SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL503 **DATE: 8/12/2023**

COURSE NAME: Data Warehousing and Mining **CLASS: TY-IT-I2-2**

LAB EXPERIMENT NO. 10 (MINI PROJ)

AIM / OBJECTIVE:

The aim is to perform comprehensive data analysis and visualization on a given dataset, employing various statistical techniques, data preprocessing, visualization, clustering, and regression model algorithms,etc.

DESCRIPTION OF EXPERIMENT:

- Conduct an exploratory analysis to understand the dataset's characteristics, distributions, relationships, and potential insights.
- Implement data preprocessing techniques to handle missing values and irrelevant columns.
- Apply statistical methods to analyze central tendency (mean, median, mode), data distribution (skewness, kurtosis), and other descriptive statistics.
- Visualize data using scatter plots, pair plots, pie plots, violin plots, heatmaps, and distribution plots to gain insights.
- Perform clustering analysis and visualize clusters on graphs.
- Utilize regression analysis for predictive modeling and graphing.
- Build Decision Tree with user-defined depth, visualize using Matplotlib.

INPUT DATA / DATASET:

- Car Data-Prices and features of various cars
- Iris Data-Sepal and Petal width and length of various flowers to classify into 3 categories
- Male Data- Various body features and measurements to gain insights on whether a person has Fatty Liver Disease
- Mall Customer- Information about Various Customers visiting the mall
- Diabetes Data- Based on various attributes like Haemoglobin, Blood Sugar we predict if the person has diabetes

PROCEDURE / ALGORITHM:

→Data Preprocessing:

Drop columns, replace NaN values, and bin numeric columns interactively.

→Data Exploration:

Explore rows, columns, and visualize correlation heatmaps.

→Data Visualization:

Generate bar graphs, scatter plots, histograms, and more using Seaborn and Matplotlib.

→K Means Clustering:

Apply K Means clustering on numeric columns and visualize clusters.

→Logistic Regression:

Perform logistic regression, display equation, accuracy, confusion matrix, and scatter plot.

→Naive Bayes:

Apply Naive Bayes classification, display accuracy, confusion matrix, and metrics.

→Designing Decision Tree:

Build Decision Tree with user-defined depth, visualize using Matplotlib.

→General App Structure:

Streamlit-based organization for seamless user interaction.

Upload CSV datasets for diverse functionalities.

TECHNOLOGY STACK USED:

Streamlit

Python

Seaborn

Matplotlib

sklearn

CODE:

```
import streamlit as st

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph_objects as go

from sklearn.cluster import KMeans

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

# Function for Data Preprocessing
def data_preprocessing(data):
    st.subheader("Data Preprocessing")

    #Drop null values
    #data = data.dropna()

    # Drop a column
    st.write("Columns in the dataset:", data.columns)
    column_to_drop = st.text_input("Enter the column to drop (if any):")
    if column_to_drop:
        data = data.drop(column_to_drop, axis=1)
        st.write(f"Column '{column_to_drop}' dropped.")

    # Replace NaN values
    st.write("Replace NaN values:")
    columns_to_replace = st.multiselect("Select columns to replace NaN values:",
data.columns)
    for column in columns_to_replace:
        if data[column].isna().sum() > 0:
            replacement_value = st.text_input(f"Enter replacement value for
'{column}':")
        else:
            st.warning("No missing values in the selected column.")
        if replacement_value:
                data[column] = data[column].fillna(replacement_value)
                st.write(f"NaN values in '{column}' replaced with
'{replacement_value}'.")
```

```
    # Binning method
    st.write("Binning Method:")
    column_for_binning = st.selectbox("Select column for binning:", data.columns)
    num_of_bins = st.slider("Number of bins:", min_value=2, max_value=20, value=5)

    if st.button("Apply Binning"):
        data[column_for_binning+'_binned'] = pd.cut(data[column_for_binning],
bins=num_of_bins, labels=False)
        st.write(f"Binning applied on '{column_for_binning}'.")

    st.write("Processed Data:")
    st.write(data)

    return data

# Function for Data Exploration
def data_exploration(data):
    st.subheader("Data Exploration")

    # Display overview of rows and columns
    st.write("Overview of Rows and Columns:")
    st.write(data.head())

    # Numeric column statistics
    numeric_columns = data.select_dtypes(include=[np.number]).columns
    selected_column = st.selectbox("Select a numeric column for statistics:",
numeric_columns)

    st.write(f"Statistics for '{selected_column}':")
    st.write(f"Mean: {data[selected_column].mean()}")
    st.write(f"Median: {data[selected_column].median()}")
    st.write(f"Mode: {data[selected_column].mode().values[0]}")
    st.write(f"Highest Value: {data[selected_column].max()}")
    st.write(f"Lowest Value: {data[selected_column].min()}")
    st.write("Range: {:.2f}".format(data[selected_column].max() -
data[selected_column].min()))
```

```
    st.write("Quartiles: Q1 = {:.2f}, Q2 = {:.2f}, Q3 =
{:.2f}".format(*data[selected_column].quantile([0.25, 0.5, 0.75])))
    st.write(f"IQR: {data[selected_column].quantile(0.75) -
data[selected_column].quantile(0.25)}")

    # Allow user to select at least two numeric columns for correlation heatmap
    selected_columns = st.multiselect("Select numeric columns for correlation
heatmap:", data.select_dtypes(include='number').columns)

    if len(selected_columns) >= 2:
            # Display correlation heatmap for selected columns
            st.subheader("Correlation Heatmap")
            correlation_matrix = data[selected_columns].corr()
            fig, ax = plt.subplots()
            sns.heatmap(correlation_matrix, annot=True, cmap="viridis", center=0,
ax=ax)
            st.pyplot(fig)
    else:
            st.warning("Please select at least two numeric columns for the
correlation heatmap.")


# Function for Data Visualization
def data_visualization(data):
    st.subheader("Data Visualization")

    # Bar graph
    st.write("Bar Graph:")
    # Select a column for the bar graph
    bar_column = st.selectbox("Select a column for the bar graph:", data.columns)
    # Create a bar graph using Seaborn
    plt.figure(figsize=(10, 6))
    sns.barplot(x=bar_column, y='count',
data=data[bar_column].value_counts().reset_index(), palette="viridis")
    plt.title(f"Bar Graph for {bar_column}")
    plt.xlabel(bar_column)
```

```
    plt.ylabel("Count")
    st.pyplot(plt)

    # Scatter plot
    st.write("Scatter Plot:")
    scatter_x = st.selectbox("Select x-axis column:", data.columns)
    scatter_y = st.selectbox("Select y-axis column:", data.columns)
    # Create a scatter plot using Seaborn
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=scatter_x, y=scatter_y, data=data)
    plt.title("Scatter Plot")
    plt.xlabel(scatter_x)
    plt.ylabel(scatter_y)
    st.pyplot(plt)

    # Histogram
    st.write("Histogram:")
    hist_column = st.selectbox("Select a column for the histogram:", data.columns)
    plt.figure(figsize=(10, 6))
    sns.histplot(data[hist_column], kde=True, color='skyblue')
    plt.title(f"Histogram for {hist_column}")
    plt.xlabel(hist_column)
    plt.ylabel("Frequency")
    st.pyplot(plt)

    # Pie chart
    st.write("Pie Chart:")
    pie_column = st.selectbox("Select a column for the pie chart:", data.columns)
    # Create a pie chart using Matplotlib
    plt.figure(figsize=(10, 6))
    data[pie_column].value_counts().plot.pie(autopct='%1.1f%%',
colors=sns.color_palette('pastel'), startangle=90)
    plt.title(f"Pie Chart for {pie_column}")
    plt.ylabel("")
    st.pyplot(plt)
```

```
    # Box plot
    st.write("Box Plot:")
    box_column = st.selectbox("Select a column for the box plot:", data.columns)
    # Create a box plot using Seaborn
    plt.figure(figsize=(10, 6))
    sns.boxplot(y=data[box_column])
    plt.title(f"Box Plot for {box_column}")
    plt.ylabel(box_column)
    st.pyplot(plt)


# Function for K Means Clustering
def k_means_clustering(data):
    st.subheader("K Means Clustering")

    # Select columns for clustering
    st.write("Select columns for clustering:")
    features = st.multiselect("Select numeric columns for clustering:",
data.select_dtypes(include=[np.number]).columns)

    if st.button("Apply Clustering"):
        if len(features) >= 2:
            # Fit KMeans model
            kmeans = KMeans(n_clusters=3)  # You can adjust the number of clusters
            data['Cluster'] = kmeans.fit_predict(data[features])

            # Plot scatter plot with clusters
            fig = px.scatter(data, x=features[0], y=features[1], color='Cluster',
title="K Means Clustering")
            st.plotly_chart(fig)

            st.write("Cluster Labels:")
            st.write(data[['Cluster']])
        else:
            st.warning("Please select at least two numeric columns for
clustering.")
```

```python
# Function for Logistic Regression
def logistic_regression(data):
    st.subheader("Logistic Regression")

    # Select columns for logistic regression
    st.write("Select columns for logistic regression:")
    X_columns = st.multiselect("Select numeric columns as features for logistic
regression:", data.select_dtypes(include=[np.number]).columns)
    y_column = st.selectbox("Select the target column for Logistic Regression:",
data.columns)

    if st.button("Apply Logistic Regression"):
        if X_columns and y_column:
            # Assuming two columns for logistic regression
            X = data[X_columns]
            y = data[y_column]

            # Split data into training and testing sets
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

            # Fit Logistic Regression model
            model = LogisticRegression()
            model.fit(X_train, y_train)

            # Make predictions
            y_pred = model.predict(X_test)

            # Display equation of the line
            equation = f"{y_column} = {model.intercept_[0]} + "
            for i, coef in enumerate(model.coef_[0]):
                equation += f"{coef:.4f}*{X_columns[i]} + "

            st.write("Equation of the Line:")
            st.write(equation[:-2])  # Remove the last '+'
```

