Hindi Vidya Prachar Samiti's

Ramniranjan Jhunjhunwala College of Arts, Science & Commerce

(Empowered Autonomous College)

Affiliated to
UNIVERSITY OF MUMBAI

DEPARTMENT OF INFORMATION TECHNOLOGY
2025 - 2026

M.Sc. (IT) PART 2 - SEM IV

RJSPIT302P – Machine Learning

Name: Vishwakarma Shivam Suresh Sushila

Roll No: 6709

Hindi Vidya Prachar Samiti's

Ramniranjan Jhunjhunwala College of Arts, Science & Commerce

(Empowered Autonomous College)

Certificate

This is to certify that Mr. Vishwakarma Shivam Suresh Suhila Roll No 6709 of M.Sc.(I.T.) Part-1 class has completed the required number of experiments in the subject of Machine Learning in the Department of Information Technology during the academic year 2025 - 2026 .

Professor In-Charge
Prof. Bharati Bhole

Co-ordinator of IT Department
Prof. Bharati Bhole

College Seal & Date

Examiner

INDEX

Practical 1: Solving a Business Problem

Jul 14, 2025

Identifying Business Problems:
The first step in any data-driven project is identifying the business problem. A business problem can range from improving customer retention, forecasting sales, optimizing supply chains, to enhancing product recommendations. Proper identification is crucial, as it lays the foundation for building data models and choosing the right methodologies.

Key steps to identify business problems:

- Stakeholder Consultation: Understand the pain points from those directly impacted by the problem, like business leaders or customers.
- Defining Objectives: Clearly articulate the objective, whether it's reducing operational costs, increasing revenue, or improving customer satisfaction.
- Understanding Constraints: Identify any limitations, such as data availability, budget, timeline, or resource constraints.
- Contextual Analysis: Analyze the broader context of the business, industry trends, and competitive positioning to ensure the solution aligns with long-term goals.
- Defining Alternative Models: Once a problem is defined, it is essential to explore multiple models to ensure that the chosen solution is effective. Data scientists must develop and test various models to determine which one best meets the business objectives.

Steps to defining alternative models:

- Data Collection: Gather data relevant to the problem, ensuring it's clean, complete, and representative.
- Model Hypothesis Development: Based on the problem, develop hypotheses and decide whether a predictive, descriptive, or prescriptive model is appropriate.
- Model Selection: Common models include:
  ➔ Supervised Learning Models (e.g., regression, decision trees) for prediction.
  ➔ Unsupervised Learning Models (e.g., clustering, association) for patternrecognition.
- Optimization Models for resource management or scheduling.
- Model Evaluation: Compare multiple models using key metrics (accuracy, precision, recall) to identify the most effective one.
- Risk and Sensitivity Analysis: Analyze the robustness of models under different conditions to select the one best suited for real-world scenarios.
  By exploring alternative models, businesses can avoid committing to a single approach prematurely and instead make an informed decision on the model that most effectively addresses their needs.

Feature selection is the process of selecting the most relevant variables (features) from a dataset that contribute significantly to the target variable output). The goal is to improve model performance by reducing dimensionality, avoiding overfitting, and enhancing interpretability. Feature selection reduces noise and computational complexity, leading to faster and more accurate models.

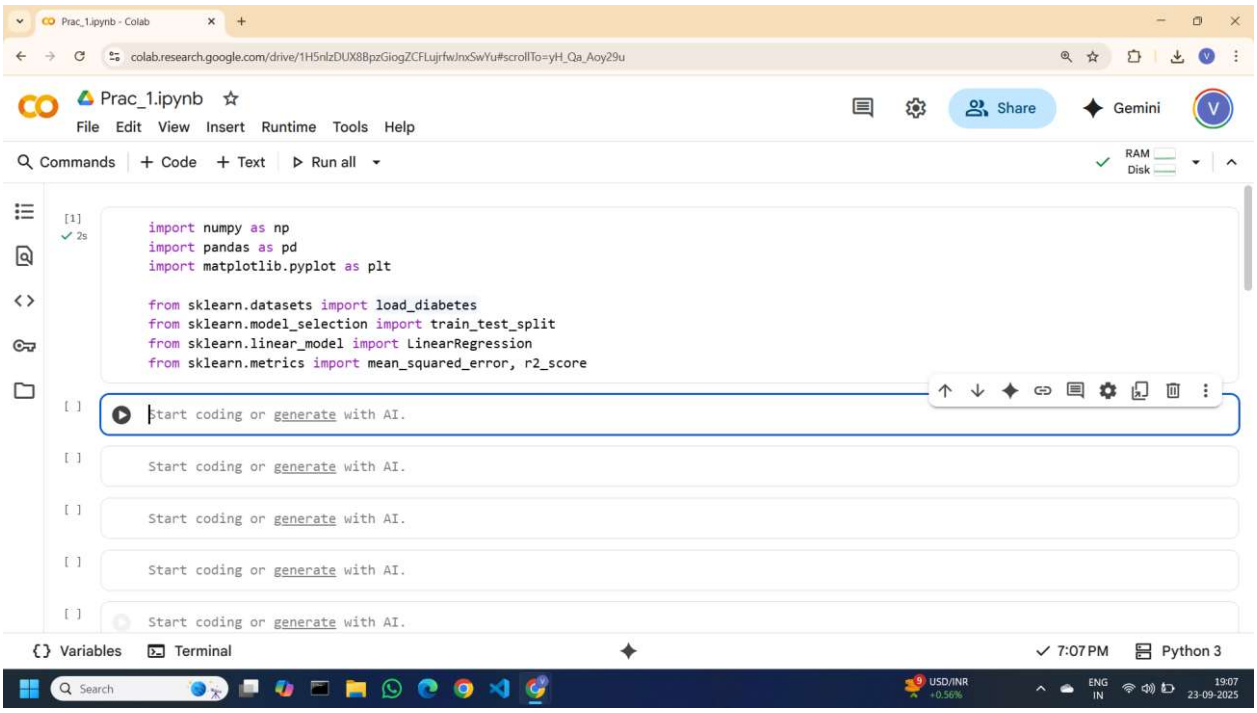
Feature engineering involves transforming raw data into meaningful features that improve model performance. It's the art of creating new features or modifying existing ones to better capture underlying patterns and relationships within the data.

Aim A: Identifying a Business Problem and Defining the Problem Statement.
Problem Statement:

- Diabetes is a chronic disease that affects millions of people worldwide. Early detection of diabetes can help in managing and preventing severe complications. The goal is to develop a predictive model that can accurately classify individuals as diabetic or non-diabetic based on several health metrics such as glucose levels, BMI, age, and others.
- The task involves using machine learning techniques to select the most relevant features that contribute to predicting whether an individual has diabetes. Proper feature selection can improve model performance, reduce overfitting, and enhance the interpretability of the model.

Step 1: Import Libraries

We first imported the necessary libraries like NumPy, Pandas, and Matplotlib for data handling and visualization. From scikit-learn, we imported the diabetes dataset, the Linear Regression model, and evaluation metrics.



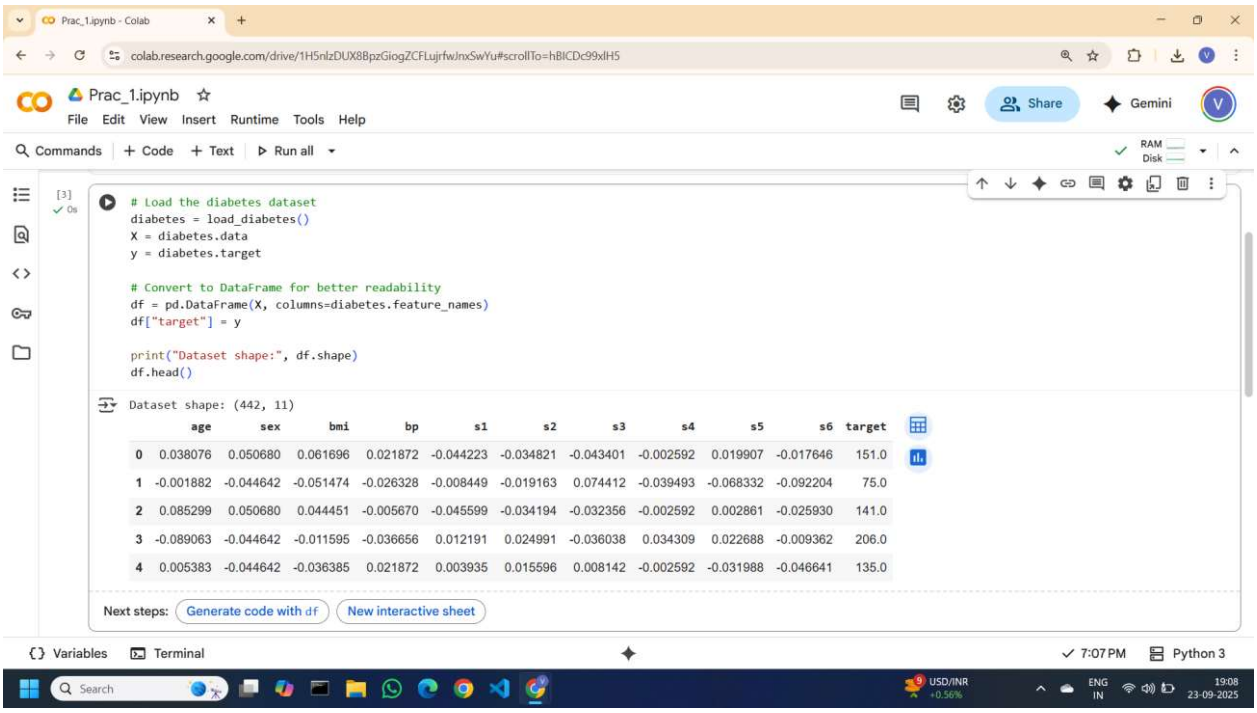
Step 2: Load Dataset

We used the Diabetes dataset provided by sklearn.datasets.

The dataset contains information about patients, including medical features (like age, BMI, blood pressure, etc.).

The target variable is a measure of disease progression (a numerical value we want to predict).

We stored the data in a DataFrame for easy visualization.



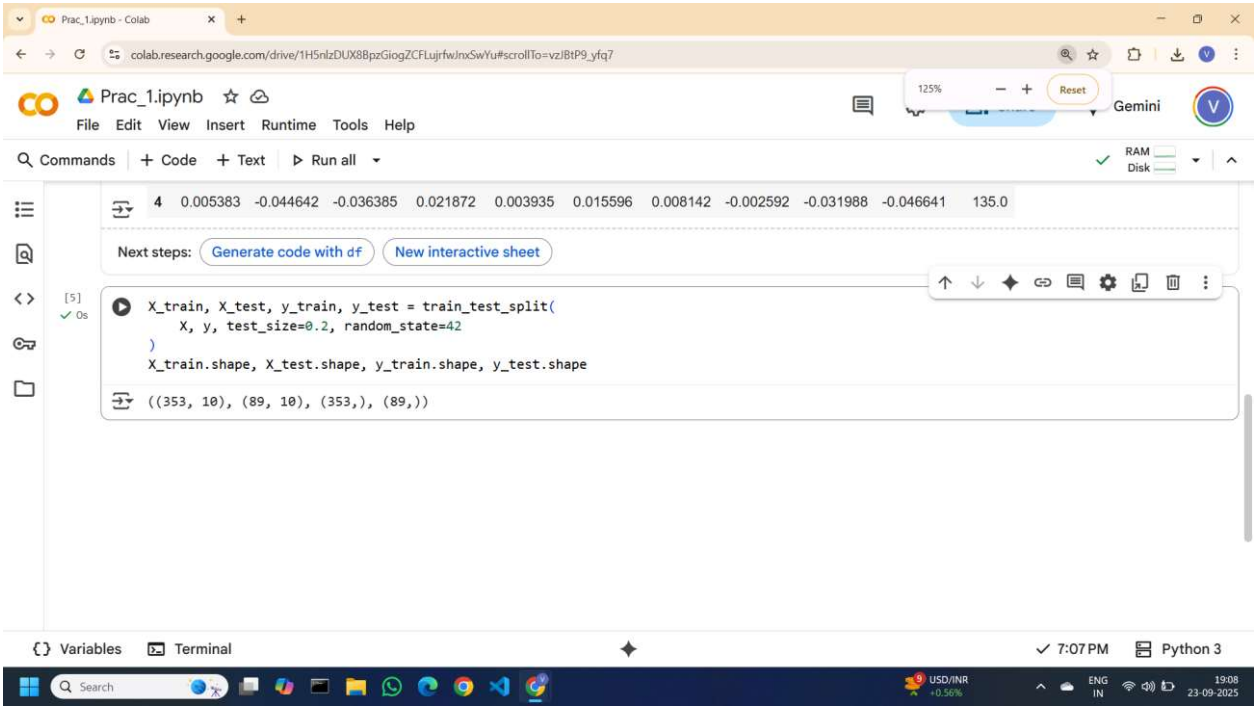
Step 3: Split into Training and Testing Sets

To build and test our model fairly, we split the dataset into:

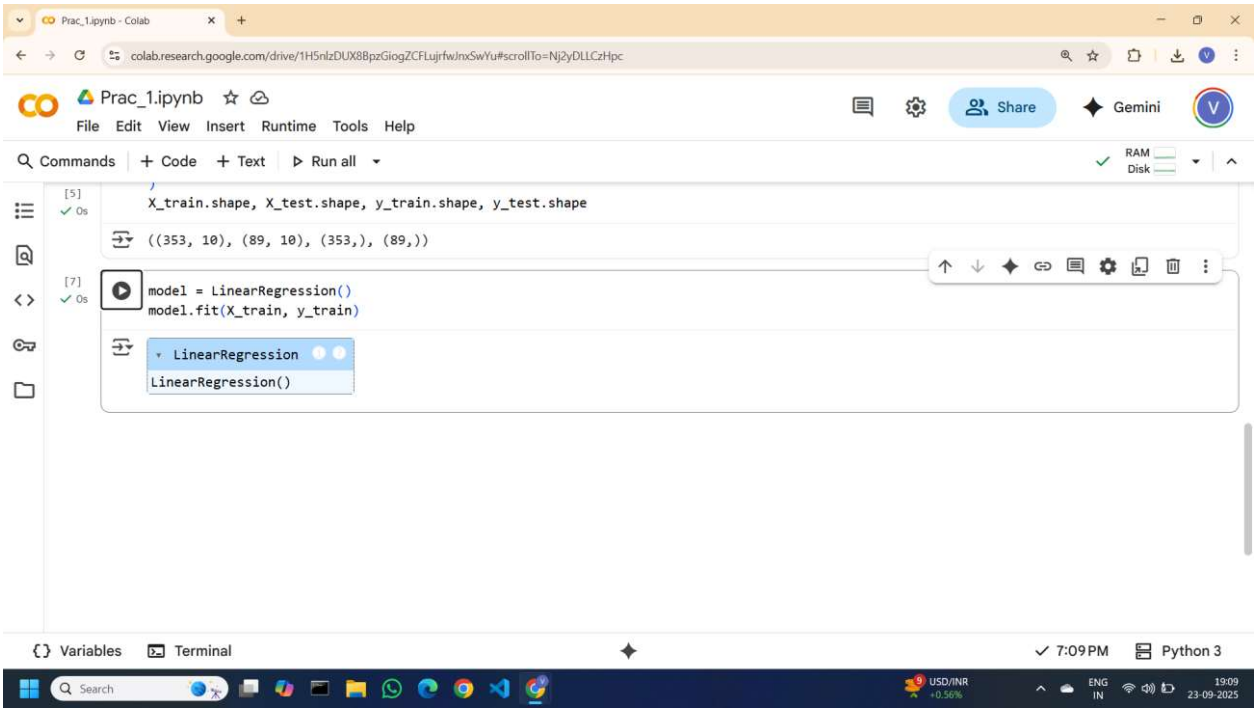
Training data (80%) → used to train the model.

Testing data (20%) → used to evaluate how well the model performs on unseen data.

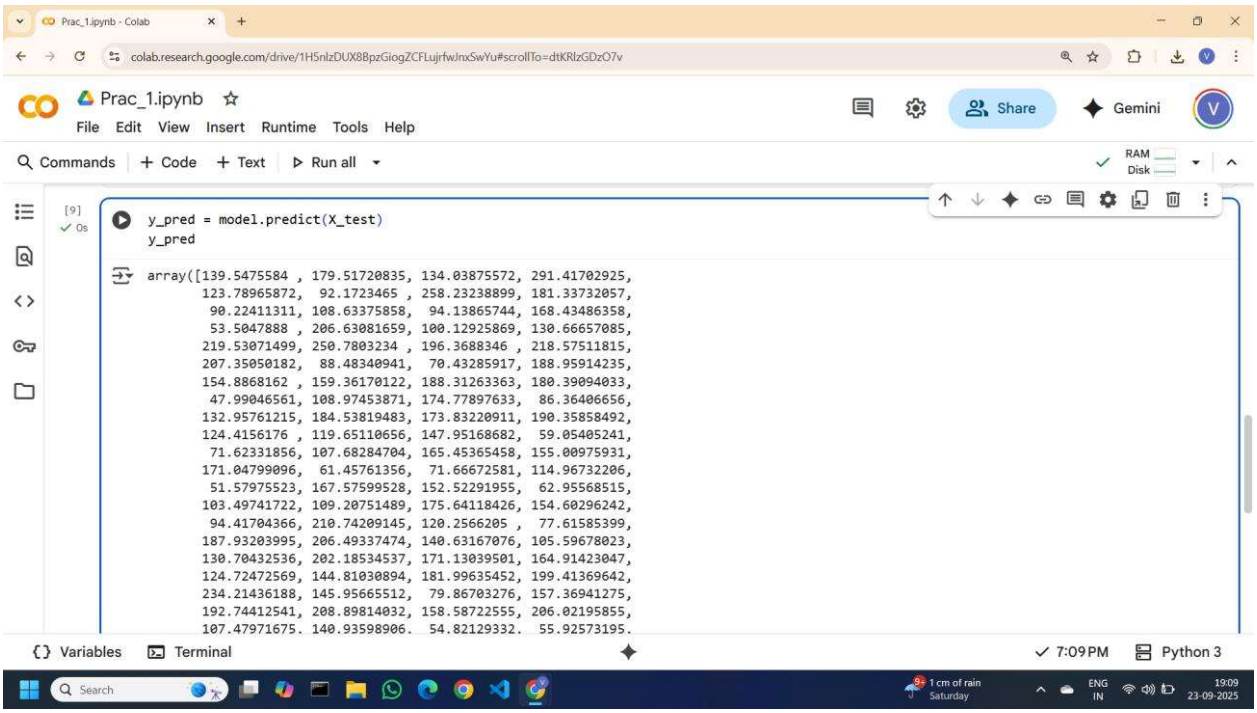
This was done using train_test_split.



Step 4: Create and Train Linear Regression Model



Step 5: Make Predictions

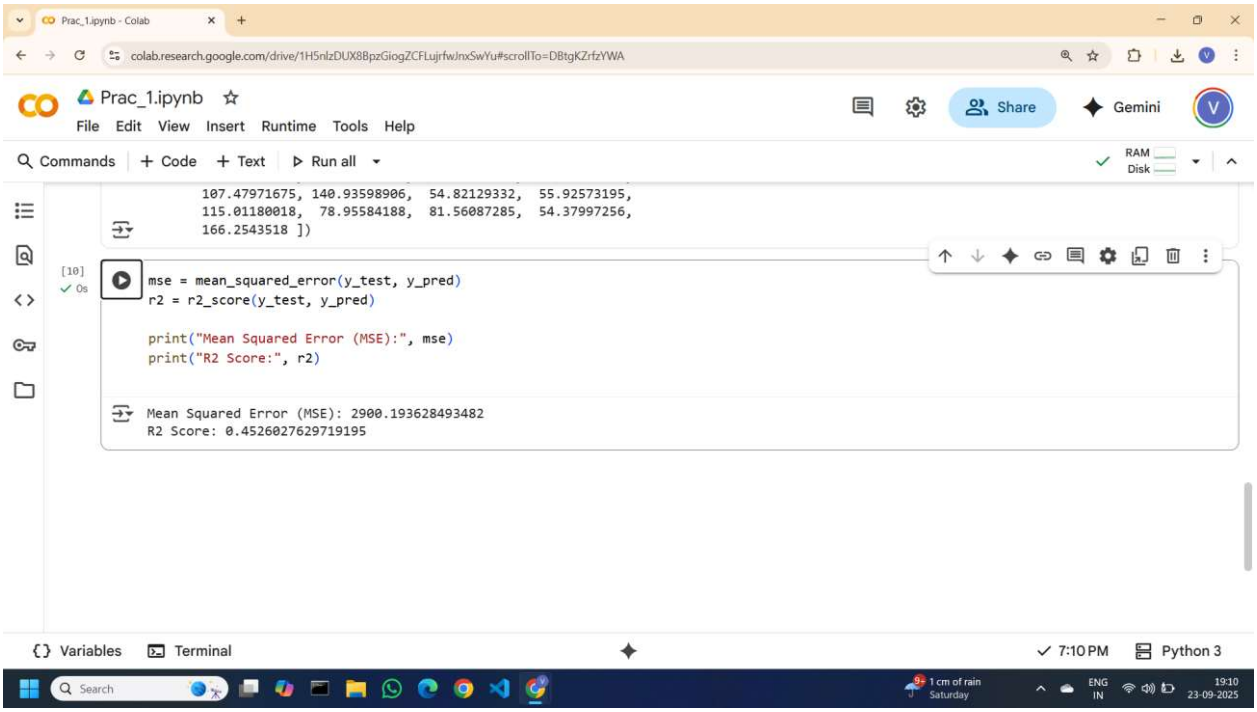


Step 6: Evaluate the Model

We measured the model's performance using two metrics:

Mean Squared Error (MSE): Tells us how far the predicted values are from the actual values (lower is better).

R^2 Score: Explains how well the model fits the data. An R^2 close to 1 means a good fit.



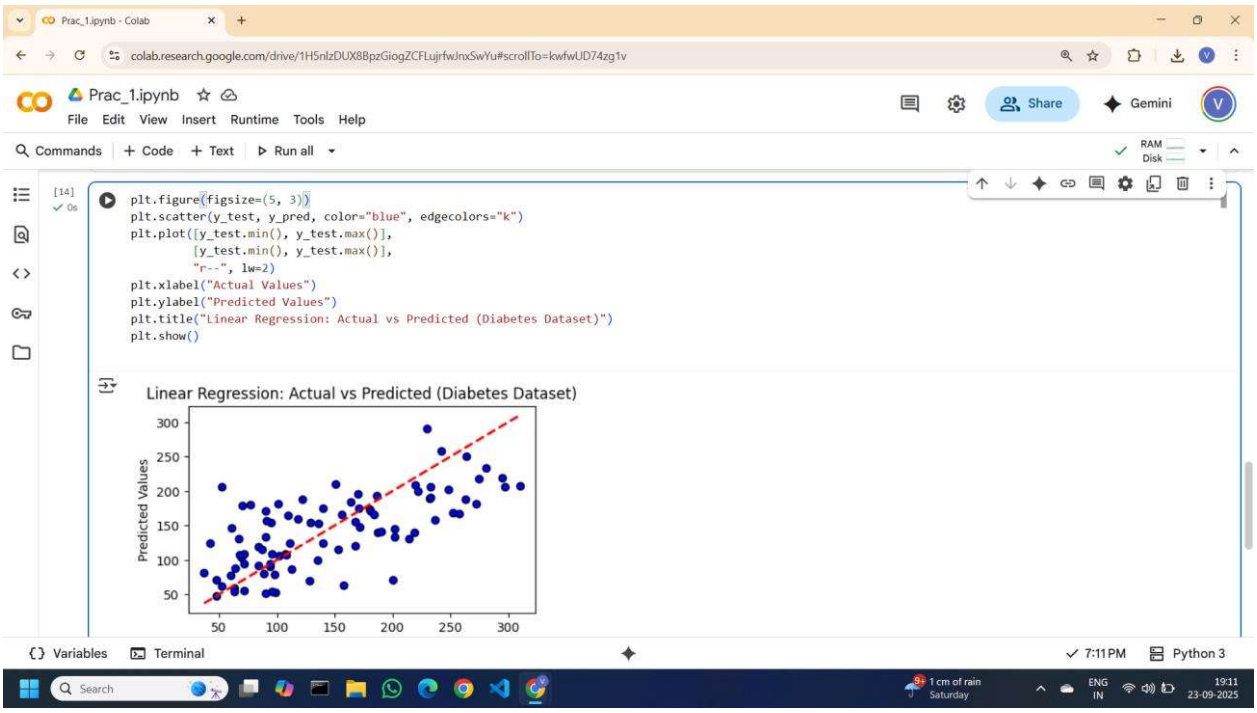
Step 7: Plot Actual vs Predicted Values

We created a scatter plot of actual vs predicted values.

Each blue dot represents a prediction compared to the actual outcome.

The red dashed line shows the ideal scenario (where predicted = actual).

