BREAST CANCER ANALYSIS MODEL

Abstract:

This project aims to develop a machine learning model for breast cancer prediction, utilizing patient data and outcomes. Starting with an in-depth exploratory analysis, we delved into the dataset's structure, identified patterns, and understood the distribution of breast cancer cases. Through rigorous data preprocessing, we ensured cleanliness and managed outliers, crucial for machine learning readiness. Feature engineering enhanced our predictive capabilities by transforming key attributes. Employing Support Vector Machines, Naive Bayes, Random Forest, and Decision Trees, we crafted various models for binary classification, rigorously evaluating their performance with metrics like accuracy, precision, recall, and F1-score. Our goal is twofold: to provide healthcare professionals with a valuable tool for assessing breast cancer recurrence risk and to empower patients with insights for informed health decisions. Leveraging machine learning potential, this project strives to contribute meaningfully to breast cancer prediction.

INTRODUCTION:

Breast cancer stands as a prevalent and impactful disease affecting countless lives worldwide. Its early detection and precise diagnosis pose significant challenges, highlighting the pivotal role of predictive modeling in healthcare. This project is dedicated to leveraging advanced machine learning (ML) techniques to predict breast cancer based on clinical and diagnostic features. The accurate prediction of breast cancer is crucial for timely intervention and effective treatment planning. Current medical practices heavily rely on exhaustive assessments, including physical examinations, imaging tests, and tissue sampling. However, ML models offer a promising avenue to complement these practices by harnessing quantitative data for diagnostic aid. The significance of this project is rooted in its potential to revolutionize diagnostic procedures. Accurate prediction models have the potential to swiftly identify potential malignancies, enabling proactive treatment strategies. These tools not only enhance the diagnostic process but also significantly contribute to improving patient outcomes and survival rates. The primary objective of this project is to harness the power of machine learning for the development of robust predictive models in breast cancer detection. By leveraging a comprehensive dataset encompassing various patient metrics, the project aims to create accurate and reliable tools for predicting breast cancer, ultimately facilitating improved patient care and outcomes.

SCOPE OF ANALYSIS:

The analysis centers on a comprehensive dataset containing clinical measurements and features associated with breast cancer cases. The dataset includes attributes such as radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension, among others. A thorough examination of the dataset's features and their correlation with the diagnosis forms the core of this analysis. Feature selection and engineering techniques are employed to determine the most influential factors contributing to breast cancer prediction. The analysis's scope may be limited by the dataset's size, quality, or inherent biases. Additionally, while ML models offer powerful prediction capabilities, they are subject to interpretability challenges, which could impact their adoption in clinical settings.

TOOLS AND ML TECHNIQUES USED:

Tools:

The project employs a suite of tools and libraries including:

- Pandas and NumPy for data manipulation
- Scikit-learn for ML model implementation
- Matplotlib and Seaborn for data visualization
- Keras and TensorFlow for neural network development

ML Techniques:

- **Exploratory data analysis (EDA):**
  visualization techniques were employed to understand feature distributions, identify outliers, and ascertain correlations among variables, enabling informed preprocessing steps and guiding subsequent model development for breast cancer prediction.

- **Linear Regression**:
  Utilized for understanding the relationship between specific features (e.g., radius_mean) and target variables (e.g., area_mean).

- **Logistic Regression:**
  Employed for binary classification of breast cancer diagnosis, providing insights into accuracy and classification reports.

- **Random Forest**:
  Utilized for its ensemble learning capability and feature importance analysis.

- **Support Vector Machines (SVM):**
  Applied for classification tasks due to its effectiveness in handling complex datasets.

- **Artificial Neural Networks (ANN) and K-Nearest Neighbors (KNN):**
  Leveraged for their adaptability in capturing intricate patterns within the dataset.

