Python Code:

```
import pandas as pd

# Load data
data = pd.read_csv("Weather.csv", sep=',', header=None)

# Extract features and labels
X = data.values[:, 0:6]

# Generalization function
def Generalize(S_in, Exin):
    features = len(Exin)
    for i in range(features):
        if(S_in[i] != Exin[i]):
            S_in[i]='?'
        else:
            S_in[i]=Exin[i]
    return(S_in)

# Initialize hypothesis
S = ['?','?','?','?','?','?']

# Find the first positive example
datalen = len(data)
for i in range(datalen):
    ex = data.values[i,:]
    if(ex[-1]=='Yes'):
        S = X[i,:]
        break

# Generalize the hypothesis
for i in range(datalen-1):
    ex = data.values[i+1,:]
    if(ex[-1] == 'Yes'):
        S = Generalize(S, X[i+1,:])

# Print the final hypothesis
print('Specific Hypothesis = ',S)
```

Python Code:

```
import pandas as pd

# Load the dataset
data = pd.read_csv("Customer_Preference.csv")
instances = data.values.tolist()

def candidate_elimination(instances):
    S = ['0'] * (len(instances[0]) - 1)  # Most specific hypothesis
    G = [['?'] * (len(instances[0]) - 1)]  # Most general hypothesis

    for instance in instances:
        x, label = instance[:-1], instance[-1]

        if label == "Yes":  # Positive example
            for i in range(len(S)):
                if S[i] == '0':
                    S[i] = x[i]
                elif S[i] != x[i]:
                    S[i] = '?'
            G = [g for g in G if all(g[i] == '?' or g[i] == x[i] for i in range(len(g)))]
        else:  # Negative example
            new_G = []
            for g in G:
                for i in range(len(g)):
                    if g[i] == '?':
                        for value in set(data.iloc[:, i]):
                            if value != x[i]:
                                new_hypothesis = g[:]
                                new_hypothesis[i] = value
                                new_G.append(new_hypothesis)
            G = [g for g in new_G if any(S[i] == '?' or g[i] == '?' or g[i] == S[i] for i in
range(len(S)))]

    return S, G
```



```
S, G = candidate_elimination(instances)
print("Final Specific Hypothesis (S):", S)
print("Final General Hypotheses (G):", G)
```

Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv('auto-mpg.csv')

# Handle missing values
df.replace('?', float('nan'), inplace=True)
df.dropna(inplace=True)

# Select features and target
X = df[['model year']]
y = df['car name'].str.contains('chevrolet chevelle malibu')

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

# Scale features (optional)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict on test data
y_pred = knn.predict(X_test)
```

Lab Manual

```
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")

# Predict for new car (model year 70)
new_car_data = [[70]]
new_car_data_scaled = scaler.transform(new_car_data)
prediction = knn.predict(new_car_data_scaled)
print(f"Prediction for model year 70: {prediction[0]}")
```

Python Code:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
try:
  df = pd.read_csv('auto-mpg.csv')
except FileNotFoundError:
  print("Error: 'auto-mpg.csv' not found. Please ensure the file is in the correct
location.")
  exit()

# Handle missing values
df.replace('?', pd.NA, inplace=True)
df.dropna(inplace=True)

# Prepare features and target
X = df[['weight', 'horsepower']]
y = df['mpg'] > 15

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

# Train decision tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on test set
y_pred = clf.predict(X_test)
```



```
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Predict for all cars and identify exceeding 15 mpg
predictions = clf.predict(X)
cars_exceeding_15mpg = df[predictions & (df['mpg'] > 15)]
print("\nCars exceeding 15 mpg (based on prediction):")
print(cars_exceeding_15mpg[['car name']])
```

Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = pd.read_csv("sms_spam.csv")

# Display data sample
print("Dataset Sample:")
print(data.head())

# Separate features and labels
X = data['Message']
y = data['Label']

# Split data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Text vectorization (bag-of-words)
vectorizer = CountVectorizer(stop_words='english')
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Train Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vectorized, y_train)

# Make predictions on test set
y_pred = nb_classifier.predict(X_test_vectorized)

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



```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Example: predict new messages
new_messages = [
    "Congratulations! You've won a free ticket to Paris. Click here to claim.",
    "Hey, are we still meeting for coffee this afternoon?",
    "You are selected for a prize of $1000. Claim now."
]
new_messages_vectorized = vectorizer.transform(new_messages)
predictions = nb_classifier.predict(new_messages_vectorized)

print("\nPredictions for new messages:")
for message, label in zip(new_messages, predictions):
    print(f"Message: {message} --> Prediction: {label}")
```