Experiment 6

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from wordcloud import WordCloud
import re
import string
#from textblob import TextBlob
import nltk
from nltk.corpus import stopwords
#import emoji
nltk.download('punkt')
nltk.download('wordnet')
from sklearn.preprocessing import LabelEncoder
import re
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import *
from sklearn.model_selection import train_test_split
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
```

Spam DataSet From Github Repo

```
url = 'https://raw.githubusercontent.com/aayushsh2003/ML/main/Experiment/Experiment%206/spam.csv'
df = pd.read_csv(url, encoding='latin-1')
```

```
df.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

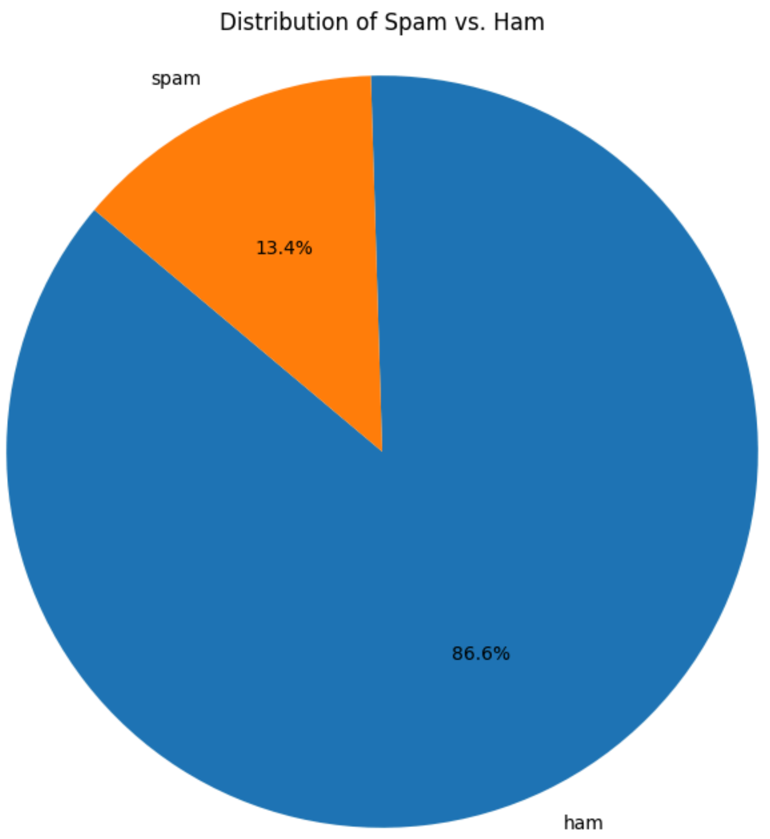
Next steps: Generate code with df | View recommended plots | New interactive sheet

```
pip install emoji --upgrade
```

```
Collecting emoji
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
```

```
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 6.9 MB/s eta 0:00:00
Installing collected packages: emoji
Successfully installed emoji-2.14.1
```

```
# Calculate the count of each label
category_counts = df['Category'].value_counts()

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Spam vs. Ham')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```




```
# Iterate through unique categories
for category in df['Category'].unique():
    # Filter the DataFrame for the current category
    filtered_df = df[df['Category'] == category]

    # Concatenate all text data for the current category
    text = ' '.join(filtered_df['Message'])

    # Generate word cloud
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

    # Plot the word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for Category: {category}')
    plt.axis('off')
    plt.show()
```

Word Cloud for Category: ham

Word Cloud for Category: spam


```
# Encode Category column
le = LabelEncoder()
df['Category']=le.fit_transform(df['Category'])
df.head()
```

	Category	Message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

Next steps: Generate code with df View recommended plots New interactive sheet

```
# Convert 'Text' column to lowercase
df['Message'] = df['Message'].str.lower()
df.head()
```

	Category	Message
0	0	go until jurong point, crazy.. available only ...
1	0	ok lar... joking wif u oni...
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor... u c already then say...
4	0	nah i don't think he goes to usf, he lives aro...

Next steps: Generate code with df View recommended plots New interactive sheet

```
# Remove extra white spaces from the 'Text' column
df['Message'] = df['Message'].str.strip()
df.head()
```

	Category	Message
0	0	go until jurong point, crazy.. available only ...
1	0	ok lar... joking wif u oni...
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor... u c already then say...
4	0	nah i don't think he goes to usf, he lives aro...

Next steps: Generate code with df View recommended plots New interactive sheet

```
# Function to remove HTML tags from text
def remove_html_tags(text):
    soup = BeautifulSoup(text, 'html.parser')
    return soup.get_text()

# Remove HTML tags from 'Text' column
df['Message'] = df['Message'].apply(remove_html_tags)
```

```
# Define a function to remove URLs using regular expressions
def remove_urls(text):
    return re.sub(r'http\S+|www\S+', '', text)

# Apply the function to the 'Text' column
df['Message'] = df['Message'].apply(remove_urls)
```

```
string.punctuation

# Define the punctuation characters to remove
punctuation = string.punctuation
# Function to remove punctuation from text
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', punctuation))

# Apply remove_punctuation function to 'Text' column
df['Message'] = df['Message'].apply(remove_punctuation)
```

```
def remove_special_characters(text):
    # Define the pattern to match special characters
    pattern = r'[^a-zA-Z0-9\s]'  # Matches any character that is not alphanumeric or whitespace

