```
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
import time

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = 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=0.2, random_state=42)

# Initialize classifiers
models = {
    'Decision Tree': DecisionTreeClassifier(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'KNN': KNeighborsClassifier()
}

# DataFrame to store results
results = pd.DataFrame(columns=['Algorithm', 'Accuracy', 'Training Time (s)', 'Prediction T

# Train and evaluate each model
for algo_name, model in models.items():
    # Measure training time
    start_train = time.time()
    model.fit(X_train, y_train)
    end_train = time.time()
    train_time = end_train - start_train

    # Measure prediction time
    start_pred = time.time()
    y_pred = model.predict(X_test)
    end_pred = time.time()
    pred_time = end_pred - start_pred

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Create a DataFrame with the current results
    current_results = pd.DataFrame([{
        'Algorithm': algo_name,
        'Accuracy': accuracy,
        'Training Time (s)': train_time,
        'Prediction Time (s)': pred_time
    }])
```

```
    # Concatenate with the main results DataFrame
    results = pd.concat([results, current_results], ignore_index=True)

# Display results
print("Performance Comparison of Classification Algorithms on Iris Dataset:\n")
print(results)
```

```
Performance Comparison of Classification Algorithms on Iris Dataset:

                Algorithm  Accuracy  Training Time (s)  Prediction Time (s)
0           Decision Tree       1.0           0.005356             0.000264
1     Logistic Regression       1.0           0.046542             0.000318
2                     KNN       1.0           0.001233             0.004997
```

```
import numpy as np

def candidate_elimination(examples):
    num_attributes = len(examples[0]) - 1  # Number of attributes in the examples
    specific_hypothesis = ['0'] * num_attributes  # most specific hypothesis
    general_hypothesis = [['?'] * num_attributes]  # most general hypothesis

    for example in examples:
        if example[-1] == 'Yes':  # positive example
            for i in range(num_attributes):
                if specific_hypothesis[i] == '0':
                    specific_hypothesis[i] = example[i]
                elif specific_hypothesis[i] != example[i]:
                    specific_hypothesis[i] = '?'
            # Remove inconsistent hypotheses from general_hypothesis
            general_hypothesis = [g for g in general_hypothesis if all(g[i] == '?' or g[i]
        else:  # negative example
            new_general_hypotheses = []
            for g in general_hypothesis:
                for i in range(num_attributes):
                    if g[i] == '?':
                        if example[i] != specific_hypothesis[i]:
                            new_hypothesis = g[:]
                            new_hypothesis[i] = specific_hypothesis[i]
                            if new_hypothesis not in new_general_hypotheses:
                                new_general_hypotheses.append(new_hypothesis)
            general_hypothesis = new_general_hypotheses

    return specific_hypothesis, general_hypothesis

# Example dataset
dataset = [
    ['Some', 'Small', 'No', 'Affordable', 'Few', 'No'],
    ['Many', 'Big', 'No', 'Expensive', 'Many', 'Yes'],
    ['Many', 'Medium', 'No', 'Expensive', 'Few', 'Yes'],
    ['Many', 'Small', 'No', 'Affordable', 'Many', 'Yes']
