STEP 1 - IMPORT NECESSARY LIBRARIES

```
import pandas as pd
```

STEP 2 - LOAD DATASET

```
# Load the dataset
df = pd.read_csv('breast_cancer_data.csv')
df
```

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | c point |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0 |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0 |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0 |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0 |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **564** | 926424 | M | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 | 0.24390 | 0 |
| **565** | 926682 | M | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 | 0.14400 | 0 |
| **566** | 926954 | M | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 | 0.09251 | 0 |
| **567** | 927241 | M | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 | 0.35140 | 0 |
| **568** | 92751 | B | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 | 0.00000 | 0 |

569 rows × 33 columns

STEP 3 - DATA OVERVIEW

```
# Data Overview: Check basic info of the dataset
print("Data Info:")
print(df.info())

# Describe the dataset
print("\nData Description:")
print(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Check for duplicate rows
print("\nDuplicate Rows:")
print(df.duplicated().sum())

# List column names
print("\nColumn Names:")
print(df.columns)
```

```
Unnamed: 32               569
dtype: int64

Duplicate Rows:
0

Column Names:
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

STEP 4 - DROP UNWANTED COLUMNS

```
# Drop 'Unnamed: 32' and 'id' columns (if they exist)
df = df.drop(columns=['Unnamed: 32', 'id'], errors='ignore')

# Verify the columns after dropping
print(df.columns)
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

STEP 4 - HANDLING NULL VALUES Why Drop Null Values? Impact on Model Performance:

Many machine learning models cannot handle missing data directly (e.g., most models in scikit-learn). Models may throw errors or perform poorly if they encounter null values. Ensuring Accurate Data:

Null values may represent missing or incomplete data, which could distort analysis or lead to biased models. For example, missing target labels or features could create an inaccurate model. Simple and Effective Cleaning:

If the dataset is large and the missing values make up only a small portion of the data, dropping the rows or columns with null values may not significantly reduce the dataset's quality. Data Imbalance:

If null values occur in a specific subset of your data that is already underrepresented (imbalanced), dropping them may result in a better-balanced dataset. However, dropping null values is not always the best solution, especially if:

The dataset is small and missing values are numerous. The missing values are concentrated in specific features critical for analysis or prediction. In cases where a significant portion of data is missing, imputation (filling in the missing values with mean, median, mode, or more sophisticated techniques like k-NN imputation) might be better than dropping them.

```
# Drop rows with missing values
df = df.dropna()

# Verify if there are any remaining missing values
print(df.isnull().sum())
```

```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst         0
compactness_worst        0
concavity_worst          0
concave points_worst     0
symmetry_worst           0
fractal_dimension_worst  0
dtype: int64
```

STEP 5- SPLITING THE DATA

```
# Define features (X) and target (y)
X = df.drop(columns=['diagnosis'])  # 'diagnosis' is the target column
```

```
y = df['diagnosis']

# Check the shape of X and y
print(X.shape)
print(y.shape)
```

```
(569, 30)
(569,)
```

## STEP 6 - LABEL ENCODING AND HANDLING IMBALANCED DATA

```
from imblearn.over_sampling import SMOTE
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

smote = SMOTE()
X_res, y_res = smote.fit_resample(X, y)
# Encode 'diagnosis' column as 0 (Benign) and 1 (Malignant)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Check the distribution of 'diagnosis'
print(df['diagnosis'].value_counts())
```

```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

```
# Check the data types of features to make sure everything is numeric
print(df.dtypes)

# Check for missing or infinite values in the features
print(df.isnull().sum())
print((df == float('inf')).sum())
```

```
diagnosis                  int64
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst         0
```

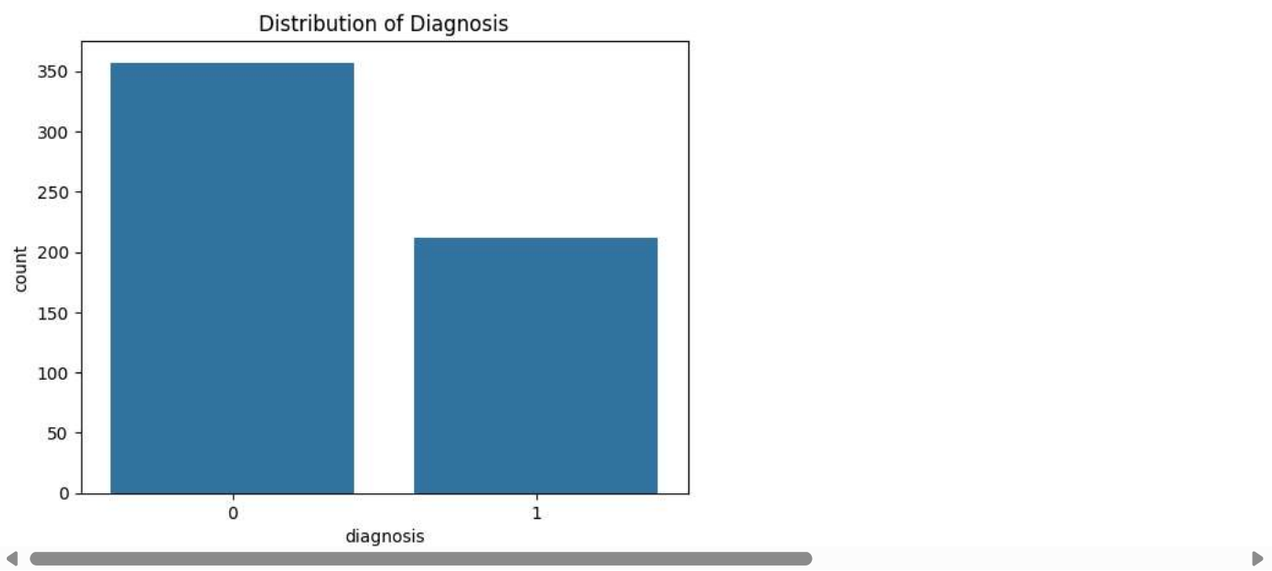
## STEP 7 - CHECKING DATA DISTRIBUTION