The closer the points are to the red line, the better the model is performing



Aim B: Python Program to Demonstrate Feature Selection Methods.
Dataset: piama_indians_diabetes.csv
Link: [Pima Indians Diabetes Database]

1. Univariate Selection

```
# Feature Selection with Univariate Statistical Tests (ANOVA F-test)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest, f_classif

# Load dataset
names = ['Pregnancies','Glucose','BloodPressure','SkinThickness', 'Insulin','BMI','DiabetesPedigree',
dataframe = read_csv('/content/sample_data/Prac_1_diabetes.csv', names=names, header=0)

X = dataframe.iloc[:, 0:8] # Separate features
Y = dataframe.iloc[:, 8]   # Separate target

# Apply SelectKBest with ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=4)
fit = selector.fit(X, Y)

# Print feature scores with names
set_printoptions(precision=3)
scores = list(zip(X.columns, fit.scores_))
print("Feature ranking (highest = more relevant):")
for name, score in sorted(scores, key=lambda x: x[1], reverse=True):
    print(f"{name}: {score:.3f}")

# Show which features were selected
selected_cols = X.columns[fit.get_support()]
print("\nSelected top 4 features:\n",selected_cols.tolist())

# Reduced dataset with top k features
selected_features = fit.transform(X)
print("\nFirst 5 rows with selected features:\n", selected_features[0:5, :])
```

Output

```
Feature ranking (highest = more relevant):
Glucose: 213.162
BMI: 71.772
Age: 46.141
Pregnancies: 39.670
DiabetesPedigreeFunction: 23.871
Insulin: 13.281
SkinThickness: 4.304
BloodPressure: 3.257

Selected top 4 features:
 ['Pregnancies', 'Glucose', 'BMI', 'Age']

First 5 rows with selected features:
 [[  6.  148.   33.6  50. ]
 [  1.   85.   26.6  31. ]
 [  8.  183.   23.3  32. ]
 [  1.   89.   28.1  21. ]
 [  0.  137.   43.1  33. ]]
```

2. Recursive Feature Selection

```
# Feature Extraction with RFE (Recursive Feature Elimination)
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Load dataset
columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness',
           'Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']
dataframe = read_csv('/content/sample_data/Prac_1_diabetes.csv', names=columns, header=0)

# Split into features and target
X = dataframe.iloc[:, 0:8]
Y = dataframe.iloc[:, 8]

# Create model
model = LogisticRegression(solver='lbfgs', max_iter=1000)

# RFE to select top 3 features
rfe = RFE(model, n_features_to_select=3)
fit = rfe.fit(X, Y)

# Print results
print("Num Features Selected: %d" % fit.n_features_)
print("Selected Features Mask: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)

# Show selected features by name
selected_features = X.columns[fit.support_]
print("\nTop 3 Selected Features:\n", selected_features.tolist())
```

Output

```
Num Features Selected: 3
Selected Features Mask: [ True False False False False  True  True False]
Feature Ranking: [1 2 4 6 5 1 1 3]

Top 3 Selected Features:
 ['Pregnancies', 'BMI', 'DiabetesPedigreeFunction']
```

3. Principal Component Analysis

```
# Feature Extraction with PCA (Principal Component Analysis)
import numpy as np
from pandas import read_csv
from sklearn.decomposition import PCA

# Load dataset
columns = ['Pregnancies','Glucose','BloodPressure','SkinThickness',
           'Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']
dataframe = read_csv('/content/sample_data/Prac_1_diabetes.csv', names=columns, header=0)

# Split into features and target
X = dataframe.iloc[:, 0:8]
Y = dataframe.iloc[:, 8]
# Apply PCA - reduce to 3 principal components
pca = PCA(n_components=3)
fit = pca.fit(X)

# Summarize results
print("Explained Variance Ratio (per component):")
print(fit.explained_variance_ratio_)

print("\nPrincipal Components (eigenvectors):")
print(fit.components_)
```

Output

```
Explained Variance Ratio (per component):
[0.889 0.062 0.026]

Principal Components (eigenvectors):
[[-2.022e-03  9.781e-02  1.609e-02  6.076e-02  9.931e-01  1.401e-02
   5.372e-04 -3.565e-03]
 [ 2.265e-02  9.722e-01  1.419e-01 -5.786e-02 -9.463e-02  4.697e-02
   8.168e-04  1.402e-01]
 [ 2.246e-02 -1.434e-01  9.225e-01  3.070e-01 -2.098e-02  1.324e-01
   6.400e-04  1.255e-01]]
```

4. Feature importance

```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier

# Correct way: let pandas read headers from file
dataframe = read_csv("/content/sample_data/Prac_1_diabetes.csv")

# Split features and target
X = dataframe.iloc[:, 0:8]
Y = dataframe.iloc[:, 8]

# Train Extra Trees Classifier
model = ExtraTreesClassifier(n_estimators=100, random_state=42)
model.fit(X, Y)

# Show feature importance
print("Feature Importances:")
for name, score in zip(X.columns, model.feature_importances_):
    print(f"{name}: {score:.4f}")
```

Output

```
Feature Importances:
Pregnancies: 0.1100
Glucose: 0.2407
BloodPressure: 0.0972
SkinThickness: 0.0791
Insulin: 0.0714
BMI: 0.1354
DiabetesPedigreeFunction: 0.1232
Age: 0.1431
```

Colab file link: Prac_1.ipynb

Practical 2: Handling Numerical, Categorical, Textual and Date-Time data

Jul 21, 2025 & Jul 28, 2025

A. Handling Numerical Data

Rescaling Features Using MinMax Scalar

1. Use scikit-learn's MinMaxScaler to rescale a feature array in the range of 0 to 1 and also in the range of -1 to 1.
Array: -200, -100, 0, 100, 200

Code

```
# Load libraries
import numpy as np
from sklearn import preprocessing

# Create feature
feature = np.array([
    [-200],
    [-100],
    [0],
    [100],
    [200]
])

# Create scaler
minmax_scale1 = preprocessing.MinMaxScaler(feature_range=(0, 1)) # range 0 to 1
minmax_scale2 = preprocessing.MinMaxScaler(feature_range=(-1, 1)) # range -1 to 1

# Scale feature
scaled_feature1 = minmax_scale1.fit_transform(feature)
scaled_feature2 = minmax_scale2.fit_transform(feature)

# Show feature
scaled_feature1, scaled_feature2
```

```
(array([[0.  ],
        [0.25],
        [0.5 ],
        [0.75],
        [1.  ]]),
 array([[-1. ],
        [-0.5],
        [ 0. ],
        [ 0.5],
        [ 1. ]]))
```

2. Use scikit-learn's MinMaxScaler to rescale the following features.
Feature_1: 10,20,30,40,50
Feature_2: 500,1000,3000,25000,100

```
import pandas as pd
from sklearn import preprocessing

data = {
   "feature1": [10, 20, 30, 40, 50],
   "feature2": [500, 1000, 3000, 25000, 100]
}

df = pd.DataFrame(data)

print(f"DataFrame:\n{df}")

scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

scaled_df = pd.DataFrame(scaled_data, columns=df.columns)

print(f"\nScaled DataFrame:\n{scaled_df}")
```

```
DataFrame:
   feature1  feature2
0        10       500
1        20      1000
2        30      3000
3        40     25000
4        50       100

Scaled DataFrame:
   feature1  feature2
0      0.00  0.016064
1      0.25  0.036145
2      0.50  0.116466
3      0.75  1.000000
4      1.00  0.000000
```

Normalizing Features using StandardScalar

Ex 1: Use scikit-learn's StandardScalar to normalize a feature array with mean 0 and standard deviation 1.
Array: -200, -100, 0, 100, 200

```
import numpy as np
from sklearn import preprocessing

x = np.array([
   [-200],
   [-100],
   [0],
   [100],
   [200]
])

 # Create scaler
scaler = preprocessing.StandardScaler()

# Transform the feature
standardized = scaler.fit_transform(x)

# Show feature
standardized
```

```
array([[-1.41421356],
       [-0.70710678],
       [ 0.        ],
       [ 0.70710678],
       [ 1.41421356]])
```

Multiple feature

```
# Multiple features
import numpy as np
import pandas as pd
from sklearn import preprocessing

data = {
   'Feature_1': [10, 20, 30, 40, 50],
   'Feature_2': [500, 1000, 3000, 25000, 100]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
display(df)

scaler = preprocessing.StandardScaler()
standardized_data = scaler.fit_transform(df)

standardized_df = pd.DataFrame(standardized_data, columns=df.columns)
print("\nStandardized DataFrame:")
display(standardized_df)
```

Original DataFrame:

	Feature_1	Feature_2
0	10	500
1	20	1000
2	30	3000
3	40	25000
4	50	100

Standardized DataFrame:

	Feature_1	Feature_2
0	-1.414214	-0.565057
1	-0.707107	-0.512930
2	0.000000	-0.304422
3	0.707107	1.989167
4	1.414214	-0.606758

Normalizing Features using RobustScalar

Ex 1: Use scikit-learn's RobustScalar to normalize a feature array.

Array: -200, -100, 0, 10000, 200

```
# Load libraries
import numpy as np
from sklearn import preprocessing

# Create feature
x = np.array([
    [-200],
    [-100],
    [0],
    [10000],
    [200]
])

# Create scaler
robust_scaler = preprocessing.RobustScaler()

# Transform feature
robust_scaled_feature = robust_scaler.fit_transform(x)
```

```
# Show feature
display(robust_scaled_feature)
```

```
array([[-0.66666667],
       [-0.33333333],
       [ 0.        ],
       [33.33333333],
       [ 0.66666667]])
```

Normalizing Features using Normalizer

Ex 2: Use scikit-learn's Normalizer to normalize the following features with l1, l2 and max norm.
Feature_1: 10,20,30,40,50
Feature_2: 500,1000,3000,25000,100

```
# Load libraries
import numpy as np
import pandas as pd
from sklearn import preprocessing

# Create features
data = {
   'Feature_1': [10, 20, 30, 40, 50],
   'Feature_2': [500, 1000, 3000, 25000, 100]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
display(df)

# Create normalizer
normalizer_l1 = preprocessing.Normalizer(norm='l1')
normalizer_l2 = preprocessing.Normalizer(norm='l2')
normalizer_max = preprocessing.Normalizer(norm='max')

# Transform features with different norms
normalized_data_l1 = normalizer_l1.transform(df)
normalized_data_l2 = normalizer_l2.transform(df)
normalized_data_max = normalizer_max.transform(df)
```

```
# Convert back to DataFrames for better visualization
normalized_df_l1 = pd.DataFrame(normalized_data_l1, columns=df.columns)
normalized_df_l2 = pd.DataFrame(normalized_data_l2, columns=df.columns)
normalized_df_max = pd.DataFrame(normalized_data_max, columns=df.columns)

# Show normalized features
print("\nNormalized DataFrame (l1 norm):")
display(normalized_df_l1)

print("\nNormalized DataFrame (l2 norm):")
display(normalized_df_l2)

print("\nNormalized DataFrame (max norm):")
display(normalized_df_max)
```

```
Original DataFrame:
```

	Feature_1	Feature_2
0	10	500
1	20	1000
2	30	3000
3	40	25000
4	50	100

	Feature_1	Feature_2
0	0.019608	0.980392
1	0.019608	0.980392
2	0.009901	0.990099
3	0.001597	0.998403
4	0.333333	0.666667

Normalized DataFrame (l2 norm):

	Feature_1	Feature_2
0	0.019996	0.999800
1	0.019996	0.999800
2	0.010000	0.999950
3	0.001600	0.999999
4	0.447214	0.894427

Normalized DataFrame (max norm):

	Feature_1	Feature_2
0	0.0200	1.0
1	0.0200	1.0
2	0.0100	1.0
3	0.0016	1.0
4	0.5000	1.0

Grouping Values using Clustering

Ex 2: Use scikit-learn's Kmeans algorithm to group the following features into 2 groups.
Feature_1: 10,20,30,40,50,10,20,30
Feature_2: 500,1000,3000,25000,100,500,200,3000

```
import pandas as pd
from sklearn.cluster import KMeans

# Create features
data = {
   'feature_1': [10, 20, 30, 40, 50, 10, 20, 30],
   'feature_2': [500, 1000, 3000, 25000, 100, 500, 200, 3000]
```

```
}

df = pd.DataFrame(data)
# Make k-means clusterer
clusterer = KMeans(n_clusters=2, random_state=0, n_init=10)

# Fit clusterer
clusterer.fit(df) # Fit to the DataFrame

# Predict values
df["group"] = clusterer.predict(df) # Predict on the DataFrame

# View first few observations
display(df.head())
```

	feature_1	feature_2	group
0	10	500	0
1	20	1000	0
2	30	3000	0
3	40	25000	1
4	50	100	0

Deleting Missing Values

Ex1: Delete the observation with null values from the following data.

```
features = np.array([
    [1.1, 11.1],
    [2.2, 22.2],
    [np.nan, 22.2],
    [3.3, 33.3],
    [np.nan, 55]
])

import numpy as np
```

```
import pandas as pd

# Create feature array with null values
features = np.array([
    [1.1, 11.1],
    [2.2, 22.2],
    [np.nan, 22.2],
    [3.3, 33.3],
    [np.nan, 55]
])

# Convert to DataFrame
df = pd.DataFrame(features, columns=['feature_1', 'feature_2'])

print("Original DataFrame:")
display(df)

# Delete observations with null values
df_cleaned = df.dropna()

print("\nDataFrame after dropping null values:")
display(df_cleaned)
```

```
Original DataFrame:
```

	feature_1	feature_2
0	1.1	11.1
1	2.2	22.2
2	NaN	22.2
3	3.3	33.3
4	NaN	55.0

```
DataFrame after dropping null values:
```

	feature_1	feature_2
0	1.1	11.1
1	2.2	22.2
3	3.3	33.3

Imputing the Missing Values

Ex1: Replace the observation with null values with the mean value for the following data.

features = np.array([
[1.1, 11.1],
[2.2, 22.2],
[np.nan, 22.2],
[3.3, 33.3],
[np.nan, 55]
[10, np.nan]
])

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

# Create feature array with missing values (corrected the last row)
features = np.array([
   [1.1, 11.1],
   [2.2, 22.2],
   [np.nan, 22.2],
   [3.3, 33.3],
   [np.nan, 55.0],
   [10.0, np.nan]
])

# Convert to DataFrame for better display
df = pd.DataFrame(features, columns=['feature_1', 'feature_2'])

print("Original DataFrame:")
display(df)

# Impute missing values with the mean
mean_imputer = SimpleImputer(strategy='mean')
features_mean_imputed = mean_imputer.fit_transform(features)

# Impute missing values with the median
median_imputer = SimpleImputer(strategy='median')
features_median_imputed = median_imputer.fit_transform(features)
```

```
# Impute missing values with the most frequent value
most_frequent_imputer = SimpleImputer(strategy='most_frequent')
features_most_frequent_imputed = most_frequent_imputer.fit_transform(features)

# Convert imputed arrays back to DataFrames for display
df_mean_imputed = pd.DataFrame(features_mean_imputed, columns=df.columns)
df_median_imputed = pd.DataFrame(features_median_imputed, columns=df.columns)
df_most_frequent_imputed = pd.DataFrame(features_most_frequent_imputed,
columns=df.columns)

print("\nDataFrame after imputing with mean:")
display(df_mean_imputed)

print("\nDataFrame after imputing with median:")
display(df_median_imputed)

print("\nDataFrame after imputing with most frequent:")
display(df_most_frequent_imputed)
```

```
Original DataFrame:
```

	feature_1	feature_2
0	1.1	11.1
1	2.2	22.2
2	NaN	22.2
3	3.3	33.3
4	NaN	55.0
5	10.0	NaN

```
DataFrame after imputing with mean:
```

	feature_1	feature_2
0	1.10	11.10
1	2.20	22.20
2	4.15	22.20
3	3.30	33.30
4	4.15	55.00
5	10.00	28.76

DataFrame after imputing with median:

	feature_1	feature_2
0	1.10	11.1
1	2.20	22.2
2	2.75	22.2
3	3.30	33.3
4	2.75	55.0
5	10.00	22.2

DataFrame after imputing with most frequent:

	feature_1	feature_2
0	1.1	11.1
1	2.2	22.2
2	1.1	22.2
3	3.3	33.3
4	1.1	55.0
5	10.0	22.2

B. Categorical Data Handling

Encoding Nominal Categorical Features using LabelBinerizer

Ex1: Encode the employee's department (Nominal) feature of following data using scikit-learn's LabelBinerizer.

Feature_1: 1,2,3,4,5

Feature_2: "Employee1","Employee2","Employee3","Employee4","Employee5"

Feature_3: "Grade1","Grade2","Grade3","Grade1","Grade2",

Feature_4: "IT Dept", "CS Dept","DS Dept","IT Dept", "CS Dept"

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer

# Create DataFrame for features
data = {
   'Feature_1': [1, 2, 3, 4, 5],
   'Feature_2': ["Employee1", "Employee2", "Employee3", "Employee4", "Employee5"],
   'Feature_3': ["Grade1", "Grade2", "Grade3", "Grade1", "Grade2"],
   'Feature_4': ["IT Dept", "CS Dept", "DS Dept", "IT Dept", "CS Dept"]
}
df = pd.DataFrame(data)
```

```
print("Original DataFrame:")
display(df)

# Create LabelBinarizer
lb = LabelBinarizer()

# Fit and transform the 'Feature_4' column
dept_encoded = lb.fit_transform(df['Feature_4'])

# Convert the encoded feature back to a DataFrame for better visualization
encoded_df = pd.DataFrame(dept_encoded, columns=lb.classes_)

print("\nEncoded 'Feature_4' using LabelBinarizer:")
display(encoded_df)
```

Original DataFrame:

	Feature_1	Feature_2	Feature_3	Feature_4
0	1	Employee1	Grade1	IT Dept
1	2	Employee2	Grade2	CS Dept
2	3	Employee3	Grade3	DS Dept
3	4	Employee4	Grade1	IT Dept
4	5	Employee5	Grade2	CS Dept

Encoded 'Feature_4' using LabelBinarizer:

	CS Dept	DS Dept	IT Dept
0	0	0	1
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Ex 2: Encode the employee's Manager and department feature of following data using scikit-learn's MultiLabelBinerizer.

EmpID: 1,2,3,4,5
Name: "Employee1","Employee2","Employee3","Employee4","Employee5"
Grade: "Grade1","Grade2","Grade3","Grade1","Grade2",
Department: "IT Dept", "CS Dept","DS Dept","IT Dept", "CS Dept"
Manager: "Employee1", "Employee2", "Employee1","Employee2", "Employee1"

```
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer

data = {
   'EmpID': [1, 2, 3, 4, 5],
   'Name': ["Employee1", "Employee2", "Employee3", "Employee4", "Employee5"],
```

```
    'Grade': ["Grade1", "Grade2", "Grade3", "Grade1", "Grade2"],
    'Department': [["IT Dept"], ["CS Dept"], ["DS Dept"], ["IT Dept"], ["CS Dept"]], #
MultiLabelBinarizer expects lists of labels
    'Manager': [["Employee1"], ["Employee2"], ["Employee1"], ["Employee2"], ["Employee1"]]
# MultiLabelBinarizer expects lists of labels
}
df_multi = pd.DataFrame(data)
print("Original DataFrame:")
display(df_multi)

# Create MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# Fit and transform the 'Department' and 'Manager' columns
# We need to combine the lists of labels for MultiLabelBinarizer
combined_labels = df_multi['Department'] + df_multi['Manager']
encoded_features = mlb.fit_transform(combined_labels)

# Convert the encoded features back to a DataFrame
encoded_df_multi = pd.DataFrame(encoded_features, columns=mlb.classes_)

print("\nEncoded 'Department' and 'Manager' using MultiLabelBinarizer:")
display(encoded_df_multi)
```

Original DataFrame:

	EmpID	Name	Grade	Department	Manager
0	1	Employee1	Grade1	[IT Dept]	[Employee1]
1	2	Employee2	Grade2	[CS Dept]	[Employee2]
2	3	Employee3	Grade3	[DS Dept]	[Employee1]
3	4	Employee4	Grade1	[IT Dept]	[Employee2]
4	5	Employee5	Grade2	[CS Dept]	[Employee1]

Encoded 'Department' and 'Manager' using MultiLabelBinarizer:

	CS Dept	DS Dept	Employee1	Employee2	IT Dept
0	0	0	1	0	1
1	1	0	0	1	0
2	0	1	1	0	0
3	0	0	0	1	1
4	1	0	1	0	0

Encoding Ordinal Categorical Features

Ex1: Encode the employee's grade and Level feature of following data using scikit-learn's MultiLabelBinerizer.

EmpID: 1,2,3,4,5

Name: "Employee1","Employee2","Employee3","Employee4","Employee5"

Grade: "Grade1","Grade2","Grade3","Grade1","Grade2",

Department: "IT Dept", "CS Dept","DS Dept","IT Dept", "CS Dept"

Level: "Junior","Senior","Class1","Junior","Senior"

```
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer

data = {
   'EmpID': [1, 2, 3, 4, 5],
   'Name': ["Employee1", "Employee2", "Employee3", "Employee4", "Employee5"],
   'Grade': [["Grade1"], ["Grade2"], ["Grade3"], ["Grade1"], ["Grade2"]], # MultiLabelBinarizer expects lists of labels
   'Department': [["IT Dept"], ["CS Dept"], ["DS Dept"], ["IT Dept"], ["CS Dept"]],
   'Level': [["Junior"], ["Senior"], ["Class1"], ["Junior"], ["Senior"]] # MultiLabelBinarizer expects lists of labels
}
```

```
df_ordinal = pd.DataFrame(data)
print("Original DataFrame:")
display(df_ordinal)

# Create MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# Fit and transform the 'Grade' and 'Level' columns
# Combine the lists of labels for MultiLabelBinarizer
combined_ordinal_labels = df_ordinal['Grade'] + df_ordinal['Level']
encoded_ordinal_features = mlb.fit_transform(combined_ordinal_labels)

# Convert the encoded features back to a DataFrame
encoded_df = pd.DataFrame(encoded_ordinal_features, columns=mlb.classes_)

print("\nEncoded 'Grade' and 'Level' using MultiLabelBinarizer:")
display(encoded_df)
```

```
Original DataFrame:
```

	EmpID	Name	Grade	Department	Level
0	1	Employee1	[Grade1]	[IT Dept]	[Junior]
1	2	Employee2	[Grade2]	[CS Dept]	[Senior]
2	3	Employee3	[Grade3]	[DS Dept]	[Class1]
3	4	Employee4	[Grade1]	[IT Dept]	[Junior]
4	5	Employee5	[Grade2]	[CS Dept]	[Senior]

```
Encoded 'Grade' and 'Level' using MultiLabelBinarizer:
```

	Class1	Grade1	Grade2	Grade3	Junior	Senior
0	0	1	0	0	1	0
1	0	0	1	0	0	1
2	1	0	0	1	0	0
3	0	1	0	0	1	0
4	0	0	1	0	0	1

Encode Dictionaries of Features

Ex1: Encode the following Dictionaries of features and display the features and feature names.

{“Red”:122, “Green”:130,”Blue”:100}
{“Green”:135,”Blue”:200}
{“Red”:132, “Green”:230,”Blue”:100}
{“Red”:152,”Blue”:150}
{“Red”:162, “Green”:140,”Blue”:200}

```
from sklearn.feature_extraction import DictVectorizer

data_dict = [
    {"Red": 122, "Green": 130, "Blue": 100},
    {"Green": 135, "Blue": 200},
    {"Red": 132, "Green": 230, "Blue": 100},
    {"Red": 152, "Blue": 150},
    {"Red": 162, "Green": 140, "Blue": 200}
]

# Create DictVectorizer and Set sparse=False to get a dense NumPy array
dict_vectorizer = DictVectorizer(sparse=False)

# Encode dictionaries
encoded_data = dict_vectorizer.fit_transform(data_dict)

# Display features and feature names
print("Encoded Features:")
display(encoded_data)

print("\nFeature Names:")
display(dict_vectorizer.get_feature_names_out())
```

```
Encoded Features:
array([[100., 130., 122.],
       [200., 135.,   0.],
       [100., 230., 132.],
       [150.,   0., 152.],
       [200., 140., 162.]])

Feature Names:
array(['Blue', 'Green', 'Red'], dtype=object)
```

Imputing Missing Class Values

Ex1: Input the categorical values of the given dataset using the SimpleImputer class using 'most_frequent' or 'constant' strategy

Reference code

```
import pandas as pd
df = pd.DataFrame([["BSc", "IT"],
                   [np.nan, "IT"],
                   ["BSc", np.nan],
                   ["BCom", "CS"]], dtype="category")
imp = SimpleImputer(strategy="most_frequent")
print(imp.fit_transform(df))
```

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

# Create DataFrame with missing categorical values
df = pd.DataFrame([["BSc", "IT"],
                   [np.nan, "IT"],
                   ["BSc", np.nan],
                   ["BCom", "CS"]], dtype="category")

print("Original DataFrame:")
display(df)

# Create SimpleImputer with 'most_frequent' strategy
imputer_most_frequent = SimpleImputer(strategy="most_frequent")

# Impute missing values
imputed_data_most_frequent = imputer_most_frequent.fit_transform(df)

print("\nDataFrame after imputing with 'most_frequent' strategy:")
display(imputed_data_most_frequent)
```

```
Original DataFrame:
```

	0	1
0	BSc	IT
1	NaN	IT
2	BSc	NaN
3	BCom	CS

```
DataFrame after imputing with 'most_frequent' strategy:
array([['BSc', 'IT'],
       ['BSc', 'IT'],
       ['BSc', 'IT'],
       ['BCom', 'CS']], dtype=object)
```

C. Textual Data Handling

Cleaning Text

Q. 1. Copy the data from the Placement Cell page of RJ College website and clean the text. Also replace the short forms in the text.

