COM7003-24/25-Artificial Intelligence

Lung Cancer Prediction Survey

Module Leader: Dr. Xin Lu

Student Name: Safa Dana

Student ID: 2404831

Contents

Introduction

Lung cancer is one of the leading causes of cancer-related deaths worldwide. Early detection of lung cancer can improve treatment outcomes and increase the chances of survival. Patients diagnosed with lung cancer through early CT screening have a 20-year lung cancer-specific survival rate of 81% (Lokaj, 2023). With advancements in technology and data science, machine learning models can help predict the likelihood of lung cancer based on medical data.

This report presents a project focused on building a predictive model for lung cancer using a Jupyter notebook. The main goal of this project is to analyze medical data, apply machine learning techniques, and evaluate the model's performance in predicting lung cancer. The report covers data preprocessing, model selection, evaluation metrics, and the final results of the prediction models.

Details of the Dataset

The dataset used in this project is named **"Lung Cancer"**, sourced from Kaggle (Bhat, 2022). It contains medical data related to lung cancer diagnosis, with the goal of predicting whether a patient has lung cancer based on various attributes. The link for the dataset is:
https://www.kaggle.com/datasets/mysarahmadbhat/lung-cancer

The dataset includes the following key features:

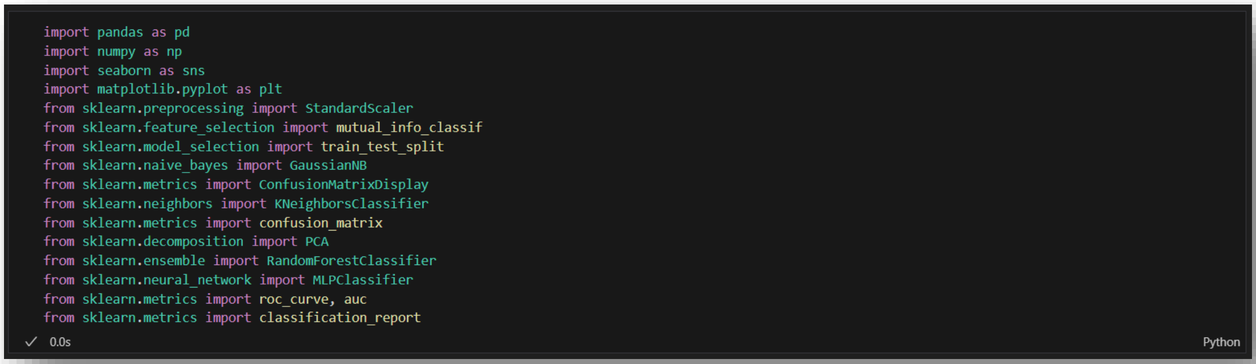
- **Age**: The age of the patient (**Integer**).
- **Gender**: The gender of the patient (**M** or **F**).
- **Smoking**: Indicates whether the patient is a smoker (**1** for **No**, **2** for **Yes**).
- **Yellow fingers**: Whether the patient's fingers are yellow, which can be a sign of heavy smoking (**1** for **No**, **2** for **Yes**).
- **Anxiety**: Whether the patient has anxiety (**1** for **No**, **2** for **Yes**).
- **Peer pressure**: Whether the patient is influenced by peer pressure (**1** for **No**, **2** for **Yes**).
- **Chronic disease**: Indicates if the patient has a chronic disease (**1** for **No**, **2** for **Yes**).
- **Fatigue**: Whether the patient often feels tired (**1** for **No**, **2** for **Yes**).
- **Allergy**: Whether the patient has allergies (**1** for **No**, **2** for **Yes**).
- **Wheezing**: Indicates if the patient experiences wheezing (**1** for **No**, **2** for **Yes**).
- **Alcohol consumption**: Whether the patient consumes alcohol (**1** for **No**, **2** for **Yes**).
- **Coughing**: Whether the patient has a persistent cough (**1** for **No**, **2** for **Yes**).
- **Shortness of breath**: Whether the patient experiences difficulty breathing (**1** for **No**, **2** for **Yes**).
- **Swallowing difficulty**: Whether the patient has trouble swallowing (**1** for **No**, **2** for **Yes**).

- **Chest pain**: Whether the patient has chest pain (**1** for **No**, **2** for **Yes**).
- **Lung Cancer**: The target variable indicating if the patient has lung cancer (**Yes** or **No**).

The dataset contains **309 rows** and **16 columns**, with the **Lung Cancer** column as the target variable for prediction. The data includes both categorical and binary features, which require preprocessing before applying machine learning models.

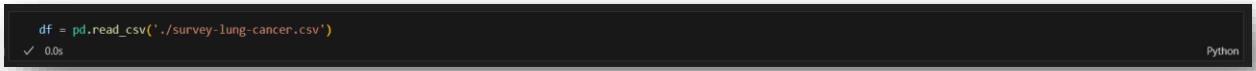
Import necessary libraries

First, important libraries must be imported to the project.

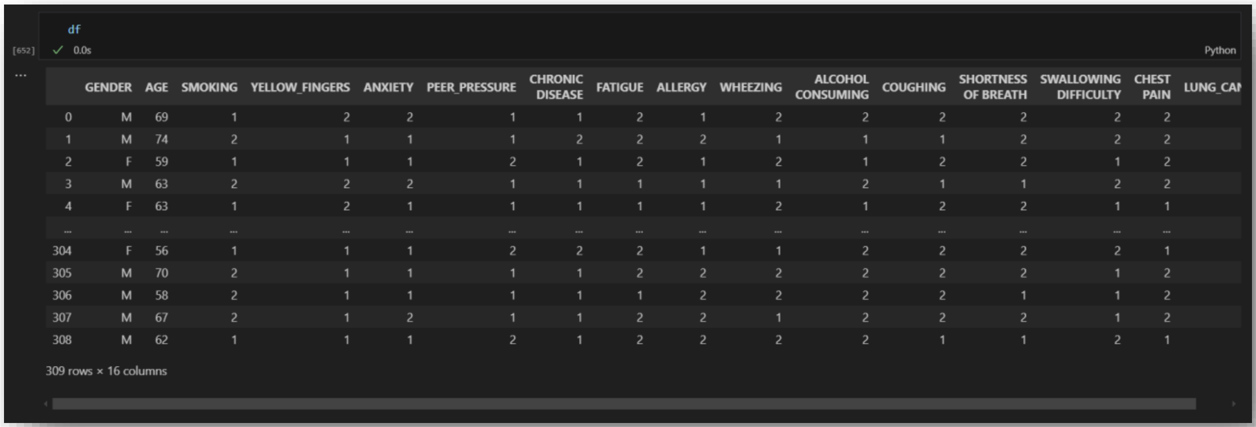


```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import mutual_info_classif
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
```

The first step after importing the libraries is to load the dataset into memory as a Data Frame.



```python
df = pd.read_csv('./survey-lung-cancer.csv')
```



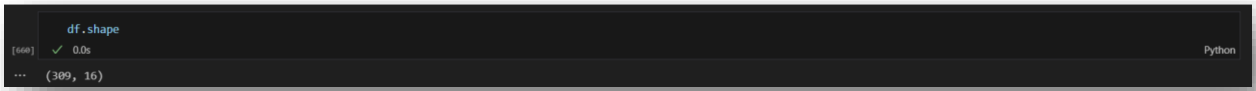
```python
df
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CA
0	M	69	1	2	2	1	1	2	1	2	2	2	2	2	2	
1	M	74	2	1	1	1	2	2	2	1	1	1	2	2	2	
2	F	59	1	1	1	2	1	2	1	2	1	2	2	1	2	
3	M	63	2	2	2	1	1	1	1	1	2	1	1	2	2	
4	F	63	1	2	1	1	1	1	1	2	1	2	2	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
304	F	56	1	1	1	2	2	2	1	1	2	2	2	2	1	
305	M	70	2	1	1	1	1	2	2	2	2	2	2	1	2	
306	M	58	2	1	1	1	1	1	2	2	2	2	1	1	2	
307	M	67	2	1	2	1	1	2	2	1	2	2	2	1	2	
308	M	62	1	1	1	2	1	2	2	2	2	1	1	2	1	

309 rows × 16 columns

Take a look at the data frame specs

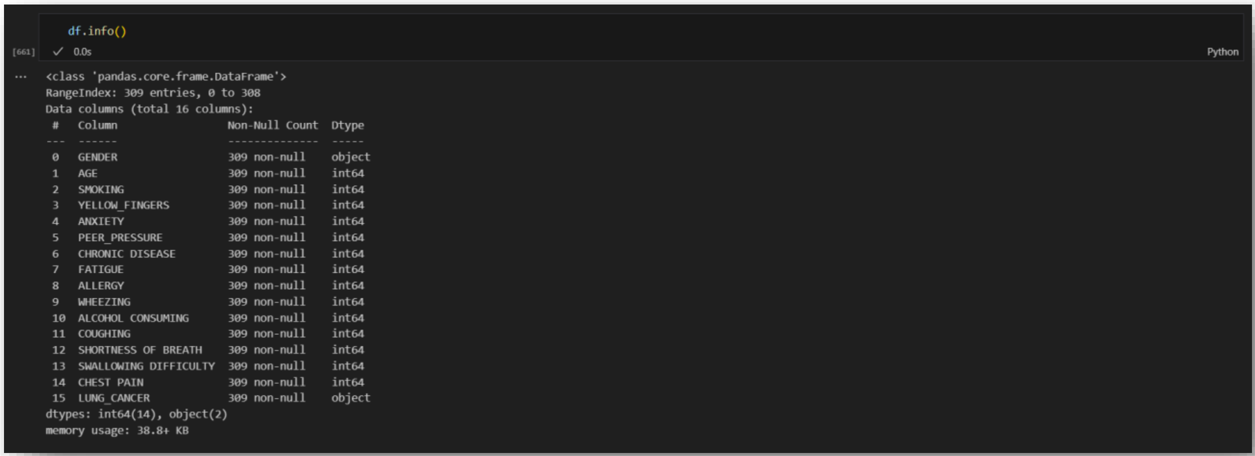
`df.shape` returns a tuple representing the number of rows and columns in a pandas DataFrame, in the format (rows, columns).