```
# Check if the 'diagnosis' column is properly encoded as 0 (Benign) and 1 (Malignant)
print(df['diagnosis'].value_counts())  # Should print counts of 0 and 1

# Check the first few rows to ensure all columns are correct
print(df.head())

# Ensure all feature columns are numeric and there's no NaN or infinite values
print(df.dtypes)  # Should show numeric data types for all feature columns
print(df.isnull().sum())  # Should show zero missing values for all columns
print((df == float('inf')).sum())  # Should show zero infinite values in any columns
```

```
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

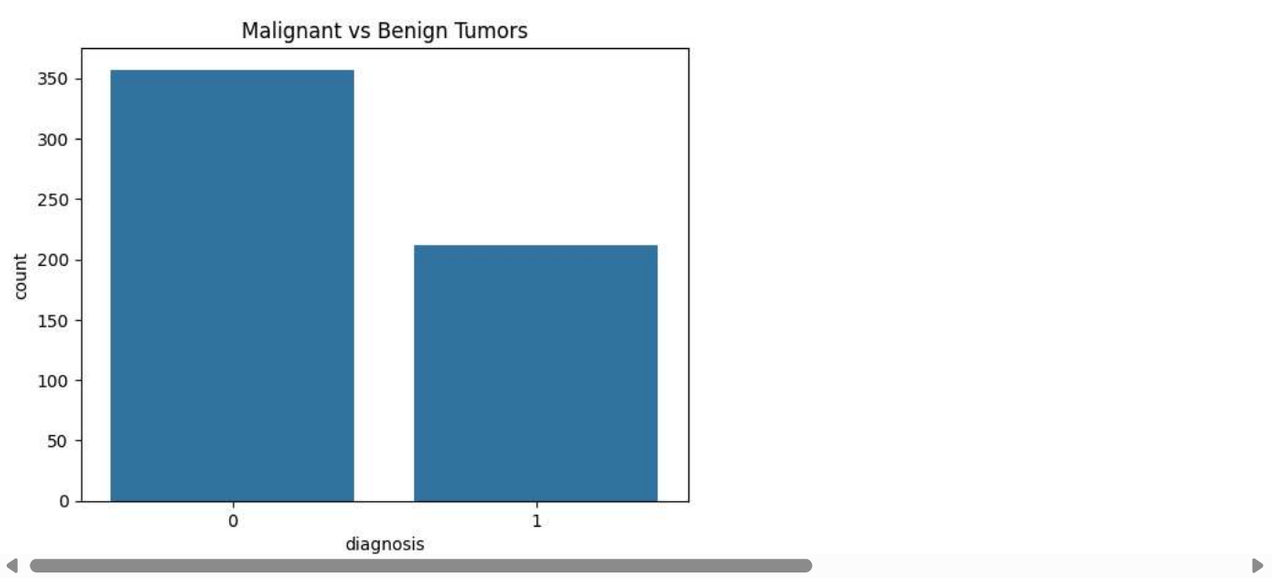
