Data Mining and Data warehouse

For

demonstrating how raw data is transformed into meaningful insights for informed decision making

BY

Anita Dhanuk

TU Registration No: 7-2-939-32-2019

TU Roll.No:10061/19

Orchid International College

A Report on data mining and data warehouse

Faculty of Management, Tribhuvan University

in partial fulfilment of the requirements for the degree of

Bachelor of Information Management

Kathmandu, August 2023

Data Preprocessing

Data preprocessing is a critical step in the data mining and machine learning pipeline that involves cleaning, transforming, and preparing raw data into a format suitable for analysis. This process addresses issues such as missing or erroneous values, normalization or standardization of data scales, handling categorical variables through encoding, and removing noise or outliers that could skew results. Data preprocessing ensures that the data is accurate, complete, and formatted correctly before applying any analytical methods. It plays a crucial role in improving the quality and effectiveness of models built on the data, ultimately leading to more reliable insights and predictions.

Introduction to the Libraries

Pandas

Pandas is a software library written for the written for the Python Programming Language for data manipulation and analysis. In particular, it offers data structures and operations for manipulation and analysis. In Particular, it offers data structures and operations for manipulating numerical tables and time series.

Key Features

- Easy data cleaning and preprocessing.
- Intuitive handling of missing data.
- Efficient merging and joining of datasets.
- Powerful group-by functionality for aggregating and transforming data.

Matplotlib

Matplotlib is a Python library for creating static, animated, and interactive visualizations, offering a wide range of plotting functionalities including line plots, scatter plots, bar charts, histograms, and customization options for labels, colors, and styles.

Key Features

- Supports a wide variety of plots.
- Publication Quality.
- Runs seamlessly on various operating system.
- Provides options for adding interactivity to plots through widgets and GUI toolkit like Tkinter.
- Simple and intuitive interface for quick and easy plotting pf data.

NumPy

NumPy is the foundation of many other libraries in the Python scientific stack and is widely used in fields such as machine learning, data analysis, engineering simulations, and scientific research due to its efficiency and ease of use in handling large datasets and performing complex numerical computations.

Key Features

- Efficient storage and manipulation of homogenous data
- Enables concise and efficient operations on entire arrays.
- Efficient handles operations between arrays of different shapes and sizes.
- Enhances performance.

Scikit-learn

Scikit-learn is widely used for both educational purposes and production-level applications, offering a robust set of tools for data scientists and machine learning practitioners to build and deploy machine learning models effectively.

Key Features

- Wide range of machine learning algorithms for classification, regression, clustering and more.
- Tools for model selection, evaluation and validation.
- Easy-to-use interface for building and training machine learning models.
- Utilities for preprocessing data, including scaling, encoding, and splitting datasets.

Data Loading and Statistics

In this section of the lab, we will discuss about the initial steps of data mining process. Loading of the data into the environment and performing basic statistical analysis to understand its structure and key characteristics. Data loading uses the libraries like pandas in python to facilities loading data from various file formats such as CSV, Excel, SQL databases, JSON and many more. In statistics, it summaries data through measures like mean, median, mode, standard deviation, variance, range and quartiles.

To analyze and manipulate the dataset, we utilized the pandas library, a powerful tool in Python for data analysis. The dataset was loaded from a CSV file using the following code in Jupyter Notebook.

```
[1] import pandas as pd

[2] data= pd.read_csv("flights.csv")

C:\Users\Dell\AppData\Local\Temp\ipykernel_17492\3019286118.py:1: DtypeWarning: Columns (7,8) have mixed types. Specify dtype option on import or set low_m
emory=False.
  data= pd.read_csv("flights.csv")

[4] data.head()
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...	ARRIVAL_TIME
0	2015	1	1	4	AS	98	N407AS	ANC	SEA	5	...	408.0
1	2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10	...	741.0
2	2015	1	1	4	US	840	N171US	SFO	CLT	20	...	811.0
3	2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20	...	756.0
4	2015	1	1	4	AS	135	N527AS	SEA	ANC	25	...	259.0

5 rows × 31 columns

```
[ ]
```

Exploring Loaded Data

After the data is loaded, the next step is to explore it to understand its structure and contents. This involves viewing a sample of the data, inspecting the column names, and understanding the data types.

1. The **.head()** function is used to view the first few rows of the DataFrame. By default, it displays the first five rows, but you can specify a different number of rows as an argument.

```
[1]: import pandas as pd

[3]: data= pd.read_csv("flights.csv")

C:\Users\Dell\AppData\Local\Temp\ipykernel_17492\3010286118.py:1: DtypeWarning: Columns (7,8) have mixed types. Specify dtype option on import or set low_memory=False.
  data= pd.read_csv("flights.csv")

[4]: data.head()
```

[4]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...	ARRIVAL_TIME
0	2015	1	1	4	AS	98	N407AS	ANC	SEA	5	...	408.0
1	2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10	...	741.0
2	2015	1	1	4	US	840	N171US	SFO	CLT	20	...	811.0
3	2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20	...	756.0
4	2015	1	1	4	AS	135	N527AS	SEA	ANC	25	...	259.0

5 rows × 31 columns

```
[ ]:
```

The **.tail()** function is used to view the last few rows of the DataFrame. By default, it displays the last five rows, but you can specify a different number of rows as an argument.

```
[6]: data.tail(7)
```

[6]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...	ARRIVA
5819072	2015	12	31	4	B6	98	N607JB	DEN	JFK	2359	...	
5819073	2015	12	31	4	B6	66	N655JB	ABQ	JFK	2359	...	
5819074	2015	12	31	4	B6	688	N657JB	LAX	BOS	2359	...	
5819075	2015	12	31	4	B6	745	N828JB	JFK	PSE	2359	...	
5819076	2015	12	31	4	B6	1503	N913JB	JFK	SJU	2359	...	
5819077	2015	12	31	4	B6	333	N527JB	MCO	SJU	2359	...	
5819078	2015	12	31	4	B6	839	N534JB	JFK	BQN	2359	...	

7 rows × 31 columns

```
[ ]:
```

2. The **.columns** attribute provides a list of the column names in the DataFrame. This is useful for understanding the structure of the dataset and for referencing specific columns in subsequent operations.

5. **.info()** provides a concise summary of the dataframe, including the number of non-null values in each column, memory usage, and data types.

```
[11]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5819079 entries, 0 to 5819078
Data columns (total 31 columns):
 #   Column               Dtype
---  ------               -----
 0   YEAR                 int64
 1   MONTH                int64
 2   DAY                  int64
 3   DAY_OF_WEEK          int64
 4   AIRLINE              object
 5   FLIGHT_NUMBER        int64
 6   TAIL_NUMBER          object
 7   ORIGIN_AIRPORT       object
 8   DESTINATION_AIRPORT  object
 9   SCHEDULED_DEPARTURE  int64
 10  DEPARTURE_TIME       float64
 11  DEPARTURE_DELAY      float64
 12  TAXI_OUT             float64
 13  WHEELS_OFF           float64
 14  SCHEDULED_TIME       float64
 15  ELAPSED_TIME         float64
 16  AIR_TIME             float64
 17  DISTANCE             int64
 18  WHEELS_ON            float64
 19  TAXI_IN              float64
 20  SCHEDULED_ARRIVAL    int64
 21  ARRIVAL_TIME         float64
 22  ARRIVAL_DELAY        float64
 23  DIVERTED             int64
 24  CANCELLED            int64
 25  CANCELLATION_REASON  object
 26  AIR_SYSTEM_DELAY     float64
 27  SECURITY_DELAY       float64
 28  AIRLINE_DELAY        float64
```

6. **.value_counts()** is useful for categorical data, it returns a series containing counts of unique values in a column, helping to identify the frequency distribution of categorical variables.

