

# ITA0628 Machine Learning - Lab Experiments

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

```
# FIND-S Algorithm Implementation
def find_s_algorithm(training_data):
    # Get the number of attributes (excluding the target label)
    num_attributes = len(training_data[0]) - 1

    # Step 1: Initialize the hypothesis to the first positive example
    for example in training_data:
        if example[-1].lower() == 'yes':
            hypothesis = example[:-1] # Exclude the label
            break

    # Step 2: Generalize hypothesis using remaining positive examples
    for example in training_data:
        if example[-1].lower() == 'yes': # Only consider positive examples
            for i in range(num_attributes):
                if hypothesis[i] != example[i]:
                    hypothesis[i] = '?' # Generalize differing attribute

    return hypothesis

# Sample training dataset
# Format: [Sky, AirTemp, Humidity, Wind, Water, Forecast, EnjoySport]
training_data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

# Run FIND-S Algorithm
hypothesis = find_s_algorithm(training_data)

# Output the result
print("Final hypothesis (most specific that fits positive examples):")
print(hypothesis)
```

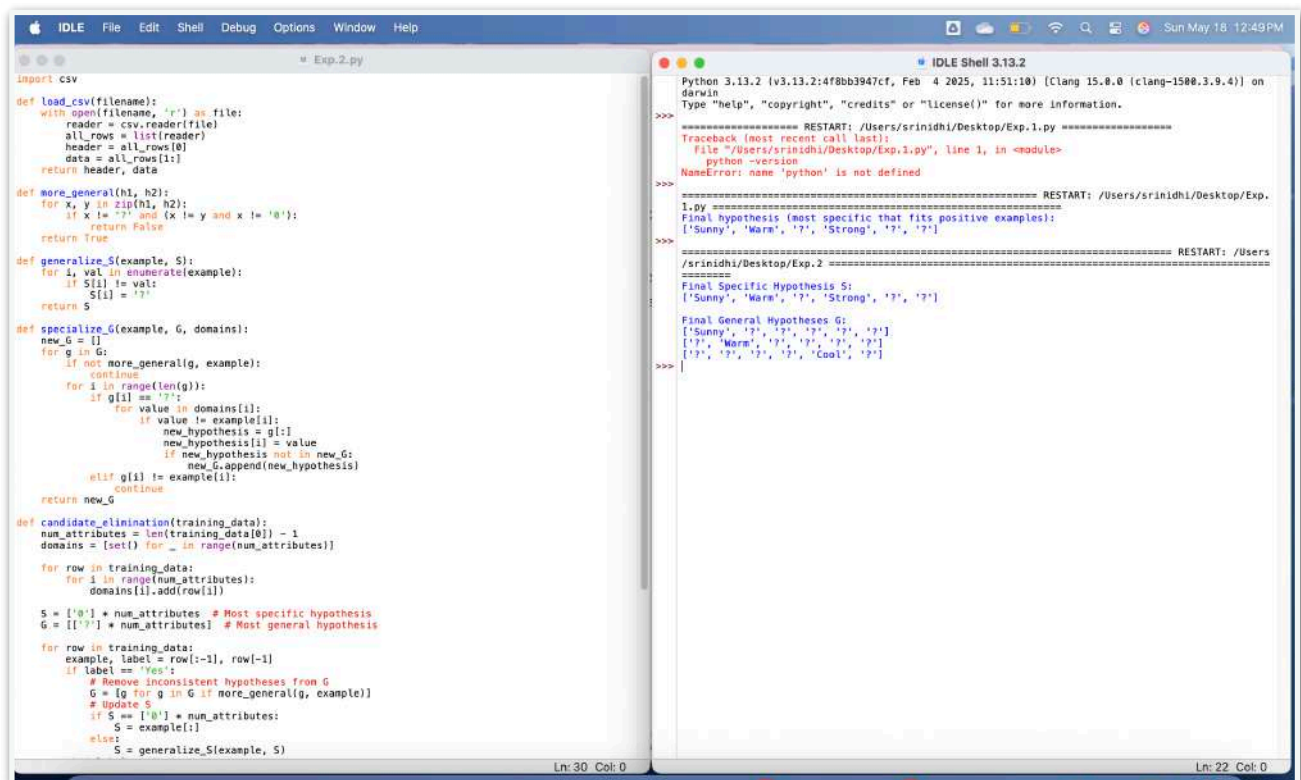
Output:



```
Python 3.13.2 (v3.13.2:4f8bb3947cf, Feb  4 2025, 11:51:10) [Clang 15.0.0 (clang-1500.3.9.4)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp.1.py =====
Traceback (most recent call last):
  File "/Users/srinidhi/Desktop/Exp.1.py", line 1, in <module>
    python -version
NameError: name 'python' is not defined
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp.1.py =====
Final hypothesis (most specific that fits positive examples):
['Sunny', 'Warm', '?', 'Strong', '?', '?']
>>>
```

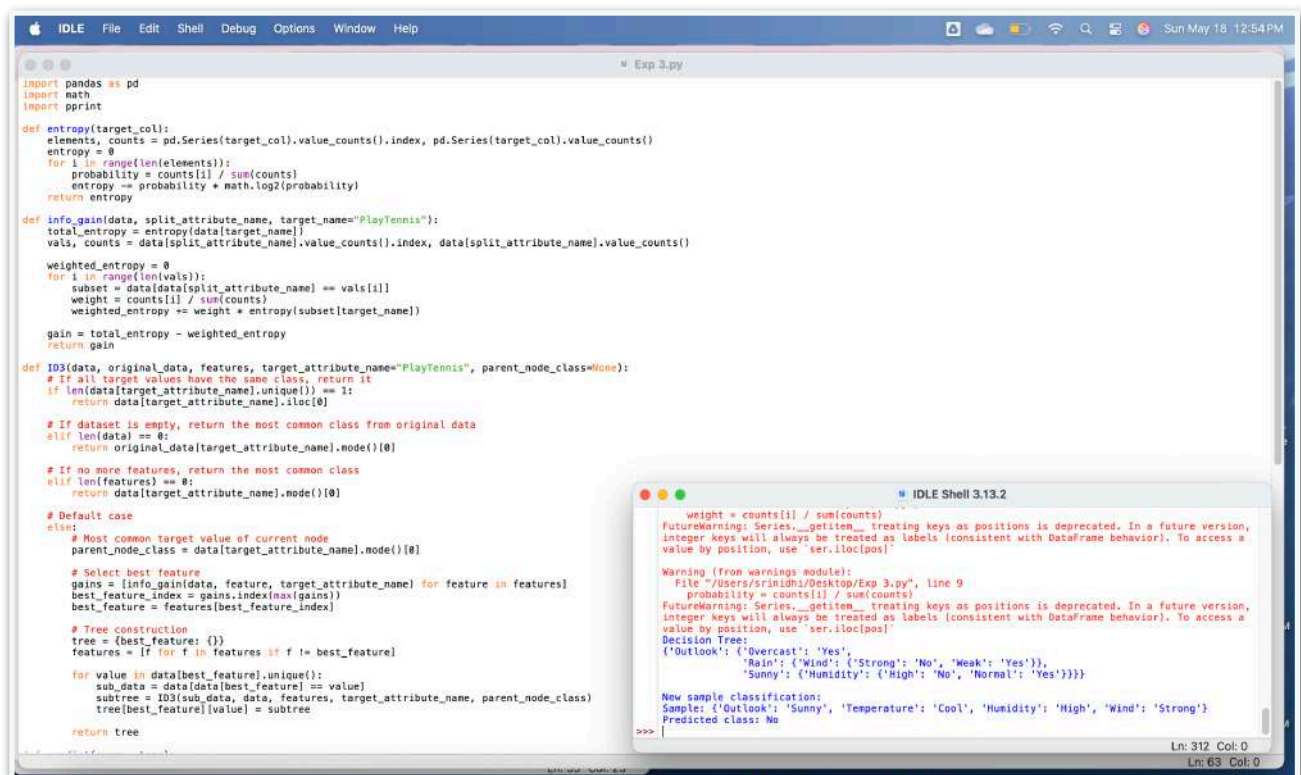
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a description of the set of all hypotheses consistent with the training examples

Output ScreenShot:



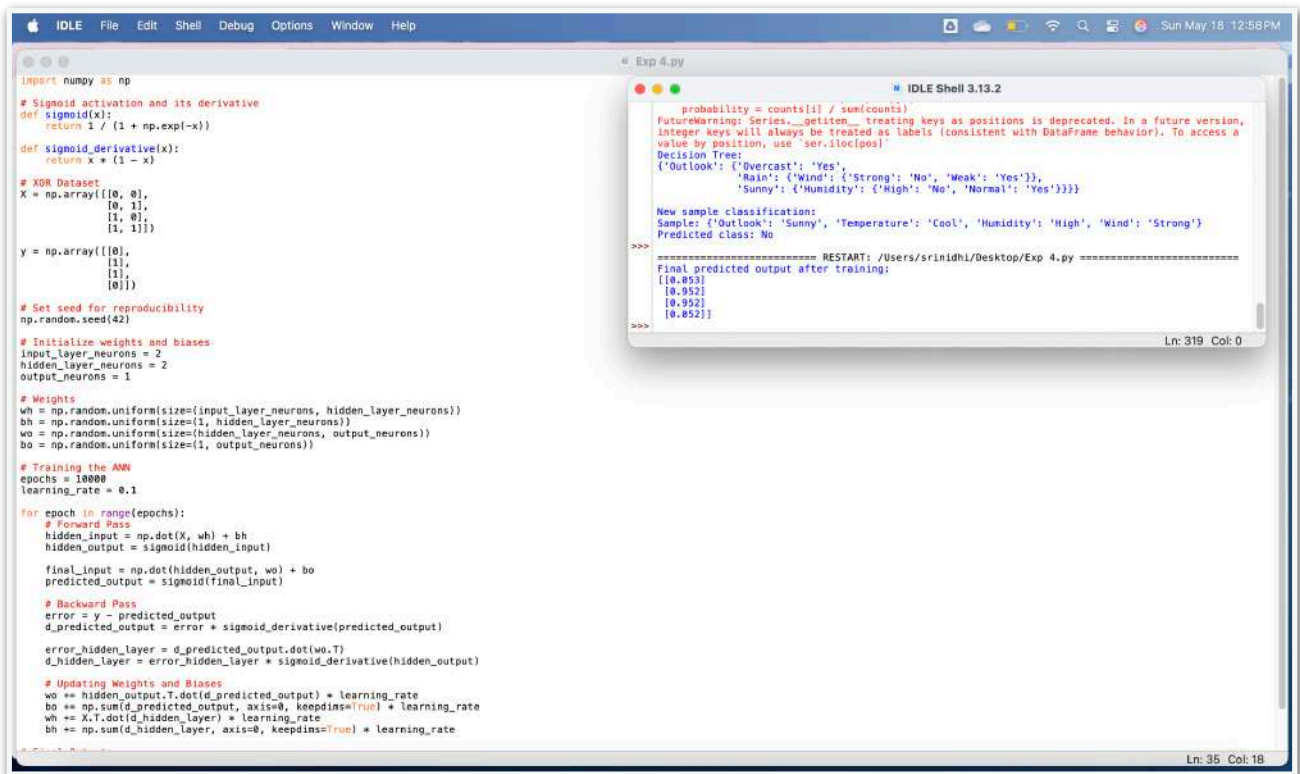
3. **Demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

Output:



#### 4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Output:



```
import numpy as np

# Sigmoid activation and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# XOR Dataset
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

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

# Set seed for reproducibility
np.random.seed(42)

# Initialize weights and biases
input_layer_neurons = 2
hidden_layer_neurons = 2
output_neurons = 1

# Weights
w1 = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
b1 = np.random.uniform(size=(1, hidden_layer_neurons))
w2 = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
b2 = np.random.uniform(size=(1, output_neurons))

# Training the ANN
epochs = 10000
learning_rate = 0.1

for epoch in range(epochs):
    # Forward Pass
    hidden_input = np.dot(X, w1) + b1
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, w2) + b2
    predicted_output = sigmoid(final_input)

    # Backward Pass
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

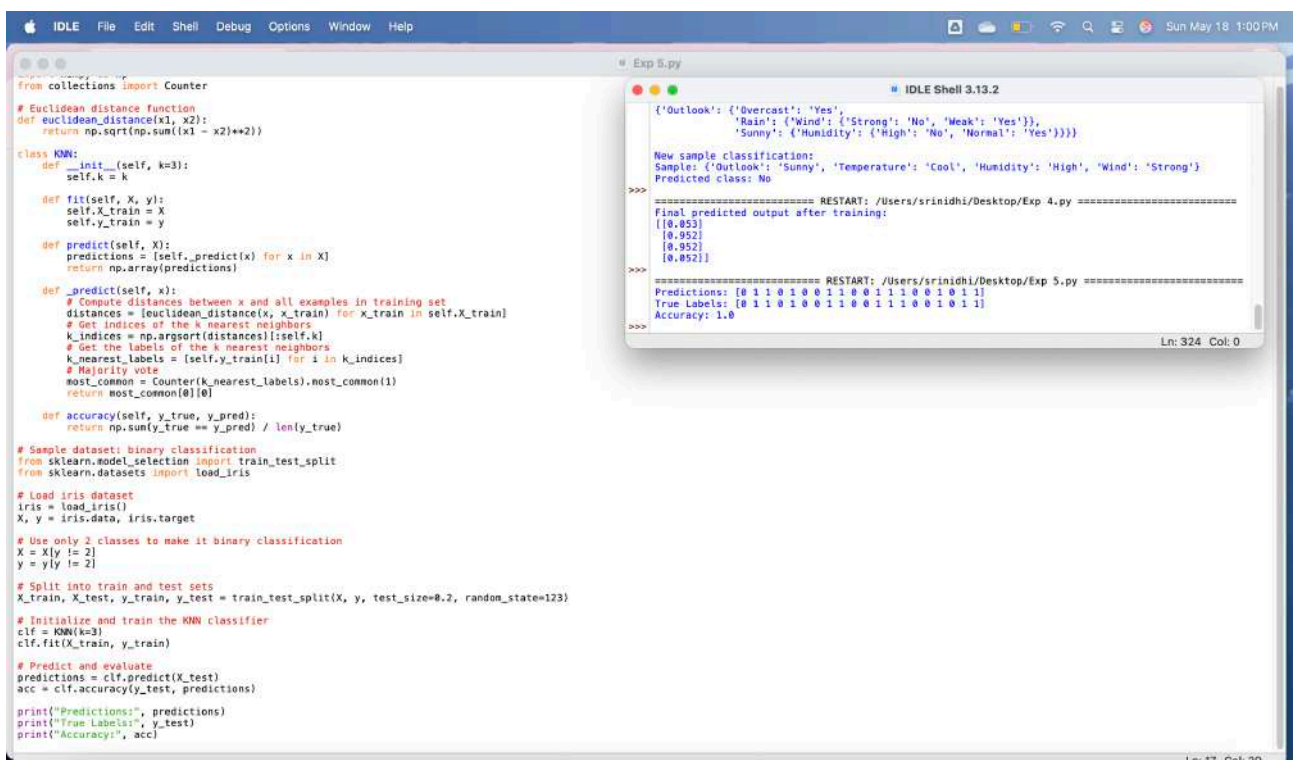
    error_hidden_layer = d_predicted_output.dot(w2.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_output)

    # Updating Weights and Biases
    w1 += hidden_output.T.dot(d_hidden_layer) * learning_rate
    b1 += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate
    w2 += hidden_output.dot(d_predicted_output) * learning_rate
    b2 += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

# Sample input
sample = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High', 'Wind': 'Strong'}
predicted_class = No
```

#### 5. Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python.

Output:



```
from collections import Counter

# Euclidean distance function
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        # Compute distances between x and all examples in training set
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
        # Get indices of the k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]
        # Get the labels of the k nearest neighbors
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        # Majority vote
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    def accuracy(self, y_true, y_pred):
        return np.sum(y_true == y_pred) / len(y_true)

# Sample dataset: binary classification
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

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

# Use only 2 classes to make it binary classification
X = X[y != 2]
y = y[y != 2]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

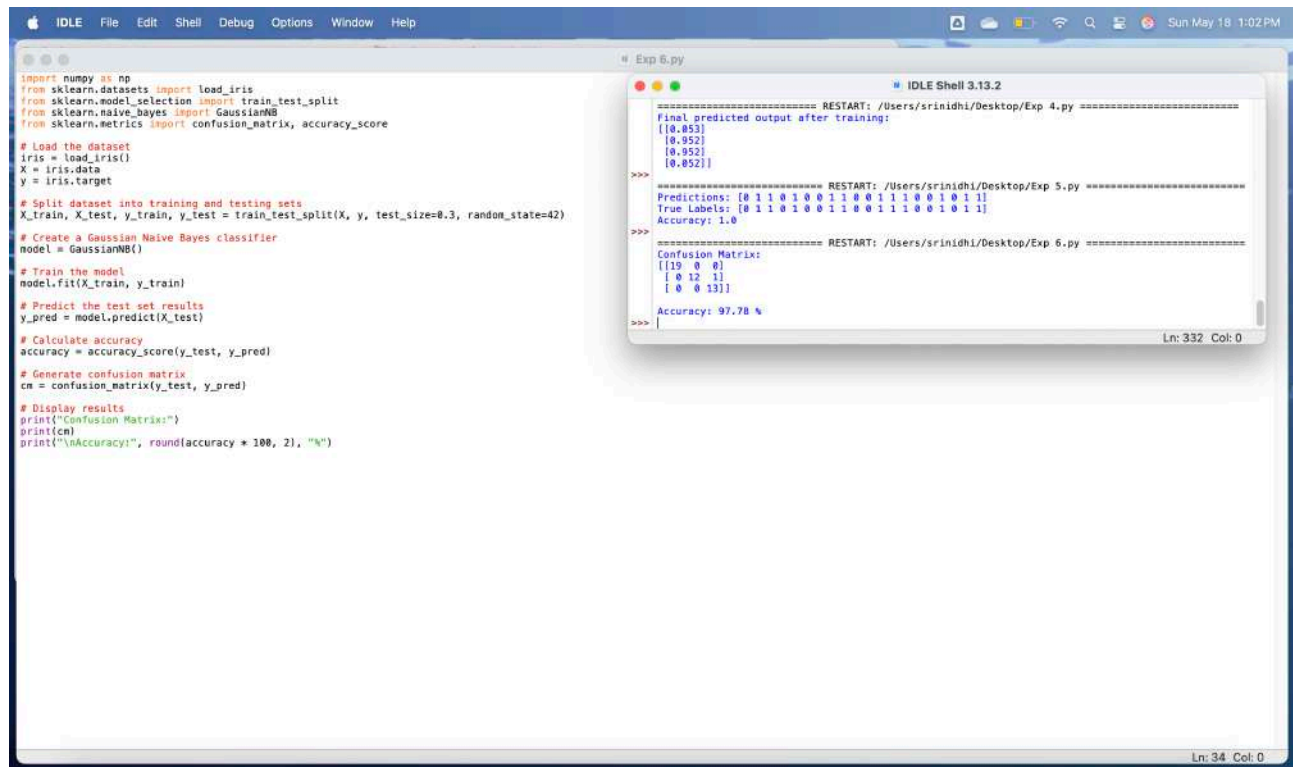
# Initialize and train the KNN classifier
clf = KNN(k=3)
clf.fit(X_train, y_train)

# Predict and evaluate
predictions = clf.predict(X_test)
acc = clf.accuracy(y_test, predictions)

print("Predictions:", predictions)
print("True Labels:", y_test)
print("Accuracy:", acc)
```