## STEP 8 - DATA DISTRIBUTION

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

# Distribution of the target variable
sns.countplot(x='diagnosis', data=df)
plt.title('Distribution of Diagnosis')
plt.show()
```



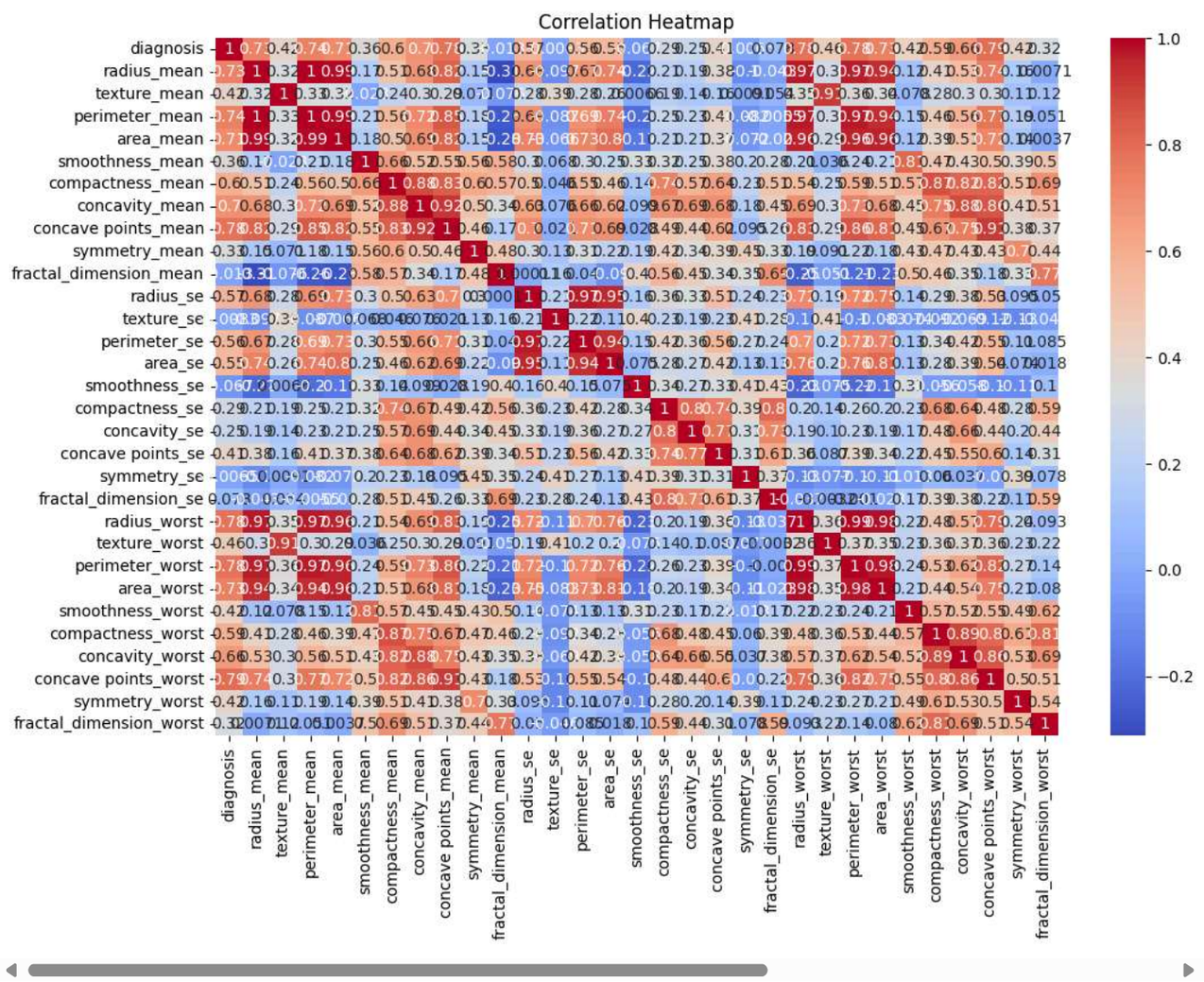

STEP 9 - FRAUD VS NON FRAUD TRANSACTIONS

```
sns.countplot(x='diagnosis', data=df)
plt.title('Malignant vs Benign Tumors')
plt.show()
```



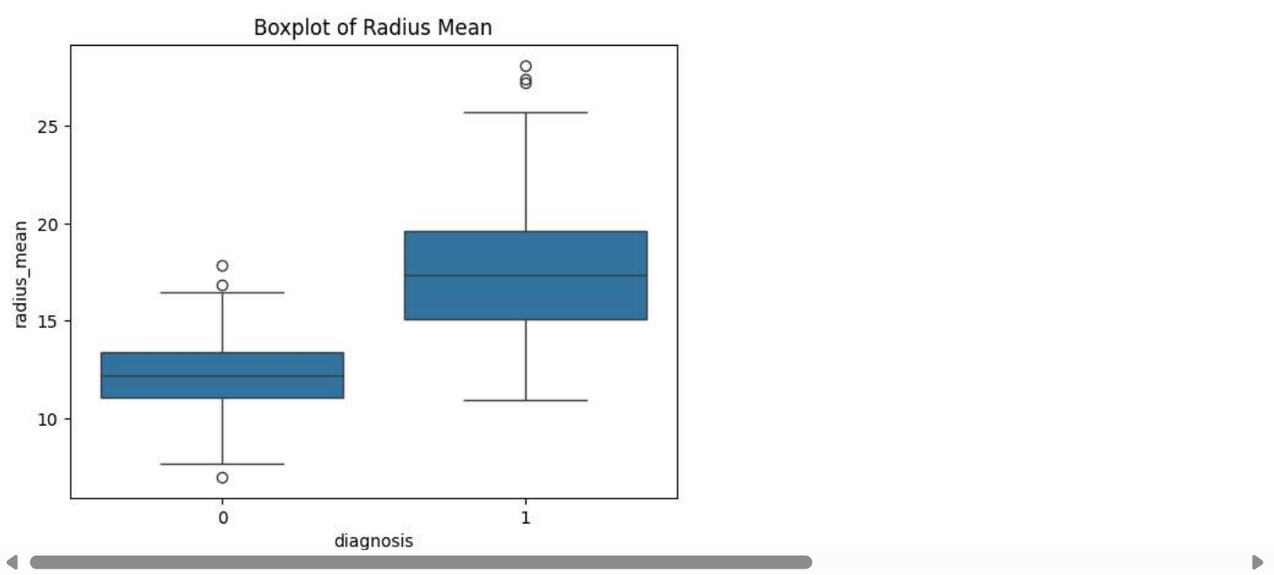

STEP 10 - CORELATION AND PATTERNS

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




STEP 11 - BOXPLOT Visualize the distribution of numerical features like radius_mean, texture_mean, etc.

