Machine Learning Model Creation:

In this project, we have developed a machine learning program designed to predict individuals at risk of lung cancer and analyze the prevalence of this disease over the years using the Lung cancer dataset. Leveraging the dataset focused on lung cancer, our model is trained on patient records that include demographic information and medical history. The primary objective is to create a predictive model capable of accurately identifying individuals prone to major medical events, specifically targeting lung cancer in this context.

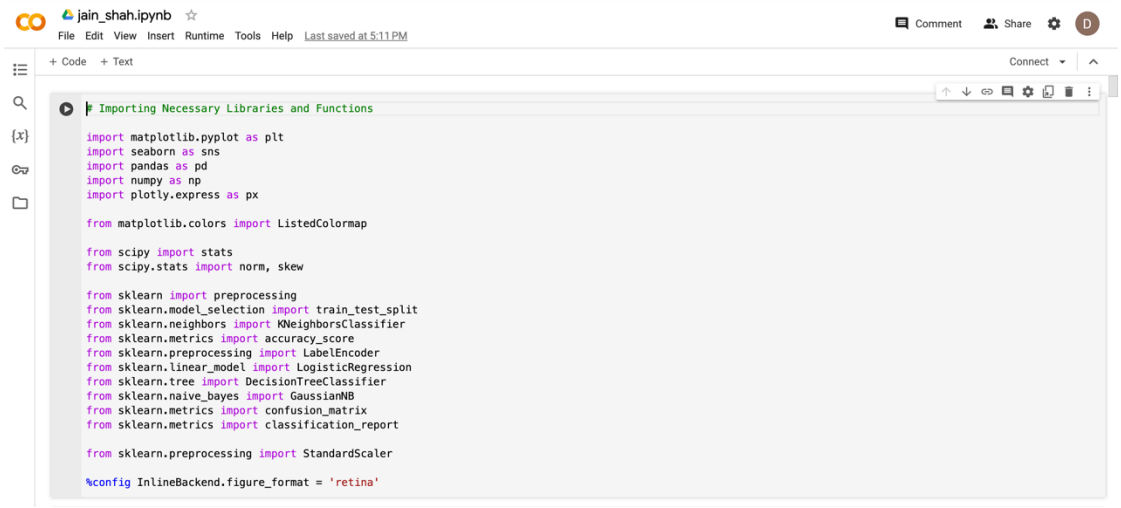
Enterprise Data Architecture plays a crucial role in simplifying, streamlining, and standardizing the accessibility of organizational data. The successful execution of an enterprise data architecture plan involves addressing policies, procedures, and standards governing data collection, storage, management, processing, and utilization across the entire organization. In many large organizations, the data environment is often highly fragmented and characterized by redundancy. This fragmentation is driven by factors such as rapid growth, legacy technology constraints, and insufficient investment in effective data asset management.

The overarching goal of this initiative is to contribute to the early identification of individuals at risk of lung cancer, thereby potentially improving patient treatment outcomes. While the primary focus is on healthcare benefits, we approach this project from the perspective of an insurance business. The developed machine learning model can be utilized to precisely analyze an individual's risk of lung cancer, allowing the insurance company to tailor plans accordingly. This personalized approach enhances the insurance business's ability to offer targeted and effective coverage based on the specific health risks identified by the model.

Implementation Details:

To quickly access the data over cloud, we implemented our project using Google Collab and cloud implementations. We start by importing the necessary libraries for performing various model analysis and tools and metrics for evaluation purposes. Link for notebook.

Step 1:
The provided code is a preamble for a Python script or Jupyter Notebook, importing various libraries and functions essential for data analysis, visualization, and machine learning. The visualization section includes `matplotlib.pyplot` for creating visualizations and `seaborn` for statistical graphics. For data manipulation, the script relies on `pandas` for efficient dataset handling and `numpy` for mathematical operations. The `plotly.express` library is included for interactive and browser-based plotting. In terms of statistical analysis and machine learning, the script imports modules from `scipy` for statistical functions and models from `scikit-learn` (`KNeighborsClassifier`, `LogisticRegression`, `DecisionTreeClassifier`, `GaussianNB`). Additionally, it includes tools for model evaluation such as `accuracy_score`, `LabelEncoder` for converting categorical labels, and metrics like `confusion_matrix` and `classification_report`. The last line configures the inline figure format for improved display in high-resolution environments. Overall, this preamble sets up a comprehensive environment for data exploration, visualization, and machine learning model development.



2. This code snippet serves as an initial setup for data analysis and machine learning tasks. It imports essential Python libraries such as Matplotlib, Seaborn, Pandas, NumPy, and Plotly Express for visualization and data manipulation. The script also includes functions from SciPy for statistical analysis and scikit-learn for machine learning tasks, covering preprocessing, model selection, classification, and evaluation. The final line configures the inline figure format for improved visualization quality. Overall, the code establishes a comprehensive environment for third-person data exploration, visualization, and machine learning model development.

```
[ ] import sys
    import os

    # Setting up a Google Colab environment for working with files stored in Google Drive

    if 'google.colab' in sys.modules:
        from google.colab import drive

        # Mounting Google Drive
        drive.mount('/content/drive/', force_remount=True)


        project_path = "/content/drive/My Drive/"
        sys.path.append(project_path)
        %cd $project_path

    # Current working directory
    print('Current working directory:', os.getcwd())
```

```
Mounted at /content/drive/
/content/drive/My Drive
Current working directory: /content/drive/My Drive
```

```
[ ] #Loading the dataset
    df = pd.read_csv('/content/drive/MyDrive/cancer_data.csv')
```

```
[ ] # Display Data
    display(df)
    print('\n')
```

3. The next few steps involve loading the dataset and displaying it as shown in the screenshot.

```
# Display Data
display(df)
print('\n')
```

	index	Patient Id	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	...	Fatigue	Weight Loss	Shortness of Breath	Wheezing	Swallowing Difficulty	Clubbing of Finger Nails	Frequ Cı
0	0	P1	33	1	2	4	5	4	3	2	...	3	4	2	2	3	1	
1	1	P10	17	1	3	1	5	3	4	2	...	1	3	7	8	6	2	
2	2	P100	35	1	4	5	6	5	5	4	...	8	7	9	2	1	4	
3	3	P1000	37	1	7	7	7	7	6	7	...	4	2	3	1	4	5	
4	4	P101	46	1	6	8	7	7	7	6	...	3	2	4	1	4	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
995	995	P995	44	1	6	7	7	7	7	6	...	5	3	2	7	8	2	
996	996	P996	37	2	6	8	7	7	7	6	...	9	6	5	7	2	4	
997	997	P997	25	2	4	5	6	5	5	4	...	8	7	9	2	1	4	
998	998	P998	18	2	6	8	7	7	7	6	...	3	2	4	1	4	2	
999	999	P999	47	1	6	5	6	5	5	4	...	8	7	9	2	1	4	