## 6. Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy.

Output:



```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

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

# Split 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)

# Create a Gaussian Naive Bayes classifier
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display results
print("Confusion Matrix:")
print(cm)
print("\nAccuracy:", round(accuracy * 100, 2), "%")
```

```
===== RESTART: /Users/srinidhi/Desktop/Exp 4.py =====
Final predicted output after training:
[[0.053]
 [0.952]
 [0.052]]

>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 5.py =====
Predictions: [0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1]
True Labels: [0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1]
Accuracy: 1.0

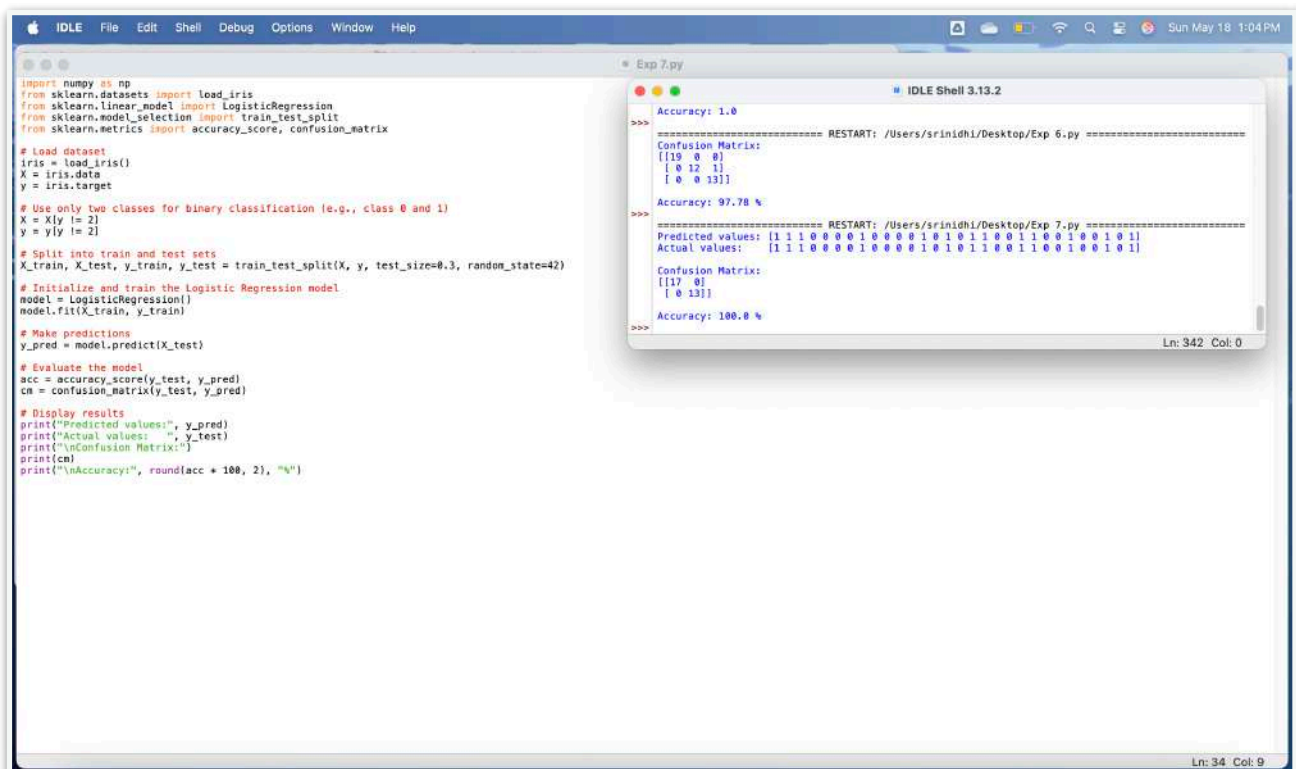
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 6.py =====
Confusion Matrix:
[[15  0  0]
 [ 0 12  1]
 [ 0  0 13]]

Accuracy: 97.78 %

Ln: 332 Col: 0
```

## 7. Write a program to implement Logistic Regression (LR) algorithm in python

Output:



```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

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

# Use only two classes for binary classification (e.g., class 0 and 1)
X = X[y != 2]
y = y[y != 2]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

# Display results
print("Predicted values:", y_pred)
print("Actual values:", y_test)
print("\nConfusion Matrix:")
print(cm)
print("\nAccuracy:", round(acc * 100, 2), "%")
```

```
Accuracy: 1.0

>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 6.py =====
Confusion Matrix:
[[15  0  0]
 [ 0 12  1]
 [ 0  0 13]]

Accuracy: 97.78 %

>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 7.py =====
Predicted values: [1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1]
Actual values:    [1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1]
Confusion Matrix:
[[17  0]
 [ 0 13]]

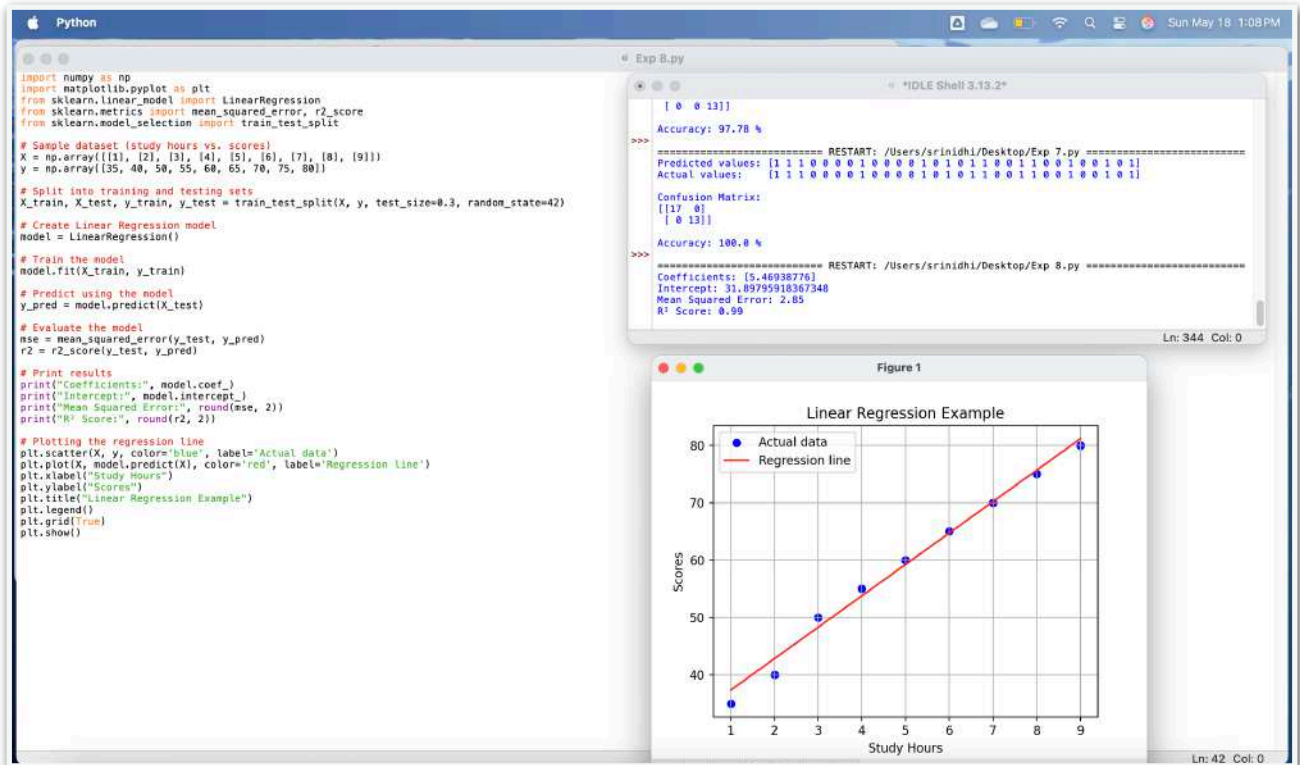
Accuracy: 100.0 %

Ln: 342 Col: 0
```