```python
df.shape
```

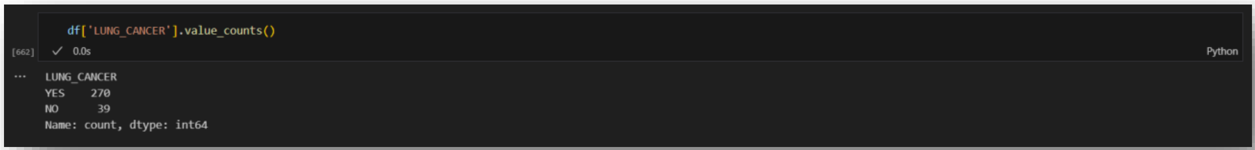
```
(309, 16)
```

`df.info()` provides a quick summary of a pandas DataFrame, showing the index, column names, non-null counts, and data types of each column.



```
df.info()
[661] ✓ 0.0s                                                    Python

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 309 non-null    object
 1   AGE                    309 non-null    int64
 2   SMOKING                309 non-null    int64
 3   YELLOW_FINGERS         309 non-null    int64
 4   ANXIETY                309 non-null    int64
 5   PEER_PRESSURE          309 non-null    int64
 6   CHRONIC DISEASE        309 non-null    int64
 7   FATIGUE                309 non-null    int64
 8   ALLERGY                309 non-null    int64
 9   WHEEZING               309 non-null    int64
 10  ALCOHOL CONSUMING      309 non-null    int64
 11  COUGHING               309 non-null    int64
 12  SHORTNESS OF BREATH    309 non-null    int64
 13  SWALLOWING DIFFICULTY  309 non-null    int64
 14  CHEST PAIN             309 non-null    int64
 15  LUNG_CANCER            309 non-null    object
dtypes: int64(14), object(2)
memory usage: 38.8+ KB
```

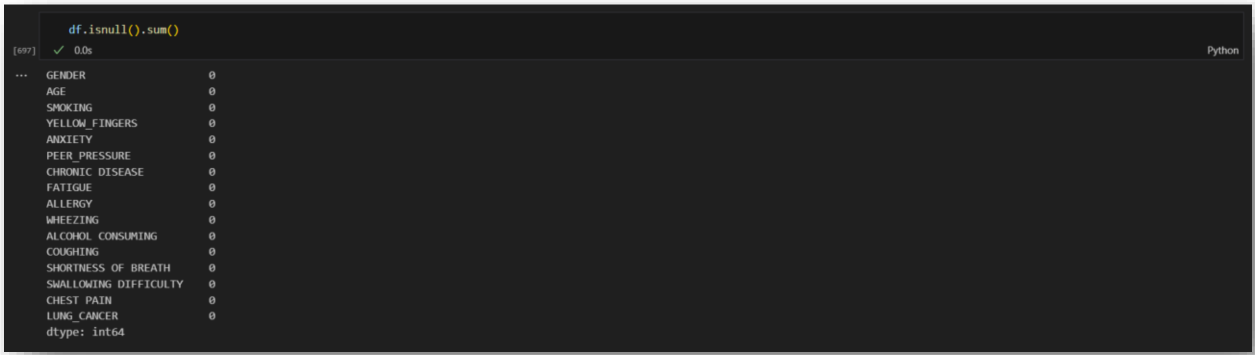
See how many people in the data frame suffer from Lung Cancer and how many don't.



```
df['LUNG_CANCER'].value_counts()
[662] ✓ 0.0s                                                    Python

LUNG_CANCER
YES    270
NO      39
Name: count, dtype: int64
```

As we can see, `270` people in the data frame suffer from lung cancer and `39` people don't have such disease.

We should always check for `null` values in the Data frame because missing data can affect data analysis and machine learning models, leading to inaccurate results or errors during processing (Faruk, 2023).



```
df.isnull().sum()
[697] ✓ 0.0s                                                    Python

GENDER                   0
AGE                      0
SMOKING                  0
YELLOW_FINGERS           0
ANXIETY                  0
PEER_PRESSURE            0
CHRONIC DISEASE          0
FATIGUE                  0
ALLERGY                  0
WHEEZING                 0
ALCOHOL CONSUMING        0
COUGHING                 0
SHORTNESS OF BREATH      0
SWALLOWING DIFFICULTY    0
CHEST PAIN               0
LUNG_CANCER              0
dtype: int64
```

As we can see there is no `null` value in our data frame.

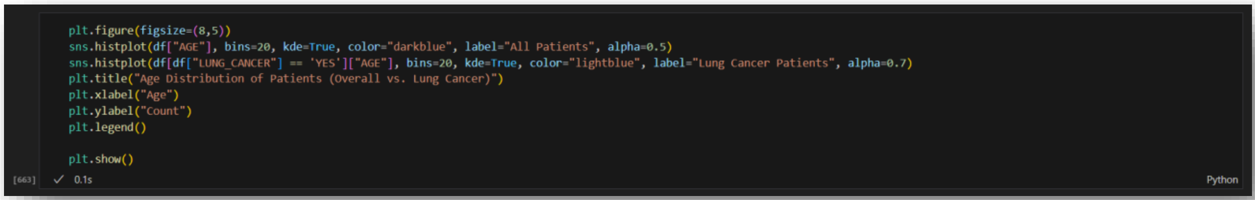
EDA

EDA techniques help show data in pictures, making it easier to see patterns, unusual values, and connections between different pieces of data (Maitra, 2023).

Data Visualization

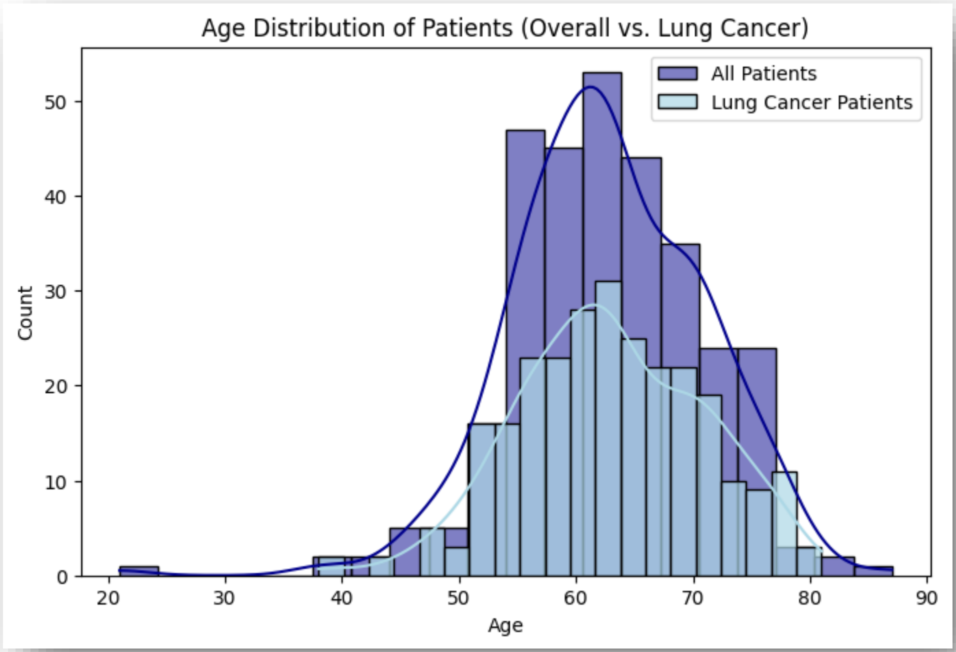
Before cleaning or modifying the data, we take a look at the data frame by visualizing them through human-friendly plots to gain a better vision about what we are going to do (Yau, 2013).

Age Distribution



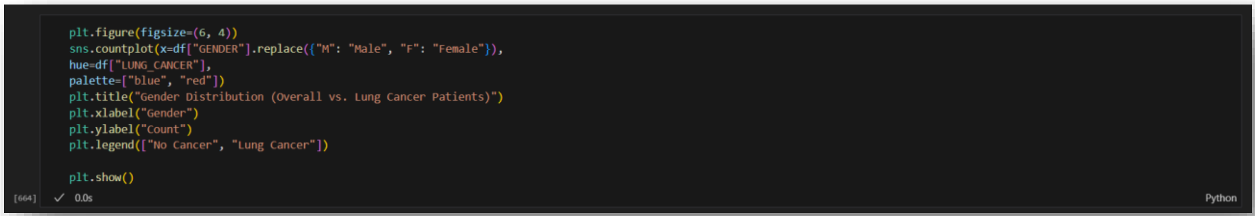
```python
plt.figure(figsize=(8,5))
sns.histplot(df["AGE"], bins=20, kde=True, color="darkblue", label="All Patients", alpha=0.5)
sns.histplot(df[df["LUNG_CANCER"] == 'YES']["AGE"], bins=20, kde=True, color="lightblue", label="Lung Cancer Patients", alpha=0.7)
plt.title("Age Distribution of Patients (Overall vs. Lung Cancer)")
plt.xlabel("Age")
plt.ylabel("Count")
plt.legend()