1000 rows × 26 columns

4. This code removes a specific column, converts all column names to lowercase, replaces spaces with underscores in column names, and then displays the cleaned DataFrame.

```
print('\n')
df.drop("index", axis=1, inplace=True)

# Cleaning Column Names
df.rename(columns=str.lower, inplace=True)
df.rename(columns={col: col.replace(" ", "_") for col in df.columns}, inplace=True)

# Display Data After Cleaning
display(df)
print('\n')
```

	patient_id	age	gender	air_pollution	alcohol_use	dust_allergy	occupational_hazards	genetic_risk	chronic_lung_disease	balanced_diet	...	fatigue	weigl
0	P1	33	1	2	4	5	4	3	2	2	...	3	
1	P10	17	1	3	1	5	3	4	2	2	...	1	
2	P100	35	1	4	5	6	5	5	4	6	...	8	
3	P1000	37	1	7	7	7	7	6	7	7	...	4	
4	P101	46	1	6	8	7	7	7	6	7	...	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	P995	44	1	6	7	7	7	7	6	7	...	5	
996	P996	37	2	6	8	7	7	7	6	7	...	9	
997	P997	25	2	4	5	6	5	5	4	6	...	8	
998	P998	18	2	6	8	7	7	7	6	7	...	3	
999	P999	47	1	6	5	6	5	5	4	6	...	8	

1000 rows × 25 columns

5. Checking for null values in the dataset in order to clean it, as more amount of null values will lead to a higher inaccuracy in the ML Model.

```
# Check For Null Values

print('\n')
df.isnull().sum()
```

```
patient_id                0
age                       0
gender                    0
air_pollution             0
alcohol_use               0
dust_allergy              0
occupational_hazards      0
genetic_risk              0
chronic_lung_disease      0
balanced_diet             0
obesity                   0
smoking                   0
passive_smoker            0
chest_pain                0
coughing_of_blood         0
fatigue                   0
weight_loss               0
shortness_of_breath       0
wheezing                  0
swallowing_difficulty     0
clubbing_of_finger_nails  0
frequent_cold             0
dry_cough                 0
snoring                   0
level                     0
dtype: int64
```

6. Displaying the information regarding the null and non-null values in the dataset along with each header respectively.

```
# Print Information
print('\n')
print(df.info())
print('\n')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   patient_id                1000 non-null   object
 1   age                       1000 non-null   int64 
 2   gender                    1000 non-null   int64 
 3   air_pollution             1000 non-null   int64 
 4   alcohol_use               1000 non-null   int64 
 5   dust_allergy              1000 non-null   int64 
 6   occupational_hazards      1000 non-null   int64 
 7   genetic_risk              1000 non-null   int64 
 8   chronic_lung_disease      1000 non-null   int64 
 9   balanced_diet             1000 non-null   int64 
 10  obesity                   1000 non-null   int64 
 11  smoking                   1000 non-null   int64 
 12  passive_smoker            1000 non-null   int64 
 13  chest_pain                1000 non-null   int64 
 14  coughing_of_blood         1000 non-null   int64 
 15  fatigue                   1000 non-null   int64 
 16  weight_loss               1000 non-null   int64 
 17  shortness_of_breath       1000 non-null   int64 
 18  wheezing                  1000 non-null   int64 
 19  swallowing_difficulty     1000 non-null   int64 
 20  clubbing_of_finger_nails  1000 non-null   int64 
 21  frequent_cold             1000 non-null   int64 
 22  dry_cough                 1000 non-null   int64 
 23  snoring                   1000 non-null   int64 
 24  level                     1000 non-null   object
dtypes: int64(23), object(2)
memory usage: 195.4+ KB
```

7. Since we faced inconvenience to document the levels/intensity in the dataset, we are using integers to make the procedure easy.

```
# Replace "level" with Integer

print('\n')
print('Cancer Levels: ', df['level'].unique())

# Replacing levels with int
df["level"].replace({'High': 2, 'Medium': 1, 'Low': 0}, inplace=True)
print('Cancer Levels: ', df['level'].unique())

print('\nColumns in dataframe: \n', df.columns)
print('\n')
```

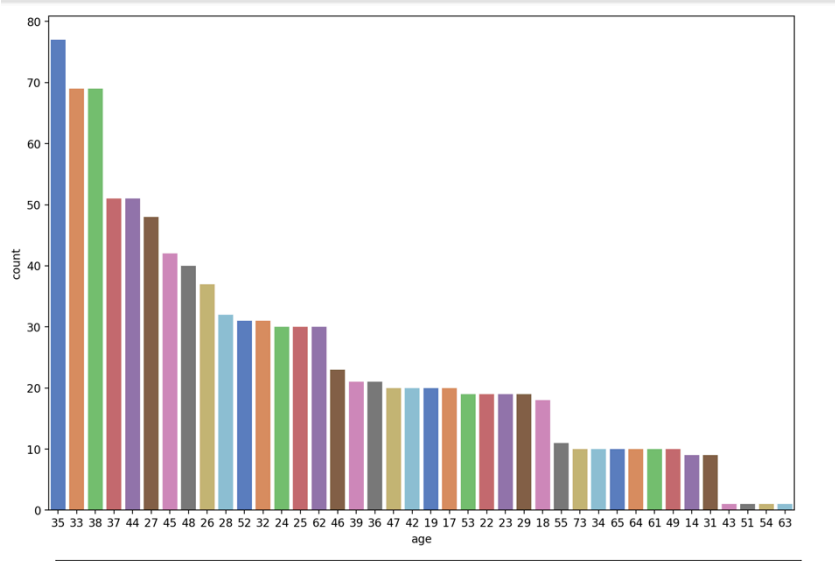
```
Cancer Levels:  ['Low' 'Medium' 'High']
Cancer Levels:  [0 1 2]

Columns in dataframe:
 Index(['patient_id', 'age', 'gender', 'air_pollution', 'alcohol_use',
       'dust_allergy', 'occupational_hazards', 'genetic_risk',
       'chronic_lung_disease', 'balanced_diet', 'obesity', 'smoking',
       'passive_smoker', 'chest_pain', 'coughing_of_blood', 'fatigue',
       'weight_loss', 'shortness_of_breath', 'wheezing',
       'swallowing_difficulty', 'clubbing_of_finger_nails', 'frequent_cold',
       'dry_cough', 'snoring', 'level'],
      dtype='object')
```

