Experiment – 1

```
import csv

def find_s_algorithm(filename):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        data = list(reader)

    attributes = data[0][:-1]
    examples = data[1:]

    hypothesis = ['0'] * len(attributes)

    for row in examples:
        if row[-1].lower() == 'yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] == '0':
                    hypothesis[i] = row[i]
                elif hypothesis[i] != row[i]:
                    hypothesis[i] = '?'

    print("Most specific hypothesis is:", hypothesis)

find_s_algorithm('finds_dataset.csv')
```

Output

```
Most specific hypothesis is: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Experiment – 2

```
import csv

def candidate_elimination(filename):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        data = list(reader)

    attributes = data[0][:-1]
    examples = data[1:]

    S = ['0'] * len(attributes)
    G = [['?'] * len(attributes)]

    for row in examples:
        instance, label = row[:-1], row[-1]
        if label.lower() == 'yes':
            for i in range(len(S)):
                if S[i] == '0':
                    S[i] = instance[i]
                elif S[i] != instance[i]:
                    S[i] = '?'
            G = [g for g in G if all(s == '?' or s == g[i] or g[i] == '?' for i, s in enumerate(S))]
        elif label.lower() == 'no':
            new_G = []
            for g in G:
                for i in range(len(g)):
                    if g[i] == '?':
                        if S[i] != '?':
                            new_hypothesis = g.copy()
                            new_hypothesis[i] = S[i]
                            if new_hypothesis not in new_G:
                                new_G.append(new_hypothesis)
            G = new_G

    print("Final Specific hypothesis S:", S)
    print("Final General hypotheses G:")
    for g in G:
        print(g)

candidate_elimination('cand_elim_dataset.csv')
```

Output

```
Final Specific hypothesis S: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General hypotheses G:
['Sunny', '?', '?', '?', '?', '?']
['?', 'Warm', '?', '?', '?', '?']
['?', '?', '?', 'Strong', '?', '?']
['?', '?', '?', '?', 'Warm', '?']
['?', '?', '?', '?', '?', 'Same']
```

Experiment – 3

```
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.preprocessing import LabelEncoder
import pandas as pd

data = pd.read_csv('id3_dataset.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

le = LabelEncoder()
X = X.apply(le.fit_transform)
y = le.fit_transform(y)

model = DecisionTreeClassifier(criterion="entropy")
model.fit(X, y)

print(export_text(model, feature_names=list(data.columns[:-1])))

sample = [[2, 1, 0, 1]]  # Example: Overcast, Cool, Normal, Strong
print("Prediction:", model.predict(sample))
```

Output

```
|--- Outlook <= 0.50
|   |--- class: 1
|--- Outlook >  0.50
|   |--- Humidity <= 0.50
|   |   |--- Outlook <= 1.50
|   |   |   |--- Wind <= 0.50
|   |   |   |   |--- class: 0
|   |   |   |--- Wind >  0.50
|   |   |   |   |--- class: 1
|   |   |--- Outlook >  1.50
|   |   |   |--- class: 0
|   |--- Humidity >  0.50
|   |   |--- Wind <= 0.50
|   |   |   |--- Temperature <= 1.00
|   |   |   |   |--- class: 0
|   |   |   |--- Temperature >  1.00
|   |   |   |   |--- class: 1
|   |   |--- Wind >  0.50
|   |   |   |--- class: 1
```

Experiment – 4

```
import numpy as np

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

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

np.random.seed(1)
weights_input_hidden = 2 * np.random.random((2, 4)) - 1
weights_hidden_output = 2 * np.random.random((4, 1)) - 1

for epoch in range(10000):
    hidden_input = np.dot(X, weights_input_hidden)
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output)
    output = sigmoid(final_input)

    error = y - output
    d_output = error * sigmoid_derivative(output)
    d_hidden = d_output.dot(weights_hidden_output.T) * sigmoid_derivative(hidden_output)

    weights_hidden_output += hidden_output.T.dot(d_output)
    weights_input_hidden += X.T.dot(d_hidden)

print("Final output after training:")
print(output)
```

Output

```
Final output after training:
[[0.02152435]
 [0.98341551]
 [0.98034016]
 [0.01804214]]
```

Experiment – 5

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('knn_dataset.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

le = LabelEncoder()
y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

print("Predicted labels:", model.predict(X_test))
print("Actual labels:", y_test.tolist())
```