```
sns.boxplot(x='diagnosis', y='radius_mean', data=df)
plt.title('Boxplot of Radius Mean')
plt.show()
```

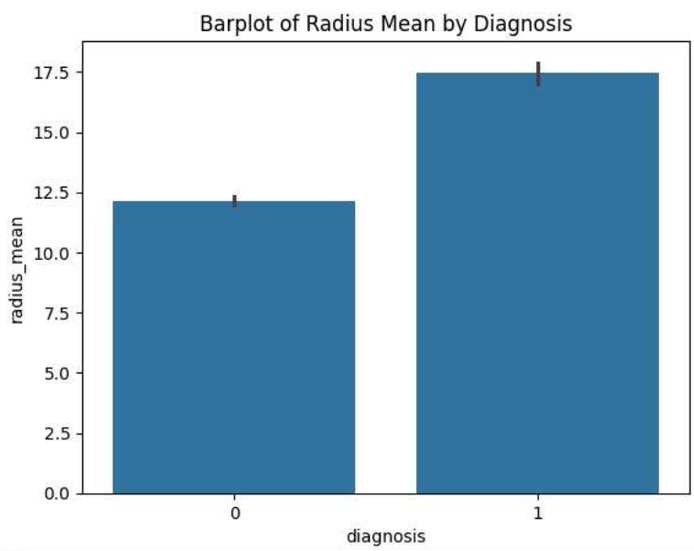



STEP 12 - PAIRPLOT (Visualize pairwise relationships between features to check for patterns or groupings.)

```
sns.pairplot(df, hue='diagnosis')
plt.show()
```

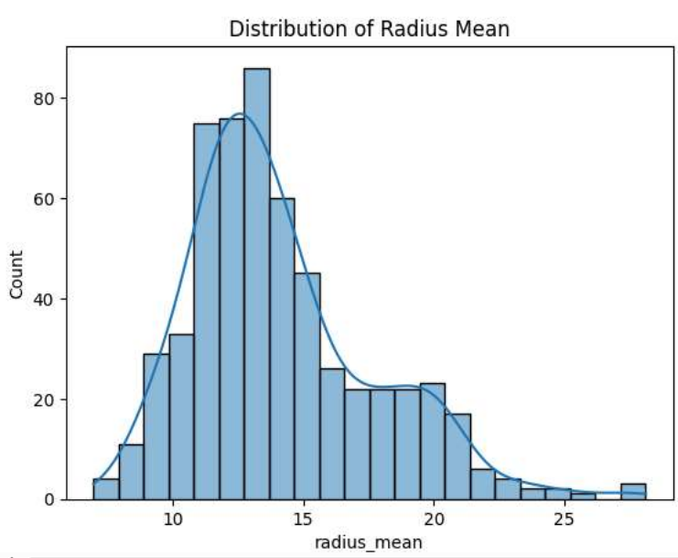
STEP 13 - BARPLOT (Compare average feature values between malignant and benign classes.)

```
sns.barplot(x='diagnosis', y='radius_mean', data=df)
plt.title('Barplot of Radius Mean by Diagnosis')
plt.show()
```



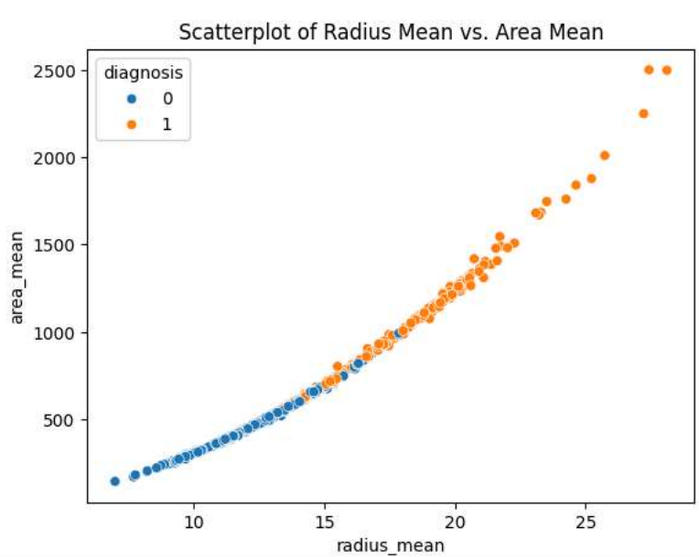

STEP 14 - HISTOGRAM (Explore the distribution of numerical features.)

