UNIFIED MENTOR
YOUR SKILL, SUCCESS & JOURNEY

Project Title	Personalized Healthcare Recommendations
Tools	Jupyter Notebook and VS code
Technologies	Machine learning
Domain	Data Analytics
Project Difficulties level	Advanced

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About Dataset

Blood datasets typically encompass a broad array of information related to hematology, blood chemistry, and related health indicators. These datasets often include data points such as blood cell counts, hemoglobin levels, hematocrit, platelet counts, white blood cell differentials, and various blood chemistry parameters such as glucose, cholesterol, and electrolyte levels.

These datasets are invaluable for medical research, clinical diagnostics, and public health initiatives. Researchers and healthcare professionals utilize blood datasets to study hematological disorders, monitor disease progression, assess treatment efficacy, and identify risk factors for various health conditions.

Machine learning techniques are often applied to blood datasets to develop predictive models for diagnosing diseases, predicting patient outcomes, and identifying biomarkers associated with specific health conditions. These models can assist clinicians in making more accurate diagnoses, designing personalized treatment plans, and improving patient care.

Additionally, blood datasets play a crucial role in epidemiological studies and population health research. By analyzing large-scale blood datasets, researchers can identify trends in blood parameters across different demographic groups, assess the prevalence of blood disorders, and evaluate the impact of lifestyle factors and environmental exposures on hematological health.

Overall, blood datasets serve as valuable resources for advancing our understanding of hematology, improving healthcare practices, and promoting better h

Personalized Healthcare Recommendations Machine Learning Project

Project Overview

The Personalized Healthcare Recommendations project aims to develop a machine learning model that provides tailored healthcare recommendations based on individual patient data. This can include recommendations for lifestyle changes, preventive measures, medications, or treatment plans. The goal is to improve patient outcomes by leveraging data-driven insights to offer personalized advice.

Project Steps

1. **Understanding the Problem**
    - The goal is to provide personalized healthcare recommendations to patients based on their health data, medical history, lifestyle, and other relevant factors.
    - Use machine learning techniques to analyze patient data and generate actionable insights.
2. **Dataset Preparation**
    - **Data Sources**: Collect data from various sources such as electronic health records (EHRs), wearable devices, patient surveys, and publicly available health datasets.
    - **Features**: Include demographic information (age, gender), medical history, lifestyle factors (diet, exercise), biometric data (blood pressure, heart rate), lab results, and medication history.
    - **Labels**: Recommendations or health outcomes (if available).
3. **Data Exploration and Visualization**
    - Load and explore the dataset using descriptive statistics and visualization techniques.
    - Use libraries like Pandas for data manipulation and Matplotlib/Seaborn for visualization.
    - Identify patterns, correlations, and distributions in the data.
4. **Data Preprocessing**
    - Handle missing values through imputation or removal.
    - Standardize or normalize continuous features.
    - Encode categorical variables using techniques like one-hot encoding.
    - Split the dataset into training, validation, and testing sets.
5. **Feature Engineering**
    - Create new features that may be useful for prediction, such as health indices or composite scores.
    - Perform feature selection to identify the most relevant features for the model.

6. **Model Selection and Training**
    - Choose appropriate machine learning algorithms based on the problem. Common choices include:
        - Logistic Regression
        - Decision Trees
        - Random Forest
        - Gradient Boosting Machines (e.g., XGBoost)
        - Support Vector Machine (SVM)
        - Neural Networks
    - Train multiple models to find the best-performing one.
7. **Model Evaluation**
    - Evaluate the models using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
    - Use cross-validation to ensure the model generalizes well to unseen data.
    - Visualize model performance using confusion matrices, ROC curves, and other relevant plots.
8. **Recommendation System Implementation**
    - Develop an algorithm to generate personalized recommendations based on the model's predictions.
    - Use techniques like collaborative filtering or content-based filtering if incorporating user feedback or preferences.
    - Ensure recommendations are interpretable and actionable for healthcare professionals and patients.
9. **Deployment (Optional)**
    - Deploy the model and recommendation system using a web framework like Flask or Django.
    - Create a user-friendly interface where healthcare professionals and patients can input data and receive recommendations.
10. **Documentation and Reporting**

- Document the entire process, including data exploration, preprocessing, feature engineering, model training, evaluation, and recommendation generation.
- Create a final report or presentation summarizing the project, results, and insights.

