Title

Parkinson’s Disease Prediction

Overview of Problem Statement

Parkinson’s disease is a progressive neurological disorder affecting movement, speech, and ove motor functions. Early detection is crucial for effective intervention and improving patient quality c The aim of this project is to develop machine learning models to predict whether a patient has Parkinson’s disease based on biomedical voice measurements and clinical indicators.

Objective

Conduct Exploratory Data Analysis (EDA) to understand feature distributions and relationships.

Preprocess the dataset by handling missing values, duplicates, scaling, and encoding.

Apply feature selection and dimensionality reduction techniques such as RFE (Recursive Featur Elimination), Variance Threshold, and PCA.

Train and compare multiple classification algorithms.

Evaluate model performance using accuracy, classification report, and confusion matrix.

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE, VarianceThreshold
from sklearn.decomposition import PCA
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Load the dataset
df = pd.read_csv("parkinson_disease.csv")
```

In [4]:
```
# Display the first few rows of the dataset to understand its structure
print("First few rows of the dataset:")
df.head()
```

```
First few rows of the dataset:
```

Out[4]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPuls
0	0	1	0.85247	0.71826	0.57227	240	239	0.0080
1	0	1	0.76686	0.69481	0.53966	234	233	0.0082
2	0	1	0.85083	0.67604	0.58982	232	231	0.0083
3	1	0	0.41121	0.79672	0.59257	178	177	0.0108
4	1	0	0.32790	0.79782	0.53028	236	235	0.0081

5 rows × 755 columns

In [5]:

```
# Display the last few rows of the dataset
print("Last few rows of the dataset:")
df.tail()
```

```
Last few rows of the dataset:
```

Out[5]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodI
751	250	0	0.80903	0.56355	0.28385	417	416	0.(
752	250	0	0.16084	0.56499	0.59194	415	413	0.(
753	251	0	0.88389	0.72335	0.46815	381	380	0.(
754	251	0	0.83782	0.74890	0.49823	340	339	0.(
755	251	0	0.81304	0.76471	0.46374	340	339	0.(

5 rows × 755 columns

In [6]:

```
# Display the size of the dataset (number of rows and columns)
print("Dataset size:")
print(df.shape)
```

```
Dataset size:
(756, 755)
```

In [7]:

```
# Display the columns of the dataset
columns = df.columns
print("Columns in the dataset:",columns)
```

```
Columns in the dataset: Index(['id', 'gender', 'PPE', 'DFA', 'RPDE', 'numPuls
'numPeriodsPulses',
       'meanPeriodPulses', 'stdDevPeriodPulses', 'locPctJitter',
       ...
       'tqwt_kurtosisValue_dec_28', 'tqwt_kurtosisValue_dec_29',
       'tqwt_kurtosisValue_dec_30', 'tqwt_kurtosisValue_dec_31',
       'tqwt_kurtosisValue_dec_32', 'tqwt_kurtosisValue_dec_33',
       'tqwt_kurtosisValue_dec_34', 'tqwt_kurtosisValue_dec_35',
       'tqwt_kurtosisValue_dec_36', 'class'],
      dtype='object', length=755)
```

In [8]:

```
# Numerical columns
numerical_features = df.select_dtypes(include='number').columns
print(numerical_features)
```

```
Index(['id', 'gender', 'PPE', 'DFA', 'RPDE', 'numPulses', 'numPeriodsPulses',
       'meanPeriodPulses', 'stdDevPeriodPulses', 'locPctJitter',
       ...
       'tqwt_kurtosisValue_dec_28', 'tqwt_kurtosisValue_dec_29',
       'tqwt_kurtosisValue_dec_30', 'tqwt_kurtosisValue_dec_31',
       'tqwt_kurtosisValue_dec_32', 'tqwt_kurtosisValue_dec_33',
       'tqwt_kurtosisValue_dec_34', 'tqwt_kurtosisValue_dec_35',
       'tqwt_kurtosisValue_dec_36', 'class'],
      dtype='object', length=755)
```

In [9]:
```
# Categorical columns
categorical_features= df.select_dtypes(include=['object']).columns
print(categorical_features)
```

