

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d uciml/pima-indians-diabetes-database
!unzip -o pima-indians-diabetes-database.zip

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")
df = pd.read_csv('diabetes.csv')

print("Dataset Loaded Successfully!")
df.head()
```

Dataset Loaded Successfully!

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1



Next steps:

Generate code with df

New interactive sheet

```
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)

for column in df.select_dtypes(include=['object']).columns:
    if not df[column].empty:
        mode_value = df[column].mode()[0]
        df[column] = df[column].fillna(mode_value)

X = df.drop('Outcome', axis=1)
y = df['Outcome']

print("Data Cleaning Complete. Missing values:")
print(df.isnull().sum())
```

Data Cleaning Complete. Missing values:

Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

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

param_grid_rf = {'n_estimators': [50, 100], 'max_depth': [None, 10, 20]}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

param_grid_svm = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
grid_svm = GridSearchCV(SVC(), param_grid_svm, cv=5)
grid_svm.fit(X_train, y_train)

print("Best RF Params:", grid_rf.best_params_)
print("Best SVM Params:", grid_svm.best_params_)
```

Best RF Params: {'max_depth': None, 'n_estimators': 50}
Best SVM Params: {'C': 0.1, 'kernel': 'linear'}

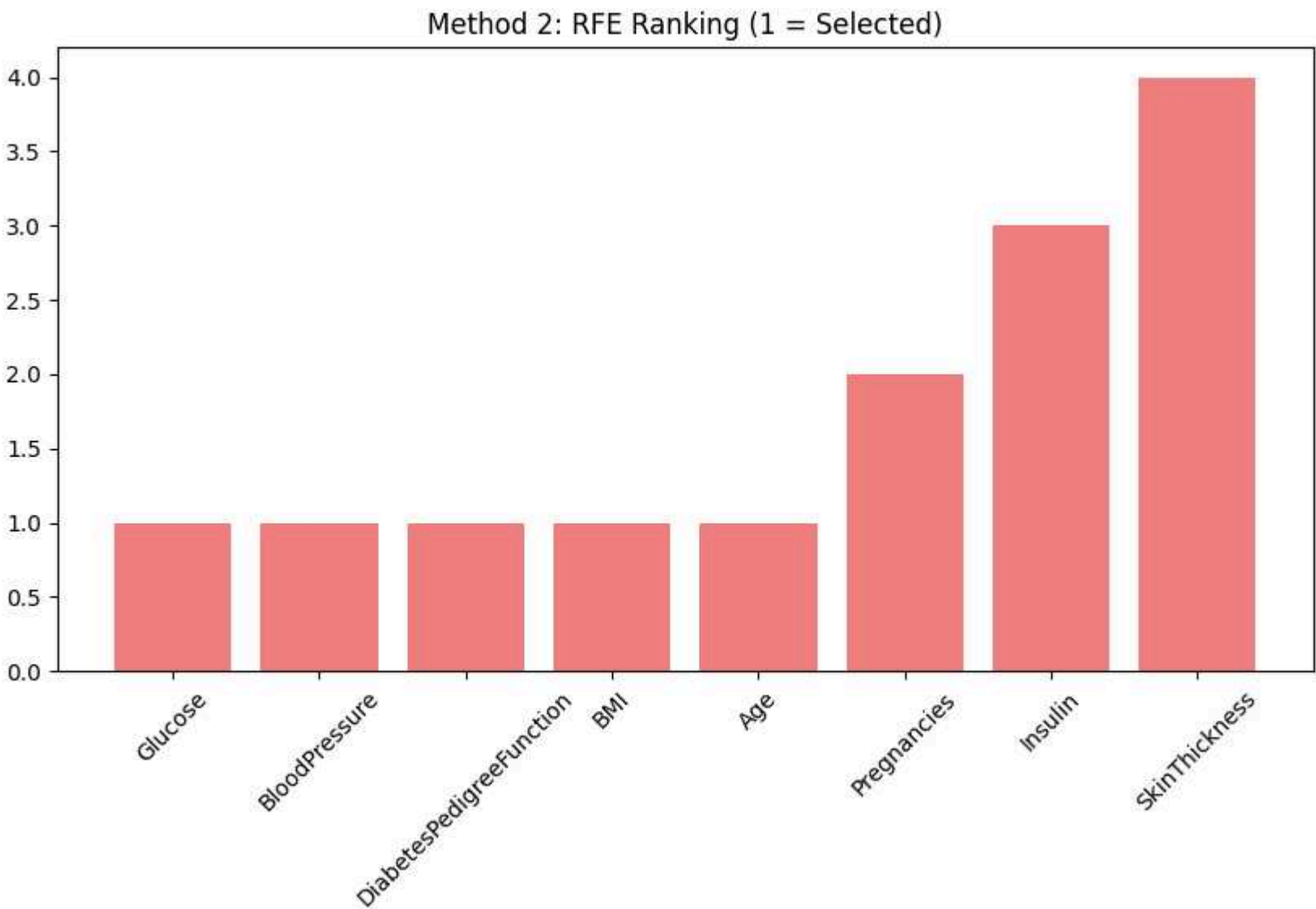
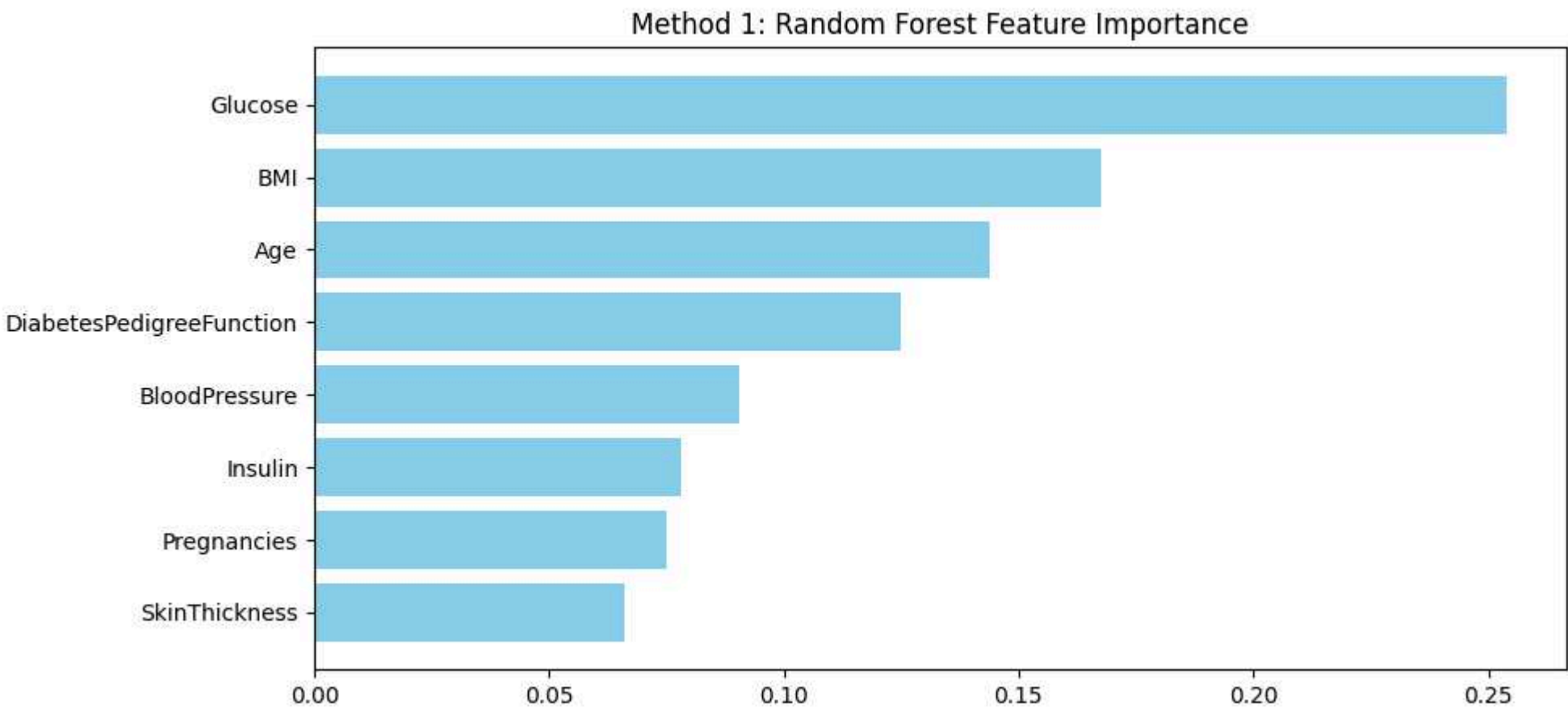
```