CODE:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix,
classification_report
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
# Load the dataset
data = pd.read_csv("D:\\SEM 5\\ML\\project\\data.csv")
print(data.head())
print(data.describe())
print(data.info())
# Missing value checking
missing_val = data.isnull().sum()
print("Missing values:", missing_val)
# Numerical columns alone
numeric_columns = data.select_dtypes(include=[np.number]).columns.tolist()
# Outlier detection and removal
def remove_outliers(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df
# Apply outlier removal function to the dataset
data_without_outliers = remove_outliers(data, numeric_columns)
# Display the updated dataset after outlier removal
```

```
print(data_without_outliers.head())
# ------------------------ SIMPLE LINEAR REGRESSION ------------------------
X = data[['radius_mean']]      # Independent variable (X)
y = data['area_mean']          # Dependent variable (y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # Split the data
into training and test sets
model = LinearRegression()  # Model building
model.fit(X_train, y_train)
predictions = model.predict(X_test)  # Prediction test
# Printing the model evaluation metrics
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
print('Mean squared error (MSE): %.2f', mean_squared_error(y_test, predictions))
print('Coefficient of determination (R^2): %.2f', r2_score(y_test, predictions))
# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', linewidth=2, label='Regression Line')
plt.title('Simple Linear Regression')
plt.xlabel('Radius Mean')
plt.ylabel('Area Mean')
plt.legend()
plt.grid(True)
plt.show()
# ------------------------ LOGISTIC REGRESSION ------------------------
# Assuming 'diagnosis' is the target variable and other columns are predictors
X = data.drop(['id', 'diagnosis'], axis=1)  # Exclude 'id' column and 'diagnosis' as predictors
y = data['diagnosis']  # Target variable
# Split the data into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create a Logistic Regression model
model = LogisticRegression()
# Train the model using the training sets
model.fit(X_train, y_train)
# Make predictions using the testing set
predictions = model.predict(X_test)
# Model evaluation
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, predictions))
print("\nClassification Report:")
print(classification_report(y_test, predictions))
# Confusion Matrix plot
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show(
# ------------------------ RANDOM FOREST ------------------------s
# Create a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100)  # You can adjust the number of estimators
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
# Model evaluation for Random Forest
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")
print("\nRandom Forest Classification Report:")
print(classification_report(y_test, rf_predictions))
# Feature importances plot for Random Forest
feature_importance = rf_model.feature_importances_
sorted_indices = np.argsort(feature_importance)[::-1]
sorted_importance = feature_importance[sorted_indices]
sorted_columns = X.columns[sorted_indices]
plt.figure(figsize=(10, 6))
sns.barplot(x=sorted_importance, y=sorted_columns, palette='viridis')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Random Forest Feature Importances')
plt.show()
# Get the first tree from the Random Forest model
first_tree = rf_model.estimators_[0]  # Change the index to view different trees
plt.figure(figsize=(12, 8))
plot_tree(first_tree, feature_names=X.columns, filled=True, rounded=True)
plt.title('Random Forest Decision Tree Visualization')
plt.show()
# ------------------------ SVM ------------------------
# Create an SVM Classifier
svm_model = SVC(kernel='linear')  # You can choose different kernels like 'rbf', 'poly', etc.
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
# Model evaluation for SVM
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy:.2f}")
print("\nSVM Classification Report:")
print(classification_report(y_test, svm_predictions))
#---- Accuracy comparison graph
models = ['SVM', 'Random Forest']
```