```
[13]: data['AIRLINE'].value_counts()

[13]: AIRLINE
      WN    1261855
      DL     875881
      AA     725984
      OO     588353
      EV     571977
      UA     515723
      MQ     294632
      B6     267048
      US     198715
      AS     172521
      NK     117379
      F9      90836
      HA      76272
      VX      61903
      Name: count, dtype: int64

[ ]:
```

Handling Missing Data

Handling missing data is a crucial step in data preprocessing, as it ensures the accuracy and reliability of your analyses. Missing data can adversely affect the quality of analysis and model performance if not properly addressed. Missing data can occur due to various reasons such as data collection errors, data entry issues, or intentional missing values. Ignoring missing data or improperly handling it can lead to biased results, reduced statistical power, and inaccurate conclusions. Therefore, it is essential to address missing data before proceeding with further analysis.

Here's a step on how to handle missing data in pandas.

1. **Checking for missing values**
   To find out if there are missing values (NaNs or null values) in our dataset, we can use .isnull() and .sum().
   The **.isnull()** method returns a DataFrame of the same shape as df with boolean values indicating where
   data is missing.
   The **.sum()** method then sums up these boolean values along each column, resulting in the count of missing
   values in each column.

```
[14]: data.isna().sum()

[14]: YEAR                       0
      MONTH                      0
      DAY                        0
      DAY_OF_WEEK                0
      AIRLINE                    0
      FLIGHT_NUMBER              0
      TAIL_NUMBER            14721
      ORIGIN_AIRPORT             0
      DESTINATION_AIRPORT        0
      SCHEDULED_DEPARTURE        0
      DEPARTURE_TIME         86153
      DEPARTURE_DELAY        86153
      TAXI_OUT               89047
      WHEELS_OFF             89047
      SCHEDULED_TIME             6
      ELAPSED_TIME          105071
      AIR_TIME              105071
      DISTANCE                   0
      WHEELS_ON              92513
      TAXI_IN                92513
      SCHEDULED_ARRIVAL          0
      ARRIVAL_TIME           92513
      ARRIVAL_DELAY         105071
      DIVERTED                   0
      CANCELLED                  0
      CANCELLATION_REASON  5729195
      AIR_SYSTEM_DELAY     4755640
      SECURITY_DELAY       4755640
      AIRLINE_DELAY        4755640
      LATE_AIRCRAFT_DELAY  4755640
      WEATHER_DELAY        4755640
      dtype: int64

[ ]:
```

2. **Dropping Missing values**
   Describe how you can remove rows or columns with missing data using .dropna(). Two ways of deletion of Missing Data are:

- **Row-wise Deletion:** This involves removing entire rows of data where any missing values are present. While simple, it can lead to loss of potentially valuable information, especially if the dataset is small.
  data.dropna(axis=0,inplace=True)

```
[15]: data_copy=data.copy()

[17]: data_copy.dropna(axis=0,inplace=True)

[18]: data_copy.isna().sum()

[18]: YEAR                  0
      MONTH                 0
      DAY                   0
      DAY_OF_WEEK           0
      AIRLINE               0
      FLIGHT_NUMBER         0
      TAIL_NUMBER           0
      ORIGIN_AIRPORT        0
      DESTINATION_AIRPORT   0
      SCHEDULED_DEPARTURE   0
      DEPARTURE_TIME        0
      DEPARTURE_DELAY       0
      TAXI_OUT              0
      WHEELS_OFF            0
      SCHEDULED_TIME        0
      ELAPSED_TIME          0
      AIR_TIME              0
      DISTANCE              0
      WHEELS_ON             0
      TAXI_IN               0
      SCHEDULED_ARRIVAL     0
      ARRIVAL_TIME          0
      ARRIVAL_DELAY         0
      DIVERTED              0
      CANCELLED             0
      CANCELLATION_REASON   0
      AIR_SYSTEM_DELAY      0
      SECURITY_DELAY        0
      AIRLINE_DELAY         0
```

- **Column-wise Deletion:** Alternatively, we can remove entire columns (features) that have a significant number of missing values if those columns are not critical for our analysis.
  data.dropna(axis=1, inplace=True)

```
[99]: data_copy2=data.copy()

[21]: data_copy2.dropna(axis=1,inplace=True)

[22]: data_copy2.isna().sum()

[22]: YEAR                  0
      MONTH                 0
      DAY                   0
      DAY_OF_WEEK           0
      AIRLINE               0
      FLIGHT_NUMBER         0
      ORIGIN_AIRPORT        0
      DESTINATION_AIRPORT   0
      SCHEDULED_DEPARTURE   0
      DISTANCE              0
      SCHEDULED_ARRIVAL     0
      DIVERTED              0
      CANCELLED             0
      dtype: int64
```

3. **Filling Missing Value**
   Replacing missing values with the mean, median or mode of the respective column. This method is simple and can preserve the overall statistical properties of the data. The median of the texture length column and then we replaced the null or missing values with the calculated median tenure length.

```
[51]: category_col = ['YEAR','AIRLINE','DISTANCE','CANCELLED','CANCELLATION_REASON']
[52]: data[category_col]=data[category_col].fillna("UNKNOWN")
[53]: data.head(10)
```

[53]:

TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...	ARRIVAL_TIME	ARRIVAL_DELAY	DIVERTED	CANCELLED	CANCELLATION_REASON
N407AS	ANC	SEA	5	...	408.0	-22.0	0	0	UNKNOWN
N3KUAA	LAX	PBI	10	...	741.0	-9.0	0	0	UNKNOWN
N171US	SFO	CLT	20	...	811.0	5.0	0	0	UNKNOWN
N3HYAA	LAX	MIA	20	...	756.0	-9.0	0	0	UNKNOWN
N527AS	SEA	ANC	25	...	259.0	-21.0	0	0	UNKNOWN
N3730B	SFO	MSP	25	...	610.0	8.0	0	0	UNKNOWN
N635NK	LAS	MSP	25	...	509.0	-17.0	0	0	UNKNOWN
N584UW	LAX	CLT	30	...	753.0	-10.0	0	0	UNKNOWN
N3LAAA	SFO	DFW	30	...	532.0	-13.0	0	0	UNKNOWN
N826DN	LAS	ATL	30	...	656.0	-15.0	0	0	UNKNOWN

```
[57]: label_endoder = LabelEncoder()
      for col in category_col:
          data[col]= label_endoder.fit_transform(data[col])
[58]: data.head(10)
```

[58]:

_TIME	ARRIVAL_DELAY	DIVERTED	CANCELLED	CANCELLATION_REASON	AIR_SYSTEM_DELAY	SECURITY_DELAY	AIRLINE_DELAY	LATE_AIRCRAFT_DELAY	WEATHER_DELAY
408.0	-22.0	0	0	4	NaN	NaN	NaN	NaN	NaN
741.0	-9.0	0	0	4	NaN	NaN	NaN	NaN	NaN
811.0	5.0	0	0	4	NaN	NaN	NaN	NaN	NaN
756.0	-9.0	0	0	4	NaN	NaN	NaN	NaN	NaN
259.0	-21.0	0	0	4	NaN	NaN	NaN	NaN	NaN
610.0	8.0	0	0	4	NaN	NaN	NaN	NaN	NaN
509.0	-17.0	0	0	4	NaN	NaN	NaN	NaN	NaN
753.0	-10.0	0	0	4	NaN	NaN	NaN	NaN	NaN
532.0	-13.0	0	0	4	NaN	NaN	NaN	NaN	NaN
656.0	-15.0	0	0	4	NaN	NaN	NaN	NaN	NaN

Using Label Encoder, the missing value which were named "UNKOWN" are been replaced by integer value.

Data Normalization

Data normalization is a crucial preprocessing step in machine learning and data analysis. It ensures that all features (variables) in the dataset have a similar scale, which can improve the performance and stability of machine learning algorithms.

Why Normalize Data?

Scale Consistency: Different features in the dataset may have different ranges of values. Normalization scales all features to a uniform range, typically between 0 and 1 or centered around 0 with a standard deviation of 1. Algorithm Sensitivity: Many machine learning algorithms perform better or converge faster when the features are on a similar scale.

Gradient Descent: Algorithms like neural networks and linear regression using gradient descent are influenced by the scale of features; normalization can help in faster convergence.

Technique of Normalization

Min-Max Scaling.

Scales the data to a fixed range (often between 0 and 1). Useful when we know the range of our data and it is relatively uniform. To normalize our data using Min-Max Scaling, first we import MinMaxScaler from sklearn.preprocessing. After importing, we follow the example usage as in the shown below

```
[61]: scaler= MinMaxScaler()

[62]: col_scalar = ['YEAR','AIRLINE','DISTANCE','CANCELLED','CANCELLATION_REASON']

[64]: data[col_scalar]=scalar.fit_transform(data[col_scalar])

[65]: data.head(10)
```

[65]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...	ARRIVAL_TIME
0	0.0	1	1	4	0.076923	98	N407AS	ANC	SEA	5	...	408.0
1	0.0	1	1	4	0.000000	2336	N3KUAA	LAX	PBI	10	...	741.0
2	0.0	1	1	4	0.846154	840	N171US	SFO	CLT	20	...	811.0
3	0.0	1	1	4	0.000000	258	N3HYAA	LAX	MIA	20	...	756.0
4	0.0	1	1	4	0.076923	135	N527AS	SEA	ANC	25	...	259.0
5	0.0	1	1	4	0.230769	806	N3730B	SFO	MSP	25	...	610.0
6	0.0	1	1	4	0.615385	612	N635NK	LAS	MSP	25	...	509.0
7	0.0	1	1	4	0.846154	2013	N584UW	LAX	CLT	30	...	753.0
8	0.0	1	1	4	0.000000	1112	N3LAAA	SFO	DFW	30	...	532.0
9	0.0	1	1	4	0.230769	1173	N826DN	LAS	ATL	30	...	656.0

10 rows × 31 columns

```
[ ]:
```

Initialization: MinMaxScaler() is initialized to create an instance of the Min-Max scaler.

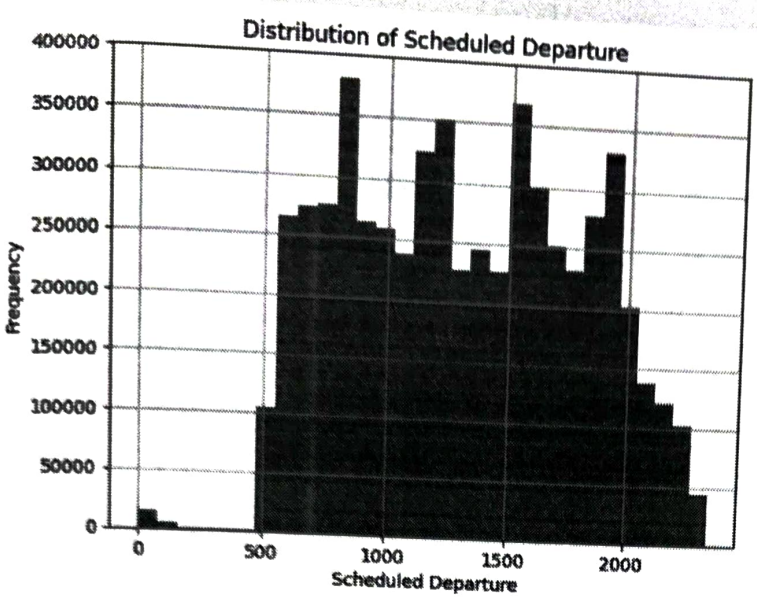
Columns to Scale: columns_to_scale is a list containing the names of columns you want to scale. Replace ['column1', 'column2', 'column3'] with the actual column names from your dataset that you wish to normalize.

Fit and Transform: scaler.fit_transform(data[columns_to_scale]) computes the minimum and maximum values for each specified column and then scales the values to the range [0, 1]. The original values in data[columns_to_scale] are replaced with the scaled values.

Exploratory Data Analysis (EDA) and Visualization

Exploratory Data Analysis (EDA) is a critical step in the data analysis process. It involves summarizing the main characteristics of a dataset and visualizing the data to uncover patterns, detect anomalies, and test hypotheses. Firstly, need to import the necessary libraries, then typically need pandas for data manipulation and analysis, and matplotlib and seaborn for visualization. Optionally, we can use plotly for interactive plots.

```
[66] data['SCHEDULED_DEPARTURE'].hist(bins=30)
plt.title('Distribution of Scheduled Departure')
plt.xlabel('Scheduled Departure')
plt.ylabel('Frequency')
plt.show()
```



Train-Test Split and Its Importance

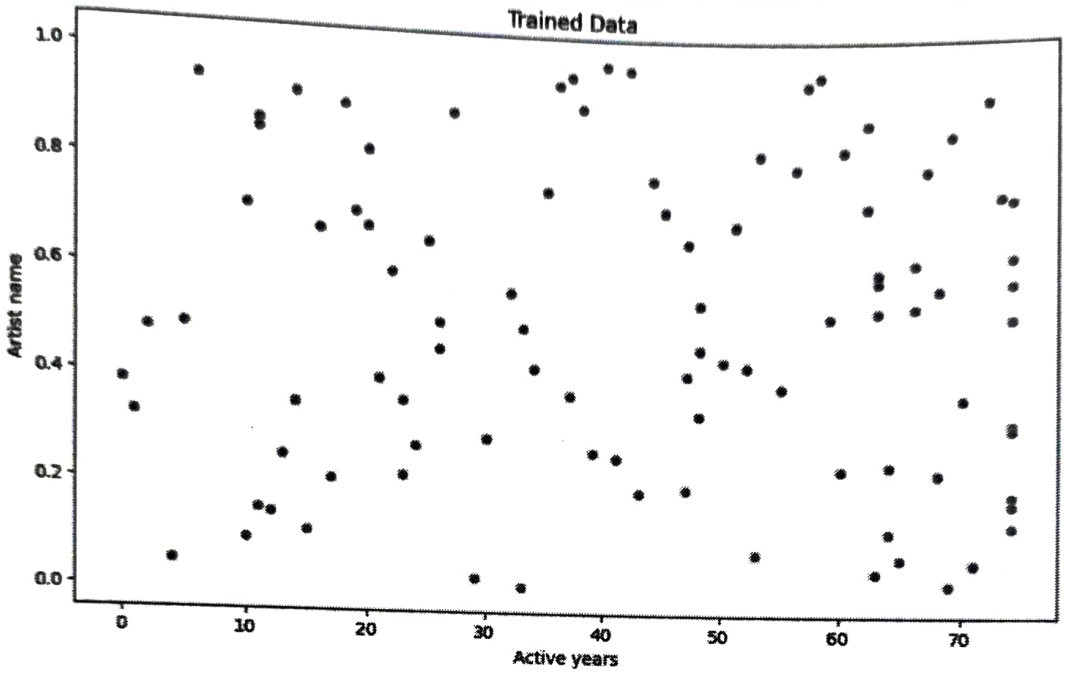
Train-Test Split is a fundamental technique used in machine learning and data analysis to evaluate the performance of a model. The dataset is divided into two subsets: the training set and the testing set. We split the dataset into training and testing sets, ensuring that the model can be properly evaluated.

Here's why it's important and how it's used:

Importance:

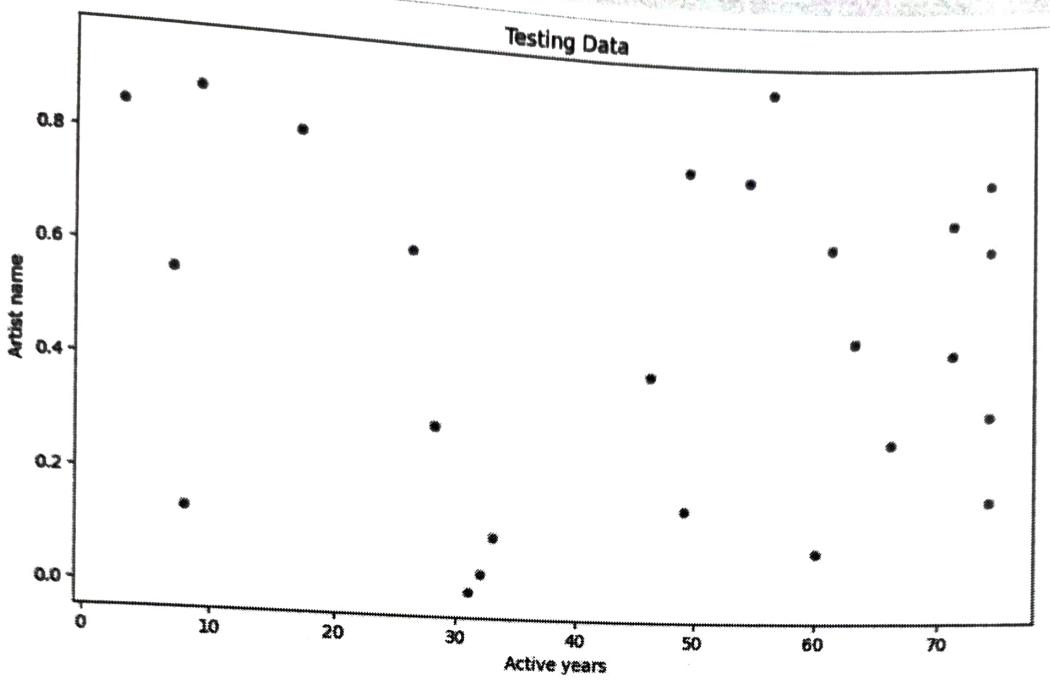
- Model Evaluation: By splitting the dataset, we can train our model on the training set and evaluate its performance on the testing set. This helps assess how well your model generalizes to unseen data.
- Overfitting and Underfitting: It helps in detecting overfitting (when a model performs well on the training data but poorly on the test data) or underfitting (when a model performs poorly on both training and test data).
- Performance Metrics: It provides an unbiased estimate of model performance. Metrics such as accuracy, precision, etc., are computed on the test set to understand how well the model is expected to perform on new data

```
[78]: X= df[["Active years"]]
      y= df["Artist name"]

[79]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[82]: plt.figure(figsize=(10,6))
      sns.scatterplot(x=X_train['Active years'],y=y_train)
      plt.title("Trained Data")
      plt.xlabel("Active years")
      plt.ylabel("Artist name")
      plt.show()
```



This visualization is useful for inspecting the distribution of active years across the dataset of artists. The random scatter of points indicates that there is no direct relationship between an artist's name and their active years, suggesting that the duration of an artist's career is independent of their identity within the dataset. This insight could be a preliminary step before applying more complex analyses or machine learning models to understand other potential patterns or correlations in the data.

```
[83]: plt.figure(figsize=(10,6))
      sns.scatterplot(x=X_test['Active years'],y=y_test)
      plt.title("Testing Data")
      plt.xlabel("Active years")
      plt.ylabel("Artist name")
      plt.show()
```



 We can see that both plots show a similar spread and density of points across the title axis, suggesting that the train-test split has maintained a consistent distribution of data.
 There are vertical clusters of points at specific title values in both the training and testing data. This indicates that certain title values are associated with a wide range of education values.
 There doesn't appear to be a strong linear relationship between title and education. The points are scattered throughout the plot without a clear upward or downward trend. This suggests that the relationship between title and education might be complex or that other factors influence education beyond title.

Conclusion

Hence, In this lab on Data Mining and Data Warehousing, we explored essential data preprocessing steps including data loading, handling missing data, converting categorical values to numeric, normalization, and exploratory data analysis (EDA). Using libraries like Pandas, Matplotlib, NumPy, and Scikit-learn, we ensured our dataset was clean and well-prepared for analysis. We highlighted the importance of train-test splitting to evaluate model performance and prevent overfitting, demonstrating a structured approach to uncovering hidden patterns and relationships within the data for informed decision-making.

Decision Tree Visualization

A decision tree shows a connected hierarchy of boxes to represent the values of records. Records are segmented into groups, which are called nodes. Each node contains records that are statistically similar to each other with respect to that target field.

Data scientists use decision tress because they can be used to visually and explicitly represent decisions and decision making. The decision tree algorithm is widely used because of its robustness to noise, tolerance against missing information, low computational cost, interpretability, fast run time, and robust predictors.

The decision tree algorithm can be thought of as a tree turned upside down since that root is at the top. When working with data, at each layer of the tree there is a differentiator that splits the samples into the two or more homogenous sets. Depending on the hyperparameters and settings of the decision tree, the tree continues to split until it reaches the final layers, referred to as the leaf.

Dataset

In order to understand a decision tree and visualize the model, we will use the dataset. Dataset helps to visualize the illustrated dataset in a decision tree. In this lab, we'll be using Python with libraries such as 'scikit-learn' for model creation and 'matplotlib' and 'graphviz' for visualization.

Steps

1. **Load the data**

   First, we should load the data into the memory. Then we can look at the features and take a peek at some of the samples of each class. Personally, I like the structure of a pandas DataFrame, so I'll share how to see the data in that form. To get all the data in one DataFrame.

```
[41]: import pandas as pd
      from sklearn.preprocessing import LabelEncoder
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import classification_report, accuracy_score,confusion_matrix
      from sklearn import tree
      from sklearn.tree import DecisionTreeClassifier, plot_tree
      import matplotlib.pyplot as plt

[2]: data = pd.read_csv("Downloads/Employee.csv")

[3]: data.head()

[3]:
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	...	Number of Dependents	Job Level	Company Size	Company Tenure	Remote Work	La Oppc
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	...	0	Mid	Medium	89	No	
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	...	3	Mid	Medium	21	No	
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	...	3	Mid	Medium	74	No	
3	65791	36	Female	7	Education	3989	Good	High	High	1	...	2	Mid	Small	50	Yes	
4	65026	56	Male	41	Education	4821	Fair	Very High	Average	0	...	0	Senior	Medium	68	No	

5 rows × 24 columns

2. Split the data

Before we can train the model to generate predictions about the weather to play tennis based on the measurements of the weather, we need to section off part of the data called the test data that will not be used to train the model. To do this, we use scikit-learns's Train_test_Split function from the model_Selection module.

```
[14]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, stratify = y,random_state=2)
[ ]:
[15]: X_train
```

[15]:

	outlook	temp	humidity	wind
13	1	2	0	0
5	1	0	1	0
9	1	2	1	1
12	0	1	1	1
1	2	1	0	0
2	0	1	0	1
7	2	2	0	1
4	1	0	1	1
11	0	2	0	0
8	2	0	1	1
3	1	2	0	1

3. Training the Decision Tree

In order to build the decision tree, we will use the scikit_learn Decsion Tree Classifier from the tree module.

```
[16]: model = DecisionTreeClassifier()
[51]: # model = DecisionTreeClassifier(criterion='entropy')
[52]: # help(model)
[17]: model.fit(X, y)
```

[17]:

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

When building a decision tree, we often prune the model using hyperprarameter tuning. When left unpruned, a decisn tree is likely to be overfit. Take a look at this helpful guide that goes through hyperparameter tuning decisions that can be made with a decision tree. For the purposes of this example, we will move forward with the decisions tree with default hyperparameters.

4. Evaluating the model

Now, we run the test data through the model and determine the predictions.

```
[28]: y_pred = model.predict(X_test)

[32]: accuracy = accuracy_score(y_pred, y_test.values)

[33]: accuracy

[33]: 1.0

[34]: classification_rep = classification_report(y_pred, y_test.values)

[35]: print(classification_rep)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	2
accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3

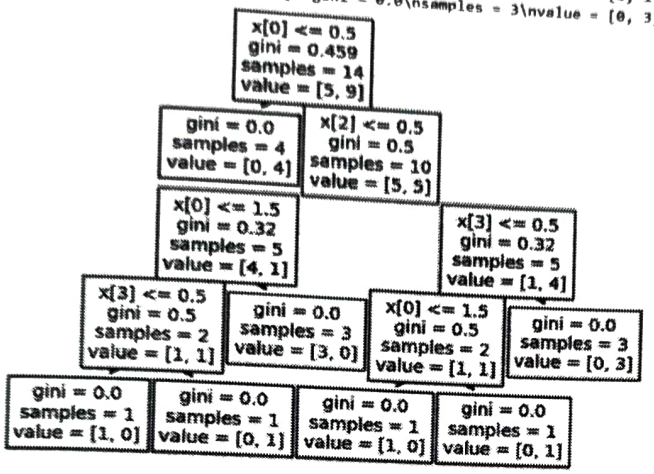
From this classification report, we can see that the accuracy score is 1.0 meaning got 100% of the predictions correct.

5. Visualizing the decision tree

In order to help make sense of the model and its results, it can be extremely helpful to view the decision tree. There are numerous ways to make the decision tree visible.

```
[21]: class_names = data['play'].unique().tolist()

[22]: class_names

[22]: [0, 1]

[23]: tree.plot_tree(model)
```

```
[23]: tree.plot_tree(model)

[23]: [Text(0.4444444444444444, 0.9, 'x[0] <= 0.5\ngini = 0.459\nsamples = 14\nvalue = [5, 9]'),
 Text(0.3333333333333333, 0.7, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5555555555555556, 0.7, 'x[2] <= 0.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]'),
 Text(0.3333333333333333, 0.5, 'x[0] <= 1.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
 Text(0.2222222222222222, 0.3, 'x[3] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.1111111111111111, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3333333333333333, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4444444444444444, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.7777777777777778, 0.5, 'x[3] <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.6666666666666666, 0.3, 'x[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.5555555555555556, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7777777777777778, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8888888888888888, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]')]
```



Finally, we can display the decision tree to see how the decisions were made in the model to classify the different weather. We can display it in the file directly in a Jupyter Notebook. From this visualization, we can see how the default parameters of a decision tree allow for continued splits to any depth to perfectly classify the data. This is why pruning is a very important practice.

Conclusion

Data scientists use decision tree classifiers for many reasons. One of the main advantages of decision trees is that their outputs are easy to read and interpret without requiring statistical knowledge, making them useful for nontechnical audiences. Because of this, having ways to visualize the model built by the decision tree classifier is beyond invaluable and using the plot_tree library is a quick way to arrive at an interpretable product.

Clustering

Clustering is a fundamental technique in unsupervised machine learning where the goal is to group similar data points into clusters. It's widely used across various domains for tasks such as customer segmentation, anomaly detection, and pattern recognition.

Algorithm used in Clustering are

1. K-Means
2. Hierarchical Clustering
3. DBCSAN
4. Mean Shift

In this lab, we will perform a clustering analysis on a customer segmentation dataset using the K-means algorithm. The analysis involves several steps including data preprocessing, visualization, determine the optimal number of clusters, and interpreting the clustering results.

Objectives

The objective of this report is to analyze and document the process and results of applying clustering techniques to a given dataset. Clustering helps in identifying natural groupings within the data, which can be useful for various applications such as customer segmentation, anomaly detection, and data exploration.

Steps

1. Installation of Required library

```
[21]: !pip install kneed

Defaulting to user installation because normal site-packages is not writeable
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: numpy>=1.14.2 in c:\users\dell\appdata\roaming\python\python311\site-packages (from kneed) (1.26.4)
Requirement already satisfied: scipy>=1.0.0 in c:\users\dell\appdata\roaming\python\python311\site-packages (from kneed) (1.13.0)
Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Installing collected packages: kneed
Successfully installed kneed-0.8.5
```

'Kneed' is a Python package that provides a knee-point detection algorithm for finding the 'knee' or elbow point in a curve, which is often used in optimization problems or data analysis where selecting the optimal number of clusters or parameters is crucial.

2. Import of the libraries

```
[23]: import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      from kneed import KneeLocator
      from sklearn.cluster import KMeans
      from sklearn.preprocessing import StandardScaler
```

The analysis uses the above libraries

'pandas': For data manipulation.

'matplotlib' and 'seaborn': For data visualization.

'kneed': For finding the optimal number of clusters.

'sklearn': For scaling data and performing K-means clustering.

3. Loading the data

The dataset is loaded into a pandas DataFrame from a CSV file named 'Customer_Segmentation.csv'.

```
[10]: data = pd.read_csv("Customer_Segmentation.csv")
[11]: data
```

[11]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

4. Feature Selection

In the below figure, three different features are selected for clustering:

- Age
- Annual Income(k$)
- Spending Score(1-100)

```
[12]: X = data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
[13]: X
```

[13]:

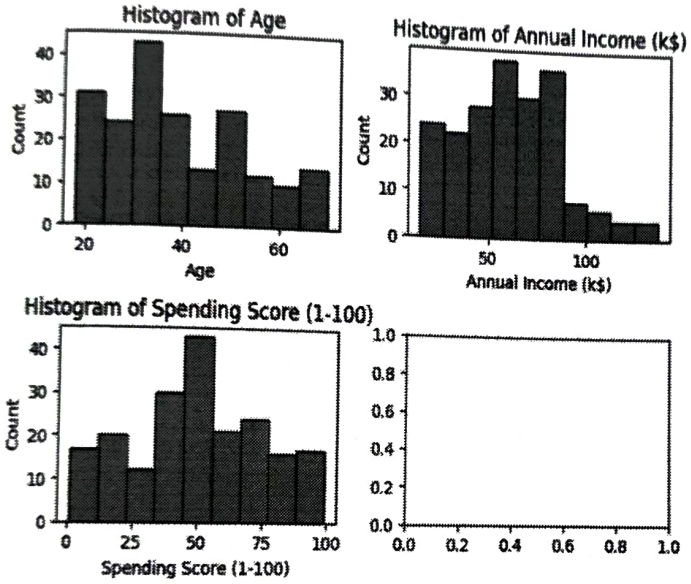
	Age	Annual Income (k$)	Spending Score (1-100)
0	19	15	39
1	21	15	81
2	20	16	6
3	23	16	77
4	31	17	40
...	...	...	...
195	35	120	79
196	45	126	28
197	32	126	74
198	32	137	18
199	30	137	83

200 rows × 3 columns

5. Data Visualization

In the below figure, Histogram are plotted for each selected feature to understand their distribution. Here to plot the data in histrogram and seaborn is used to plot the histogram.