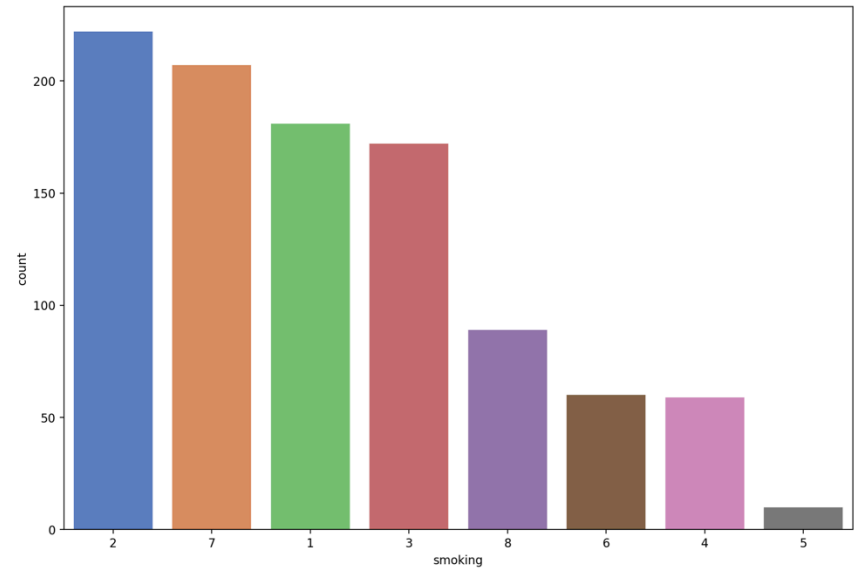
Data Visualizations:

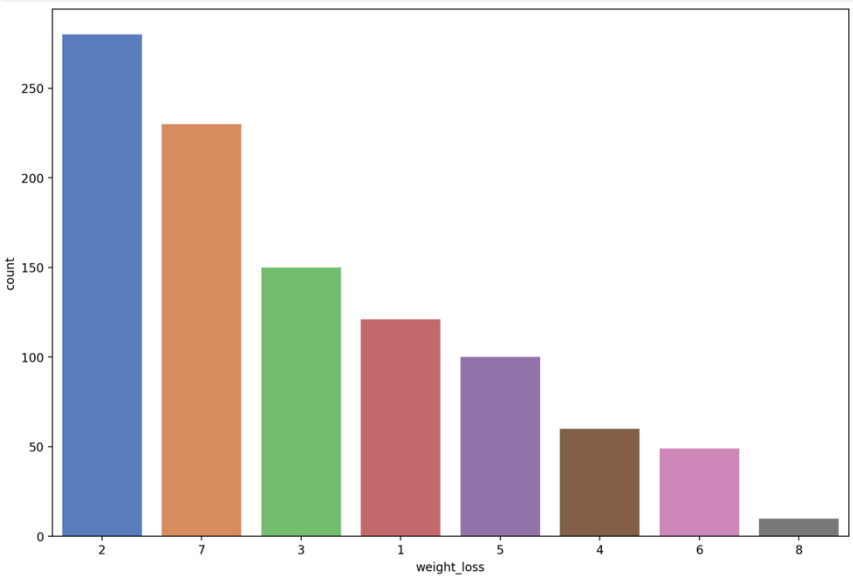
8. This code generates a series of count plots to visually analyze categorical data in the DataFrame (`df`). It employs the Matplotlib and Seaborn libraries for plotting. The first loop

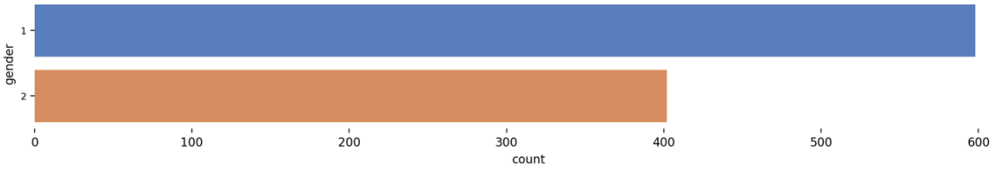
iterates through the elements in the `group1` list and creates individual count plots for each categorical variable in `group1`. Each plot is displayed with a specific order of categories based on their frequency in the dataset, and the color palette is set to 'muted'. The second loop operates similarly but is designed for variables in `group2`, creating horizontal count plots. Both sets of plots are shown with specified figure sizes, and axis labels are set to the respective categorical variable names. Additionally, adjustments to tick sizes and styles are made for better readability. Overall, this code facilitates a visual examination of the distribution of categorical variables in the DataFrame, aiding in the identification of patterns and trends in the data.

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns

    print('\n')
    for i in group1:
        fig, ax = plt.subplots(1, 1, figsize=(12, 8))
        sns.countplot(x=df[i], data=df, order=df[i].value_counts().index, palette='muted')
        plt.xlabel(i)
        plt.xticks(fontsize=10)
        plt.show()

    for i in group2:
        fig, ax = plt.subplots(1,1, figsize=(15, 2))
        sns.countplot(y = df[i],data=df, order=df[i].value_counts().index, palette='muted')
        plt.ylabel(i)
        plt.yticks(fontsize=8)
        plt.box(False)
        plt.show()