```
sns.histplot(df['radius_mean'], kde=True)
plt.title('Distribution of Radius Mean')
plt.show()
```




STEP 15 - SCATERRPLOT (Analyze relationships between two continuous features)

```
sns.scatterplot(x='radius_mean', y='area_mean', hue='diagnosis', data=df)
plt.title('Scatterplot of Radius Mean vs. Area Mean')
plt.show()
```




STEP 16 - FEATURE SCALING

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_res)
```

STEP 17 - TRAIN TEST SPLIT

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_res, test_size=0.2, random_state=42)
```

STEP 18 - FITTING LOGISTIC REGRESSION AND CHECKING ITS ACCURACY

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize the model
logreg = LogisticRegression(max_iter=1000)

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

# Predict on the test set
y_pred_logreg = logreg.predict(X_test)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f'Logistic Regression Accuracy: {accuracy_logreg:.4f}')
```

```
Logistic Regression Accuracy: 0.9720
```

STEP 19 - FITTING RANDOM FOREST CLASSIFIER AND CHECKITS ACCURACY

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the model
rf = RandomForestClassifier()

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

# Predict on the test set
y_pred_rf = rf.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf:.4f}')
```

```
Random Forest Accuracy: 0.9580
```

STEP 20 - FITTING SVM AND CHECKING ITS ACCURACY

```
from sklearn.svm import SVC

# Initialize the model
svm = SVC()

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

# Predict on the test set
y_pred_svm = svm.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'Support Vector Machine Accuracy: {accuracy_svm:.4f}')
```

```
Support Vector Machine Accuracy: 0.9650
```

STEP 21 - FITTING KNN AND CHECK ITS ACCURACY

```
from sklearn.neighbors import KNeighborsClassifier

# Initialize the model
knn = KNeighborsClassifier()

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

# Predict on the test set
y_pred_knn = knn.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'K-Nearest Neighbors Accuracy: {accuracy_knn:.4f}')
```

```
K-Nearest Neighbors Accuracy: 0.9510
```

STEP 22 - FITTING DECISION TREE AND CHECKING ITS ACCURACY(ADDITIONAL)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
# Initialize the model
dt = DecisionTreeClassifier()

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

# Predict on the test set
y_pred_dt = dt.predict(X_test)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# Print the accuracy
print(f'Decision Tree Accuracy: {accuracy_dt:.4f}')
```

```
Decision Tree Accuracy: 0.9580
```

## STEP 23 - FITTING GRADIENT BOOSTING CLASSIFIER AND CHECKING ITS ACCURACY(ADDITIONAL)

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Initialize the model
gb = GradientBoostingClassifier()

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

# Predict on the test set
y_pred_gb = gb.predict(X_test)

# Evaluate the model
accuracy_gb = accuracy_score(y_test, y_pred_gb)

# Print the accuracy
print(f'Gradient Boosting Accuracy: {accuracy_gb:.4f}')
```

```
Gradient Boosting Accuracy: 0.9510
```

## STEP 24 - COMPARE MODEL ACCURTACIES

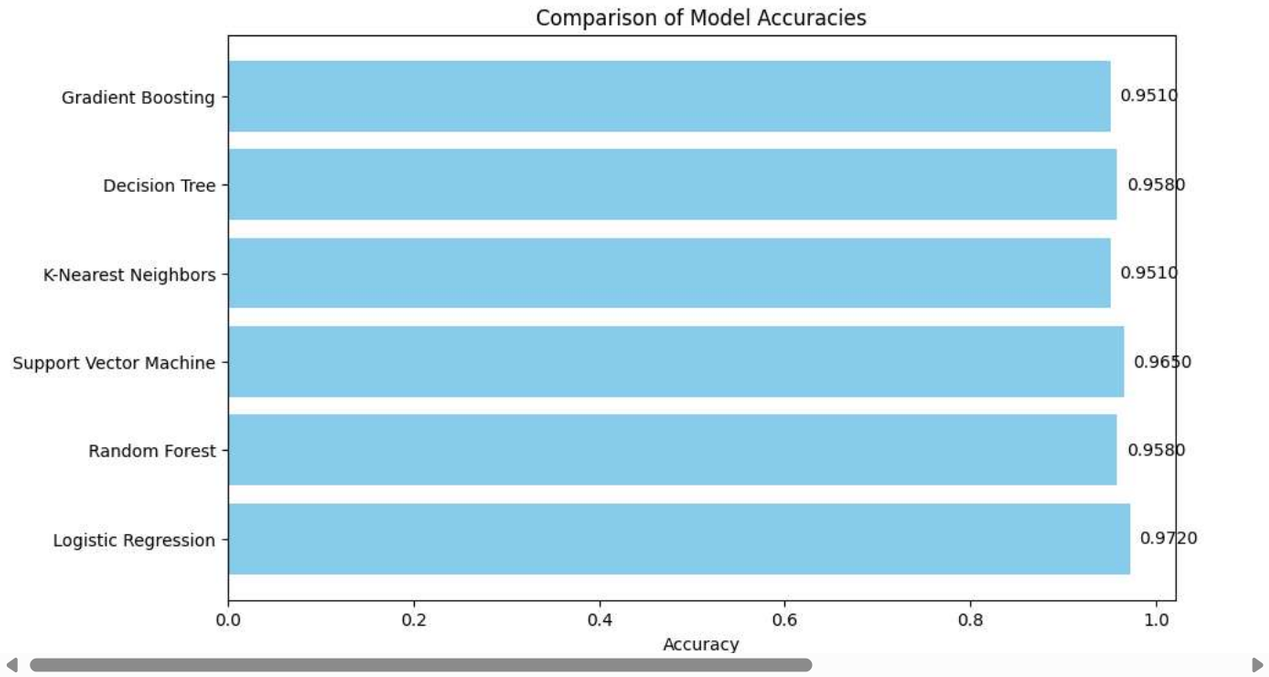
```
# Print out the accuracies
print(f'Logistic Regression Accuracy: {accuracy_logreg:.4f}')
print(f'Random Forest Accuracy: {accuracy_rf:.4f}')
print(f'Support Vector Machine Accuracy: {accuracy_svm:.4f}')
print(f'K-Nearest Neighbors Accuracy: {accuracy_knn:.4f}')
print(f'Decision Tree Accuracy: {accuracy_dt:.4f}')
print(f'Gradient Boosting Accuracy: {accuracy_gb:.4f}')
```