```
accuracies = [svm_accuracy, rf_accuracy]
plt.figure(figsize=(8, 6))
plt.bar(models, accuracies, color=['orange', 'green'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim(0.85, 1.0)  # Adjust the y-axis limits for better visualization
plt.show()
```

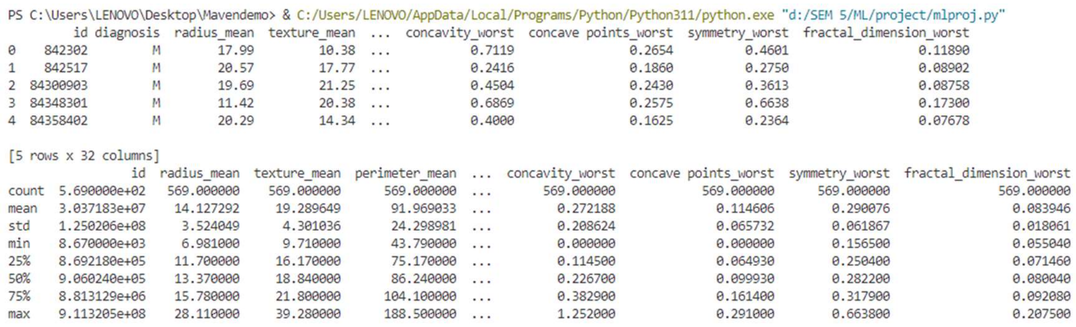
ANN and KNN CODE:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
# Load the dataset (replace 'file_path' with your actual file path)
file_path = "D:\SEM 5\ML\project\data.csv"
data = pd.read_csv(file_path)
# Data preprocessing for KNN
X_knn = data.drop(['id', 'diagnosis'], axis=1)  # Features for KNN
y_knn = data['diagnosis']  # Target variable for KNN
label_encoder = LabelEncoder()
y_knn = label_encoder.fit_transform(y_knn)
# Scale features for KNN
```

```
scaler_knn = StandardScaler()
X_knn_scaled = scaler_knn.fit_transform(X_knn)
# Perform PCA for KNN
pca_knn = PCA(n_components=2)
X_knn_pca = pca_knn.fit_transform(X_knn_scaled)
# Split data for KNN
X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X_knn_pca, y_knn, test_size=0.2,
random_state=42)
# Initialize the KNN classifier
k = 5  # Number of neighbors (can be adjusted)
knn = KNeighborsClassifier(n_neighbors=k)
# Train the KNN model
knn.fit(X_train_knn, y_train_knn)
# Predictions using KNN on the test set
y_pred_knn = knn.predict(X_test_knn)
# Evaluate the KNN model
accuracy_knn = accuracy_score(y_test_knn, y_pred_knn)
print(f"KNN Accuracy: {accuracy_knn:.2f}")
print("\nKNN Classification Report:")
print(classification_report(y_test_knn, y_pred_knn))
print("\nKNN Confusion Matrix:")
print(confusion_matrix(y_test_knn, y_pred_knn))
# Plotting for KNN with decision boundary
plt.figure(figsize=(6, 4))

# Plotting decision boundary by creating a mesh grid
h = .02  # Step size in the mesh
x_min, x_max = X_knn_pca[:, 0].min() - 1, X_knn_pca[:, 0].max() + 1
y_min, y_max = X_knn_pca[:, 1].min() - 1, X_knn_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                    np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Plot also the test points
plt.scatter(X_test_knn[:, 0], X_test_knn[:, 1], c=y_test_knn, cmap=plt.cm.coolwarm)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('KNN Classification')
plt.show()
X = data.drop(['id', 'diagnosis'], axis=1)  # Features
y = data['diagnosis']  # Target variable
# Split the dataset 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)
# Standardize features by scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
# Initialize the ANN model
model = Sequential()
# Add input layer and hidden layers
model.add(Dense(units=32, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=16, activation='relu'))
# Add output layer
# Adjust units based on your target variable type (e.g., binary classification)
model.add(Dense(units=1, activation='sigmoid'))  # For binary classification
```

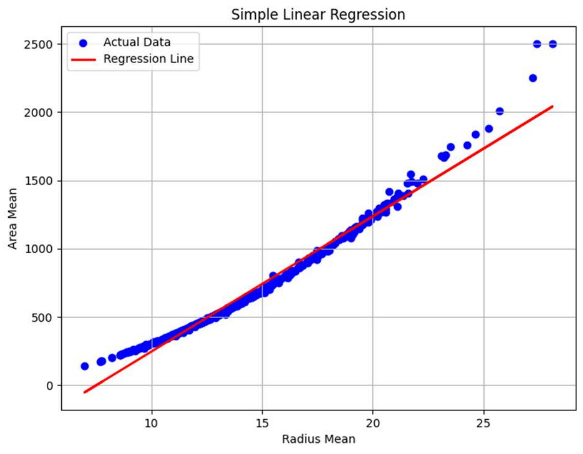
```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model and store the history
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy*100:.2f}%')
# Plot training & validation accuracy values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()
```