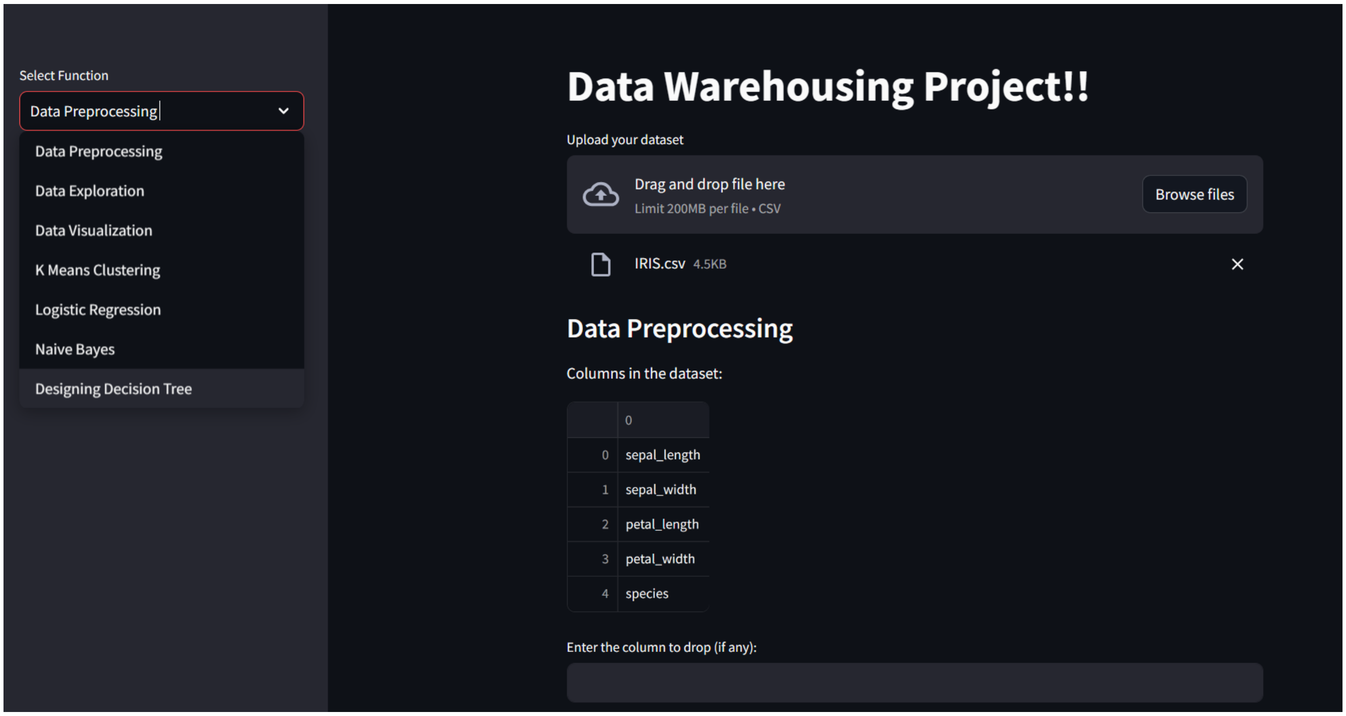
```
            # Display accuracy score and confusion matrix
            accuracy = accuracy_score(y_test, y_pred)
            confusion_mat = confusion_matrix(y_test, y_pred)

            st.write(f"Accuracy: {accuracy}")
            st.write("Confusion Matrix:")
            st.write(confusion_mat)

            # Plot graph
            if len(X_columns) >= 2:
                fig = px.scatter(data, x=X_columns[0], y=X_columns[1],
color=y_column, title="Logistic Regression")
                st.plotly_chart(fig)
        else:
            st.warning("Please select at least one feature column and a target
column for logistic regression.")
def naive_bayes(data):
    st.subheader("Naive Bayes")

    # Select columns for Naive Bayes
    st.write("Select columns for Naive Bayes:")
    X_columns = st.multiselect("Select numeric columns as features for Naive
Bayes:", data.select_dtypes(include=[np.number]).columns)
    y_column = st.selectbox("Select the target column for Naive Bayes:",
data.columns)

    if st.button("Apply Naive Bayes"):
        if X_columns and y_column:
            X = data[X_columns]
            y = data[y_column]

            # Split data into training and testing sets
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
            # Fit Naive Bayes model
            model = GaussianNB()
            model.fit(X_train, y_train)

            # Make predictions
            y_pred = model.predict(X_test)

            # Display accuracy score and confusion matrix
            accuracy = accuracy_score(y_test, y_pred)
            confusion_mat = confusion_matrix(y_test, y_pred)

            st.write(f"Accuracy: {accuracy}")
            st.write("Confusion Matrix:")
            st.write(confusion_mat)

        else:
            st.warning("Please select at least one feature column and a target
column for Naive Bayes.")

def decision_tree(data):
    st.subheader("Decision Tree")

    # Select columns for Decision Tree
    st.write("Select columns for Decision Tree:")
    X_columns = st.multiselect("Select numeric columns as features for Decision
Tree:", data.select_dtypes(include=[np.number]).columns)
    y_column = st.selectbox("Select the target column for Decision Tree:",
data.columns)

    # Set max_depth parameter
    max_depth = st.slider("Select the maximum depth of the Decision Tree:",
min_value=1, max_value=10, value=3)

    if st.button("Apply Decision Tree"):
        if X_columns and y_column:
            X = data[X_columns]
```