    print('\n')
```





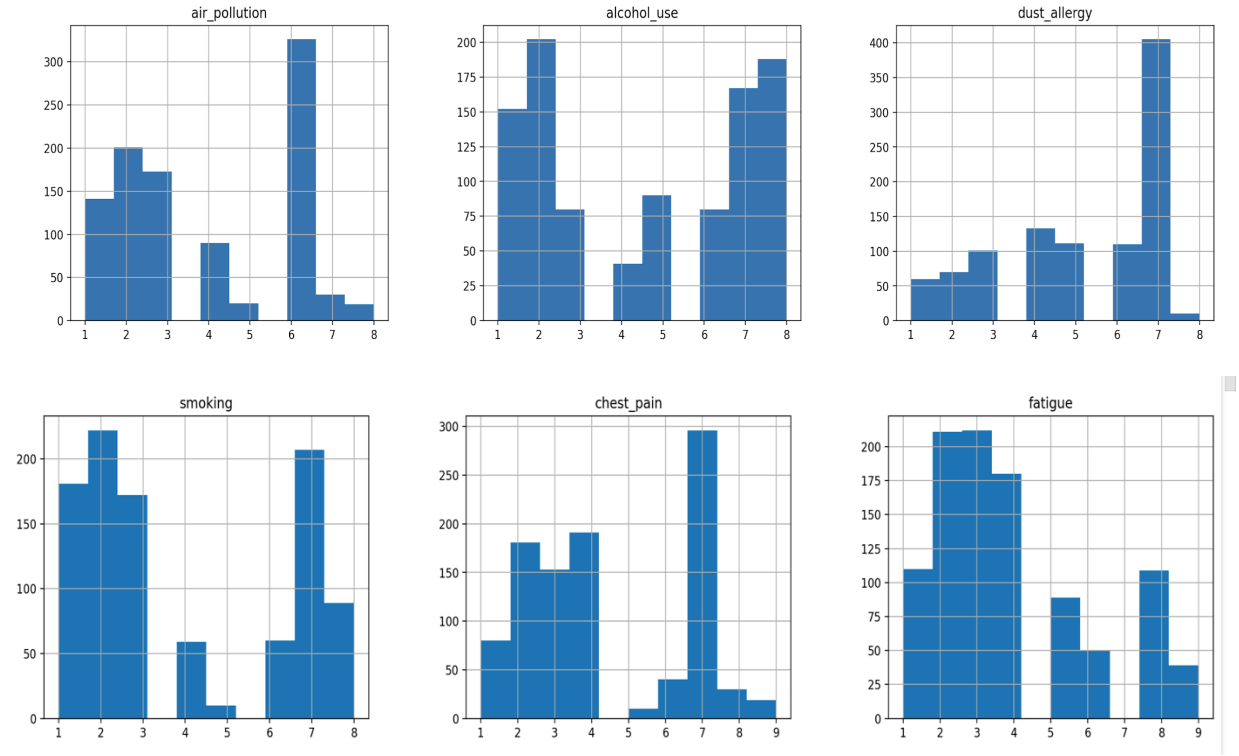




9. This code segment generates histograms for numerical variables in the DataFrame (`df`) represented by the `group3` list. It utilizes Matplotlib to create a 3x2 grid of subplots with a total of six histograms. The loop iterates through the variables in `group3`, creating individual histograms for each variable using the `hist` method in Pandas. The resulting histograms are displayed within the subplot grid, and each subplot is labeled with the corresponding variable name. The `print('\n')` statement adds a newline for better formatting in the output. Overall, this

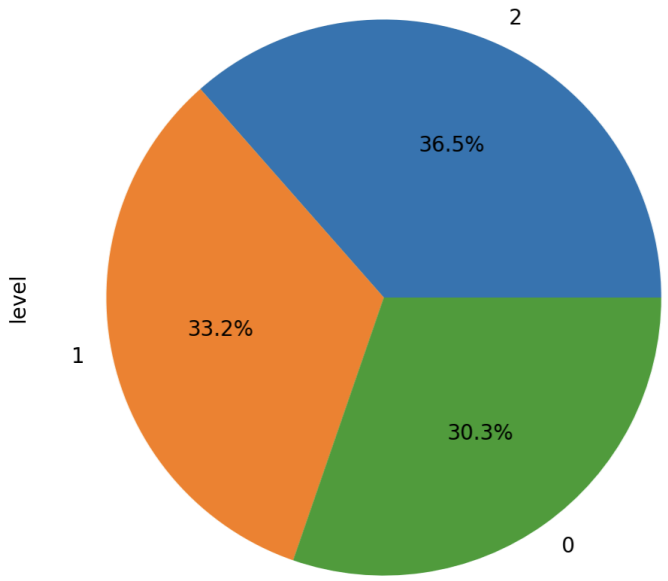
code provides a visual representation of the distribution of numerical variables, allowing for a quick overview of their underlying patterns and characteristics.

```
# Histograms
print('\n')
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20, 10))
ax = ax.flatten()
i = 0

for c in group3:
    df.hist(c, figsize=(4,4), ax=ax[i], label=f'{c}')
    i = i + 1;
plt.show()
print('\n')
```



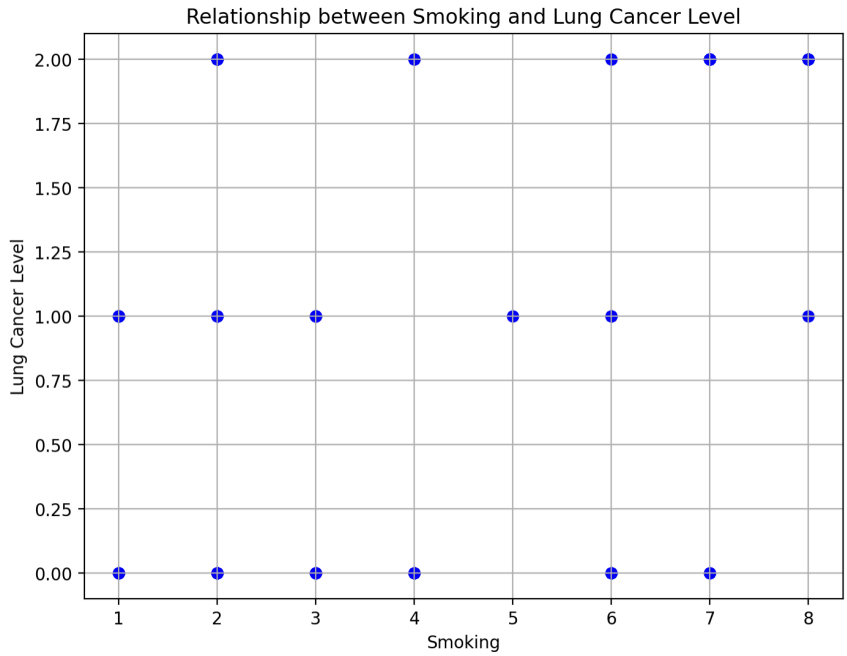
10. Now we will create a pie chart for the data visualization for the levels high, low and medium.

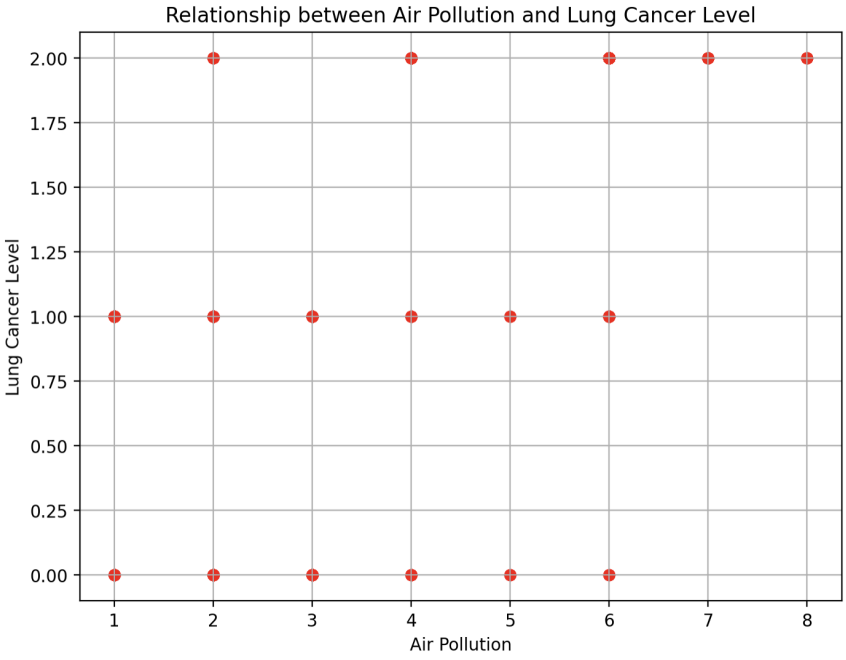
```
[ ]  # Pie Chart

     df['level'].value_counts().plot(kind='pie', figsize=(6, 6), autopct='%1.1f%%')
     plt.show()
```



11. A scatter plot has been created for smoking and label of lung cancer as well as Air pollution and Label of lung cancer