```
Index([], dtype='object')
```

EDA (Exploratory Data Analysis)

In [10]:
```
# Get a summary of the dataset
print("Summary of the dataset:")
df.info()
```

```
Summary of the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: float64(749), int64(6)
memory usage: 4.4 MB
```

In [11]:
```
# Display statistical summary for all columns within the dataFram
df.describe(include='all')
```

Out[11]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeri
count	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	7
mean	125.500000	0.515873	0.746284	0.700414	0.489058	323.972222	3
std	72.793721	0.500079	0.169294	0.069718	0.137442	99.219059	
min	0.000000	0.000000	0.041551	0.543500	0.154300	2.000000	
25%	62.750000	0.000000	0.762833	0.647053	0.386537	251.000000	2
50%	125.500000	1.000000	0.809655	0.700525	0.484355	317.000000	3
75%	188.250000	1.000000	0.834315	0.754985	0.586515	384.250000	3
max	251.000000	1.000000	0.907660	0.852640	0.871230	907.000000	9

8 rows × 755 columns

In [12]:
```
# Check for Null Values
print("Checking for missing values:")
print(df.isnull().sum())
```

```
Checking for missing values:
id                            0
gender                        0
PPE                           0
DFA                           0
RPDE                          0
                             ..
tqwt_kurtosisValue_dec_33     0
tqwt_kurtosisValue_dec_34     0
tqwt_kurtosisValue_dec_35     0
tqwt_kurtosisValue_dec_36     0
class                         0
Length: 755, dtype: int64
```

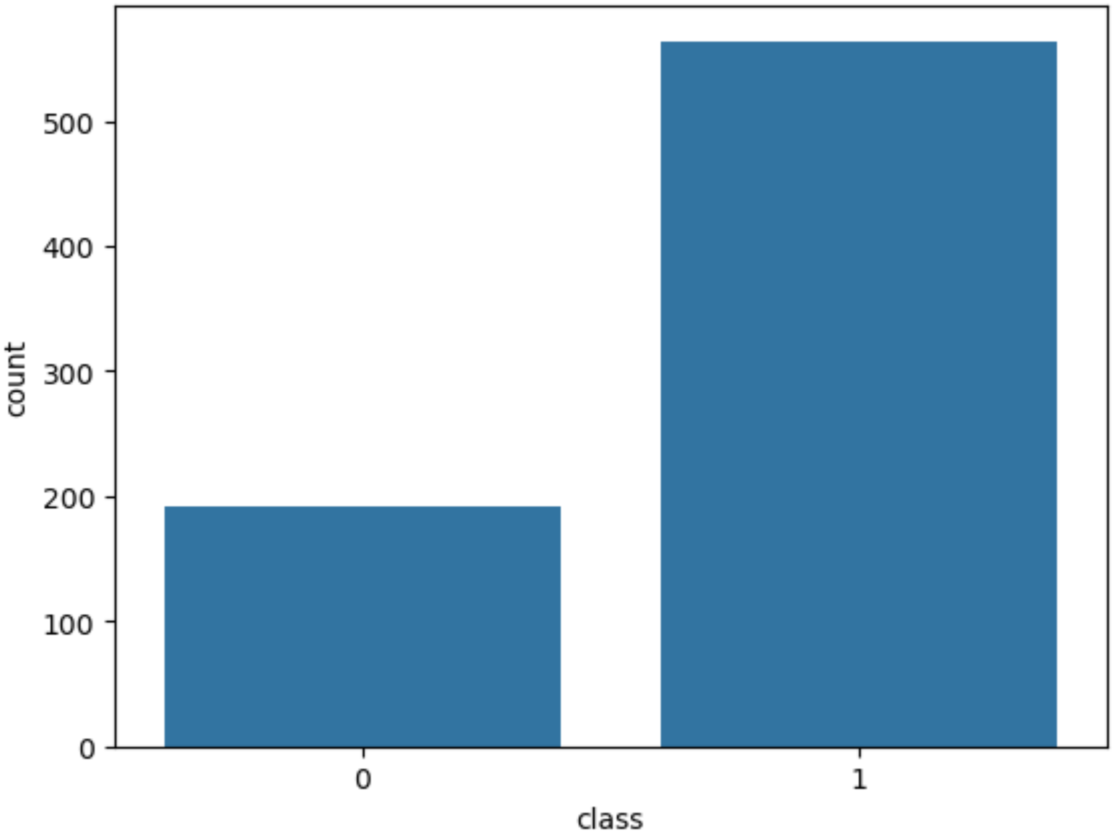
In [13]:
```
# Check for Null Duplicates
print("Checking for duplicate records:")
print(df.duplicated().sum())
```