Example: You can get the basic idea how you can create a project from here

Here's a basic example using Python and scikit-learn to build a personalized healthcare recommendation system:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Load the dataset
# Example: Using a mock dataset with patient data
data = pd.read_csv('healthcare_data.csv')

# Explore the dataset
```

```
print(data.head())
print(data.describe())

# Preprocess the data
# Separate features and labels
X = data.drop('recommendation', axis=1)
y = data['recommendation']

# Identify numerical and categorical features
numerical_features = ['age', 'blood_pressure', 'cholesterol', 'heart_rate']
categorical_features = ['gender', 'smoking_status', 'exercise_level']

# Create preprocessing pipelines for numerical and categorical features
numerical_pipeline = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_pipeline = Pipeline(steps=[
    ('encoder', OneHotEncoder(drop='first'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_pipeline, numerical_features),
    ('cat', categorical_pipeline, categorical_features)
])

# Split the 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)
```

```
# Create a pipeline that includes preprocessing and model training
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Train the model
model_pipeline.fit(X_train, y_train)

# Make predictions
y_pred = model_pipeline.predict(X_test)

# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Generate personalized recommendations (mock example)
def generate_recommendations(patient_data):
    prediction = model_pipeline.predict(patient_data)
    # Map predictions to actual recommendations (this will depend on your dataset
and model)
    recommendation_mapping = {0: 'No action needed', 1: 'Regular check-up', 2:
'Lifestyle changes', 3: 'Medication'}
    return recommendation_mapping[prediction[0]]

# Example patient data for recommendation generation
example_patient_data = pd.DataFrame({
    'age': [45],
```

```
    'gender': ['Male'],
    'blood_pressure': [130],
    'cholesterol': [200],
    'heart_rate': [80],
    'smoking_status': ['Non-smoker'],
    'exercise_level': ['Moderate']
})

print(generate_recommendations(example_patient_data))
```

This code demonstrates loading a healthcare dataset, preprocessing the data, training a Random Forest classifier, evaluating the model, and generating personalized recommendations.

Additional Tips

- Incorporate domain expertise to ensure the recommendations are clinically relevant and safe.
- Use explainable AI techniques to make the model's decisions interpretable for healthcare professionals.
- Continuously update the model with new data to improve its accuracy and relevance over time.

Example: You can get the basic idea how you can create a project from here

Sample code and output

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import classification_report,
accuracy_score

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# You can write up to 20GB to the current directory
(/kaggle/working/) that gets preserved as output when you create
a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they
won't be saved outside of the current session
```

```
/kaggle/input/medical-reccomadation-dataset/medical data.csv
```

In [2]:

```
# loadin the datset
data =
pd.read_csv('/kaggle/input/medical-reccomadation-dataset/medica
l data.csv')
```

In [3]:

```
#   Verify Column Names
```

```
print(data.columns)
```

```
Index(['Name', 'DateOfBirth', 'Gender', 'Symptoms', 'Causes',
'Disease',
       'Medicine'],
      dtype='object')
```

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 287 entries, 0 to 286
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Name         241 non-null    object
 1   DateOfBirth  241 non-null    object
 2   Gender       242 non-null    object
 3   Symptoms     247 non-null    object
 4   Causes       245 non-null    object
 5   Disease      249 non-null    object
 6   Medicine     242 non-null    object
dtypes: object(7)
```

```
memory usage: 15.8+ KB
```

In [5]:

```
data.isnull().sum()
```

Out[5]:

```
Name            46
DateOfBirth     46
Gender          45
Symptoms        40
Causes          42
Disease         38
Medicine        45

dtype: int64
```

In [6]:

```
# Check for missing values in each column
missing_values = data.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Name            46
DateOfBirth     46
Gender          45
Symptoms        40
Causes          42
Disease         38
Medicine        45
dtype: int64
```

In [7]:

```
# Handling missing values---i use  Data Imputation
# Numeric Columns (DateOfBirth): I impute missing values using
the mean or median.
data['DateOfBirth'] = pd.to_datetime(data['DateOfBirth'],
errors='coerce')

# Calculate median excluding NaT values (Not a time)
median_date = data['DateOfBirth'].dropna().median()

# Fill missing values with the median date
data['DateOfBirth'].fillna(median_date, inplace=True)

# Categorical Columns (Gender, Symptoms, Causes, Disease,
Medicine):I impute missing values with the mode (most frequent
```

```
value).
categorical_columns = ['Gender', 'Symptoms', 'Causes',
'Disease', 'Medicine']
for column in categorical_columns:
    data[column].fillna(data[column].mode()[0], inplace=True)
```

```
/tmp/ipykernel_19/1203149450.py:3: UserWarning: Parsing dates
in %d-%m-%Y format when dayfirst=False (the default) was
specified. Pass `dayfirst=True` or specify a format to silence
this warning.
  data['DateOfBirth'] = pd.to_datetime(data['DateOfBirth'],
errors='coerce')
```

In [8]:

```
# Check for missing values in each column
missing_values = data.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Name    46
dtype: int64
```

In [9]:

```
# Handle the missing values in column Name by imputation method
data['Name'].fillna('Unknown', inplace=True)
```

In [10]:

```
data.isnull().sum()
```

Out[10]:

```
Name           0
DateOfBirth    0
Gender         0
Symptoms       0
Causes         0
Disease        0
Medicine       0

dtype: int64
```

In [11]:

```
# Check for missing values in each column
missing_values = data.isnull().sum()
```

```
# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Series([], dtype: int64)
```

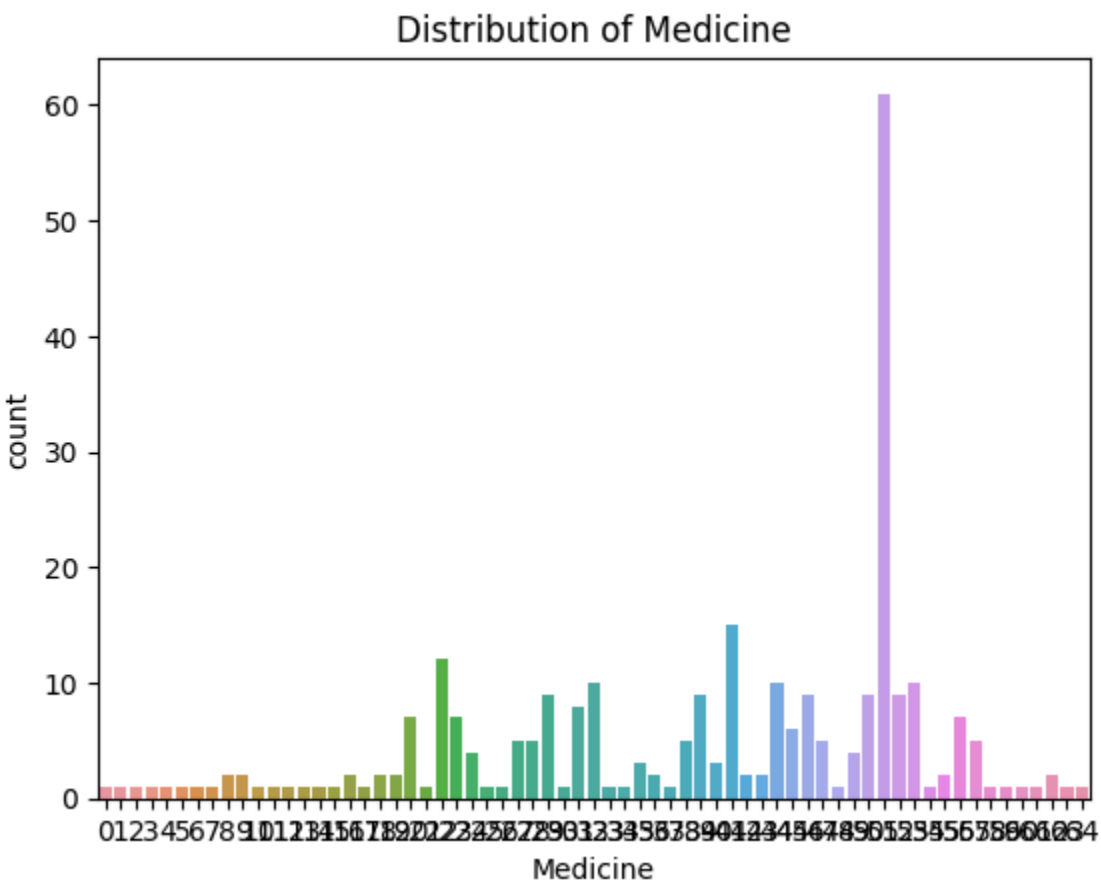
In [12]:

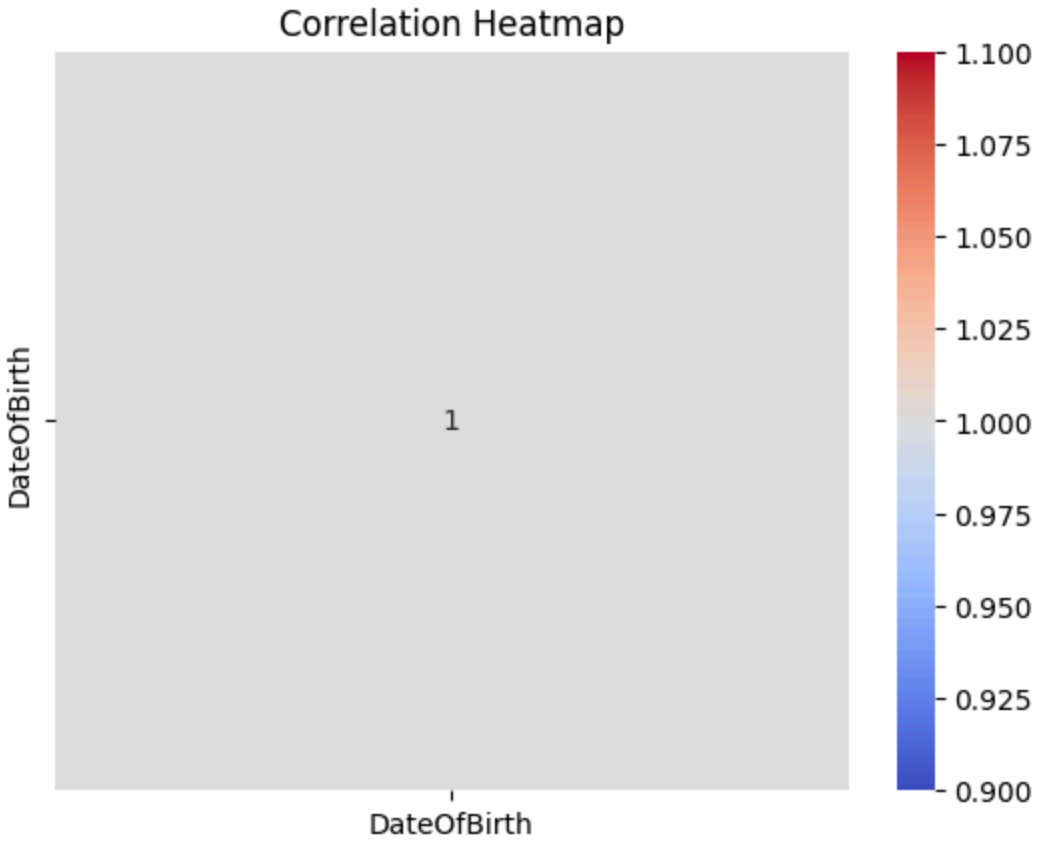
```
# Since most of my data is categorical, i want to encode it into a numerical format suitable for machine learning models.
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Encode categorical variables
label_encoders = {}
for column in data.columns:
    if data[column].dtype == 'object':
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le
```

In [13]:

```
# Exploratory Data Analysis (EDA)
# Class distribution for Medicine
```

```
sns.countplot(data=data, x='Medicine')
plt.title('Distribution of Medicine')
plt.show()

# Correlation heatmap for numerical columns
numeric_columns = ['DateOfBirth']
sns.heatmap(data[numeric_columns].corr(), annot=True,
cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```





In [14]:

```
# Data Processing : We need to encode categorical variables for
the machine learning models.
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
for column in categorical_columns:
    data[column] = encoder.fit_transform(data[column])
```

In [15]:

```
# splitting the datset: We divide the dataset into training and
```

```
testing sets.
from sklearn.model_selection import train_test_split

X = data.drop('Medicine', axis=1)
y = data['Medicine']

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

In [16]:

```
# Drop the DateOfBirth column from the datasets
X_train = X_train.drop('DateOfBirth', axis=1)
X_test = X_test.drop('DateOfBirth', axis=1)

# Fit the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_dt = dt_classifier.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {accuracy_dt}")
```

```
Decision Tree Accuracy: 0.8448275862068966
```

In [17]:

```
# Random classifier
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

y_pred_rf = rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf}")
```

```
Random Forest Accuracy: 0.896551724137931
```

In [18]:

```
# Logistic Regression (for binary classification)
from sklearn.linear_model import LogisticRegression

logreg_classifier = LogisticRegression(random_state=42)
logreg_classifier.fit(X_train, y_train)
```

```
y_pred_logreg = logreg_classifier.predict(X_test)
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f"Logistic Regression Accuracy: {accuracy_logreg}")
```

```
Logistic Regression Accuracy: 0.5344827586206896


/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_l
ogistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data
as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver
options:

https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression
  n_iter_i = _check_optimize_result(
```

In [19]:

```
# model evaluation---lets Evaluate the models using accuracy,
```

```
precision, recall, and F1-score.
from sklearn.metrics import classification_report

print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))

print("\nRandom Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred_logreg))
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           5       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         1
          10       0.00      0.00      0.00         1
          13       0.00      0.00      0.00         1
          15       0.00      0.00      0.00         0
          19       1.00      1.00      1.00         1
          20       1.00      0.50      0.67         2
          21       0.00      0.00      0.00         0
          22       1.00      1.00      1.00         1
```

23	1.00	1.00	1.00	3
27	1.00	1.00	1.00	2
28	0.50	1.00	0.67	1
29	0.50	1.00	0.67	1
31	1.00	1.00	1.00	3
32	1.00	1.00	1.00	4
35	0.00	0.00	0.00	0
38	1.00	1.00	1.00	2
40	1.00	1.00	1.00	1
41	1.00	0.67	0.80	3
44	1.00	0.67	0.80	3
45	1.00	1.00	1.00	2
46	1.00	0.50	0.67	2
47	1.00	1.00	1.00	2
50	1.00	1.00	1.00	2
51	0.94	1.00	0.97	15
53	1.00	1.00	1.00	1
54	0.00	0.00	0.00	1
56	0.00	0.00	0.00	0
57	1.00	0.67	0.80	3
accuracy			0.84	58
macro avg	0.65	0.62	0.62	58
weighted avg	0.90	0.84	0.86	58

Random Forest Classification Report:

	precision	recall	f1-score	support
9	0.00	0.00	0.00	1
10	0.00	0.00	0.00	1
13	0.00	0.00	0.00	1
15	0.00	0.00	0.00	0
19	1.00	1.00	1.00	1
20	1.00	1.00	1.00	2
21	0.00	0.00	0.00	0
22	1.00	1.00	1.00	1
23	1.00	1.00	1.00	3
27	1.00	1.00	1.00	2
28	1.00	1.00	1.00	1
29	0.50	1.00	0.67	1
31	1.00	1.00	1.00	3
32	1.00	1.00	1.00	4
35	0.00	0.00	0.00	0
38	1.00	1.00	1.00	2
39	0.00	0.00	0.00	0
40	1.00	1.00	1.00	1
41	1.00	1.00	1.00	3
44	1.00	1.00	1.00	3
45	1.00	1.00	1.00	2

```
           46       1.00      0.50      0.67         2
           47       1.00      1.00      1.00         2
           50       1.00      1.00      1.00         2
           51       0.94      1.00      0.97        15
           53       1.00      1.00      1.00         1
           54       0.00      0.00      0.00         1
           57       1.00      0.67      0.80         3

    accuracy                           0.90        58
   macro avg       0.69      0.68      0.68        58
weighted avg       0.91      0.90      0.90        58


Logistic Regression Classification Report:
              precision    recall  f1-score   support

           5       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         1
          10       0.00      0.00      0.00         1
          13       0.00      0.00      0.00         1
          19       0.00      0.00      0.00         1
          20       1.00      0.50      0.67         2
          21       0.00      0.00      0.00         0
          22       0.50      1.00      0.67         1
          23       1.00      1.00      1.00         3
```

27	1.00	0.50	0.67	2
28	1.00	1.00	1.00	1
29	0.33	1.00	0.50	1
31	0.00	0.00	0.00	3
32	1.00	1.00	1.00	4
38	0.00	0.00	0.00	2
39	0.00	0.00	0.00	0
40	1.00	1.00	1.00	1
41	1.00	0.33	0.50	3
44	0.14	0.33	0.20	3
45	0.00	0.00	0.00	2
46	0.50	0.50	0.50	2
47	0.50	1.00	0.67	2
50	1.00	1.00	1.00	2
51	0.69	0.60	0.64	15
53	0.50	1.00	0.67	1
54	0.00	0.00	0.00	1
56	0.00	0.00	0.00	0
57	1.00	0.33	0.50	3
accuracy			0.53	58
macro avg	0.43	0.43	0.40	58
weighted avg	0.61	0.53	0.54	58

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
```

```
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
```

```
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
```

```
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classi
fication.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Decision Tree has an accuracy of 0.84 and an f1-score (weighted average) of 0.86.

Random Forest has an accuracy of 0.90 and an f1-score (weighted average) of 0.90.* **

Logistic Regression has an accuracy of 0.53 and an f1-score (weighted average) of 0.54.**

***Random Forest seems to be the best model among the three based on both accuracy and the f1-score, as it has the highest values for both metrics.**

In [20]:

```
# Model Interpretation

import matplotlib.pyplot as plt

# lets get feature importances
```

```
importances = rf_classifier.feature_importances_

# lets sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature
importances
names = [X.columns[i] for i in indices]
```

In [ ]:

Reference