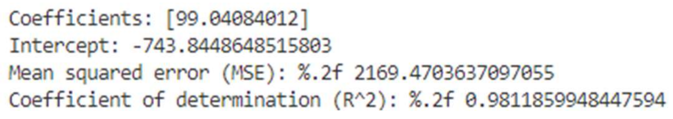
OUTPUT:



```
PS C:\Users\LENOVO\Desktop\Mavendemo> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python311/python.exe "d:/SEM 5/ML/project/mlproj.py"
         id diagnosis  radius_mean  texture_mean  ...  concavity_worst  concave points_worst  symmetry_worst  fractal_dimension_worst
0    842302         M        17.99         10.38  ...           0.7119                0.2654          0.4601                  0.11890
1    842517         M        20.57         17.77  ...           0.2416                0.1860          0.2750                  0.08902
2  84300903         M        19.69         21.25  ...           0.4504                0.2430          0.3613                  0.08758
3  84348301         M        11.42         20.38  ...           0.6869                0.2575          0.6638                  0.17300
4  84358402         M        20.29         14.34  ...           0.4000                0.1625          0.2364                  0.07678

[5 rows x 32 columns]
                 id  radius_mean  texture_mean  perimeter_mean  ...  concavity_worst  concave points_worst  symmetry_worst  fractal_dimension_worst
count  5.690000e+02   569.000000    569.000000      569.000000  ...       569.000000            569.000000      569.000000               569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033  ...         0.272188              0.114606        0.290076                 0.083946
std    1.250206e+08     3.524049      4.301036       24.298981  ...         0.208624              0.065732        0.061867                 0.018061
min    8.670000e+03     6.981000      9.710000       43.790000  ...         0.000000              0.000000        0.156500                 0.055040
25%    8.692180e+05    11.700000     16.170000       75.170000  ...         0.114500              0.064930        0.250400                 0.071460
50%    9.060240e+05    13.370000     18.840000       86.240000  ...         0.226700              0.099930        0.282200                 0.080040
75%    8.813129e+06    15.780000     21.800000      104.100000  ...         0.382900              0.161400        0.317900                 0.092080
max    9.113205e+08    28.110000     39.280000      188.500000  ...         1.252000              0.291000        0.663800                 0.207500
```

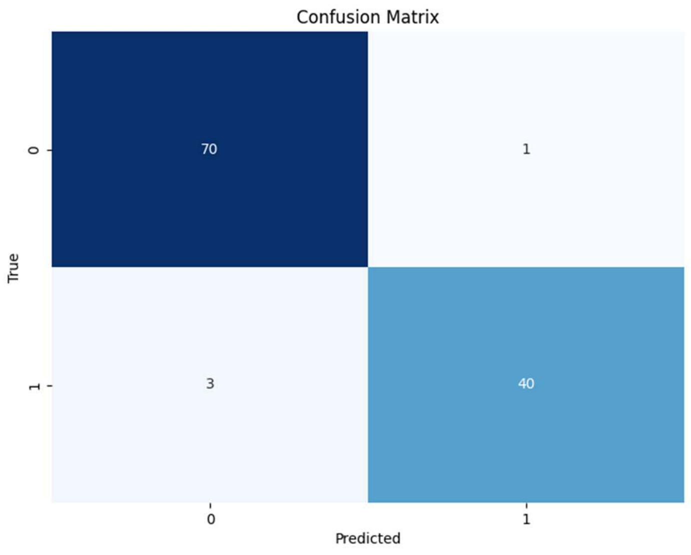
SIMPLR LINEAR REGRESSION:





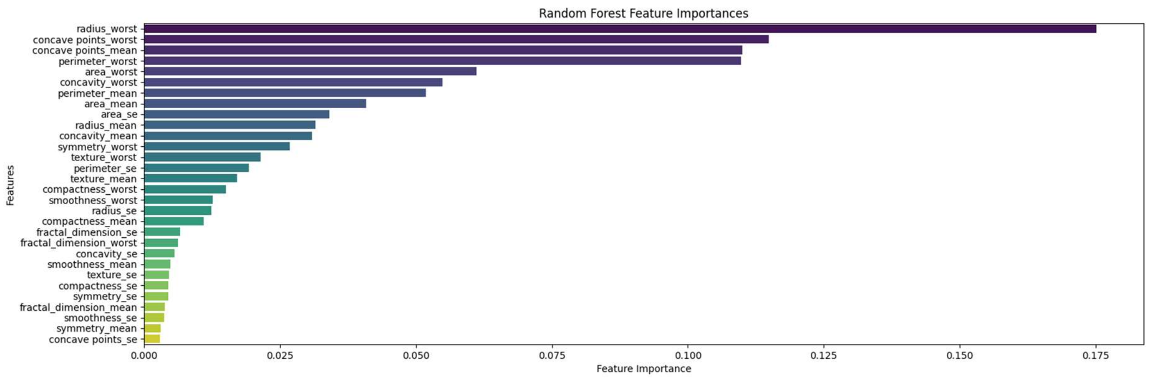
```
Coefficients: [99.04084012]
Intercept: -743.8448648515803
Mean squared error (MSE): %.2f 2169.4703637097055
Coefficient of determination (R^2): %.2f 0.9811859948447594
```