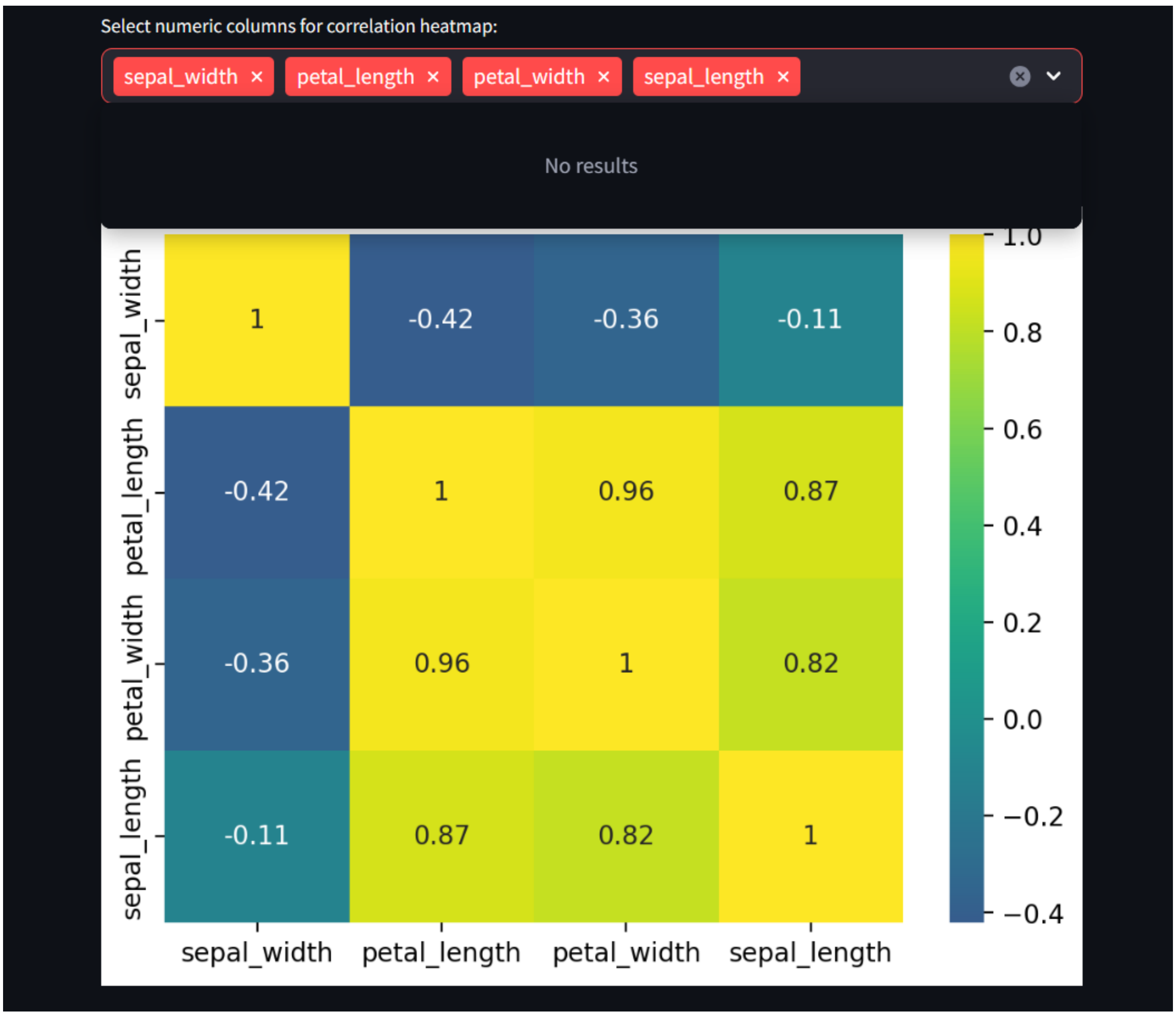
```
            y = data[y_column]

            # Split data into training and testing sets
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

            # Fit Decision Tree model
            model = DecisionTreeClassifier(max_depth=max_depth)
            model.fit(X_train, y_train)

            # Display Decision Tree structure
            st.write("Decision Tree Structure:")
            tree_structure = export_text(model, feature_names=X_columns)
            st.text(tree_structure)

            # Visualize Decision Tree using matplotlib
            plt.figure(figsize=(10, 7))
            plot_tree(model, filled=True, feature_names=X_columns,
class_names=list(map(str, model.classes_)), rounded=True)
            st.pyplot(plt)

        else:
            st.warning("Please select at least one feature column and a target
column for Decision Tree.")

# Streamlit App
def main():
    st.title("Data Warehousing Project!!")

    # Upload dataset
    uploaded_file = st.file_uploader("Upload your dataset", type=["csv"])

    if uploaded_file is not None:
        data = pd.read_csv(uploaded_file)  # Adjust based on the file type

        # Sidebar navigation
```

```
        option = st.sidebar.selectbox("Select Function", ["Data Preprocessing",
"Data Exploration", "Data Visualization", "K Means Clustering", "Logistic
Regression","Naive Bayes","Designing Decision Tree"])

        if option == "Data Preprocessing":
            data = data_preprocessing(data)

        elif option == "Data Exploration":
            data_exploration(data)

        elif option == "Data Visualization":
            data_visualization(data)

        elif option == "K Means Clustering":
            k_means_clustering(data)

        elif option == "Logistic Regression":
            logistic_regression(data)

        elif option == "Naive Bayes":
            naive_bayes(data)

        elif option == "Designing Decision Tree":
            decision_tree(data)

if __name__ == "__main__":
    main()
```

OUTPUT:



Select Function
Data Preprocessing
Data Preprocessing
Data Exploration
Data Visualization
K Means Clustering
Logistic Regression
Naive Bayes
Designing Decision Tree
Data Warehousing Project!!
Upload your dataset
Drag and drop file here
Limit 200MB per file • CSV
Browse files
IRIS.csv 4.5KB
Data Preprocessing
Columns in the dataset:
0
0 sepal_length
1 sepal_width
2 petal_length
3 petal_width
4 species
Enter the column to drop (if any):