plt.show()
```

```
[663]  ✓ 0.1s                                                    Python
```

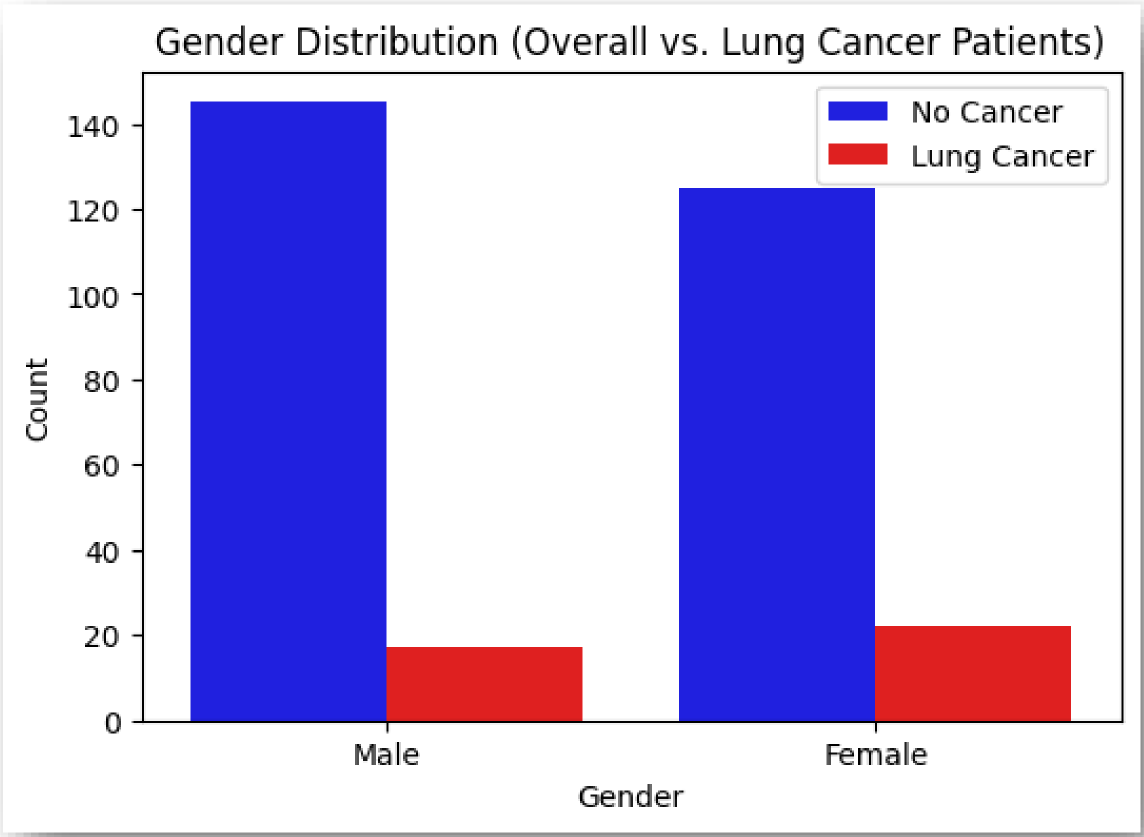


The dark blue bars are for all patients, and the light blue bars are just for lung cancer patients. The curved lines show the overall age trends. Most lung cancer patients are between 55 and 75 years old, with the highest number around 60 years old. This helps see if certain ages are more at risk for lung cancer.

Gender distribution

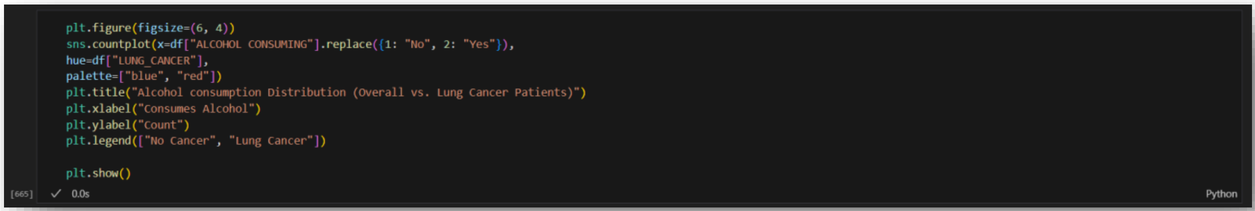


```python
plt.figure(figsize=(6, 4))
sns.countplot(x=df["GENDER"].replace({"M": "Male", "F": "Female"}),
hue=df["LUNG_CANCER"],
palette=["blue", "red"])
plt.title("Gender Distribution (Overall vs. Lung Cancer Patients)")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.legend(["No Cancer", "Lung Cancer"])

plt.show()
```



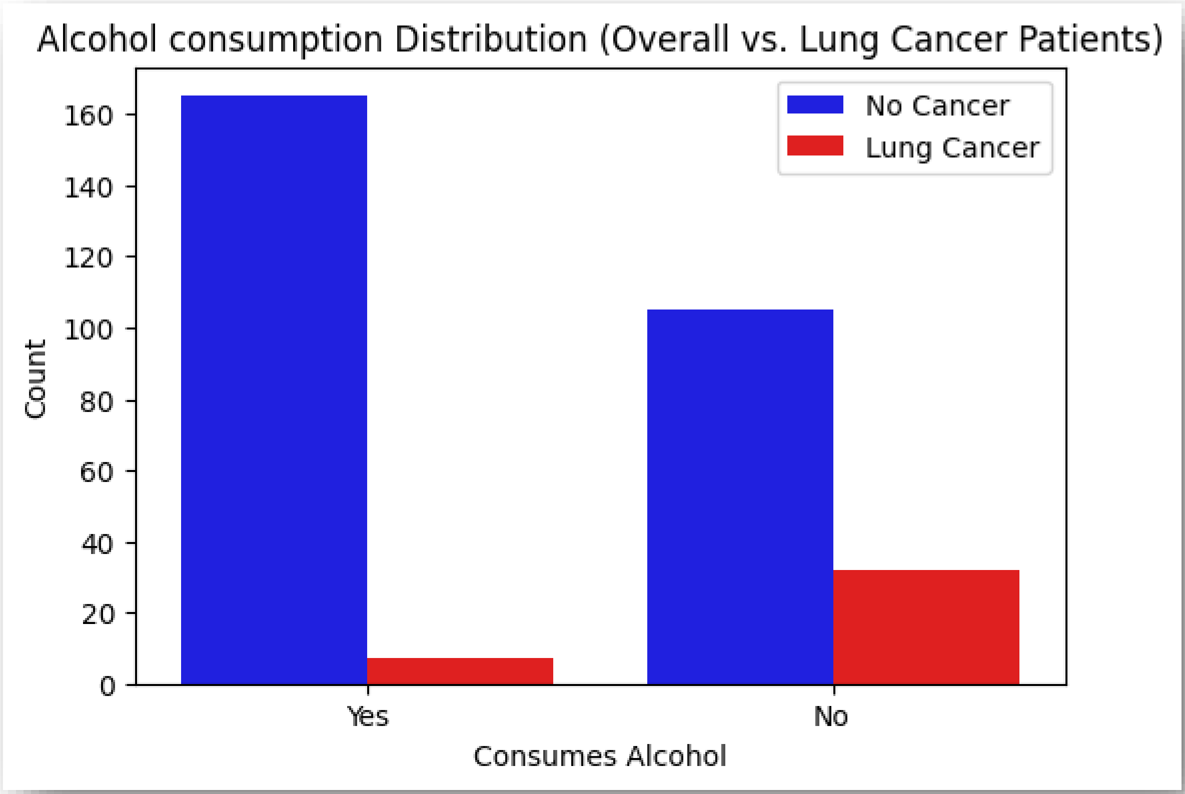
The blue bars are total male and females in the data frame. The red bars are male and females who suffer from the lung cancer.

Distribution based on Alcohol consumption



```python
plt.figure(figsize=(6, 4))
sns.countplot(x=df["ALCOHOL CONSUMING"].replace({1: "No", 2: "Yes"}),
hue=df["LUNG_CANCER"],
palette=["blue", "red"])
plt.title("Alcohol consumption Distribution (Overall vs. Lung Cancer Patients)")
plt.xlabel("Consumes Alcohol")
plt.ylabel("Count")
plt.legend(["No Cancer", "Lung Cancer"])