from sklearn.feature_selection import SelectKBest, chi2, RFE

# Method 1: Feature Importance Random Forest
best_rf = grid_rf.best_estimator_
importances = best_rf.feature_importances_
df_importance = pd.DataFrame({'Feature': X.columns, 'Importance': importances}).sort_values(by='Importance', ascending=True)

# Method 2: RFE Ranking
rfe = RFE(estimator=best_rf, n_features_to_select=5)
rfe.fit(X, y)
ranking_df = pd.DataFrame({'Feature': X.columns, 'RFE_Ranking': rfe.ranking_}).sort_values('RFE_Ranking')

# Plotting Method 1
plt.figure(figsize=(10, 5))
plt.barh(df_importance['Feature'], df_importance['Importance'], color='skyblue')
plt.title("Method 1: Random Forest Feature Importance")
plt.show()

# Plotting Method 2
plt.figure(figsize=(10, 5))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'], color='lightcoral')
plt.title("Method 2: RFE Ranking (1 = Selected)")
plt.xticks(rotation=45)
plt.show()
```



```
from sklearn.metrics import accuracy_score, precision_score, recall_score

selected_features = X.columns[rfe.get_support()]
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

performance = []
for name, model in {"Random Forest": best_rf, "SVM": grid_svm.best_estimator_}.items():
    model.fit(X_train_sel, y_train)
    y_pred = model.predict(X_test_sel)
    performance.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred)
    })

print("\n FINAL PERFORMANCE TABLE ")
print(pd.DataFrame(performance))
```

FINAL PERFORMANCE TABLE				
	Model	Accuracy	Precision	Recall
0	Random Forest	0.759740	0.650000	0.709091
1	SVM	0.766234	0.679245	0.654545