```
Logistic Regression Accuracy: 0.9720
Random Forest Accuracy: 0.9580
Support Vector Machine Accuracy: 0.9650
K-Nearest Neighbors Accuracy: 0.9510
Decision Tree Accuracy: 0.9580
Gradient Boosting Accuracy: 0.9510
```

## STEP 25 - VISUALIZING THE COMPARISION OF ALL MODELS ACCURACIES

```
import matplotlib.pyplot as plt

# Print out all the accuracies
print(f'Logistic Regression Accuracy: {accuracy_logreg:.4f}')
print(f'Random Forest Accuracy: {accuracy_rf:.4f}')
print(f'Support Vector Machine Accuracy: {accuracy_svm:.4f}')
print(f'K-Nearest Neighbors Accuracy: {accuracy_knn:.4f}')
print(f'Decision Tree Accuracy: {accuracy_dt:.4f}')
print(f'Gradient Boosting Accuracy: {accuracy_gb:.4f}')

# Accuracies of the 6 models
accuracies = {
    "Logistic Regression": accuracy_logreg,
    "Random Forest": accuracy_rf,
    "Support Vector Machine": accuracy_svm,
    "K-Nearest Neighbors": accuracy_knn,
    "Decision Tree": accuracy_dt,
    "Gradient Boosting": accuracy_gb
}

# Plotting the accuracies
plt.figure(figsize=(10, 6))
plt.barh(list(accuracies.keys()), list(accuracies.values()), color='skyblue')
plt.xlabel('Accuracy')
plt.title('Comparison of Model Accuracies')

# Displaying the accuracy values on the bars
for i, v in enumerate(accuracies.values()):
    plt.text(v + 0.01, i, f'{v:.4f}', va='center', color='black')

plt.show()
```

```
Logistic Regression Accuracy: 0.9720
Random Forest Accuracy: 0.9580
Support Vector Machine Accuracy: 0.9650
K-Nearest Neighbors Accuracy: 0.9510
Decision Tree Accuracy: 0.9580
Gradient Boosting Accuracy: 0.9510
```




STEP 26 - IMPORTING NECESSARY LIBRARIES FOR EVALUATION

```
from sklearn.metrics import classification_report, f1_score, accuracy_score, confusion_matrix
```

STEP 27 - EVALUATING LOGISTIC REGRESSION MODEL

```
# Logistic Regression Predictions
y_pred_logreg = logreg.predict(X_test)

# Accuracy
print("Logistic Regression:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_logreg):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_logreg, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_logreg, target_names=['B', 'M']))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logreg))
print("\n")
```

```
Logistic Regression:
Accuracy: 0.9790
F1 Score: 0.9799
Classification Report:
              precision    recall  f1-score   support

           B       0.99      0.97      0.98        69
           M       0.97      0.99      0.98        74

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143

Confusion Matrix:
[[67  2]
 [ 1 73]]
```

STEP 28 - EVALUATING RANDOM FOREST CLASSIFIER

```
# Random Forest Predictions
y_pred_rf = rf.predict(X_test)

# Accuracy
print("Random Forest Classifier:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_rf, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=['B', 'M']))
```

```
# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
print("\n")
```

```
Random Forest Classifier:
Accuracy: 0.9650
F1 Score: 0.9655
Classification Report:
              precision    recall  f1-score   support

           B       0.94      0.99      0.96        69
           M       0.99      0.95      0.97        74

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143

Confusion Matrix:
[[68  1]
 [ 4 70]]
```

## STEP 29 - EVALUATING SUPPORT VECTOR MACHINE MODEL

