state dictates the consistency of the randomness process of the model. It en
# I used 42 for random_state as it is just the common number used.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# SVR can handle linear and non linear regression. Even if the relationship between S
svr_model = SVR(kernel="linear", C=1.0)

# Train the model
svr_model.fit(X_train, y_train)

prediction = svr_model.predict(X_test)

# Evaluate the model using mean squared error and R-squared
# A smaller MSE indicates that the model's predictions are closer to the actual value
mse = mean_squared_error(y_test, prediction)
r2 = r2_score(y_test, prediction)
print(f"Mean Squared Error: {mse:.2f}")

#R-Squared ranges from 0 to 1
#When value is 0, the model explains none of the variance is important in the target
#When value is 1, the model perfectly predicts the target variable
print(f"R-squared: {r2:.2f}")
```

```
Mean Squared Error: 2.59
R-squared: 0.89
```

Appendix C: Time-series Analysis

Task 1: Data Exploration

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pmdarima as pm
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [2]:
```
# a) explore the data set
gold_df = pd.read_csv('gold_price_data.csv')
gold_df
```

Out[2]:

	Date	Value
0	1970-01-01	35.20
1	1970-04-01	35.10
2	1970-07-01	35.40
3	1970-10-01	36.20
4	1971-01-01	37.40
...	...	...
10782	2020-03-09	1672.50
10783	2020-03-10	1655.70
10784	2020-03-11	1653.75
10785	2020-03-12	1570.70
10786	2020-03-13	1562.80

10787 rows × 2 columns

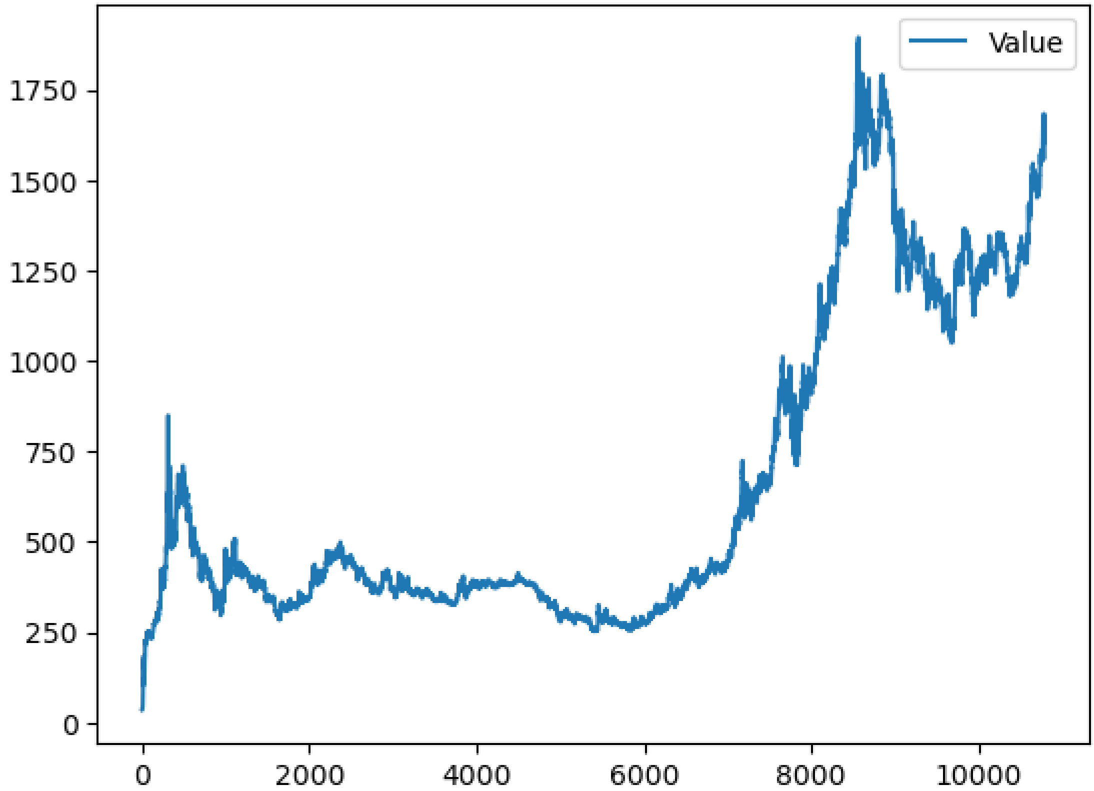
In [3]:
```
gold_df.shape
```

Out[3]:
```
(10787, 2)
```

In [4]:
```
gold_df.info()
gold_df.plot()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10787 entries, 0 to 10786
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    10787 non-null  object 
 1   Value   10787 non-null  float64
dtypes: float64(1), object(1)
memory usage: 168.7+ KB
```

Out[4]: <Axes: >



In [5]:
```
gold_df.describe()
```

Out[5]:

	Value
count	10787.000000
mean	653.596634
std	434.030848
min	35.100000
25%	349.200000
50%	409.350000
75%	1061.625000
max	1895.000000

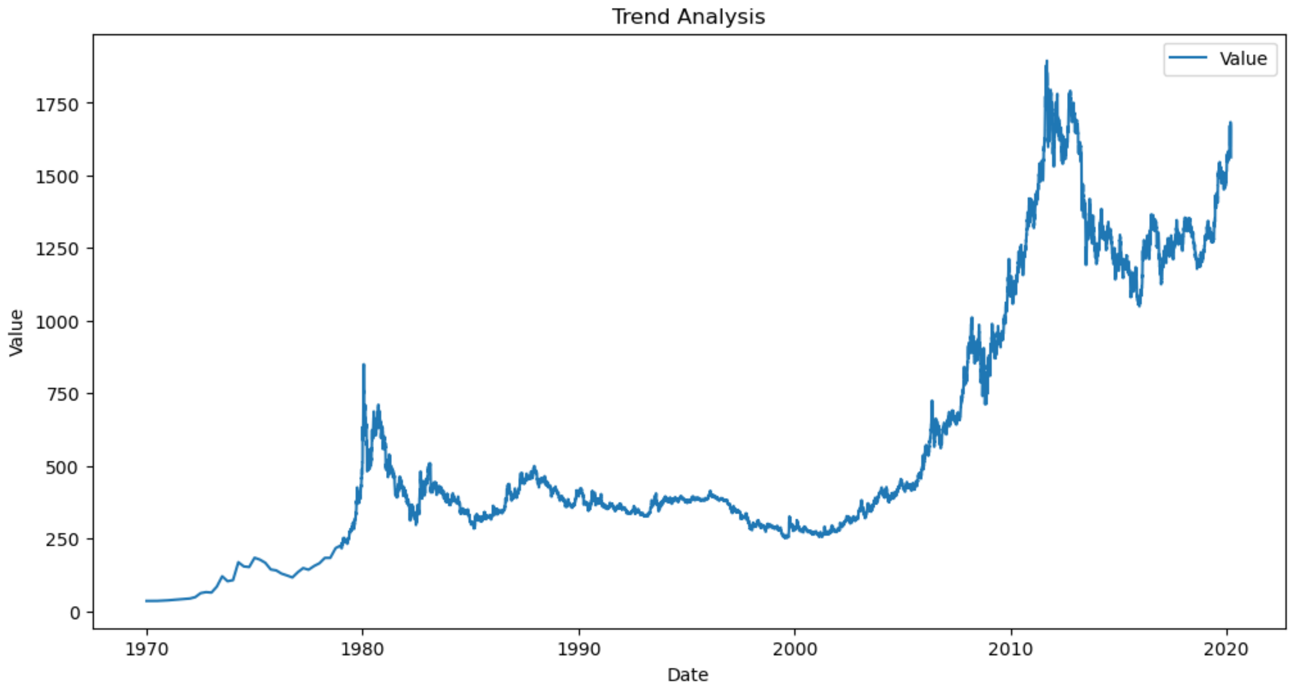
In [6]:
```
#converting data type object to date
#gold_df['Date'] = pd.to_datetime(gold_df['Date'])
#gold_df
```

In [7]:
```
gold_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10787 entries, 0 to 10786
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    10787 non-null  object 
 1   Value   10787 non-null  float64
dtypes: float64(1), object(1)
memory usage: 168.7+ KB
```

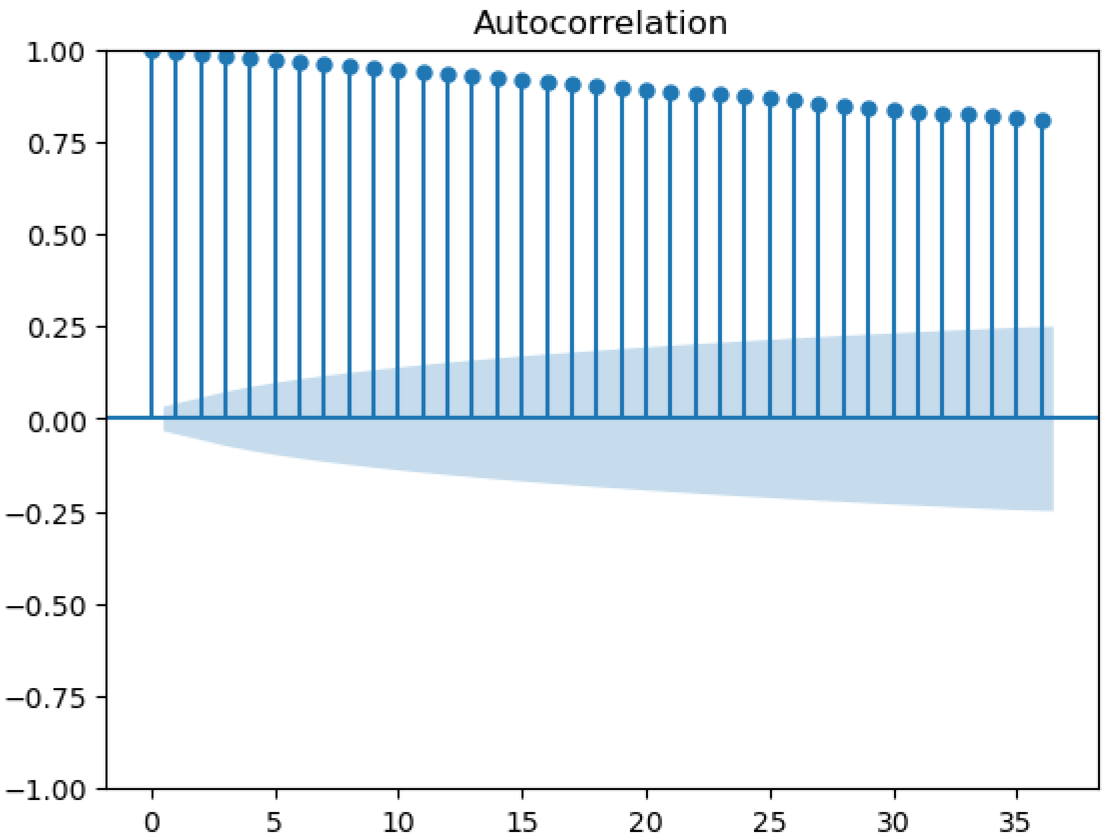
Task 2: Trend Analysis

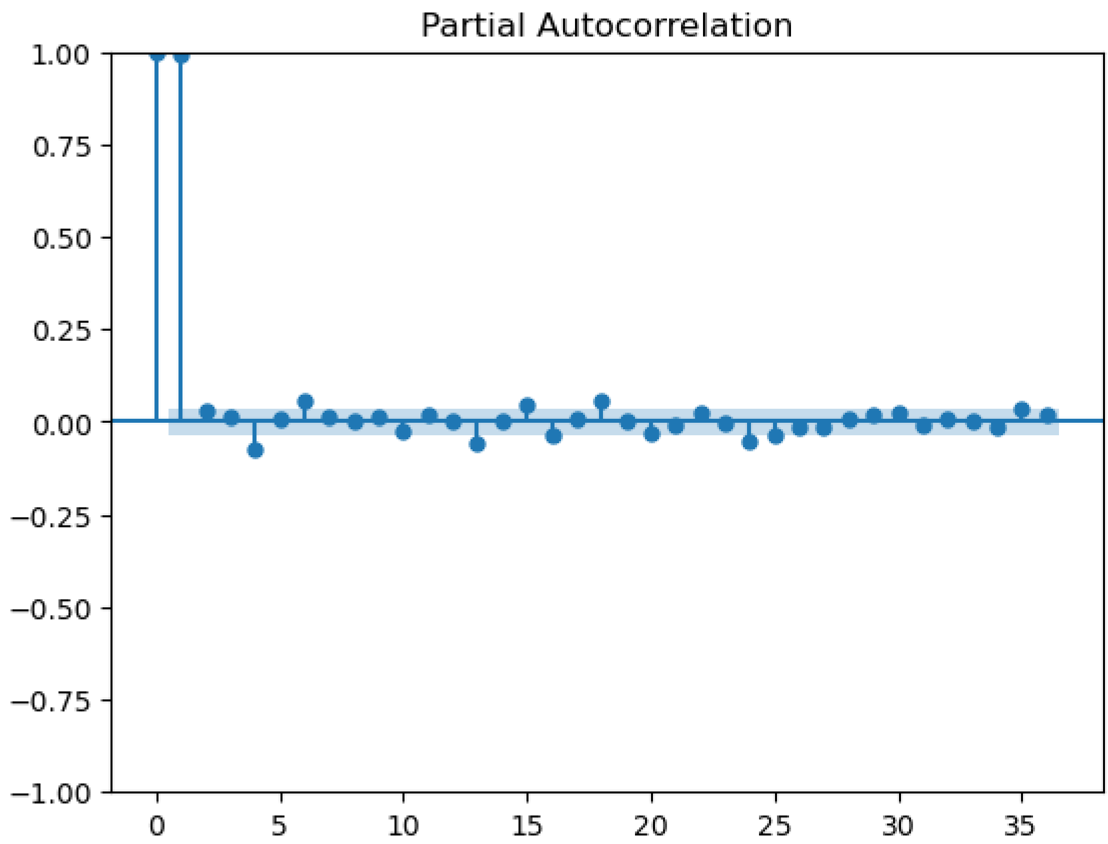
In [8]:
```
#Line plot
gold_df = gold_df.reset_index()

