Pr1 Implementation of Simple Linear Regression.

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

data = {'Hours': [1, 2, 4, 5, 6],
        'Marks': [50, 60, 80, 90, 100]}
df = pd.DataFrame(data)
print(df)

X = df[['Hours']].values
Y = df[['Marks']].values



model = LinearRegression()
model.fit(X, Y)



slope = model.coef_[0][0]
intercept = model.intercept_[0]
print(f"Equation of line: y = {slope:.2f}X + {intercept:.2f}")
Step 6: Predict values and visualize

df['Predicted Marks'] = model.predict(X)  # Add predictions to DataFrame
plt.scatter(df['Hours'], df['Marks'], color='blue', label='Actual Marks')
plt.plot(df['Hours'], df['Predicted Marks'], color='red', label='Regression Line')
plt.xlabel('Hours Studied')
plt.ylabel('Marks Scored')
plt.title('Linear Regression: Hours vs Marks')
plt.legend()
```

```
plt.show()


hours_to_predict = np.array([[3]])
predicted_marks = model.predict(hours_to_predict)[0][0]
print(f"Predicted marks for 3 hours of study: {predicted_marks}")
```

Pr2 Implementation of Multiple Linear Regression.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression



data = {
    'Area': [2600, 3000, 3200, 3600, 4000, 4100],
    'Bedroom': [3, 4, NaN, 3, 5, 6],
    'Age': [20, 15, 18, 30, 8, 8],
    'Price': [550000, 565000, 610000, 595000, 760000, 810000]
}



df = pd.DataFrame(data)



print("Before filling NaN:\n", df)
df.fillna(0, inplace=True)
print("\nAfter filling NaN with 0:\n", df)
```

```
sns.heatmap(df.corr(), annot=True)
plt.title("Feature Correlation Heatmap")
plt.show()


sns.relplot(x='Area', y='Price', data=df)
plt.title("Area vs Price")
plt.show()

X = df[['Area', 'Bedroom', 'Age']]

y = df['Price']


model = LinearRegression()
model.fit(X, y)


print("\nIntercept:", model.intercept_)
print("Coefficients:", model.coef_)

predicted_price = model.predict([[3000, 3, 15]])
print("\nPredicted Price for Area=3000, Bedroom=3, Age=15:", predicted_price[0])
```

Pr3 Implementation of Logistic Regression

```
modeling
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv('titanic-training-data.csv')

df.sample(10)

df.dtypes

df.info()

df.isnull().sum()

sns.countplot(x='Survived', data=df)

sns.countplot(x='Survived', hue='Sex', data=df)

pd.crosstab(df['Survived'], df['Sex'])

sns.countplot(x='Survived', hue='Pclass', data=df)

pd.crosstab(df['Survived'], df['Pclass'])

sns.boxplot(x='Pclass', y='Age', data=df)

df = df.drop("Cabin", axis=1, errors="ignore")

df.head()

df = df.dropna()
```

```
df.shape
df.head()

df = pd.get_dummies(columns=['Sex', 'Embarked', 'Pclass'], data=df)

df.head()

df = df.drop(["PassengerId", "Name", "Ticket", "Fare"], axis=1, errors="ignore")

df.head()

df.dtypes

X = df.drop("Survived", axis=1)  # Features (all columns except 'Survived')
Y = df["Survived"]  # Target variable (Survived)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1)

model_1 = LogisticRegression()
model_1.fit(X_train, Y_train)

train_accuracy = model_1.score(X_train, Y_train)

df.dtypes

X = df.drop("Survived", axis=1)  # Features (all columns except 'Survived')
Y = df["Survived"]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1)
```

```
model_1 = LogisticRegression()
model_1.fit(X_train, Y_train)

train_accuracy = model_1.score(X_train, Y_train)

print("Classification Report:")
print(classification_report(Y_test, predictions))


cm = metrics.confusion_matrix(Y_test, predictions, labels=[1, 0])

df_cm = pd.DataFrame(cm, index=["1", "0"], columns=["Predict 1", "Predict 0"])

plt.figure(figsize=(7, 5))
sns.heatmap(df_cm, annot=True, fmt="g")
plt.title("Confusion Matrix")
plt.show()
```

