```
import time
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

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

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

# Initialize models
models = {
    'Decision Tree': DecisionTreeClassifier(),
    'Logistic Regression': LogisticRegression(max_iter=200),
    'KNN': KNeighborsClassifier()
}

# Dictionary to store the results
results = {}

# Train and evaluate each model
for model_name, model in models.items():
    start_time = time.time()
    model.fit(X_train, y_train)
    training_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    prediction_time = time.time() - start_time

    accuracy = accuracy_score(y_test, y_pred)

    results[model_name] = {
        'accuracy': accuracy,
        'training_time': training_time,
        'prediction_time': prediction_time
    }

# Create a DataFrame to compare the results
results_df = pd.DataFrame(results).T
results_df = results_df[['accuracy', 'training_time', 'prediction_time']]

print(results_df)
```

```
                     accuracy  training_time  prediction_time
Decision Tree             1.0       0.011393         0.000276
Logistic Regression       1.0       0.043086         0.000322
KNN                       1.0       0.005454         0.036225
```

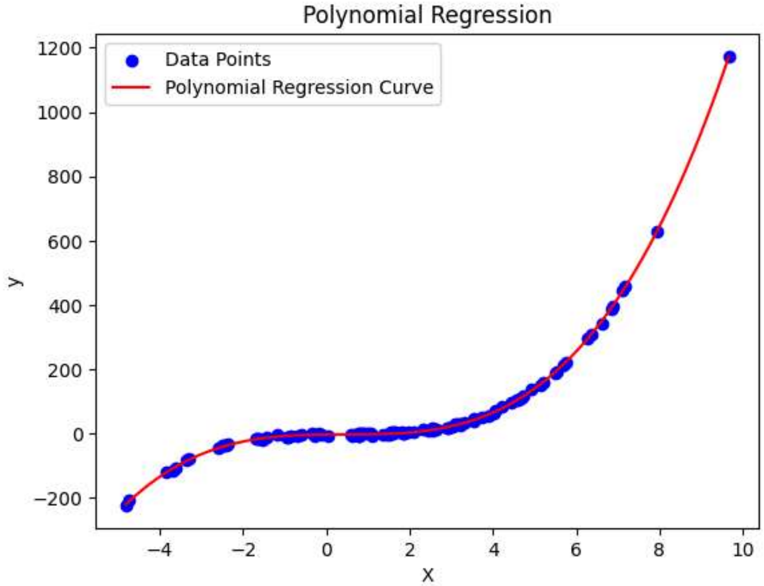
```
import pandas as pd

# Load the dataset
data = {
    'Example': [1, 2, 3, 4],
    'Citations': ['Some', 'Many', 'Many', 'Many'],
    'Size': ['Small', 'Big', 'Medium', 'Small'],
    'In Library': ['No', 'No', 'No', 'No'],
    'Price': ['Affordable', 'Expensive', 'Expensive', 'Affordable'],
    'Editions': ['Few', 'Many', 'Few', 'Many'],
    'Buy': ['No', 'Yes', 'Yes', 'Yes']
}
df = pd.DataFrame(data)
df.set_index('Example', inplace=True)

# Extract features and target
features = df.columns[:-1]
target = df.columns[-1]

# Initialize the most specific and most general hypotheses
def get_most_specific_hypothesis(num_features):
```

```
    return ['0'] * num_features

def get_most_general_hypothesis(num_features):
    return ['?'] * num_features

S = get_most_specific_hypothesis(len(features))
G = [get_most_general_hypothesis(len(features))]

# Function to check if one hypothesis is more general than another
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == '?' or (x != '0' and (x == y or y == '0'))
        more_general_parts.append(mg)
    return all(more_general_parts)

# Update S and G based on the training examples
for index, row in df.iterrows():
    if row[target] == 'Yes':  # Positive example
        # Remove hypotheses from G that are inconsistent with the example
        G = [g for g in G if more_general(g, row[:-1].values)]
        # Update S to be consistent with the example
        for i, val in enumerate(row[:-1]):
            if S[i] == '0':
                S[i] = val
            elif S[i] != val:
                S[i] = '?'
        # Remove more general hypotheses from G
        G = [g for g in G if more_general(g, S)]
    else:  # Negative example
        # Update G to be consistent with the example
        G_new = []
        for g in G:
            for i, val in enumerate(row[:-1]):
                if g[i] == '?':
                    for x in set(df[features[i]]):
                        if x != val:
                            new_hypothesis = g.copy()
                            new_hypothesis[i] = x
                            if more_general(new_hypothesis, S):
                                G_new.append(new_hypothesis)
                elif g[i] != val:
                    G_new.append(g)
        G = G_new
        G = [g for g in G if more_general(g, S)]

print(f"Most Specific Hypothesis S: {S}")
print(f"Set of General Hypotheses G: {G}")
```

```
Most Specific Hypothesis S: ['Many', '?', 'No', '?', '?']
Set of General Hypotheses G: [['Many', '?', '?', '?', '?']]
```