    # Replace special characters with an empty string
    clean_text = re.sub(pattern, '', text)

    return clean_text

# Apply the function to the 'Message' column
df['Message'] = df['Message'].apply(remove_special_characters)
```

```
# Define a function to remove non-alphanumeric characters
def remove_non_alphanumeric(text):
    return re.sub(r'[^a-zA-Z0-9\s]', '', text)

# Apply the function to the "Message" column
df['Message'] = df['Message'].apply(remove_non_alphanumeric)
```

```
# Define a dictionary of chat word mappings
chat_words = {
    "AFAIK": "As Far As I Know",
    "AFK": "Away From Keyboard",
    "ASAP": "As Soon As Possible",
    "ATK": "At The Keyboard",
    "ATM": "At The Moment",
```

```
    "A3": "Anytime, Anywhere, Anyplace",
    "BAK": "Back At Keyboard",
    "BBL": "Be Back Later",
    "BBS": "Be Back Soon",
    "BFN": "Bye For Now",
    "B4N": "Bye For Now",
    "BRB": "Be Right Back",
    "BRT": "Be Right There",
    "BTW": "By The Way",
    "B4": "Before",
    "B4N": "Bye For Now",
    "CU": "See You",
    "CUL8R": "See You Later",
    "CYA": "See You",
    "FAQ": "Frequently Asked Questions",
    "FC": "Fingers Crossed",
    "FWIW": "For What It's Worth",
    "FYI": "For Your Information",
    "GAL": "Get A Life",
    "GG": "Good Game",
    "GN": "Good Night",
    "GMTA": "Great Minds Think Alike",
    "GR8": "Great!",
    "G9": "Genius",
    "IC": "I See",
    "ICQ": "I Seek you (also a chat program)",
    "ILU": "ILU: I Love You",
    "IMHO": "In My Honest/Humble Opinion",
    "IMO": "In My Opinion",
    "IOW": "In Other Words",
    "IRL": "In Real Life",
    "KISS": "Keep It Simple, Stupid",
    "LDR": "Long Distance Relationship",
    "LMAO": "Laugh My A.. Off",
    "LOL": "Laughing Out Loud",
    "LTNS": "Long Time No See",
    "L8R": "Later",
    "MTE": "My Thoughts Exactly",
    "M8": "Mate",
    "NRN": "No Reply Necessary",
    "OIC": "Oh I See",
    "PITA": "Pain In The A..",
    "PRT": "Party",
    "PRW": "Parents Are Watching",
    "QPSA?": "Que Pasa?",
    "ROFL": "Rolling On The Floor Laughing",
    "ROFLOL": "Rolling On The Floor Laughing Out Loud",
    "ROTFLMAO": "Rolling On The Floor Laughing My A.. Off",
    "SK8": "Skate",
    "STATS": "Your sex and age",
    "ASL": "Age, Sex, Location",
    "THX": "Thank You",
    "TTFN": "Ta-Ta For Now!",
    "TTYL": "Talk To You Later",
    "U": "You",
    "U2": "You Too",
    "U4E": "Yours For Ever",
    "WB": "Welcome Back",
    "WTF": "What The F...",
    "WTG": "Way To Go!",
    "WUF": "Where Are You From?",
    "W8": "Wait...",
    "7K": "Sick:-D Laugher",
    "TFW": "That feeling when",
    "MFW": "My face when",
    "MRW": "My reaction when",
    "IFYP": "I feel your pain",
    "TNTL": "Trying not to laugh",
    "JK": "Just kidding",
    "IDC": "I don't care",
    "ILY": "I love you",
    "IMU": "I miss you",
    "ADIH": "Another day in hell",
    "ZZZ": "Sleeping, bored, tired",
```

```
    "WYWH": "Wish you were here",
    "TIME": "Tears in my eyes",
    "BAE": "Before anyone else",
    "FIMH": "Forever in my heart",
    "BSAAW": "Big smile and a wink",
    "BWL": "Bursting with laughter",
    "BFF": "Best friends forever",
    "CSL": "Can't stop laughing"
}
```

```
# Function to replace chat words with their full forms
def replace_chat_words(text):
    words = text.split()
    for i, word in enumerate(words):
        if word.lower() in chat_words:
            words[i] = chat_words[word.lower()]
    return ' '.join(words)

# Apply replace_chat_words function to 'Text' column
df['Message'] = df['Message'].apply(replace_chat_words)
```

```
# Download NLTK stopwords corpus
nltk.download('stopwords')

# Get English stopwords from NLTK
stop_words = set(stopwords.words('english'))

# Function to remove stop words from text
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply remove_stopwords function to 'Text' column
df['Message'] = df['Message'].apply(remove_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
!pip install emoji --upgrade
import emoji

# Function to remove emojis from text
def remove_emojis(text):
    return emoji.demojize(text)

# Apply remove_emojis function to 'Text' column
df['Message'] = df['Message'].apply(remove_emojis)
```

```
Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-packages (2.14.1)
```

```
# Initialize the Porter Stemmer
porter_stemmer = PorterStemmer()

# Apply stemming
df['Message_stemmed'] = df['Message'].apply(lambda x: ' '.join([porter_stemmer.stem(word) for word in x.split()]))
```