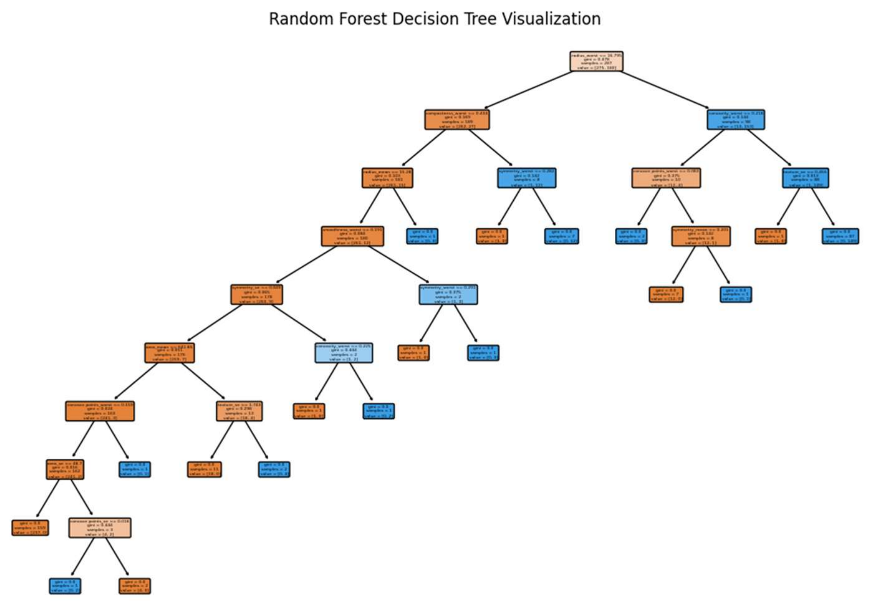
LOGISTIC REGRESSION:



```
Accuracy: 0.96

Confusion Matrix:
[[70  1]
 [ 3 40]]

Classification Report:
              precision    recall  f1-score   support

           B       0.96      0.99      0.97        71
           M       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

RANDOM FOREST MODEL:



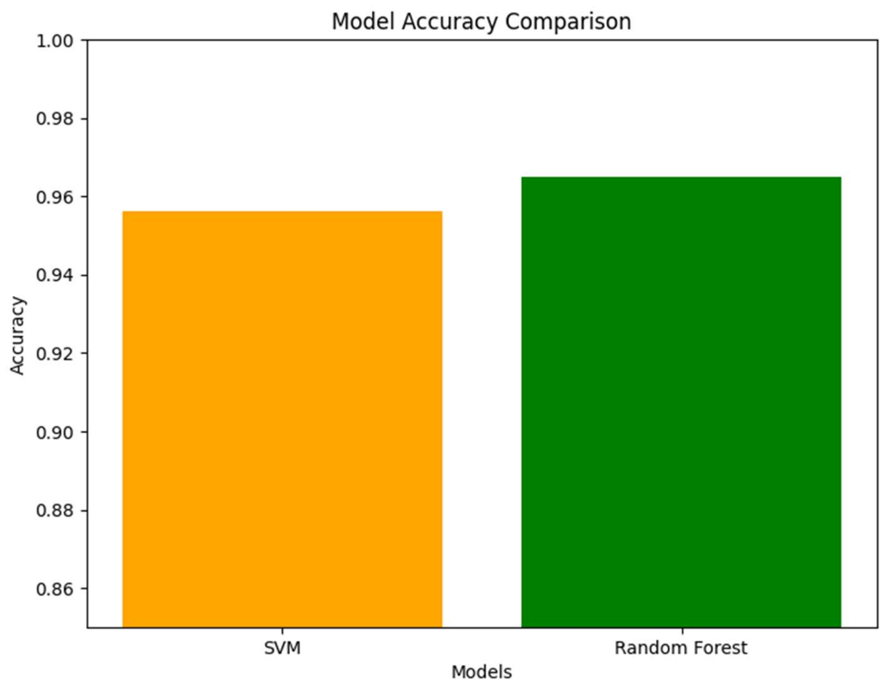


