Experiment No 6

* 1. **Aim/Purpose of the Experiment**

To familiarize the students with Model building using K-Means Clustering.

* 1. **Learning Outcomes**

Knowledge of the Data cleaning, Data preparation, modelling K-Means Clustering, and different libraries in python.

* 1. **Prerequisites**

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

* 1. **Materials/Equipment/Apparatus / Devices/Software required**

Jupyter Notebook.

* 1. **Introduction and Theory**

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. It's widely used for classification tasks where the dependent variable is categorical and has two possible outcomes.

Here's an overview of the key concepts and components of logistic regression:

* Binary outcome: Logistic regression is specifically designed for situations where the dependent variable (also known as the response variable or target variable) is binary, meaning it has only two possible outcomes. These outcomes are typically represented as 0 and 1, or as "success" and "failure", "yes" and "no", etc.
* Logistic function (sigmoid function): In logistic regression, the relationship between the predictor variables and the probability of the binary outcome is modeled using the logistic function, also known as the sigmoid function. The logistic function is an S-shaped curve that maps any real-valued number to a value between 0 and 1, representing probabilities.
* Probability prediction: Unlike linear regression, where the output is continuous, logistic regression predicts the probability that a given observation belongs to a particular category (e.g., the probability of a customer buying a product). The predicted probabilities are then used to make classifications.
* Logit transformation: The logistic function is expressed in terms of the log-odds, also known as the logit function. The logit of the probability of the event occurring (p) is defined as the logarithm of the odds ratio (p / (1 - p)). Mathematically, it can be represented as log(p / (1 - p)).
* Model parameters: Similar to linear regression, logistic regression estimates parameters (coefficients) that define the relationship between the predictor variables and the log-odds of the binary outcome. These parameters are estimated using maximum likelihood estimation or other optimization techniques.
* Interpretation of coefficients: The coefficients obtained from logistic regression represent the change in the log-odds of the outcome associated with a one-unit change in the corresponding predictor variable, holding other variables constant.
* Decision boundary: In logistic regression, a decision boundary is used to classify observations into different categories based on their predicted probabilities. The decision boundary is typically set at 0.5, meaning that observations with predicted probabilities greater than 0.5 are classified into one category, while those with predicted probabilities less than or equal to 0.5 are classified into the other category.

**K-Means Clustering**

1. Read and visualise the data

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

​

import datetime as dt

​

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

​

from scipy.cluster.hierarchy import linkage

from scipy.cluster.hierarchy import dendrogram

from scipy.cluster.hierarchy import cut\_tree

# read the dataset

retail\_df = pd.read\_csv("Online\_Retail.csv", sep=",", encoding="ISO-8859-1", header=0)

retail\_df.head()

# read the dataset

retail\_df = pd.read\_csv("Online\_Retail.csv", sep=",", encoding="ISO-8859-1", header=0)

retail\_df.head()

2. Clean the data

# missing values

round(100\*(retail\_df.isnull().sum())/len(retail\_df), 2)

# drop all rows having missing values

retail\_df = retail\_df.dropna()

retail\_df.shape

retail\_df.head()

# new column: amount

retail\_df['amount'] = retail\_df['Quantity']\*retail\_df['UnitPrice']

retail\_df.head()

3. Prepare the data for modelling

#R (Recency): Number of days since last purchase

#F (Frequency): Number of tracsactions

#M (Monetary): Total amount of transactions (revenue contributed)

# monetary

grouped\_df = retail\_df.groupby('CustomerID')['amount'].sum()

grouped\_df = grouped\_df.reset\_index()

grouped\_df.head()

# frequency

frequency = retail\_df.groupby('CustomerID')['InvoiceNo'].count()

frequency = frequency.reset\_index()

frequency.columns = ['CustomerID', 'frequency']

frequency.head()

# merge the two dfs

grouped\_df = pd.merge(grouped\_df, frequency, on='CustomerID', how='inner')

grouped\_df.head()

retail\_df.head()

# recency

# convert to datetime

retail\_df['InvoiceDate'] = pd.to\_datetime(retail\_df['InvoiceDate'],

format='%d-%m-%Y %H:%M')

retail\_df.head()