```
# Intlize CountVectorizer
cv = CountVectorizer()

# Fitting CountVectorizer on X
X = cv.fit_transform(df['Message_stemmed']).toarray()
y = df['Category']
```

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

```
# Gaussian Naive Bayes
gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)
gnb_pred = gnb_model.predict(X_test)
```

```
gnb_accuracy = accuracy_score(y_test, gnb_pred)
gnb_precision = precision_score(y_test, gnb_pred, average='weighted')
gnb_recall = recall_score(y_test, gnb_pred, average='weighted')
gnb_conf_matrix = confusion_matrix(y_test, gnb_pred)
```

```
# Multinomial Naive Bayes with tuned parameters
mnb_model = MultinomialNB(alpha=0.1)
mnb_model.fit(X_train, y_train)
mnb_pred = mnb_model.predict(X_test)
```

```
mnb_accuracy = accuracy_score(y_test, mnb_pred)
mnb_precision = precision_score(y_test, mnb_pred, average='weighted')
mnb_recall = recall_score(y_test, mnb_pred, average='weighted')
mnb_conf_matrix = confusion_matrix(y_test, mnb_pred)
```

```
print("Multinomial Naive Bayes:")
print(f"The accuracy score of MultinomialNB is {mnb_accuracy}, The Precision Score is {mnb_precision},The Recall Score
print(f"The Confusion matrix is \n{mnb_conf_matrix}")
print("\n")

print("Gaussian Naive Bayes:")
print(f"The accuracy score of GaussianNB is {gnb_accuracy}, The Precision Score is {gnb_precision},The Recall Score is
print(f"The Confusion matrix is \n{gnb_conf_matrix}")
print("\n")
```

```
Multinomial Naive Bayes:
The accuracy score of MultinomialNB is 0.9721973094170404, The Precision Score is 0.9728833837324558,The Recall Score is 0.97219730
The Confusion matrix is
[[947  19]
 [ 12 137]]


Gaussian Naive Bayes:
The accuracy score of GaussianNB is 0.863677130044843, The Precision Score is 0.9180444492929181,The Recall Score is 0.86367713004
The Confusion matrix is
[[828 138]
 [ 14 135]]
```

Experiment 7

Write a program to construct aBayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

```
!pip install pgmpy
```

Show hidden output

+ Code + Text

```
!pip install --upgrade pgmpy
```

Show hidden output

```
!pip install --upgrade pgmpy  # Ensure pgmpy is up-to-date
from pgmpy.models import DiscreteBayesianNetwork # Import DiscreteBayesianNetwork instead of BayesianNetwork or Bayesian

cancer_model = DiscreteBayesianNetwork([('Pollution', 'Cancer'),
                              ('Smoker', 'Cancer'),
                              ('Cancer', 'Xray'),
                              ('Cancer', 'Dyspnoea')])
print(cancer_model)
```

Show hidden output

```
print(cancer_model)
```

```
DiscreteBayesianNetwork with 5 nodes and 4 edges
```

```
cancer_model.nodes()   #to print nodes
```

```
NodeView(('Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea'))
```

```
cancer_model.edges() # to print edges
```

```
OutEdgeView([('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')])
```

Conditional Probability Distribution

```
cancer_model.get_cpds()   # to show conditional probability Distribution
```

```
[]
```

Creation of Conditional Probability Table

```
from pgmpy.factors.discrete import TabularCPD
```

```
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
                      values=[[0.9],[0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
                      values=[[0.3],[0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
                      values=[[0.03, 0.05, 0.001, 0.02],
                      [0.97, 0.95, 0.999, 0.98]],
                      evidence=['Smoker', 'Pollution'],
                      evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
                      values=[[0.9, 0.2],[0.1, 0.8]],
                      evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
                      values=[[0.65, 0.3],[0.35, 0.7]],
                      evidence=['Cancer'], evidence_card=[2])
```

Double-click (or enter) to edit

```
# associating the parameters with the model structure
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)

#checking if the cpds are valid for the model
```

```
cancer_model.get_cpds()
cancer_model.check_model()
```

```
True
```

```
#If you want to stick with DiscreteBayesianNetwork and identify active trails.
#This approach identifies all nodes on the active trail, including the start and end nodes.
active_trail = cancer_model.active_trail_nodes('Pollution', observed=['Cancer'])
# Check if 'Smoker' is in the active trail
is_active = 'Smoker' in active_trail['Pollution']
print(f"Is there an active trail between Pollution and Smoker given Cancer? {is_active}")
```

```
Is there an active trail between Pollution and Smoker given Cancer? True
```

```
cancer_model.local_independencies('Xray')     #Xray and Dyspnoea are Independent
```

```
(Xray ⊥ Dyspnoea, Smoker, Pollution | Cancer)
```

```
cancer_model.local_independencies('Pollution')
```

```
(Pollution ⊥ Smoker)
```

```
cancer_model.local_independencies('Smoker')
```

```
(Smoker ⊥ Pollution)
```

```
cancer_model.local_independencies('Dyspnoea')
```

```
(Dyspnoea ⊥ Smoker, Pollution, Xray | Cancer)
```

```
cancer_model.local_independencies('Cancer')
```

```
cancer_model.local_independencies('Dyspnoea')
```

```
(Dyspnoea ⊥ Smoker, Pollution, Xray | Cancer)
```

```
cancer_model.local_independencies('Pollution')
```

```
(Pollution ⊥ Smoker)
```

```
cancer_model.get_independencies()
```

```
(Pollution ⊥ Dyspnoea | Cancer)
(Smoker ⊥ Dyspnoea | Cancer)
(Pollution ⊥ Smoker)
(Dyspnoea ⊥ Xray | Cancer)
(Smoker ⊥ Xray | Cancer)
(Pollution ⊥ Xray | Cancer)
```