```
# SVM Predictions
y_pred_svm = svm.predict(X_test)

# Accuracy
print("Support Vector Machine (SVM):")
print(f'Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_svm, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=['B', 'M']))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
print("\n")
```

```
Support Vector Machine (SVM):
Accuracy: 0.9650
F1 Score: 0.9664
Classification Report:
              precision    recall  f1-score   support

           B       0.97      0.96      0.96        69
           M       0.96      0.97      0.97        74

    accuracy                           0.97       143
   macro avg       0.97      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

Confusion Matrix:
[[66  3]
 [ 2 72]]
```

## STEP 30 - EVALUATING KNN MODEL

```
# KNN Predictions
y_pred_knn = knn.predict(X_test)

# Accuracy
print("K-Nearest Neighbors (KNN):")
print(f'Accuracy: {accuracy_score(y_test, y_pred_knn):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_knn, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_knn, target_names=['B', 'M']))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn))
print("\n")
```

```
K-Nearest Neighbors (KNN):
Accuracy: 0.9580
F1 Score: 0.9600
Classification Report:
              precision    recall  f1-score   support

           B       0.97      0.94      0.96        69
           M       0.95      0.97      0.96        74

    accuracy                           0.96       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143
```

```
Confusion Matrix:
[[65  4]
 [ 2 72]]
```

STEP 31 - EVALUATONG DECISION TREE CLASSIFIER(ADDITIONAL)

```
from sklearn.metrics import f1_score, classification_report, confusion_matrix

# Decision Tree Predictions
y_pred_dt = dt.predict(X_test)

# Accuracy
print("Decision Tree:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_dt, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=['B', 'M']))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt))
print("\n")
```

```
Decision Tree:
Accuracy: 0.9580
F1 Score: 0.9583
Classification Report:
              precision    recall  f1-score   support

           B       0.93      0.99      0.96        69
           M       0.99      0.93      0.96        74

    accuracy                           0.96       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143

Confusion Matrix:
[[68  1]
 [ 5 69]]
```

STEP 32 - EVALUATING GRADIENT BOOSTING CLASSIFIER (ADDITIONAL)

```
# Gradient Boosting Predictions
y_pred_gb = gb.predict(X_test)

# Accuracy
print("Gradient Boosting:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_gb):.4f}')

# F1 Score
print(f'F1 Score: {f1_score(y_test, y_pred_gb, pos_label="M"):.4f}')

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_gb, target_names=['B', 'M']))

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_gb))
print("\n")
```

```
Gradient Boosting:
Accuracy: 0.9510
F1 Score: 0.9524
Classification Report:
              precision    recall  f1-score   support

           B       0.94      0.96      0.95        69
           M       0.96      0.95      0.95        74

    accuracy                           0.95       143
   macro avg       0.95      0.95      0.95       143
weighted avg       0.95      0.95      0.95       143

Confusion Matrix:
[[66  3]
 [ 4 70]]
```

STEP 33 - Hyperparameter Tuning Example with GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# Example: Tuning Logistic Regression
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
```

```
    'solver': ['liblinear']
}

grid_search_logreg = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)
grid_search_logreg.fit(X_train, y_train)

# Best parameters found
print("Best Parameters for Logistic Regression: ", grid_search_logreg.best_params_)

# Best model evaluation
best_logreg = grid_search_logreg.best_estimator_
y_pred_logreg = best_logreg.predict(X_test)

print("Tuned Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Tuned Logistic Regression F1 Score:", f1_score(y_test, y_pred_logreg, pos_label="M"))
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best Parameters for Logistic Regression:  {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Tuned Logistic Regression Accuracy: 0.972027972027972
Tuned Logistic Regression F1 Score: 0.972972972972973
```