```
Checking for duplicate records:
1
```

Data Visualization

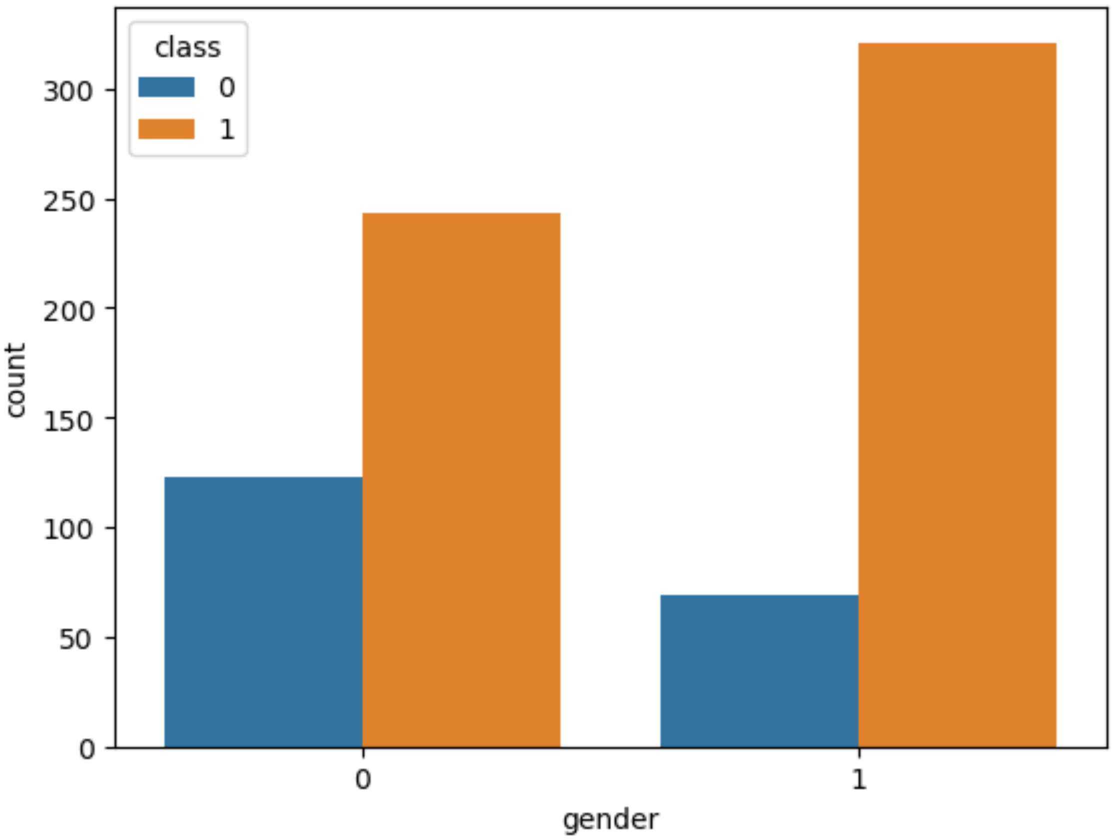
In [14]:
```
# Checking for data balanced
sns.countplot(x=df['class'])
```

Out[14]: <Axes: xlabel='class', ylabel='count'>



In [15]:
```
# distribution of the 'gender' feature in the dataset
sns.countplot(x=df['gender'], hue = df['class'])
```

Out[15]: <Axes: xlabel='gender', ylabel='count'>



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Data Preprocessing

In [16]:
```
# Drop the 'id' column from the DataFrame
df.drop(columns=['id'], inplace=True)
```

In [17]:
```
# Removes duplicate rows from the DataFrame
df.drop_duplicates(inplace=True)
```

In [18]:
```
# Display the size of the dataset (number of rows and columns)
print("Dataset size:")
print(df.shape)
```