Output

```
Predicted labels: [1 0]
Actual labels: [1, 0]
```

Experiment – 6

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

data = pd.read_csv('naive_bayes_dataset.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Encode categorical variables
le = LabelEncoder()
X = X.apply(le.fit_transform)
y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output

```
Confusion Matrix:
 [[1 2]
 [1 1]]
Accuracy: 0.4
```

Experiment – 7

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = pd.read_csv('logistic_regression_dataset.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Predicted:", y_pred)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output

```
Predicted: [1 1 1]
Accuracy: 1.0
```

Experiment – 8

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

data = pd.read_csv('linear_regression_dataset.csv')
X = data[['Experience']]
y = data['Salary']

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

print("Slope:", model.coef_)
print("Intercept:", model.intercept_)

plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.title("Linear Regression - Experience vs Salary")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```

Output

```
========================== RES
Slope: [4224.24242424]
Intercept: 35666.66666666667
```

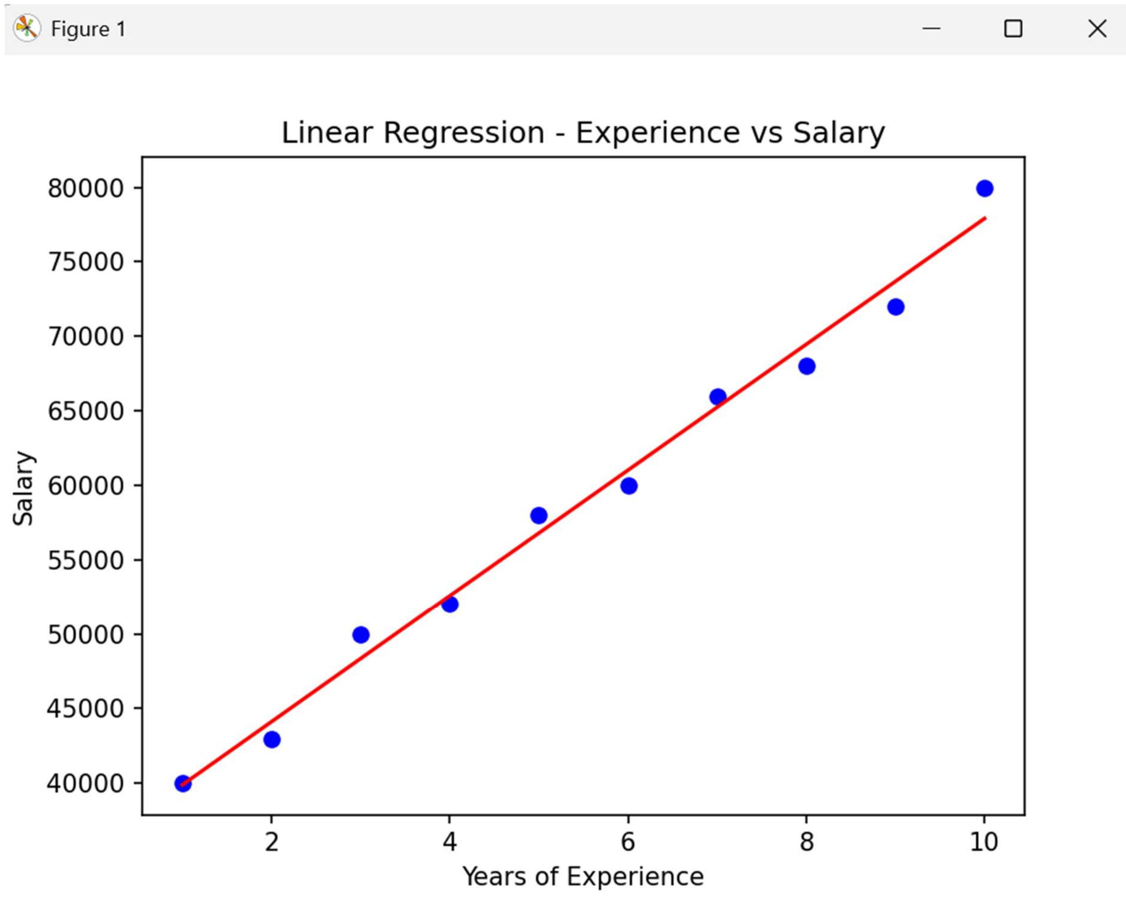
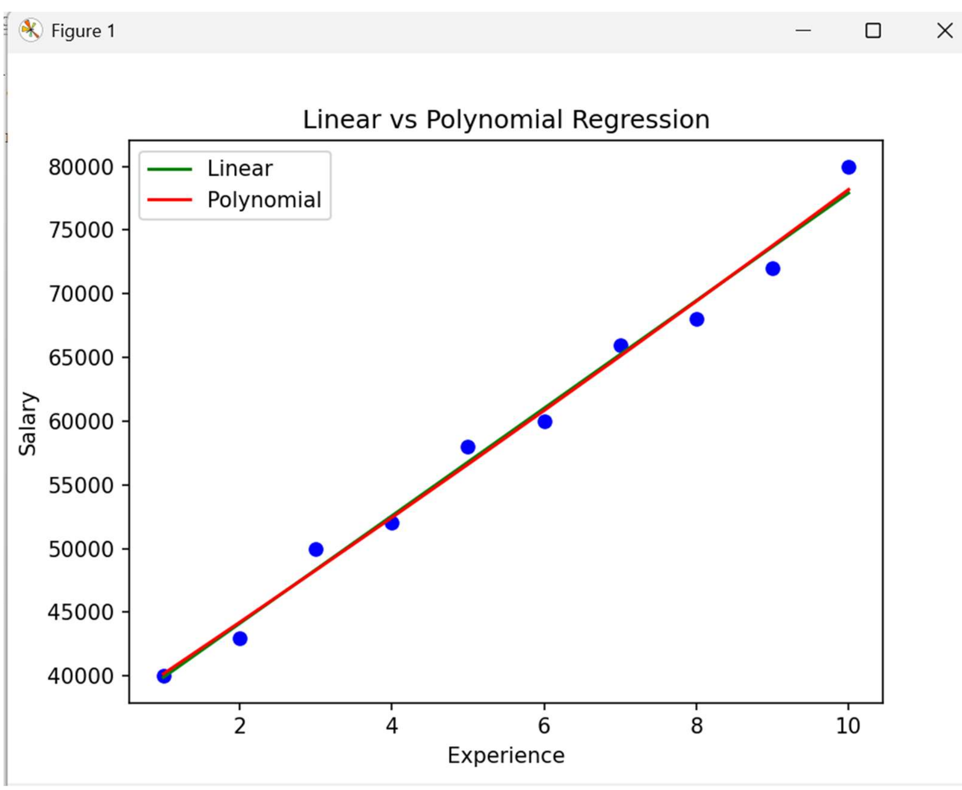