```
[8] text_data = [
        "   Placement Cell in the R. J. College plays a vital role and is becoming a key department of the institute.   ",
        "   As there are large numbers of colleges coming up,   ",
        "   the competition for employment is increasing every day and the job of placement is becoming a challenging one.   "
        ]

    # Strip whitespaces
    strip_whitespace = [string.strip() for string in text_data]
    strip_whitespace
```

```
['Placement Cell in the R. J. College plays a vital role and is becoming a key department of the institute.',
 'As there are large numbers of colleges coming up,',
 'the competition for employment is increasing every day and the job of placement is becoming a challenging one.']
```

```
[9] # Remove periods
    remove_periods = [string.replace(".", "") for string in strip_whitespace]
    remove_periods
```

```
['Placement Cell in the R J College plays a vital role and is becoming a key department of the institute',
 'As there are large numbers of colleges coming up,',
 'the competition for employment is increasing every day and the job of placement is becoming a challenging one']
```

```
[13] # Create function to convert string to upper case
     def upper_case(string: str) -> str:
      return string.upper()

     # Apply function
     [upper_case(string) for string in remove_periods]
```

```
['PLACEMENT CELL IN THE R J COLLEGE PLAYS A VITAL ROLE AND IS BECOMING A KEY DEPARTMENT OF THE INSTITUTE',
 'AS THERE ARE LARGE NUMBERS OF COLLEGES COMING UP,',
 'THE COMPETITION FOR EMPLOYMENT IS INCREASING EVERY DAY AND THE JOB OF PLACEMENT IS BECOMING A CHALLENGING ONE']
```

```
# Create function to convert string to lower case
def lower_case(string: str) -> str:
 return string.lower()

# Apply function
[lower_case(string) for string in remove_periods]
```

```
['placement cell in the r j college plays a vital role and is becoming a key department of the institute',
 'as there are large numbers of colleges coming up,',
 'the competition for employment is increasing every day and the job of placement is becoming a challenging one']
```

```
[16] # Import library
import re
# Create function
def replace_letters_with_X(string: str) -> str:
 return re.sub(r"[a-zA-Z]", "X", string)

# Apply function
[replace_letters_with_X(string) for string in remove_periods]
```

```
['XXXXXXXXX XXXX XX XXX X X XXXXXXX XXXXX X XXXXX XXXX XXX XX XXXXXXXX X XXX XXXXXXXXXX XX XXX XXXXXXXXX',
 'XX XXXXX XXX XXXXX XXXXXXX XX XXXXXXXX XXXXXX XX,',
 'XXX XXXXXXXXXXX XXX XXXXXXXXXX XX XXXXXXXXXX XXXXX XXX XXX XXX XXX XX XXXXXXXXX XX XXXXXXXX X XXXXXXXXXXX XXX']
```

Parsing and Cleaning HTML

Q. 2. Get the HTML data of About Us Page of R J College Website with the help of web scraping process and extract the details of contact.

```
import requests

url = "https://www.rjcollege.edu.in/contact-us/"
res = requests.get(url)

html_data  = res.content
html_data
```

```
</svg></span><span class="pojo-a11y-toolbar-text">Readable Font</span>\t\t\t\t\t\t\t\t</a></li><li class="pojo-a11y-toolba
link pojo-a11y-btn-reset" data-action="reset" tabindex="-1" role="button">\n<span class="pojo-a11y-toolbar-icon"><svg vers
width="1em" viewBox="0 0 448 448"><title>Reset</title><path fill="currentColor" d="M384 224c0 105.75-86.25 192-192 192-57.
0.5-10.75l34.25-34.5c1.75-1.5 4-2.25 6.25-2.25 2.25 0.25 4.5 1.25 5.75 3 24.5 31.75 61.25 49.75 101 49.75 70.5 0 128-57.5
87 34.25l34.25 34.5c4.75 4.5 6 11.5 3.5 17.25-2.5 6-8.25 10-14.75 10h-112c-8.75 0-16-7.25-16-16v-112c0-6.5 4-12.25 10-14.7
32.25c35.25-33.25 83-53 132.25-53 105.75 0 192 86.25 192 192z"></path></svg></span>\n<span class="pojo-a11y-toolbar-text">
<style></style><style></style><style></style> <script type="litespeed/javascript">var post_grid_vars={"siteUrl":"https:\\/
```

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(html_data)
```

```
[53] soup.find("div", {"class": "wpb_wrapper"}).text
```

```
'PRINCIPAL: Dr. Himanshu DawdaEMAIL ID: rjcollege@rjcollege.edu.inTEL NO.:Landline 1022 25151763Landline 2022 25152731Landline
SRamniranjan Jhunjhunwala College, Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai 400086, Maharashtra, INDIA.'
```

```
[54] [soup.find("div", {"class": "wpb_wrapper"}).find_all("p")]
```

```
[[<p><strong>PRINCIPAL: </strong><strong>Dr. Himanshu Dawda</strong></p>,
  <p><strong>EMAIL ID: </strong>rjcollege@rjcollege.edu.in</p>,
  <p><strong>TEL NO.:</strong></p>,
  <p> </p>,
  <p><strong>ADDRESS</strong></p>,
  <p>Ramniranjan Jhunjhunwala College, Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai 400086, Maharashtra, INDIA.</p>]]
```

```
[55] soup.find("div", {"class": "wpb_wrapper"}).find("table").text
```

```
'Landline 1022 25151763Landline 2022 25152731Landline 309867846817Landline 309867447849'
```

```
# Extract and print all text inside <p> tags
for p in soup.find_all("p"):
    print(p.text.strip())

# Extract and print all table data
for table in soup.find_all("table"):
    for row in table.find_all("tr"):
        # Extract all header or data cells in the row
        cells = row.find_all(["th", "td"])
        cell_texts = [cell.get_text(strip=True) for cell in cells]
        print("\t".join(cell_texts))  # Tab-separated for clarity
```

```
Hindi Vidya Prachar Samiti's RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE & COMMERCE  (EMPOWERED AUTONOMOUS)
Knowledge is all Ambrosia
PRINCIPAL: Dr. Himanshu Dawda
EMAIL ID: rjcollege@rjcollege.edu.in
TEL NO.:

ADDRESS
Ramniranjan Jhunjhunwala College, Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai 400086, Maharashtra, INDIA.

Accessibility Tools
Landline 1      022 25151763
Landline 2      022 25152731
Landline 3      09867846817
Landline 3      09867447849
```

Q. 3. Demonstrate or perform the following operations on the admissions data on the IT department page of RJ College.

a. Extract data of admissions in simple text format

```
# Import necessary libraries for web scraping and parsing
import requests
from bs4 import BeautifulSoup

# Define the URL of the IT department page
url = "https://www.rjcollege.edu.in/informationtechnology/#1556781569586-47b686e1-3048"
# Send a GET request to the URL
res = requests.get(url)

# Get the HTML content from the response
html_data  = res.content
# Create a BeautifulSoup object to parse the HTML
soup = BeautifulSoup(html_data)

# Display the first 1000 characters of the HTML data to verify
html_data[:1000]
```

```
b'<!DOCTYPE html><html lang="en-US"><head><script data-no-optimize="1">var
litespeed_docref=sessionStorage.getItem("litespeed_docref");litespeed_docref&&(Object.defineProperty(document,"referrer",{get:function()
litespeed_docref}}),sessionStorage.removeItem("litespeed_docref"));</script> <meta charset="UTF-8"><meta name="viewport" content="width=
width, initial-scale=1"><link rel="profile" href="http://gmpg.org/xfn/11"><link rel="pingback" href="https://www.rjcollege.edu.in/xmlrpc
<title>INFORMATION TECHNOLOGY &#8211; Hindi Vidya Prachar Samiti&#039;s RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE &amp; COMMERCE
(EMPOWERED AUTONOMOUS)</title>\n <script type="text/template" id="pvc-stats-view-template"><i class="pvc-stats-icon medium" aria-hidden="
<svg xmlns="http://www.w3.org/2000/svg" version="1.0" viewBox="0 0 502 315" preserveAspectRatio="xMidYMid meet"><g transform="translate(
scale(0.1,-0.1)" fill="" stroke="none"><path d="M2394 3279 l-29 -30 -3 -207'
```

```
[27] # Find all div elements with the class "vc_tta-panel-body"
data = soup.find_all("div", class_="vc_tta-panel-body")

# Select the 7th element (index 6) from the found divs, as it contains the admissions data
target_data = data[6]
# Extract the text content from the selected div
extracted_data = target_data.text
# Display the extracted text
print(extracted_data)
```

```
'MSc. IT: M.Sc. IT is a 2 years post graduate degree program. During these 2 years, students are given both theoretical as well as
wledge in the field of IT (Information Technology).Curriculum:\xa0 The MSc IT program consists of four semesters. In each semester
l study four subjects.This program is designed to fulfill the latest demand in the field of IT Industry. Major areas of focus would
puting, AWS, Digital Image Processing and Computer Vision, Artificial Intelligence, Data Analytics, Robotics, Computer Forensics,
g\xa0 etc.Eligibility:\xa0 Bachelors' degree in Information Technology/Computer Science/Physics/Stats/Electronics or equivalent exa
a recognized university.'
```

b. Clean data.

```
# Clean the extracted data
# Remove leading/trailing whitespace
cleaned_data = extracted_data.strip()

# Convert to lowercase
cleaned_data = cleaned_data.lower()

# Replace multiple spaces/newlines with a single space using regular expressions
import re
cleaned_data = re.sub(r'\s+', ' ', cleaned_data)

# Display the cleaned data
cleaned_data
```

```
'msc. it: m.sc. it is a 2 years post graduate degree program. during these 2 years, students are given both theoretical as well as
wledge in the field of it (information technology).curriculum: the msc it program consists of four semesters. in each semester, st
udy four subjects.this program is designed to fulfill the latest demand in the field of it industry. major areas of focus would be
ng, aws, digital image processing and computer vision, artificial intelligence, data analytics, robotics, computer forensics, ethi
c.eligibility: bachelors' degree in information technology/computer science/physics/stats/electronics or equivalent examination fr
d university.'
```

c. Remove all punctuations.

```
[42] # Import necessary libraries for handling Unicode data and system specifics
     import unicodedata
     import sys

     # Create a dictionary to map punctuation characters to None
     # This is used for efficient removal of punctuation using translate()
     punctuation = dict.fromkeys(i for i in range(sys.maxunicode) if unicodedata.category(chr(i)).startswith('P'))

     # Remove punctuation using the translate method on the cleaned_data string
     no_punct_text = cleaned_data.translate(punctuation)

     # Update cleaned_text to be the punctuation-removed and stripped string
     cleaned_text = no_punct_text.strip()

     # Display the text after removing punctuation
     cleaned_text
```

```
'msc it msc it is a 2 years post graduate degree program during these 2 years students are given both theoretical as well as practi
in the field of it information technologycurriculum the msc it program consists of four semesters in each semester students will st
ectsthis program is designed to fulfill the latest demand in the field of it industry major areas of focus would be cloud computing
image processing and computer vision artificial intelligence data analytics robotics computer forensics ethical hacking etceligibil
degree in information technologycomputer sciencephysicsstatselectronics or equivalent examination from a recognized university'
```

d. Tokenize text and remove all stop words.

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Tokenize the cleaned text into words
tokens = word_tokenize(cleaned_text.lower()) # Tokenize the already lowercased and punctuation removed text

# Get the set of English stop words
stop_words = set(stopwords.words('english'))

# Filter out stop words from the tokens
filtered_tokens = [word for word in tokens if word not in stop_words]
# Display the filtered tokens
filtered_tokens
```

```
'msc',
'2',
'years',
'post',
'graduate',
'degree',
'program',
'2',
'years',
'students',
'given',
'theoretical',
'well',
'practical',
'knowledge',
'field',
'information',
```

e. Stemming words.

```
from nltk.stem import PorterStemmer

# Create a PorterStemmer object
ps = PorterStemmer()
# Apply stemming to each word in the filtered tokens
stemmed_words = [ps.stem(word) for word in filtered_tokens]
# Display the stemmed words
stemmed_words
```

```
 'year',
 'student',
 'given',
 'theoret',
 'well',
 'practic',
 'knowledg',
 'field',
 'inform',
 'technologycurriculum',
 'msc',
 'program',
 'consist',
 'four',
 'semest',
 'semest',
 'student',
 'studi',
```

f. Display Part of speech tagging of each word.

```
import nltk
nltk.download('averaged_perceptron_tagger_eng')

# Perform part-of-speech tagging on the stemmed words
pos_tags = nltk.pos_tag(stemmed_words)
# Display the part-of-speech tags
pos_tags
```

```
[('msc', 'NN'),
 ('msc', 'NN'),
 ('2', 'CD'),
 ('year', 'NN'),
 ('post', 'NN'),
 ('graduat', 'NN'),
 ('degre', 'NN'),
 ('program', 'NN'),
 ('2', 'CD'),
 ('year', 'NN'),
 ('student', 'NN'),
 ('given', 'VBN'),
 ('theoret', 'RB'),
 ('well', 'RB'),
 ('practic', 'JJ'),
 ('knowledg', 'NNS'),
 ('field', 'NN'),
 ('inform', 'NN'),
 ('technologycurriculum', 'NN'),
 ('msc', 'NN'),
 ('program', 'NN'),
```

D. Date Time Data Handling

Q. Store the following data into a dataframe and display the datewise, timewise temperature and humidity as per the Indian and US date and time.
Given Data:
High & Low Weather Summary for the Past Weeks

	Temperature	Humidity	Pressure
High	31 °C (18 Jul, 14:30)	96% (20 Jul, 17:30)	1006 mbar (20 Jul, 17:30)
Low	25 °C (23 Jul, 05:30)	74% (18 Jul, 14:30)	998 mbar (25 Jul, 05:30)
Average	28 °C	88%	1003 mbar

```
import pandas as pd
from datetime import datetime
# Import the pytz library to handle timezones.
import pytz

# Define two timezone objects: Indian Standard Time (IST) and US Eastern Time (ET).
IST = pytz.timezone('Asia/Kolkata')
ET = pytz.timezone('US/Eastern')
```

```
[59] # Create a list of dictionaries, where each dictionary represents a row of data.
     # This includes 'Type', 'Temperature', 'Humidity', and datetime for both temperature and humidity in IST.
     data = [
         {
             'Type': 'High',
             'Temperature': 31,
             # Localize the datetime object to the IST timezone.
             'Temp_IST': IST.localize(datetime(2025, 7, 18, 14, 30)),
             'Humidity': 96,
             'Humidity_IST': IST.localize(datetime(2025, 7, 20, 17, 30))
         },
         {
             'Type': 'Low',
             'Temperature': 25,
             'Temp_IST': IST.localize(datetime(2025, 7, 23, 5, 30)),
             'Humidity': 74,
             'Humidity_IST': IST.localize(datetime(2025, 7, 18, 14, 30))
         },
         {
             'Type': 'Average',
             'Temperature': 28,
             # Use pandas.NaT (Not a Time) to represent a missing datetime value.
             'Temp_IST': pd.NaT,
             'Humidity': 88,
             'Humidity_IST': pd.NaT
         }
     ]

     # Create a pandas DataFrame from the list of dictionaries.
     df = pd.DataFrame(data)
     df
```

	Type	Temperature	Temp_IST	Humidity	Humidity_IST
0	High	31	2025-07-18 14:30:00+05:30	96	2025-07-20 17:30:00+05:30
1	Low	25	2025-07-23 05:30:00+05:30	74	2025-07-18 14:30:00+05:30
2	Average	28	NaT	88	NaT

```
# Convert the datetime values in 'Temp_IST' and 'Humidity_IST' columns to the US Eastern Timezone (ET).
# .apply() is used here to iterate over each element in the column.
# The lambda function checks if the value is not NaT (missing) before converting the timezone.
df['Temp_US'] = df['Temp_IST'].apply(lambda x: x.tz_convert(ET) if pd.notnull(x) else pd.NaT)
df['Humidity_US'] = df['Humidity_IST'].apply(lambda x: x.tz_convert(ET) if pd.notnull(x) else pd.NaT)

# Display the resulting DataFrame.
# display(df)
display(df[:2])
```

	Type	Temperature	Temp_IST	Humidity	Humidity_IST	Temp_US	Humidity_US
0	High	31	2025-07-18 14:30:00+05:30	96	2025-07-20 17:30:00+05:30	2025-07-18 05:00:00-04:00	2025-07-20 08:00:00-04:00
1	Low	25	2025-07-23 05:30:00+05:30	74	2025-07-18 14:30:00+05:30	2025-07-22 20:00:00-04:00	2025-07-18 05:00:00-04:00

Colab file link: Prac_2.ipynb

Practical 3: Linear Regression

Aug 4, 2025

A. Fitting a Linear Relationship and Multiple linear regression

1. Simple Linear Relationship

```
# import pandas as pd
# df = pd.read_csv('/content/DATA.csv')
# print(df)

from sklearn.linear_model import LinearRegression
import numpy as nd

x=nd.array([2024,2025]).reshape(-1, 1)
y=nd.array([11500,52350])
# x, y = x.reshape(-1, 1)
regression = LinearRegression()

model = regression.fit(x, y)
model.predict([[2026]])
```

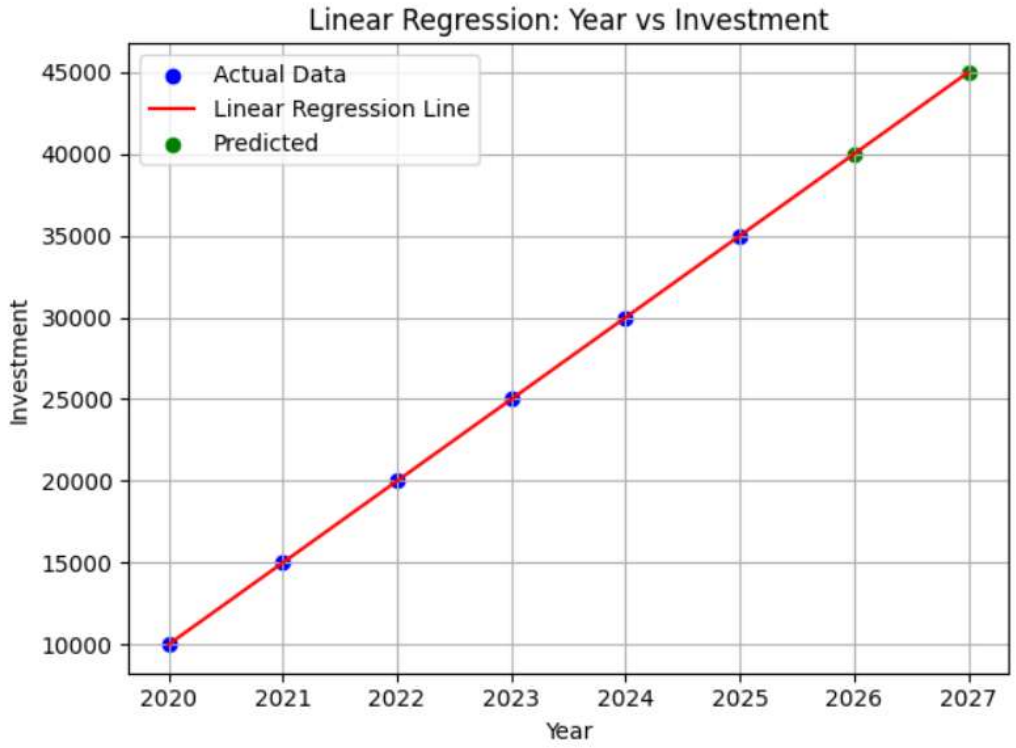
```
array([93200.])
```

2. Linear Relationship with graph plotting

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

# Sample linear dataset
data = {
    'Year': [2020, 2021, 2022, 2023, 2024, 2025],
    'Investment': [10000, 15000, 20000, 25000, 30000, 35000]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Independent (X) and Dependent (y) variables
x = df[['Year']]          # 2D array
y = df['Investment']      # 1D array

# Create and train the linear regression model
model = LinearRegression()
model.fit(x, y)

# Predict for future year(s)
future_years = np.array([2026, 2027]).reshape(-1, 1)
future_predictions = model.predict(future_years)

# Print predictions
for year, pred in zip(future_years.flatten(), future_predictions):
    print(f"Predicted Investment in {year}: {pred:.2f}")
```

```
# Plot the original data
plt.scatter(x, y, color='blue', label='Actual Data')

# Plot the regression line
x_line = np.linspace(2020, 2027, 100).reshape(-1, 1)
y_line = model.predict(x_line)
plt.plot(x_line, y_line, color='red', label='Linear Regression Line')

# Highlight future predictions
plt.scatter(future_years, future_predictions, color='green', label='Predicted')

# Labels and formatting
plt.xlabel("Year")
plt.ylabel("Investment")
plt.title("Linear Regression: Year vs Investment")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Predicted Investment in 2026: 40000.00
Predicted Investment in 2027: 45000.00
```

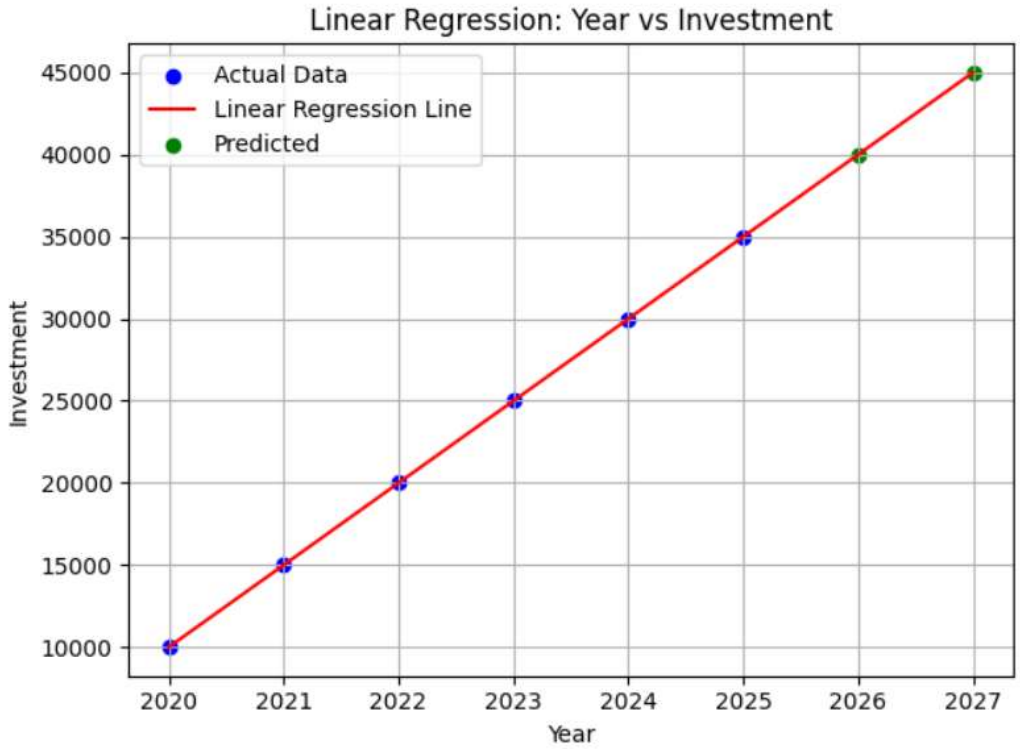


B. Fitting a Nonlinear Relationship

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
import matplotlib.pyplot as plt

# Sample nonlinear dataset
data = {
    'Year': [2015, 2016, 2017, 2018, 2019, 2020],
    'Investment': [5000, 7000, 13000, 21000, 31000, 45000]  # Increases faster = nonlinear
}
df = pd.DataFrame(data)

# Independent and dependent variables
x = df[['Year']]
y = df['Investment']

# Convert x to polynomial features (degree 2 for quadratic)
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)  # adds x^2 term

# Fit linear regression on the transformed data
model = LinearRegression()
model.fit(x_poly, y)
```

```
# Predict for future year(s)
future_years = np.array([2026, 2027]).reshape(-1, 1)
future_predictions = model.predict(future_years)

# Print predictions
for year, pred in zip(future_years.flatten(), future_predictions):
    print(f"Predicted Investment in {year}: {pred:.2f}")

# Plot the original data
plt.scatter(x, y, color='blue', label='Actual Data')

# Plot the regression line
x_line = np.linspace(2020, 2027, 100).reshape(-1, 1)
y_line = model.predict(x_line)
plt.plot(x_line, y_line, color='red', label='Linear Regression Line')

# Highlight future predictions
plt.scatter(future_years, future_predictions, color='green', label='Predicted')

# Labels and formatting
plt.xlabel("Year")
plt.ylabel("Investment")
plt.title("Linear Regression: Year vs Investment")
plt.legend()
```

```
Predicted Investment in 2026: 40000.00
Predicted Investment in 2027: 45000.00
```



C. Reducing Variance with Regularization

```
# Import necessary libraries
import pandas as pd # Library for data manipulation and analysis (e.g., creating DataFrames)
import numpy as np # Library for numerical operations, especially with arrays
from sklearn.preprocessing import StandardScaler # For standardizing features (scaling data to h
from sklearn.model_selection import train_test_split # For splitting data into training and test
```

```
[7] # Load the dataset into a pandas DataFrame
    # The dataset contains information about housing in the Boston area
    df = pd.read_csv("/content/sample_data/Prac_3_Boston_HousingData.csv")
```

[8]
```
# Display the first and last 3 rows of the DataFrame to get a glimpse of the data structure and values
df.head(3), df.tail(3)
```

```
(      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
 0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296     15.3
 1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242     17.8
 2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242     17.8

         B  LSTAT  MEDV
 0  396.90   4.98  24.0
 1  396.90   9.14  21.6
 2  392.83   4.03  34.7  ,
         CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
 503  0.06076  0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273     21.0
 504  0.10959  0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273     21.0
 505  0.04741  0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273     21.0

           B  LSTAT  MEDV
 503  396.90   5.64  23.9
 504  393.45   6.48  22.0
 505  396.90   7.88  11.9  )
```

[9]
```
df.shape
```

```
(506, 14)
```

```
# Check for missing values in each column
# This is important for data cleaning and preprocessing
df.isnull().sum()
```

	0
CRIM	20
ZN	20
INDUS	20
CHAS	20
NOX	0
RM	0
AGE	20
DIS	0
RAD	0
TAX	0
PTRATIO	0
B	0
LSTAT	20
MEDV	0