Select numeric columns for correlation heatmap:
sepal_width
petal_length
petal_width
sepal_length
No results
1 -0.42 -0.36 -0.11
-0.42 1 0.96 0.87
-0.36 0.96 1 0.82
-0.11 0.87 0.82 1
sepal_width
petal_length
petal_width
sepal_length
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Data Exploration

Overview of Rows and Columns:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa

Select a numeric column for statistics:

sepal_length

Statistics for 'sepal_length':

Mean: 5.843333333333334

Median: 5.8

Mode: 5.0

Highest Value: 7.9

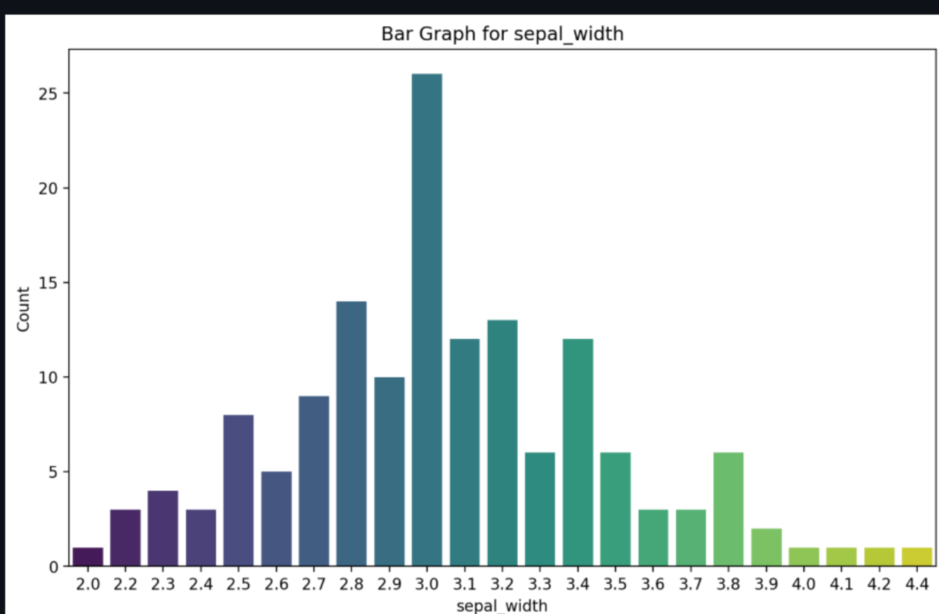
Lowest Value: 4.3

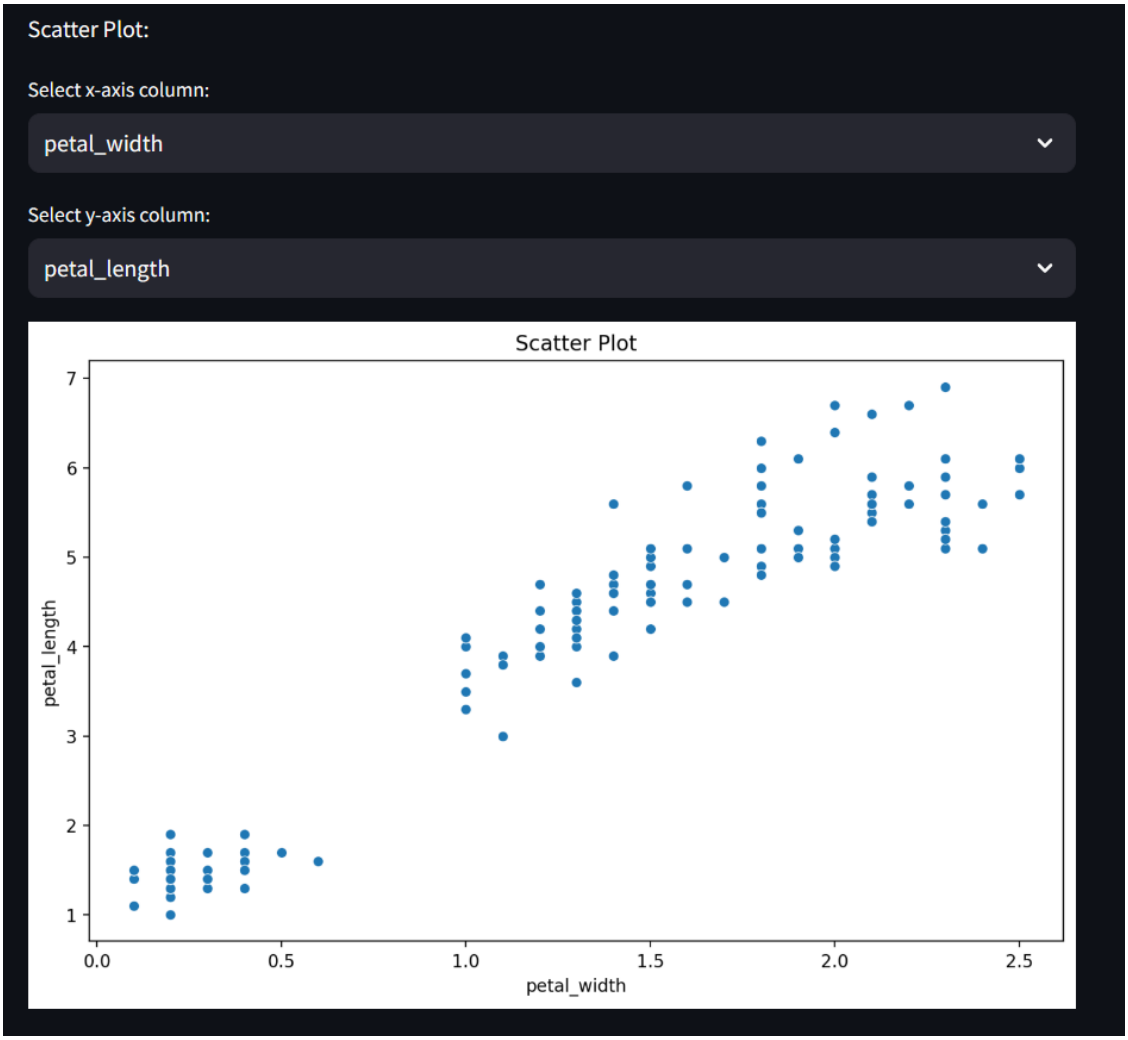
Data Visualization

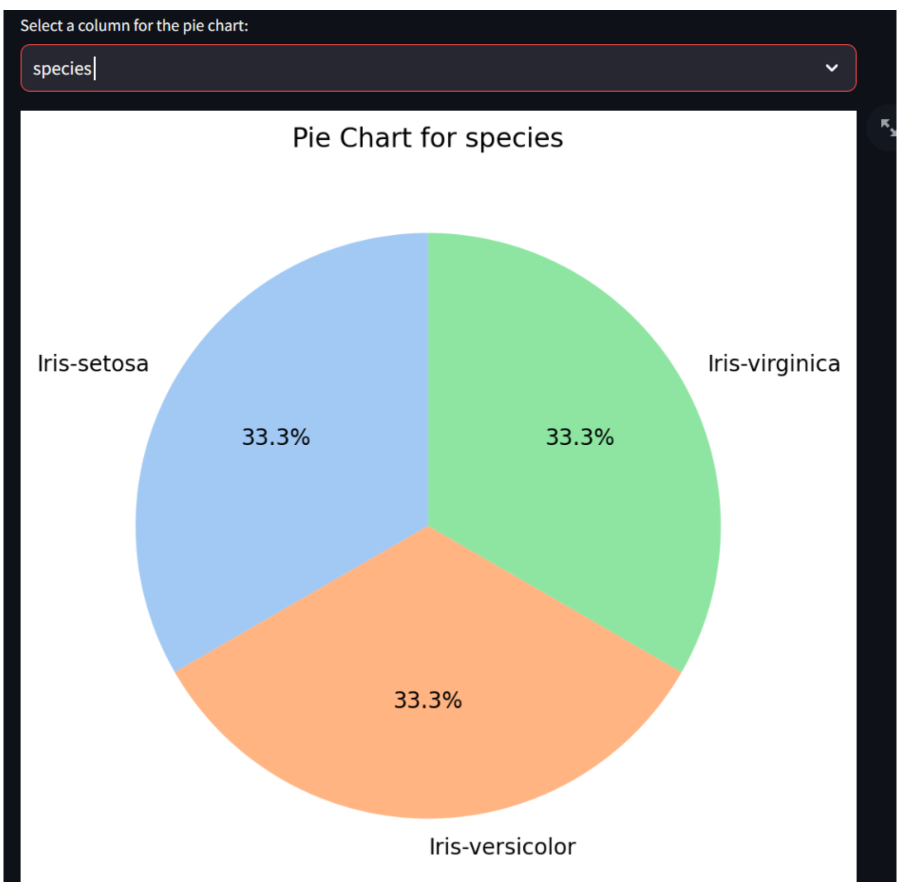
Bar Graph:

Select a column for the bar graph:

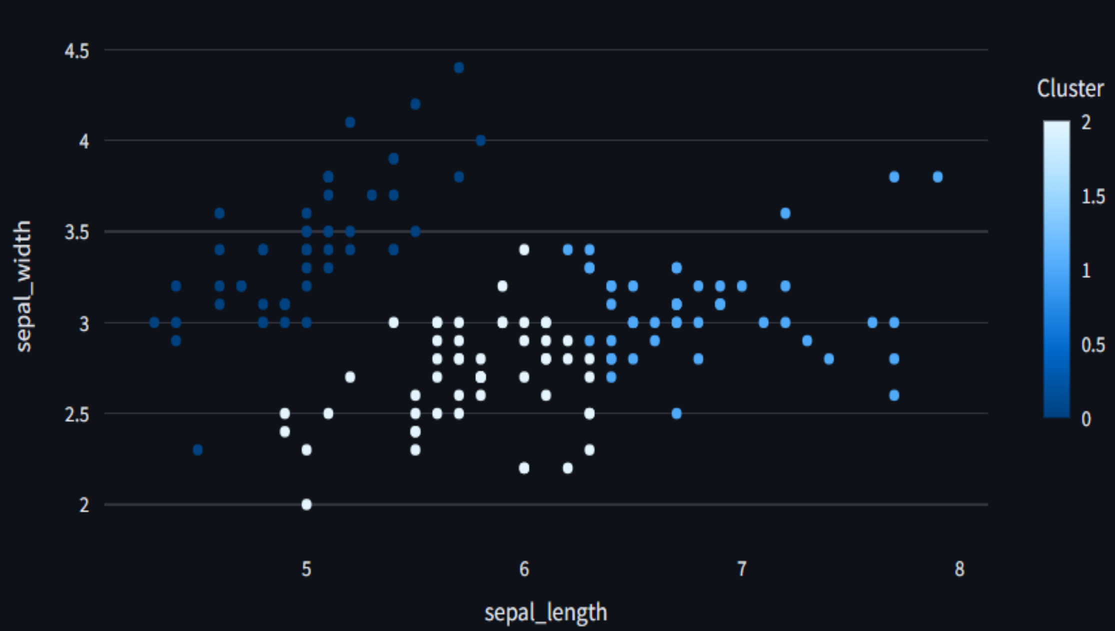
sepal_width




Scatter Plot:
Select x-axis column:
petal_width
Select y-axis column:
petal_length
Scatter Plot
petal_length
petal_width
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