```
# Scatter Plot: Smoking and Label of Lung Cancer
plt.figure(figsize=(8, 6))
plt.scatter(df['smoking'], df['level'], alpha=0.5, color='blue')
plt.title('Relationship between Smoking and Lung Cancer Level')
plt.xlabel('Smoking')
plt.ylabel('Lung Cancer Level')
plt.grid(True)

print('\n')

# Scatter Plot: Air Pollution and Label of Lung Cancer
plt.figure(figsize=(8, 6))
plt.scatter(df['air_pollution'], df['level'], alpha=0.5, color='red')
plt.title('Relationship between Air Pollution and Lung Cancer Level')
plt.xlabel('Air Pollution')
plt.ylabel('Lung Cancer Level')
plt.grid(True)
plt.show()

print('\n')
```


Relationship between Smoking and Lung Cancer Level
Lung Cancer Level
Smoking
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3
4
5
6
7
8

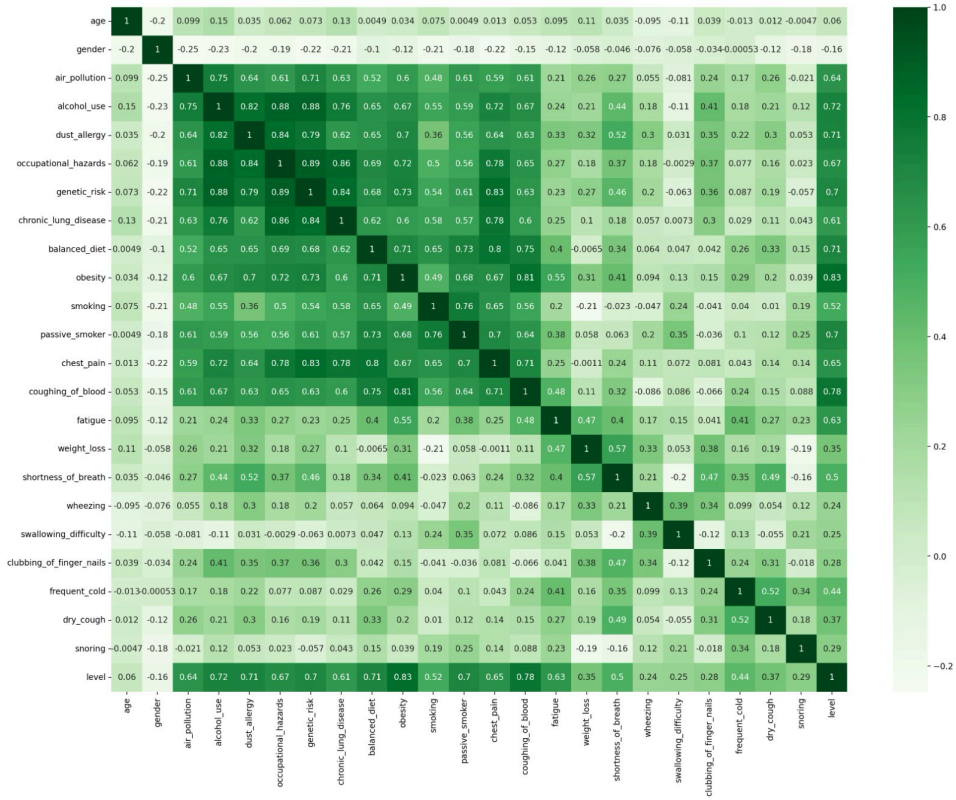

Relationship between Air Pollution and Lung Cancer Level
Lung Cancer Level
Air Pollution
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3
4
5
6
7
8

12. This code segment performs a train-test split on the dataset, separating features (`X`) and labels (`y`) into training and testing sets. It utilizes the `train_test_split` function from scikit-learn, with a test size of 20% and a fixed random state for reproducibility (`random_state=42`). The resulting split data is stored in four variables: `X_train` and `y_train` represent the features and labels of the training set, while `X_test` and `y_test` correspond to the testing set. The subsequent print statements display the shapes of the training and testing sets, providing information about the number of samples and features in each set for both features and labels. This is a crucial step in preparing the data for machine learning model training and evaluation.

```
[ ] # Train-Test Split

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    print('\nTrain Shape\n')
    print('X train shape: ', X_train.shape)
    print('Y train shape: ', y_train.shape)
    print('\n\nTest Shape\n')
    print('X test shape: ', X_test.shape)
    print('Y test shape: ', y_test.shape)
    print('\n')
```

```
Train Shape

X train shape:  (800, 23)
Y train shape:  (800,)


Test Shape

X test shape:  (200, 23)
Y test shape:  (200,)
```

13. Creating a heatmap



Testing the ML Models

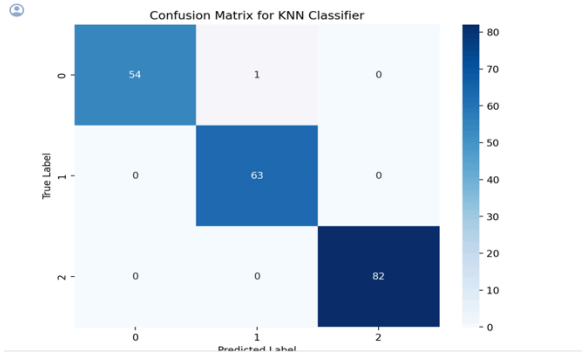
1. This code segment implements the k-Nearest Neighbors (KNN) classification algorithm. It first creates a KNN classifier with 5 neighbors using the `KNeighborsClassifier` from scikit-learn. The model is then trained on the training data (`X_train` and `y_train`) using the `fit` method. Subsequently, predictions are made on the test data (`X_test`), and the accuracy of the model is calculated by comparing the predicted labels (`y_pred_knn`) with the actual labels of the test set (`y_test`). The accuracy score is then printed, representing the proportion of correctly classified instances. This provides an evaluation metric for assessing the performance of the KNN model on the given dataset.

```
[ ]  # KNN

     knn = KNeighborsClassifier(n_neighbors=5)
     knn.fit(X_train, y_train)

     # Making predictions and calculating the accuracy
     y_pred_knn = knn.predict(X_test)
     accuracy_knn = accuracy_score(y_test, y_pred)

     print(f'The accuracy of KNN is {accuracy_knn}')
```

```
The accuracy of KNN is 0.995
```

2. This code generates a confusion matrix for evaluating the performance of the K-Nearest Neighbors (KNN) classifier. The confusion matrix is constructed using the `confusion_matrix` function from scikit-learn, comparing the true labels (`y_test`) against the predicted labels (`y_pred_knn`). The resulting matrix is then visualized as a heatmap using Matplotlib and Seaborn. The heatmap includes annotations of the numerical values in each cell, representing the count of instances for each combination of true and predicted labels. This graphical representation allows for a detailed examination of the model's performance in terms of correct and incorrect predictions across different classes. The title, x-axis label, and y-axis label provide additional context for interpreting the confusion matrix.