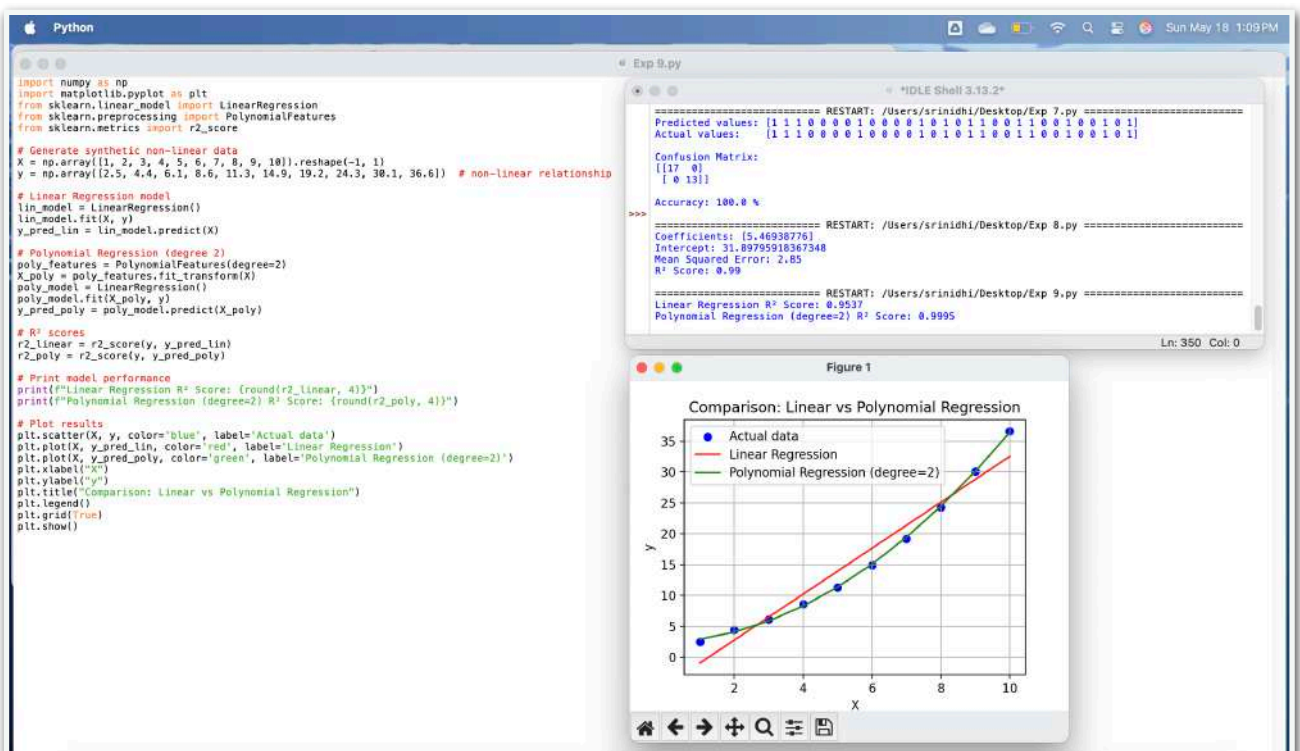
## 8. Write a program to implement Linear Regression (LR) algorithm in python

Output:



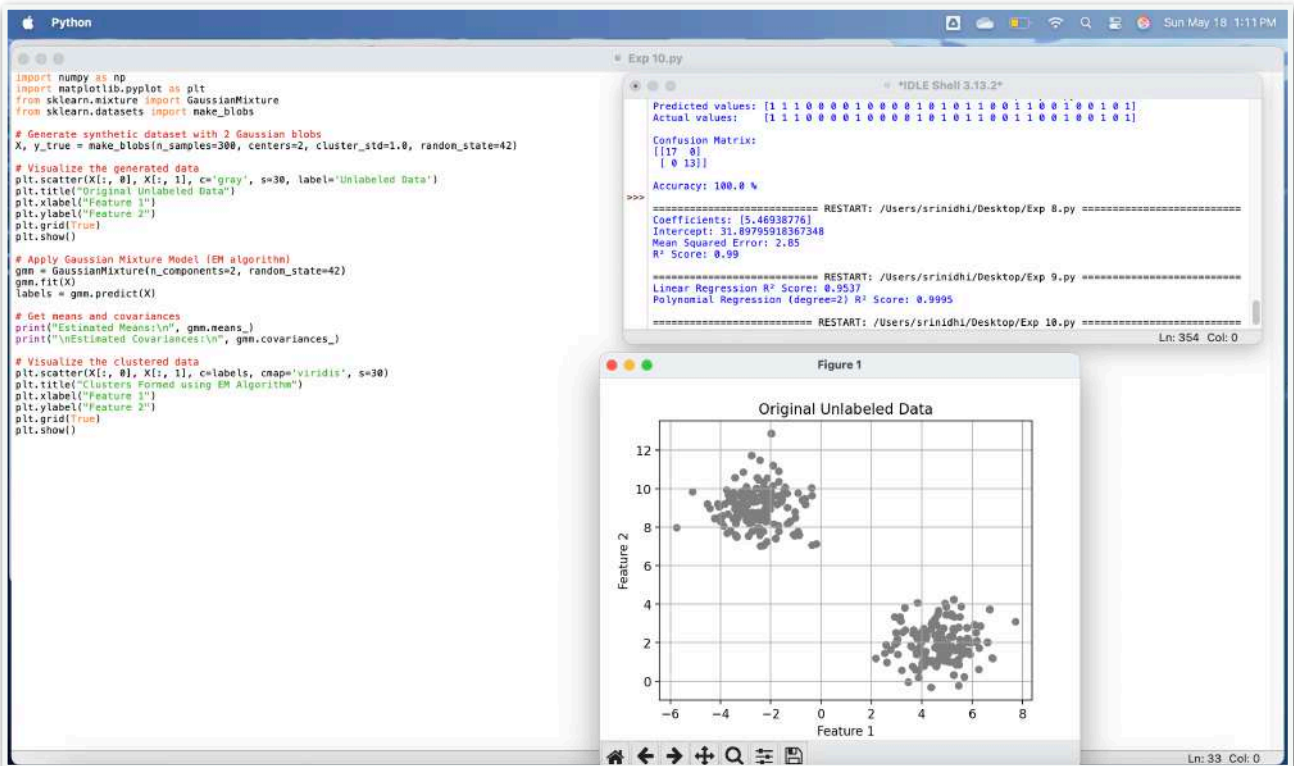
## 9. Compare Linear and Polynomial Regression using Python

Output:



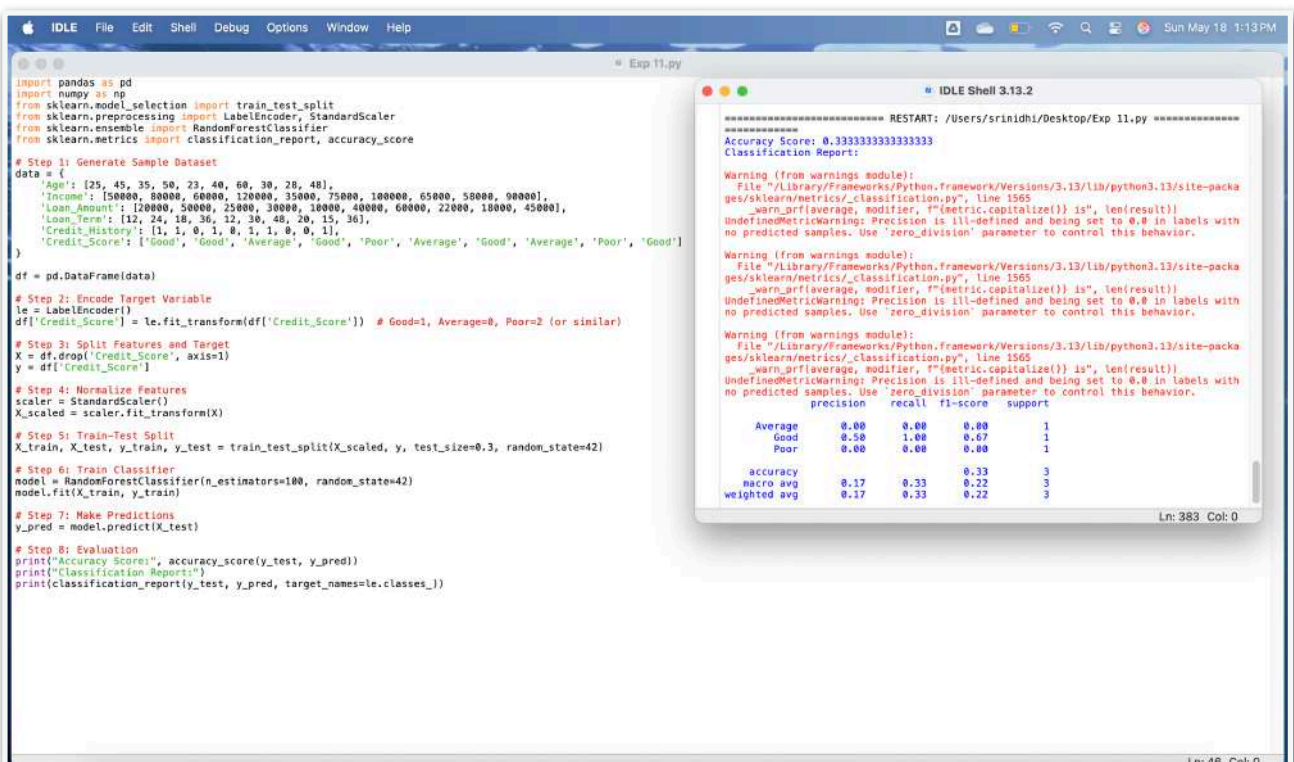
## 10. Write a Python Program to Implement Expectation & Maximization Algorithm