```
# Dropping rows with missing values is one way to handle them, although imputation is another common method.
df = df.dropna()
# Display the shape after dropping rows to see how many were removed
df.shape
```

```
(394, 14)
```

```
[12] # Display descriptive statistics for the DataFrame
# This provides insights into the central tendency, dispersion, and shape of the dataset's distribution
display(df.describe())
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000	394.000000
mean	3.690136	11.460660	11.000863	0.068528	0.553215	6.280015	68.932741	3.805268	9.403553	406.431472	18.537563	358.490939	12.769112	22.359645
std	9.202423	23.954082	6.908364	0.252971	0.113112	0.697985	27.888705	2.098571	8.633451	168.312419	2.166460	89.283295	7.308430	9.142979
min	0.006320	0.000000	0.460000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	2.600000	1.730000	5.000000
25%	0.081955	0.000000	5.130000	0.000000	0.453000	5.879250	45.475000	2.110100	4.000000	280.250000	17.400000	376.707500	7.125000	16.800000
50%	0.268880	0.000000	8.560000	0.000000	0.538000	6.201500	77.700000	3.199200	5.000000	330.000000	19.100000	392.190000	11.300000	21.050000
75%	3.435973	12.500000	18.100000	0.000000	0.624000	6.605500	94.250000	5.116700	24.000000	666.000000	20.200000	396.900000	17.117500	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

```
# Separate features (X) and target (y) variables
# Features (X) are the independent variables used to predict the target
features = df.drop('MEDV', axis=1) # Features are all columns except 'MEDV' (Median value of owner-occupied homes)
# Target (y) is the dependent variable we want to predict
target = df['MEDV'] # Target is the 'MEDV' column

# Display the features and target DataFrames/Series
features.head(3), target.head(3)
```

```
(      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
 0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296     15.3
 1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242     17.8
 2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242     17.8

         B  LSTAT
 0  396.90   4.98
 1  396.90   9.14
 2  392.83   4.03  ,
 0    24.0
 1    21.6
 2    34.7
 Name: MEDV, dtype: float64)
```

```
# Initialize StandardScaler
scaler = StandardScaler()
# Standardize the features by removing the mean and scaling to unit variance
features_standardized = scaler.fit_transform(features)
# fit_transform calculates the mean and standard deviation of the training data and then transforms the data.

# Split the standardized data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features_standardized, target, test_size=0.2, random_state=42)
```

```
# Import necessary libraries for Ridge Regression and evaluation
from sklearn.linear_model import Ridge # For Ridge Regression model
from sklearn.metrics import mean_squared_error, r2_score # For evaluating model performance
```

```
# Initialize the Ridge Regression model with alpha=0.5 (regularization strength)
regression = Ridge(alpha=0.5)
# Train the model using the training data
# The model learns the relationship between the features (x_train) and the target (y_train).
model = regression.fit(x_train, y_train)

# Make predictions on the test set
predication = model.predict(x_test)
```

```
# Mean Squared Error (MSE) measures the average of the squared errors between the actual and predicted values.
# Lower MSE indicates a better fit.
mse = mean_squared_error(y_test, predication)
# R-squared (R2) score represents the proportion of the variance in the dependent variable that is predictable
# from the independent variables. A higher R2 indicates a better fit.
r2 = r2_score(y_test, predication)

# Print the evaluation metrics
print(f"Ridge Regression- MSE: {mse:2f} & R2: {r2:2f}")
```

```
Ridge Regression- MSE: 31.491818 & R2: 0.626637
```

```
# Display the standardized features of the 51st element in the test set and its corresponding prediction
# This shows an example of the input to the trained model and the model's output (prediction) for that input.
x_test[50], predication[50]
```

```
(array([-0.38101147, -0.47905119, -0.59291383, -0.27123695, -0.93136712,
        -0.70867344, -1.27930094,  0.6179341 , -0.74265727, -1.03172412,
        -0.29466235,  0.43074011,  0.18918482]),
 np.float64(21.55131745019286))
```

D. Reducing Features with Lasso Regression.

```
# Import necessary libraries for Lasso Regression
from sklearn.linear_model import Lasso # For Lasso Regression model
```

```
# Initialize the Lasso Regression model with alpha=0.5 (regularization strength)
lasso_regression = Lasso(alpha=0.5)
# Train the model using the training data
# The model learns the relationship between the features (x_train) and the target (y_train)
model = lasso_regression.fit(x_train, y_train)
```

```
[21] # Make predictions on the test set using the trained Lasso model
     lasso_predication = model.predict(x_test)
     # Display the predictions
     lasso_predication
```

```
array([25.56912192, 23.26225212, 22.11465773, 28.37045335, 17.84396765,
       32.64546156, 22.86868041, 31.43583418, 30.32674764, 15.0116637 ,
       22.67643471, 38.48168932, 27.62575408, 15.50123334, 18.34497004,
       27.13658873, 20.43938459, 20.8032398 , 20.96357594, 14.33517465,
       18.71449001, 20.19684571, 18.24088432, 29.9571628 , 23.83596357,
       16.80305809, 25.90959176, 28.05140546, 20.41975437, 25.71114569,
       37.62454883, 17.52284969, 20.48302455, 18.42605333, 16.31964044,
       21.94079439, 22.54554363, 22.73493079, 21.8591052 , 22.5754731 ,
       27.0956879 , 32.36228518, 21.16347065, 28.59950771, 34.93833315,
       19.27651401, 27.33452048, 10.26563441, 20.92127392, 26.70061791,
       20.8619414 , 23.91127324, 15.79873522, 18.7710549 , 17.28301917,
       24.17692568, 38.15140467, 20.34620408, 17.60475048, 21.99666115,
       20.16131246, 24.30495903, 17.09104856, 27.33675635, -1.04326594,
       31.69086846, 17.81131794, 28.89623412, 24.84369928, 22.60724804,
       31.21211865, 31.15265387, 16.68162652, 21.05662768, 18.57828792,
       34.84872478, 20.10464117, 30.02991653, 17.61857733])
```

```
# Mean Squared Error (MSE) measures the average of the squared errors between the actual and predicted values.
# Lower MSE indicates a better fit.
mse = mean_squared_error(y_test, lasso_predication) # Calculate Mean Squared Error (MSE)
# R-squared (R2) score represents the proportion of the variance in the dependent variable that is predictable
# from the independent variables. A higher R2 indicates a better fit.
r2 = r2_score(y_test, lasso_predication) # Calculate R-squared (R2) score

# Print the evaluation metrics for the Lasso Regression model
print(f"Lasso Regression- MSE: {mse:2f} & R2: {r2:2f}")
```

```
lasso Regression- MSE: 37.728586 & R2: 0.552695
```

```
coefficients = pd.Series(model.coef_, index=features.columns)
# Display the coefficients
display(coefficients)

# selecting feature which are having non zero values
selected_features = coefficients[coefficients != 0].index
print(selected_features.values)
```

	0
CRIM	-0.402041
ZN	0.000000
INDUS	-0.000000
CHAS	0.236836
NOX	-0.250207
RM	3.106760
AGE	-0.000000
DIS	-0.000000
RAD	-0.000000
TAX	-0.268261
PTRATIO	-1.848074
B	0.572120
LSTAT	-2.905391

dtype: float64

```
['CRIM' 'CHAS' 'NOX' 'RM' 'TAX' 'PTRATIO' 'B' 'LSTAT']
```

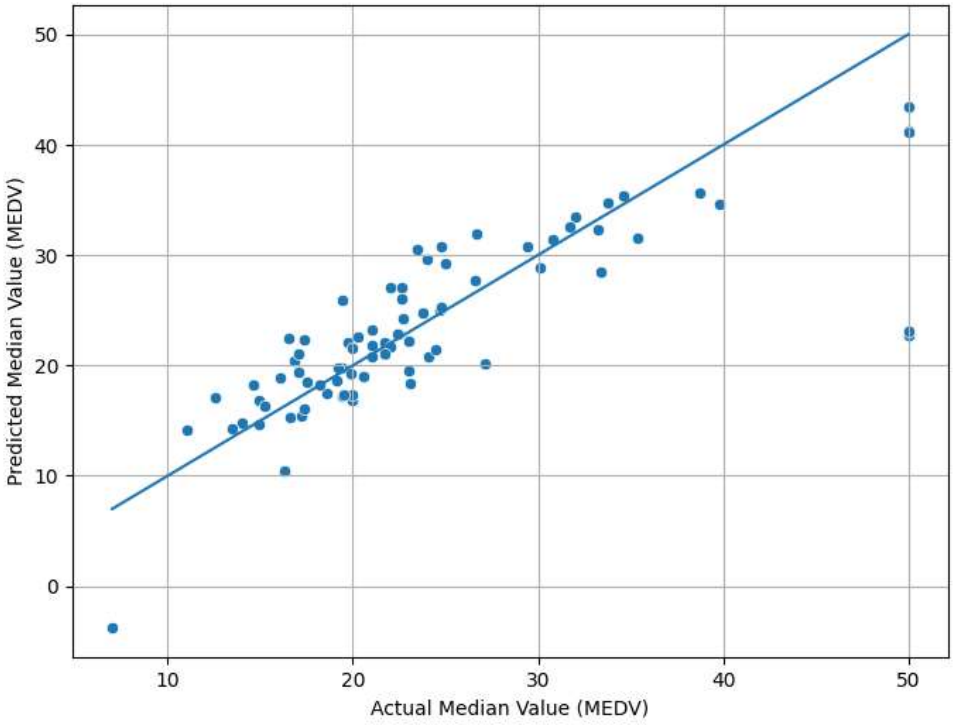
E. Least Square Regression algorithm

```
# Import the Linear Regression model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Initialize the Linear Regression model
linear_regression = LinearRegression()

# Train the model using the training data
# The model learns the coefficients that minimize the sum of squared errors between predictions and actual values.
linear_model = linear_regression.fit(x_train, y_train)

# Make predictions on the test set
linear_predication = linear_model.predict(x_test)

# Evaluate the Linear Regression model's performance
# Mean Squared Error (MSE) measures the average of the squared errors.
linear_mse = mean_squared_error(y_test, linear_predication)
# R-squared (R2) score represents the proportion of variance explained by the model.
linear_r2 = r2_score(y_test, linear_predication)

# Print the evaluation metrics for the Linear Regression model
print(f"Linear Regression - MSE: {linear_mse:2f} & R2: {linear_r2:2f}")
```

```
Linear Regression - MSE: 31.454048 & R2: 0.627085
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot of actual vs. predicted values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=linear_predication)

# Add a diagonal line representing perfect prediction
# This line helps visualize how close the predictions are to the actual values
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()])

plt.xlabel("Actual Median Value (MEDV)")
plt.ylabel("Predicted Median Value (MEDV)")
plt.grid(True)
plt.show()
```



Colba file link: [Prac_3.ipynb]

Practical 4: Trees and Forests

Aug 11, 2025

Explain decision tree classifier and decision tree regressor algorithms with the method that is used to split data into tree branches.

Decision Tree Classifier
A Decision Tree Classifier is a supervised learning algorithm used for classification tasks. It splits the data into subsets based on feature values, creating a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a class label.

Method of Splitting Data (Classification):
To split data, the Decision Tree Classifier uses Gini Impurity or Information Gain (based on entropy) as criteria to measure the "purity" of a node. It chooses the feature and threshold that best splits the data by reducing impurity or maximizing information gain.

- Gini Impurity: Measures how often a randomly chosen element would be incorrectly classified if randomly labeled based on the distribution of class labels in the node.
- Information Gain: Measures the reduction in entropy after a split. Entropy represents the randomness in the data.

Evaluation Methods for Decision Tree Classifier:

1. Accuracy: Percentage of correct predictions.
2. Precision, Recall, F1-Score: Useful for imbalanced datasets.
3. Confusion Matrix: Shows true positives, true negatives, false positives, and false negatives.
4. ROC-AUC: Evaluates the tradeoff between true positive rate and false positive rate for different threshold values.

Decision Tree Regressor
A Decision Tree Regressor is a similar algorithm used for regression tasks. Instead of classifying, it predicts a continuous value by learning decision rules from data.

Method of Splitting Data (Regression):
The Decision Tree Regressor splits data by minimizing the Mean Squared Error (MSE) or Mean Absolute Error (MAE) between predicted and actual values. The algorithm selects splits that minimize the sum of squared residuals or the absolute difference between predicted and actual values.

Evaluation Methods for Decision Tree Regressor:

1. Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
2. Mean Absolute Error (MAE): Measures the average absolute difference between actual and predicted values.
3. R-squared (R2): Indicates the proportion of variance in the target variable that is explained by the model.
4. Root Mean Squared Error (RMSE): Square root of the MSE, easier to interpret in the context of the problem.

Explain Random Forest Classifier and Random Forest Regressor with the method that is used to split data into tree branches.

Random Forest Classifier

A Random Forest Classifier is an ensemble learning method that constructs multiple decision trees and aggregates their predictions (majority voting for classification) to produce a final result. The random forest adds randomness by selecting random subsets of features for each tree to improve generalization and reduce overfitting.

Method of Splitting Data (Classification):

Each decision tree in the random forest uses Gini Impurity or Information Gain, just like in a single decision tree. However, random feature selection at each split ensures that different trees focus on different aspects of the data, improving robustness.

Evaluation Methods for Random Forest Classifier:

1. Accuracy: Overall percentage of correct predictions.
2. Precision, Recall, F1-Score: Helpful in imbalanced datasets.
3. Confusion Matrix: For evaluating classification performance.
4. Out-of-Bag (OOB) Score: In random forests, samples not used to train a tree can be used to evaluate it, providing a cross-validated performance estimate.

Random Forest Regressor

A Random Forest Regressor is similar to a Random Forest Classifier but used for regression tasks. It builds multiple decision trees, and the final prediction is the average of all trees' predictions.

Method of Splitting Data (Regression):

Similar to the Decision Tree Regressor, random forests for regression minimize MSE or MAE at each split. Additionally, randomness is introduced by selecting random features and random samples for each tree.

Evaluation Methods for Random Forest Regressor:

1. Mean Squared Error (MSE): Measures the average squared error between predicted and actual values.
2. Mean Absolute Error (MAE): Average absolute difference between predicted and actual values.
3. R-squared (R2): Proportion of variance explained by the model.
4. Out-of-Bag (OOB) Error: In regression, OOB error provides a performance estimate based on samples that were not used in training a specific tree.

Methods of evaluation of random tree algorithms:

1. Classification Metrics:

- Accuracy, Precision, Recall, F1-Score: As with decision trees.
- ROC Curve and AUC: Evaluate the trade-off between true positive rate and false positive rate across different threshold settings.

2. Regression Metrics:

- Mean Absolute Error (MAE), Mean Squared Error (MSE), R-squared: As with decision trees.
- Feature Importance: Random forests can also provide insights into the importance of different features for making predictions.

Dataset: Loan Prediction Problem Dataset

```
# Import necessary libraries for data manipulation and model evaluation
import pandas as pd
import numpy as np

# Load the dataset from the specified CSV file into a pandas DataFrame
df = pd.read_csv("/content/sample_data/Prac_4_Loan_Predication.csv")
# Display the first few rows of the DataFrame to inspect the data
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

```
# Convert categorical features to numerical representations using mapping
df['Gender'] = df['Gender'].map({'Male':1, 'Female':0})
df['Married'] = df['Married'].map({'Yes':1, 'No':0})
# Replace '3+' in Dependents column with 3
df['Dependents'] = df['Dependents'].replace('3+', 3)
df['Education'] = df['Education'].map({'Graduate':1, 'Not Graduate':0})
df['Self_Employed'] = df['Self_Employed'].map({'Yes':1, 'No':0})
df['Property_Area'] = df['Property_Area'].map({'Urban':2, 'Rural':0, 'Semiurban':1})
# Map the target variable 'Loan_Status' to numerical values
df['Loan_Status'] = df['Loan_Status'].map({'Y':1, 'N':0})

df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	1.0	0.0	0	1	0.0	5849	0.0	NaN	360.0	1.0
1	LP001003	1.0	1.0	1	1	0.0	4583	1508.0	128.0	360.0	1.0
2	LP001005	1.0	1.0	0	1	1.0	3000	0.0	66.0	360.0	1.0
3	LP001006	1.0	1.0	0	0	0.0	2583	2358.0	120.0	360.0	1.0
4	LP001008	1.0	0.0	0	1	0.0	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	0.0	0.0	0	1	0.0	2900	0.0	71.0	360.0	1.0
610	LP002979	1.0	1.0	3	1	0.0	4106	0.0	40.0	180.0	1.0
611	LP002983	1.0	1.0	1	1	0.0	8072	240.0	253.0	360.0	1.0
612	LP002984	1.0	1.0	2	1	0.0	7583	0.0	187.0	360.0	1.0
613	LP002990	0.0	0.0	0	1	1.0	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

```
# Perform one-hot encoding for the 'Property_Area' column to handle categorical data
df = pd.get_dummies(df, columns=['Property_Area'], drop_first=True)
# Display the updated DataFrame
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	1.0	0.0	0	1	0.0	5849	0.0	NaN	360.0	1.0
1	LP001003	1.0	1.0	1	1	0.0	4583	1508.0	128.0	360.0	1.0
2	LP001005	1.0	1.0	0	1	1.0	3000	0.0	66.0	360.0	1.0
3	LP001006	1.0	1.0	0	0	0.0	2583	2358.0	120.0	360.0	1.0
4	LP001008	1.0	0.0	0	1	0.0	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	0.0	0.0	0	1	0.0	2900	0.0	71.0	360.0	1.0
610	LP002979	1.0	1.0	3	1	0.0	4106	0.0	40.0	180.0	1.0
611	LP002983	1.0	1.0	1	1	0.0	8072	240.0	253.0	360.0	1.0
612	LP002984	1.0	1.0	2	1	0.0	7583	0.0	187.0	360.0	1.0
613	LP002990	0.0	0.0	0	1	1.0	4583	0.0	133.0	360.0	0.0

614 rows × 14 columns

```
# Check for missing values in each column and sum them up
df.isnull().sum()
```

	0
Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Loan_Status	0
Property_Area_1	0
Property_Area_2	0

```
# Fill missing values in categorical columns with the mode (most frequent value)
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
df['Married'] = df['Married'].fillna(df['Married'].mode()[0])
df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])
df['Self_Employed'] = df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mode()[0])

# Fill missing values in numerical columns with the mean
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())

# Checking for null values again to confirm they are filled
df.isnull().sum()
```

	0
Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Loan_Status	0
Property_Area_1	0
Property_Area_2	0

dtype: int64

```
# Check the data types of each column in the DataFrame
df.dtypes
```

	0
Loan_ID	object
Gender	float64
Married	float64
Dependents	object
Education	int64
Self_Employed	float64
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Loan_Status	int64
Property_Area_1	bool
Property_Area_2	bool

```
# Import train_test_split for splitting data into training and testing sets
from sklearn.model_selection import train_test_split

# Define features (x) by dropping 'Loan_ID' and 'Loan_Status'
x = df.drop(['Loan_ID', 'Loan_Status'], axis=1)
# Define the target variable (y)
y = df['Loan_Status']

# Split the data into training and testing sets (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Print the shapes of the resulting sets to verify the split
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((491, 12), (491,), (123, 12), (123,))
```

A. Training a Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize a Decision Tree Classifier with entropy criterion
dt = DecisionTreeClassifier(criterion='entropy', random_state= 42)

# Train the decision tree model on the training data
dt.fit(x_train, y_train)
# Make predictions on the training and testing data
dt_pred_train = dt.predict(x_train)
dt_pred_test = dt.predict(x_test)

# Evaluate the model on the training data
accuracy = accuracy_score(y_train, dt_pred_train)
conf_mat = confusion_matrix(y_train, dt_pred_train)
class_report = classification_report(y_train, dt_pred_train)

# Print evaluation metrics for the training data
print("Training Data Report\nAccuracy: ", accuracy)
print("\nConfusion Matrix: \n", conf_mat)
print("\nClassification Report: \n", class_report)


# Evaluate the model on the testing data
accuracy = accuracy_score(y_test, dt_pred_test)
conf_mat = confusion_matrix(y_test, dt_pred_test)
class_report = classification_report(y_test, dt_pred_test)

# Print evaluation metrics for the testing data
print("\nTesting Data Report\nAccuracy: ", accuracy)
print("\nConfusion Matrix: \n", conf_mat)
print("\nClassification Report: \n", class_report)
```

```
Training Data Report
Accuracy:  1.0

Confusion Matrix:
 [[149   0]
 [  0 342]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       149
           1       1.00      1.00      1.00       342

    accuracy                           1.00       491
   macro avg       1.00      1.00      1.00       491
weighted avg       1.00      1.00      1.00       491


Testing Data Report
Accuracy:  0.7398373983739838

Confusion Matrix:
 [[20 23]
 [ 9 71]]

Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.47      0.56        43
           1       0.76      0.89      0.82        80

    accuracy                           0.74       123
   macro avg       0.72      0.68      0.69       123
weighted avg       0.73      0.74      0.73       123
```

B. Training a Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize a Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(random_state=42)

# Train the decision tree regressor on the training data
dt_regressor.fit(x_train, y_train)

# make prediction
y_train_pred = dt_regressor.predict(x_train)
y_test_pred = dt_regressor.predict(x_test)

#evaluate
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Data Report\nMean Squared Error (MSE):", mse_train)
print("R-squared (R2) Score:", r2_train)

# Evaluate the model on the testing data
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\nTesting Data Report\nMean Squared Error (MSE):", mse_test)
print("R-squared (R2) Score:", r2_test)
```

```
Training Data Report
Mean Squared Error (MSE): 0.0
R-squared (R2) Score: 1.0

Testing Data Report
Mean Squared Error (MSE): 0.2926829268292683
R-squared (R2) Score: -0.2872093023255815
```

C. Training a Random Forest Classifier

```
# Building Random forest tree
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest Classifier with entropy criterion
rf = RandomForestClassifier(criterion='entropy', random_state= 42)

# Train the random forest model on the training data
rf.fit(x_train, y_train)
# Make predictions on the training and testing data
rf_pred_train = rf.predict(x_train)
rf_pred_test = rf.predict(x_test)


# Evaluate the model on the training data
accuracy = accuracy_score(y_train, rf_pred_train)
conf_mat = confusion_matrix(y_train, rf_pred_train)
class_report = classification_report(y_train, rf_pred_train)

# Print evaluation metrics for the training data
print("Accuracy: ", accuracy)
print("\nConfusion Matrix: \n", conf_mat)
print("\nClassification Report: \n", class_report)


# Evaluate the model on the testing data
accuracy = accuracy_score(y_test, rf_pred_test)
conf_mat = confusion_matrix(y_test, rf_pred_test)
class_report = classification_report(y_test, rf_pred_test)

# Print evaluation metrics for the testing data
print("\nTesting Data Report\nAccuracy: ", accuracy)
print("\nConfusion Matrix: \n", conf_mat)
print("\nClassification Report: \n", class_report)
```

```
Accuracy:  1.0

Confusion Matrix:
 [[149   0]
 [  0 342]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       149
           1       1.00      1.00      1.00       342

    accuracy                           1.00       491
   macro avg       1.00      1.00      1.00       491
weighted avg       1.00      1.00      1.00       491


Testing Data Report
Accuracy:  0.7723577235772358

Confusion Matrix:
 [[18 25]
 [ 3 77]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.42      0.56        43
           1       0.75      0.96      0.85        80

    accuracy                           0.77       123
   macro avg       0.81      0.69      0.70       123
weighted avg       0.79      0.77      0.75       123
```

D. Training a Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize a Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the random forest regressor on the training data
rf_regressor.fit(x_train, y_train)

# Make predictions
y_train_pred = rf_regressor.predict(x_train)
y_test_pred = rf_regressor.predict(x_test)

# Evaluate on training data
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)
# Evaluate on training data
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Data Report\nMean Squared Error (MSE):", mse_train)
print("R-squared (R2) Score:", r2_train)
# Evaluate on testing data
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\nTesting Data Report\nMean Squared Error (MSE):", mse_test)
print("R-squared (R2) Score:", r2_test)
```

```
Training Data Report
Mean Squared Error (MSE): 0.02312097759674134
R-squared (R2) Score: 0.8906152439263707

Testing Data Report
Mean Squared Error (MSE): 0.17908130081300813
R-squared (R2) Score: 0.21240668604651147
```

Colab file link: Prac 4

Practical 5: K-Nearest Neighbors Classifier

Sep 12, 2025

Explain K-nearest Neighbour classifier with an example. Demonstrate the method of calculation of the center in each iteration.

K-Nearest Neighbors (KNN) Classifier

The K-Nearest Neighbors (KNN) classifier is a supervised learning algorithm used for classification tasks. It works by finding the 'k' closest training examples in the feature space and assigning the most common class label among them to the new data point.

- Step 1: Choose the number 'k', the number of nearest neighbors to consider.
- Step 2: Calculate the distance between the new data point and all the data points in the training set (common distance metrics include Euclidean distance).
- Step 3: Identify the 'k' nearest neighbors to the new data point.
- Step 4: Determine the class label based on the majority class among these 'k' neighbors.

Calculation Example (simplified):

1. Distance Calculation: For each training example, compute the distance from the new data point.
2. Neighbor Selection: Sort the training examples by distance and select the top 'k' nearest ones.
3. Voting: Count the number of votes for each class among these 'k' neighbors.
4. Classification: Assign the class label with the most votes to the new data point.

K-Means Algorithm

Theory: K-Means is an unsupervised learning algorithm used for clustering. It partitions a dataset into 'k' clusters by iteratively improving the placement of cluster centers.

1. Initialization: Choose 'k' initial cluster centers randomly.
2. Assignment Step: Assign each data point to the nearest cluster center.
3. Update Step: Recalculate the cluster centers as the mean of all data points assigned to each cluster.
4. Repeat: Iterate the assignment and update steps until cluster centers no longer change significantly.

Evaluation Methods:

1. Within-Cluster Sum of Squares (WCSS): Measures the variance within each cluster. Lower WCSS indicates better clustering.
2. Silhouette Score: Measures how similar a data point is to its own cluster compared to other clusters. Scores range from -1 to 1, with higher values indicating better clustering.

3. Davies-Bouldin Index: Evaluates the average similarity ratio of each cluster with its most similar cluster. Lower values indicate better clustering.

Explain Radius-Based Nearest Neighbor Classifier

Theory: The Radius-Based Nearest Neighbor Classifier classifies a data point based on the class of its neighbors within a specified radius.

- Step 1: Define a radius within which to search for neighbors.
- Step 2: For each new data point, find all training examples within this radius.
- Step 3: Assign the most frequent class label among these neighbors to the new data point.

Features		Target		Features		Target
X1	Y1			X2	Y2	
Brightness	Saturation	Class		Brightness	Saturation	Class
40	20	Red		20	35	?
50	50	Blue		55	60	?
60	90	Blue		62	90	?
10	25	Red				
70	70	Blue				
60	10	Red				
25	80	Blue				

A. Implement the various distance calculation methods

1. **Euclidean Distance:** The Euclidean Distance between two points in Euclidean space is the length of the line segment connecting them. It's calculated using the Pythagorean theorem.

2. **Manhattan Distance (Taxicab or City Block):** The Manhattan Distance is the sum of the absolute differences of their Cartesian coordinates. It's called Manhattan distance

because it reflects the real-world distances one would have to walk in a city laid out in a grid.

3. **Minkowski Distance:** The Minkowski Distance is a generalized form of both the Euclidean and Manhattan distances. It's calculated as the p-th root of the sum of the absolute differences raised to the power p.

	Features		Target
	X1	Y1	
	Brightness	Saturation	Class
1	40	20	Red
2	50	50	Blue
3	60	90	Blue
4	10	25	Red
5	70	70	Blue
6	60	10	Red
7	25	80	Blue

	Features		Target
	X2	Y2	
	Brightness	Saturation	Class
1	20	35	?
2	55	60	?
3	62	90	?

Euclidean =SQRT(POWER(A5-20, 2) + POWER(B5-35, 2))
Manhattan =ABS(A5-20)+ABS(B5-35)
Minkowski (p=3) =((ABS(A5 - 20)^3 + ABS(B5 - 35)^3))^(1/3)

Distances for (20, 35)		
		p = 3
Euclidean	Manhattan	Minkowski
25	35	22.489707
33.54102	45	31.201257
68.007353	95	61.302537
14.142136	20	12.59921
61.032778	85	55.164795
47.169906	65	43.021262
45.276926	50	45.020567

Distances for (55, 60)		
		p = 3
Euclidean	Manhattan	Minkowski
42.720019	55	40.691115
11.18034	15	10.400419
30.413813	35	30.046225
57.008771	80	51.172299
18.027756	25	16.355332
50.249378	55	50.016661
36.055513	50	32.710663

Distances for (62, 90)		
		p = 3
Euclidean	Manhattan	Minkowski
73.375745	92	70.716985
41.761226	52	40.356808
2	2	2
83.240615	117	74.604316
21.540659	28	20.417875
80.024996	82	80.000417
38.327536	47	37.241902

	Features		Target
	X1	Y1	
	Brightness	Saturation	Class
1	40	20	Red
2	50	50	Blue
3	60	90	Blue
4	10	25	Red
5	70	70	Blue
6	60	10	Red
7	25	80	Blue

	Features		Target
	X2	Y2	
	Brightness	Saturation	Class
1	20	35	Red
2	55	60	Blue
3	62	90	Red

```
brightness = [10, 25, 40, 50, 60, 60, 70]
saturation = [25, 80, 20, 50, 90, 10, 70]

for x, y in zip(brightness, saturation):
  print(x, y)
```

```
10 25
25 80
40 20
50 50
60 90
60 10
70 70
```

```
import math

# Function to calculate Euclidean distance between two points
def euclidean_distance(point1, point2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(point1, point2)))

# Function to calculate Manhattan distance between two points
def manhattan_distance(point1, point2):
    return sum(abs(x - y) for x, y in zip(point1, point2))

# Function to calculate Minkowski distance between two points with order p
def minkowski_distance(point1, point2, p):
    if p <= 0:
        raise ValueError("Order 'p' must be greater than 0")
    return sum(abs(x - y) ** p for x, y in zip(point1, point2)) ** (1 / p)

# Calculate and print the distances for sample points of brightness and saturation
print("Euclidean Distance:", euclidean_distance(brightness, saturation))
print("Manhattan Distance:", manhattan_distance(brightness, saturation))
print("Minkowski Distance (p=3):", minkowski_distance(brightness, saturation, 3))
```

```
Euclidean Distance: 83.96427811873333
Manhattan Distance: 170
Minkowski Distance (p=3): 69.08677732783606
```

```python
from scipy.spatial import distance

# Calculate Euclidean Distance using scipy
euclidean = distance.euclidean(brightness, saturation)

# Calculate Manhattan Distance (also called cityblock in scipy)
manhattan = distance.cityblock(brightness, saturation)

# Calculate Minkowski Distance (generalized, with p = 3 for example) using scipy
minkowski_p3 = distance.minkowski(brightness, saturation, p=3)

# Output the calculated distances
print("Euclidean Distance (scipy):", euclidean)
print("Manhattan Distance (scipy):", manhattan)
print("Minkowski Distance (p=3, scipy):", minkowski_p3)
```

```
Euclidean Distance (scipy): 83.96427811873333
Manhattan Distance (scipy): 170
Minkowski Distance (p=3, scipy): 69.08677732783606
```

B. Creating a Nearest Neighbor Classifier using Euclidean distance method

```python
from sklearn import datasets
from sklearn.neighbors import NearestNeighbors
import pandas as pd

# Load the dataset from the specified CSV file
df = pd.read_csv("/content/sample_data/Prac_5_Dataset.csv")
display(df)

# Separate features (Brightness and Saturation) from the target variable (Class)
features = df.drop("Class", axis=1)
print("\n\nFeatures: \n", features)

# Initialize the NearestNeighbors model with 2 neighbors and Euclidean distance metric
nearest_neighbors_euclidian = NearestNeighbors(n_neighbors=2, metric='euclidean').fit(features)
```

	Brightness	Saturation	Class
0	40	20	Red
1	50	50	Blue
2	60	90	Blue
3	10	25	Red
4	70	70	Blue
5	60	10	Red
6	25	80	Blue

```
Features:
    Brightness  Saturation
0          40          20
1          50          50
2          60          90
3          10          25
4          70          70
5          60          10
6          25          80
```

```
# Define a new observation (Brightness, Saturation)
new_observation = [20, 35]

# Find the k nearest neighbors for the new observation
distances, indices = nearest_neighbors_euclidian.kneighbors([new_observation])

for d, i in zip(distances.flatten(), indices.flatten()):
  print(i, " --> ", d)
```

```
3  -->  14.142135623730951
0  -->  25.0
```

Distances for (20, 35)		
		p = 3
Euclidean	**Manhattan**	**Minkowski**
25	35	22.489707
33.54102	45	31.201257
68.007353	95	61.302537
14.142136	20	12.59921
61.032778	85	55.164795
47.169906	65	43.021262
45.276926	50	45.020567

```
# Define a new observation (Brightness, Saturation)
new_observation = [55, 60]

# Find the k nearest neighbors for the new observation
distances, indices = nearest_neighbors_euclidian.kneighbors([new_observation])

for d, i in zip(distances.flatten(), indices.flatten()):
  print(i, " --> ", d)
```

```
1  -->  11.180339887498949
4  -->  18.027756377319946
```

Distances for (55, 60)		
		p = 3
Euclidean	**Manhattan**	**Minkowski**
42.720019	55	40.691115
11.18034	15	10.400419
30.413813	35	30.046225
57.008771	80	51.172299
18.027756	25	16.355332
50.249378	55	50.016661
36.055513	50	32.710663

```
# Initialize the NearestNeighbors model with 2 neighbors and manhattan, minkowski distance metric
nearest_neighbors_manhattan = NearestNeighbors(n_neighbors=2, metric='manhattan').fit(features)
nearest_neighbors_minkowski = NearestNeighbors(n_neighbors=2, metric='minkowski', p=3).fit(features)

# Define a new observation (Brightness, Saturation)
new_observation = [62, 90]

# Find the k nearest neighbors for the new observation
print("Manhattan : ")
distances, indices = nearest_neighbors_manhattan.kneighbors([new_observation])
for d, i in zip(distances.flatten(), indices.flatten()):
  print(i, " --> ", d)

print("\nMinkowski : ")
distances, indices = nearest_neighbors_minkowski.kneighbors([new_observation])
for d, i in zip(distances.flatten(), indices.flatten()):
  print(i, " --> ", d)
```

```
Manhattan :
2  -->  2.0
4  -->  28.0

Minkowski :
2  -->  2.0
4  -->  20.417874888005848
```

Distances for (62, 90)		
		p = 3
Euclidean	**Manhattan**	**Minkowski**
73.375745	92	70.716985
41.761226	52	40.356808
2	**2**	**2**
83.240615	117	74.604316
21.540659	**28**	**20.417875**
80.024996	82	80.000417
38.327536	47	37.241902

C. Creating a Radius-Based Nearest Neighbor Classifier.

```python
from sklearn.neighbors import RadiusNeighborsClassifier
import pandas as pd

# Load the dataset from the specified CSV file
df = pd.read_csv("/content/sample_data/Prac_5_Dataset.csv")

# Separate features (Brightness and Saturation) from the target variable (Class)
features = df.drop("Class", axis=1)
target = df["Class"]

print("Features: \n", features)
print("\nTarget: \n", target)

# Initialize the RadiusNeighborsClassifier with a radius of 20
rnn = RadiusNeighborsClassifier(
    radius=20,
    n_jobs=-1 # Use all available CPU cores
    ).fit(features, target)
```

```
Features:
    Brightness  Saturation
0           40          20
1           50          50
2           60          90
3           10          25
4           70          70
5           60          10
6           25          80

Target:
 0     Red
1    Blue
2    Blue
3     Red
4    Blue
5     Red
6    Blue
Name: Class, dtype: object
```

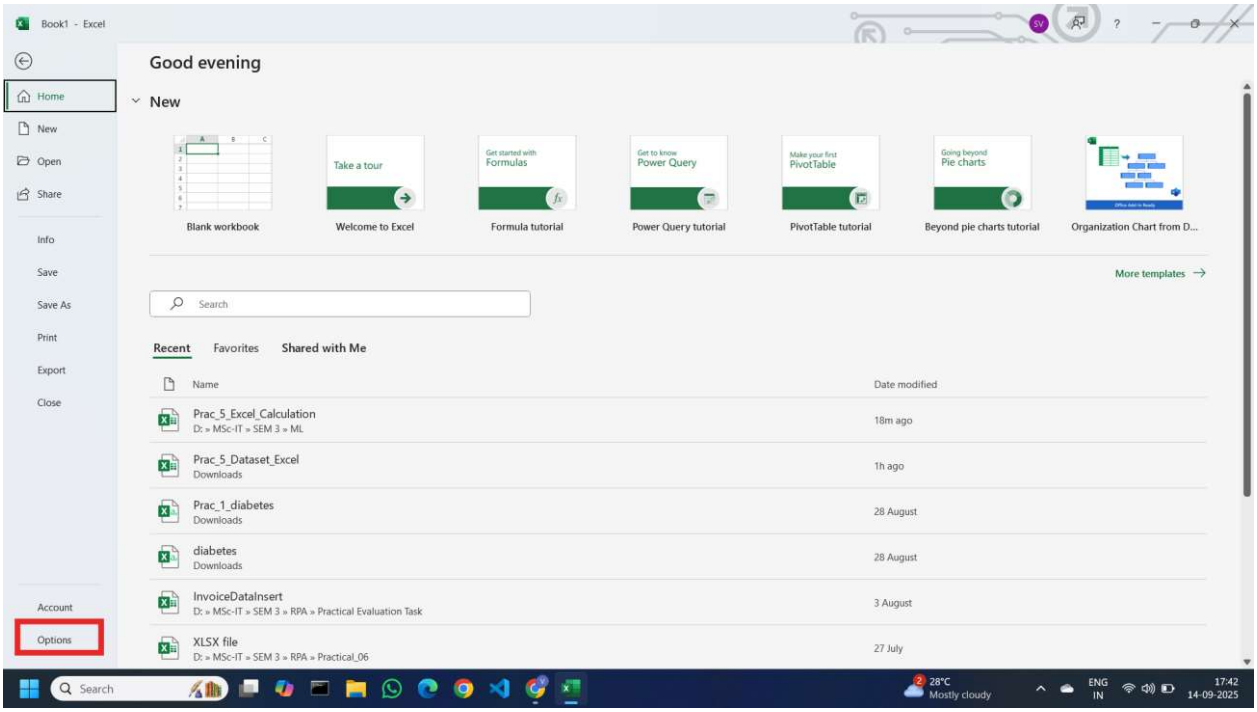
```python
# Define new observations to classify
new_observations = [[20, 35]]

# Predication of class
rnn.predict(new_observations)
```

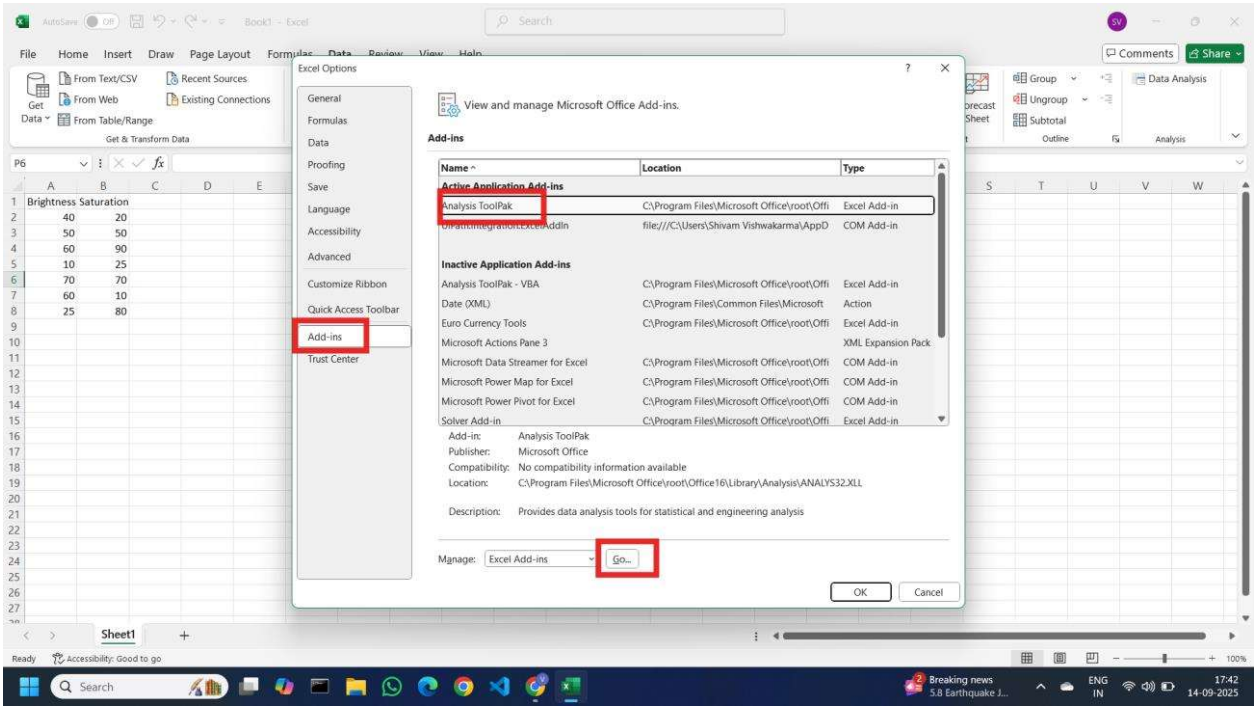
```
/usr/local/lib/python3.12/dist-packages/
  warnings.warn(
array(['Red'], dtype=object)
```

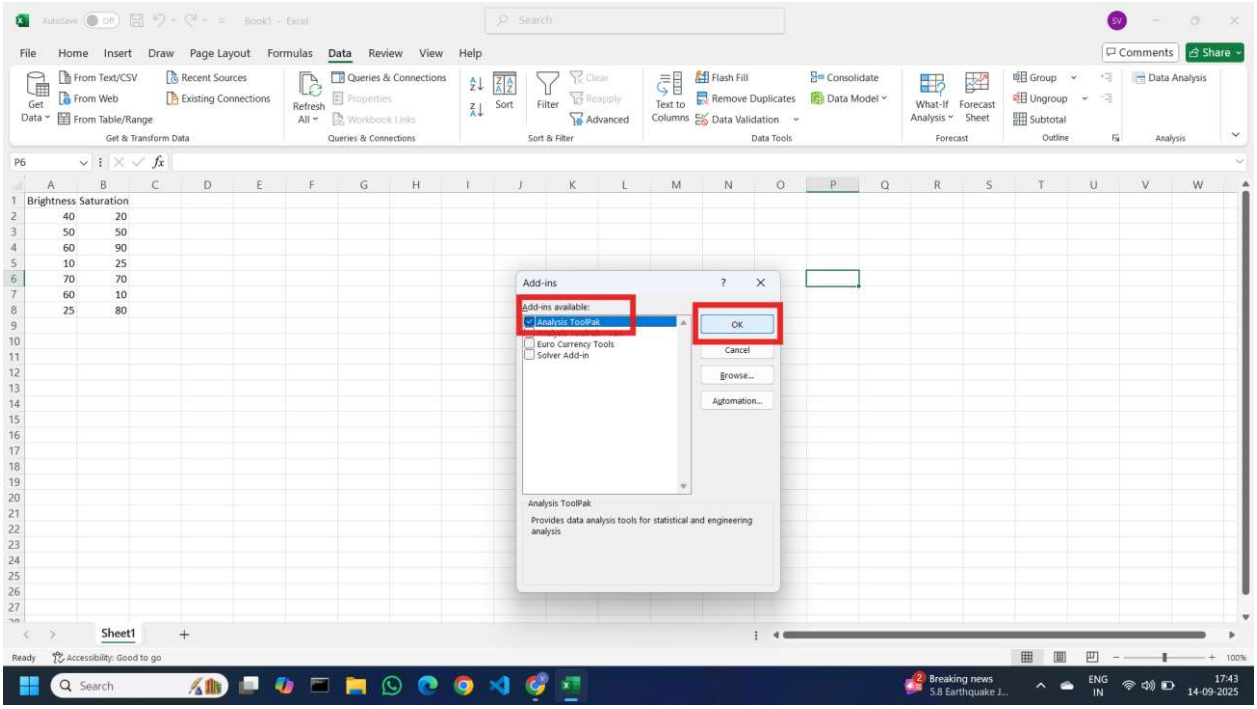
```
# Define new observations to classify
new_observations = pd.DataFrame([
    [20, 35],
    [55, 60],
    [62, 90]
], columns=["Brightness", "Saturation"])

new_observations["Predicated Class"] = rnn.predict(new_observations)
new_observations
```

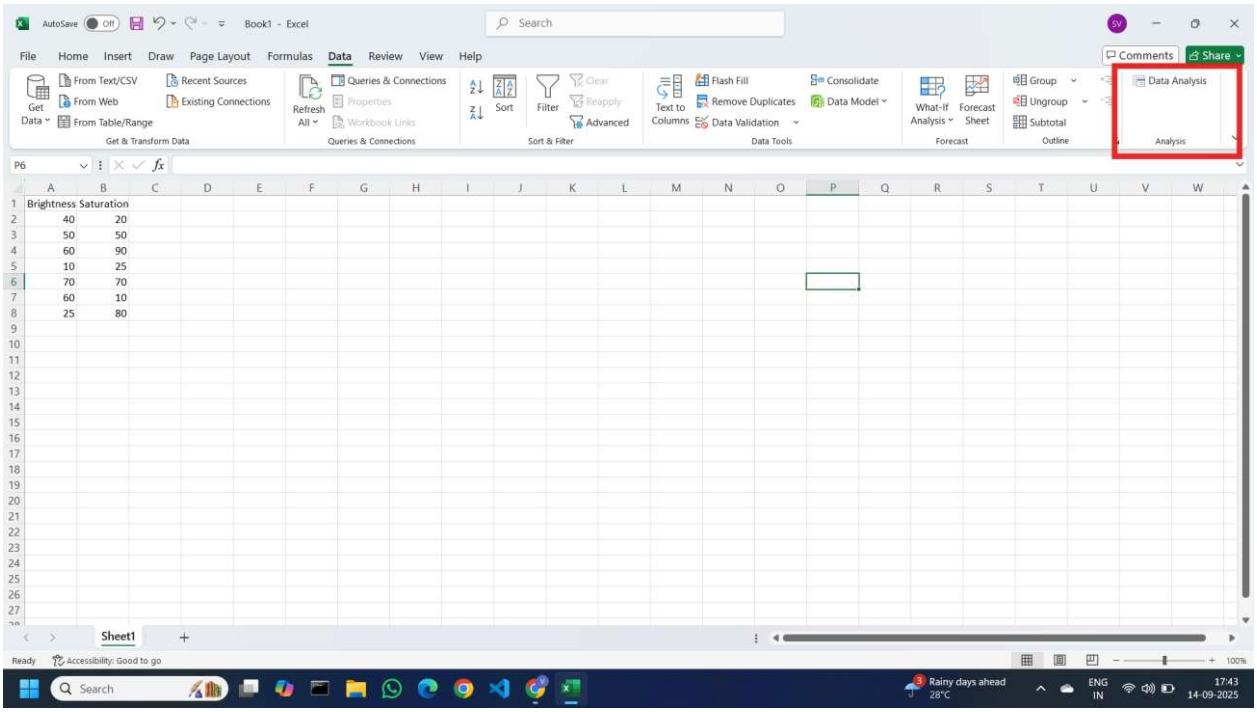
	Brightness	Saturation	Predicated Class
0	20	35	Red
1	55	60	Blue
2	62	90	Blue

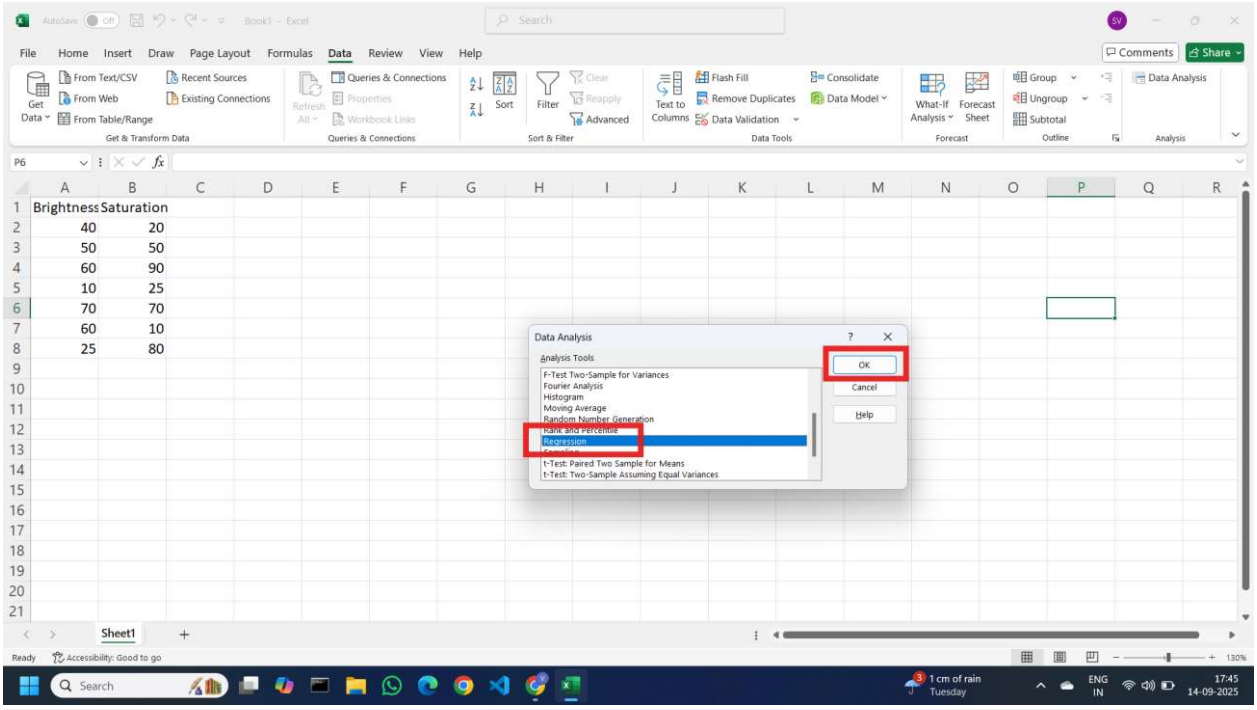
Extra

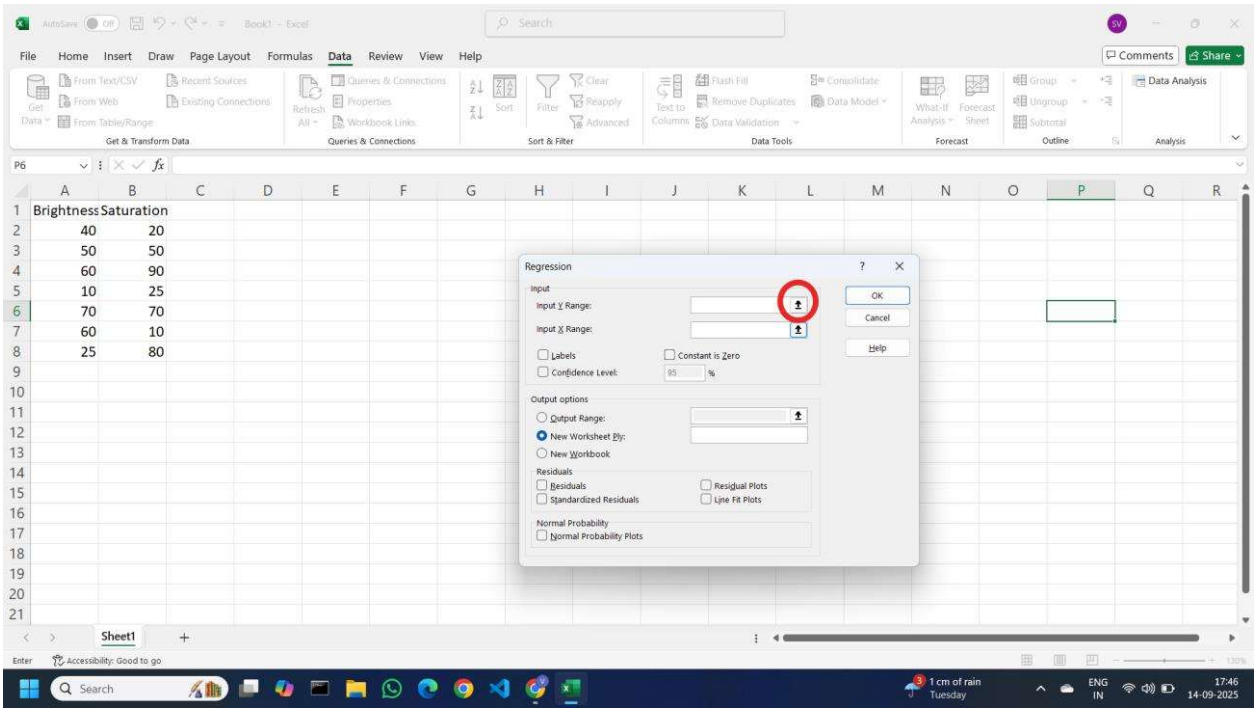

Book1 - Excel
Good evening
New
Home
New
Open
Share
Info
Save
Save As
Print
Export
Close
Account
Options
Blank workbook
Take a tour
Welcome to Excel
Formula tutorial
Power Query tutorial
PivotTable tutorial
Beyond pie charts tutorial
Organization Chart from D...
More templates
Search
Recent
Favorites
Shared with Me
Name
Date modified
Prac_5_Excel_Calculation
D: » MSc-IT » SEM 3 » ML
18m ago
Prac_5_Dataset_Excel
Downloads
1h ago
Prac_1_diabetes
Downloads
28 August
diabetes
Downloads
28 August
InvoiceDataInsert
D: » MSc-IT » SEM 3 » RPA » Practical Evaluation Task
3 August
XLSX file
D: » MSc-IT » SEM 3 » RPA » Practical_06
27 July
28°C
Mostly cloudy
ENG
IN
17:42
14-09-2025

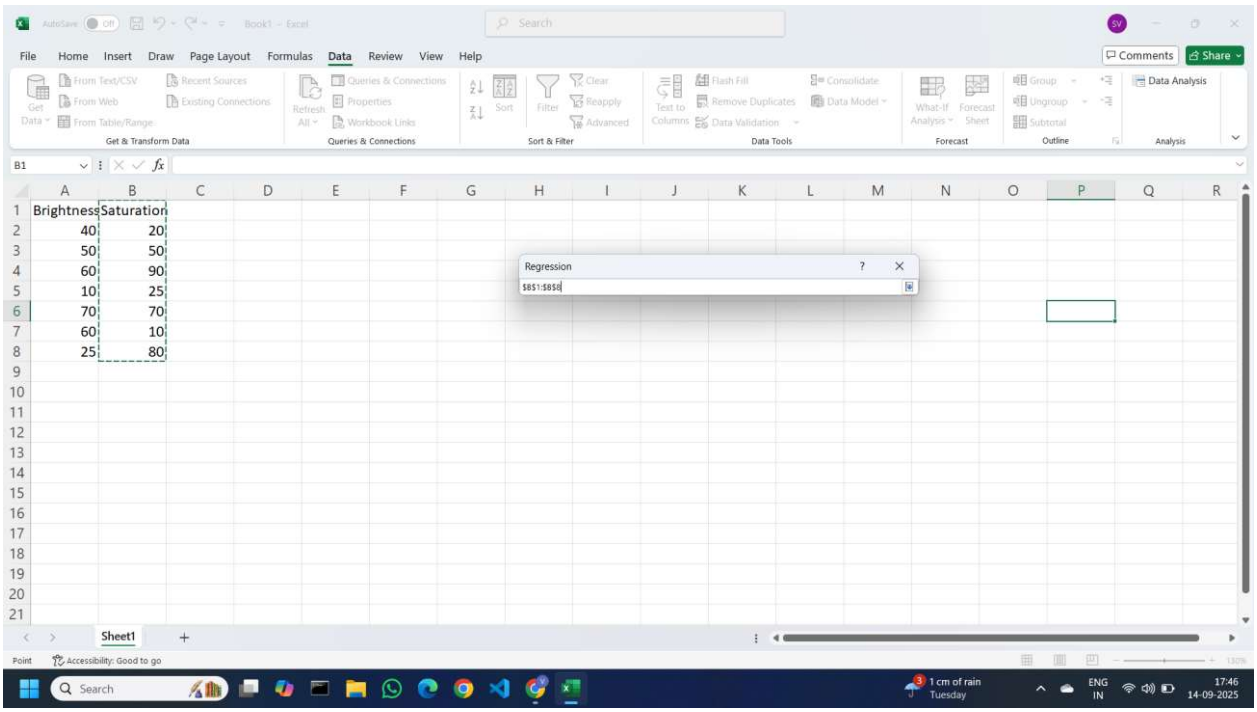

Excel Options
General
Formulas
Data
Proofing
Save
Language
Accessibility
Advanced
Customize Ribbon
Quick Access Toolbar
Add-ins
Trust Center
View and manage Microsoft Office Add-ins.
Add-ins
Name
Location
Type
Active Application Add-ins
Analysis ToolPak
C:\Program Files\Microsoft Office\root\Offi
Excel Add-in
COM Add-in
Inactive Application Add-ins
Analysis ToolPak - VBA
Date (XML)
Euro Currency Tools
Microsoft Actions Pane 3
Microsoft Data Streamer for Excel
Microsoft Power Map for Excel
Microsoft Power Pivot for Excel
Solver Add-in
Add-in: Analysis ToolPak
Publisher: Microsoft Office
Compatibility: No compatibility information available
Location: C:\Program Files\Microsoft Office\root\Office16\Library\Analysis\ANALYS32.XLL
Description: Provides data analysis tools for statistical and engineering analysis
Manage: Excel Add-ins
Go...
OK
Cancel
Sheet1
Ready
Accessibility: Good to go
Breaking news
5.8 Earthquake J...
ENG
IN
17:42
14-09-2025

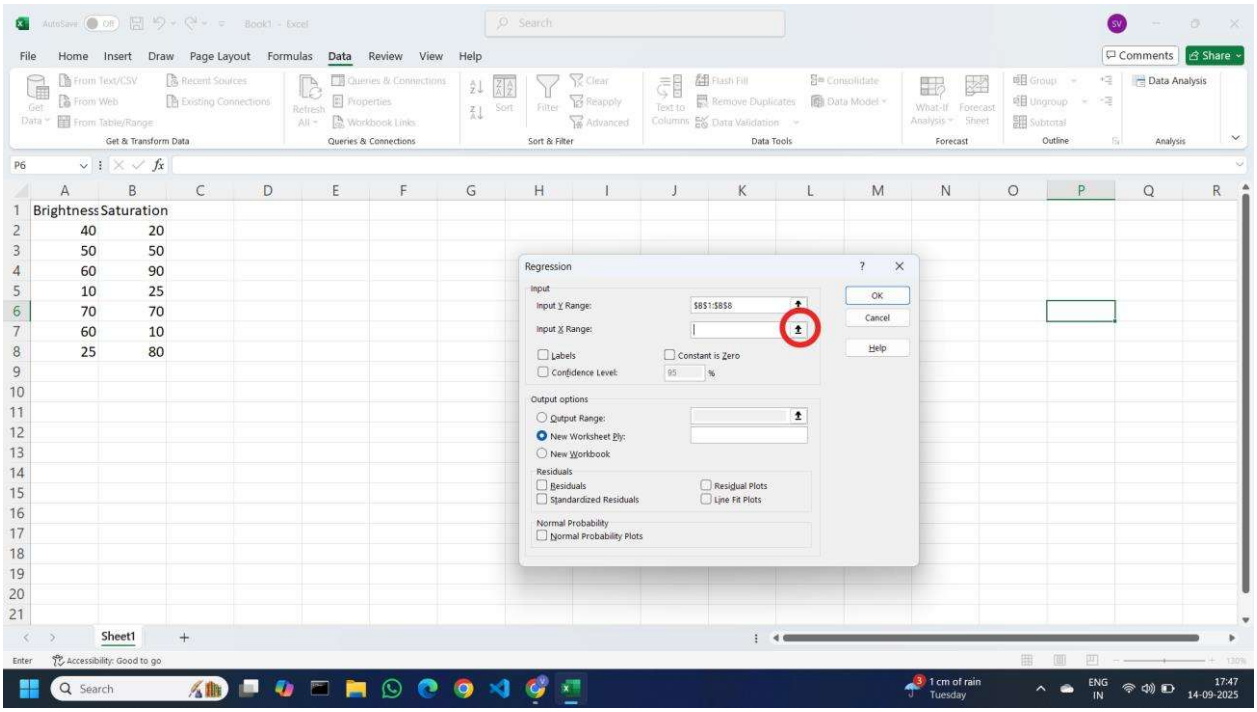

Add-ins
Add-ins available:
Analysis ToolPak
Euro Currency Tools
Solver Add-in
OK
Cancel
Browse...
Automation...
Analysis ToolPak
Provides data analysis tools for statistical and engineering analysis

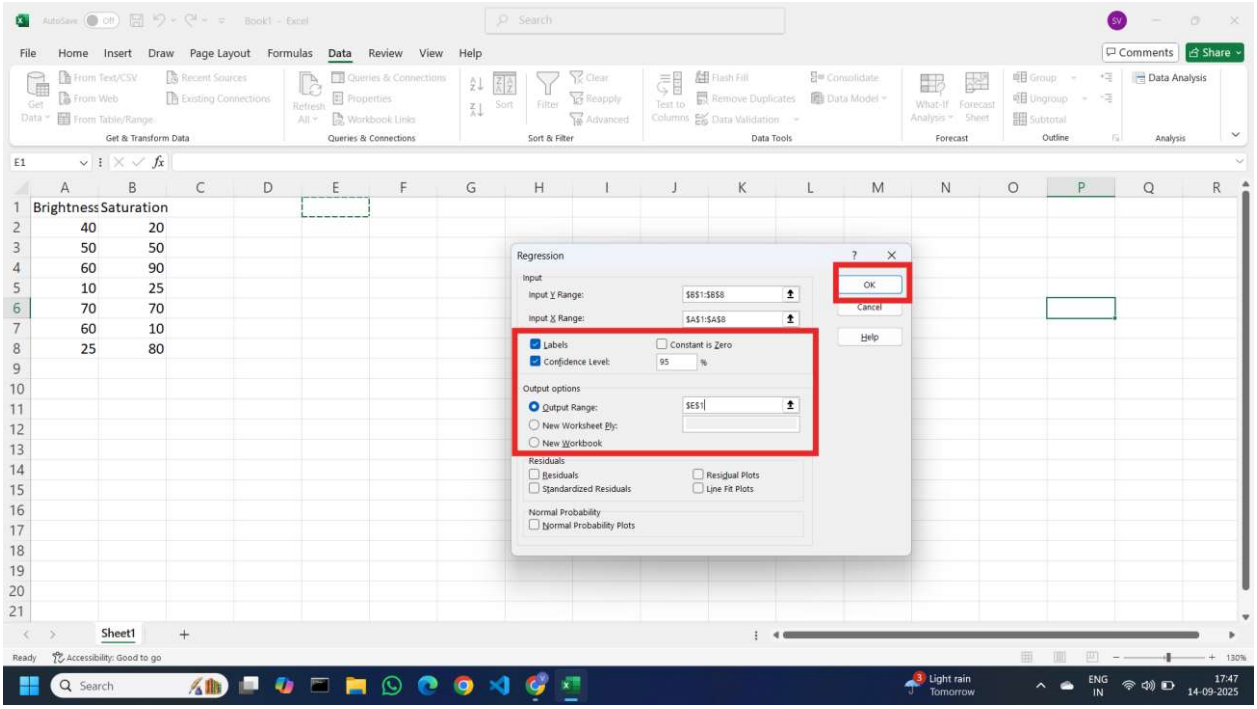


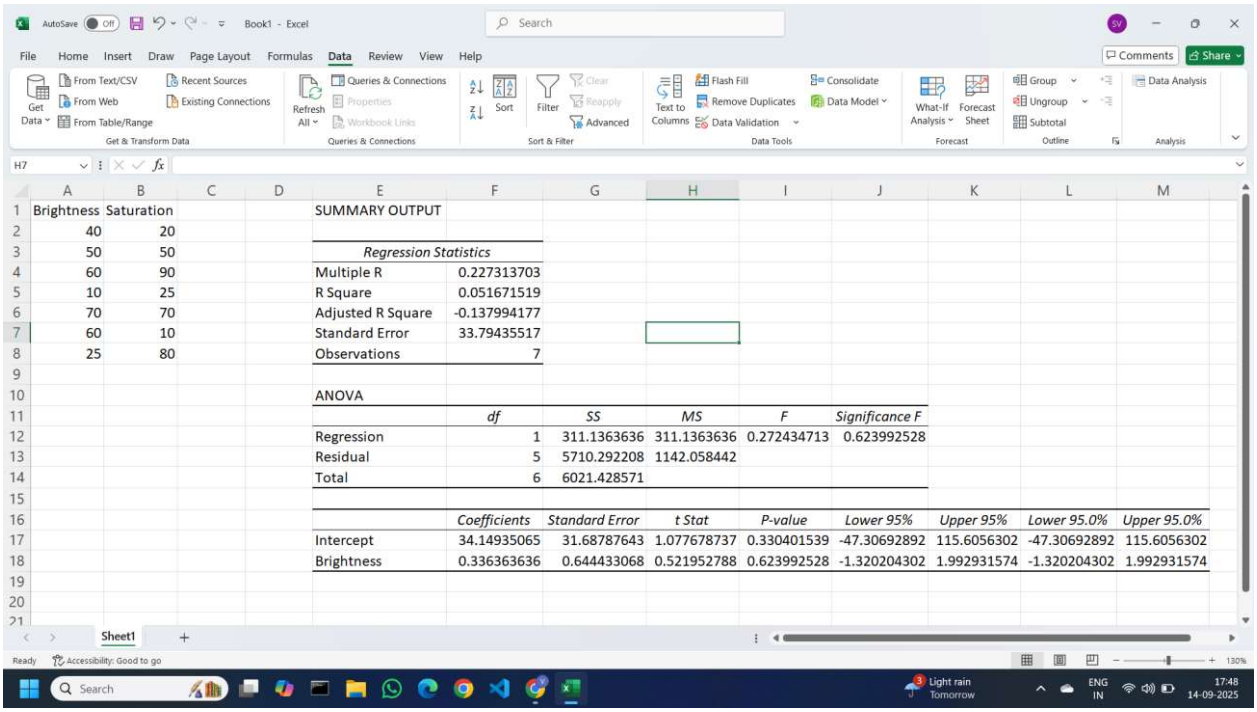

Data Analysis
Analysis Tools
F-Test Two-Sample for Variances
Fourier Analysis
Histogram
Moving Average
Random Number Generation
Rank and Percentile
Regression
t-Test: Paired Two Sample for Means
t-Test: Two-Sample Assuming Equal Variances
OK
Cancel
Help


Regression
Input
Input Y Range:
Input X Range:
Labels
Constant is Zero
Confidence Level:
Output options
Output Range:
New Worksheet Ply:
New Workbook
Residuals
Residuals
Residual Plots
Standardized Residuals
Line Fit Plots
Normal Probability
Normal Probability Plots
OK
Cancel
Help


Book1 - Excel
Search
File Home Insert Draw Page Layout Formulas Data Review View Help
Comments Share
Get & Transform Data
Queries & Connections
Sort & Filter
Data Tools
Forecast
Outline
Analysis
B1
Brightness Saturation
40 20
50 50
60 90
10 25
70 70
60 10
25 80
Regression
$B$1:$B$8
Sheet1
Point
Accessibility: Good to go
1 cm of rain
Tuesday
ENG
IN
17:46
14-09-2025


Book1 - Excel
Search
File Home Insert Draw Page Layout Formulas Data Review View Help
Comments Share
Get & Transform Data
Queries & Connections
Sort & Filter
Data Tools
Forecast
Outline
Analysis
P6
Brightness Saturation
40 20
50 50
60 90
10 25
70 70
60 10
25 80
Regression
Input
Input Y Range: $B$1:$B$8
Input X Range:
Labels
Constant is Zero
Confidence Level: 95 %
OK
Cancel
Help
Output options
Output Range:
New Worksheet Ply:
New Workbook
Residuals
Residuals
Residual Plots
Standardized Residuals
Line Fit Plots
Normal Probability
Normal Probability Plots
Sheet1
Enter
Accessibility: Good to go
1 cm of rain
Tuesday
ENG
IN
17:47
14-09-2025





Brightness	Saturation
40	20
50	50
60	90
10	25
70	70
60	10
25	80

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.227313703
R Square	0.051671519
Adjusted R Square	-0.137994177
Standard Error	33.79435517
Observations	7

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	311.1363636	311.1363636	0.272434713	0.623992528
Residual	5	5710.292208	1142.058442		
Total	6	6021.428571			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	34.14935065	31.68787643	1.077678737	0.330401539	-47.30692892	115.6056302	-47.30692892	115.6056302
Brightness	0.336363636	0.644433068	0.521952788	0.623992528	-1.320204302	1.992931574	-1.320204302	1.992931574

Excel Calculation [Prac_5_Excel_Calculation.xlsx]
Colab file link: [Prac_5.ipynb]

Practical 6: Logistic Regression

Sep 18, 2025

Logistic Regression is a statistical algorithm used for binary classification tasks. It predicts the probability that a given input belongs to one of two classes. Despite its name, it is actually a classification algorithm, not a regression algorithm.

How Logistic Regression Works:

1. Linear Combination of Inputs: Logistic Regression computes a weighted sum of the input features (like in Linear Regression). Each feature has a coefficient (or weight) that defines its importance in predicting the target outcome.
2. Sigmoid Function: To convert this linear combination into a probability value (between 0 and 1), the output is passed through the sigmoid function
3. Decision Boundary: Logistic Regression sets a threshold (usually 0.5) to decide the class label. If the output probability is greater than or equal to 0.5, the prediction is classified as 1 (positive class), otherwise 0 (negative class).
4. Cost Function: Instead of using Mean Squared Error (like in Linear Regression), Logistic Regression uses a special cost function called log-loss or binary cross-entropy to measure how well the model predicts probabilities
5. Optimization: The model coefficients are optimized using gradient descent, which minimizes the cost function by iteratively adjusting the weights to reduce prediction errors.

Key Points:

- Logistic Regression is good for binary classification problems.
- The output is a probability value, interpreted as the likelihood of the input belonging to a certain class.
- The algorithm is simple, interpretable, and performs well with linearly separable

A. Training a Binary Classifier

```
# Load libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Load data
iris_data = pd.read_csv('/content/sample_data/Iris.csv')

# Drop Id column if it exists
iris_data = iris_data.drop(columns=["Id"])

# Select first 100 samples (two classes only)
features = iris_data.iloc[:100, :-1].values
target = iris_data.iloc[:100, -1].values

# Standardize features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)

# Train logistic regression
logistic_regression = LogisticRegression(random_state=0)
model = logistic_regression.fit(features_standardized, target)

# New observation (must also be standardized)
new_observation = [[0.5, 0.5, 0.5, 0.5]]
new_observation_standardized = scaler.transform(new_observation)

# Predict
predicted_class = model.predict(new_observation_standardized)
predicted_probabilities = model.predict_proba(new_observation_standardized)

predicted_class, predicted_probabilities
```

```
(array(['Iris-setosa'], dtype=object), array([[0.96202919, 0.03797081]]))
```

B. Training a Multiclass Classifier

```
# Load libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Load data from CSV
iris_data = pd.read_csv("/content/sample_data/Iris.csv")

# If the dataset has an 'Id' column, drop it
if "Id" in iris_data.columns:
    iris_data = iris_data.drop(columns=["Id"])

# Separate features and target
features = iris_data.iloc[:, :-1].values   # all feature columns
target = iris_data.iloc[:, -1].values      # target (species)

# Standardize features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
# Create one-vs-rest logistic regression
logistic_regression = LogisticRegression(random_state=0, multi_class="ovr")
model = logistic_regression.fit(features_standardized, target)

# New observation (must also be standardized)
new_observation = [[0.5, 0.5, 0.5, 0.5]]
new_observation_standardized = scaler.transform(new_observation)

# Predict class
predicted_class = model.predict(new_observation_standardized)

# Predict probabilities
predicted_probabilities = model.predict_proba(new_observation_standardized)
predicted_class, predicted_probabilities
```

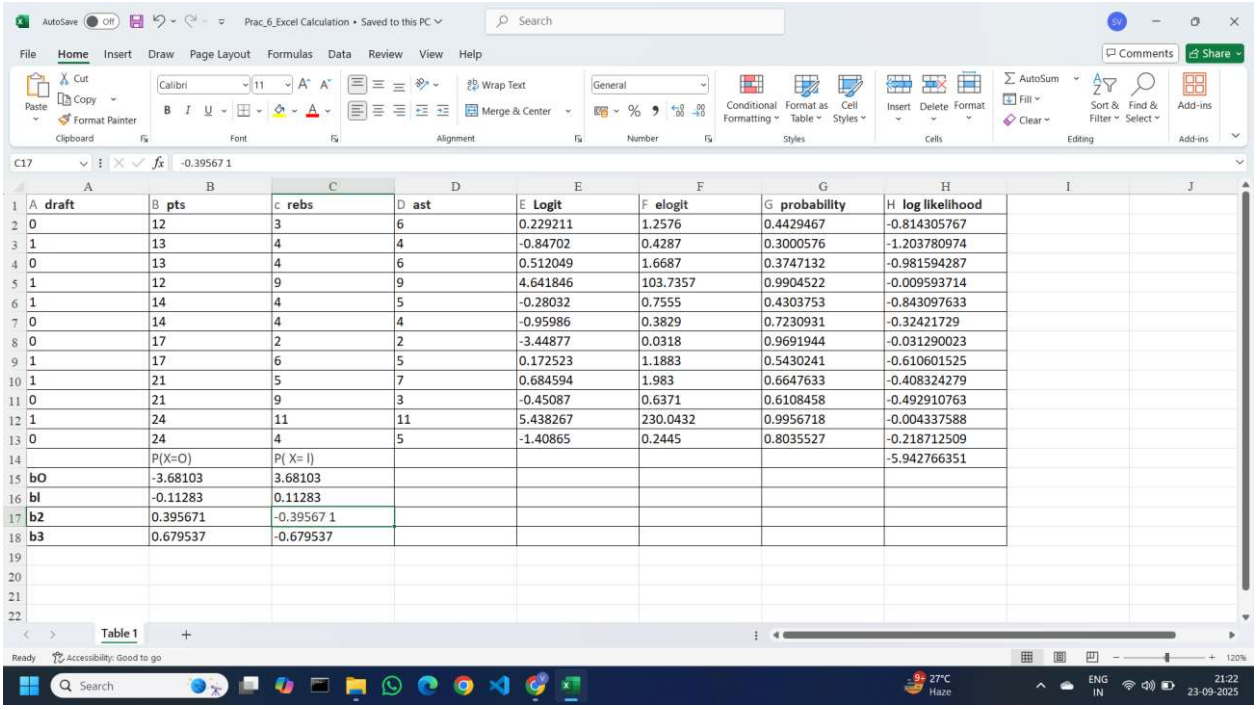
```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:125
  warnings.warn(
(array(['Iris-versicolor'], dtype=object),
 array([[4.67281620e-01, 5.32662082e-01, 5.62975656e-05]]))
```

C. Reducing Variance Through Regularization

```
import pandas as pd
from sklearn.linear_model import LogisticRegressionCV
from sklearn.preprocessing import StandardScaler

# Load data from CSV
iris_data = pd.read_csv('/content/sample_data/Iris.csv')

# Drop Id column if it exists
if "Id" in iris_data.columns:
    iris_data = iris_data.drop(columns=["Id"])

# Separate features and target
features = iris_data.iloc[:, :-1].values   # features
target = iris_data.iloc[:, -1].values      # target (species)

# Standardize features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
# Logistic regression with cross-validation and L2 regularization
logistic_regression = LogisticRegressionCV(
    penalty='l2',
    Cs=10,
    random_state=0,
    n_jobs=-1,
    multi_class="ovr"   # make it explicit for Iris dataset
)

# Train model
model = logistic_regression.fit(features_standardized, target)

# New observation (must also be standardized)
new_observation = [[0.5, 0.5, 0.5, 0.5]]
new_observation_standardized = scaler.transform(new_observation)

# Predict class
predicted_class = model.predict(new_observation_standardized)

# Predict probabilities
predicted_probabilities = model.predict_proba(new_observation_standardized)

predicted_class, predicted_probabilities
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logi
  warnings.warn(
(array(['Iris-versicolor'], dtype=object),
 array([[4.12227332e-01, 5.87772553e-01, 1.14658259e-07]]))
```

Perform logistic regression for a dataset that shows whether or not college basketball players got drafted into the NBA (draft: 0 = no, 1 = yes) based on their average points, rebounds, and assists in the previous season.