```
cm = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.title('Confusion Matrix for KNN Classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



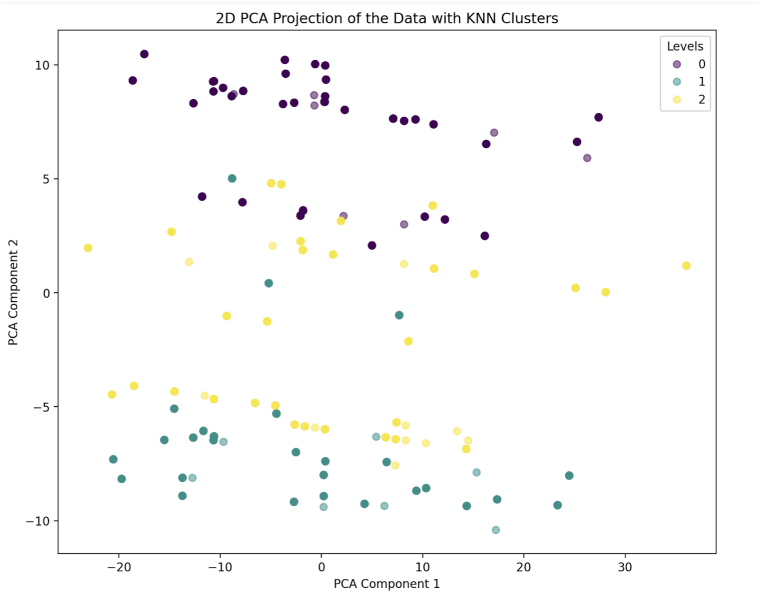
3. This code generates and prints the classification report for the K-Nearest Neighbors (KNN) classifier. The `classification_report` function from scikit-learn is employed, taking the true labels (`y_test`) and the predicted labels (`y_pred_knn`). The classification report provides a detailed summary of the model's performance, including precision, recall, and F1-score for each class, as well as overall accuracy. This information is valuable for assessing the classifier's ability to correctly classify instances within each class and provides a comprehensive view of its performance across different categories. The printed output offers insights into the precision and recall metrics, aiding in a more nuanced evaluation of the model's effectiveness.

```python
# Classification Report

print('\nClassification Report for KNN\n')
print(classification_report(y_test, y_pred_knn))
print('\n')
```

```
Classification Report for Decision Tree

              precision    recall  f1-score   support

           0       1.00      0.98      0.99        55
           1       0.98      1.00      0.99        63
           2       1.00      1.00      1.00        82

    accuracy                           0.99       200
   macro avg       0.99      0.99      0.99       200
weighted avg       1.00      0.99      0.99       200
```

4. This code snippet utilizes Principal Component Analysis (PCA) for dimensionality reduction and visualizes the data in a 2D space with clusters assigned by the K-Nearest Neighbors (KNN) classifier. The `PCA` class from scikit-learn is employed with `n_components=2` to project the data onto a 2D plane. The resulting reduced feature set is stored in `X_reduced`. The KNN model is then used to predict clusters for each data point. The data is visualized using a scatter plot where each point represents an instance in the reduced feature space, colored according to the assigned KNN cluster. The `plt.scatter` function creates the scatter plot, and labels, title, and legend elements are added for clarity. This visualization provides insights into the distribution of data points in the reduced space and the clusters assigned by the KNN algorithm.

```python
from sklearn.decomposition import PCA
import numpy as np

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

clusters = knn.predict(X)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=clusters, cmap='viridis', alpha=0.5)
plt.title('2D PCA Projection of the Data with KNN Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(*scatter.legend_elements(), title="Levels")
plt.show()
```



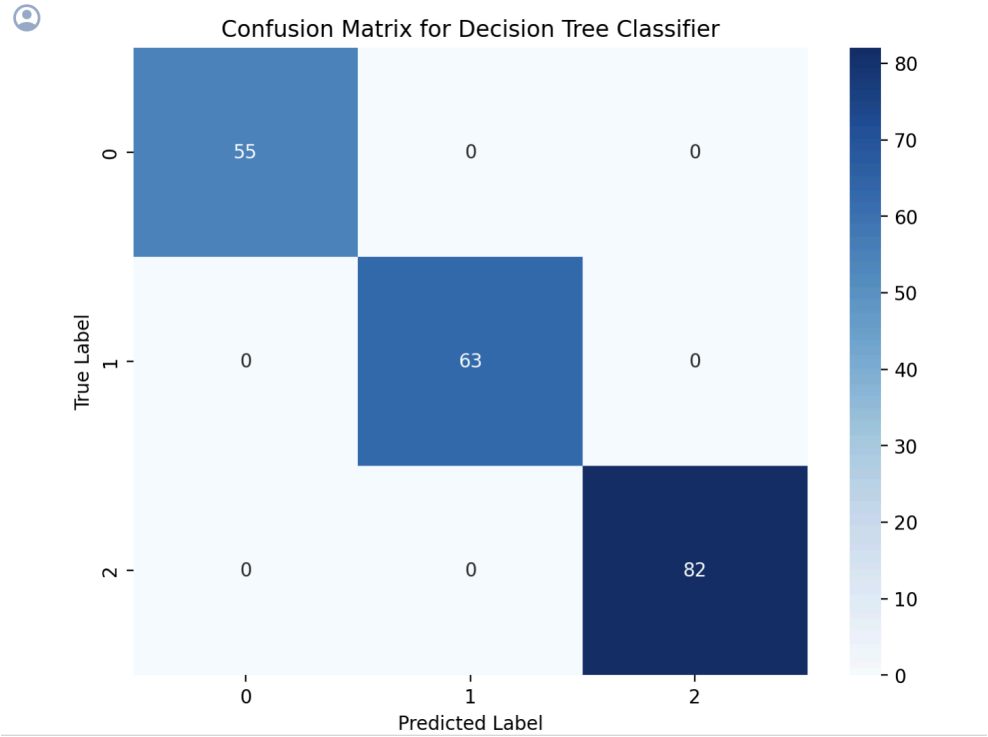
5. This code segment implements a Decision Tree classifier using scikit-learn. It creates a DecisionTreeClassifier with a specified random state for reproducibility (`random_state=42`) and trains the model on the training data (`X_train` and `y_train`) using the `fit` method. Subsequently, predictions are made on the test data (`X_test`), and the accuracy of the model is calculated by comparing the predicted labels (`y_pred_dtree`) with the actual labels of the test set (`y_test`). The accuracy score is then printed, providing an evaluation metric for assessing the performance of the Decision Tree classifier on the given dataset.

```
# Decision Tree

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train, y_train)

y_pred_dtree = dtree.predict(X_test)

accuracy_dtree = accuracy_score(y_test, y_pred_dtree)
print(f'The accuracy of Decision Tree is {accuracy_dtree}')
```

```
The accuracy of Decision Tree is 1.0
```