Output:



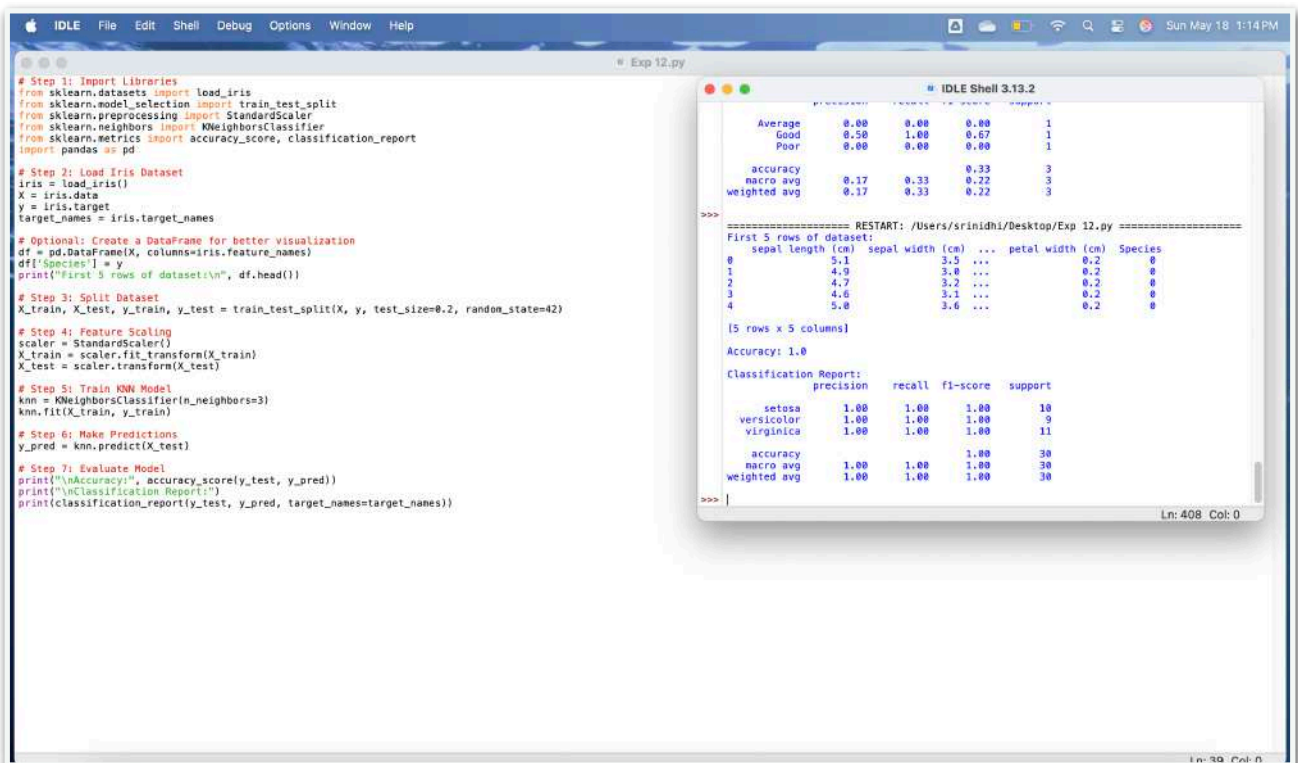
## 11. Write a program for the task of Credit Score Classification

Output:



## 12. Implement Iris Flower Classification using KNN

Output:



```
# Step 1: Import Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# Step 2: Load Iris Dataset
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Optional: Create a DataFrame for better visualization
df = pd.DataFrame(X, columns=iris.feature_names)
df['Species'] = y
print("First 5 rows of dataset:\n", df.head())

# Step 3: Split Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 5: Train KNN Model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Step 6: Make Predictions
y_pred = knn.predict(X_test)

# Step 7: Evaluate Model
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 12.py =====

First 5 rows of dataset:

	sepal length (cm)	sepal width (cm)	... petal width (cm)	Species	
0	5.1	3.5	...	0.2	0
1	4.9	3.0	...	0.2	0
2	4.7	3.2	...	0.2	0
3	4.6	3.1	...	0.2	0
4	5.0	3.6	...	0.2	0

[5 rows x 5 columns]

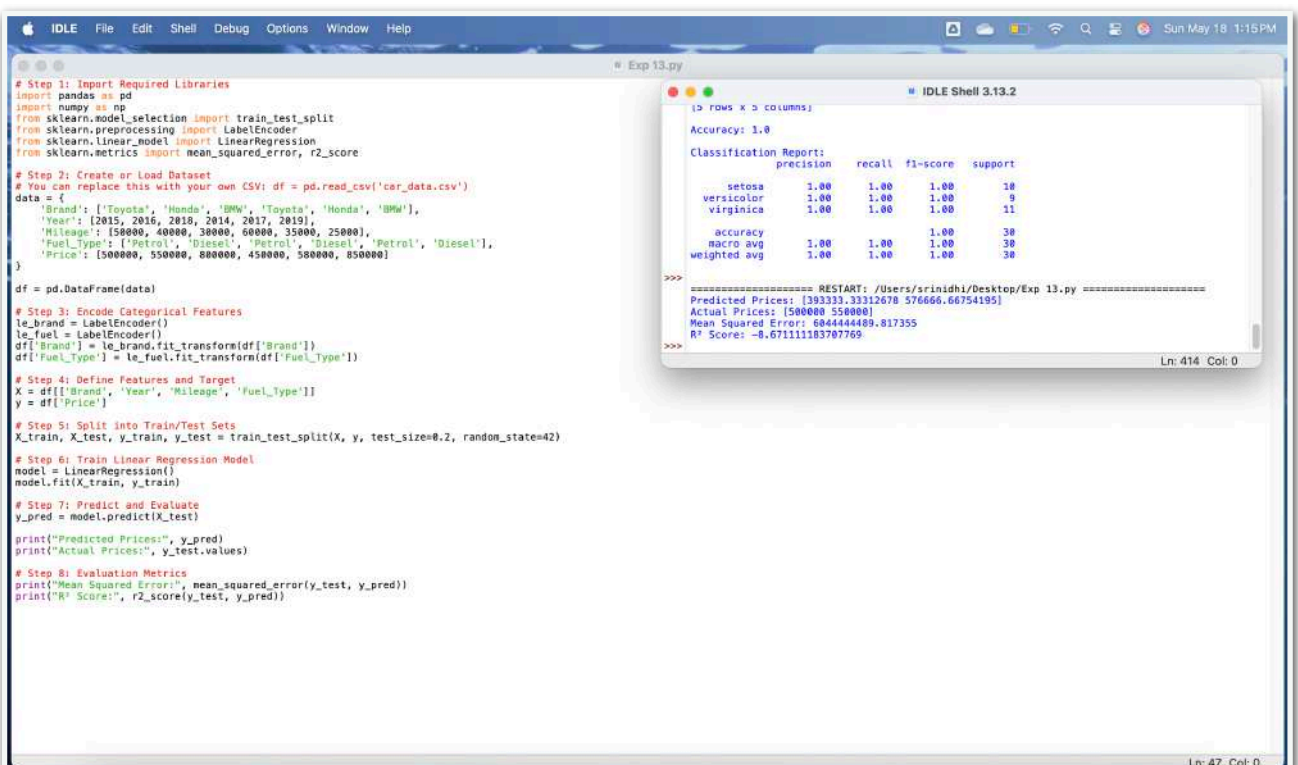
Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

## 13. Implement the Car Price Prediction Model using Python.

Output:



```
# Step 1: Import Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 2: Create or Load Dataset
# You can replace this with your own CSV: df = pd.read_csv('car_data.csv')
data = {
    'Brand': ['Toyota', 'Honda', 'BMW', 'Toyota', 'Honda', 'BMW'],
    'Year': [2015, 2016, 2018, 2014, 2017, 2019],
    'Mileage': [50000, 40000, 30000, 60000, 35000, 25000],
    'Fuel_Type': ['Petrol', 'Diesel', 'Petrol', 'Diesel', 'Petrol', 'Diesel'],
    'Price': [500000, 550000, 800000, 450000, 580000, 850000]
}

df = pd.DataFrame(data)

# Step 3: Encode Categorical Features
le_brand = LabelEncoder()
le_fuel = LabelEncoder()
df['Brand'] = le_brand.fit_transform(df['Brand'])
df['Fuel_Type'] = le_fuel.fit_transform(df['Fuel_Type'])

# Step 4: Define Features and Target
X = df[['Brand', 'Year', 'Mileage', 'Fuel_Type']]
y = df['Price']

# Step 5: Split into Train/Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 7: Predict and Evaluate
y_pred = model.predict(X_test)

print("Predicted Prices:", y_pred)
print("Actual Prices:", y_test.values)

# Step 8: Evaluation Metrics
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 13.py =====

Predicted Prices: [393333.33312678 576666.66754195]

Actual Prices: [500000 550000]