plt.show()
```

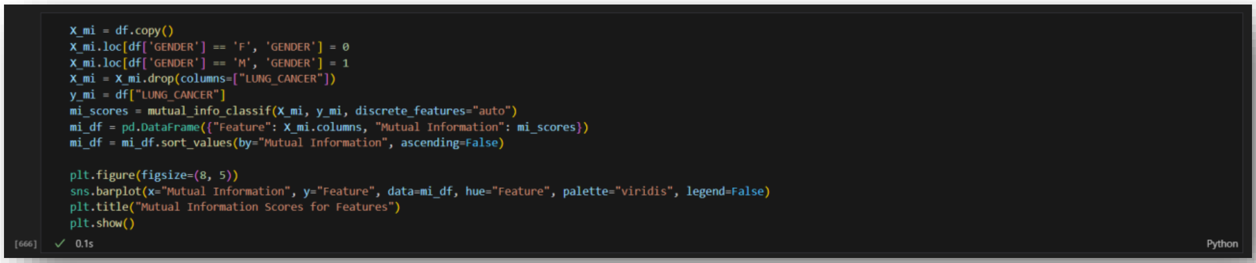
```
[665]  ✓ 0.0s                                                    Python
```



The blue bars are total of people who consume alcohol. The red bars are people who consume alcohol and suffer from lung cancer.

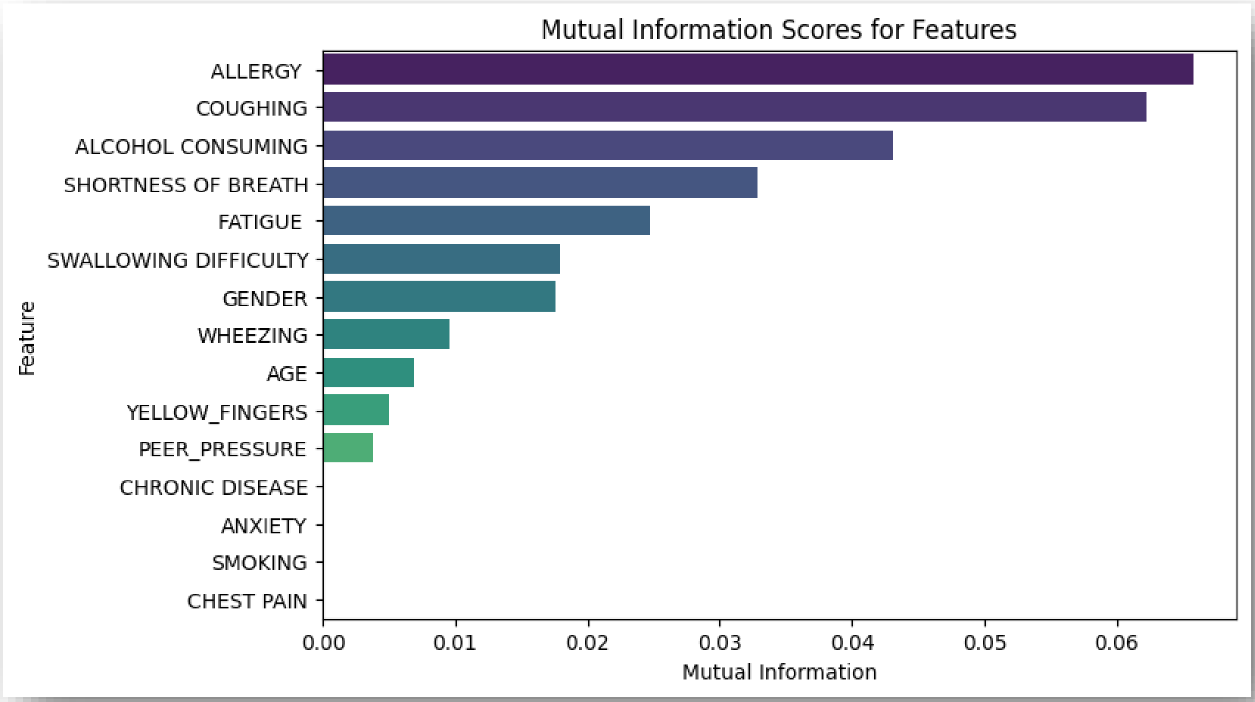
Mutual Information score

The mutual information score measures how much information two things share. Mutual information is commonly used in medical imaging to measure the relationship between variables. It is especially useful in image registration, where it helps align images by maximizing shared information (Maes et al., 1997).



```
X_mi = df.copy()
X_mi.loc[df['GENDER'] == 'F', 'GENDER'] = 0
X_mi.loc[df['GENDER'] == 'M', 'GENDER'] = 1
X_mi = X_mi.drop(columns=["LUNG_CANCER"])
y_mi = df["LUNG_CANCER"]
mi_scores = mutual_info_classif(X_mi, y_mi, discrete_features="auto")
mi_df = pd.DataFrame({"Feature": X_mi.columns, "Mutual Information": mi_scores})
mi_df = mi_df.sort_values(by="Mutual Information", ascending=False)

plt.figure(figsize=(8, 5))
sns.barplot(x="Mutual Information", y="Feature", data=mi_df, hue="Feature", palette="viridis", legend=False)
plt.title("Mutual Information Scores for Features")
plt.show()
```

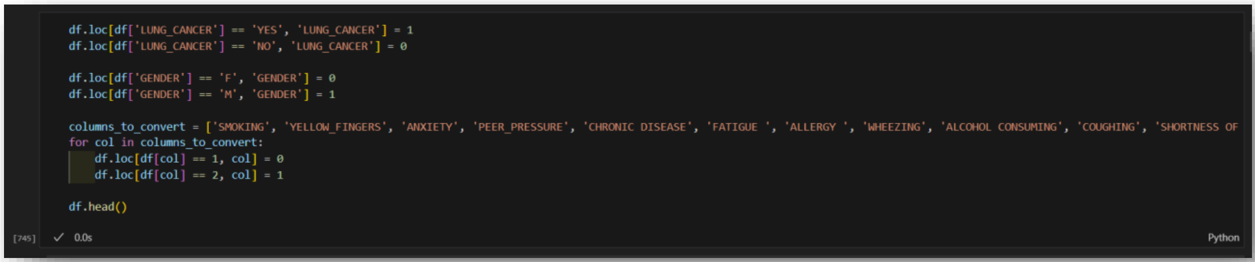
```
[666]  ✓  0.1s                                                        Python
```



This means “Allergy” has the most effect on the outcome and “Chest Pain” has the least effect.

Data Cleaning and Preparation

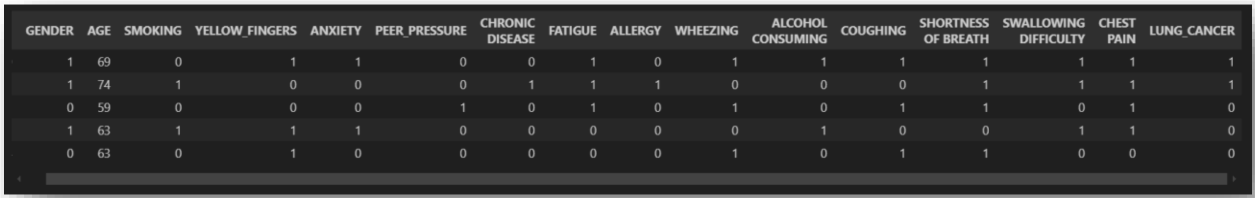
We already checked that we don’t have any null values to be concerned about in our dataset. Instead, we shall make some changes in the dataset to make it more ready for machine learning algorithms. Binary representation is desirable for its memory efficiency (Zhirong Wu, 2015). In our dataset we have “YES/NO”, “1/2” variables that can be converted to simple “0/1” as representatives for “False/True”. Also “Gender” which is defined as “M” for Male and “F” for Female can be mapped to “0” for Female and “1” for Male.



```python
df.loc[df['LUNG_CANCER'] == 'YES', 'LUNG_CANCER'] = 1
df.loc[df['LUNG_CANCER'] == 'NO', 'LUNG_CANCER'] = 0

df.loc[df['GENDER'] == 'F', 'GENDER'] = 0
df.loc[df['GENDER'] == 'M', 'GENDER'] = 1

columns_to_convert = ['SMOKING', 'YELLOW_FINGERS', 'ANXIETY', 'PEER_PRESSURE', 'CHRONIC DISEASE', 'FATIGUE ', 'ALLERGY ', 'WHEEZING', 'ALCOHOL CONSUMING', 'COUGHING', 'SHORTNESS OF
for col in columns_to_convert:
    df.loc[df[col] == 1, col] = 0
    df.loc[df[col] == 2, col] = 1

df.head()
```

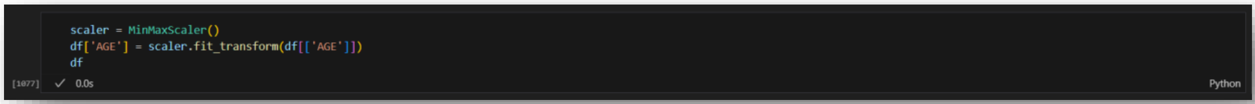
[745] ✓ 0.0s Python

Now, all our binary features are changed to “0/1” for “False/True” or “Female/Male”



GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
1	69	0	1	1	0	0	1	0	1	1	1	1	1	1	1
1	74	1	0	0	0	1	1	1	0	0	0	1	1	1	1
0	59	0	0	0	1	0	1	0	1	0	1	1	0	1	0
1	63	1	1	1	0	0	0	0	0	1	0	0	1	1	0
0	63	0	1	0	0	0	0	0	1	0	1	1	0	0	0

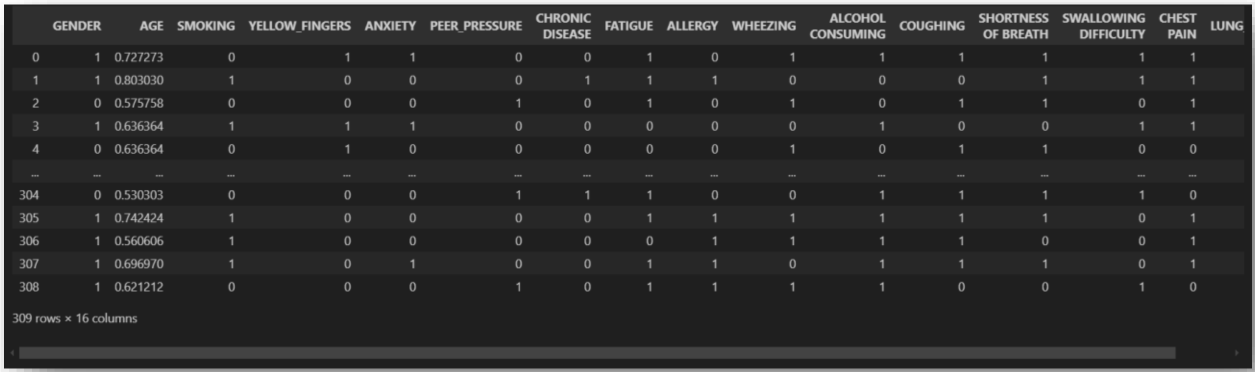
Now we Normalize the “AGE”