```
cancer_model.get_cpds()
print(cancer_model.get_cpds('Pollution'))
```

```
+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+
```

```
cancer_model.get_cpds()
print(cancer_model.get_cpds('Cancer'))
```

```
+-----------+--------------+--------------+--------------+--------------+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |
+-----------+--------------+--------------+--------------+--------------+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |
+-----------+--------------+--------------+--------------+--------------+
```

```
cancer_model.get_cpds()
#conditional probabilities
```

```
[<TabularCPD representing P(Pollution:2) at 0x797c11002310>,
 <TabularCPD representing P(Smoker:2) at 0x797c102a7190>,
 <TabularCPD representing P(Cancer:2 | Smoker:2, Pollution:2) at 0x797c110197d0>,
 <TabularCPD representing P(Xray:2 | Cancer:2) at 0x797c11018110>,
 <TabularCPD representing P(Dyspnoea:2 | Cancer:2) at 0x797c1101bf90>]
```

Inferencing with Bayesian Network with Variable Elimination

```
from pgmpy.inference import VariableElimination
cancer_infer = VariableElimination(cancer_model)

q=cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q)
```

```
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+
```

```
r=cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1, 'Pollution':1})
print(r)
```

```
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

```
s=cancer_infer.query(variables=['Cancer'], evidence={'Pollution':1})
print(r)
```

```
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

```
r=cancer_infer.query(variables=['Smoker'], evidence={'Cancer': 1})
print(r)
```

```
+-----------+---------------+
| Smoker    |   phi(Smoker) |
+===========+===============+
| Smoker(0) |        0.2938 |
+-----------+---------------+
| Smoker(1) |        0.7062 |
+-----------+---------------+
```

Start coding or generate with AI.

Experiment 8

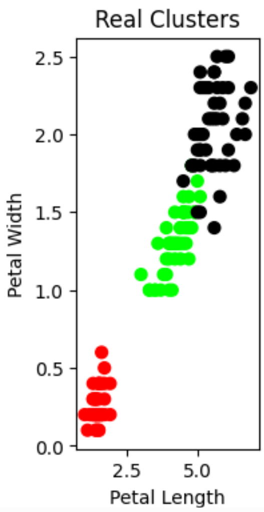
Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
import pandas as pd
import numpy as np
```

```
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

```
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(6,4))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```



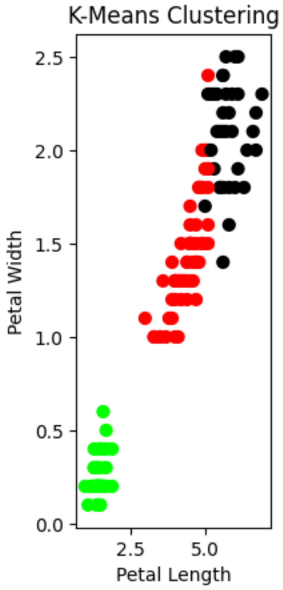

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

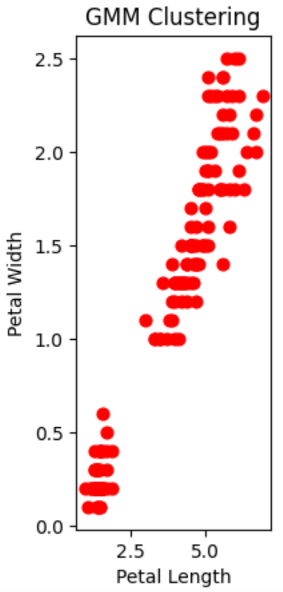
```
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```




```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
gmm = GaussianMixture(n_components=40)
gmm.fit(xs)
plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```




```
print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.
```