Pr4 Implementation of Multi-Class Classification

```
import pandas as pd

dataset = pd.read_csv("/content/Iris.csv")



print(dataset['Species'].unique())

dataset['Species'].replace({'Iris-setosa': 1, 'Iris-versicolor': 2, 'Iris-virginica': 3},
inplace=True)
dataset

from sklearn.model_selection import train_test_split

len(X_train)

# len(X_test) helps you understand the size of the testing data used for model
evaluation

len(X_test)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=200)

lr.fit(X_train, y_train)
LogisticRegression()

(X_test)
lr.predict(X_test)
```

```
X_test
```

```
lr.score(X_test, y_test)
```

```
import seaborn as sns
```

```
sns.pairplot(dataset[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm', 'Species']], hue='Species')
```

Pr5 Implementation of Decision Tree Algorithm

```
import pandas as pd
```

```
df = pd.read_csv('/content/salaries.csv')
```

```
df.head()
```

```
inputs = df.drop('salary_more_then_100k', axis='columns')
inputs
```

```
target = df['salary_more_then_100k']
target
```

```
from sklearn.preprocessing import LabelEncoder
```

```
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
```

```
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
```

```
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])
inputs

inputs_n = inputs.drop(['company','job','degree'],axis='columns')
inputs_n

from sklearn import tree
model = tree.DecisionTreeClassifier()
model.fit(inputs_n,target)

model.score(inputs_n,target)

model.predict([[2,1,0]])

(company_n=2, job_n=1, degree_n=1)
model.predict([[2,1,1]])
```

Pr5 a

Gini_index

```
import pandas as pd   # For data manipulation and analysis
import numpy as np    # For numerical operations
import matplotlib.pyplot as plt   # For plotting graphs

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import tree  # For creating decision tree model
from sklearn.tree import DecisionTreeClassifier, plot_tree  # Decision tree classifier for model
and plot_tree for visualization.

df = pd.read_csv('/content/tennis.csv')

df  # This will show the first few rows of the dataset
```

```
le = LabelEncoder()

df['outlook_n'] = le.fit_transform(df['outlook'])  # Converting 'outlook' column to numeric values
df['temp_n'] = le.fit_transform(df['temp'])  # Converting 'temp' column to numeric values
df['humidity_n'] = le.fit_transform(df['humidity'])  # Converting 'humidity' column to numeric values
df['windy_n'] = le.fit_transform(df['windy'])  # Converting 'windy' column to numeric values
df['play_n'] = le.fit_transform(df['play'])  # Converting 'play' column (target) to numeric values (0 or 1)

df

df = df.drop(['outlook', 'temp', 'humidity', 'windy', 'play'], axis='columns')  # Dropping the non numeric columns

df  # This will show the dataframe with only the numeric columns

independent_variable = df.drop('play_n', axis='columns')
dependent_variable = df['play_n']

model = tree.DecisionTreeClassifier()  # Creating the decision tree model

model.fit(independent_variable,  dependent_variable)

model_score = model.score(independent_variable, dependent_variable)

model_score

prediction = model.predict([[1, 2, 0, 1]])

prediction
```

```
plt.figure(figsize=(120, 80))

plot_tree(model, filled=True, feature_names=independent_variable.columns,
class_names=le.classes_)
plt.show()
```

Pr6 Implementation of Random Forest Algorithm

```
import pandas as pd
from sklearn.datasets import load_digits

df = pd.read_csv('/content/digits.csv')

dir(digits)

%matplotlib inline

import matplotlib.pyplot as plt

plt.gray()

for i in range(4):
    plt.matshow(digits.images[i])
    plt.show()

df = pd.DataFrame(digits.data)

df.head()

df['target'] = digits.target

df.head()
```

```
df[0:12]

X = df.drop('target', axis='columns')
y = df['target']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=20)

model.fit(X_train, y_train)

accuracy = model.score(X_test, y_test)
print("Model Accuracy on Test Set:", accuracy)

y_predicted = model.predict(X_test)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_predicted)

cm

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn

plt.figure(figsize=(10,7))
```

```
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix - Random Forest on Digits Dataset')
plt.show()
```