Select a column for the pie chart:
species
Pie Chart for species
Iris-setosa
Iris-virginica
Iris-versicolor
33.3%
33.3%
33.3%

K Means Clustering



Cluster Labels:

	Cluster
48	0
49	0
50	1
51	1
52	1
53	2
54	1

Logistic Regression

Select columns for logistic regression:

Select numeric columns as features for logistic regression:

neckcircumference × thighcircumfere… × shoulderlength ×

Select the target column for Logistic Regression:

Diagnosis

Apply Logistic Regression

Equation of the Line:

Diagnosis = 46.07283176558783 + -0.0659*neckcircumference + -0.0263*thighcircumference + -0.0159*shoulderlength

Accuracy: 0.835985312117503

Confusion Matrix:

0	1
251	86
48	432

Naive Bayes

Select columns for Naive Bayes:

Select numeric columns as features for Naive Bayes:

shoulderlength × neckcircumference × hipbreadth × handlength × chestheight × thighcircumfere… × Heightin × Weightlbs ×

Select the target column for Naive Bayes:

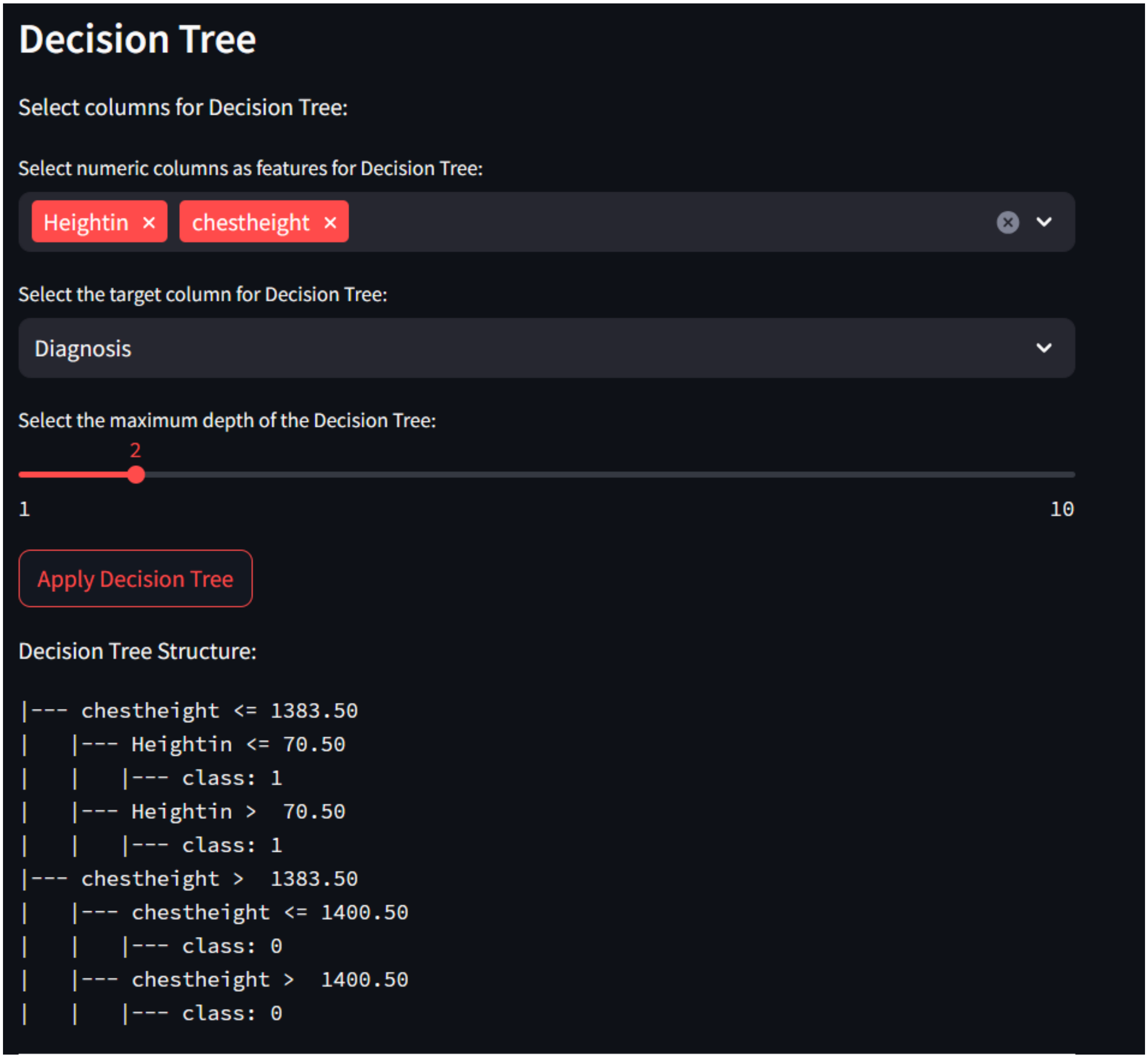
Diagnosis

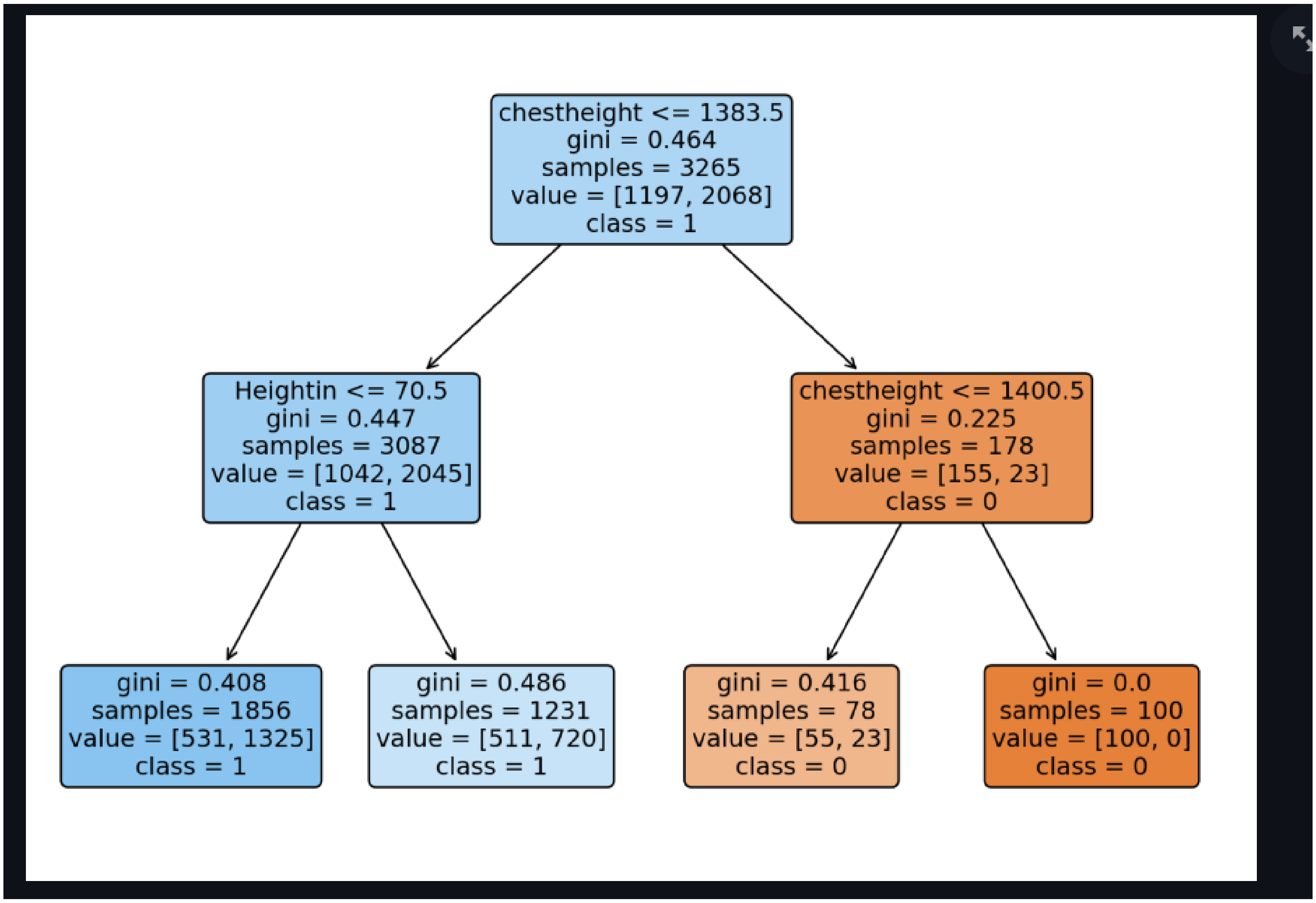
Apply Naive Bayes

Accuracy: 0.8518971848225214

Confusion Matrix:

0	1
273	64
57	423


Decision Tree
Select columns for Decision Tree:
Select numeric columns as features for Decision Tree:
Heightin ×
chestheight ×
Select the target column for Decision Tree:
Diagnosis
Select the maximum depth of the Decision Tree:
2
1
10
Apply Decision Tree
Decision Tree Structure:
|--- chestheight <= 1383.50
| |--- Heightin <= 70.50
| | |--- class: 1
| |--- Heightin > 70.50
| | |--- class: 1
|--- chestheight > 1383.50
| |--- chestheight <= 1400.50
| | |--- class: 0
| |--- chestheight > 1400.50
| | |--- class: 0



CONCLUSION:

In conclusion, this project successfully employed a systematic approach to analyze and visualize the dataset. Through comprehensive data preprocessing, statistical analysis, and diverse visualization techniques, valuable insights regarding data characteristics, patterns, and correlations were unearthed. The findings highlight the significance of data-driven methodologies in extracting actionable insights for informed decision-making, showcasing the power of data analysis and visualization in uncovering meaningful trends within datasets.