# **Implementing K-means Clustering**

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

dataset = pd.read\_csv('/content/Mall\_Customers (1).csv')

x = dataset.iloc[:, [3, 4]].values

from sklearn.cluster import KMeans

wcss\_list=[]

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Read the dataset

dataset = pd.read\_csv('/content/Mall\_Customers (1).csv')

# Extract the relevant features

x = dataset.iloc[:, [3, 4]].values

# Initialize an empty list to store the WCSS values

wcss\_list = []

# Iterate over a range of cluster numbers

for i in range(1, 11):

    # Create a KMeans object with the specified number of clusters

    kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

    # Fit the KMeans object to the data

    kmeans.fit(x)

    # Append the WCSS value to the list

    wcss\_list.append(kmeans.inertia\_)

# Plot the Elbow Method graph

plt.plot(range(1, 11), wcss\_list)

plt.title('The Elbow Method Graph')

plt.xlabel('Number of clusters (k)')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans (n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1], s=100, c='blue', label='0')

mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1], s=100, c='green', label='1')

mtp.scatter(x[y\_predict == 2, 0], x[y\_predict == 2, 1], s=100, c='red', label='2')

mtp.scatter(x[y\_predict == 4, 0], x[y\_predict == 4, 1], s=100, c='magenta', label='4')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

**(2)Implementing Hierarchical Clustering**

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

dataset = pd.read\_csv('/content/Mall\_Customers (1).csv')

x = dataset.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as shc

dendro=shc.dendrogram (shc.linkage(x, method="ward"))

mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()

from sklearn.cluster import AgglomerativeClustering

hc=AgglomerativeClustering (n\_clusters=5, affinity='euclidean', linkage='ward')

y\_pred= hc.fit\_predict(x)#if in conda use metric\_pred= hc.fit\_predict(x)

mtp.scatter (x[y\_pred==0,0], x[y\_pred==0,1], s=100, c='blue', label='Cluster1')

mtp.scatter(x[y\_pred==1,0], x[y\_pred==1,1], s=100, c='green', label='Cluster2')

mtp.scatter(x[y\_pred==2,0], x[y\_pred==2,1],s=100, c='red', label='Cluster3')

mtp.scatter(x[y\_pred==3,0], x[y\_pred==3,1], s=100, c='cyan', label='Cluster4')

mtp.scatter(x[y\_pred==4,0], x [y\_pred==4,1], s=100, c='magenta', label='Cluster5')

mtp.title('Clusters of Customers')

mtp.xlabel('Annual Income(k$)')

mtp.ylabel('Spending Score(1-100)')

mtp.legend()

mtp.show()

**(3)Implementation of Apriori Algorithm**.

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

dataset = pd.read\_csv('/content/Aprior.csv')

dataset

dataset.shape

transactions=[]

for i in range(0, 9):

  transactions.append([str(dataset.values[i,j]) for j in range(0,5)])

from apyori import apriori

rules = list(apriori(transactions=transactions, min\_support=0.003, min\_confidence=0.2))

results=list(rules)

results

for item in results:

    pair = item[0]

    if len(pair) > 1:

        items = [x for x in pair]

        print("Rule:" + items[0] + "->" + items[1])

    else:

        print("Rule:" + str(pair))

    print("Support:" + str(item[1]))

    print("Confidence:" + str(item[2][0][2]))

    print("Lift:" + str(item[2][0][3]))

    print("================ ===============")

print(len(results))

**Implementation of Market Basket Analysis.**

import pandas as pd

from mlxtend.frequent\_patterns import fpgrowth

from mlxtend.frequent\_patterns import association\_rules

data = pd.read\_csv('Bakery.csv')

print(data)

encoded\_data = pd.get\_dummies(data)

print(encoded\_data)

data.dropna(inplace=True)

encoded\_data = pd.get\_dummies(data, dtype=bool)

encoded\_data = encoded\_data.astype(bool)

frequent\_itemsets = fpgrowth (encoded\_data, min\_support=0.01, use\_colnames=True)

print(frequent\_itemsets)

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1)

print(rules.sort\_values('lift', ascending=False).head(10))

**(5)Reinforcement Learninga. Calculating Reward b. Discounted Reward c. Calculating Optimal quantities d. Implementing Q Learning e. Setting up an Optimal Action**

import random

from typing import List

import random

from typing import List

class SampleEnvironment:

    def \_\_init\_\_(self):

        self.steps\_left = 20

    def get\_observation(self) -> List[float]:

        return [0.0, 0.0, 0.0]

    def get\_action(self) -> List[int]:

        return [0, 1]

    def is\_done(self) -> bool:

        return self.steps\_left == 0

    def action(self, action: int) -> float:

        if self.is\_done():

            raise Exception("Game is over")

        self.steps\_left -= 1

        return random.random()

class Agent:

    def \_\_init\_\_(self):

        self.total\_reward = 0.0

    def step(self, env: SampleEnvironment):

        current\_obs = env.get\_observation()

        print("Observation {}".format(current\_obs))

        actions = env.get\_action()

        print("Action {}".format(actions))

        reward = env.action(random.choice(actions))

        self.total\_reward += reward

        print("Reward {}".format(self.total\_reward))

if \_\_name\_\_ == "\_\_main\_\_":

    env = SampleEnvironment()

    agent = Agent()

    i = 0

    while not env.is\_done():

        i += 1

        print("step {}".format(i))

        agent.step(env)

    print("Total reward got:%.4f" % agent.total\_reward)

**6. Time Series Analysisa. Checking Stationary b. Converting a non-stationary data to stationary c. Implementing Dickey Fuller Test d. Plot ACF and PACF e. Generating the ARIMA plot f. TSA Forecasting**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

import seaborn as sns

import warnings

import pandas as pd

import warnings

warnings.filterwarnings("ignore")

# Load the Air Passengers dataset

series = pd.read\_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv', header=0, index\_col=0)

series.head()

def check\_stationary (series):

    """

    This function checks the stationarity of a time series using the Augmented Dickey-Fuller test.

    Args:

        series: A Pandas Series object containing the time series data.

    Returns:

        None

    """

    result = adfuller(series)

    print('ADF Statistic: %f' % result[0])

    print('p-value: %f' % result[1])

    print('Critical Values:')

    for key, value in result[4].items():

        print('\t%s: %.3f' % (key, value))

    print('\t%s: %.3f' % (key, value))

check\_stationary (series)

def make\_stationary (series):

    """

    This function makes a time series stationary by differencing it and dropping the missing values.

    Args:

        series: A Pandas Series object containing the time series data.

    Returns:

        A Pandas Series object containing the stationary time series data.

    """

    stationary\_series = series.diff().dropna()

    return stationary\_series

stationary\_series = make\_stationary (series)

check\_stationary (stationary\_series)

plot\_acf(stationary\_series)

plot\_pacf(stationary\_series)

plt.show()

#Plot the original time series:

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

plt.plot(series.index, series ["Passengers"])

plt.title("Airline Passengers")

plt.xlabel("Year")

plt.ylabel("Passengers")

plt.show()

**7. Boosting a. Cross Validation b. AdaBoost**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

import matplotlib.pyplot as plt

iris = load\_iris()

print(iris)

X, y = iris.data, iris.target

# Import necessary modules

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

# Load the Iris dataset

iris = load\_iris()

# Extract data and target variables

X, y = iris.data, iris.target

# Check if X is defined

if 'X' not in globals():

    raise ValueError("Variable 'X' is not defined. Please load the dataset first.")

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

clf **=** aBoostClassifier(n\_estimators**=**50,learning\_rate**=**1,random\_state**=**42)

cv\_scores = cross\_val\_score(clf, X\_train, y\_train, cv=5)

print("Cross-Validation Score:", cv\_scores)

print("Mean Cross-Validation Score", cv\_scores.mean())

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Test set accuracy:", accuracy)

n\_estimators\_values = [10, 50, 100, 500, 1000, 5000]

cv\_scores\_mean = []

for n\_estimators in n\_estimators\_values:

    clf = AdaBoostClassifier(n\_estimators= n\_estimators, learning\_rate=1, random\_state=

                               0)

    cv\_scores = cross\_val\_score(clf, X\_train, y\_train, cv = 5)

    cv\_scores\_mean.append(cv\_scores.mean())

plt.plot(n\_estimators\_values, cv\_scores\_mean, marker='o')

plt.xlabel('Number of Trees')

plt.ylabel('Mean Cross-Validation Score')

plt.title("AdaBoost Performance with Different Numbers of Trees")

plt.show()

learning\_rates = np.linspace (0.1, 2, 20)

cv\_scores\_mean = []

for learning\_rate in learning\_rates:

    clf = AdaBoostClassifier(n\_estimators=50, learning\_rate=learning\_rate, random\_state=0)

    cv\_scores = cross\_val\_score (clf, X\_train, y\_train, cv=5)

    cv\_scores\_mean.append(cv\_scores.mean())

plt.plot(learning\_rates, cv\_scores\_mean, marker='o')

plt.xlabel('Learning Rate')

plt.ylabel('Mean Cross-Validation Score')

plt.title('AdaBoost Performance with Different Learning Rates')

plt.show()