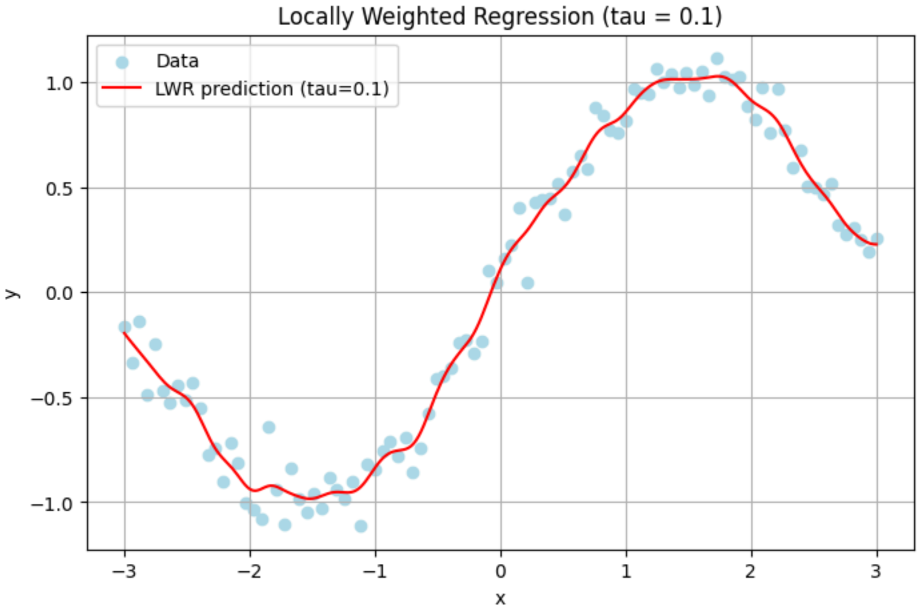
Experiment 9

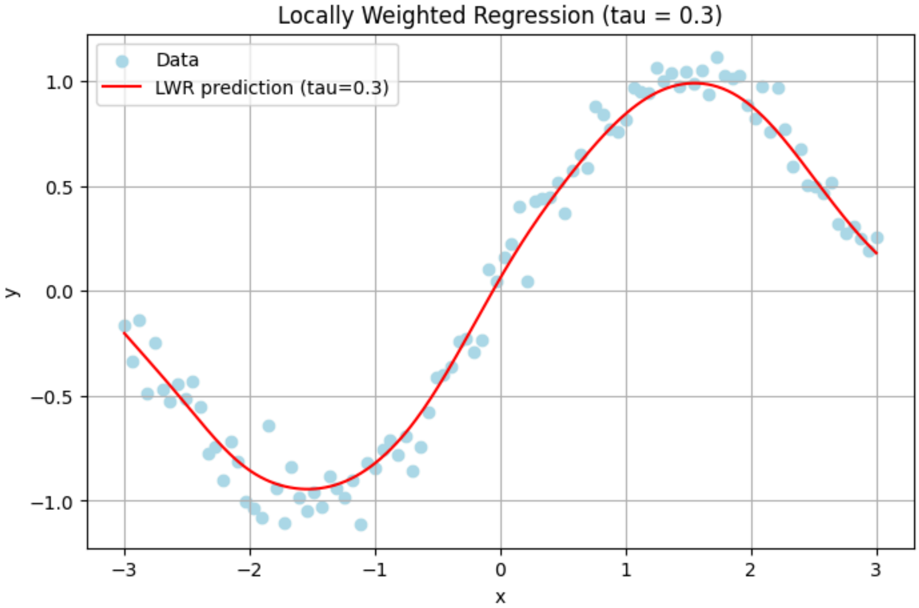
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

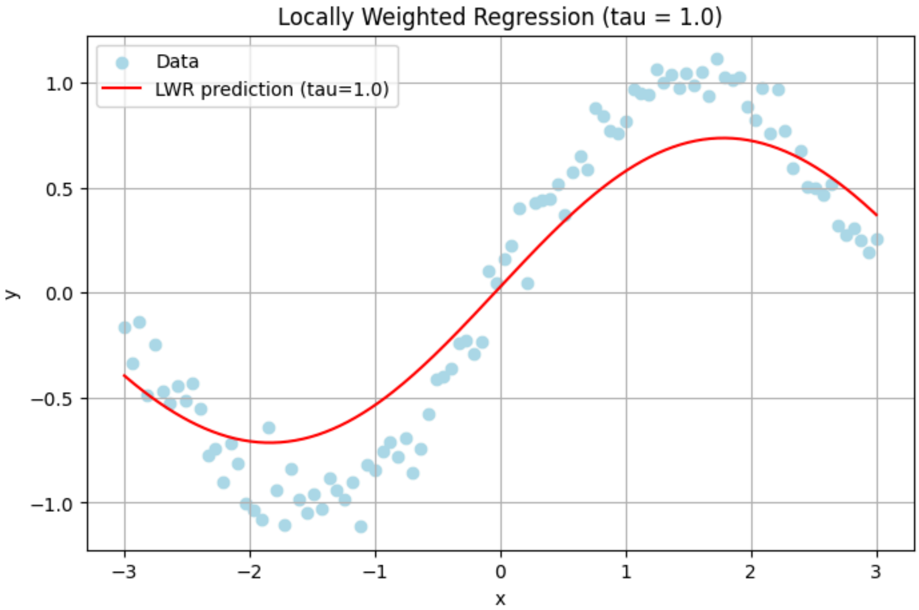
LWR is a non-parametric regression algorithm that fits a local linear model around each query point using a weighted least squares method, where nearby points have more influence (higher weight) than far ones.

