1. Implement and demonstrate the Find-S algorithm for finding the most specific hypothesis.

```
import csv

sport=[]
with open('ENJOYSPORT.csv', newline='')as csvfile:
    read=csv.reader(csvfile)
    for reader in read:
        sport.append(reader)

hyp=['0']*(len(sport[0])-1)
step=0
for row in sport[1:]:
    if row[-1]=='1':
        print(f'Step {step}: {hyp}')
        for i in range(len(row)-1):
            if hyp[i]=='0':
                hyp[i]=row[i]
            elif hyp[i]!=row[i]:
                hyp[i]='?'
        step+=1

print(f'Step {step}: {hyp}')
```

2. Implement and demonstrate the Candidate Elimination algorithm using a data set stored as a .CSV file.

```
import csv

sport=[]
with open('ENJOYSPORT.csv',newline='') as csvfile:
    temp=csv.reader(csvfile)
    for row in temp:
        sport.append(row)

specific=['0']*(len(sport[0])-1)
general=[]
step=0
```

```
for tuple in sport[1:]:
    if tuple[-1]=='1':
        for i in range(len(specific)):
            if specific[i]=='0':
                specific[i]=tuple[i]
            elif specific[i]!=tuple[i]:
                specific[i]='?'
            for h in general:
                if any (h[j]!='?' and h[j]!=specific[j] for j in
range(len(specific))):
                    general.remove(h)

    if tuple[-1]=='0':
        for i in range(len(specific)):
            if tuple[i]!=specific[i]:
                r=[]
                for x in range(len(specific)):
                    if x==i: r.append(specific[i])
                    else: r.append('?')
                for x in range(0,6):
                    if r[i]!='?':
                        general.append(r)
                        break
    print('STEP',step,': ')
    print(f"Current Hypothesis:{tuple[:-1]}")
    print("Specific hypothesis:", specific)
    print("General hypothesis:", general)
    step+=1
    print('\n')
```

3. Demonstrate data Preprocessing (Data Cleaning, Integration and Transformation) operations on a suitable data.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

```
df=pd.read_csv('housing.csv')
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)

columns=df.columns
df2=[-122.10, 37.05, 58.0, 3503.0, 752.0, 1504.0, 734.0, 3.2705,
278800.0, 'NEAR BAY']
df2=pd.DataFrame([df2],columns=columns)
df=pd.concat([df,df2],ignore_index=True)

num_cols=['longitude','latitude','housing_median_age','total_rooms','to
tal_bedrooms','population','households','median_income','median_house_v
alue']
cat_cols=['ocean_proximity']

imputer=SimpleImputer(strategy='mean')
df[num_cols]=imputer.fit_transform(df[num_cols])
df=pd.get_dummies(df,columns=['ocean_proximity'],drop_first=True)
ss=StandardScaler()
df[num_cols] = ss.fit_transform(df[num_cols])

df.to_csv('processed_data.csv', index=False)
print(df.tail())
```

4. Demonstrate the working of SVM classifier for a suitable dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.datasets import load_iris
from sklearn.inspection import DecisionBoundaryDisplay
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
Y = iris.target
```

```
X = X[:, :2]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

svm = SVC(kernel='rbf', gamma='auto')
svm.fit(X_train, Y_train)
Y_pred = svm.predict(X_test)

print("Accuracy: ", accuracy_score(Y_test, Y_pred) * 100)
print(f'Classification Report:\n{classification_report(Y_test,
Y_pred)}')
print(f'Confusion matrix:\n{confusion_matrix(Y_test, Y_pred)}')

DecisionBoundaryDisplay.from_estimator(svm, X, cmap=plt.cm.Spectral,
alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('SVM Decision Boundary')
plt.show()
```

5. Implement and demonstrate the working of the Decision Tree algorithm.

```
from sklearn import tree
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn import tree

iris=load_iris()
X=iris.data
Y=iris.target
```

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,
random_state=42)

dt=DecisionTreeClassifier(
    max_depth=5,
    random_state=42,
    min_samples_leaf=5,
    max_leaf_nodes=5,
    min_samples_split=5
)

dt.fit(X_train,Y_train)

Y_pred=dt.predict(X_test)

new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])
new_sample_scaled = (new_sample)
dt_prediction = dt.predict(new_sample_scaled)

print(f'SVM Prediction for the new sample:
{iris.target_names[dt_prediction[0]]}')
print("Accuracy: ",accuracy_score(Y_test,Y_pred)*100)
print(f'Classification Report:{classification_report(Y_test,Y_pred)}')
print(f'Confusion matrix:\n{confusion_matrix(Y_test,Y_pred)}')

plt.figure(figsize=(8,10))
tree.plot_tree(dt, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
plt.show()
```