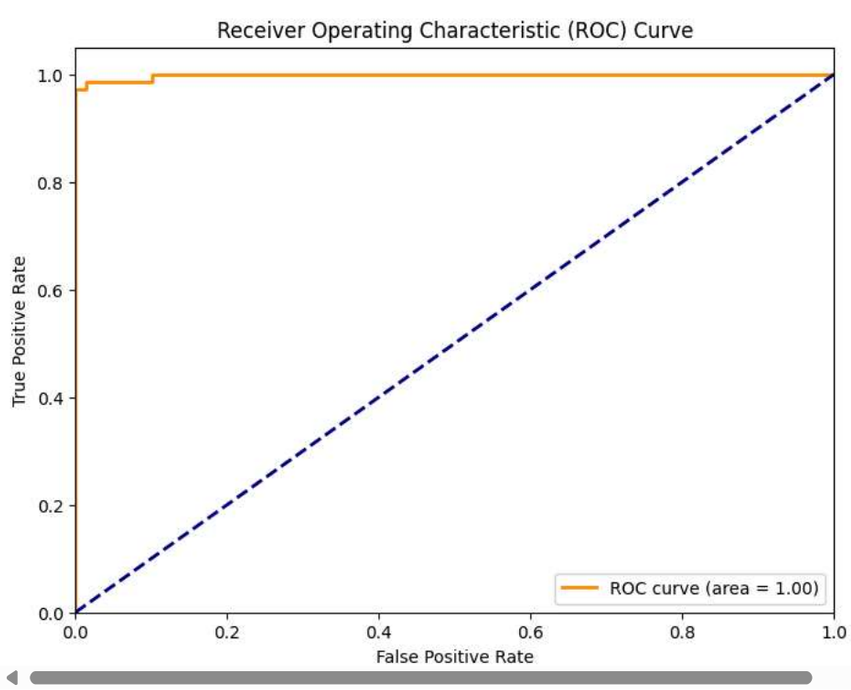
STEP 34 - CROSS VALIDATION

```
from sklearn.model_selection import cross_val_score

# Cross-validation for Logistic Regression
cv_scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring='accuracy')
print("Logistic Regression Cross-Validation Accuracy: ", cv_scores.mean())

# Cross-validation for Random Forest
cv_scores_rf = cross_val_score(rf, X_train, y_train, cv=5, scoring='accuracy')
print("Random Forest Cross-Validation Accuracy: ", cv_scores_rf.mean())
```

```
Logistic Regression Cross-Validation Accuracy:  0.9737452326468345
Random Forest Cross-Validation Accuracy:  0.9737299771167048
```

STEP 35- RECIEVER OPERATOR CHARACTERISTICS (This ROC curve visually demonstrates how well your model is performing in distinguishing fraudulent from non-fraudulent transactions, with the AUC (Area Under the Curve) as a performance metric.)

```
# ROC Curve for Logistic Regression
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1], pos_label="M")
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




STEP 36- PRECISION RECALL CURVE (This curve helps you evaluate the tra
between precision and recall, especially when the classes are imbalanc

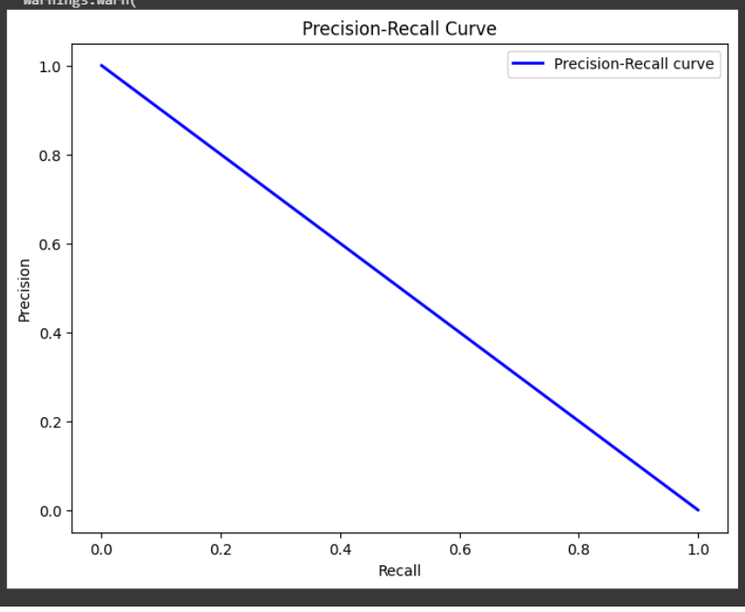
STEP 36- PRECISION RECALL CURVE (This curve helps you evaluate the tradeoff between precision and recall, especially when the classes are

STEP 36- PRECISION RECALL CURVE (This curve helps you evaluate the tradeoff between precision and recall, especially when the classes are imbalanced.)

STEP 36- PRECISION RECALL CURVE (This curve helps you evaluate the tradeoff between precision and recall, especially when the classes are imbalanced.)

```
[ ] from sklearn.metrics import precision_recall_curve

    # Calculate Precision-Recall curve for Logistic Regression
    precision, recall, _ = precision_recall_curve(y_test, logreg.predict_proba(X_test)[:, 1], pos_label=1)

    # Plot Precision-Recall curve
    plt.figure(figsize=(8, 6))
    plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend(loc='best')
    plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:1033: UserWarning: No positive class found in y_true, recall is set to one for all thresholds.
  warnings.warn(
```




Conclusion This project involved a comprehensive approach to detecting fraudulent transactions:

Data Collection and Exploration: We imported the dataset, checked for missing values, duplicates, and irrelevant columns, and explored the data to understand its structure.

Data Preprocessing: We cleaned the data by handling missing values, encoding categorical variables, and removing unnecessary columns.

Exploratory Data Analysis (EDA): We analyzed the data distribution, examined fraud vs. non-fraud patterns, and visualized feature correlations to gain insights into the dataset.

Model Selection & Fitting: We applied and trained multiple machine learning models—Logistic Regression, Random Forest, SVM, and KNN—using the processed data.

Model Evaluation: Models were evaluated using accuracy, F1-score, confusion matrix, and classification report to assess performance, especially on imbalanced data.

Model Comparison: We compared model performance to identify the best-performing algorithm for fraud detection.