τ (tau): Bandwidth parameter controlling how "local" the regression is.

```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic dataset
def generate_data():
    X = np.linspace(-3, 3, 100)
    y = np.sin(X) + 0.1 * np.random.randn(100)
    return X.reshape(-1, 1), y

# Add intercept term
def add_bias(X):
    return np.hstack([np.ones((X.shape[0], 1)), X])

# Weight matrix for a given query point x_query
def get_weights(X, x_query, tau):
    m = X.shape[0]
    weights = np.exp(-np.square(X - x_query).flatten() / (2 * tau**2))
    return np.diag(weights)

# Locally Weighted Regression
def locally_weighted_regression(X, y, tau, X_query):
    X_bias = add_bias(X)
    y_pred = []

    for x in X_query:
        W = get_weights(X, x, tau)
        theta = np.linalg.pinv(X_bias.T @ W @ X_bias) @ X_bias.T @ W @ y
        x_bias = np.array([1, x[0]])
        y_hat = x_bias @ theta
        y_pred.append(y_hat)

    return np.array(y_pred)

# Main function
def main():
    X, y = generate_data()
    X_query = np.linspace(-3, 3, 300).reshape(-1, 1)

    for tau in [0.1, 0.3, 1.0]:
        y_pred = locally_weighted_regression(X, y, tau, X_query)

        plt.figure(figsize=(8, 5))
        plt.scatter(X, y, label="Data", color="lightblue")
        plt.plot(X_query, y_pred, label=f"LWR prediction (tau={tau})", color="red")
        plt.title(f"Locally Weighted Regression (tau = {tau})")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.legend()
        plt.grid(True)
        plt.show()

main()
```








```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
```

```
import pandas as pd
import numpy as np
```

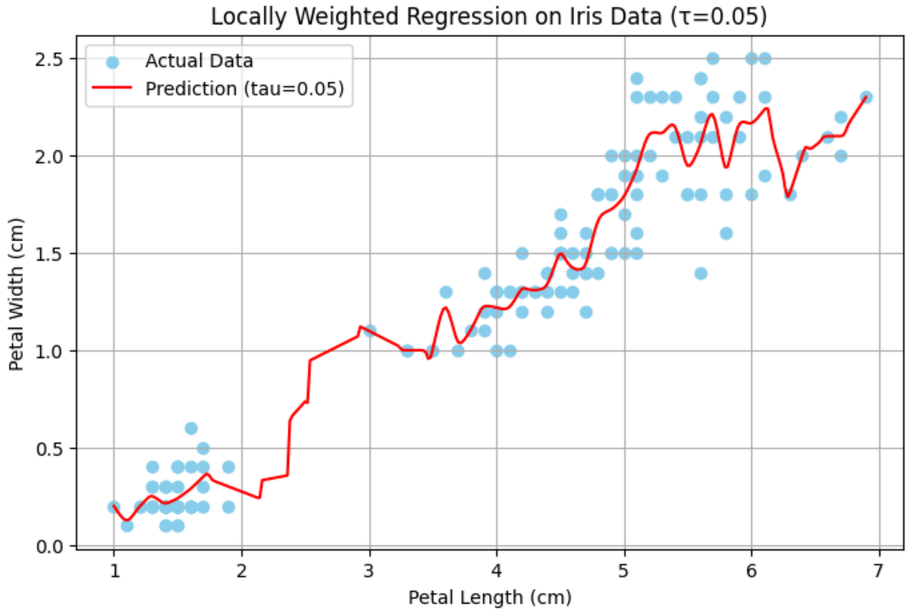
```
Start coding or generate with AI.
```

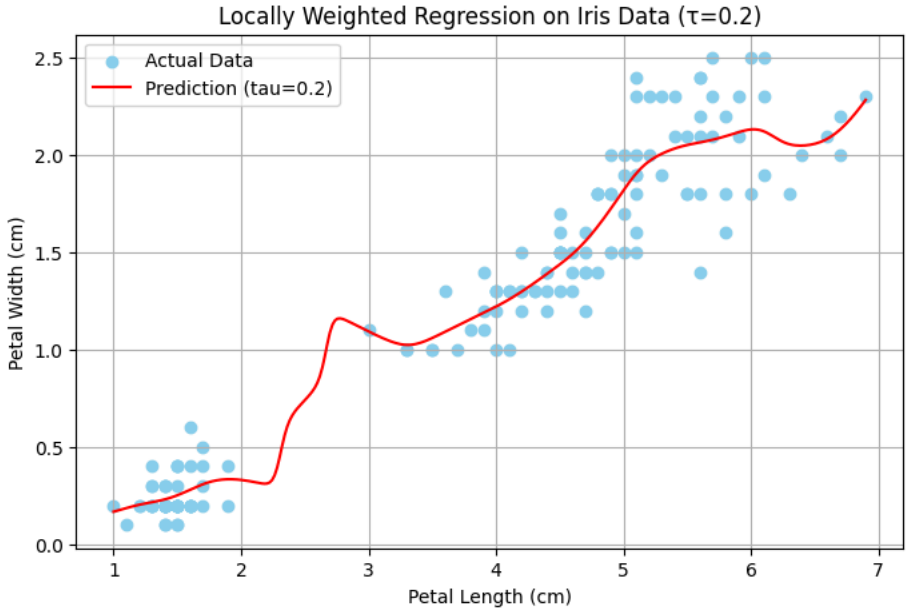
```
Start coding or generate with AI.
```