]

# Applying the Candidate-Elimination algorithm
specific, general = candidate_elimination(dataset)
print("Specific hypothesis:", specific)
print("General hypotheses:", general)
```

```
Specific hypothesis: ['Many', '?', 'No', '?', '?']
General hypotheses: []
```

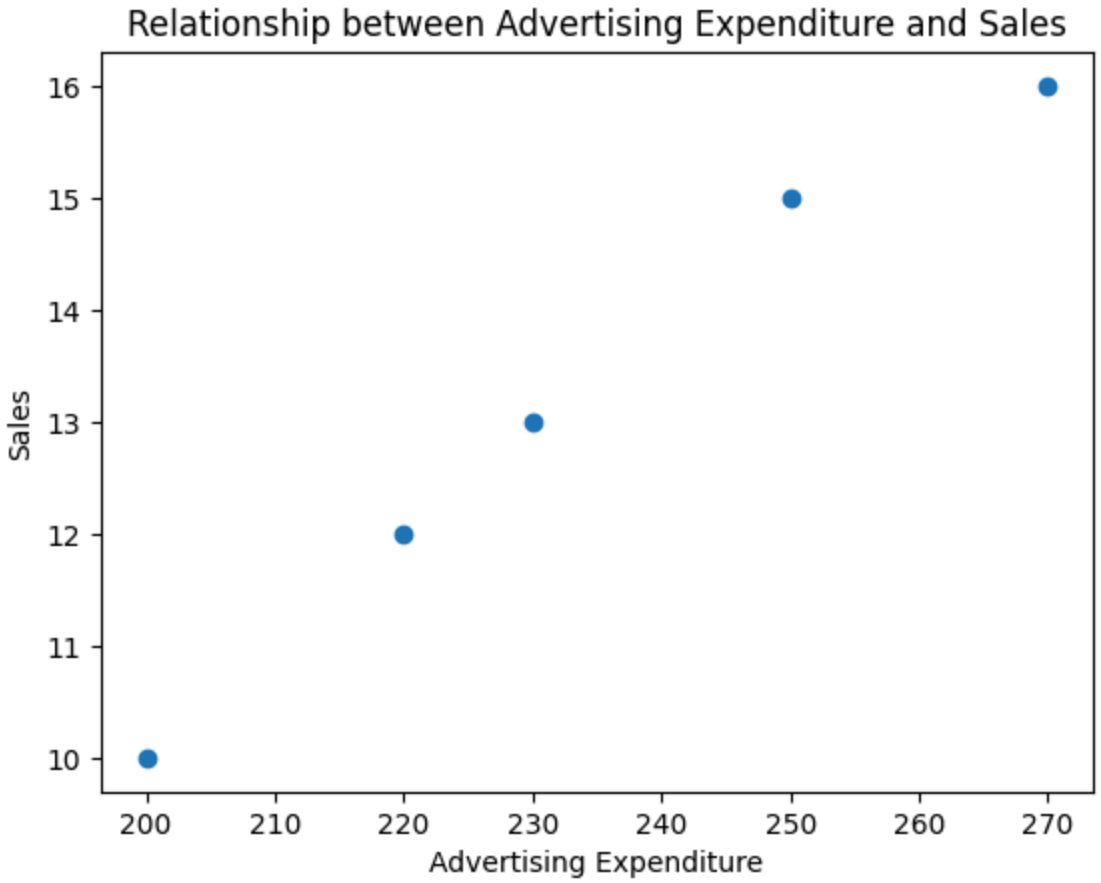
```
import pandas as pd

# Define 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)

# Initialize the hypothesis
hypothesis = None

# Find-S algorithm
for index, row in df.iterrows():
    if row['Buy'] == 'Yes':
        if hypothesis is None:
            hypothesis = row.drop('Example')
        else:
            for attr in hypothesis.index:
                if hypothesis[attr] != row[attr]:
                    hypothesis[attr] = '?'

# Output the hypothesis
print("The most specific hypothesis is:")
print(hypothesis)
```

```
The most specific hypothesis is:
Citations     Many
Size             ?
In Library      No
Price            ?
Editions         ?
Buy            Yes
Name: 1, dtype: object
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Assuming 'data' is your dictionary with keys 'sales', 'advertising', and 'quarter'
data_dict = {
    'sales': [10, 12, 13, 15, 16],
    'advertising': [200, 220, 230, 250, 270],
    'quarter': [1, 2, 3, 4, 1]
}

# Convert dictionary to DataFrame
data = pd.DataFrame(data_dict)

# a) Print the first five rows
print(data.head())

# b) Basic statistical computations
print(data.describe())

# c) Columns and their data types
print(data.dtypes)

# d) Explore the data using scatter plot
plt.scatter(data['advertising'], data['sales'])
plt.xlabel('Advertising Expenditure')
plt.ylabel('Sales')
plt.title('Relationship between Advertising Expenditure and Sales')
plt.show()

# e) Handle null values by replacing with mode (if any)
data = data.fillna(data.mode().iloc[0])

# f) Split data into train and test sets (assuming 80-20 split)
X = data[['advertising']]  # Feature
y = data['sales']          # Target

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

# g) Train and predict using linear regression
model = LinearRegression()
model.fit(X_train, y_train)

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

# Print the coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
   sales  advertising  quarter
0     10          200        1
1     12          220        2
2     13          230        3
3     15          250        4
4     16          270        1
            sales  advertising  quarter
count   5.000000     5.000000  5.00000
mean   13.200000   234.000000  2.20000
std     2.387467    27.018512  1.30384
min    10.000000   200.000000  1.00000
25%    12.000000   220.000000  1.00000
50%    13.000000   230.000000  2.00000
75%    15.000000   250.000000  3.00000
max    16.000000   270.000000  4.00000
sales          int64
advertising    int64
quarter        int64
dtype: object
```




```
Coefficients: [0.08785047]
Intercept: -7.3644859813084125
Mean Squared Error: 0.001397501965237305
```

```
import pandas as pd

# Define the dataset
data = {
    'Example': [1, 2, 3, 4],
    'Shape': ['Circular', 'Circular', 'Oval', 'Oval'],
    'Size': ['Large', 'Large', 'Large', 'Large'],
    'Color': ['Light', 'Light', 'Dark', 'Light'],
    'Surface': ['Smooth', 'Irregular', 'Smooth', 'Irregular'],
    'Thickness': ['Thick', 'Thick', 'Thin', 'Thick'],
    'Target Concept': ['+', '+', '-', '+']
}

df = pd.DataFrame(data)

# Initialize the hypothesis
hypothesis = ['0', '0', '0', '0', '0']
```