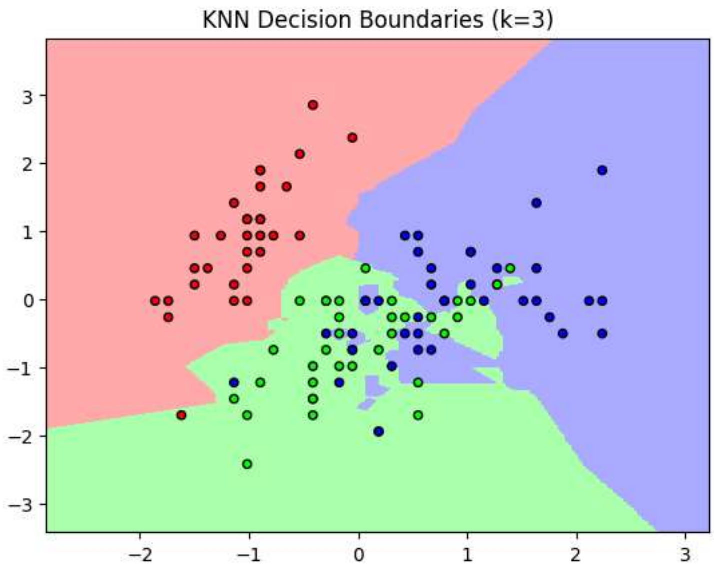
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generate a synthetic dataset
np.random.seed(0)
X = 2 - 3 * np.random.normal(0, 1, 100)
y = X - 2 * (X ** 2) + 1.5 * (X ** 3) + np.random.normal(-3, 3, 100)

X = X[:, np.newaxis]

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

# Transform features to polynomial features
poly = PolynomialFeatures(degree=3)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

# Train the polynomial regression model
model = LinearRegression()
model.fit(X_poly_train, y_train)

# Make predictions
y_train_pred = model.predict(X_poly_train)
y_test_pred = model.predict(X_poly_test)

# Evaluate the model
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print(f"Training MSE: {mse_train}")
print(f"Training R^2: {r2_train}")
print(f"Test MSE: {mse_test}")
print(f"Test R^2: {r2_test}")

# Visualize the results
plt.scatter(X, y, color='blue', label='Data Points')
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_poly_range = poly.transform(X_range)
y_range_pred = model.predict(X_poly_range)
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Regression Curve')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Polynomial Regression')
plt.show()
```

```
Training MSE: 9.063856548037108
Training R^2: 0.9997112440964597
Test MSE: 11.180742120161817
Test R^2: 0.9995350181165003
```



```
import numpy as np
import pandas as pd
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, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

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

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

# Standardize the features (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Visualize the results (for the first two features)
def plot_decision_boundaries(X, y, model, title="Decision Boundaries"):
    h = 0.02  # step size in the mesh
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    # Create mesh grid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Predict for each point in the mesh grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(title)
    plt.show()

# Only visualize using the first two features
X_vis = X_train[:, :2]
y_vis = y_train
knn_vis = KNeighborsClassifier(n_neighbors=3)
knn_vis.fit(X_vis, y_vis)