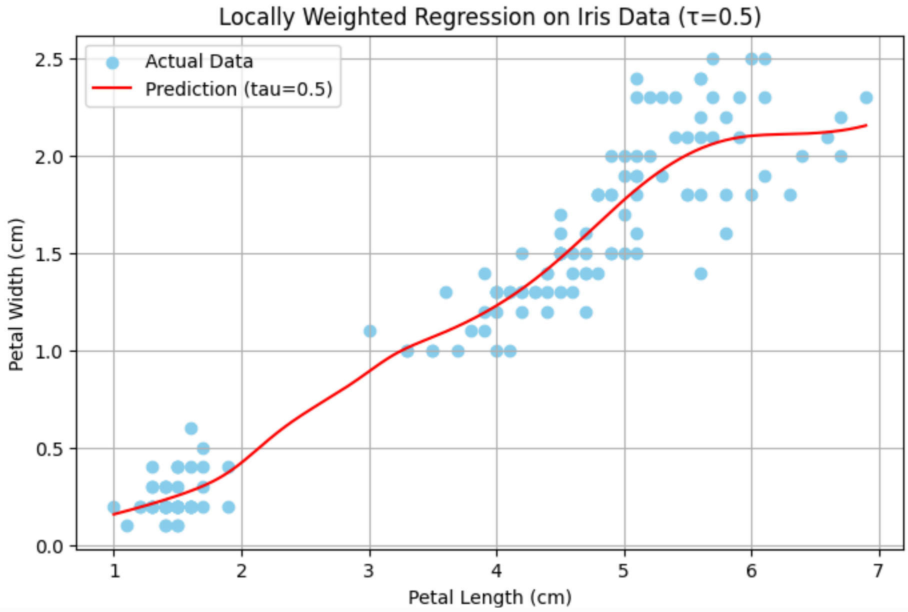
```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

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

# Add intercept to X
def add_bias(X):
    return np.hstack([np.ones((X.shape[0], 1)), X])

# Gaussian weights
def get_weights(X, x_query, tau):
    weights = np.exp(-np.square(X - x_query).flatten() / (2 * tau**2))
    return np.diag(weights)

# Locally Weighted Regression
def locally_weighted_regression(X, y, tau, X_query):
    X_bias = add_bias(X)
    y_pred = []

    for x in X_query:
        W = get_weights(X, x, tau)
        theta = np.linalg.pinv(X_bias.T @ W @ X_bias) @ X_bias.T @ W @ y
        x_bias = np.array([1, x[0]])
        y_hat = x_bias @ theta
        y_pred.append(y_hat)

    return np.array(y_pred)

# Load Iris dataset
def load_iris_data():
    iris = datasets.load_iris()
    X = iris.data[:, 2].reshape(-1, 1)  # Petal length
    y = iris.data[:, 3]                 # Petal width
    return X, y

# Main
def main():
    X, y = load_iris_data()
    X_query = np.linspace(X.min(), X.max(), 300).reshape(-1, 1)

    for tau in [0.05, 0.2, 0.5]:
        y_pred = locally_weighted_regression(X, y, tau, X_query)

        plt.figure(figsize=(8, 5))
        plt.scatter(X, y, label="Actual Data", color="skyblue")
        plt.plot(X_query, y_pred, label=f"Prediction (tau={tau})", color="red")
        plt.title(f"Locally Weighted Regression on Iris Data (τ={tau})")
        plt.xlabel("Petal Length (cm)")
        plt.ylabel("Petal Width (cm)")
        plt.legend()
        plt.grid(True)
        plt.show()

main()
```


Locally Weighted Regression on Iris Data (τ=0.05)
Actual Data
Prediction (tau=0.05)
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


Locally Weighted Regression on Iris Data (τ=0.2)
Actual Data
Prediction (tau=0.2)
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


Locally Weighted Regression on Iris Data (τ=0.5)
Actual Data
Prediction (tau=0.5)
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Experiment 10

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

KNN Algorithm implementation on Iris dataset

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

```
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels
```

```
# Print the details of the iris dataset
print("Iris data keys:", iris.keys())
print("\nFeature names:", iris.feature_names)
print("\nTarget names:", iris.target_names)

print("\nFirst 5 data points:\n", iris.data[:5])
print("\nTarget values:\n", iris.target)
```

```
Iris data keys: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Target names: ['setosa' 'versicolor' 'virginica']

First 5 data points:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]

Target values:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

```
# Function to find the best k using cross-validation
def find_best_k(X_train, y_train, k_range):
    mean_accuracies = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X_train, y_train, cv=5)  # 5-fold cross-validation
        mean_accuracies.append(scores.mean())
    return mean_accuracies

# Determine the best k
k_values = range(1, 21)
mean_accuracies = find_best_k(X_train, y_train, k_values)
best_k = k_values[np.argmax(mean_accuracies)]
best_accuracy = np.max(mean_accuracies)
```

```
# Print the best k and its accuracy
print(f"The best value of k is {best_k} with cross-validated accuracy {best_accuracy:.2f}")

# Train the KNN model with the best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)
```

```
The best value of k is 3 with cross-validated accuracy 0.95
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

The best value of k is 3 with cross-validated accuracy 0.95

```
# Make predictions on the test data
y_pred = knn_best.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of KNN with k={best_k}: {accuracy:.2f}")
```

```

Accuracy of KNN with k=3: 1.00
```