```
Random Forest Accuracy: 0.96

Random Forest Classification Report:
              precision    recall  f1-score   support

           B       0.96      0.99      0.97        71
           M       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

SVM:

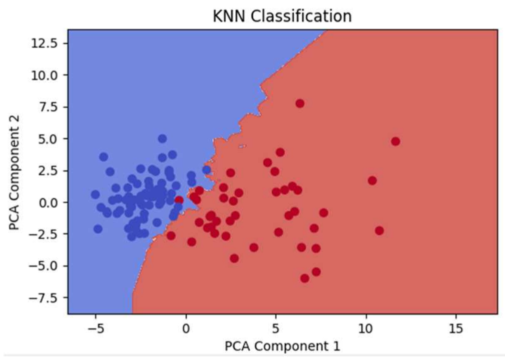
```
SVM Accuracy: 0.96

SVM Classification Report:
              precision    recall  f1-score   support

           B       0.95      0.99      0.97        71
           M       0.97      0.91      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

MODEL COMPARISON:



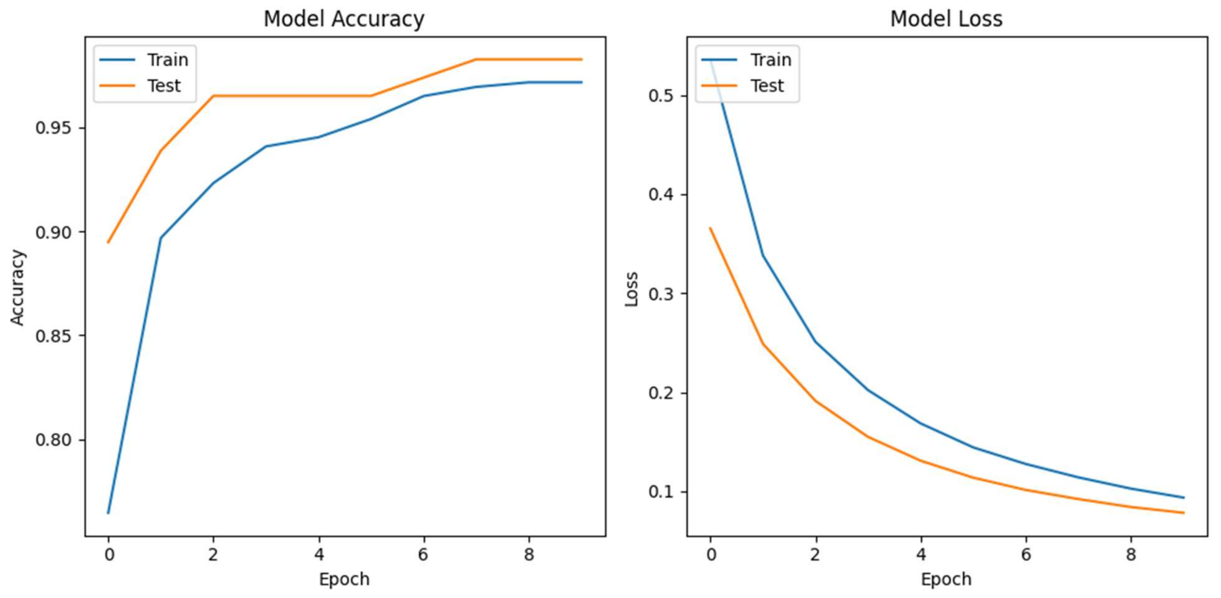
K-NN:

```
KNN Accuracy: 0.97

KNN Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98        71
           1       0.95      0.98      0.97        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114


KNN Confusion Matrix:
[[69  2]
 [ 1 42]]
```



ANN:

Accuracy: 98.25%



CONCLUSION:

Summarize the key findings, insights, and observations derived from the analysis. Discuss the strengths and weaknesses of the employed models and techniques. Emphasize the significance of accurate breast cancer prediction methodologies and potential future directions for improvement or expansion of the analysis.

This outline provides a structured framework for your project report. Ensure to expand each section with relevant details, analysis results, visualizations, and interpretations to provide a comprehensive understanding of your work. Adjustments can be made based on the specific results and insights gained during your analysis.