```python
scaler = MinMaxScaler()
df['AGE'] = scaler.fit_transform(df[['AGE']])
df
```

[1077] ✓ 0.0s Python

The output will be like this

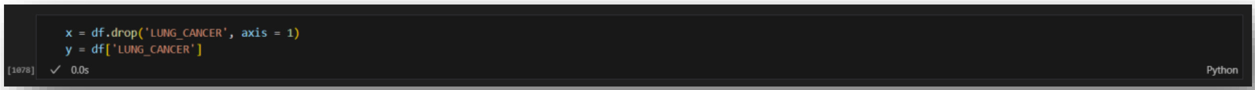


	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG
0	1	0.727273	0	1	1	0	0	1	0	1	1	1	1	1	1	
1	1	0.803030	1	0	0	0	1	1	1	0	0	0	1	1	1	
2	0	0.575758	0	0	0	1	0	1	0	1	0	1	1	0	1	
3	1	0.636364	1	1	1	0	0	0	0	0	1	0	0	1	1	
4	0	0.636364	0	1	0	0	0	0	0	1	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
304	0	0.530303	0	0	0	1	1	1	0	0	1	1	1	1	0	
305	1	0.742424	1	0	0	0	0	1	1	1	1	1	1	0	1	
306	1	0.560606	1	0	0	0	0	0	1	1	1	1	0	0	1	
307	1	0.696970	1	0	1	0	0	1	1	0	1	1	1	0	1	
308	1	0.621212	0	0	0	1	0	1	1	1	1	0	0	1	0	

309 rows × 16 columns

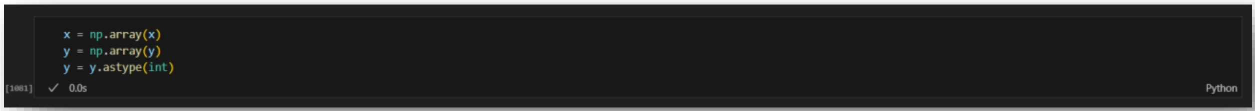
Data Preprocessing

Dividing dataset to X for features and Y for answer



```
x = df.drop('LUNG_CANCER', axis = 1)
y = df['LUNG_CANCER']
[1078]  ✓ 0.0s                                                   Python
```

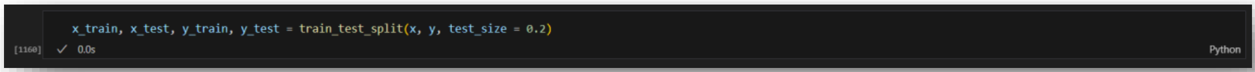
We convert X and Y to NumPy arrays to make them compatible with machine learning libraries like scikit-learn. NumPy arrays are faster and use less memory than lists (Oliphant & Oliphant, 2015). They also allow easy, vectorized operations on data without needing loops.



```
x = np.array(x)
y = np.array(y)
y = y.astype(int)
[1081]  ✓ 0.0s                                                   Python
```

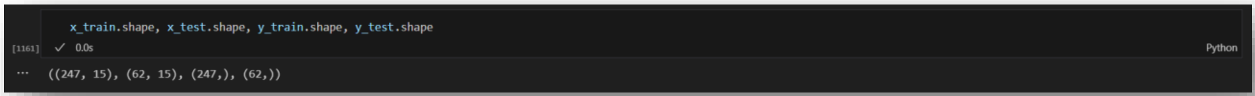
Train/Test split

Now it’s time to split the data into two parts, one for training the AI model and the other for testing the model. Usually it’s split to %80 for training and %20 for testing. The 80/20 or 70/30 ratio for splitting data is used to ensure enough data for training while still having enough to test the model’s performance. These ratios are common because they balance effective training and testing the model on new data (Hastie, Tibshirani, & Friedman, 2009).



```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
[1160]  ✓ 0.0s                                                   Python
```

As we can see, 247 rows are designated for training and 62 rows are designated for testing the model



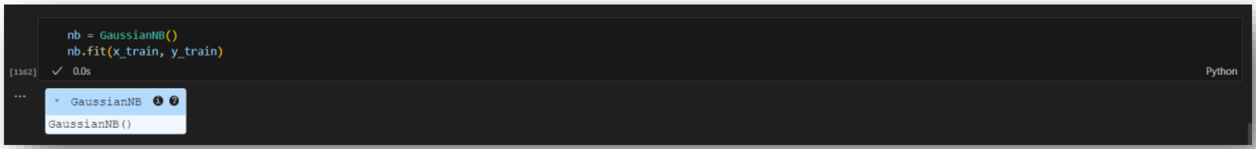
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
[1161]  ✓ 0.0s                                                   Python
...  ((247, 15), (62, 15), (247,), (62,))
```

Machine Learning Algorithms

We will try four different ML algorithms to predict Lung Cancer using our dataset. The algorithms are "Naïve Bayes", "KNN", "Random Forest" and "Neural Networks". After training each model we will use a method called `classification_report()` to evaluate the accuracy of the model. The `classification_report()` gives a summary of a classification model's performance. It shows important details like precision, recall, and f1-score for each class. We will also show the Confusion Matrix for the model. A confusion matrix is a table that shows how well a classification model works. It compares the model's predictions with the real answers, showing where it was right and where it was wrong. To show the confusion matrix we can use `confusion_matrix()` method from `scikit-learn` library which gives a plain text as output, but we prefer to use `ConfusionMatrixDisplay()` class from the same library to show it as a plot.

Naïve Bayes

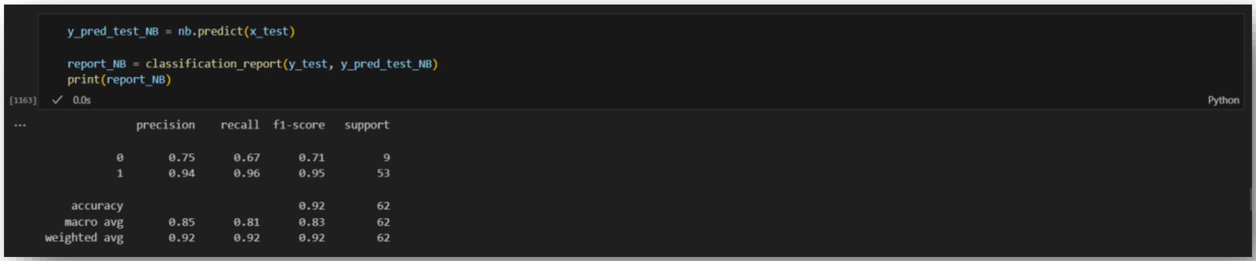
Train the model



```
nb = GaussianNB()
nb.fit(x_train, y_train)
```

```
GaussianNB()
```

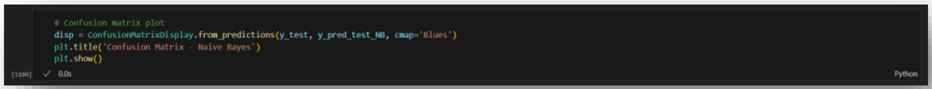
The performance



```
y_pred_test_NB = nb.predict(x_test)

report_NB = classification_report(y_test, y_pred_test_NB)
print(report_NB)
```

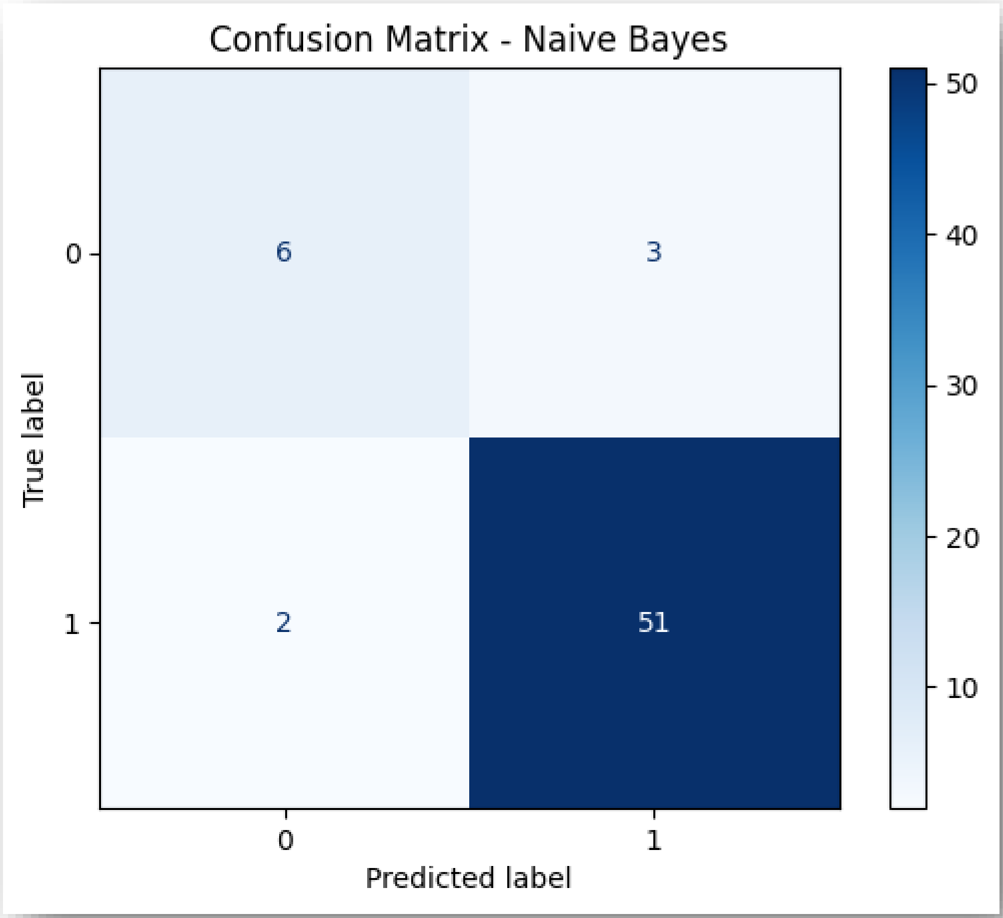
```
              precision    recall  f1-score   support

           0       0.75      0.67      0.71         9
           1       0.94      0.96      0.95        53

    accuracy                           0.92        62
   macro avg       0.85      0.81      0.83        62
weighted avg       0.92      0.92      0.92        62
```