Confusion Matric for Decision Tree classifier

```
cm_dtree = confusion_matrix(y_test, y_pred_dtree)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dtree, annot=True, fmt="d", cmap='Blues')
plt.title('Confusion Matrix for Decision Tree Classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



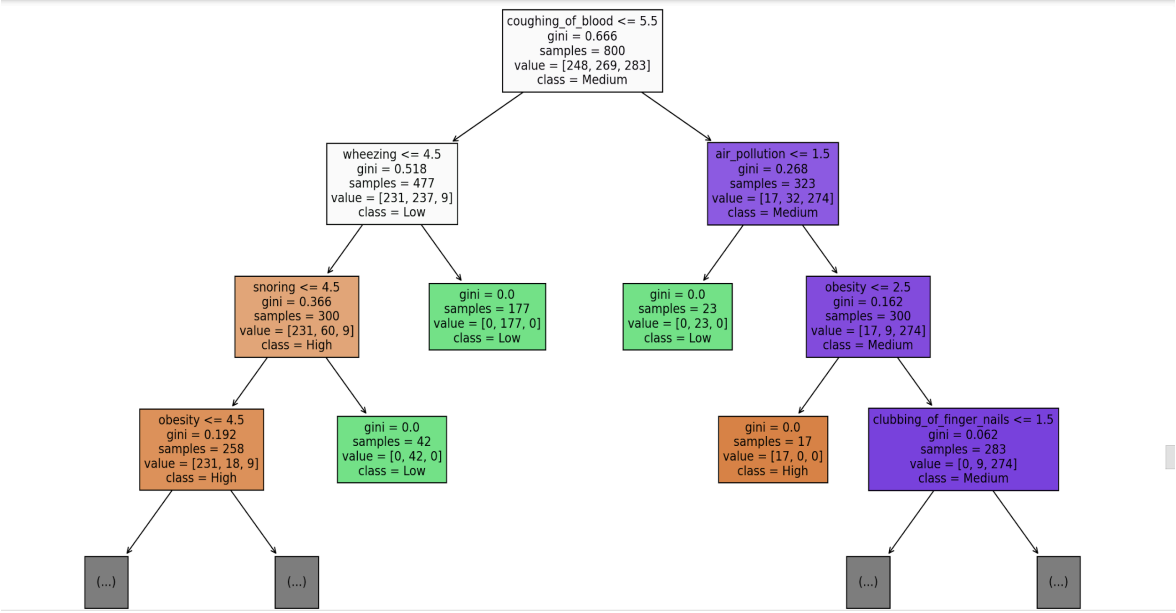
The classification report for the decision tree is as follows:

```
[ ]  # Classification Report

     print('\nClassification Report for Decision Tree\n')
     print(classification_report(y_test, y_pred_dtree))
     print('\n')
```

```
Classification Report for Decision Tree

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00        63
           2       1.00      1.00      1.00        82

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

6. This code generates a visual representation of the Decision Tree classifier using the `plot_tree` function from scikit-learn's `tree` module. The resulting decision tree is plotted with Matplotlib, and features such as node colors, feature names, and class names are specified for better interpretation. The `max_depth=3` parameter limits the depth of the tree to enhance readability, and the resulting plot provides a truncated view of the Decision Tree. This visualization offers insights into the structure of the classifier, illustrating decision nodes, splits, and leaf nodes. It serves as a valuable tool for understanding how the model makes decisions based on the input features.

```
[ ]  from sklearn.tree import plot_tree

     # Plotting the decision tree
     plt.figure(figsize=(20, 10))
     plot_tree(dtree, filled=True, feature_names=X.columns, class_names=label_encoder.classes_, max_depth=3)
     plt.title('Decision Tree Classifier (Truncated View)')
     plt.show()
```

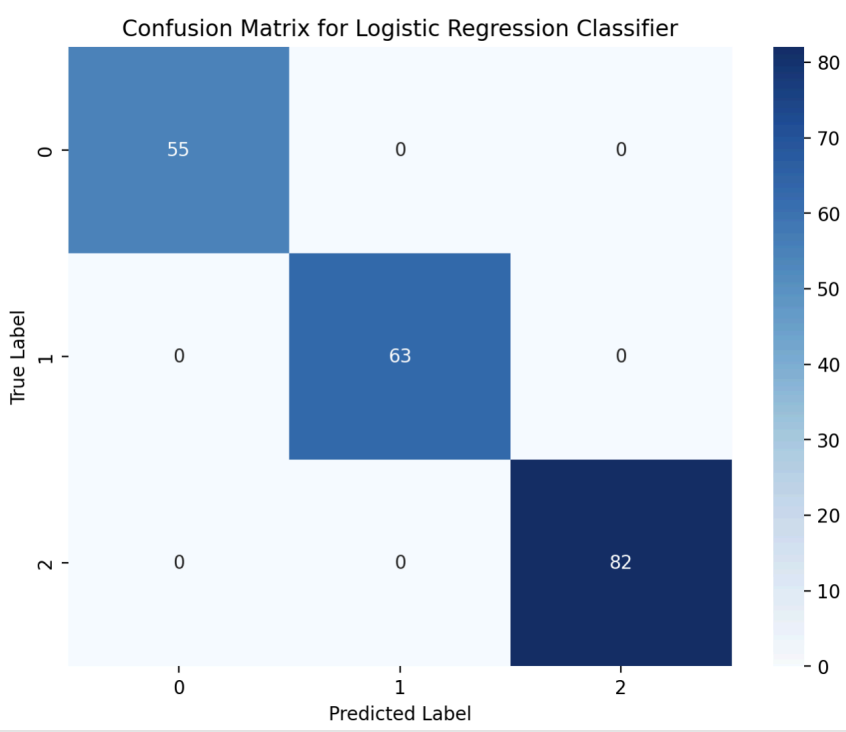


7. This code segment implements a Logistic Regression classifier using scikit-learn. It creates a LogisticRegression model with a specified maximum number of iterations and random state for reproducibility (`max_iter=2000, random_state=42`). The model is then trained on the training data (`X_train` and `y_train`) using the `fit` method. Subsequently, predictions are made on the test data (`X_test`), and the accuracy of the model is calculated by comparing the predicted labels (`y_pred_logreg`) with the actual labels of the test set (`y_test`). The accuracy score is then printed, providing an evaluation metric for assessing the performance of the Logistic Regression classifier on the given dataset.

```
[ ] # Logistic Regression

    from sklearn.linear_model import LogisticRegression

    logreg = LogisticRegression(max_iter=2000, random_state=42)
    logreg.fit(X_train, y_train)

    y_pred_logreg = logreg.predict(X_test)

    accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
    print(f'The accuracy of Logistic Regression is {accuracy_logreg}')
```

```
The accuracy of Decision Tree is 1.0
```

```
cm_logreg = confusion_matrix(y_test, y_pred_logreg)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_logreg, annot=True, fmt="d", cmap='Blues')
plt.title('Confusion Matrix for Logistic Regression Classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```
# Classification Report

print('\nClassification Report for Logistic Regression\n')
print(classification_report(y_test, y_pred_logreg))
print('\n')
```