	A draft	B pts	C rebs	D ast	E Logit	F elogit	G probability	H log likelihood
2	0	12	3	6	0.229211	1.2576	0.4429467	-0.814305767
3	1	13	4	4	-0.84702	0.4287	0.3000576	-1.203780974
4	0	13	4	6	0.512049	1.6687	0.3747132	-0.981594287
5	1	12	9	9	4.641846	103.7357	0.9904522	-0.009593714
6	1	14	4	5	-0.28032	0.7555	0.4303753	-0.843097633
7	0	14	4	4	-0.95986	0.3829	0.7230931	-0.32421729
8	0	17	2	2	-3.44877	0.0318	0.9691944	-0.031290023
9	1	17	6	5	0.172523	1.1883	0.5430241	-0.610601525
10	1	21	5	7	0.684594	1.983	0.6647633	-0.408324279
11	0	21	9	3	-0.45087	0.6371	0.6108458	-0.492910763
12	1	24	11	11	5.438267	230.0432	0.9956718	-0.004337588
13	0	24	4	5	-1.40865	0.2445	0.8035527	-0.218712509
14		P(X=O)	P(X= I)					-5.942766351
15	bO	-3.68103	3.68103					
16	bl	-0.11283	0.11283					
17	b2	0.395671	-0.39567 1					
18	b3	0.679537	-0.679537					

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression

# Load data
data = pd.read_csv('/content/sample_data/Prac_6_nba.csv')

# Features and target
X = data[['Points', 'Rebounds', 'Assists']]
y = data['Draft']

# Fit logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Coefficients and intercept
intercept = model.intercept_[0]
coefficients = model.coef_[0]
print("Intercept:", intercept)
print("Coefficients (Points, Rebounds, Assists):", coefficients)
```

```
# Calculate logit (z)
logit = intercept + np.dot(X, coefficients)

# Calculate e-logit (exp(logit))
elogit = np.exp(logit)

# Calculate probability P(x=1)
prob_1 = 1 / (1 + np.exp(-logit))

# Calculate probability P(x=0)
prob_0 = 1 - prob_1

# Calculate log-likelihood
log_likelihood = np.sum(y * np.log(prob_1) + (1 - y) * np.log(prob_0))

# Combine everything in a DataFrame
results = X.copy()
results['Logit'] = logit
results['Exp(Logit)'] = elogit
results['P(Draft=1)'] = prob_1
results['P(Draft=0)'] = prob_0
results['Actual'] = y

print(results)
print("\nLog-Likelihood:", log_likelihood)
```

```
Intercept: -13.796234028748152
Coefficients (Points, Rebounds, Assists): [0.88942755 0.47032258 0.44624388]
   Points  Rebounds  Assists     Logit  Exp(Logit)  P(Draft=1)  P(Draft=0)  \
0    15.2       5.1      3.2  3.549690   34.802540    0.972069    0.027931
1     8.5       4.0      2.1 -3.417697    0.032788    0.031747    0.968253
2    12.1       6.3      4.0  1.713847    5.550273    0.847335    0.152665
3     7.5       3.5      1.5 -4.810033    0.008148    0.008082    0.991918
4    14.0       7.0      5.0  4.179229   65.315488    0.984921    0.015079
5     9.0       4.2      2.3 -2.789670    0.061441    0.057885    0.942115
6    16.5       5.5      3.8  5.161822  174.481992    0.994301    0.005699
7     6.8       3.0      1.8 -5.533920    0.003950    0.003935    0.996065
8    13.2       6.0      4.5  2.774243   16.026484    0.941268    0.058732
9    10.0       4.5      2.5 -1.669897    0.188266    0.158438    0.841562

   Actual
0       1
1       0
2       1
3       0
4       1
5       0
6       1
7       0
8       1
9       0
```

Colab file link: [Prac_6.ipynb]

Practical 7: Support Vector Machines

Sep 18, 2025

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks.

Objective: The goal of an SVM is to find the best boundary (or hyperplane) that separates data points of different classes. For classification, this boundary is chosen to maximize the margin between the classes.

How It Works

1. Hyperplane: In a 2D space, a hyperplane is a line that separates the classes. In higher dimensions, it's a plane or more complex surface.
2. Margin: The margin is the distance between the hyperplane and the closest data points from each class. SVM aims to maximize this margin.
3. Support Vectors: These are the data points closest to the hyperplane and are critical in defining the margin. They lie on the boundary of the margin.
4. Decision Function: For a given point, the SVM decision function will classify it based on which side of the hyperplane it falls on.

Types

1. Linear SVM: Used when classes are linearly separable (i.e., you can draw a straight line or hyperplane to separate the classes).
2. Non-Linear SVM: Used when classes are not linearly separable. It applies kernel functions (like the polynomial or radial basis function) to map data into a higher-dimensional space where a linear hyperplane can be used to separate the classes.

Kernel Trick

- Linear Kernel: No transformation, used for linearly separable data.
- Polynomial Kernel: Maps data into a higher-dimensional space using polynomial Functions.
- Radial Basis Function (RBF) Kernel: Maps data into an infinite-dimensional space, making it useful for complex decision boundaries.

Applications

- Text Classification: SVMs are often used for tasks like spam detection.
- Image Classification: They are effective in distinguishing between different types of Images.
- Bioinformatics: SVMs can classify proteins and genes.

Advantages

- Effective in high-dimensional spaces.
- Robust to overfitting, especially in high-dimensional space.

Disadvantages

- Can be computationally intensive for large datasets.
- The choice of kernel and parameters can be tricky.

A. Training a Linear Classifier

Importing Libraries

```
# Importing libraires
from sklearn.svm import LinearSVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
from matplotlib import pyplot as plt
```

Loading and Preprocessing Data

```
# load data with only two classes and features
iris = datasets.load_iris()
features = iris.data[:100, [0, 2]]
target = iris.target[:100]
```

Standardizing Features

```
# Standardizing Features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
print(features_standardized)
```

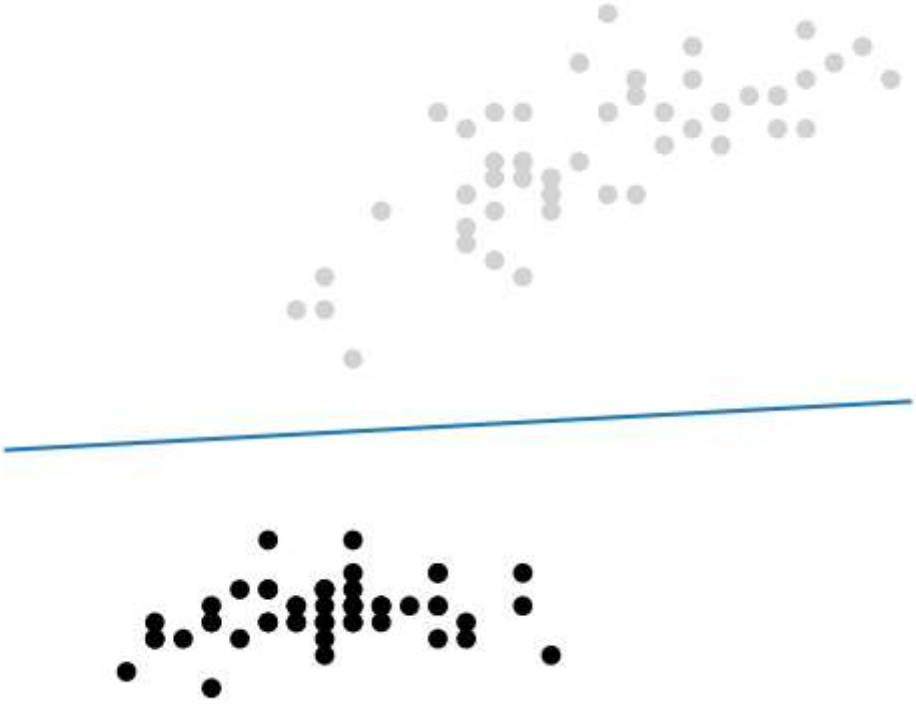
```
[-1.67741667 -1.08231219]
[-0.73768744 -0.87430856]
[-0.5810659  -0.66630494]
[-1.05093052 -1.01297765]
[-0.5810659  -0.87430856]
[-1.36417359 -1.01297765]
[-0.26782283 -0.94364311]
[-0.73768744 -1.01297765]
[ 2.39474331  1.27506221]
[ 1.45501408  1.13639313]
[ 2.23812177  1.4137313 ]
[ 0.04542025  0.78972042]
[ 1.61163562  1.20572767]
[ 0.35866332  1.13639313]
```

Creating and Training the Model

```
# create support vector classifier
svc = LinearSVC(C=1.0)
# train the model
model = svc.fit(features_standardized, target)
```

Plotting the Data and Hyperplane

```
import matplotlib.pyplot as plt
color = ["black" if c==0 else "lightgrey" for c in target]
plt.scatter(features_standardized[:, 0], features_standardized[:, 1], c=color)
w = svc.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
yy = a * xx - (svc.intercept_[0]) / w[1]
plt.plot(xx, yy)
plt.axis("off")
plt.show()
```



Making Predictions

```
new_observation = [[-2, 3]]
new_observation_standardized = scaler.transform(new_observation)
print(new_observation_standardized)
```

```
[[-11.70119506   0.09637501]]
```

```
predicted_class = model.predict(new_observation_standardized)
print(f"Predicted Class for New Obs {new_observation}: {predicted_class[0]}")
```

```
Predicted Class for New Obs [[-2, 3]]: 1
```

B. Handling Linearly Inseparable Classes Using Kernels

Dataset: Generate the random data in the range of 0 to 200

Module required: !pip install mlxtend

```
!pip install mlxtend
```

```
Requirement already satisfied:
```

Import necessary libraries

```
# import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from mlxtend.plotting import plot_decision_regions
```

Generate random data

```
# set random seed for repoductiblity.
np.random.seed(0)

# generate random data for two number
features = np.random.randn(200,2)
```

Create linearly inseparable classes using XOR logic

```
# use XOR logic to generate lineraly inseparable classes
target_xor = np.logical_xor(features[:,0] > 0, features[:,1] > 0)
target = np.where(target_xor, 0, 1)
```

Train SVM with RBF kernel

```
# create a support vector classifier with a aradial basisi function(RBF) kernel
svc_rbf = SVC(kernel='rbf',random_state=0, gamma=0.1, C=1)

# train the classifier with the RBF Kernel
svc_rbf.fit(features, target)
```

```
SVC
SVC(C=1, gamma=0.1, random_state=0)
```

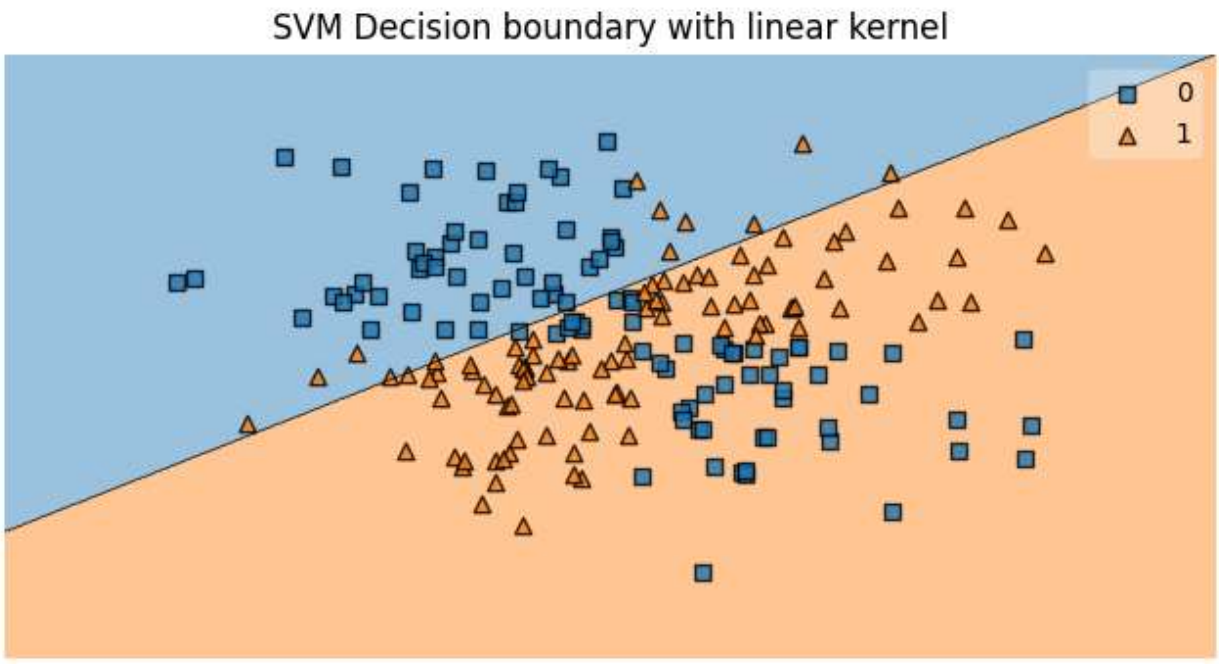
Train SVM with linear kernel

```
# create a support vector classifier with a linear kernel for comparison
svc_linear = SVC(kernel='linear',random_state=0,C=1)

# train the classifier with the RBF Kernel
svc_linear.fit(features, target)
```

```
SVC
SVC(C=1, kernel='linear', random_state=0)
```

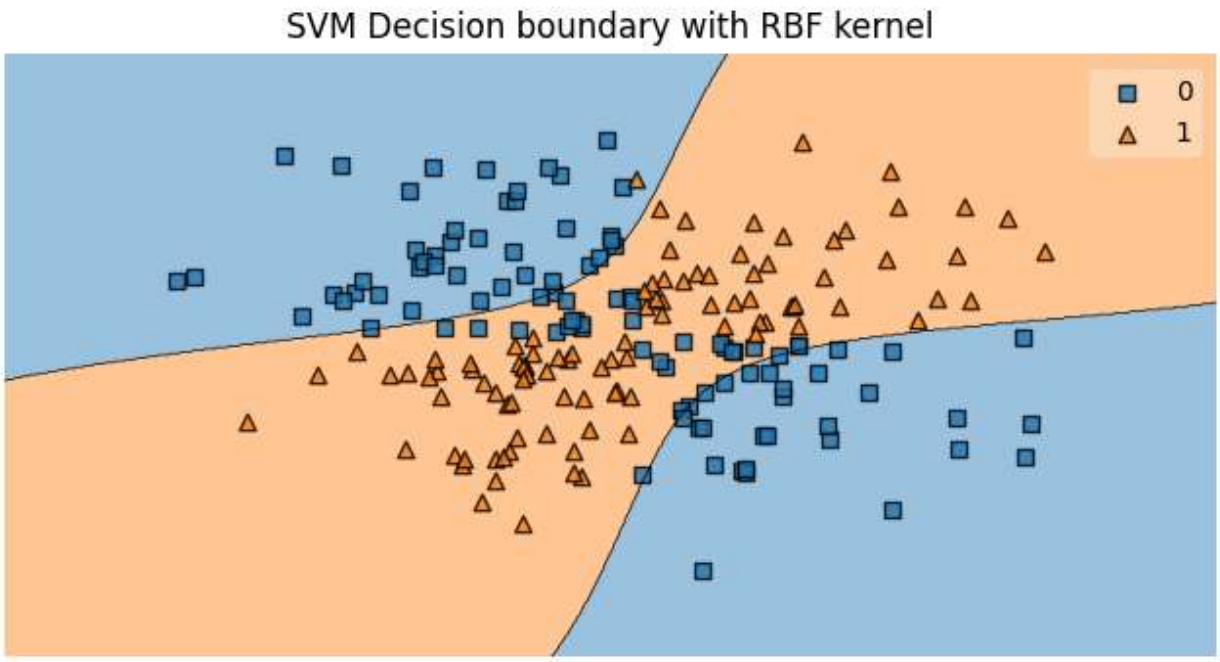
Plot decision boundaries for the linear kernel:

```
# plot decision boundary for the linear kernel
plt.figure(figsize=(8,4))
plot_decision_regions(features, target, clf=svc_linear)
plt.title("SVM Decision boundary with linear kernel")
plt.axis("off")
plt.show()
```



Plot decision boundaries for the RBF kernel

```
# plot decision boundary for the RBF Kernel
plt.figure(figsize=(8,4))
plot_decision_regions(features, target, clf=svc_rbf)
plt.title("SVM Decision boundary with RBF kernel")
plt.axis("off")
plt.show()
```



Colab file link: Prac_7.ipynb

Practical 8: Naive Bayes

Sep 18, 2025

Naive Bayes Classifier with an Example

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' Theorem. It assumes that the features used in classification are independent of each other (hence the term "naive"). This simplicity makes it efficient and useful for many tasks, particularly in text classification and spam filtering.

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

Example:SpamEmailClassification

Suppose we want to classify emails as "spam" or "not spam" based on their contents. Using Naive Bayes, we calculate the probability that an email is spam given the occurrence of certain words, like "free" or "money". By calculating the likelihood of each word occurring in spam and non-spam emails and using Bayes' Theorem, we can classify new emails based on their word content.

Note on Text Analysis

Text analysis is the process of converting unstructured textual data into structured data to gain meaningful insights. This is widely used in applications like sentiment analysis, text classification, information retrieval, and more. The goal of text analysis is to extract useful information from textual content and enable data-driven decisions. The techniques often involve natural language processing (NLP) methods such as tokenization, stemming, and lemmatization, followed by tasks like sentiment analysis or topic modeling.

Bag of Words Method

The Bag of Words (BoW) method is a fundamental technique used in text analysis for converting text into numerical features. It treats text data as a "bag" of words, disregarding grammar and word order, and focuses on word frequency. Each unique word in the dataset becomes a feature, and the value of the feature corresponds to the word's frequency in the text.

Steps in BoW:

1. Tokenization: Split text into individual words or tokens.
2. Vocabulary Creation: Create a set of unique words from the text.
3. Vectorization: Represent the text by creating a vector where each element corresponds to the count of a word from the vocabulary in the text.

For example, given the two sentences:

- Sentence 1: "I love cats."

- Sentence 2: "I love dogs."

The vocabulary would be: [I, love, cats, dogs]. The sentences would then be represented as vectors:

- Sentence 1: [1, 1, 1, 0] (word 'cats' appears once, 'dogs' does not appear)
- Sentence 2: [1, 1, 0, 1] (word 'dogs' appears once, 'cats' does not appear)

This method is simple but effective for many NLP tasks like document classification.

Bayesian Network

A Bayesian Network is a graphical model that represents a set of variables and their conditional dependencies using a directed acyclic graph (DAG). Each node in the graph represents a random variable, and the edges represent the probabilistic dependencies between these variables.

In a Bayesian Network, each node is associated with a probability distribution. The network captures the conditional independence relationships among the variables, which makes it useful for reasoning under uncertainty, particularly in complex domains like medical diagnosis, robotics, and genetics.

For example, a Bayesian Network might be used to model a medical diagnosis where the nodes represent symptoms, diseases, and test results. Given evidence about certain symptoms or test outcomes, the network can calculate the probabilities of different diseases.

Key Properties:

- Nodes: Represent random variables (e.g., symptoms, test results).
- Edges: Represent conditional dependencies (e.g., the presence of one symptom depends on a disease).
- Conditional Probability Tables (CPTs): Specify the probability of each node given its parents in the graph.

Bayesian Networks are widely used because they simplify the computation of complex probabilistic relationships and allow for efficient reasoning and inference.

Dataset: Prac_8_spam_ham_dataset.csv

A. Training a Classifier for Continuous Features,

```
#import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

```
#load the dataset
data = pd.read_csv("/content/sample_data/Prac_8_spam_ham_dataset.csv")
data.head()
```

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\nth...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\n(see a...	0
2	3624	ham	Subject: neon retreat\nho ho ho , we ' re arou...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\nthis deal is to ...	0

```
le = LabelEncoder()
data['label'] = le.fit_transform(data['label'])
data['text'] = le.fit_transform(data['text'])

print(data['label'])
print(data['text'])
```

```
0       0
1       0
2       0
3       1
4       0
       ..
5166    0
5167    0
5168    0
5169    0
5170    1
Name: label, Length: 5171, dtype: int64
0       1209
1       1985
2       2774
3       3158
4       3613
        ...
5166    3254
5167     125
5168     495
5169    2223
5170    2195
Name: text, Length: 5171, dtype: int64
```

```
x = data.drop('label', axis=1) #features
y = data['label'] #target variables.
```

```
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
nb_classifier = GaussianNB()
nb_classifier.fit(x_train,y_train)
```

```
GaussianNB
GaussianNB()
```

```
y_read = nb_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_read)
print(f"Accuracy of the naive baise classifier :{accuracy * 100:2f}%")
```

```
Accuracy of the naive baise classifier :100.000000%
```

```
comparison = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_read})
print(comparison.head(10))  # shows first 10 rows
```

```
   Actual  Predicted
0       0          0
1       1          1
2       0          0
3       0          0
4       0          0
5       0          0
6       0          0
7       0          0
8       0          0
9       0          0
```

B. Training a Naive Bayes Classifier for Binary Features

```
# Import necessary libraries
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# dataset: list of texts and corresponding labels
texts = [
    "I love this product",                          # positive
    "This is the worst product ever",               # negative
    "Amazing product, very satisfied",              # positive
    "I hate this product, very bad",                # negative
    "This product is okay",                         # positive
    "Terrible experience, would not recommend",     # negative
    "Absolutely fantastic, I will buy again",       # positive
    "Horrible quality, very disappointed",          # negative
    "Great value for the price",                    # positive
    "Not worth the money",                          # negative
    "Best purchase I've made",                      # positive
    "Awful product, broke after one use",           # negative
    "Really good, exceeded expectations",           # positive
    "I regret buying this",                         # negative
    "Superb quality and fast delivery",             # positive
    "Complete waste of money",                      # negative
    "Love it! Works perfectly",                     # positive
    "Doesn't work as advertised",                    # negative
    "Highly recommend this to everyone",            # positive
    "Worst experience I've had with a product"      # negative
]

# Corresponding labels for the text (1 = positive, 0 = negative)
labels = [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

```
# Step 1: Convert text data into bag of words features using CountVectorizer
vectorizer = CountVectorizer(lowercase=True)

# Transform the text data into a bag of words feature matrix
X = vectorizer.fit_transform(texts)

# Step 2: Split the data into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Step 3: Initialize the Naive Bayes classifier
classifier = MultinomialNB()

# Step 4: Train the classifier on the training data
classifier.fit(X_train, y_train)

# Step 5: Make predictions on the test data
y_pred = classifier.predict(X_test)

# Step 6: Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Optional: Show the predictions vs actual labels
print("Predicted labels:", y_pred)
print("Actual labels:", y_test)
```

```
Accuracy: 100.00%
Predicted labels: [1 0 0 0]
Actual labels: [1, 0, 0, 0]
```

```
# Train on all data
classifier.fit(X, labels)

# Test on new (unseen) example
test_text = ["This is a terrible product", 'This is an awesome car']
test_features = vectorizer.transform(test_text)
prediction = classifier.predict(test_features)

print(f"Predicted label: {prediction}")  # 0 = negative, 1 = positive
```

```
Predicted label: [0 1]
```

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(classifier, X, labels, cv=2)
print(f"Cross-validated accuracy: {scores.mean() * 100:.2f}%")
```

```
Cross-validated accuracy: 50.00%
```

Colab file link: [Prac_8.ipynb]

Practical 9: Clustering

Sep 19, 2025

---

1. Clustering Using K-Means

K-Means is a partition-based clustering algorithm that aims to divide a dataset into kkk clusters, where each data point belongs to the cluster with the nearest mean (centroid). The algorithm iterates between assigning data points to the nearest centroid and updating centroids based on the assigned points.

Advantages:

- Efficiency: K-Means is computationally efficient and scales well with large datasets.
- Simplicity: The algorithm is straightforward to understand and implement.
- Convergence: It usually converges quickly to a local minimum.

Disadvantages:

- Predefined Clusters: The number of clusters kkk needs to be specified in advance, which may not always be known.
- Sensitivity to Initialization: The final clusters can depend on the initial placement of centroids, leading to different results for different runs.
- Assumes Spherical Clusters: It assumes that clusters are spherical and equally sized, which may not be true for all datasets.

Speeding Up K-Means:

- K-Means++ Initialization: Improves the choice of initial centroids to help find a better solution and speed up convergence.
- Mini-Batch K-Means: Uses small random samples of the data to update centroids, which speeds up the process for large datasets.

2. Clustering Using Mean Shift

Mean Shift is a centroid-based clustering algorithm that iteratively shifts each data point to the average of data points in its neighborhood. It identifies clusters based on the density of data points and does not require specifying the number of clusters in advance.

Advantages:

- No Need for kkk: It does not require specifying the number of clusters beforehand.
- Flexible Shape: Can identify clusters of arbitrary shape and size.
- Density-Based: Works well with clusters of varying densities.

Disadvantages:

- Computational Complexity: Can be computationally expensive, especially for large datasets.
- Bandwidth Selection: Requires choosing a bandwidth parameter, which can significantly affect the results.