Confusion Matrix



```
# Confusion matrix plot
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred_test_NB, cmap='Blues')
plt.title('Confusion Matrix - Naive Bayes')
plt.show()
```

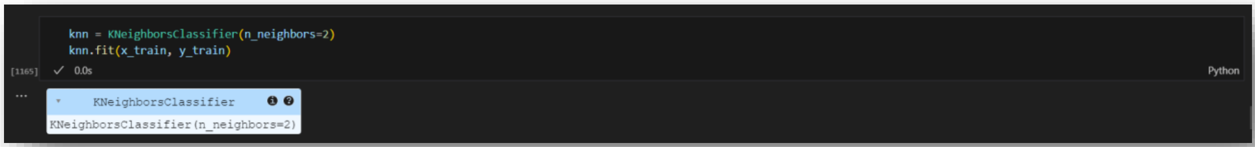
```
[1180]  ✓  0.0s                                                    Python
```



KNN

K-Nearest Neighbors (KNN) is an algorithm that predicts a result by looking at the closest examples in the data and choosing the most common answer among them.

Train the model

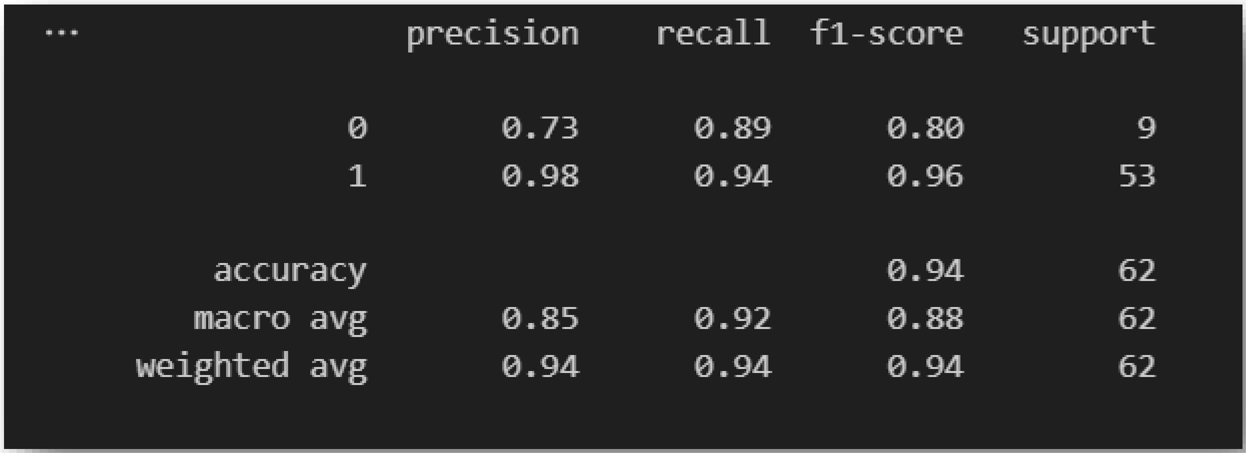


```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train, y_train)
```

```
[1165] ✓ 0.0s                                                Python
```

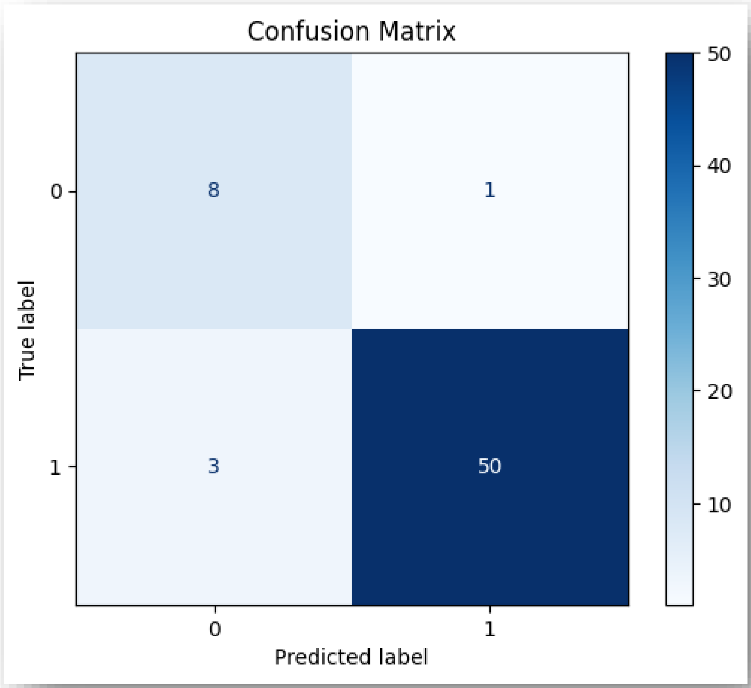
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=2)
```

Performance



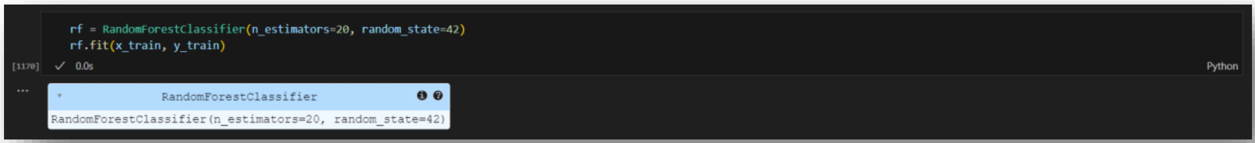
```
              precision    recall  f1-score   support

           0       0.73      0.89      0.80         9
           1       0.98      0.94      0.96        53

    accuracy                           0.94        62
   macro avg       0.85      0.92      0.88        62
weighted avg       0.94      0.94      0.94        62
```

Confusion Matrix



Random Forest

Random Forest is an algorithm that makes predictions by combining the results of many decision trees, improving accuracy and reducing errors.

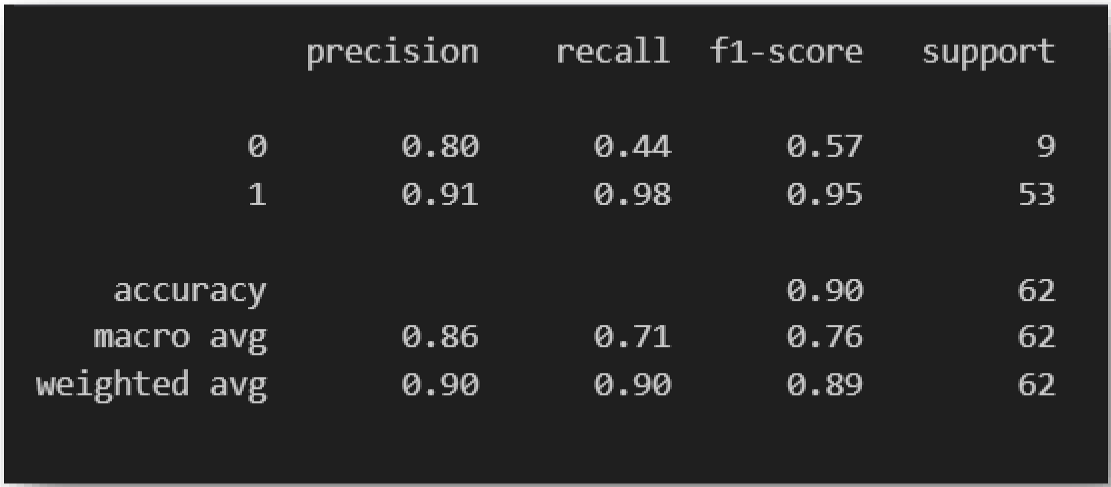


```
rf = RandomForestClassifier(n_estimators=20, random_state=42)
rf.fit(x_train, y_train)
```