```
[16]: fig, axs = plt.subplots(2, 2)

      # Flatten the subplots array into a 1D array
      axs = axs.ravel()

      # Iterate over the columns of the X DataFrame and their corresponding subplot axes
      for i, col in enumerate(X.columns):
          # Plot a histogram of the current column on the corresponding subplot axis
          sns.histplot(X[col], ax=axs[i])

          # Set the title of the subplot to indicate the column name
          axs[i].set_title('Histogram of ' + col)

      plt.tight_layout()
      plt.show()
```



We can see in the above figure, histogram is plotted on the basis of feature selection (Age, Annual Income and Spending Score).

6. Data Scaling

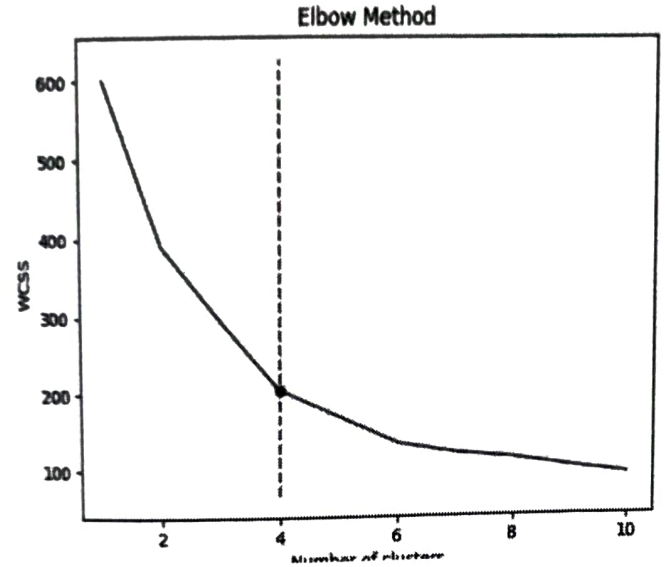
The features are scaled using 'StandardScaler' to ensure they have a mean of 0 and a standard of deviation of 1.

```
[15]: # Create a StandardScaler object
      scaler = StandardScaler()

      # Scale the data using the fit_transform method
      X_scaled = scaler.fit_transform(X)
```

7. Determining Optimal Number of Clusters

The Elbow method is used to find the optimal number of clusters. The Within-Cluster Sum if Squares (WCSS) is calculated for different values of K (number of clusters), and the 'knee' point is identified using the 'kneeLocator'.

```
# compute WCSS for different values of k
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# plot WCSS vs no  of clusters
sns.lineplot(x=range(1, 11), y=wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters ')
plt.ylabel('WCSS')

# find the knee location
knee = KneeLocator(range(1, 11), wcss, curve='convex', direction='decreasing')

# plot the knee location
plt.vlines(knee.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.scatter(knee.knee, knee.knee_y, color='red', s=30)

plt.show()
```



8. K-means Clustering

K-means clustering is performed with the optimal number of clusters determined from the Elbow method.

```
# Instantiate a KMeans object with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the K-means model to the scaled data and obtain the cluster labels for each data point
y_kmeans = kmeans.fit_predict(X_scaled)
```

The cluster labels are added to the original dataset.

```
# add the cluster labels to the original data
data['cluster'] = y_kmeans

data
```

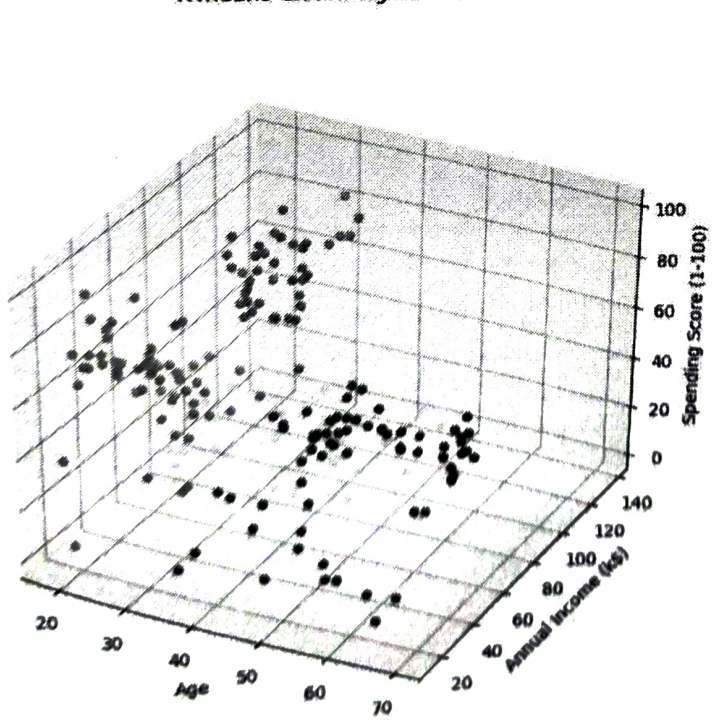
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	1	Male	19	15	39	2
1	2	Male	21	15	81	2
2	3	Female	20	16	6	2
3	4	Female	23	16	77	2
4	5	Female	31	17	40	2
...	...	...	...	...	...	...
195	196	Female	35	120	79	1
196	197	Female	45	126	28	3
197	198	Male	32	126	74	1
198	199	Male	32	137	18	3
199	200	Male	30	137	83	1

200 rows × 6 columns

9. 3D Visualization of Clusters

A 3D scatter plot is created to visualize the clusters in the dataset.

```
[20]: fig = plt.figure(figsize=(10, 8))
      ax = fig.add_subplot(111, projection='3d')

      # Create a 3D scatter plot of the data points
      ax.scatter3D(data['Age'], data['Annual Income (k$)'], data['Spending Score (1-100)'], c=data['cluster'], cmap='viridis')

      # Set labels for each axis
      ax.set_xlabel('Age')
      ax.set_ylabel('Annual Income (k$)')
      ax.set_zlabel('Spending Score (1-100)')

      # Set a title for the plot
      plt.title('K-means Clustering: 3D Plot')

      ax.set_box_aspect(aspect=None, zoom=0.9)
      plt.show()
```



Conclusion

The analysis successfully segments the customers into distinct clusters based on their age, annual income, and spending score. The optimal number of clusters was determined using the Elbow method, and the clusters were visualized in a 3D scatter plot.

Association

In the context of data mining and machine learning, "association" refers to discovering relationships or patterns among a set of items in large datasets. This process is commonly used to identify rules that highlight how certain items often appear together within transactions or events.

Association Rule Learning

Association rule learning is a technique to uncover interesting relationships or associations between variables in large databases. This is often applied in market basket analysis, where the goal is to understand the purchase behavior of customers by finding associations between different items in their shopping carts.

Objectives

The objectives of this lab is to perform association rule learning on customer shopping data using the Apriori algorithm, The goal is to identify interesting relationships between items purchased by customers

Steps and classification

1. Installation of Required Libraries

```
[1] !pip install mlxtend

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: mlxtend in c:\users\dell\appdata\roaming\python\python311\site-packages (0.23.1)
Requirement already satisfied: scipy>=1.2.1 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (1.13.0)
Requirement already satisfied: numpy>=1.16.2 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (1.26.4)
Requirement already satisfied: pandas>=0.24.2 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (2.2.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (1.4.2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (3.8.4)
Requirement already satisfied: joblib>=0.13.2 in c:\users\dell\appdata\roaming\python\python311\site-packages (from mlxtend) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (4.51.
1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.
5)
Requirement already satisfied: packaging>=20.0 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (24.0)
Requirement already satisfied: pillow>=8 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (2.
9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\appdata\roaming\python\python311\site-packages (from pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\appdata\roaming\python\python311\site-packages (from pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\appdata\roaming\python\python311\site-packages (from scikit-learn>=1.0.2->mlxtend)
(3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\dell\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->ml
xtend) (1.16.0)
```

```
[5] !pip install --upgrade openpyxl

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: openpyxl in c:\users\dell\appdata\roaming\python\python311\site-packages (3.1.0)
Collecting openpyxl
  Using cached openpyxl-3.1.5-py2.py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: et-xmlfile in c:\programdata\anaconda3\lib\site-packages (from openpyxl) (1.1.0)
Using cached openpyxl-3.1.5-py2.py3-none-any.whl (250 kB)
Installing collected packages: openpyxl
  Attempting uninstall: openpyxl
    Found existing installation: openpyxl 3.1.0
    Uninstalling openpyxl-3.1.0:
      Successfully uninstalled openpyxl-3.1.0
Successfully installed openpyxl-3.1.5
```

```
[4]: !pip install openpyxl==3.1.0

Defaulting to user installation because normal site-packages is not writeable
Collecting openpyxl==3.1.0
  Using cached openpyxl-3.1.0-py2.py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: et-xmlfile in c:\programdata\anaconda3\lib\site-packages (from openpyxl==3.1.0) (1.1.0)
Using cached openpyxl-3.1.0-py2.py3-none-any.whl (250 kB)
Installing collected packages: openpyxl
  Attempting uninstall: openpyxl
    Found existing installation: openpyxl 3.1.5
    Uninstalling openpyxl-3.1.5:
      Successfully uninstalled openpyxl-3.1.5
Successfully installed openpyxl-3.1.0
```

Purpose: Ensure the required libraries (mlxtend, openpyxl) are installed and up-to-date.

- mlxtend: Provides the 'apriori' function for generating frequent itemsets and 'association_rules' for deriving rules from these itemsets.
- 'openpyxl' : Ensures compatibility with the version required for reading the dataset from an Excel file

2. Import Libraries

```
[1]: import numpy as np
     import pandas as pd
     from mlxtend.frequent_patterns import apriori, association_rules
```

Purpose: Importing necessary libraries for data manipulation and association rule learning.

3. Load and Explore Data

```
[1]: import numpy as np
     import pandas as pd
     from mlxtend.frequent_patterns import apriori, association_rules

[2]: cust_data = pd.read_excel('Customer_Shopping_data.xlsx')
     cust_data.head()
```

[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Purpose: Load the dataset and display the first few rows to understand its structure.

4. Data Cleaning and Preparation

Data cleaning and preparation are vital steps in any data analysis or machine learning project. They ensure that the data is accurate, consistent, and in a suitable format for analysis. Proper data cleaning and preparation lead to more reliable results, better performance of algorithms, and meaningful insights. This foundational process is essential for making informed decisions based on data.

```
# Remove the unnecessary spaces in the description
cust_data['Description'] = cust_data['Description'].str.strip()

# Dropping the rows without any invoice number
cust_data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)
cust_data['InvoiceNo'] = cust_data['InvoiceNo'].astype('str')
```

```
cust_data
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

541909 rows × 8 columns

In the above figure. 'cust_data' removes the unnecessary spaces using str.strip() because of the following reasons:

- Ensures Consistency
- Prevents Errors in Analysis
- Facilitates Accurate Grouping and Aggregation
- Improving Matching and Merging
- Enhances Data Quality.

Dropping rows with missing invoice number for ensuring Data integrity, for making the accurate transaction Identification and avoiding Analytical Error.

5. Data Subsetting

Data Subsetting is the process of selecting a specific portion of your dataset based on certain criteria. This is crucial for focusing on relevant data, improving analysis efficiency, and reducing computational complexity.

In our Lab, we are focusing on subsetting the dataset for transactions from a specific country (The United kingdom).

```
cust_data_uk = cust_data[cust_data['Country'] =="United Kingdom"]
    .groupby(['InvoiceNo', 'Description'])['Quantity']
    .sum().unstack().reset_index().fillna(0)
    .set_index('InvoiceNo')
```

```
cust_data_uk
```

Description	*Boombox iPod Classic	*USB Office Mirror Ball	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD	12 HANGING EGGS HAND PAINTED	12 IVORY ROSE PEG PLACE SETTINGS	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	...	wrongly coded 20713	wrongly coded 23343	wrongly coded-23343	wrongly marked
InvoiceNo															
536365	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
536366	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
536367	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
536368	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
536369	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
C581484	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
C581490	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
C581499	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
C581568	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
C581569	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

6. One-Hot Encoding

One-Hot Encoding is a purpose of converting categorical data into a binary format, indicating the presence or absence of each category (or item) with 1s and 0s, respectively. In the context of transaction data, one-hot encoding helps transforming the data into a format suitable for association rule mining algorithms like Apriori.

In our lab, the transaction data are being converting using into a one-hot encoded format where the presence of an item in a transaction is indicated by 1 and absence by 0.

```
[9] def one_hot_encoding(x):
        if(x<= 0):
            return 0
        if(x>= 1):
            return 1

[10] cust_data_UK_encoded = cust_data_UK.map(one_hot_encoding)

[11] cust_data_UK_encoded
```

[11]:

Description	*Boombox Ipod Classic	*USB Office Mirror Ball	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD	12 HANGING EGGS HAND PAINTED	12 IVORY ROSE PEG PLACE SETTINGS	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	...	wrongly coded 20713	wrongly coded 23343	wrongly coded-23343	wrongly marked
InvoiceNo															
536365	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
536366	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
536367	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
536368	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
536369	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
C581484	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
C581490	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
C581499	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

7. Apply Apriori Algorithm

Applying the Apriori algorithm to transaction data helps to identify frequent itemsets and generate association rules, providing valuable insight for market basket analysis, customer behavior understanding and data-driven decision making. By transforming the data into a suitable format and using the Apriori algorithm, businesses can uncover meaningful patterns and relationships that inform various strategic initiatives.

As shown in the following figure, Apriori Algorithm is being applied here for building the model. Here, low_memory is also being used to enabling the low_memory parameter in the Apriori function helps prevent memory errors with large datasets.

```
# Building the model
frq_items = apriori(cust_data_UK_encoded, min_support=0.01, use_colnames=True, low_memory=True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric="lift", min_threshold=1)
rules = rules.sort_values(['confidence', 'lift'], ascending=[False, False])
```

[17]:
```
rules.head()
```

[17]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
1694	(REGENCY TEA PLATE ROSES, REGENCY TEA PLATE PINK)	(REGENCY TEA PLATE GREEN)	0.010708	0.015246	0.010118	0.944915	61.979129	0.009955	17.877078	0.994515
1726	(CHARLOTTE BAG PINK POLKADOT, WOODLAND CHARLOT...	(RED RETROSPOT CHARLOTTE BAG)	0.011344	0.041064	0.010527	0.928000	22.599107	0.010061	13.318562	0.966716
1693	(REGENCY TEA PLATE GREEN, REGENCY TEA PLATE PINK)	(REGENCY TEA PLATE ROSES)	0.010935	0.017288	0.010118	0.925311	53.524760	0.009929	13.157428	0.992167
1698	(CHARLOTTE BAG SUKI DESIGN, STRAWBERRY CHARLOT...	(RED RETROSPOT CHARLOTTE BAG)	0.011797	0.041064	0.010799	0.915385	22.291891	0.010315	11.332885	0.966543
1754	(REGENCY CAKESTAND 3 TIER, ROSES REGENCY TEACU...	(GREEN REGENCY TEACUP AND SAUCER)	0.013204	0.042379	0.011979	0.907216	21.407007	0.011419	10.321022	0.966042

8. Generating Association Rules


```
# Building the model
frq_items = apriori(cust_data_UK_encoded, min_support=0.01, use_colnames=True, low_memory=True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric="lift", min_threshold=1)
rules = rules.sort_values(['confidence', 'lift'], ascending=[False, False])
```

[17]:
```
rules.head()
```

[17]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
1694	(REGENCY TEA PLATE ROSES, REGENCY TEA PLATE PINK)	(REGENCY TEA PLATE GREEN)	0.010708	0.015246	0.010118	0.944915	61.979129	0.009955	17.877078	0.994515
1726	(CHARLOTTE BAG PINK POLKADOT, WOODLAND CHARLOT...	(RED RETROSPOT CHARLOTTE BAG)	0.011344	0.041064	0.010527	0.928000	22.599107	0.010061	13.318562	0.966716
1693	(REGENCY TEA PLATE GREEN, REGENCY TEA PLATE PINK)	(REGENCY TEA PLATE ROSES)	0.010935	0.017288	0.010118	0.925311	53.524760	0.009929	13.157428	0.992167
1698	(CHARLOTTE BAG SUKI DESIGN, STRAWBERRY CHARLOT...	(RED RETROSPOT CHARLOTTE BAG)	0.011797	0.041064	0.010799	0.915385	22.291891	0.010315	11.332885	0.966543
1754	(REGENCY CAKESTAND 3 TIER, ROSES REGENCY TEACU...	(GREEN REGENCY TEACUP AND SAUCER)	0.013204	0.042379	0.011979	0.907216	21.407007	0.011419	10.321022	0.966042

It's purpose is to generate association rules from the frequent itemsets based on the lift metric and sort them by confidence and lift in descending order.

9. Experiment for different min_support

[18]:
```
# Experiment with different min_support values
for min_sup in [0.01, 0.02, 0.05]:
    frq_items = apriori(cust_data_UK_encoded, min_support=min_sup, use_colnames=True)
    rules = association_rules(frq_items, metric="lift", min_threshold=1)
    print(f"Number of rules with min_support {min_sup}: {len(rules)}")
```

```
Number of rules with min_support 0.01: 1810
Number of rules with min_support 0.02: 88
Number of rules with min_support 0.05: 0
```

[ ]:

Its purpose is to evaluate how the number of generated rules changes with different minimum support values.

Conclusion

This lab successfully demonstrates the process of performing association rule learning on customer shopping data. By following the steps outlined, one can identify interesting itemsets and rules that can provide valuable insights into customer purchasing behavior. Adjusting parameters and optimizing data types are crucial for handling large datasets efficiently.

Naïve Bayes Sentiment Classification

Introduction

The objective of this project is to perform sentiment analysis on a dataset of movie reviews. Sentiment analysis aims to determine the sentiment behind a piece of text, which can be positive or negative. In this Lab, we use a combination of natural language processing (NLP) techniques and machine learing algorithms to classify movie reviews as positive or negative.

Data Loading and Preprocessing

Data Loading

The dataset used in this project contains movie reviews along with their corresponding sentiment labels. The data is loaded from a CSV file.

[8]:
```
# Step 1: Load the dataset
data = pd.read_csv('IMDB Dataset - IMDB Dataset.csv', encoding='utf-8')
```

19]:
```
data.head()
```

19]:

	review	sentiment	processed_review
0	One of the other reviewers has mentioned that ...	positive	one reviewer mentioned watching oz episode you...
1	A wonderful little production. The...	positive	wonderful little production br br filming tech...
2	I thought this was a wonderful way to spend ti...	positive	thought wonderful way spend time hot summer we...
3	Basically there's a family where a little boy ...	negative	basically there family little boy jake think t...
4	Petter Mattei's "Love in the Time of Money" is...	positive	petter matteis love time money visually stunni...

Data Preprocessing

Preprocessing involves several steps to clean and prepare the text data for analysis.

1. Tokenization: It is the process of breaking down text into smaller units called tokens, which can be words or phrases.
2. Stop Words Removal: Stop words are common words that usually do not carry significant meaning and are often removed to focus on more meaningful words.
3. Stemming: It is the process of reducing words to their base or root form, treating different forms of the same word as a single entity.

[6]:
```
# Step 2: Text Preprocessing
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

[7]:
```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Tokenize the text
    tokens = nltk.word_tokenize(text)
    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)
```

[8]:
```
# Apply preprocessing to the 'review' column
data['processed_review'] = data['review'].apply(preprocess_text)
```

[9]:
```
data
```

[9]:

	review	sentiment	processed_review
0	One of the other reviewers has mentioned that ...	positive	one reviewer mentioned watching oz episode you...
1	A wonderful little production. The...	positive	wonderful little production br br filming tech...
2	I thought this was a wonderful way to spend ti...	positive	thought wonderful way spend time hot summer we...
3	Basically there's a family where a little boy ...	negative	basically there family little boy jake think t...
4	Petter Mattei's "Love in the Time of Money" is...	positive	petter matteis love time money visually stunni...
...	...	...	...
49995	I thought this movie did a down right good job...	positive	thought movie right good job wasnt creative or...
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative	bad plot bad dialogue bad acting idiotic direc...
49997	I am a Catholic taught in parochial elementary...	negative	catholic taught parochial elementary school nu...
49998	I'm going to have to disagree with the previou...	negative	im going disagree previous comment side maltin...
49999	No one expects the Star Trek movies to be high...	negative	one expects star trek movie high art fan expec...

50000 rows × 3 columns

Data Splitting

The dataset is split into training and testing sets to evaluate the model's performance on unseen data.

[10]:
```
# Step 3: Split the dataset into training and testing sets
X = data['processed_review']
y = data['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

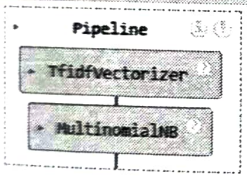
Model Building and Training

A machine learning pipeline is created using 'TfidfVectorizer' and 'Native Bayes' classifier. 'TfidfVectorizer' coverts the text data into numerical features, and 'Naïve Bayes' is used for classification.

[11]:
```
# Step 4: Create a pipeline with TfidfVectorizer and Naive Bayes
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(ngram_range=(1, 2), max_features=5000, min_df=5, max_df=0.7)),
    ('classifier', MultinomialNB())
])
```

[12]:
```
# Step 5: Train the model
pipeline.fit(X_train, y_train)
```

[12]:



Model Evaluation

The trained model is evaluated using the test set. Evaluation metrics inclide accuracy, confusion matrix, and classification report.

[13]:
```
# Step 6: Make predictions on the test set
y_pred = pipeline.predict(X_test)
```

[14]:
```
# Step 7: Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8566

Confusion Matrix:
[[4188  773]
 [ 661 4378]]

Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.84      0.85      4961
    positive       0.85      0.87      0.86      5039

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

Testing with New Data

The model is tested with new, unseen reviews to predict their sentiment.

[16]:
```
# Step 8: Test with new reviews
new_reviews = [
    "This movie was excellent! I loved it.",
    "Terrible film. I hated every minute of it.",
    "An average movie, nothing special."
]
```

[17]:
```
# Preprocess new reviews
processed_new_reviews = [preprocess_text(review) for review in new_reviews]

# Make predictions
new_predictions = pipeline.predict(processed_new_reviews)

for review, sentiment in zip(new_reviews, new_predictions):
    print(f"Review: {review}")
    print(f"Predicted Sentiment: {sentiment}\n")
```

```
Review: This movie was excellent! I loved it.
Predicted Sentiment: positive

Review: Terrible film. I hated every minute of it.
Predicted Sentiment: negative

Review: An average movie, nothing special.
Predicted Sentiment: negative
```

Conclusion

The Naïve Bayes sentiment classification model achieved an accuracy of 86% in classifying movie reviews as positive or negative. The TfidfVectorizer and Naïve Bayes classifier proved to be effective for this task. Future work could involve exploring more advanced models and techniques, such as deep learning, to further improve accuracy.