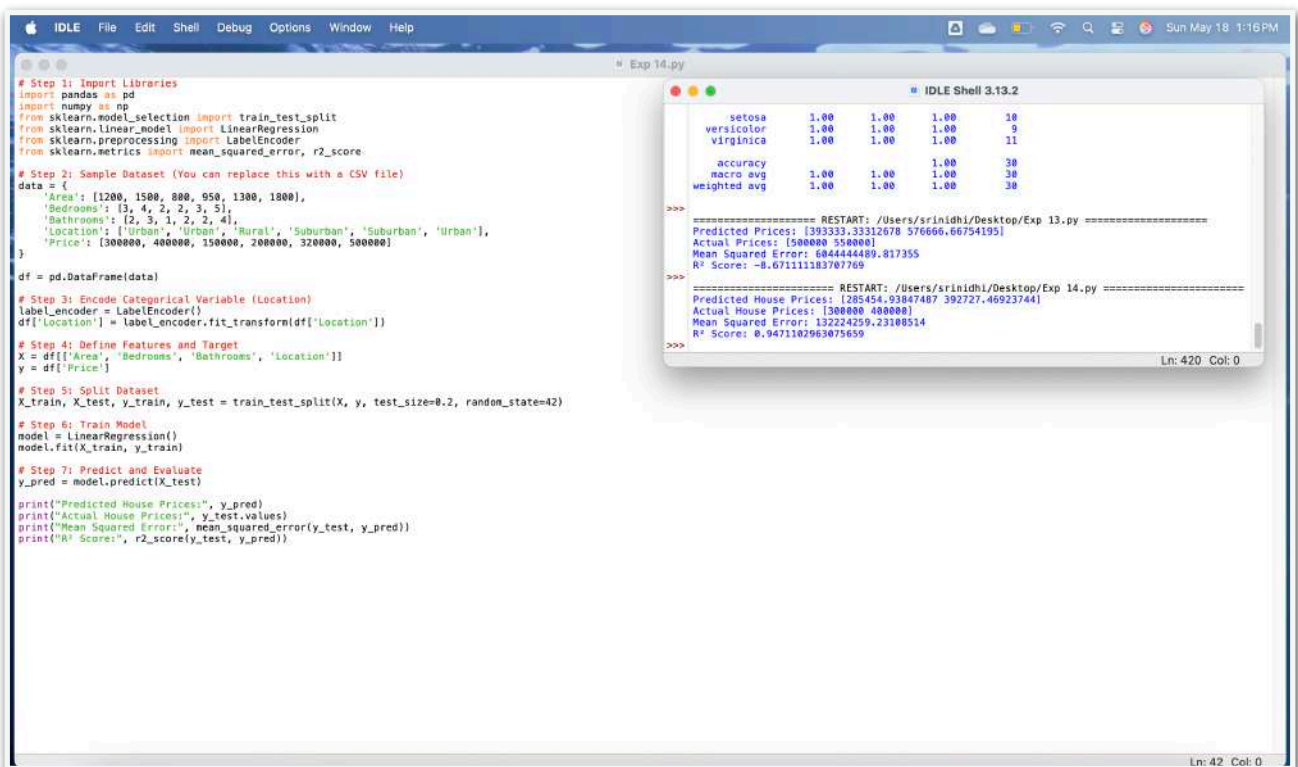
Mean Squared Error: 684444489.817355

R^2 Score: -0.67111183707769



## 14. Implement House price Prediction using appropriate machine learning algorithm

Output:



```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score

# Step 2: Sample Dataset (You can replace this with a CSV file)
data = {
    'Area': [1200, 1500, 800, 950, 1300, 1800],
    'Bedrooms': [3, 4, 2, 2, 3, 5],
    'Bathrooms': [2, 3, 1, 2, 5, 4],
    'Location': ['Urban', 'Urban', 'Rural', 'Suburban', 'Suburban', 'Urban'],
    'Price': [300000, 400000, 150000, 200000, 320000, 500000]
}

df = pd.DataFrame(data)

# Step 3: Encode Categorical Variable (Location)
label_encoder = LabelEncoder()
df['Location'] = label_encoder.fit_transform(df['Location'])

# Step 4: Define Features and Target
X = df[['Area', 'Bedrooms', 'Bathrooms', 'Location']]
y = df['Price']

# Step 5: Split Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Train Model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 7: Predict and Evaluate
y_pred = model.predict(X_test)

print("Predicted House Prices:", y_pred)
print("Actual House Prices:", y_test.values)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 13.py =====

	1.00	1.00	1.00	10
setosa	1.00	1.00	1.00	9
versicolor	1.00	1.00	1.00	11
virginica	1.00	1.00	1.00	30
accuracy	1.00	1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Predicted Prices: [393333.33312678 576666.66754195]  
Actual Prices: [500000 550000]  
Mean Squared Error: 684444409.817355  
R^2 Score: -8.67111183707769

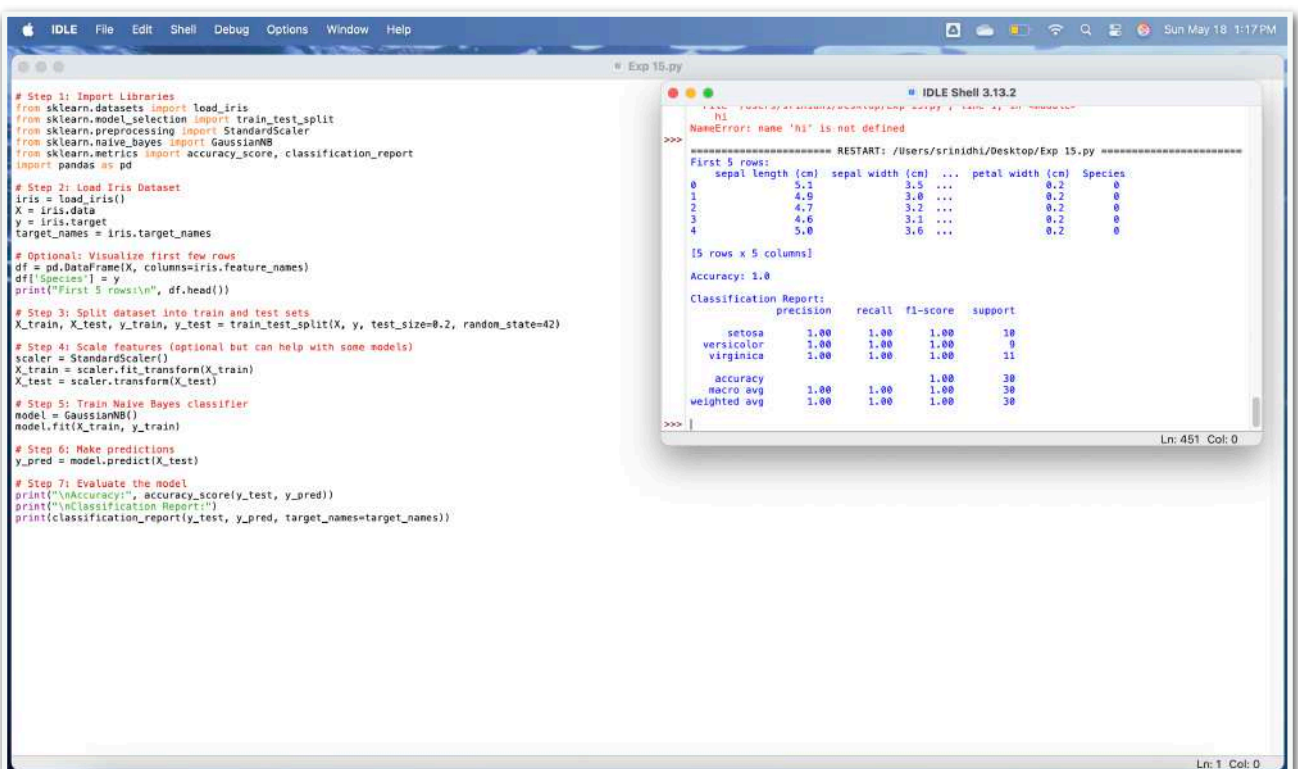
===== RESTART: /Users/srinidhi/Desktop/Exp 14.py =====

	1.00	1.00	1.00	30
setosa	1.00	1.00	1.00	9
versicolor	1.00	1.00	1.00	11
virginica	1.00	1.00	1.00	30
accuracy	1.00	1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Predicted House Prices: [285454.93847487 392727.46923744]  
Actual House Prices: [300000 400000]  
Mean Squared Error: 13224259.23108514  
R^2 Score: 0.947182963075659

## 15. Implement Iris Flower Classification using Naive Bayes classifier

Output:



```
# Step 1: Import Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# Step 2: Load Iris Dataset
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Optional: Visualize first few rows
df = pd.DataFrame(X, columns=iris.feature_names)
df['Species'] = y
print("First 5 rows:\n", df.head())

# Step 3: Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Scale features (optional but can help with some models)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 5: Train Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 15.py =====

NameError: name 'hj' is not defined

First 5 rows:

	sepal length (cm)	sepal width (cm)	...	petal width (cm)	Species
0	5.1	3.5	...	0.2	0
1	4.9	3.0	...	0.2	0
2	4.7	3.2	...	0.2	0
3	4.6	3.1	...	0.2	0
4	5.0	3.6	...	0.2	0

[5 rows x 5 columns]