```
Dataset size:
(755, 754)
```

Split features and target

In [19]:
```
X = df.drop(columns=['class'])
y = df['class']
```

Scale features

In [20]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Train-test split

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_siz
```

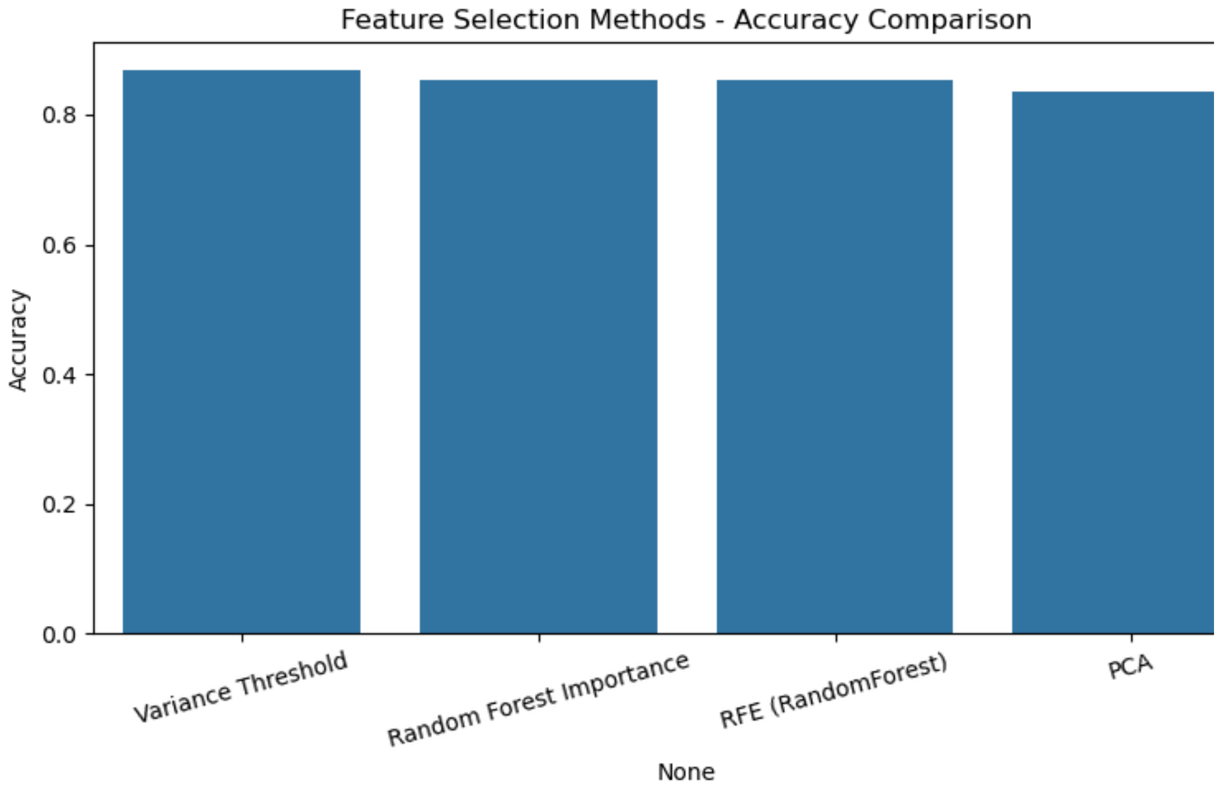
In [ ]:

Model Training & Evaluation

```
In [22]: #  Initialize result storage
         detailed_results = {}

         # 1. Random Forest Feature Importance
         rf = RandomForestClassifier(random_state=42)
         rf.fit(X_train, y_train)
         importances = pd.Series(rf.feature_importances_, index=X.columns).sort_va
         top_rf_features = importances.head(50).index
         X_train_rf = X_train[:, [X.columns.get_loc(f) for f in top_rf_features]]
         X_test_rf = X_test[:, [X.columns.get_loc(f) for f in top_rf_features]]
         model_rf = RandomForestClassifier(random_state=42)
         model_rf.fit(X_train_rf, y_train)
         pred_rf = model_rf.predict(X_test_rf)
         detailed_results['Random Forest Importance'] = classification_report(y_te

         # 2. RFE with Random Forest (optimized)
         rfe_model = RFE(estimator=RandomForestClassifier(n_estimators=50, random_
         X_train_rfe = rfe_model.fit_transform(X_train, y_train)
         X_test_rfe = rfe_model.transform(X_test)
         model_rfe = RandomForestClassifier(random_state=42)
         model_rfe.fit(X_train_rfe, y_train)
         pred_rfe = model_rfe.predict(X_test_rfe)
         detailed_results['RFE (RandomForest)'] = classification_report(y_test, pr

         # 3. Variance Threshold
         var_thresh = VarianceThreshold(threshold=0.01)
         X_train_var = var_thresh.fit_transform(X_train)
         X_test_var = var_thresh.transform(X_test)
         model_var = RandomForestClassifier(random_state=42)
         model_var.fit(X_train_var, y_train)
         pred_var = model_var.predict(X_test_var)
         detailed_results['Variance Threshold'] = classification_report(y_test, pr

         # 4. PCA
         pca = PCA(n_components=50)
         X_train_pca = pca.fit_transform(X_train)
         X_test_pca = pca.transform(X_test)
         model_pca = RandomForestClassifier(random_state=42)
         model_pca.fit(X_train_pca, y_train)
         pred_pca = model_pca.predict(X_test_pca)
         detailed_results['PCA'] = classification_report(y_test, pred_pca, output_

         # Compile Metrics for Comparison
         metrics_df = pd.DataFrame({
             method: {
                 'Accuracy': rep['accuracy'],
                 'Precision': rep['weighted avg']['precision'],
                 'Recall': rep['weighted avg']['recall'],
                 'F1-Score': rep['weighted avg']['f1-score']
             }
             for method, rep in detailed_results.items()
         }).T

         #  Sort by Accuracy in Descending Order
         metrics_df = metrics_df.sort_values(by='Accuracy', ascending=False)

         #  Display Final Table
         print("Feature Selection Method Comparison (Accuracy, Precision, Recall, 
         display(metrics_df)
```

```
Feature Selection Method Comparison (Accuracy, Precision, Recall, F1-Score):
```

	Accuracy	Precision	Recall	F1-Score
Variance Threshold	0.867550	0.888574	0.867550	0.854337
Random Forest Importance	0.854305	0.864701	0.854305	0.841794
RFE (RandomForest)	0.854305	0.864701	0.854305	0.841794
PCA	0.834437	0.855852	0.834437	0.814122



Results

Logistic Regression: Provided baseline accuracy.

Random Forest Classifier: Outperformed Logistic Regression with higher accuracy and F1-score

Important features included voice-related biomarkers like jitter, shimmer, and HNR.

Conclusion

Machine learning models can effectively predict Parkinson’s disease using voice and biomedical indicators. Random Forest showed better predictive performance than Logistic Regression. Feat selection and dimensionality reduction improved model efficiency and interpretability.

Conclusion

Machine learning models can effectively predict Parkinson’s disease using voice and biomedical indicators. Random Forest showed better predictive performance than Logistic Regression. Feat selection and dimensionality reduction improved model efficiency and interpretability.