plot_decision_boundaries(X_vis, y_vis, knn_vis, title="KNN Decision Boundaries (k=3)")
```

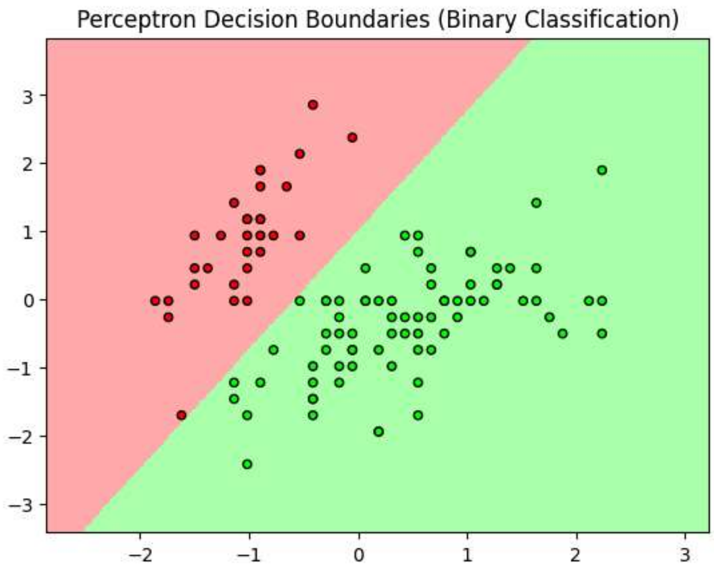
```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```



```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

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

# For simplicity, let's reduce the problem to binary classification (e.g., Setosa vs. Non-Setosa)
# This is necessary because the Perceptron is a binary classifier
y_binary = (y != 0).astype(int)  # Setosa is 0, Versicolor and Virginica are 1

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_binary, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Perceptron model
perceptron = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perceptron.fit(X_train, y_train)

# Make predictions
y_pred_train = perceptron.predict(X_train)
y_pred_test = perceptron.predict(X_test)

# Evaluate the model
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)
print(f"Training Accuracy: {accuracy_train}")
print(f"Test Accuracy: {accuracy_test}")
print("Classification Report (Test Set):")
print(classification_report(y_test, y_pred_test))
print("Confusion Matrix (Test Set):")
print(confusion_matrix(y_test, y_pred_test))

# Visualize the decision boundaries for the first two features (optional)
def plot_decision_boundaries(X, y, model, title="Decision Boundaries"):
    h = 0.02  # step size in the mesh
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00'])

    # Create mesh grid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Predict for each point in the mesh grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(title)
    plt.show()

# Only visualize using the first two features
X_vis = X_train[:, :2]
y_vis = y_train
perceptron_vis = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perceptron_vis.fit(X_vis, y_vis)

plot_decision_boundaries(X_vis, y_vis, perceptron_vis, title="Perceptron Decision Boundaries (Binary Classification)")
```

```
# Note: For multi-class classification, a one-vs-rest strategy can be implemented.
```

```
Training Accuracy: 1.0
Test Accuracy: 1.0
Classification Report (Test Set):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        26

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix (Test Set):
[[19  0]
 [ 0 26]]
```




```
import pandas as pd

# Create the dataset
data = {
    'Example': [1, 2, 3, 4],
    'Citations': ['Some', 'Many', 'Many', 'Many'],
    'Size': ['Small', 'Big', 'Medium', 'Small'],
    'In Library': ['No', 'No', 'No', 'No'],
    'Price': ['Affordable', 'Expensive', 'Expensive', 'Affordable'],
    'Editions': ['Few', 'Many', 'Few', 'Many'],
    'Buy': ['No', 'Yes', 'Yes', 'Yes']
}

df = pd.DataFrame(data)

# Extract the features and labels
features = df.columns[1:-1]
X = df[features]
y = df['Buy']

# Initialize the most specific hypothesis
hypothesis = ['0'] * len(features)

# Find-S algorithm
for i, row in X.iterrows():
    if y[i] == 'Yes':  # Only consider positive examples
        for j in range(len(hypothesis)):
            if hypothesis[j] == '0':
                hypothesis[j] = row[j]
            elif hypothesis[j] != row[j]:
                hypothesis[j] = '?'

print(f"Most specific hypothesis: {hypothesis}")
```

```
Most specific hypothesis: ['Many', '?', 'No', '?', '?']
```

```python
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression, make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report
import matplotlib.pyplot as plt

# Generate a synthetic dataset for regression
X_reg, y_reg = make_regression(n_samples=100, n_features=1, noise=0.1, random_state=42)

# Generate a synthetic dataset for classification
X_clf, y_clf = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, n_classes=2, random_state=42)

# Split the regression data into training and test sets
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y_reg, test_size=0.3, random_state=42)

# Split the classification data into training and test sets
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X_clf, y_clf, test_size=0.3, random_state=42)

# Train Linear Regression model
linear_reg = LinearRegression()
linear_reg.fit(X_train_reg, y_train_reg)
y_pred_reg = linear_reg.predict(X_test_reg)

# Evaluate Linear Regression model
mse_reg = mean_squared_error(y_test_reg, y_pred_reg)
print(f"Linear Regression MSE: {mse_reg}")

# Train Logistic Regression model
logistic_reg = LogisticRegression()
logistic_reg.fit(X_train_clf, y_train_clf)
y_pred_clf = logistic_reg.predict(X_test_clf)

# Evaluate Logistic Regression model
accuracy_clf = accuracy_score(y_test_clf, y_pred_clf)
print(f"Logistic Regression Accuracy: {accuracy_clf}")
print("Classification Report:")
print(classification_report(y_test_clf, y_pred_clf))

# Visualize the results for Linear Regression
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_test_reg, y_test_reg, color='blue', label='Actual')
plt.plot(X_test_reg, y_pred_reg, color='red', label='Predicted')
plt.title('Linear Regression')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()

# Visualize the results for Logistic Regression
plt.subplot(1, 2, 2)
plt.scatter(X_test_clf[:, 0], X_test_clf[:, 1], c=y_test_clf, cmap='viridis', marker='o', edgecolor='k', s=100, label='Actual')
plt.scatter(X_test_clf[:, 0], X_test_clf[:, 1], c=y_pred_clf, cmap='coolwarm', marker='x', edgecolor='k', s=100, label='Predicted')
plt.title('Logistic Regression')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

plt.show()
```

```
Linear Regression MSE: 0.010347302683438472
Logistic Regression Accuracy: 0.9666666666666667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        16
           1       0.93      1.00      0.97        14

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

<ipython-input-9-257a282f56c5>:55: UserWarning: You passed a edgecolor/edgecolors ('k')
  plt.scatter(X_test_clf[:, 0], X_test_clf[:, 1], c=y_pred_clf, cmap='coolwarm', marker
```