Accuracy: 1.0

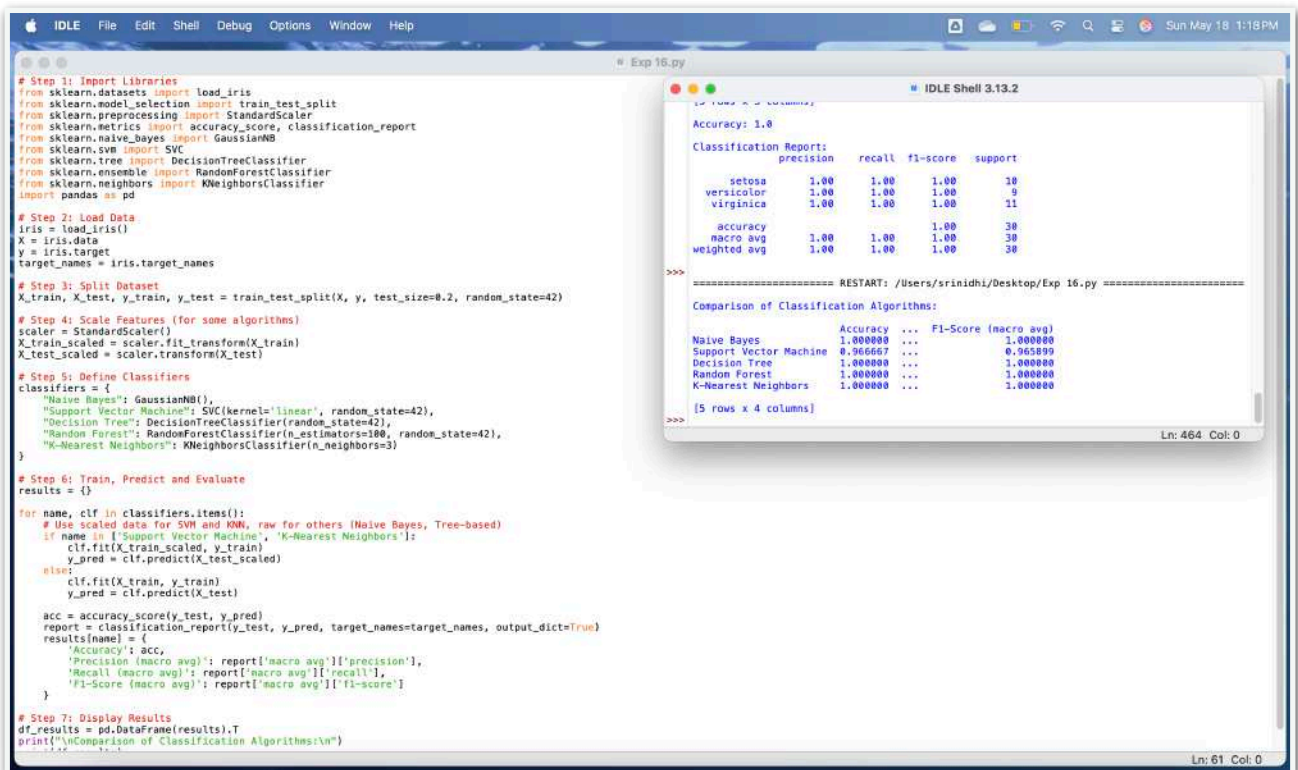
Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy	1.00	1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30



## 16. Compare different types Classification Algorithms and evaluate their performance.

Output:



```
# Step 1: Import Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

# Step 2: Load Data
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Step 3: Split Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Scale Features (for some algorithms)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Define Classifiers
classifiers = {
    "Naive Bayes": GaussianNB(),
    "Support Vector Machine": SVC(kernel='linear', random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3)
}

# Step 6: Train, Predict and Evaluate
results = {}
for name, clf in classifiers.items():
    # Use scaled data for SVM and KNN, raw for others (Naive Bayes, Tree-based)
    if name in ["Support Vector Machine", "K-Nearest Neighbors"]:
        clf.fit(X_train_scaled, y_train)
        y_pred = clf.predict(X_test_scaled)
    else:
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=target_names, output_dict=True)
    results[name] = {
        'Accuracy': acc,
        'Precision (macro avg)': report['macro avg']['precision'],
        'Recall (macro avg)': report['macro avg']['recall'],
        'F1-Score (macro avg)': report['macro avg']['f1-score']
    }

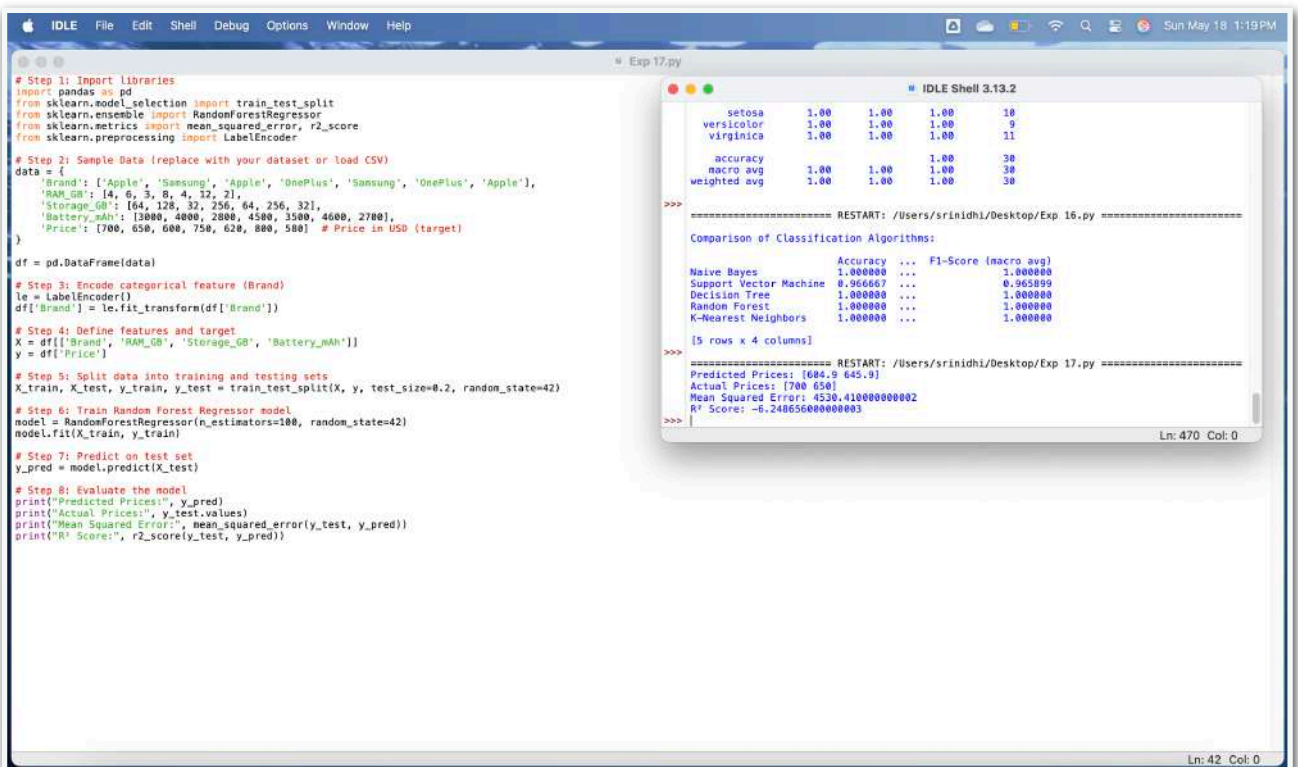
# Step 7: Display Results
df_results = pd.DataFrame(results).T
print("\nComparison of Classification Algorithms:\n")
```

Accuracy: 1.0				
Classification Report:				
	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolour	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy	1.00	1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 16.py =====
Comparison of Classification Algorithms:
=====
Accuracy ... F1-Score (macro avg)
Naive Bayes 1.000000 ... 1.000000
Support Vector Machine 0.966667 ... 0.966667
Decision Tree 1.000000 ... 1.000000
Random Forest 1.000000 ... 1.000000
K-Nearest Neighbors 1.000000 ... 1.000000
[5 rows x 4 columns]
>>>
```

## 17. Implement Mobile Price Prediction using appropriate machine learning algorithm

Output:



```
# Step 1: Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder

# Step 2: Sample Data (replace with your dataset or load CSV)
data = {
    'Brand': ['Apple', 'Samsung', 'Apple', 'OnePlus', 'Samsung', 'OnePlus', 'Apple'],
    'RAM_GB': [4, 6, 3, 8, 4, 12, 2],
    'Storage_GB': [64, 128, 32, 256, 64, 256, 32],
    'Battery_mAh': [1800, 4000, 2800, 4500, 1500, 4000, 2700],
    'Price': [700, 650, 600, 750, 620, 800, 580] # Price in USD (target)
}

df = pd.DataFrame(data)

# Step 3: Encode categorical feature (Brand)
le = LabelEncoder()
df['Brand'] = le.fit_transform(df['Brand'])

# Step 4: Define features and target
X = df[['Brand', 'RAM_GB', 'Storage_GB', 'Battery_mAh']]
y = df['Price']

# Step 5: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Train Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 7: Predict on test set
y_pred = model.predict(X_test)