```
[1170] ✓ 0.0s                                                    Python
```

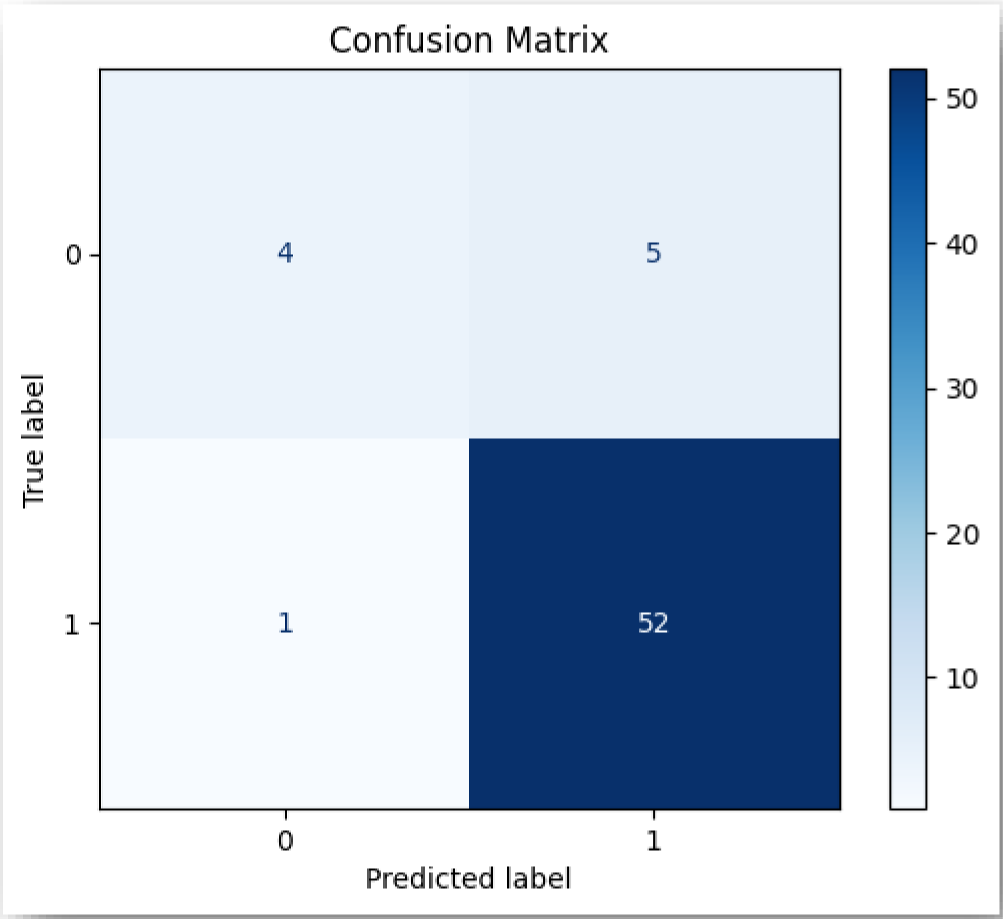
```
RandomForestClassifier(n_estimators=20, random_state=42)
```

Performance



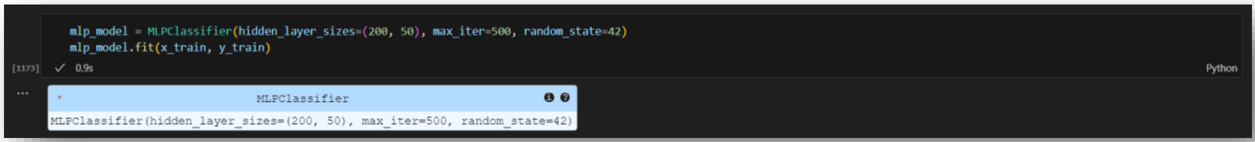
```
              precision    recall  f1-score   support

           0       0.80      0.44      0.57         9
           1       0.91      0.98      0.95        53

    accuracy                           0.90        62
   macro avg       0.86      0.71      0.76        62
weighted avg       0.90      0.90      0.89        62
```

Confusion Matrix



Neural Network

A Neural Network (Multi-Layer Perceptron) is an algorithm that mimics how the human brain works, using layers of connected nodes (neurons) to learn patterns and make predictions.

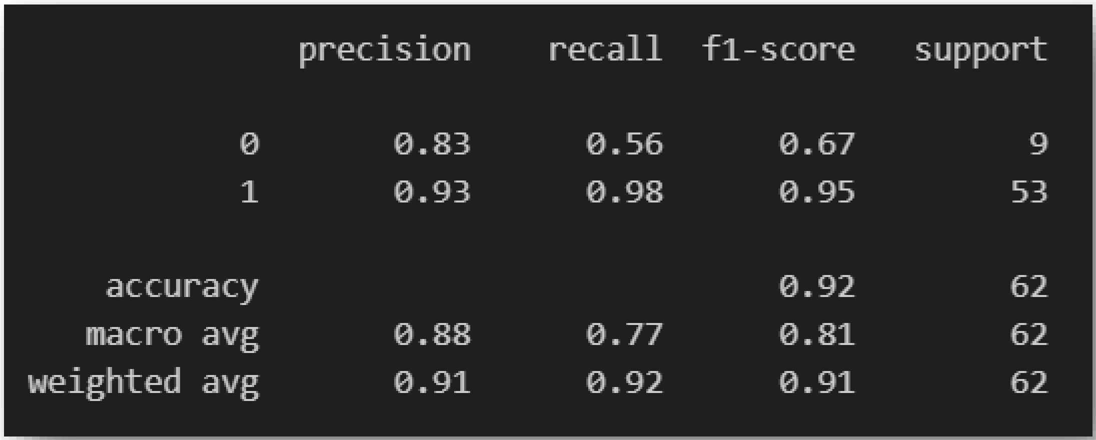


```
mlp_model = MLPClassifier(hidden_layer_sizes=(200, 50), max_iter=500, random_state=42)
mlp_model.fit(x_train, y_train)
```

[1173] ✓ 0.9s Python

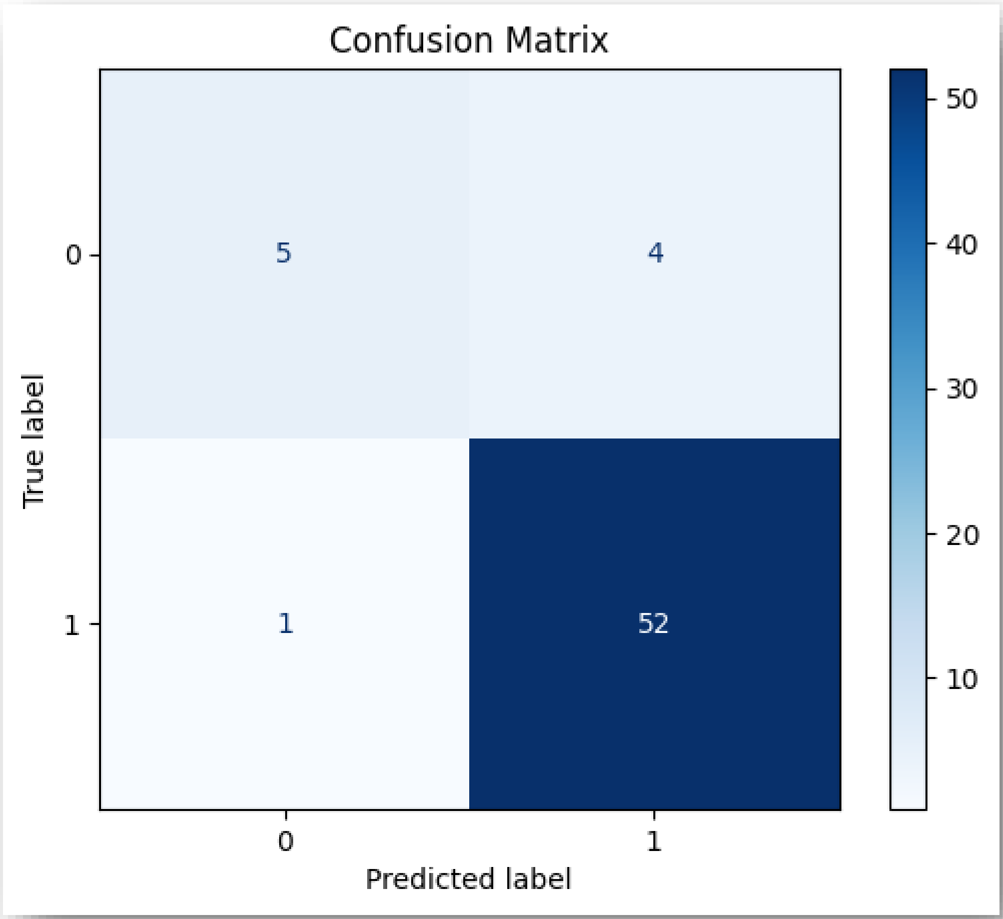
```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(200, 50), max_iter=500, random_state=42)
```

Performance



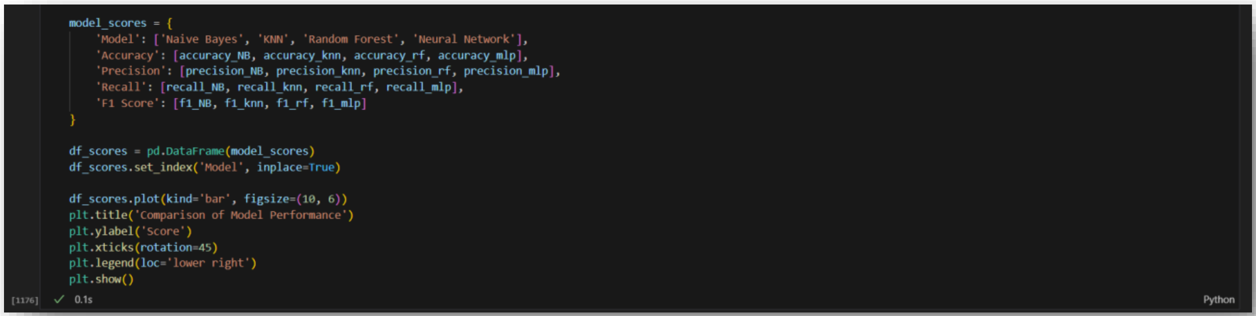
```
              precision    recall  f1-score   support