Pr7 Implementation of K-Means Clustering Algorithm

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

df = pd.read_csv("income.csv")
df.head()

import matplotlib.pyplot as plt

plt.scatter(df.Age, df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.title('Age vs Income Scatter Plot')
plt.show()

from sklearn.cluster import KMeans

km = KMeans(n_clusters=3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])

y_predicted
```

```
df['cluster'] = y_predicted

df.head()

km.cluster_centers_

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]

df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',
label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

df
plt.scatter(df.Age,df['Income($)'])
```

```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted

df['cluster']=y_predicted
df.head()

df['cluster']=y_predicted
df.head()

df1 = df[df.cluster==0]

df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',l
abel='centroid')
plt.legend()

sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Pr8 Implementation of Association Rule Mining

```
import warnings
warnings.filterwarnings('ignore',  category=DeprecationWarning)

import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

data = {'TID': [1, 2, 3, 4, 5],
        'Milk': [1, 0, 1, 1, 0],
        'Bread': [1, 1, 1, 0, 1],
        'Butter': [0, 1, 0, 1, 1],
        'Cheese': [1, 0, 1, 1, 0]}

df = pd.DataFrame(data).set_index('TID')

df = df.astype(bool)

print("Dataset:\n", df)

frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
print("\nFrequent Itemsets:\n", frequent_itemsets)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

rules = rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
print("\nAssociation Rules:\n", rules)

strong_rules = rules[rules['lift'] > 1]
print("\nStrong Rules (Lift > 1):\n", strong_rules)
```

Pr9 Implementation of a Neural Network

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense

#Link to download diabetes dataset :
https://github.com/25-Madhuri/Dataset/blob/main/diabetes.csv

data = pd.read_csv('/content/diabetes.csv')
data.head()

import seaborn as sns
data['Outcome'].value_counts().plot(kind='bar')

predictors = data.iloc[:, 0:8]
response = data.iloc[:, 8]

X_train, X_test, Y_train, Y_test = train_test_split(predictors, response, test_size=0.2,
random_state=0)

print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)

from keras.models import Sequential
from keras.layers import Dense

kerasmodel = Sequential()
```

```
kerasmodel.add(Dense(units=12, activation='relu', input_dim=8))
features (X_train.shape[1])
kerasmodel.add(Dense(units=8, activation='relu'))
kerasmodel.add(Dense(units=1, activation='sigmoid'))

kerasmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

kerasmodel.fit(X_train, Y_train, epochs=150, batch_size=10)

_, accuracy = kerasmodel.evaluate(X_train, Y_train)
print('Train Accuracy: %.2f' % (accuracy*100))

_, accuracy = kerasmodel.evaluate(X_test, Y_test)
print('Test Accuracy: %.2f' % (accuracy*100))

from sklearn.metrics import accuracy_score

y_pred_prob = kerasmodel.predict(X_test)

y_pred = (y_pred_prob > 0.5).astype(int)

accuracy = accuracy_score(Y_test, y_pred)

print("Test Accuracy:", accuracy)
```

Pr10 Implementation of Back-propagation Algorithm of Neural Network

```
import numpy as np

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
```

```
input_size = 2
hidden_size = 2
output_size = 1
learning_rate = 0.1

np.random.seed(42)
weights_input_hidden = np.random.rand(input_size, hidden_size)
weights_hidden_output = np.random.rand(hidden_size, output_size)
bias_hidden = np.random.rand(1, hidden_size)
bias_output = np.random.rand(1, output_size)



def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
   return x * (1 - x)

epochs = 10000
 for epoch in range(epochs):

   hidden_input = np.dot(X, weights_input_hidden) + bias_hidden
   hidden_output = sigmoid(hidden_input)
   final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
   final_output = sigmoid(final_input)

loss = np.mean((y - final_output) ** 2)

error_output = y - final_output
d_output = error_output * sigmoid_derivative(final_output)
error_hidden = d_output.dot(weights_hidden_output.T)
```

```
    d_hidden = error_hidden * sigmoid_derivative(hidden_output)

    weights_hidden_output += hidden_output.T.dot(d_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden) * learning_rate
    bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate
    bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("\nTrained Neural Network Output:")
print(final_output)
```