# Convert 'Date' column to datetime type
gold_df['Date'] = pd.to_datetime(gold_df['Date'])

# Set 'Date' as the index
gold_df.set_index('Date', inplace=True)

# Plot the time series data
plt.figure(figsize=(12, 6))
plt.plot(gold_df['Value'], label='Value')
plt.title('Trend Analysis')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.show()
```



Task 3: Seasonality Assessment

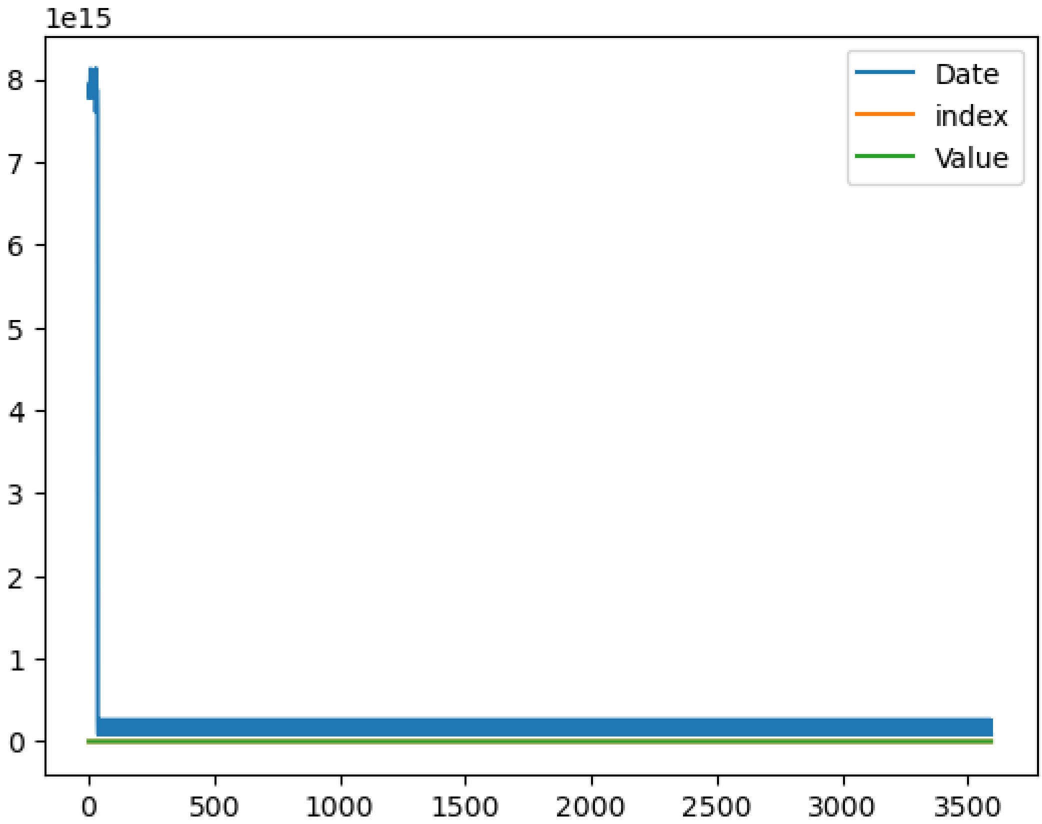
In [9]:
```
#Checking the stationarity of time series
#Method 1: ACF plot and PACF plot
#ACF(autocorrelation function) - is the correlation of the time series with its lags
#PACF - (partail auto correlation function)
total_rows = len(gold_df)
split_index = total_rows // 3
gold_df_train = gold_df.iloc[:split_index]
gold_df_test = gold_df.iloc[split_index:]
gold_df_train = gold_df_train.reset_index()

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
acf_original=plot_acf(gold_df_train['Value'].values)
pacf_original=plot_pacf(gold_df_train['Value'].values)
```





In [10]:
```
#ADF Test
gold_df_train_diff=gold_df_train.diff().dropna()
gold_df_train_diff.plot()
```

Out[10]: <Axes: >



In [11]:
```
from statsmodels. tsa.stattools import adfuller
adf_test=adfuller(gold_df_train['Value'])
print (f'p-value:{adf_test[1]}')
```

```
p-value:0.005754726484247235
```

In [12]:
```
# Fitting ARIMA model
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(gold_df_train['Value'], order=(2,1,0))
model_fit=model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Value   No. Observations:                 3595
Model:                 ARIMA(2, 1, 0)   Log Likelihood              -12367.668
Date:                Thu, 08 Feb 2024   AIC                          24741.336
Time:                        22:29:26   BIC                          24759.897
Sample:                             0   HQIC                         24747.951
                               - 3595
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0577      0.005    -10.962      0.000      -0.068      -0.047
ar.L2         -0.0284      0.004     -6.898      0.000      -0.036      -0.020
sigma2        57.0818      0.332    172.145      0.000      56.432      57.732
===================================================================================
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):            156812.80
Prob(Q):                              0.85   Prob(JB):                         0.00
Heteroskedasticity (H):               0.07   Skew:                             0.32
Prob(H) (two-sided):                  0.00   Kurtosis:                        35.35
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

In [13]:
```
import pmdarima as pm
auto_arima=pm.auto_arima(gold_df_train['Value'],stepwise=False, seasonal=False)
auto_arima
```

Out[13]:
```
ARIMA(0,1,5)(0,0,0)[0] intercept
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [14]:
```
auto_arima.summary()
```
Out[14]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	3595
Model:	SARIMAX(0, 1, 5)	**Log Likelihood**	-12326.625
Date:	Thu, 08 Feb 2024	**AIC**	24667.249
Time:	22:29:44	**BIC**	24710.559
Sample:	0	**HQIC**	24682.685
	- 3595		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	0.0845	0.124	0.681	0.496	-0.159	0.328
ma.L1	-0.0589	0.006	-9.578	0.000	-0.071	-0.047
ma.L2	-0.0078	0.005	-1.493	0.135	-0.018	0.002
ma.L3	0.1182	0.006	18.468	0.000	0.106	0.131
ma.L4	-0.0270	0.006	-4.640	0.000	-0.038	-0.016
ma.L5	-0.0920	0.007	-13.687	0.000	-0.105	-0.079
sigma2	55.7931	0.354	157.598	0.000	55.099	56.487

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	138105.86
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.08	**Skew:**	0.38
Prob(H) (two-sided):	0.00	**Kurtosis:**	33.36

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Task 4: Anomaly Detection

In [16]:
```
from scipy.stats import zscore
threshold = 3
gold_df['z_score'] = zscore(gold_df['Value'])
anomalies = gold_df[abs(gold_df['z_score']) > threshold]
anomalies
```
Out[16]:

	index	**Value**	**z_score**
Date			

In [17]:
```
gold_df
```

Out[17]:

	index	Value	z_score
Date			
1970-01-01	0	35.20	-1.424842
1970-04-01	1	35.10	-1.425072
1970-07-01	2	35.40	-1.424381
1970-10-01	3	36.20	-1.422538
1971-01-01	4	37.40	-1.419773
...	...	...	...
2020-03-09	10782	1672.50	2.347646
2020-03-10	10783	1655.70	2.308937
2020-03-11	10784	1653.75	2.304444
2020-03-12	10785	1570.70	2.113089
2020-03-13	10786	1562.80	2.094887

10787 rows × 3 columns

In [ ]:
```

```

Appendix D : Clustering

Task 1 Data Preparation

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
#a) Load dataset
mall_customers_df = pd.read_csv('mall_customers.csv')
mall_customers_df.head(500)
```