           0       0.83      0.56      0.67         9
           1       0.93      0.98      0.95        53

    accuracy                           0.92        62
   macro avg       0.88      0.77      0.81        62
weighted avg       0.91      0.92      0.91        62
```

Confusion Matrix



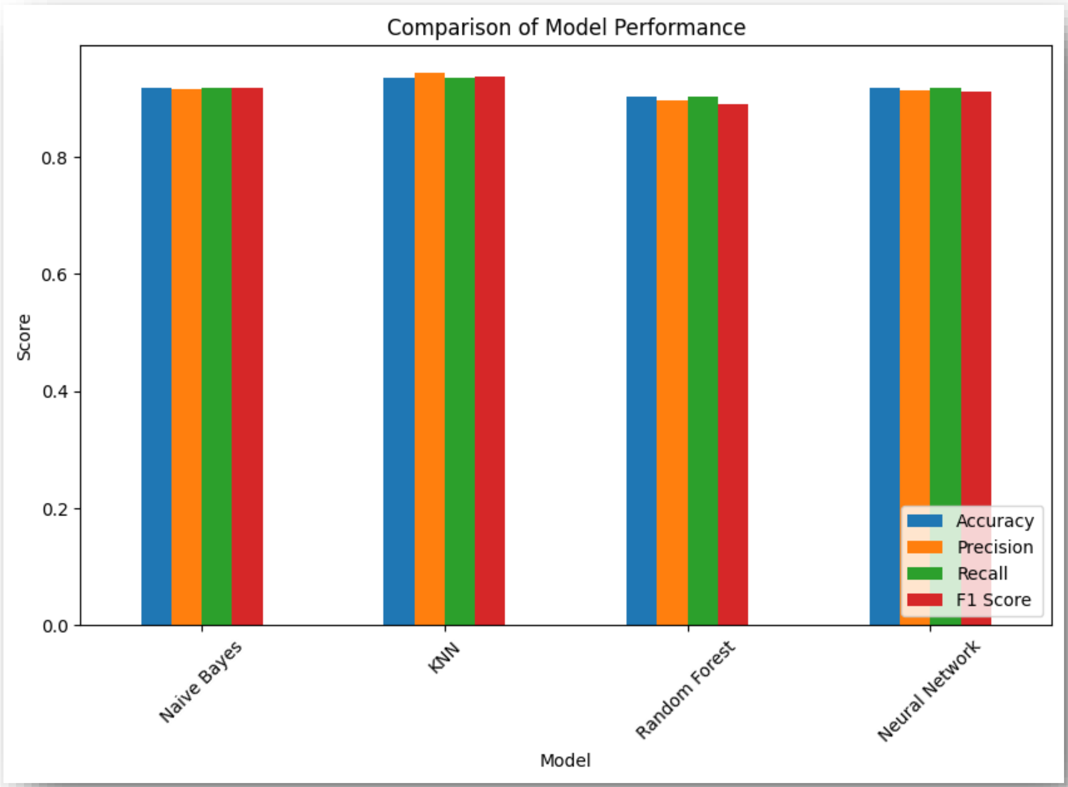
Model Comparison

To evaluate the performance of different machine learning models, I compared several algorithms, including Naive Bayes, K-Nearest Neighbors (KNN), Random Forest, and Neural Network. I used metrics like accuracy, precision, recall and F1-score to measure how well each model performed. By analyzing these results, I was able to identify which model had the best balance of accuracy and reliability for my classification task. This comparison helped me choose the most suitable model for my dataset.



```python
model_scores = {
    'Model': ['Naive Bayes', 'KNN', 'Random Forest', 'Neural Network'],
    'Accuracy': [accuracy_NB, accuracy_knn, accuracy_rf, accuracy_mlp],
    'Precision': [precision_NB, precision_knn, precision_rf, precision_mlp],
    'Recall': [recall_NB, recall_knn, recall_rf, recall_mlp],
    'F1 Score': [f1_NB, f1_knn, f1_rf, f1_mlp]
}

df_scores = pd.DataFrame(model_scores)
df_scores.set_index('Model', inplace=True)

df_scores.plot(kind='bar', figsize=(10, 6))
plt.title('Comparison of Model Performance')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(loc='lower right')
plt.show()
```

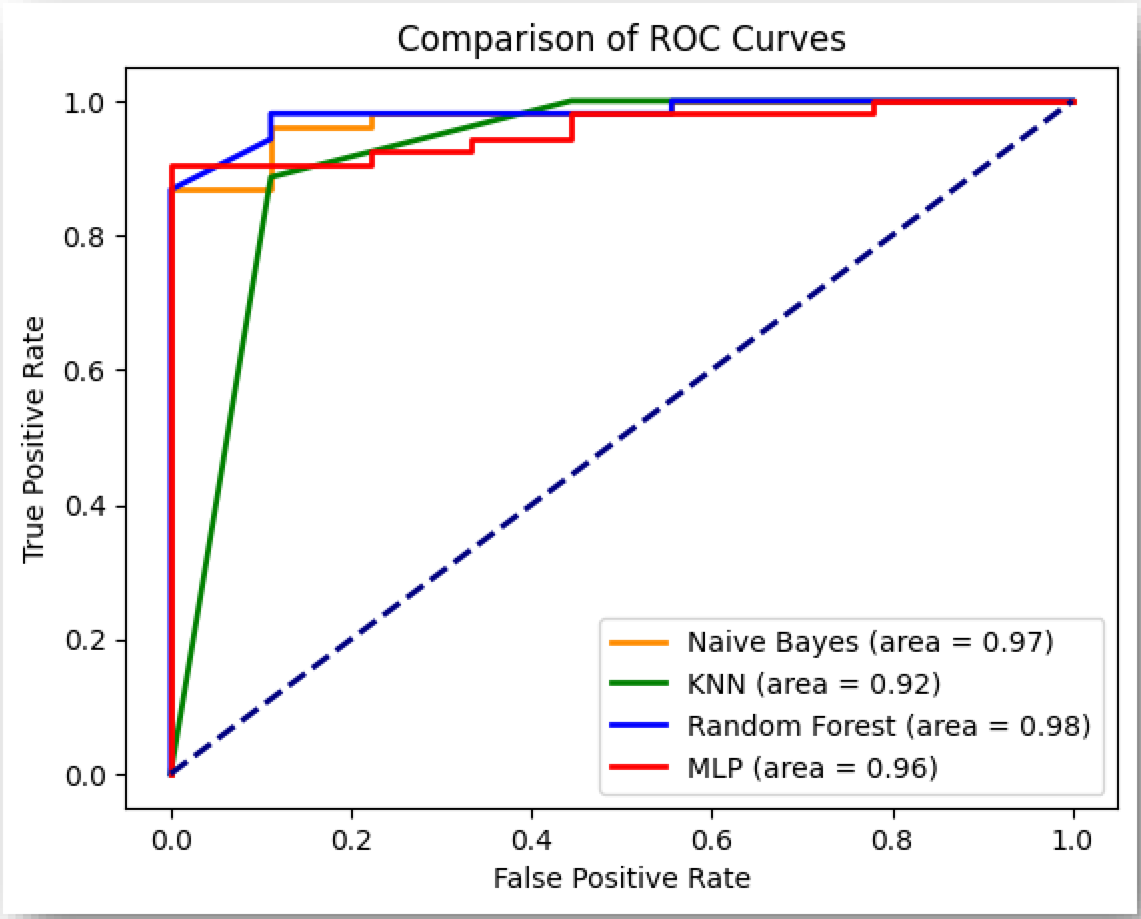
```
[1176]  ✓ 0.1s                                                    Python
```



The results showed that all models had high accuracy, with performance metrics very close to each other. Among them, the KNN model performed slightly better than the others, demonstrating a small advantage in terms of accuracy and evaluation metrics. This suggests that while all models are effective, KNN may be the best choice for this classification task.

ROC Curve

An ROC curve (Receiver Operating Characteristic curve) is a graph that shows how well a classification model can separate the positive and negative classes. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds. The Area Under the Curve (AUC) tells us how good the model is; the closer to 1, the better the model. An AUC of 0.5 means the model is no better than random guessing.



Conclusion

While KNN performed slightly better than other models in accuracy, precision, recall, and F1-score, its ROC AUC score was a bit lower than some of the other models. A model can perform well in accuracy, precision, recall, and F1-score, but still have a lower ROC AUC compared to others (Fawcett, 2006). This means that, although KNN is generally a strong performer, other models might do a better job at distinguishing between classes across different thresholds. Overall, KNN is still a good choice, but the ROC AUC suggests that other models might be more consistent in some situations.

References

Bhat, M. A. (2022). *Lung Cancer*. Retrieved 02/05/2025 from https://www.kaggle.com/datasets/mysarahmadbhat/lung-cancer

Faruk, O. (2023). *Data preprocessing: Identifying and Handling Null Values, High and Low Cardinality, Leakage, and Multicollinearity*. Retrieved 03/05/2025 from https://www.linkedin.com/pulse/data-preprocessing-identifying-handling-null-values-high-omar-faruk/

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861-874. https://doi.org/https://doi.org/10.1016/j.patrec.2005.10.010

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*.

Lokaj, R. (2023). *Early CT Lung Cancer Screening Significantly Improves Survival in Lung Cancer*. Retrieved 03/05/2025 from https://www.cancernetwork.com/view/early-ct-lung-cancer-screening-significantly-improves-survival-in-lung-cancer

Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., & Suetens, P. (1997). Multimodality image registration by maximization of mutual information. *IEEE Trans Med Imaging*, *16*(2), 187-198. https://doi.org/10.1109/42.563664

Maitra, M. (2023). *Importance and Impact of Exploratory Data Analysis in Data Science*. Retrieved 03/05/2025 from https://dzone.com/articles/importance-and-impact-of-exploratory-data-analysis

Oliphant, T. E., & Oliphant, T. E. (2015). *Guide to NumPy*. Continuum Press. https://books.google.co.uk/books?id=g58ljgEACAAJ

Yau, C. (2013). *R Tutorial with Bayesian Statistics Using OpenBUGS*.

Zhirong Wu, D. L., Xiaoou Tang. (2015). Adjustable Bounded Rectifiers: Towards Deep Binary Representations. 11. https://www.mdpi.com/2076-3417/11/17/7825