Figure 1
Linear Regression - Experience vs Salary
80000
75000
70000
65000
60000
55000
50000
45000
40000
Salary
2
4
6
8
10
Years of Experience

Experiment – 9

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

data = pd.read_csv('linear_regression_dataset.csv')
X = data[['Experience']]
y = data['Salary']

# Linear Regression
lin_model = LinearRegression()
lin_model.fit(X, y)

# Polynomial Regression
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
poly_model = LinearRegression()
poly_model.fit(X_poly, y)

# Plot
plt.scatter(X, y, color='blue')
plt.plot(X, lin_model.predict(X), label='Linear', color='green')
plt.plot(X, poly_model.predict(X_poly), label='Polynomial', color='red')
plt.legend()
plt.title("Linear vs Polynomial Regression")
plt.xlabel("Experience")
plt.ylabel("Salary")
plt.show()
```

Output



Experiment – 10

```
import pandas as pd
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

data = pd.read_csv('em_dataset.csv')
X = data[['Height']]

model = GaussianMixture(n_components=2)
model.fit(X)
labels = model.predict(X)

plt.scatter(X, [0]*len(X), c=labels, cmap='coolwarm')
plt.title("GMM Clustering via EM Algorithm")
plt.xlabel("Height")
plt.show()

print("Means of clusters:", model.means_.flatten())
print("Weights:", model.weights_)
```

Output