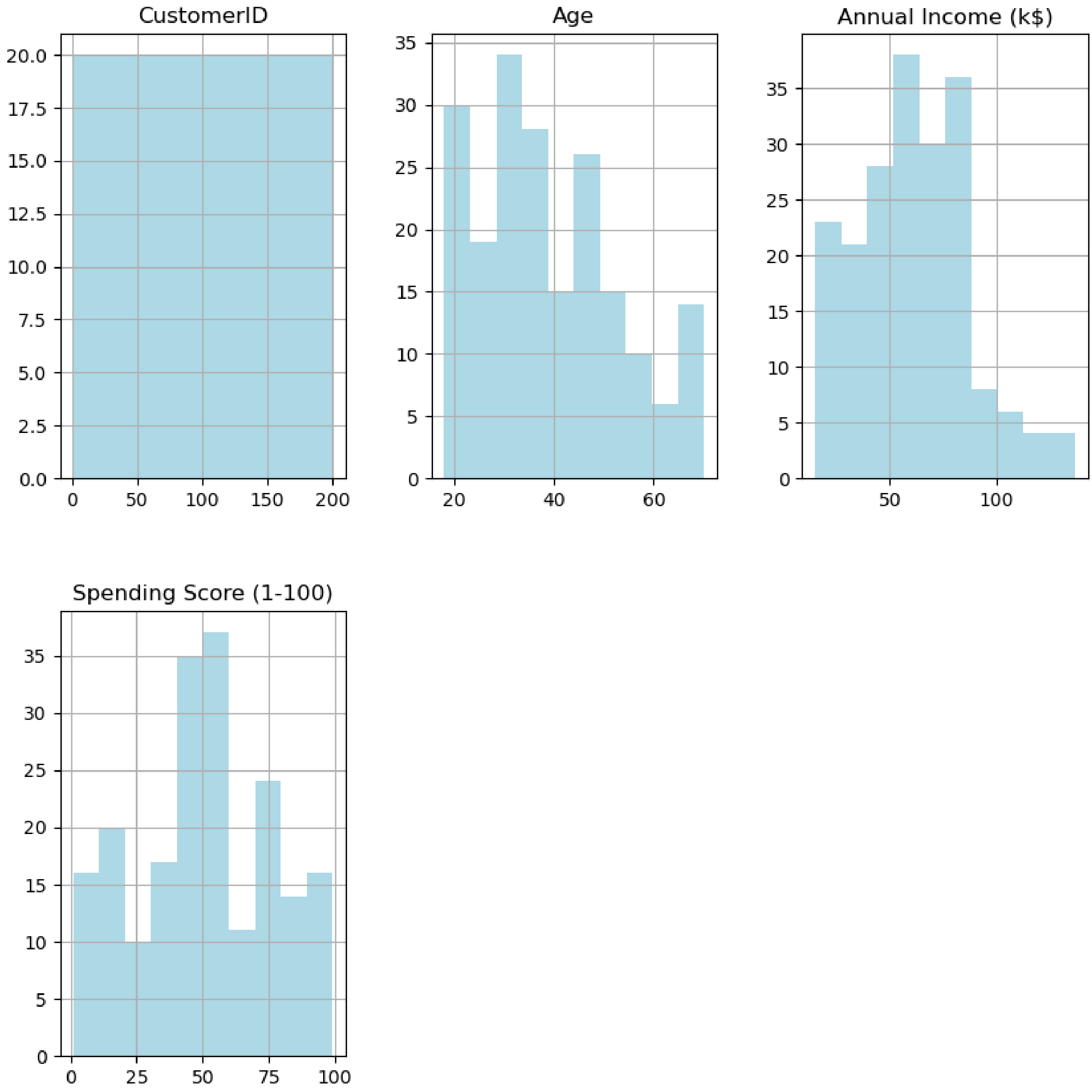
Out[2]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19.0	15.0	39
1	2	Male	21.0	15.0	81
2	3	Female	20.0	16.0	6
3	4	Female	23.0	16.0	77
4	5	Female	31.0	17.0	40
...	...	...	...	...	...
195	196	Female	35.0	120.0	79
196	197	Female	45.0	126.0	28
197	198	Male	32.0	126.0	74
198	199	Male	32.0	137.0	18
199	200	Male	30.0	137.0	83

200 rows × 5 columns

In [3]:
```python
mall_customers_df.hist(figsize=(10,10), color="lightblue", layout=(2,3))
```

Out[3]:
```
array([[<Axes: title={'center': 'CustomerID'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Annual Income (k$)'}>],
       [<Axes: title={'center': 'Spending Score (1-100)'}>, <Axes: >,
        <Axes: >]], dtype=object)
```



In [4]:
```python
mall_customers_df.shape
```

Out[4]:
```
(200, 5)
```

In [5]:
```
mall_customers_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   CustomerID              200 non-null    int64  
 1   Gender                  199 non-null    object 
 2   Age                     197 non-null    float64
 3   Annual Income (k$)      198 non-null    float64
 4   Spending Score (1-100)  200 non-null    int64  
dtypes: float64(2), int64(2), object(1)
memory usage: 7.9+ KB
```

In [6]:
```
mall_customers_df.describe()
```

Out[6]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	197.000000	198.000000	200.000000
mean	100.500000	38.944162	60.878788	50.200000
std	57.879185	14.026648	26.200427	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	29.000000	42.250000	34.750000
50%	100.500000	36.000000	62.000000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [7]:
```
mall_customers_df.isnull().sum()
```

Out[7]:
```
CustomerID                0
Gender                    1
Age                       3
Annual Income (k$)        2
Spending Score (1-100)    0
dtype: int64
```

In [8]:
```
mall_customers_df.duplicated().any()
```

Out[8]:
```
False
```

3 Data Cleaning and Preprocessing techniques

In [9]:
```
mall_customers_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   CustomerID              200 non-null    int64  
 1   Gender                  199 non-null    object 
 2   Age                     197 non-null    float64
 3   Annual Income (k$)      198 non-null    float64
 4   Spending Score (1-100)  200 non-null    int64  
dtypes: float64(2), int64(2), object(1)
memory usage: 7.9+ KB
```

In [10]:
```python
#a.1 Use SimpleImputer
# We replace the missing values with the mean value of the respective columns
# This only applies to numerical columns
feature_to_select = ['Age','Annual Income (k$)', 'Spending Score (1-100)']

# Create an instance of SimpleImputer
simp_imputer = SimpleImputer(missing_values=np.nan, strategy="mean")

# Fir the imputer on the data
simp_imputer.fit(mall_customers_df[feature_to_select])

X_imputed = simp_imputer.transform(mall_customers_df[feature_to_select])

mall_customers_df[feature_to_select] = X_imputed

mall_customers_df
```

Out[10]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19.0	15.0	39.0
1	2	Male	21.0	15.0	81.0
2	3	Female	20.0	16.0	6.0
3	4	Female	23.0	16.0	77.0
4	5	Female	31.0	17.0	40.0
...	...	...	...	...	...
195	196	Female	35.0	120.0	79.0
196	197	Female	45.0	126.0	28.0
197	198	Male	32.0	126.0	74.0
198	199	Male	32.0	137.0	18.0
199	200	Male	30.0	137.0	83.0

200 rows × 5 columns

In [11]:
```python
#a.2 using dropna for missing categorical values like "Gender"
mall_customers_df = mall_customers_df.dropna(subset=['Gender'], axis=0)
mall_customers_df
```

Out[11]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19.0	15.0	39.0
1	2	Male	21.0	15.0	81.0
2	3	Female	20.0	16.0	6.0
3	4	Female	23.0	16.0	77.0
4	5	Female	31.0	17.0	40.0
...	...	...	...	...	...
195	196	Female	35.0	120.0	79.0
196	197	Female	45.0	126.0	28.0
197	198	Male	32.0	126.0	74.0
198	199	Male	32.0	137.0	18.0
199	200	Male	30.0	137.0	83.0

199 rows × 5 columns

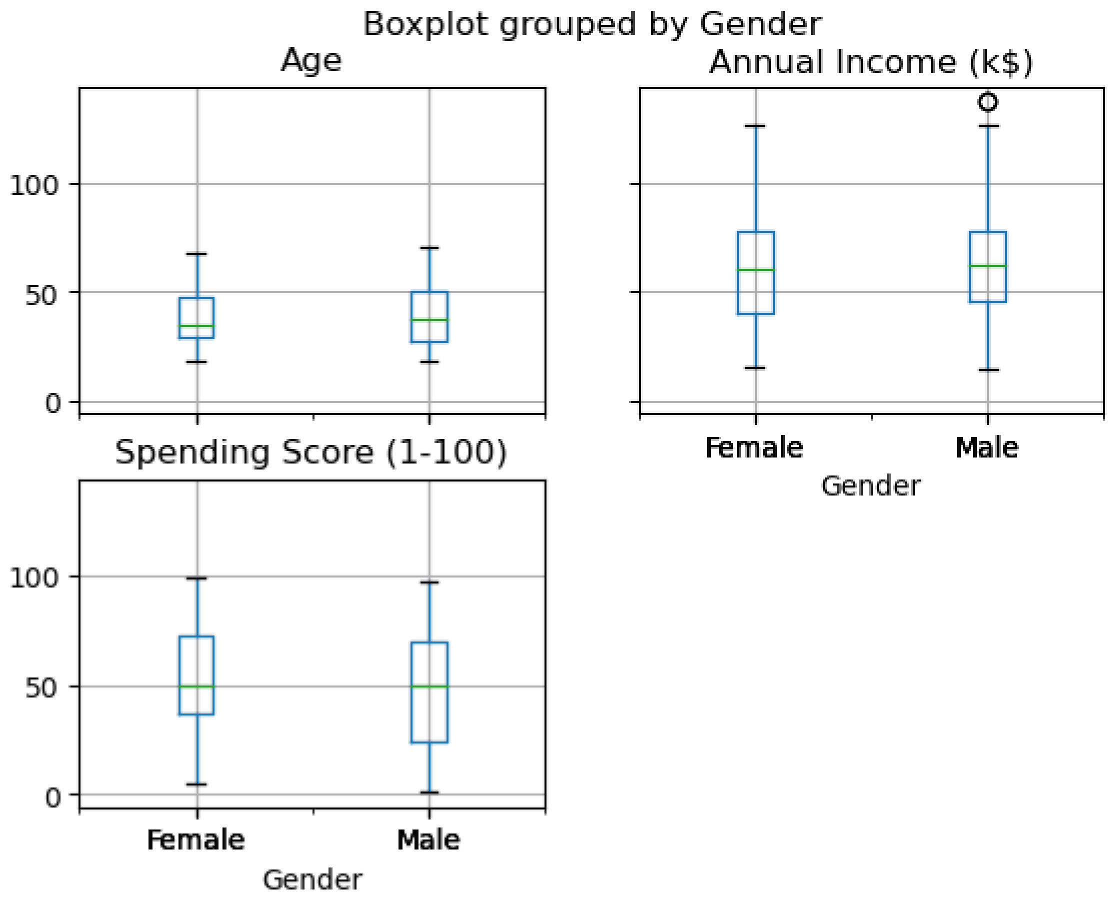
In [12]:
```python
#dropping the customer ID
mall_customers_df = mall_customers_df.drop('CustomerID', axis=1)
mall_customers_df
```

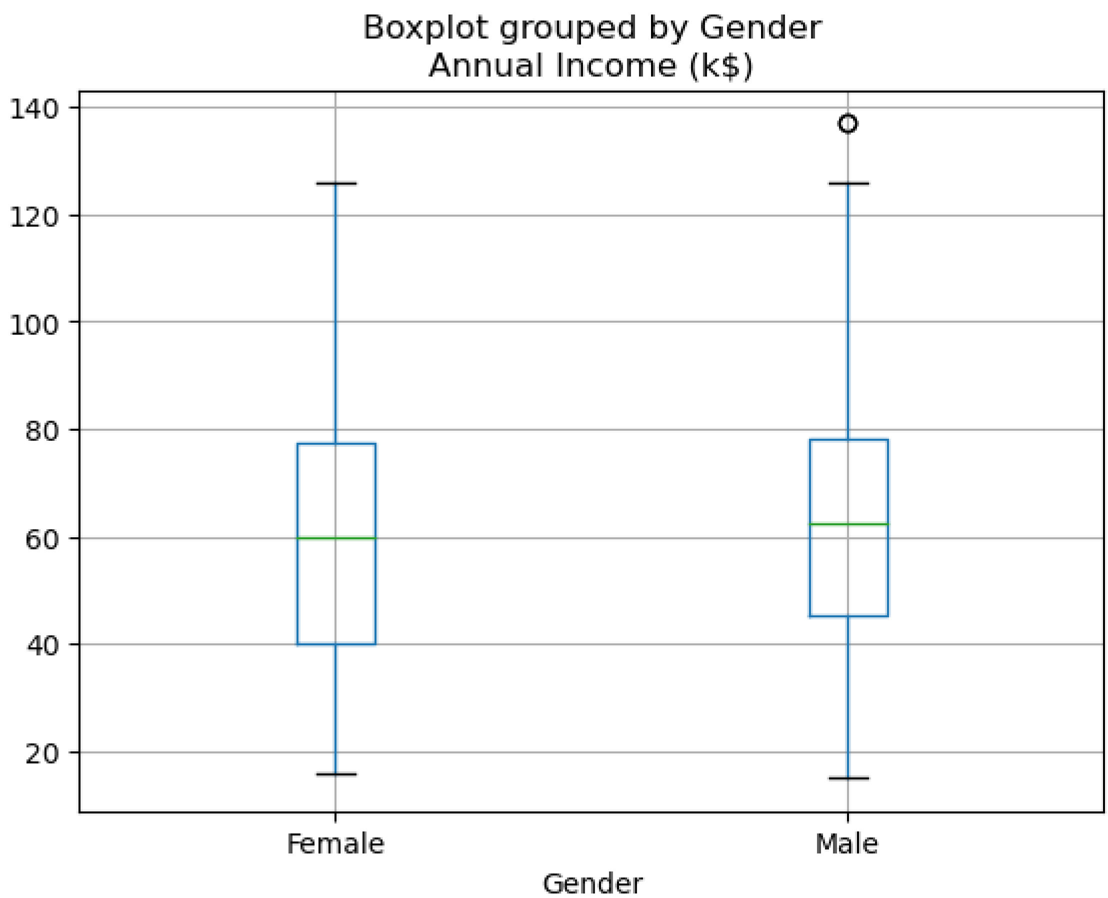
Out[12]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19.0	15.0	39.0
1	Male	21.0	15.0	81.0
2	Female	20.0	16.0	6.0
3	Female	23.0	16.0	77.0
4	Female	31.0	17.0	40.0
...	...	...	...	...
195	Female	35.0	120.0	79.0
196	Female	45.0	126.0	28.0
197	Male	32.0	126.0	74.0
198	Male	32.0	137.0	18.0
199	Male	30.0	137.0	83.0

199 rows × 4 columns

In [13]:
```
#a.3 visual dataset using boxplot to check for outliers
mall_customers_df.boxplot(column=['Age', 'Annual Income (k$)', 'Spending Score (1-100
mall_customers_df.boxplot(column=['Annual Income (k$)'], by='Gender')
plt.show()
```





In [14]:
```
#a.3 Handing Outliers
# Detect outliers using Using Z Score
# Based on the Boxplot, Annual Income yields outliers. We can remove these entries
from scipy.stats import zscore
annual_df = mall_customers_df.copy()
annual_df['Annual_Z_Score'] = zscore(annual_df['Annual Income (k$)'])
threshold = 2
zscore_annual_outliers = ((annual_df['Annual_Z_Score'] < (-1*threshold)) | (annual_df
print(f'Annual Outliers using Z Score: {zscore_annual_outliers.sum()}')
mall_customers_df = mall_customers_df[~zscore_annual_outliers]
mall_customers_df
```

```
Annual Outliers using Z Score: 8
```

Out[14]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19.0	15.0	39.0
1	Male	21.0	15.0	81.0
2	Female	20.0	16.0	6.0
3	Female	23.0	16.0	77.0
4	Female	31.0	17.0	40.0
...	...	...	...	...
187	Male	28.0	101.0	68.0
188	Female	41.0	103.0	17.0
189	Female	36.0	103.0	85.0
190	Female	34.0	103.0	23.0
191	Female	32.0	103.0	69.0

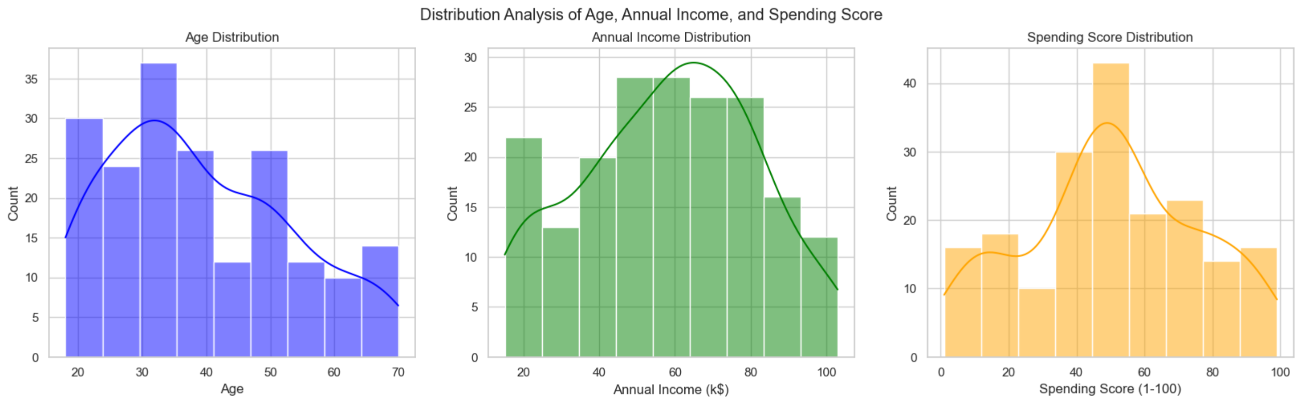
191 rows × 4 columns

In [15]:
```
#b.1 Using Correlation Analysis. We want to understand the relationship Age and Annua
correlation_matrix = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score
print(correlation_matrix)
```

```
                             Age  Annual Income (k$)  Spending Score (1-100)
Age                     1.000000            0.007192               -0.325696
Annual Income (k$)      0.007192            1.000000               -0.014391
Spending Score (1-100) -0.325696           -0.014391                1.000000
```

In [16]:
```
#b.2 Histogram
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style of the seaborn plot
sns.set(style='whitegrid')

# Create a figure and axis objects
fig, axs = plt.subplots(1, 3, figsize=(20, 5))

# Plot the distribution of age, annual income, and spending score
sns.histplot(data=mall_customers_df, x='Age', kde=True, color='blue', ax=axs[0])
sns.histplot(data=mall_customers_df, x='Annual Income (k$)', kde=True, color='green',
sns.histplot(data=mall_customers_df, x='Spending Score (1-100)', kde=True, color='ora

# Set the titles of the plots
axs[0].set_title('Age Distribution')
axs[1].set_title('Annual Income Distribution')
axs[2].set_title('Spending Score Distribution')

# Set the title for the entire plot
fig.suptitle('Distribution Analysis of Age, Annual Income, and Spending Score')

# Display the plots
```

Out[16]:
```
Text(0.5, 0.98, 'Distribution Analysis of Age, Annual Income, and Spending Score')
```



In [17]:
```
#b.3 using Descriptive Statistics and Summary Metrics
spending_stat = mall_customers_df.describe()
print('Summary Statistics')
spending_stat
```

```
Summary Statistics
```

Out[17]:

	Age	Annual Income (k$)	Spending Score (1-100)
count	191.000000	191.000000	191.000000
mean	39.046536	58.401872	50.324607
std	14.184980	23.004659	25.517370
min	18.000000	15.000000	1.000000
25%	28.000000	42.000000	35.000000
50%	36.000000	60.878788	50.000000
75%	49.000000	76.500000	72.000000
max	70.000000	103.000000	99.000000

In [18]:

```
spending_range = mall_customers_df['Spending Score (1-100)'].max() - mall_customers_d
spending_std = mall_customers_df['Spending Score (1-100)'].std()
print(f'Spending Score Range: {spending_range}')
print(f'Spending Score Standard Deviation: {spending_std}')
```

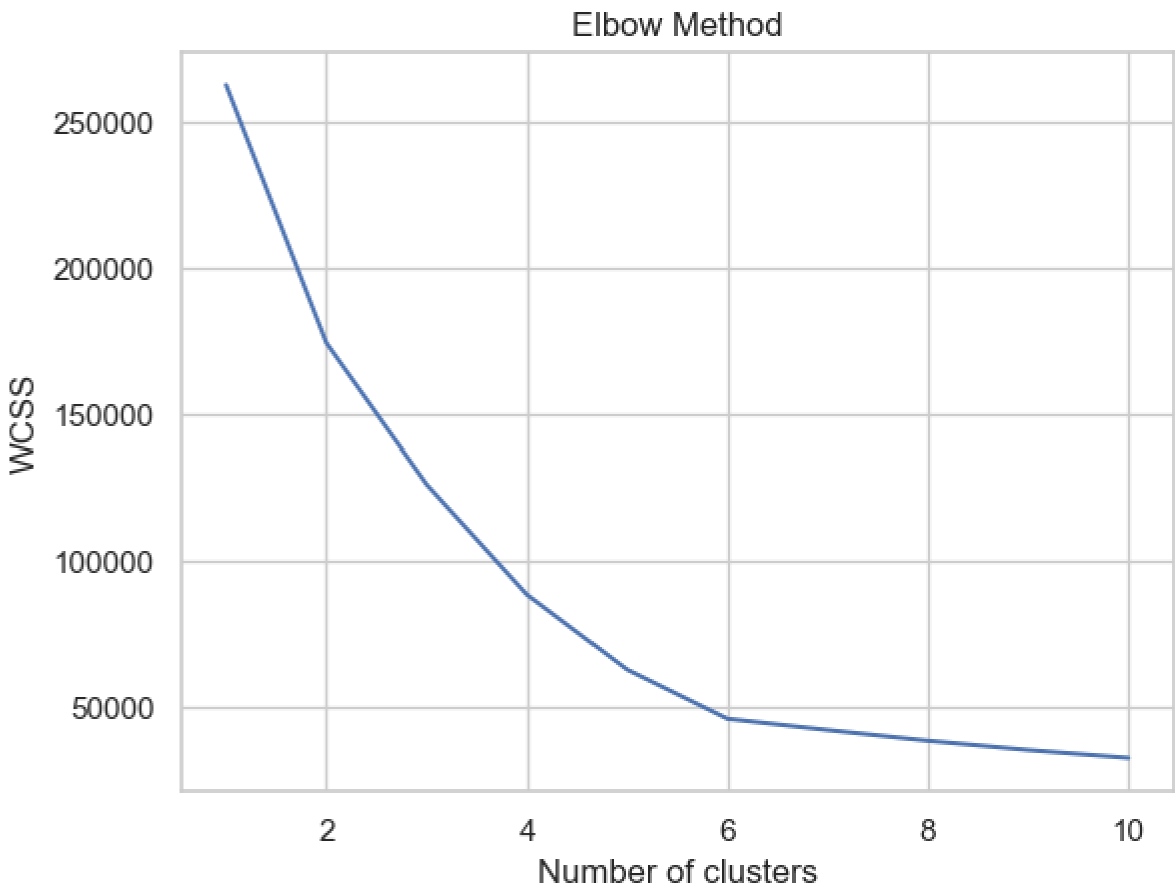
```
Spending Score Range: 98.0
Spending Score Standard Deviation: 25.517370177313552
```

Task 2 : Unsupervised Algorithm Implementation

In [33]:
```python
#1) # using K-Means
# First we identify the number of clusters using the elbow method

from sklearn.cluster import KMeans

# Select the features to use for clustering
features = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

# Determine the optimal number of clusters using the elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(features)
    wcss.append(kmeans.inertia_)

# Plot the WCSS values
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
```

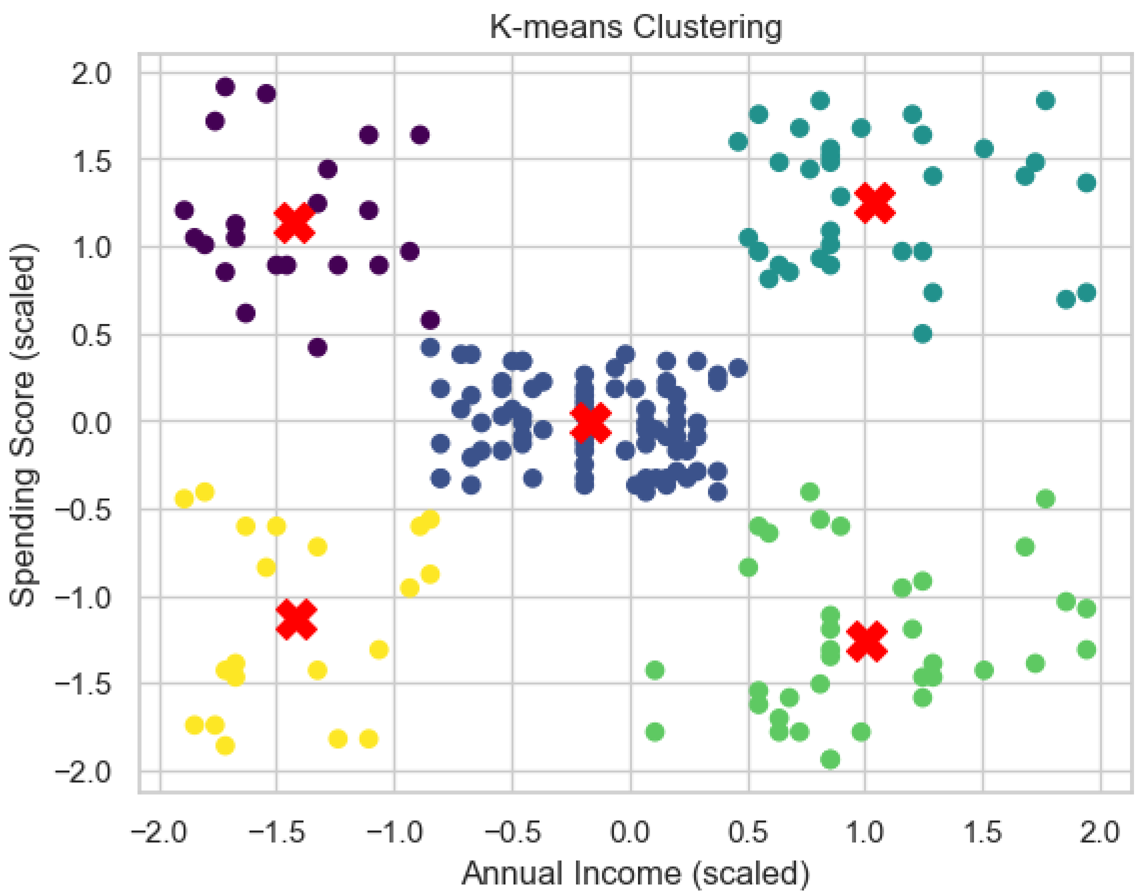
```
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```



In [38]:

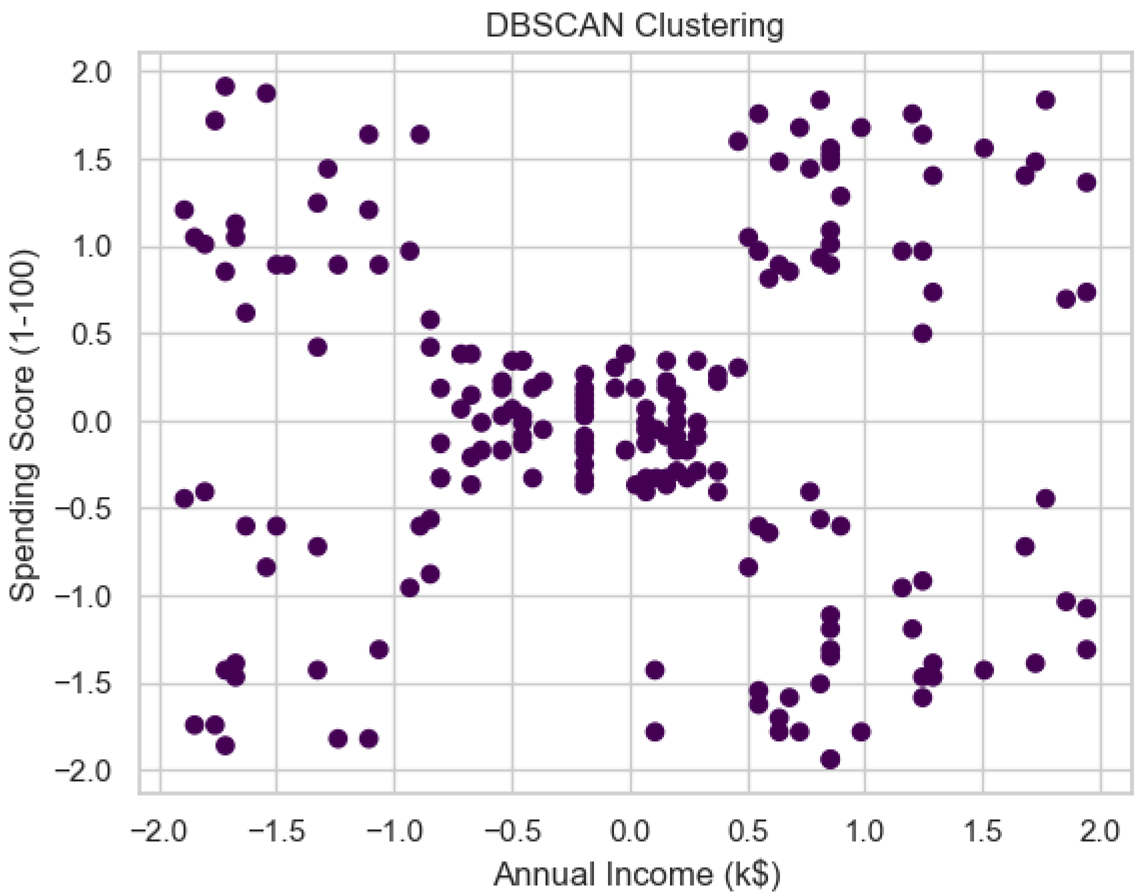
```
#1) # using K-Means
from sklearn.cluster import KMeans

features = mall_customers_df[['Annual Income (k$)', 'Spending Score (1-100)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
# Choose the number of clusters (K)
k = 5

# Initialize KMeans
kmeans = KMeans(n_clusters=k, random_state=42)

# Fit the model to the data
kmeans.fit(scaled_features)

# Get cluster assignments for each data point
cluster_labels = kmeans.labels_

plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=cluster_labels, cmap='vir
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='X',
plt.xlabel('Annual Income (scaled)')
plt.ylabel('Spending Score (scaled)')
plt.title('K-means Clustering')
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```



In [42]:

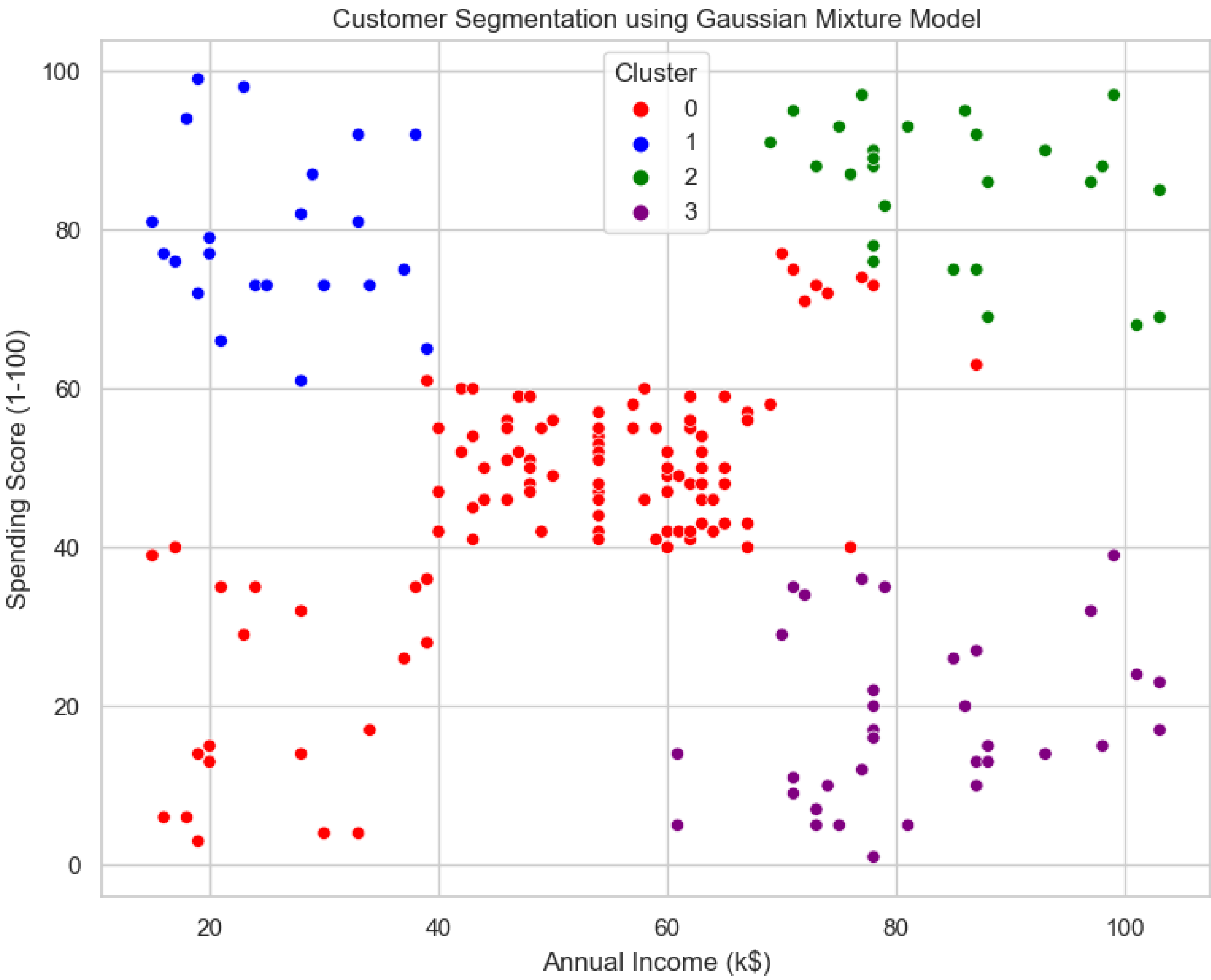
```
#2) Using Density Based Spatial Clustering of Applications with Noise (DBSCAN)
# Popular choice for clustering of datasets
# Unlike K Means, DBSCAN automatically determines the number of clusters
# DBSCAN can automatically identifies outliers and exclude them from the clustering
from sklearn.cluster import DBSCAN

# Assume you're interested in two features: 'Annual Income' and 'Spending Score'
# Both are crucial features for understanding the customer buying behavior
# Both are Linear such as Customers with higher-income may spend more.
X = mall_customers_df[["Annual Income (k$)", "Spending Score (1-100)"]]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

# Initialize DBSCAN
# eps=5 maximum distance between data points that may considered them within the same
dbscan = DBSCAN(eps=5)  # Adjust parameters as needed

# Fit the model
dbscan.fit(scaled_features)

# Get cluster labels (-1 indicates noise/outliers)
cluster_labels = dbscan.labels_

# Visualize the clusters
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=cluster_labels, cmap="vir
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("DBSCAN Clustering")
plt.show()


# Number of clusters (excluding noise points)
num_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
print(f"Estimated number of clusters: {num_clusters}")

# Identify noise points (outliers)
num_noise = list(cluster_labels).count(-1)
print(f"Estimated number of noise points: {num_noise}")
```




```
Estimated number of clusters: 1
Estimated number of noise points: 0
```

In [21]:
```
#3) Using Gaussian Mixture Model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import seaborn as sns

# Assume you're interested in two features: 'Annual Income' and 'Spending Score'
X = mall_customers_df[["Annual Income (k$)", "Spending Score (1-100)"]]

# Initialize Gaussian Mixture Model
gmm = GaussianMixture(n_components=4, random_state=2021)  # Specify the number of clu

# Fit the model
gmm.fit(X)

# Predict cluster labels
cluster_labels = gmm.predict(X)

# Add cluster labels to the original dataframe
mall_customers_df["Cluster"] = cluster_labels

# Visualize the clusters
plt.figure(figsize=(9, 7))
sns.scatterplot(data=mall_customers_df, x="Annual Income (k$)", y="Spending Score (1-
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("Customer Segmentation using Gaussian Mixture Model")
plt.savefig("Customer_Segmentation_GMM_Python.png", format="png", dpi=150)
plt.show()
```

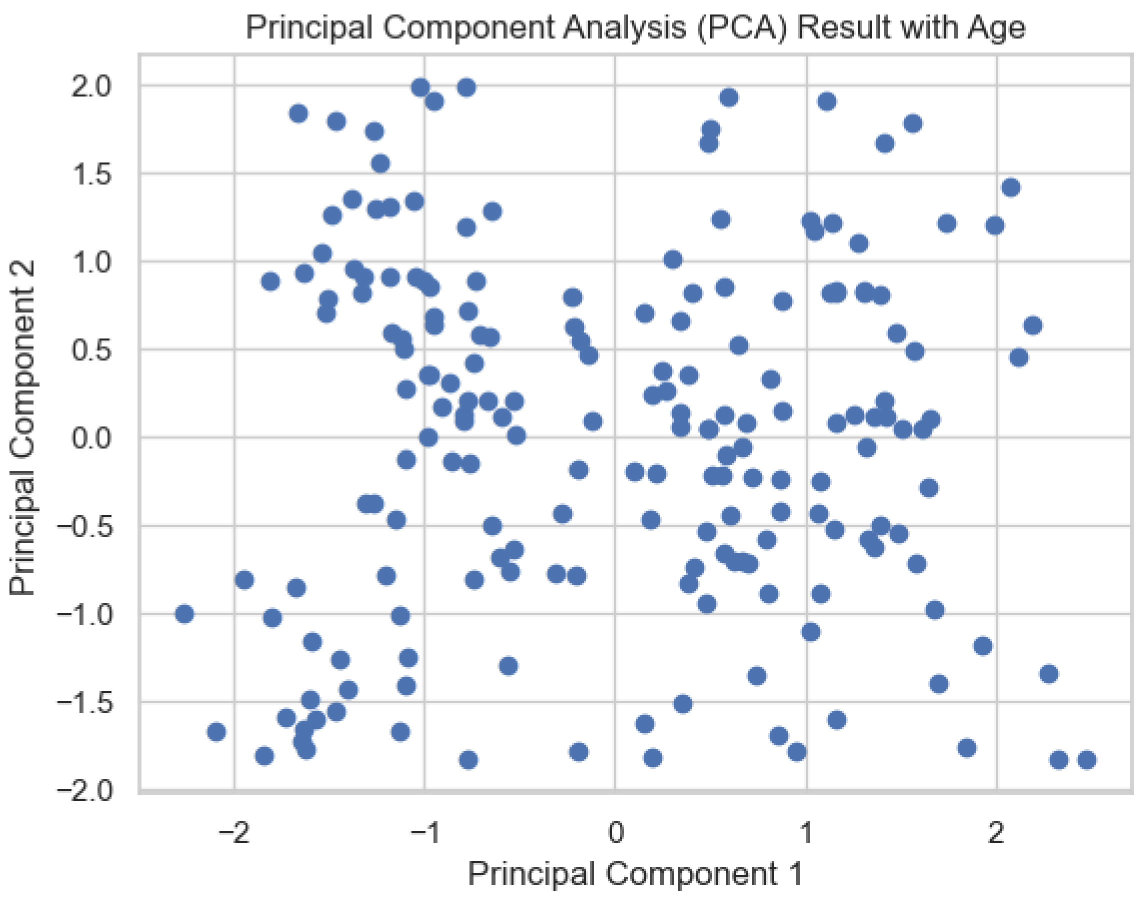
```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\AppData\Local\Temp\ipykernel_19820\4004794799.py:21: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  mall_customers_df["Cluster"] = cluster_labels
```



B) Apply Principal Component Analysis (PCA) to reduce the dimensionality of a given dataset. Describe the steps involved in PCA

In [22]:
```
# Perform PCA for Age, Annual Income, and Spending Score feature only.
# PCA is a technique for reducing the dimensionality of the dataset while ensuring the
# PCA will create a new set of features that will capture most of the important data p
X = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values

# Normalize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)
```

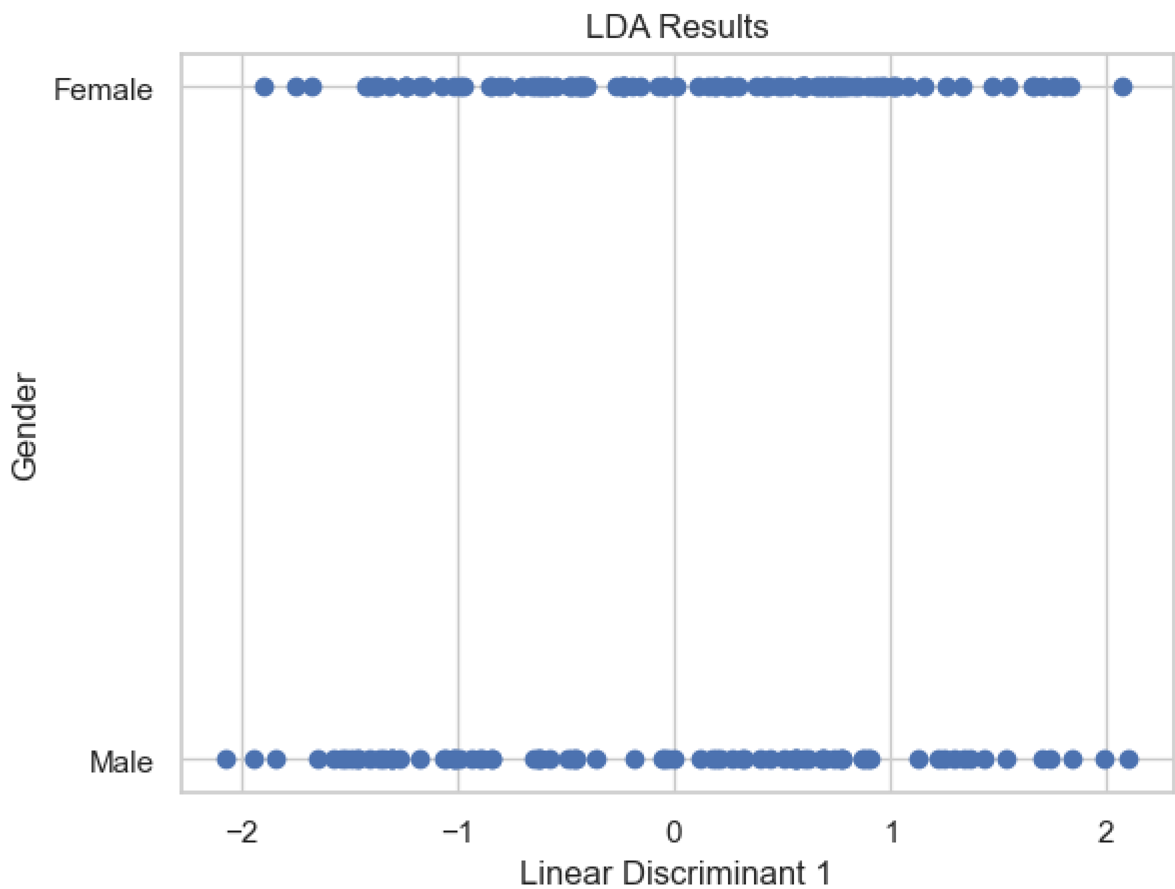
In [23]:
```
from sklearn.decomposition import PCA

# Initialize PCA up to 2 components
# This will create 2 columns/components
pca = PCA(n_components=2)

# Fit and transform the scaled features
pca_result = pca.fit_transform(scaled_features)

# Create a DataFrame with the PCA results
plt.scatter(pca_result[:,0], pca_result[:,1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Principal Component Analysis (PCA) Result with Age')
plt.show()
```



C) Linear Discriminant Analysis (LDA) to perform dimensionality reduction. Describe the steps involved in LDA

In [24]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Select features for LDA (including Age, Annual Income, and Spending Score)
X = mall_customers_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values

# Use Gender as the target feature. LDA works better with Categorical variables
# Replacing 1 and 0 to Categorical value which is Male and Female.
y = mall_customers_df['Gender'].values

# Standardize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

# Apply LDA with n_components=1
lda = LinearDiscriminantAnalysis(n_components=1)

# lda_result is the reduced-dimensional space
lda_result = lda.fit_transform(scaled_features, y)

lda_coefficients = lda.coef_
print("LDA Coefficients:", lda_coefficients)

# Create a DataFrame with the LDA results
#lda_df = pd.DataFrame(lda_result, columns=['Gender'])

# Visualize LDA results
plt.scatter(lda_result[:,0], y)
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Gender')
plt.title('LDA Results')
plt.show()
```

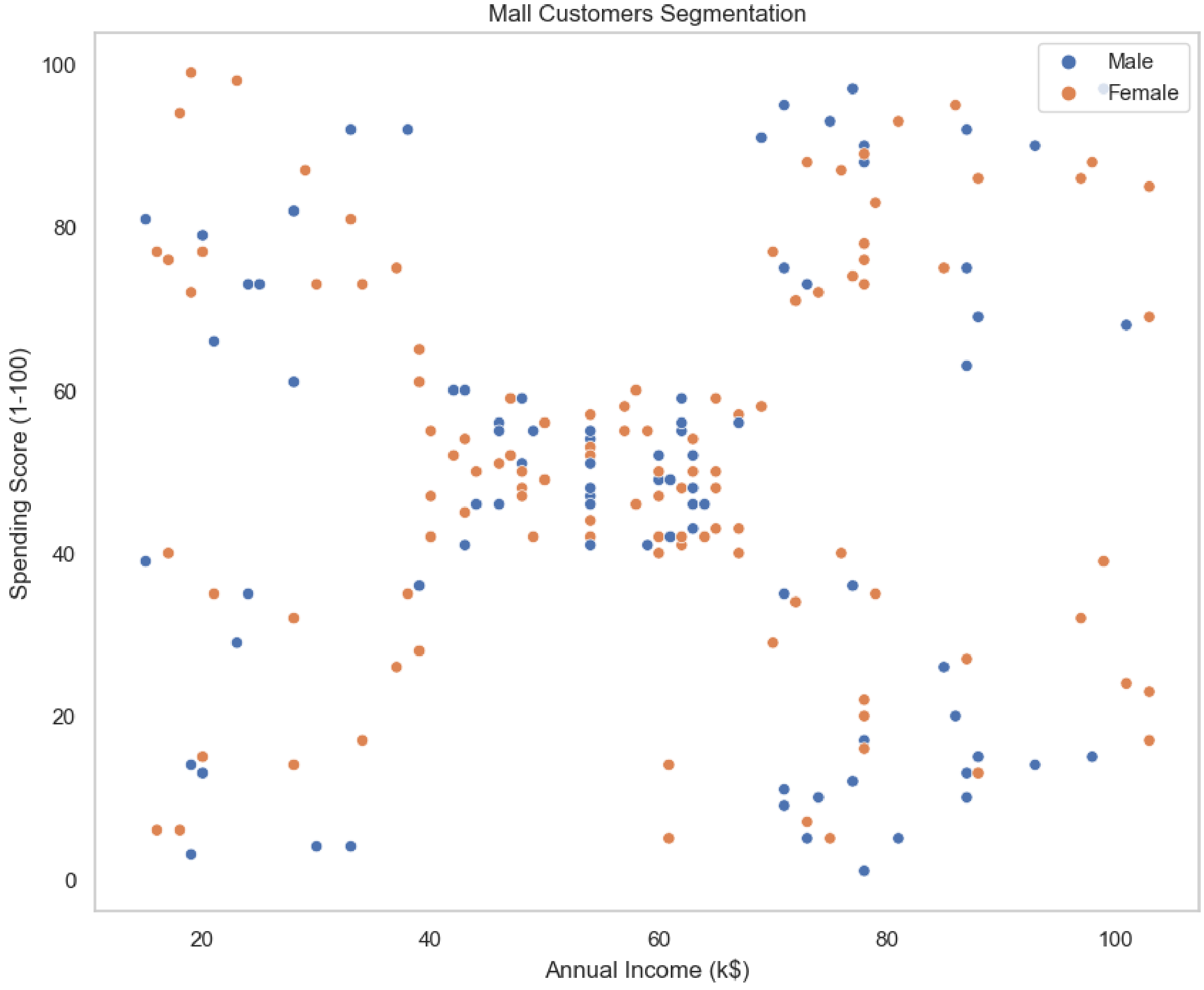
```
LDA Coefficients: [[ 0.12459596  0.05086515 -0.07629103]]
```



D) Ways to visualize segmented data

In [25]:

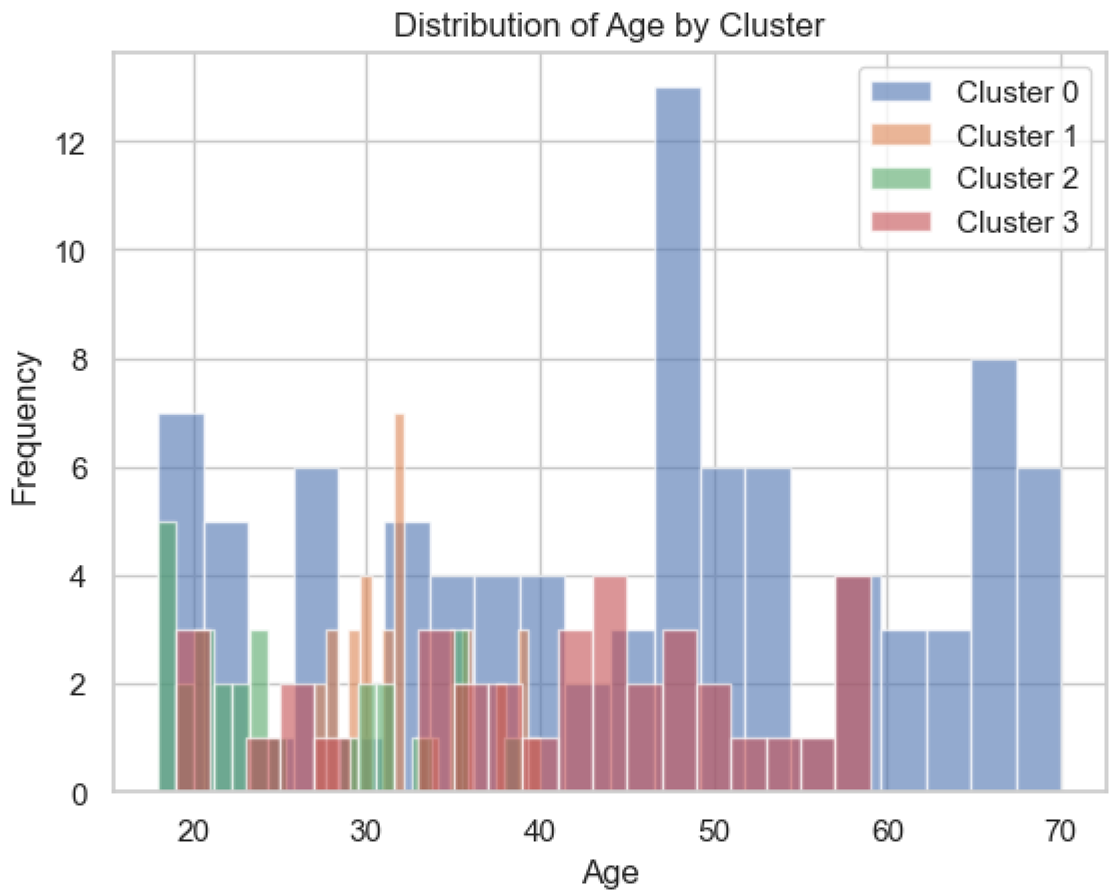
```python
# Using Cluster Plot
x = "Annual Income (k$)"
y = "Spending Score (1-100)"
hue = "Gender"

mall_customer_gender_df = mall_customers_df.copy()

# Create the scatter plot
plt.figure(figsize=(10, 8))

# The mall_customer_gender_df is a segmented data based on the Gender (Male/Female) vo
sns.scatterplot(data=mall_customer_gender_df, x=x, y=y, hue=hue)
plt.xlabel(x)
plt.ylabel(y)
plt.title("Mall Customers Segmentation")
plt.legend(loc="upper right")
plt.grid()
plt.show()
```



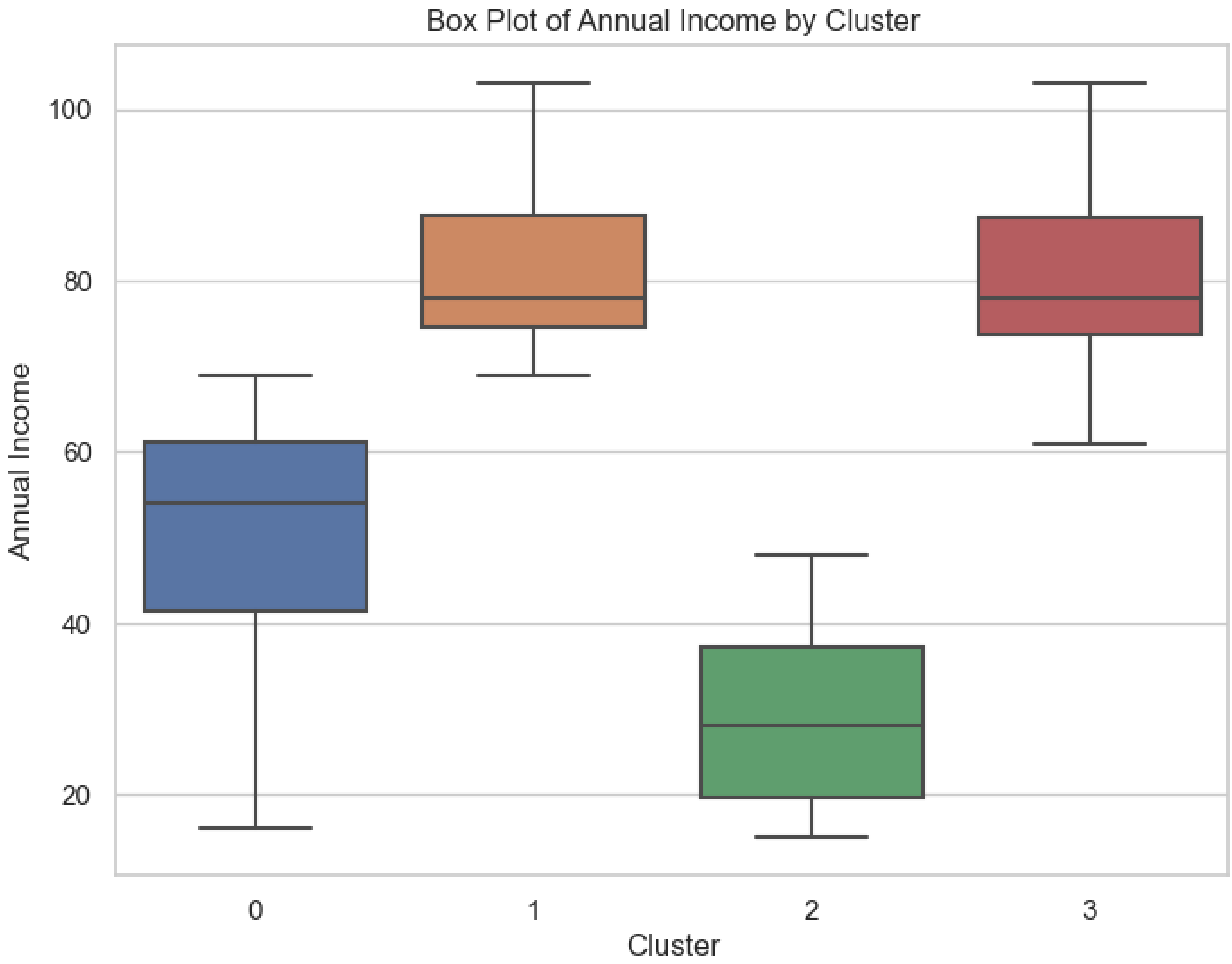
In [26]:

```
# Using Histogram plot
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Select relevant features
X = mall_customers_df[["Age", "Annual Income (k$)", "Spending Score (1-100)"]]

k = 4
# Fit K-means with the chosen K (e.g., K=5)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

# Add cluster labels to the original data
mall_customers_df["Cluster"] = kmeans.labels_

# Create historgram based on the number of clusters k
for cluster_id in range(k):
    plt.hist(mall_customers_df[mall_customers_df["Cluster"] == cluster_id]["Age"], bi
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Distribution of Age by Cluster")
plt.legend()
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\AppData\Local\Temp\ipykernel_19820\3156836239.py:16: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  mall_customers_df["Cluster"] = kmeans.labels_
```


Distribution of Age by Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Frequency
0
2
4
6
8
10
12
20
30
40
50
60
70
Age

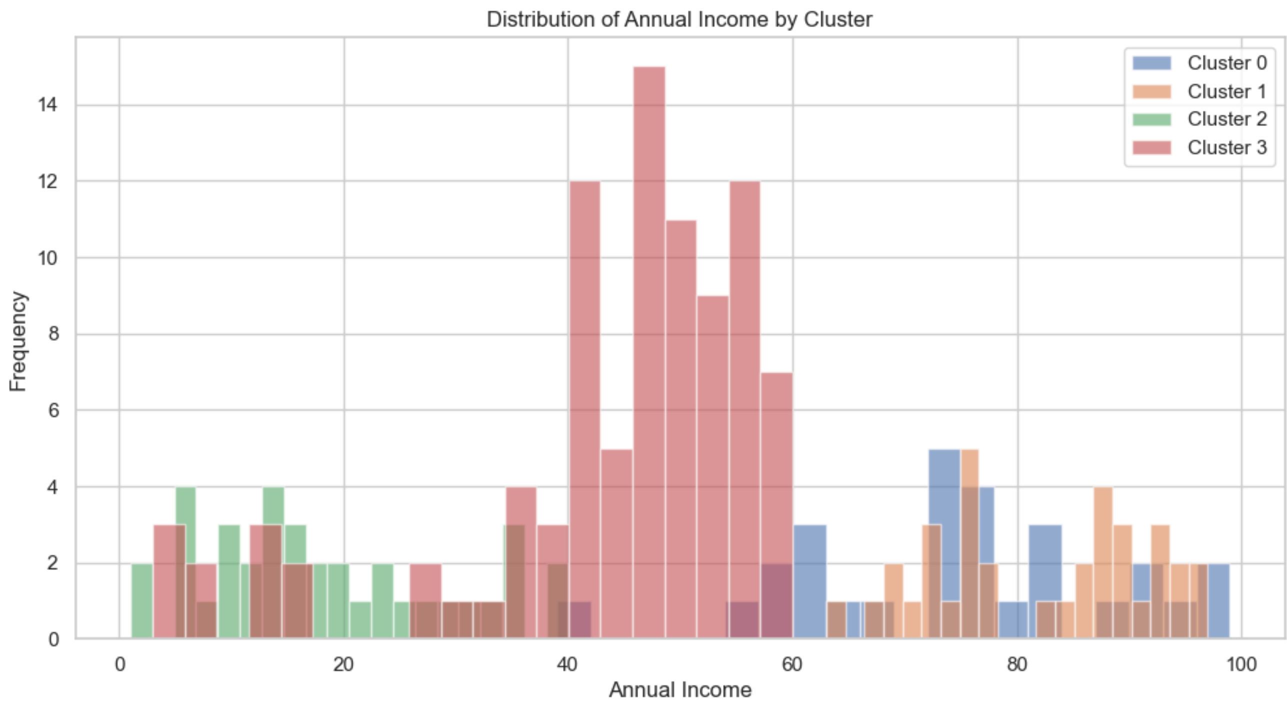
In [27]:
```
# Using Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x="Cluster", y="Annual Income (k$)", data=mall_customers_df)
plt.xlabel("Cluster")
plt.ylabel("Annual Income")
plt.title("Box Plot of Annual Income by Cluster")
plt.show()
```



In [28]:

```
# Choose the optimal K based on the plot (e.g., K=4)
optimal_k = 4

# Fit K-means with the chosen K
kmeans_final = KMeans(n_clusters=optimal_k, init="k-means++")
kmeans_final.fit(X)

# Add cluster labels to the original data
mall_customers_df["Cluster"] = kmeans_final.labels_

# Create histograms for each cluster based on "Annual Income"
plt.figure(figsize=(12, 6))
for cluster_id in range(optimal_k):
    plt.hist(mall_customers_df[mall_customers_df["Cluster"] == cluster_id]["Spending S

plt.xlabel("Annual Income")
plt.ylabel("Frequency")
plt.title("Distribution of Annual Income by Cluster")
plt.legend()
plt.show()
```

```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWa
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\torri\AppData\Local\Temp\ipykernel_19820\2050094159.py:9: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  mall_customers_df["Cluster"] = kmeans_final.labels_
```



Appendix E: Bank Customer Churn Dataset

customer_	credit_sco	country	gender	age	tenure	balance	products_r	credit_card	active_me	estimated_
15634602	619	France	Female	42	2	0	1	1	1	101348.9
15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.6
15619304	502	France	Female	42	8	159660.8	3	1	0	113931.6
15701354	699	France	Female	39	1	0	2	0	0	93826.63
15737888	850	Spain	Female	43	2	125510.8	1	1	1	79084.1
15574012	645	Spain	Male	44	8	113755.8	2	1	0	149756.7
15592531	822	France	Male	50	7	0	2	1	1	10062.8
15656148	376	Germany	Female	29	4	115046.7	4	1	0	119346.9
15792365	501	France	Male	44	4	142051.1	2	0	1	74940.5
15592389	684	France	Male	27	2	134603.9	1	1	1	71725.73
15767821	528	France	Male	31	6	102016.7	2	0	0	80181.12
15737173	497	Spain	Male	24	3	0	2	1	0	76390.01
15632264	476	France	Female	34	10	0	2	1	0	
15691483	549	France	Female	25	5	0	2	0	0	190857.8
15600882	635	Spain	Female	35	7	0	2	1	1	65951.65
15643966	616	Germany	Male	45	3	143129.4	2	0	1	64327.26
15737452	653	Germany	Male	58	1	132602.9	1	1	0	5097.67
15788218	549	Spain	Female	24	9	0	2	1	1	14406.41
15661507	587	Spain	Male	45	6	0	1	0	0	158684.8
15568982	726	France	Female	24	6	0	2	1	1	54724.03
15577657	732	France	Male	41	8	0	2	1	1	170886.2
15597945	636	Spain	Female	32	8	0	2	1	0	138555.5
15699309	510	Spain	Female	38	4	0	1	1	0	118913.5
15725737	669	France	Male	46	3	0	2	0	1	8487.75
15625047	846	France	Female	38	5	0	1	1	1	187616.2
15738191	577	France	Male	25	3	0	2	0	1	124508.3
15736816	756	Germany	Male	36	2	136815.6	1	1	1	170042
15700772	571	France	Male	44	9	0	2	0	0	38433.35
15728693	574	Germany	Female	43	3	141349.4	1	1	1	100187.4
15656300	411	France	Male	29	0	59697.17	2	1	1	53483.21
15589475	591	Spain	Female	39	3	0	3	1	0	140469.4
15706552	533	France	Male	36	7	85311.7	1	0	1	156731.9
15750181	553	Germany	Male	41	9	110112.5		0	0	81898.81
15659428	520	Spain	Female	42	6	0	2	1	1	34410.55
15732963	722	Spain	Female	29	9	0	2	1	1	142033.1
15794171	475	France	Female	45	0	134264	1	1	0	27822.99
15788448	490	Spain	Male	31	3	145260.2	1	0	1	114066.8
15729599	804	Spain	Male	33	7	76548.6	1	0	1	98453.45

Appendix F: Advertising Dataset

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5
13.2	15.9	49.6	5.6
228.3	16.9	26.2	20.5
62.3	12.6	18.3	9.7
262.9	3.5	19.5	17
142.9	29.3	12.6	15
240.1	16.7	22.9	20.9
248.8	27.1	22.9	18.9
70.6	16	40.8	10.5
292.9	28.3	43.2	21.4
112.9	17.4	38.6	11.9
97.2	1.5	30	13.2
265.6	20	0.3	17.4
95.7	1.4	7.4	11.9
290.7	4.1	8.5	17.8
266.9	43.8	5	25.4
74.7	49.4	45.7	14.7

Appendix G: Gold Price Dataset

Date	Value
1/01/1970	35.2
1/04/1970	35.1
1/07/1970	35.4
1/10/1970	36.2
1/01/1971	37.4
1/04/1971	38.9
1/07/1971	40.1
1/10/1971	42
3/01/1972	43.5
3/04/1972	48.3
3/07/1972	62.1
2/10/1972	65.5
1/01/1973	63.9
2/04/1973	84.4
2/07/1973	120.1
1/10/1973	103
1/01/1974	106.7
1/04/1974	168.4
1/07/1974	154.1
1/10/1974	151.8
1/01/1975	183.9
1/04/1975	177.3
1/07/1975	166.5
1/10/1975	143.5
1/01/1976	140.3
1/04/1976	129.2
1/07/1976	122.9
1/10/1976	116
3/01/1977	134.5
1/04/1977	148.3
1/07/1977	142.6
3/10/1977	155.5
2/01/1978	165
3/04/1978	183.4
3/07/1978	183.3
2/10/1978	217.1
29/12/1978	226
1/01/1979	226

Appendix H: Mall Customers Dataset

CustomerID	Gender	Age	Annual Income (k$)
1	Male	19	15
2	Male	21	15
3	Female	20	16
4	Female	23	16
5	Female	31	17
6	Female	22	17
7	Female	35	18
8	Female	23	18
9	Male	64	19
10	Female	30	19
11	Male	67	19
12	Female	35	19
13	Female	58	20
14	Female	24	20
15	Male	37	20
16	Male	22	20
17	Female	35	21
18	Male	20	21
19	Male	52	23
20	Female	35	23
21	Male		24
22	Male	25	24
23	Female	46	
24	Male	31	25
25	Female	54	28
26	Male	29	28
27	Female	45	28
28	Male	35	28
29			29
30	Female		29
31	Male	60	30
32	Female	21	30
33	Male	53	33
34	Male	18	33
35	Female	49	
36	Female	21	33
37	Female	42	34
38	Female	30	34