# compute the max date

max\_date = max(retail\_df['InvoiceDate'])

max\_date

# compute the diff

retail\_df['diff'] = max\_date - retail\_df['InvoiceDate']

retail\_df.head()

# recency

last\_purchase = retail\_df.groupby('CustomerID')['diff'].min()

last\_purchase = last\_purchase.reset\_index()

last\_purchase.head()

# merge

grouped\_df = pd.merge(grouped\_df, last\_purchase, on='CustomerID', how='inner')

grouped\_df.columns = ['CustomerID', 'amount', 'frequency', 'recency']

grouped\_df.head()

# number of days only

grouped\_df['recency'] = grouped\_df['recency'].dt.days

grouped\_df.head()

# 1. outlier treatment

plt.boxplot(grouped\_df['recency'])

# two types of outliers:

# - statistical

# - domain specific

# removing (statistical) outliers

Q1 = grouped\_df.amount.quantile(0.05)

Q3 = grouped\_df.amount.quantile(0.95)

IQR = Q3 - Q1

grouped\_df = grouped\_df[(grouped\_df.amount >= Q1 - 1.5\*IQR) & (grouped\_df.amount <= Q3 + 1.5\*IQR)]

# outlier treatment for recency

Q1 = grouped\_df.recency.quantile(0.05)

Q3 = grouped\_df.recency.quantile(0.95)

IQR = Q3 - Q1

grouped\_df = grouped\_df[(grouped\_df.recency >= Q1 - 1.5\*IQR) & (grouped\_df.recency <= Q3 + 1.5\*IQR)]

# outlier treatment for frequency

Q1 = grouped\_df.frequency.quantile(0.05)

Q3 = grouped\_df.frequency.quantile(0.95)

IQR = Q3 - Q1

grouped\_df = grouped\_df[(grouped\_df.frequency >= Q1 - 1.5\*IQR) & (grouped\_df.frequency <= Q3 + 1.5\*IQR)]

# 2. rescaling

rfm\_df = grouped\_df[['amount', 'frequency', 'recency']]

# instantiate

scaler = StandardScaler()

# fit\_transform

rfm\_df\_scaled = scaler.fit\_transform(rfm\_df)

rfm\_df\_scaled.shape

rfm\_df\_scaled = pd.DataFrame(rfm\_df\_scaled)

rfm\_df\_scaled.columns = ['amount', 'frequency', 'recency']

rfm\_df\_scaled.head()

4. Modelling

# k-means with some arbitrary k

kmeans = KMeans(n\_clusters=4, max\_iter=50)

kmeans.fit(rfm\_df\_scaled)

kmeans.labels\_

# assign the label

grouped\_df['cluster\_id'] = kmeans.labels\_

grouped\_df.head()

# plot

sns.boxplot(x='cluster\_id', y='amount', data=grouped\_df)

* 1. **Operating Procedure**
* Open Jupyter note book
* Take a new python file
* Type the code
* Run it
* Take inputs from user
* Observe the results
* Verify the results manually
* Store the note book file
  1. **Precautions and/or Troubleshooting**

**Precautions:**

* Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).
* Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."
* Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.
* Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.
* Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.
* Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.
* Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

**Troubleshooting:**

* Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
* Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
* Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
* Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
* Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
* Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.
  1. **Observations**

Observe the results obtained in each operation.

* 1. **Calculations & Analysis**

Calculations should be given for each operation.

* 1. **Result & Interpretation**

Result should be printed and pasted in laboratory copy found from Jupyter note book.

* 1. **Follow-up Questions**
* What is K means Clustering Algorithm?
* Is Feature Scaling required for the K means Algorithm?
* Why do you prefer Euclidean distance over Manhattan distance in the K means Algorithm?
* Does centroid initialization affect K means Algorithm?
* What are the advantages and disadvantages of the K means Algorithm?
* How to decide the optimal number of K in the K means Algorithm?
* What are the possible stopping conditions in the K means Algorithm?
* What is the effect of the number of variables on the K means Algorithm?
  1. **Extension and Follow-up Activities (if applicable)**

NA

* 1. **Assessments**
  2. **Suggested reading**

NA