```
from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/DataSets/nyc_weather.csv')

df

#!pip install -q xlrd

import pandas as pd

df1 = pd.read_excel('/content/drive/MyDrive/DataSets/nyc_weather.xlsx')

df1

import pandas as pd

df2=pd.read_xml('/content/drive/MyDrive/DataSets/nyc_weather.xml')

df2

import pandas as pd

df3=pd.read_json('/content/drive/MyDrive/DataSets/nyc_weather.json')

df3

# Conversion of Dataframe to .CSV File

a = {'Name' : ['Harish','Greeshma','Maheshwar','Mohit'], 'Age' : [20,19,19,20]}

b = pd.DataFrame(a)

b.to_csv('a.csv')

print("Dataframe converted to .CSV file Successfully.\nA file named a.csv is created in the Current Working Directory.")

# Conversion of Dataframe to .XLSX File

a = {'Name' : ['Harish','Greeshma','Maheshwar','Mohit'], 'Age' : [20,19,19,20]}

b = pd.DataFrame(a)

b.to_excel('a.xlsx')

print("Dataframe converted to .xlsx file Successfully.\nA file named a.xlsx is created in the Current Working Directory.")

#Using the group_by Method

g=df.groupby('Temperature')  print(g)

g.first()

g.groups
g.get_group(43)
```

```
df['Temperature'].max()
df['Temperature'].mean()
df.describe()
print(df.min())
```

2.write a program that extracts the words (features) used in a sentence.

```
# Tokenisation of Words
import nltk
nltk.download('punkt')
a = input("Enter any sentence: ")
tokens = nltk.word_tokenize(a)
tokens
# Tokenisation of Sentences
nltk.download('punkt')
a = input("Enter any sentence: ")
tokens = nltk.sent_tokenize(a)
tokens
```

3.edge detection(nitin)

```
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread('rrr.jpg')
# Convert image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(gray, (3, 3), 0)
edges = cv2.Canny(blurred, 50, 150)
# Show the output image
cv2_imshow(edges)
cv2.waitKey(0)
```

4.surf/sift

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
///from google.colab import drive
drive.mount('/content/drive')
///img = cv2.imread('/content/drive/MyDrive/img1.png')
plt.imshow(img)
///img.shape
#Harris Corner Detection
img1 = img
gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)
dst = cv2.dilate(dst, None)
img1[dst > 0.08 * dst.max()] = [0, 255, 0]
plt.imshow(img1)
#Scale Invariant Feature Transform (SIFT) Feature Detection
img2 = img
gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT_create()
kp, des = sift.detectAndCompute(gray,None)
var = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
img2 = cv2.drawKeypoints (gray, kp, img2, var)
plt.imshow(img2)
```

6.Pca(s)

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Initialize the PCA model with two components
pca = PCA(n_components=2)

# Fit the model to the data and transform the data onto the first two principal components
X_pca = pca.fit_transform(X)

# Plot the transformed data points
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Iris Dataset after PCA')
plt.show()
```

Week 7 Linear discriminant analysis(s)

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Load the iris dataset and extract only the first two classes for binary classification
iris = load_iris()
X = iris.data[:100, :2]  # only first two features
y = iris.target[:100]

# Initialize the LDA model
lda = LinearDiscriminantAnalysis()

# Fit the model to the datap
lda.fit(X, y)

# Predict the class labels for new data
new_X = [[5.1, 3.5], [7.0, 3.2], [6.3, 3.3]]
predicted_y = lda.predict(new_X)
print(predicted_y)
```

Week 8linear Regression

```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/drive/MyDrive/DataSets/cars.csv')
df
///df.describe()
///plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),cmap = "YlGnBu",annot=True)
///from sklearn.model_selection import train_test_split
X = df.drop('selling_price', axis=1)
Y = df['selling_price']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
///from sklearn.linear_model import LinearRegression
# created the model
model = LinearRegression()
# training of the model
model.fit(X_train, y_train)
///# predicting on the test data.
y_pred = model.predict(X_test)
y_pred[:15].round(decimals=2)
///# actual price of test/new cars.
y_test[:15]
///# score on training data
model.score(X_train, y_train)
///# parameters
model.coef_.round(2)
///model.intercept_s
```

```
///# metrics
from sklearn.metrics import mean_squared_error as mserror
from sklearn.metrics import r2_score
from math import sqrt
mse = mserror(y_test, y_pred)
print(mse) # mean squared error
print(sqrt(mse)) # root mean squared error
print(r2_score(y_test, y_pred))
```

5. Univariate, Multivariate analysis, Visualization using correlation matrix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
iris_data = load_iris()
df = pd.DataFrame(data=iris_data.data,
columns=iris_data.feature_names)
df.head()
df['target'] = pd. Series(iris_data. target)
df
df_0=df.loc[df['target']==0]
df_1=df.loc[df['target']==1]
df_2=df.loc[df['target']==2]
//univariate
plt.plot(df_0['sepal length (cm)'],np.zeros_like(df_0['sepal length (cm)']),'o')
plt.plot(df_1['sepal length (cm)'],np.zeros_like(df_1['sepal length (cm)']),'+')
plt.plot(df_2['sepal length (cm)'],np.zeros_like(df_2['sepal length (cm)']),'-')
print(df.describe())
sns.histplot(data=df, x='sepal length (cm)', bins=10)
sns.histplot(data=df, x='sepal width (cm)', bins=10)
sns.histplot(data=df, x='petal length (cm)', bins=10)
sns.histplot(data=df, x='petal width (cm)', bins=10)
sns.boxplot(data=df, x='sepal length (cm)')
sns.boxplot(data=df, x='sepal width (cm)')
sns.boxplot(data=df, x='petal length (cm)')
sns.boxplot(data=df, x='petal width (cm)')
//multivariate
sns.FacetGrid(df,hue="target",height=5).map(
```

```
plt.scatter,"sepal length (cm)","petal length (cm)"
).add_legend()
plt.show()
sns.pairplot(df,hue="target",diag_kind="hist",height=3).add_legend()
plt.show()
//visualization using correlation matrix
sns.heatmap(df.corr(),annot=True)
```

12. SVM to classify iris.print correct and wrong predictions

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
i = datasets.load_iris()
df = pd.DataFrame(i['data'],columns=i['feature_names'])
df
df.info()
df.describe()
sns.heatmap(df.corr(),cmap = "Greens",annot=True)
X,Y = i.data, i.target
from sklearn import preprocessing
X = preprocessing.normalize(X)
X = pd.DataFrame(X,columns=i['feature_names'])
from sklearn.model_selection import train_test_split
X_train , X_test , Y_train ,Y_test = train_test_split (X, Y, test_size=0.3,
random_state=10)
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
svc = make_pipeline(StandardScaler(),SVC(gamma='auto'))
svc.fit(X_train,Y_train)
Y_pred = svc.predict(X_test)
from sklearn.metrics import accuracy_score
print('Accuracy Score:',accuracy_score(Y_test,Y_pred))
j=0
correct=[]
```

```
false=[]
for i in range(len(Y_test)):
  if(Y_test[i]==Y_pred[j]):
    correct.append(X_test.iloc[j])
  else:
    false.append(X_test.iloc[j])
  j=j+1
correct = pd.DataFrame(correct)
correct
```

13.Regression tree for cost estimation by assuming any numerical dataset.

```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/drive/MyDrive/DataSets/cars.csv')
df
df.describe()
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),cmap = "YlGnBu",annot=True)
from sklearn.model_selection import train_test_split
X = df.drop('selling_price', axis=1)
Y = df['selling_price']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth=5)
dtr.fit(X_train, y_train)
y_pred = dtr.predict(X_test)
y_pred[:15].round(decimals=2)
y_test[:15]
print("Model Score on Training Data =",dtr.score(X_train, y_train))
print("Model Score on Test Data =",dtr.score(X_test, y_test))
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(40,20))
a = plot_tree(dtr, feature_names=X.columns, class_names=Y, filled=True,
rounded=True, fontsize=14)
c = df.corr()
f = c[abs(c['selling_price']) > 0.3].index.tolist()
```

```
f.remove('selling_price')
df1 = df[f + ['selling_price']]
df1.head()
a = df1.drop('selling_price',axis=1)
b = df1['selling_price']
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.3, random_state=1)
r = DecisionTreeRegressor(max_depth = 5)
d = r.fit(a_train,b_train)
print("Model Score on Training Data =",d.score(a_train,b_train))
print("Model Score on Test Data =",d.score(a_test,b_test))
b_pred = d.predict(a_test)
plt.scatter(b_test, b_pred)
plt.xlabel("Actual Selling Price")
plt.ylabel("Predicted Selling Price")
plt.title("Actual vs. Predicted Selling Price")
plt.show()
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(b_test, b_pred)
print("Mean Squared Error:", mse)
```

15.single neural network and test for different logic gates

```
import tensorflow as tf
import numpy as np
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y_and = np.array([[0], [0], [0], [1]])
Y_or = np.array([[0], [1], [1], [1]])
Y_xor = np.array([[0], [1], [1], [0]])
model = tf.keras.Sequential([ tf.keras.layers.Dense(2, input_shape=(2,),
activation='relu'), tf.keras.layers.Dense(1, activation='sigmoid') ])
print("Training model for AND gate...")
model.fit(X, Y_and, epochs=1000, verbose=0)
print("AND gate output:", np.round(model.predict(X)))
print("Training model for OR gate...")
model.fit(X, Y_or, epochs=1000, verbose=0)
print("OR gate output:", np.round(model.predict(X)))
print("Training model for XOR gate...")
model.fit(X, Y_xor, epochs=1000, verbose=0)
print("XOR gate output:", np.round(model.predict(X)))
X = np.array([[0], [1]])
Y_not = np.array([[1], [0]])
model1 = tf.keras.Sequential([tf.keras.layers.Dense(1, input_shape=(1,),
activation='sigmoid')])
model1.compile(loss='binary_crossentropy', optimizer='adam')
print("Training model for NOT gate...")
model1.fit(X, Y_not, epochs=1000, verbose=0)
print("NOT gate output:", np.round(model1.predict(X)))
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('https://storage.googleapis.com/download.tensorflow.org/data/iris_training.csv')
df
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values
from tensorflow.keras.utils import to_categorical
Y = to_categorical(Y)
model = keras.Sequential([
 layers.Dense(10, activation='relu', input_shape=[4]),
 layers.Dense(8, activation='relu'),
 layers.Dense(3, activation='softmax')
])
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
tf.keras.backend.clear_session()
history = model.fit(X, Y, epochs=50, batch_size=10, validation_split=0.2)
df1 = pd.read_csv('https://storage.googleapis.com/download.tensorflow.org/data/iris_test.csv')
X_test = df1.iloc[:, :-1].values
Y_test = df1.iloc[:, -1].values
Y_test = to_categorical(Y_test)
model.evaluate(X_test, Y_test)
```

Week16 ann

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Convert the target variable to one-hot encoded labels
y = to_categorical(y)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Scale the data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = Sequential()
# Add a fully connected layer with 10 neurons, using ReLU activation
model.add(Dense(10, input_dim=4, activation='relu'))
# Add an output layer with 3 neurons, using softmax activation
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train the model on the scaled training data for 50 epochs
model.fit(X_train_scaled, y_train, epochs=50, batch_size=10)
```