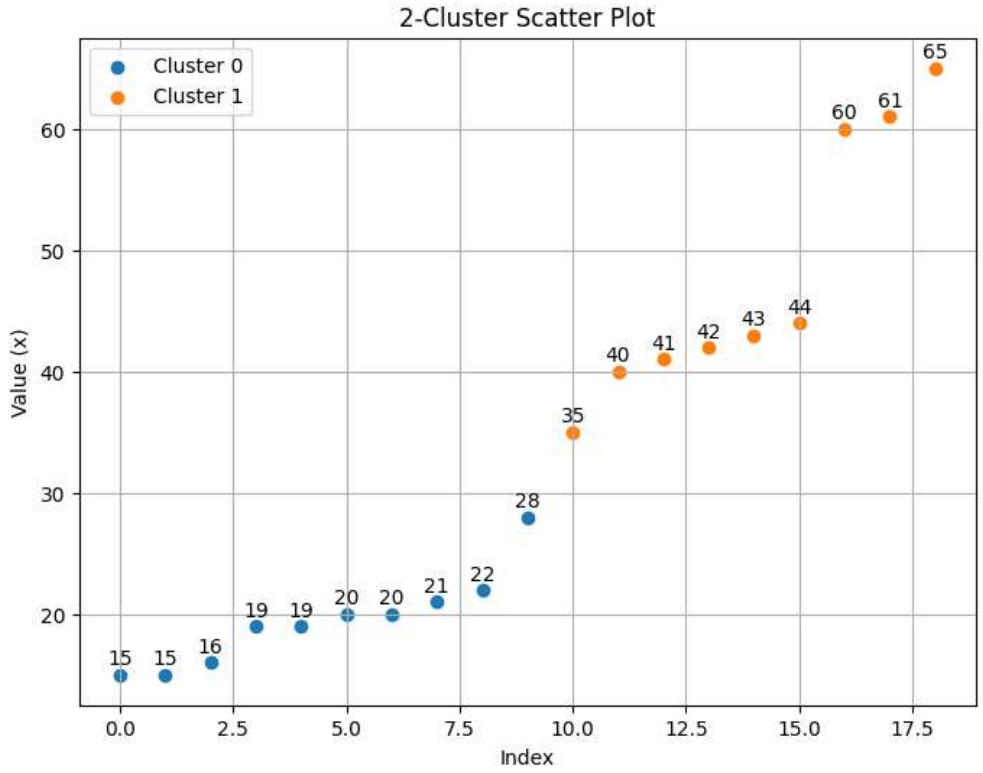
A. Clustering Using K-Means: Speeding Up K-Means Clustering

```
import pandas as pd
from sklearn.cluster import KMeans

# Create the DataFrame
features = pd.DataFrame(
    [15, 15, 16, 19, 19, 20, 20, 21, 22, 28, 35, 40, 41, 42, 43, 44, 60, 61, 65],
    columns=['x']
)

model = KMeans(n_clusters=2, random_state=19).fit(features) # Fit the KMeans model
predictions = model.labels_ # Get cluster predictions
features['cluster'] = predictions # Add predictions to the DataFrame

print(features) # Display the features with cluster
```

```
     x  cluster
0   15        0
1   15        0
2   16        0
3   19        0
4   19        0
5   20        0
6   20        0
7   21        0
8   22        0
9   28        0
10  35        1
11  40        1
12  41        1
13  42        1
14  43        1
15  44        1
16  60        1
17  61        1
18  65        1
```

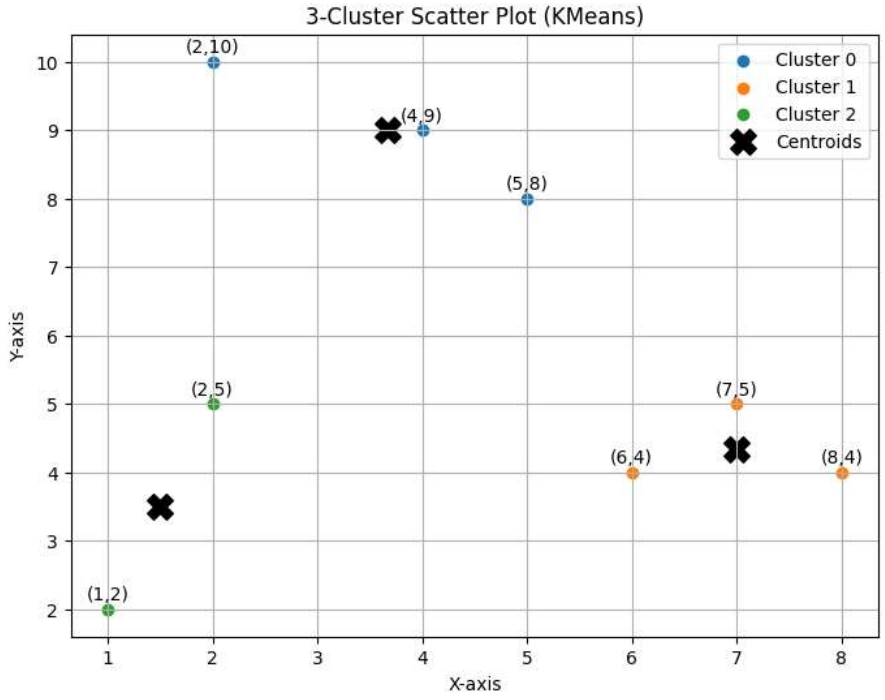
Excel Calulation Prac_9_Excel_Calculation.xlsx

```
import matplotlib.pyplot as plt

# Plot scatter chart
plt.figure(figsize=(8, 6))
for cluster in features['cluster'].unique():
    cluster_points = features[features['cluster'] == cluster]
    plt.scatter(cluster_points.index, cluster_points['x'], label=f'Cluster {cluster}')

# Add labels on points
for i, txt in enumerate(features['x']):
    plt.annotate(txt, (i, features['x'][i]), textcoords="offset points", xytext=(0, 5), ha='center')

plt.title("2-Cluster Scatter Plot")
plt.xlabel("Index")
plt.ylabel("Value (x)")
plt.legend()
plt.grid(True)
plt.show()
```



3 Cluster with Data point(X,Y)

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Data points (xi, yi) from your table
data = {
    'x': [4, 2, 5, 7, 6, 8, 2, 1],
    'y': [9, 10, 8, 5, 4, 4, 5, 2]
}
features = pd.DataFrame(data)

# Fit the KMeans model with 3 clusters
model = KMeans(n_clusters=3, random_state=19).fit(features)

# Add cluster predictions
features['cluster'] = model.labels_
# Plot scatter chart with cluster coloring
plt.figure(figsize=(8, 6))
for cluster in features['cluster'].unique():
    cluster_points = features[features['cluster'] == cluster]
    plt.scatter(cluster_points['x'], cluster_points['y'], label=f'Cluster {cluster}')

# Plot centroids
centroids = model.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=200, label='Centroids')

# Add labels to each point
for i, row in features.iterrows():
    plt.annotate(f"({row['x']},{row['y']})", (row['x'], row['y']), textcoords="offset points", xytext=(0, 5), ha='center')

plt.title("3-Cluster Scatter Plot (KMeans)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```




```
Centroid coordinates (x,y):
Cluster 0: (3.67, 9.00)
Cluster 1: (7.00, 4.33)
Cluster 2: (1.50, 3.50)
```

B. Clustering Using Meanshift

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MeanShift

# Load Iris dataset
iris = datasets.load_iris()
features = iris.data

# Standardize features
scaler = StandardScaler()
features_std = scaler.fit_transform(features)

cluster = MeanShift() # Create MeanShift model (without n_jobs)

model = cluster.fit(features_std) # Fit model

labels = model.labels_ # Get cluster labels
print(labels)
```

output

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

C. Clustering Using DBSCAN

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

# Load Iris dataset
iris = datasets.load_iris()
features = iris.data

# Standardize features
scaler = StandardScaler()
features_std = scaler.fit_transform(features)

# Create DBSCAN model (without n_jobs)
cluster = DBSCAN(eps=0.5, min_samples=5)  # you can adjust eps and min_samples if needed

model = cluster.fit(features_std) # Fit model

labels = model.labels_  # Get cluster labels
print(labels)
```

output

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0 -1 -1  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
  0  0  1  1  1  1  1  1 -1 -1  1 -1 -1  1 -1  1  1  1  1  1 -1  1  1  1
 -1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1 -1  1  1  1  1  1 -1  1  1
  1  1 -1  1 -1  1  1  1  1 -1 -1 -1 -1 -1  1  1  1  1 -1  1  1 -1 -1 -1
  1  1 -1  1  1 -1  1  1  1 -1 -1 -1  1  1  1 -1 -1  1  1  1  1  1  1  1
  1  1  1  1 -1  1]
```

D. Clustering Using Hierarchical Merging

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

# Load Iris dataset
iris = datasets.load_iris()
features = iris.data

# Standardize features
scaler = StandardScaler()
features_std = scaler.fit_transform(features)

# Create Agglomerative Clustering model
cluster = AgglomerativeClustering(n_clusters=3)

model = cluster.fit(features_std) # Fit model

labels = model.labels_ # Get cluster labels
print(labels)
```

output

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 2 1 1 1 1 1 1 1 0 0 0 2 0 2 0 2 0 2 2 0 2 0 2 0 2 2 2 2 0 0 0 0
 0 0 0 0 0 2 2 2 2 0 2 0 0 2 2 2 2 0 2 2 2 2 0 2 2 0 0 0 0 0 0 2 0 0 0 0
 0 0 0 0 0 0 0 0 2 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 0]
```

Colab file link: Prac_9.ipynb

Practical 10: Association rule learning

Sep 22, 2025

Apriori Algorithm

The Apriori algorithm is used for mining frequent itemsets and discovering association rules in transactional datasets. It is commonly used in market basket analysis to identify items that frequently occur together in transactions.

How It Works:

Generate Frequent Itemsets:

- Step 1: Scan the dataset to count the frequency of each item (1-itemsets) and keep those that meet the minimum support threshold.
- Step 2: Generate candidate itemsets of length kkk from the frequent itemsets
- of length k−lk−lk−l.
- Step 3: Scan the dataset again to count the frequency of these candidate
- itemsets.
- Step 4: Prune the itemsets that do not meet the minimum support threshold.
- Step 5: Repeat steps 2-4 until no more frequent itemsets can be generated.

Generate Association Rules:

- Calculate the confidence of these rules and retain the rules that meet the minimum confidence threshold.

Key Concepts:

- Support: The proportion of transactions that contain a particular itemset.
- Confidence: The probability that item BBB is purchased when item AAA is Purchased.
- Lift: The ratio of the observed support to the expected support if AAA and BBB were independent.

FP-Growth Algorithm

The FP-Growth algorithm is also used for mining frequent itemsets but is generally more efficient than the Apriori algorithm, especially for large datasets. It overcomes the computational inefficiency of generating candidate itemsets by using a more compact data str.

How It Works:

1. Build the FP-Tree:

Step 1: Scan the dataset to determine the frequency of each item and retain only the items that meet the minimum support threshold.

Step 2: Create the FP-Tree, a compressed tree structure where each node represents an item, and edges represent itemsets. The tree is built in a way that retains itemset counts and maintains itemset ordering.

2. Mine the FP-Tree:
Step 1: For each item in the FP-Tree, create its conditional pattern base, which consists of the subset of the database that contains that item.
Step 2: Build a conditional FP-Tree for each conditional pattern base.
Step 3: Recursively mine the conditional FP-Trees to find frequent itemsets.

Key Concepts:
FP-Tree: A compact representation of the dataset that stores itemsets in a way that avoids generating candidate itemsets.
Conditional FP-Tree: A tree built from the subset of data related to a specific item, used for finding frequent itemsets that include that item.

Advantages of FP-Growth Over Apriori:

- Efficiency: FP-Growth avoids generating a large number of candidate itemsets, making it faster and more scalable.
- Compact Data Structure: The FP-Tree reduces the size of the data needed to be processed, which can lead to faster computations

Ref: Market Basket Analysis (Apriori, Eclat, FP-Growth)

A. Association Rule Learning

```
import pandas as pd

# Create the transaction dataframe
data = {
    "TID" : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Milk": [1, 0, 0, 1, 0, 1, 1, 1, 0, 1],
    "Apple": [1, 1, 0, 1, 1, 0, 0, 1, 1, 1],
    "Butter": [1, 0, 0, 0, 1, 1, 1, 0, 0, 1],
    "Bread": [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
    "Orange": [1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
    "Spinach": [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]
}

df = pd.DataFrame(data)
print(df)
```

	TID	Milk	Apple	Butter	Bread	Orange	Spinach
0	1	1	1	1	1	1	1
1	2	0	1	0	1	1	1
2	3	0	0	0	1	0	0
3	4	1	1	0	0	1	1
4	5	0	1	1	1	1	1
5	6	1	0	1	1	1	0
6	7	1	0	1	0	0	0
7	8	1	1	0	1	1	0
8	9	0	1	0	0	1	0
9	10	1	1	1	1	1	0

```
# Compute the total count of transactions in the dataset
txn_count = len(df)
txn_count
```

```
10
```

```
# Get the list of item names, excluding TID
items = df.columns.drop('TID')
items
```

```
Index(['Milk', 'Apple', 'Butter', 'Bread', 'Orange', 'Spinach'], dtype='object')
```

```
# Initialize an empty list to store support values
support_values = []

# Calculate support for each item
for itemset in items:
    support_value = df[itemset].sum() / txn_count
    support_values.append((itemset, support_value))

# Convert the results into a dataframe
df_support = pd.DataFrame(support_values, columns=['Itemset', 'Support'])

# Display the results
print(df_support)
```

```
   Itemset  Support
0     Milk      0.6
1    Apple      0.7
2   Butter      0.5
3    Bread      0.7
4   Orange      0.8
5  Spinach      0.4
```

```python
# Finding the frequency of itemsets containing two items
# Import combinations function from itertools to generate all possible combinations
from itertools import combinations

# Compute the support for each itemset of size 2
combo2_support_values_list = []
for combo in combinations(items, 2):
    combo_support = df[list(combo)].all(axis=1).mean()
    combo2_support_values_list.append((list(combo), combo_support))

# Convert the results into a dataframe
df_combo2_support = pd.DataFrame(combo2_support_values_list, columns=['Itemset', 'Support'])

# Display the results
print(df_combo2_support)
```

```
              Itemset  Support
0       [Milk, Apple]      0.4
1      [Milk, Butter]      0.4
2       [Milk, Bread]      0.4
3      [Milk, Orange]      0.5
4     [Milk, Spinach]      0.2
5     [Apple, Butter]      0.3
6      [Apple, Bread]      0.5
7     [Apple, Orange]      0.7
8    [Apple, Spinach]      0.4
9     [Butter, Bread]      0.4
10   [Butter, Orange]      0.4
11  [Butter, Spinach]      0.2
12    [Bread, Orange]      0.6
13   [Bread, Spinach]      0.3
14  [Orange, Spinach]      0.4
```

```python
# Define minimum support threshold
min_support_threshold = 0.4

# Drop itemsets with support less than or equal to 0.4
df_combo2_support = df_combo2_support[df_combo2_support['Support'] >= min_support_threshold]

# Display the results
print(df_combo2_support)
```

```
              Itemset  Support
0       [Milk, Apple]      0.4
1      [Milk, Butter]      0.4
2       [Milk, Bread]      0.4
3      [Milk, Orange]      0.5
6      [Apple, Bread]      0.5
7     [Apple, Orange]      0.7
8    [Apple, Spinach]      0.4
9     [Butter, Bread]      0.4
10   [Butter, Orange]      0.4
12    [Bread, Orange]      0.6
14  [Orange, Spinach]      0.4
```

```
# Establish Association Rules

lift_values = []
# Calculate lift for two-item itemsets
for index, row in df_combo2_support.iterrows():
    if isinstance(row['Itemset'], list) and len(row['Itemset']) == 2:
        item1, item2 = row['Itemset']
        support_item1 = df_support[df_support['Itemset'] == item1]['Support'].values[0]
        support_item2 = df_support[df_support['Itemset'] == item2]['Support'].values[0]
        lift = (row['Support']) / ((support_item1)*(support_item2))
        lift_values.append({'Rule': f'[{item1}, {item2}]', 'Lift': lift})

# Convert the results into a DataFrame
df_lift = pd.DataFrame(lift_values, columns=['Rule', 'Lift'])

# Display the results
print(df_lift)
```

```
                 Rule      Lift
0       [Milk, Apple]  0.952381
1      [Milk, Butter]  1.333333
2       [Milk, Bread]  0.952381
3      [Milk, Orange]  1.041667
4      [Apple, Bread]  1.020408
5     [Apple, Orange]  1.250000
6    [Apple, Spinach]  1.428571
7     [Butter, Bread]  1.142857
8    [Butter, Orange]  1.000000
9     [Bread, Orange]  1.071429
10  [Orange, Spinach]  1.250000
```

Interpretation of Lift Values

- Lift > 1: Positive association — items are more likely to be bought together than expected by chance.
- Lift = 1: No association — items are bought together exactly as often as random chance.
- Lift < 1: Negative association — items are less likely to be bought together than expected.

6 [Apple, Spinach] 1.428571

since the lift of Apple and spinach is maximum then it more frequent to be brought together

	Rule	Confidence
0	Milk → Apple	0.666667
1	Apple → Milk	0.571429
2	Milk → Butter	0.666667
3	Butter → Milk	0.800000
4	Milk → Bread	0.666667
5	Bread → Milk	0.571429
6	Milk → Orange	0.833333
7	Orange → Milk	0.625000
8	Apple → Bread	0.714286
9	Bread → Apple	0.714286
10	Apple → Orange	1.000000
11	Orange → Apple	0.875000
12	Apple → Spinach	0.571429
13	Spinach → Apple	1.000000
14	Butter → Bread	0.800000
15	Bread → Butter	0.571429
16	Butter → Orange	0.800000
17	Orange → Butter	0.500000
18	Bread → Orange	0.857143
19	Orange → Bread	0.750000
20	Orange → Spinach	0.500000
21	Spinach → Orange	1.000000

10 Apple → Orange 1.0000 || 13 Spinach → Apple 1.0000 || 21 Spinach → Orange 1.0000

	Rule	Lift
0	[Milk, Apple]	0.952381
1	[Milk, Butter]	1.333333
2	[Milk, Bread]	0.952381
3	[Milk, Orange]	1.041667
4	[Apple, Bread]	1.020408
5	[Apple, Orange]	1.250000
6	[Apple, Spinach]	1.428571
7	[Butter, Bread]	1.142857
8	[Butter, Orange]	1.000000
9	[Bread, Orange]	1.071429
10	[Orange, Spinach]	1.250000

B. Apriori algorithm

```
# Install machine learning extensions (mlxtend) library
!pip install mlxtend
```

```
# Load dataset and view first 5 rows
import numpy as np
import pandas as pd
import warnings

df = pd.read_csv(
    "/content/sample_data/Prac_10_Market_Basket_Optimisation_Modified.csv",
    header=None,
    index_col=None,
    names=[f"Item_{i+1}" for i in range(5)]
    )
df.head()
```

	Item_1	Item_2	Item_3	Item_4	Item_5
0	shrimp	almonds	avocado	vegetables mix	green grapes
1	mineral water	milk	energy bar	whole wheat rice	green tea
2	turkey	burgers	mineral water	eggs	cooking oil
3	shrimp	chocolate	chicken	honey	oil
4	turkey	fresh tuna	tomatoes	spaghetti	mineral water

```
# View shape of the dataset
print(f"The dataset contains {df.shape[0]} transactions and each transaction has a maximum of {df.shape[1]} items or less.")
```

```
The dataset contains 3345 transactions and each transaction has a maximum of 5 items or less.
```

```
# Generate transaction lists
txns = df.fillna("").values.tolist()
txns = [[item for item in txn if item != ''] for txn in txns]
txns = [[item.strip() for item in txn] for txn in txns]

# Create a list of unique ids for the transactions
ids = [i + 1 for i in range(len(txns))]

# Initialize an empty list
data =[]
# Iterate through transactions and add them to the DataFrame with IDs
for i, txn in enumerate(txns):
    data.extend([{'TID': ids[i], 'Item': item} for item in txn])

df_txn = pd.DataFrame(data)
print(df_txn.head(20))
```

```
    TID             Item
0     1           shrimp
1     1          almonds
2     1          avocado
3     1   vegetables mix
4     1     green grapes
5     2    mineral water
6     2             milk
7     2       energy bar
8     2  whole wheat rice
9     2        green tea
10    3           turkey
11    3          burgers
12    3    mineral water
13    3             eggs
14    3      cooking oil
15    4           shrimp
16    4        chocolate
17    4          chicken
18    4            honey
19    4              oil
```

```
# Perform one hot encoding using TransactionEncoder
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder() # Create a TransactionEncoder

# Fit and transform the transaction data
te_array = te.fit(txns).transform(txns)
te_columns = te.columns_ # Extract the column names

# Create a DataFrame from the one-hot encoded array
df1 = pd.DataFrame(te_array, columns=te.columns_)

df1.head() # Display the results
```

	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	body spray	bramble	brownies	bug spray	burger sauce
0	True	False	False	True	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False

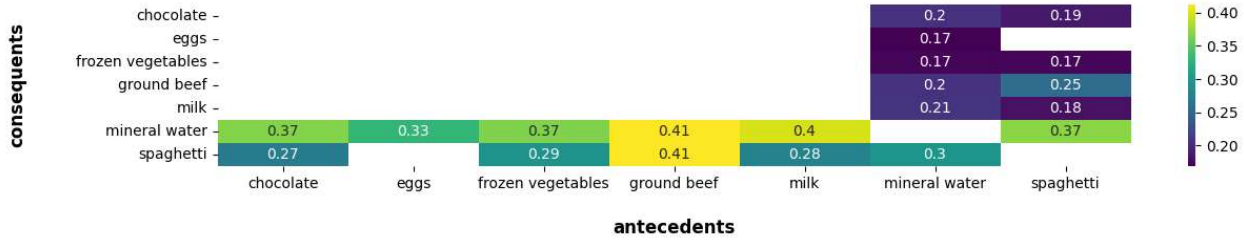
```
# Generate frequent itemsets
from mlxtend.frequent_patterns import apriori

# Applying Apriori algorithm assuming an item has to appear
#  in at least 4% of the total transaction to be considered as frequent
min_support_threshold = 0.05
frequent_itemsets = apriori(df1, min_support=min_support_threshold, use_colnames=True)
print(frequent_itemsets)
```

```
     support                itemsets
0   0.051420                (avocado)
1   0.144993                (burgers)
2   0.085501                   (cake)
3   0.072347                (chicken)
4   0.206876              (chocolate)
5   0.053214            (cooking oil)
6   0.191031                   (eggs)
7   0.054709                (escalope)
8   0.145291             (french fries)
9   0.175785          (frozen vegetables)
10  0.091181             (grated cheese)
11  0.098655                (green tea)
12  0.185052              (ground beef)
13  0.088789             (herb & pepper)
14  0.194021                      (milk)
15  0.372795             (mineral water)
```

```
# Generate association rules from the frequent itemsets assuming the likelihood of purchasing the antecedent,
# followed by the consequent has to be at least 30% to be considered significant or interesting
from mlxtend.frequent_patterns import association_rules
confidence_threshold = 0.05
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence_threshold)

# Sorting rules by confidence, support, and lift
sorted_rules = rules.sort_values(['confidence', 'support', 'lift'], ascending=[False, False, False])
sorted_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard
10	(ground beef)	(mineral water)	0.185052	0.372795	0.076233	0.411955	1.105043	1.0	0.007247	1.066593	0.116643	0.158287
12	(ground beef)	(spaghetti)	0.185052	0.298954	0.075934	0.410339	1.372585	1.0	0.020612	1.188898	0.333086	0.186081
14	(milk)	(mineral water)	0.194021	0.372795	0.077130	0.397535	1.066362	1.0	0.004800	1.041064	0.077213	0.157509
18	(spaghetti)	(mineral water)	0.298954	0.372795	0.110613	0.370000	0.992502	1.0	-0.000836	0.995563	-0.010661	0.197123
1	(chocolate)	(mineral water)	0.206876	0.372795	0.075934	0.367052	0.984594	1.0	-0.001188	0.990926	-0.019346	0.150742
6	(frozen vegetables)	(mineral water)	0.175785	0.372795	0.064275	0.365646	0.980823	1.0	-0.001257	0.988730	-0.023172	0.132716
4	(eggs)	(mineral water)	0.191031	0.372795	0.063079	0.330203	0.885750	1.0	-0.008136	0.936411	-0.137519	0.125970
19	(mineral water)	(spaghetti)	0.372795	0.298954	0.110613	0.296712	0.992502	1.0	-0.000836	0.996813	-0.011902	0.197123
8	(frozen vegetables)	(spaghetti)	0.175785	0.298954	0.051719	0.294218	0.984158	1.0	-0.000833	0.993290	-0.019156	0.122261

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a heatmap to visualize the relationships between the antecedents and consequents
# Convert frozensets to strings and remove 'frozenset' from the representation
rules['antecedents'] = rules['antecedents'].apply(lambda x: ', '.join(list(x)))
rules['consequents'] = rules['consequents'].apply(lambda x: ', '.join(list(x)))

# Show frequency or strength of item associations
pivot_table = rules.pivot(index="consequents", columns="antecedents", values="confidence")
plt.figure(figsize=(14, 2))
sns.heatmap(pivot_table, annot=True, cmap="viridis")
plt.xlabel("\nantecedents", fontsize=12, fontweight=600)
plt.ylabel("consequents\n\n", fontsize=12, fontweight=600)
plt.gca().set_facecolor("white")
plt.show()
```


consequents
chocolate
eggs
frozen vegetables
ground beef
milk
mineral water
spaghetti
0.2
0.19
0.17
0.17
0.17
0.2
0.25
0.21
0.18
0.37
0.37
0.33
0.37
0.41
0.4
0.37
0.27
0.29
0.41
0.28
0.3
chocolate
eggs
frozen vegetables
ground beef
milk
mineral water
spaghetti
antecedents
0.40
0.35
0.30
0.25
0.20