6. Implement Random Forest classifier using python programming.

```
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report,
confusion_matrix
from sklearn.datasets import load_iris
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree

iris=load_iris()
X=iris.data
print(iris.DESCR)
Y=iris.target

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,
random_state=42)

clf=RandomForestClassifier(
    n_estimators=5,
    random_state=42,
    min_samples_leaf=10,
    max_leaf_nodes=10,
    min_samples_split=5
)

clf.fit(X_train,Y_train)

Y_pred=clf.predict(X_test)

new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])
new_sample_scaled = (new_sample)
dt_prediction = clf.predict(new_sample_scaled)

print(f'SVM Prediction for the new sample:
{iris.target_names[dt_prediction[0]]}')
print("Accuracy: ",accuracy_score(Y_test,Y_pred)*100)
print(f'Classification Report:{classification_report(Y_test,Y_pred)}')
print(f'Confusion matrix:\n{confusion_matrix(Y_test,Y_pred)}')

plt.figure(figsize=(8,10))
tree.plot_tree(clf.estimators_[0], feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
plt.show()
```

7. Demonstrate the text classifier using Naive Bayes classifier algorithm.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import LabelEncoder

docs = [
    "I love programming in Python.",
    "Machine learning is fascinating.",
    "The weather today is sunny.",
    "Python is a versatile programming language.",
    "The news is about the latest technology trends.",
]
labels = ["Technology", "Technology", "Weather", "Technology", "News"]

label_encoder = LabelEncoder()
Y = label_encoder.fit_transform(labels)

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

clf = MultinomialNB()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)

print("Accuracy: ", accuracy_score(Y_test, Y_pred) * 100)
print(f'Classification Report:\n{classification_report(Y_test, Y_pred,
labels=label_encoder.classes_)}')
print(f'Confusion Matrix:\n{confusion_matrix(Y_test, Y_pred)}')

new_text = "Today is a great day to learn new technology."
new_text_transformed = vectorizer.transform([new_text])
```

```
ans = clf.predict(new_text_transformed)
print(f'Prediction for the new text: {label_encoder.classes_[ans[0]]}')
```

or..

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

newsgroups = fetch_20newsgroups(subset='all')
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(newsgroups.data)
y = newsgroups.target

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

nb_clf = MultinomialNB()
nb_clf.fit(X_train, y_train)
y_pred = nb_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=newsgroups.target_names))
print(f"Confusion Matrix:{confusion_matrix(y_test,y_pred)}")
```

8. Implement the Naive Bayesian classifier for a sample training data set stored as a .CSV file.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score,classification_report,
confusion_matrix
import matplotlib.pyplot as plt

data = pd.read_csv('opinion.csv')

X = data.iloc[:, :-1]
y = data.iloc[:, -1]

label_encoders = {}
for column in X.columns:
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
    label_encoders[column] = le

print("\nNow the train data is:\n", X.head())

le_target = LabelEncoder()
y = le_target.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy is:", accuracy)
print(f'Classification Report:{classification_report(y_test,y_pred)}')
print(f'Confusion matrix:\n{confusion_matrix(y_test,y_pred)}')
```

9. Construct a Bayesian network to analyze the diagnosis of heart patients using heart diseases dataset.

```
import numpy as np
import pandas as pd
from pgmpy.models import BayesianNetwork
```

```
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
import matplotlib.pyplot as plt

heartdisease = pd.read_csv('heart.csv')
heartdisease = heartdisease.replace('?', np.nan)

print('Few examples from the dataset are given below')
print(heartdisease.head())

model = BayesianNetwork([('age', 'trestbps'), ('age', 'fbs'),
                         ('sex', 'trestbps'), ('exang', 'trestbps'),
                         ('trestbps', 'heartdisease'),
                         ('fbs', 'heartdisease'),
                         ('heartdisease', 'restecg'),
                         ('heartdisease', 'thalach'),
                         ('chol', 'heartdisease')
                         ])

print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartdisease, estimator=MaximumLikelihoodEstimator)

print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

print('\n1. Probability of HeartDisease given Age=30')
q = HeartDisease_infer.query(variables=['heartdisease'],
evidence={'age': 30})
print(q.values[1])

print('\n2. Probability of HeartDisease given cholesterol=254')
q = HeartDisease_infer.query(variables=['heartdisease'],
evidence={'chol': 254})
print(q.values[1])
```

10. Implement KNN classification algorithm with an appropriate dataset and analyze the results.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

knn = KNeighborsClassifier()
param_grid = {'n_neighbors': range(1, 21)}

grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

best_knn = grid_search.best_estimator_
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

y_pred = best_knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
```

```
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

mean_scores = grid_search.cv_results_['mean_test_score']
k_values = param_grid['n_neighbors']

plt.figure(figsize=(10, 6))
plt.plot(k_values, mean_scores, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Cross-Validated Accuracy')
plt.title('KNN Classifier Performance for Different k Values')
plt.grid(True)
plt.xticks(k_values)
plt.show()
```

Datasets:

1. Opinion.csv:

Sky,AirTemp,Forecast,EnjoySport
Sunny,Cold,Change,1
Sunny,Warm,Same,0
Overcast,Cold,Change,1
Sunny,Warm,Same,0
Sunny,Warm,Same,1
Overcast,Warm,Same,0
Sunny,Warm,Change,1

2. Enjoysport.csv:

Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport
Sunny,Warm,Normal,Strong,Warm,Same,1
Sunny,Warm,High,Strong,Warm,Same,1
Rainy,Cold,High,Strong,Warm,Change,0
Sunny,Warm,High,Strong,Cool,Change,1