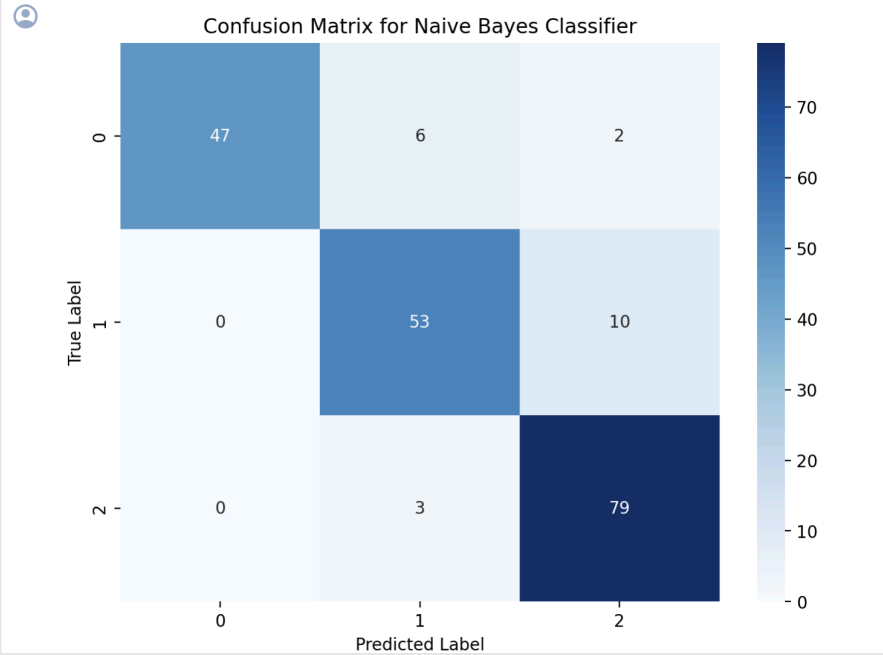
```
Classification Report for Logistic Regression

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00        63
           2       1.00      1.00      1.00        82

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

8. This code segment implements a Naive Bayes classifier using scikit-learn's Gaussian Naive Bayes model. It creates a `GaussianNB` object and trains the model on the training data (`X_train` and `y_train`) using the `fit` method. Subsequently, predictions are made on the test data (`X_test`), and the accuracy of the model is calculated by comparing the predicted labels (`y_pred_nb`) with the actual labels of the test set (`y_test`). The accuracy score is then printed, providing an evaluation metric for assessing the performance of the Naive Bayes classifier on the given dataset.

```
# Naive Bayes

from sklearn.naive_bayes import GaussianNB

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

y_pred_nb = nb.predict(X_test)

accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f'The accuracy of Naive Bayes is {accuracy_nb}')
```

```
The accuracy of Naive Bayes is 0.895
```

```
cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt="d", cmap='Blues')
plt.title('Confusion Matrix for Naive Bayes Classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




```
# Classification Report

print('\nClassification Report for Naive Bayes\n')
print(classification_report(y_test, y_pred_nb))
print('\n')
```

```
Classification Report for Naive Bayes

              precision    recall  f1-score   support

           0       1.00      0.85      0.92        55
           1       0.85      0.84      0.85        63
           2       0.87      0.96      0.91        82

    accuracy                           0.90       200
   macro avg       0.91      0.89      0.89       200
weighted avg       0.90      0.90      0.90       200
```

9. This code segment creates a Pandas DataFrame named `scores_df` to organize and display the accuracy scores of different machine learning models. The accuracy scores are stored in a dictionary named `model_scores`, where each model's name and corresponding accuracy score are specified. The models include K-Nearest Neighbors, Decision Tree, Logistic Regression, and Naive Bayes. The DataFrame is then constructed using the `pd.DataFrame` constructor,

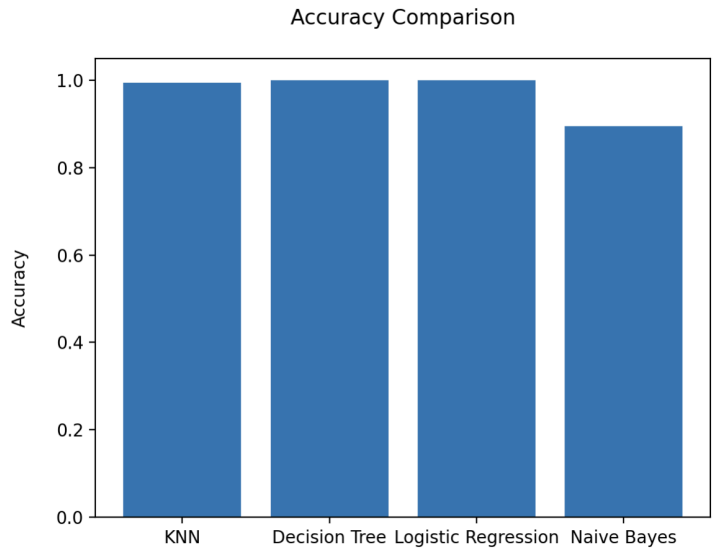
providing a clear tabular representation of the model names and their respective accuracy scores. The resulting DataFrame can be used for easy comparison and analysis of the performance of different models on the given dataset.

```
import pandas as pd

model_scores = {
    "Model": ["K-Nearest Neighbors", "Decision Tree", "Logistic Regression", "Naive Bayes"],
    "Accuracy Score": [accuracy, accuracy_dtree, accuracy_logreg, accuracy_nb]
}

scores_df = pd.DataFrame(model_scores)
scores_df
```

	Model	Accuracy Score
0	K-Nearest Neighbors	0.995
1	Decision Tree	1.000
2	Logistic Regression	1.000
3	Naive Bayes	0.895

10. This code snippet generates a bar plot to visually compare the accuracy scores of different machine learning models. It uses Matplotlib to create a bar chart where the x-axis represents the model names ('KNN', 'Decision Tree', 'Logistic Regression', 'Naive Bayes'), and the y-axis represents the corresponding accuracy scores. The plot is labeled with axes titles and a main title for clarity. This visualization allows for a quick comparison of the performance of each model, providing a visual summary of their accuracy scores. The `print('\n')` statements add newlines for better formatting in the output.

```
# Bar Plot

print('\n')
plt.bar(x,y)
plt.xlabel('\n Models')
plt.ylabel("Accuracy\n")
plt.title('Accuracy Comparison\n')
plt.show()
print('\n')
```



11. This code segment uses the `pickle` module to serialize and save the trained Decision Tree model (`dtree`) to a file named "prediction_model.pkl." The file is opened in binary write mode (``'wb'`), and the `pickle.dump` function is employed to store the model in the file. This serialized model file can later be used to load the model and make predictions without retraining. The specified file path ("/content/drive/MyDrive/prediction_model.pkl") suggests that the file is saved in the Google Drive directory.

```
import pickle

with open('/content/drive/MyDrive/prediction_model.pkl', 'wb') as file:
    pickle.dump(dtree, file)
```