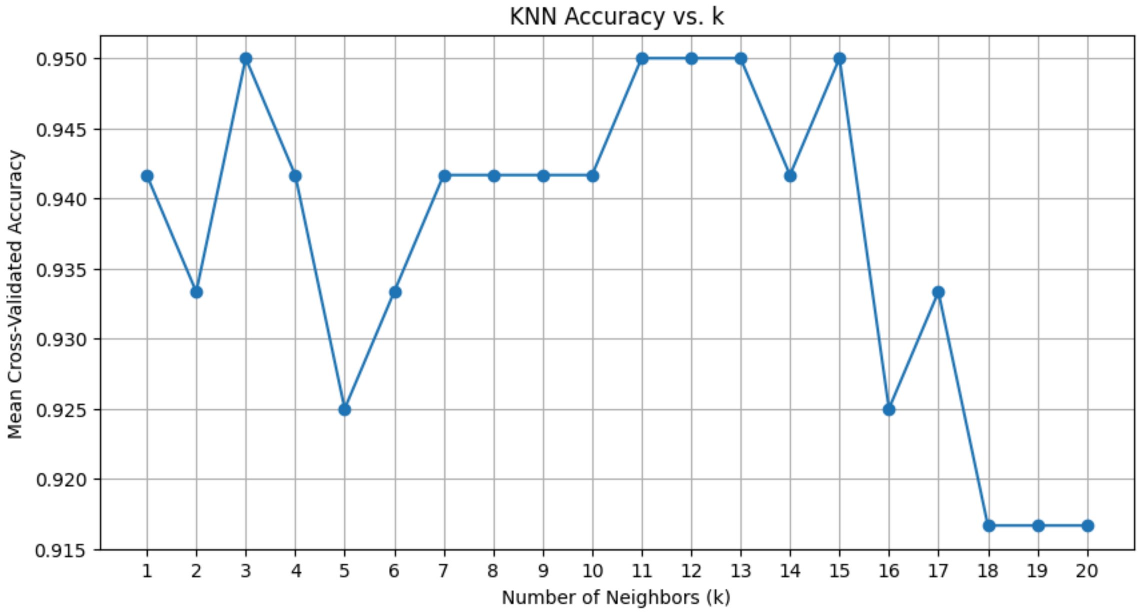
```
# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Visualize the accuracy for different values of k
plt.figure(figsize=(10, 5))
plt.plot(k_values, mean_accuracies, marker='o')
plt.title('KNN Accuracy vs. k')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Cross-Validated Accuracy')
plt.xticks(k_values)
plt.grid()
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.

+ Code + Text

Start coding or generate with AI.

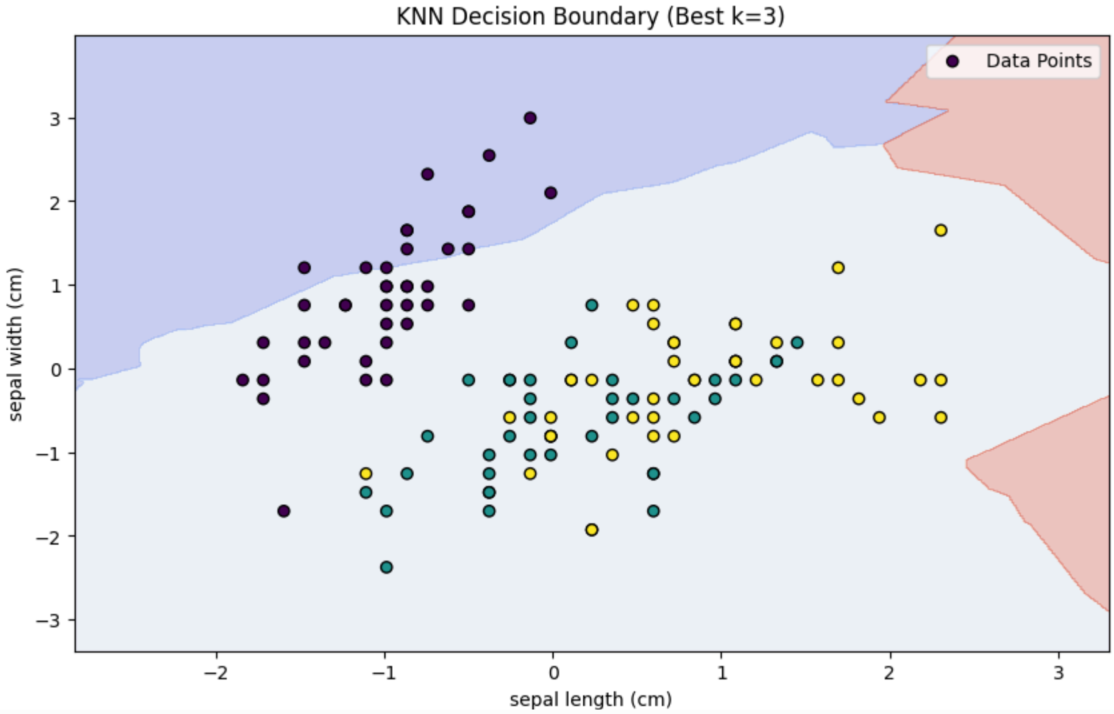
Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
# Optional: Visualize the decision boundary (for the first two features)
def plot_decision_boundary(X, y, model):
    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, 0.01),
                         np.arange(y_min, y_max, 0.01))

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

    # Plot the decision boundary
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', marker='o', label='Data Points')
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
    plt.title(f"KNN Decision Boundary (Best k={best_k})")
    plt.legend()
    plt.show()

# Plot the decision boundary using the first two features
plot_decision_boundary(X_train[:, :2], y_train, knn_best)
```


KNN Decision Boundary (Best k=3)
Data Points
sepal width (cm)
sepal length (cm)
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3