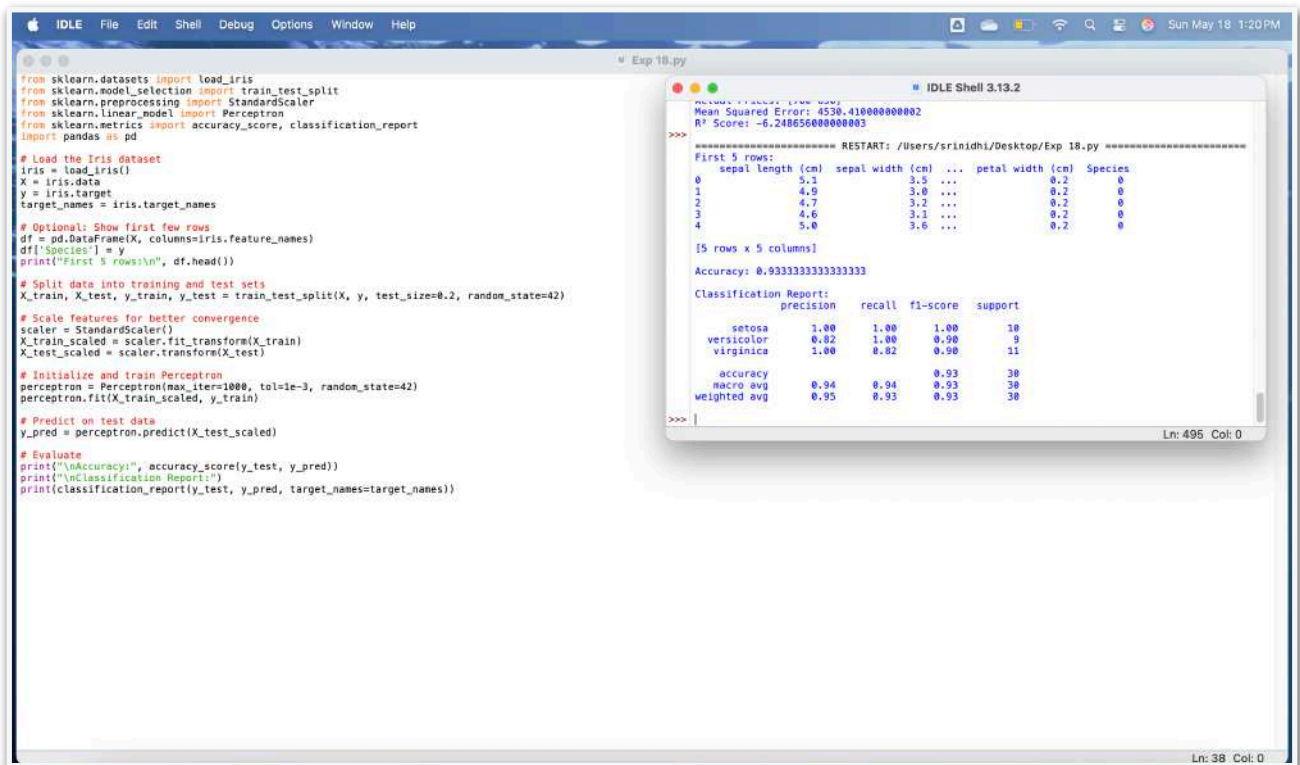
# Step 8: Evaluate the model
print("Predicted Prices:", y_pred)
print("Actual Prices:", y_test.values)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```

Accuracy: 1.0				
Classification Report:				
	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolour	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy	1.00	1.00	1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 16.py =====
Comparison of Classification Algorithms:
=====
Accuracy ... F1-Score (macro avg)
Naive Bayes 1.000000 ... 1.000000
Support Vector Machine 0.966667 ... 0.966667
Decision Tree 1.000000 ... 1.000000
Random Forest 1.000000 ... 1.000000
K-Nearest Neighbors 1.000000 ... 1.000000
[5 rows x 4 columns]
>>>
===== RESTART: /Users/srinidhi/Desktop/Exp 17.py =====
Predicted Prices: [684.9 645.9]
Actual Prices: [700 650]
Mean Squared Error: 4530.410000000002
R^2 Score: -6.248650000000003
>>>
```

## 18. Implement Perceptron based IRIS classification

Output:



```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

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

# Optional: Show first few rows
df = pd.DataFrame(X, columns=iris.feature_names)
df['Species'] = y
print("First 5 rows:\n", df.head())

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features for better convergence
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train Perceptron
perceptron = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perceptron.fit(X_train_scaled, y_train)

# Predict on test data
y_pred = perceptron.predict(X_test_scaled)

# Evaluate
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 18.py =====

First 5 rows:

	sepal length (cm)	sepal width (cm)	...	petal width (cm)	Species
0	5.1	3.5	...	0.2	0
1	4.9	3.0	...	0.2	0
2	4.7	3.2	...	0.2	0
3	4.6	3.1	...	0.2	0
4	5.0	3.6	...	0.2	0

[5 rows x 5 columns]

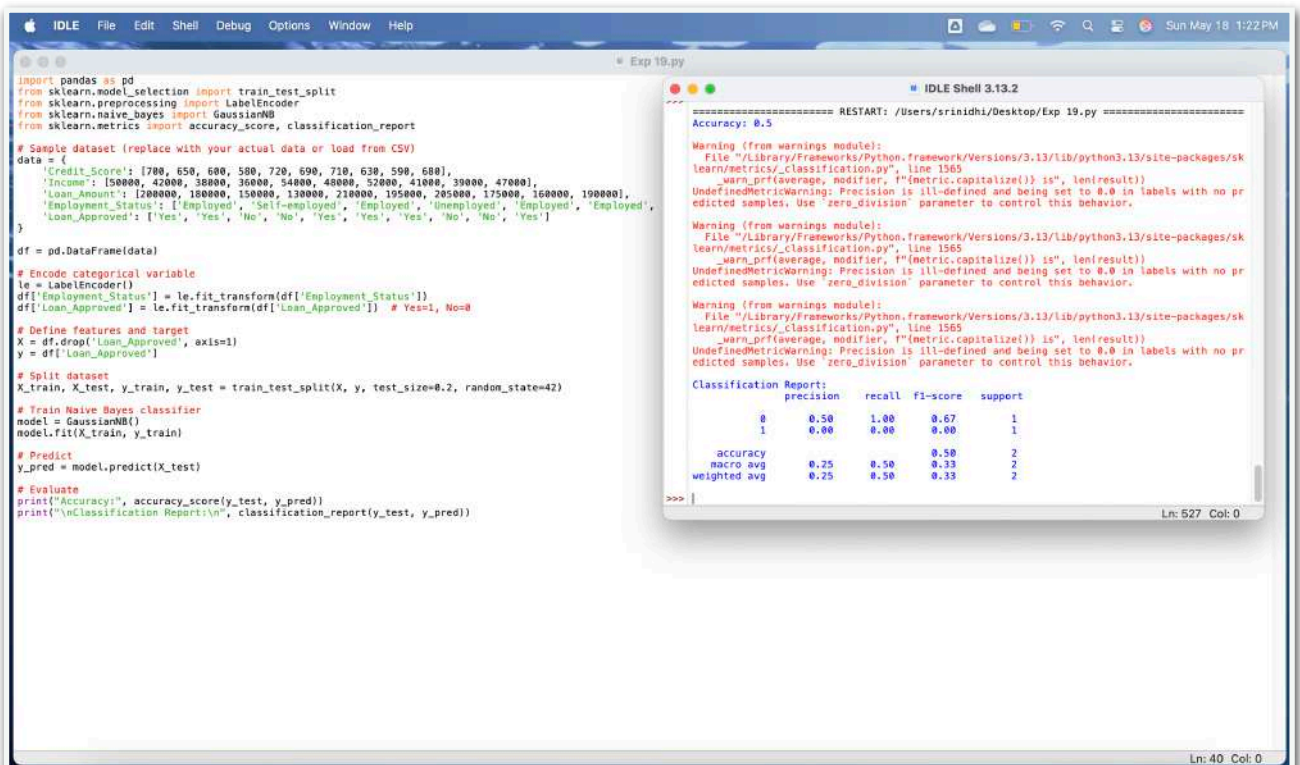
Accuracy: 0.9333333333333333

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	0.82	1.00	0.90	9
virginica	1.00	0.82	0.90	11
accuracy			0.93	30
macro avg	0.94	0.94	0.93	30
weighted avg	0.95	0.93	0.93	30

## 19. Implementation of Naive Bayes classification for Bank Loan prediction

Output:



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Sample dataset (replace with your actual data or load from CSV)
data = {
    'Credit_Score': [700, 650, 600, 580, 720, 690, 710, 630, 590, 680],
    'Income': [50000, 42000, 38000, 36000, 54000, 48000, 52000, 41000, 39000, 47000],
    'Loan_Amount': [200000, 180000, 150000, 130000, 210000, 195000, 205000, 175000, 160000, 190000],
    'Employment_Status': ['Employed', 'Self-employed', 'Employed', 'Unemployed', 'Employed', 'Employed'],
    'Loan_Approved': ['Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Encode categorical variable
le = LabelEncoder()
df['Employment_Status'] = le.fit_transform(df['Employment_Status'])
df['Loan_Approved'] = le.fit_transform(df['Loan_Approved']) # Yes=1, No=0

# Define features and target
X = df.drop('Loan_Approved', axis=1)
y = df['Loan_Approved']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

===== RESTART: /Users/srinidhi/Desktop/Exp 19.py =====

Accuracy: 0.5

Classification Report:

	precision	recall	f1-score	support
0	0.50	1.00	0.67	1
1	0.00	0.00	0.00	1
accuracy			0.50	2
macro avg	0.25	0.50	0.33	2
weighted avg	0.25	0.50	0.33	2

## 20. Implement Future Sales Prediction using a suitable machine learning algorithm

Output:

