

```
# Aishah Abdullha Saeed Al Dhanhani
# عزيزتي دكتورة هاجر, انا استخدم نفس الديتاسيت اللي استخدمتها في المشروع الاول و صادف انتي استخدمتيها في محاضرة 9 اتمنى انها ما تكون مشكلة
```

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('/content/drive/MyDrive/First project /data.csv')
```

df.head()

|   | id       | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | ... | texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_worst | cor points_v |
|---|----------|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|---------------------|-----|---------------|-----------------|------------|------------------|-------------------|-----------------|--------------|
| 0 | 842302   | M         | 17.99       | 10.38        | 122.80         | 1001.0    | 0.11840         | 0.27760          | 0.3001         | 0.14710             | ... | 17.33         | 184.60          | 2019.0     | 0.1622           | 0.6656            | 0.7119          | 0            |
| 1 | 842517   | M         | 20.57       | 17.77        | 132.90         | 1326.0    | 0.08474         | 0.07864          | 0.0869         | 0.07017             | ... | 23.41         | 158.80          | 1956.0     | 0.1238           | 0.1866            | 0.2416          | 0            |
| 2 | 84300903 | M         | 19.69       | 21.25        | 130.00         | 1203.0    | 0.10960         | 0.15990          | 0.1974         | 0.12790             | ... | 25.53         | 152.50          | 1709.0     | 0.1444           | 0.4245            | 0.4504          | 0            |
| 3 | 84348301 | M         | 11.42       | 20.38        | 77.58          | 386.1     | 0.14250         | 0.28390          | 0.2414         | 0.10520             | ... | 26.50         | 98.87           | 567.7      | 0.2098           | 0.8663            | 0.6869          | 0            |
| 4 | 84358402 | M         | 20.29       | 14.34        | 135.10         | 1297.0    | 0.10030         | 0.13280          | 0.1980         | 0.10430             | ... | 16.67         | 152.20          | 1575.0     | 0.1374           | 0.2050            | 0.4000          | 0            |

5 rows × 33 columns

```
df.isnull().sum()
```

|                         | 0   |
|-------------------------|-----|
| id                      | 0   |
| 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   |
| Unnamed: 32             | 569 |

dtype: int64

```
missing_percentage = df.isnull().mean() * 100

threshold = 30

# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage > threshold].index

# Drop the columns
df = df.drop(columns=columns_to_drop)

# Display the resulting dataframe after removing columns with too many missing values
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {list(df.columns)}")
```

Columns dropped: ['Unnamed: 32']  
Remaining columns: ['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',

Start coding or [generate](#) with AI.

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:

    df[column] = label_encoder.fit_transform(df[column])
df.head()
```

|   | id       | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave<br>points_mean | ... | radius_worst | texture_worst | perimeter_worst | area_worst | smoothness_worst | compactness_worst | concavity_worst |
|---|----------|-----------|-------------|--------------|----------------|-----------|-----------------|------------------|----------------|------------------------|-----|--------------|---------------|-----------------|------------|------------------|-------------------|-----------------|
| 0 | 842302   | 1         | 17.99       | 10.38        | 122.80         | 1001.0    | 0.11840         | 0.27760          | 0.3001         | 0.14710                | ... | 25.38        | 17.33         | 184.60          | 2019.0     | 0.1622           | 0.6656            | 0.1985          |
| 1 | 842517   | 1         | 20.57       | 17.77        | 132.90         | 1326.0    | 0.08474         | 0.07864          | 0.0869         | 0.07017                | ... | 24.99        | 23.41         | 158.80          | 1956.0     | 0.1238           | 0.1866            | 0.2743          |
| 2 | 84300903 | 1         | 19.69       | 21.25        | 130.00         | 1203.0    | 0.10960         | 0.15990          | 0.1974         | 0.12790                | ... | 23.57        | 25.53         | 152.50          | 1709.0     | 0.1444           | 0.4245            | 0.2376          |
| 3 | 84348301 | 1         | 11.42       | 20.38        | 77.58          | 386.1     | 0.14250         | 0.28390          | 0.2414         | 0.10520                | ... | 14.91        | 26.50         | 98.87           | 567.7      | 0.2098           | 0.8663            | 0.4308          |
| 4 | 84358402 | 1         | 20.29       | 14.34        | 135.10         | 1297.0    | 0.10030         | 0.13280          | 0.1980         | 0.10430                | ... | 22.54        | 16.67         | 152.20          | 1575.0     | 0.1374           | 0.2050            | 0.3463          |

5 rows × 32 columns

```
# Fill missing values in numeric columns with mean
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)
```

```
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequent value)
    df[column] = df[column].fillna(mode_value)
```

```
df.isnull().sum()
```

|                         | 0 |
|-------------------------|---|
| id                      | 0 |
| 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

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'criterion': ['gini', 'entropy']
}

grid_search_rf = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                              param_grid=param_grid_rf,
                              cv=5, scoring='accuracy')

grid_search_rf.fit(X_train_k, y_train)

best_rf_model = grid_search_rf.best_estimator_
print("Best Parameters for Random Forest:", grid_search_rf.best_params_)
```

Best Parameters for Random Forest: {'criterion': 'entropy', 'max\_depth': None, 'n\_estimators': 100}

```
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['rbf', 'linear'],
    'gamma': ['scale', 'auto']
}

grid_search_svm = GridSearchCV(SVC(), param_grid_svm, cv=5, scoring='accuracy')
grid_search_svm.fit(X_train_k, y_train)

print("Best Parameters for SVM:", grid_search_svm.best_params_)
```

Best Parameters for SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

```
x = df.drop(['diagnosis','id'], axis=1)
y = df["diagnosis"]
```

```
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(score_func=chi2, k=8)
```

```
x_new = selector.fit_transform(x, y)
```

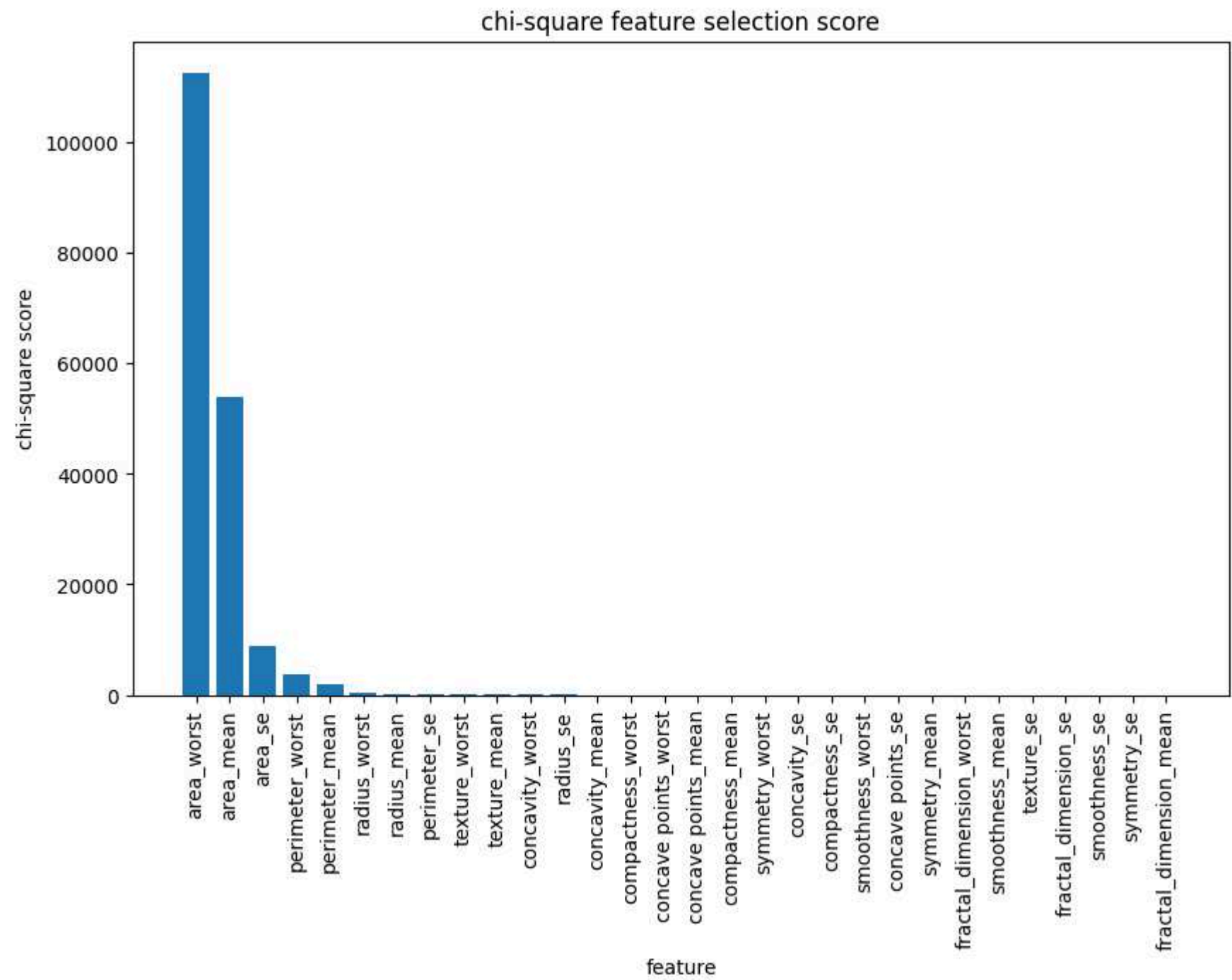
```
print("selected featured:", selector.get_support())
print("score:", selector.scores_)
```

selected featured: [ True False True True False False False False False False False False  
 True True False False False False False True False True True  
 False False False False False False]  
score: [2.66104917e+02 9.38975081e+01 2.01110286e+03 5.39916559e+04  
1.49899264e-01 5.40307549e+00 1.97123536e+01 1.05440354e+01  
2.57379775e-01 7.43065536e-05 3.46752472e+01 9.79353970e-03  
2.50571896e+02 8.75850471e+03 3.26620664e-03 6.13785332e-01  
1.04471761e+00 3.05231563e-01 8.03633831e-05 6.37136566e-03  
4.91689157e+02 1.74449400e+02 3.66503542e+03 1.12598432e+05  
3.97365694e-01 1.93149220e+01 3.95169151e+01 1.34854195e+01  
1.29886140e+00 2.31522407e-01]

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.DataFrame({'feature': x.columns, 'score': selector.scores_})
data = data.sort_values(by='score' , ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(data['feature'], data['score'])
plt.xlabel("feature")
plt.ylabel("chi-square score")
plt.title("chi-square feature selection score")
plt.xticks(rotation=90)
plt.show()
```



```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=15)
rfe.fit(x, y)
print("Selected Features:", rfe.get_support())
print("Ranking:", rfe.ranking_)
```

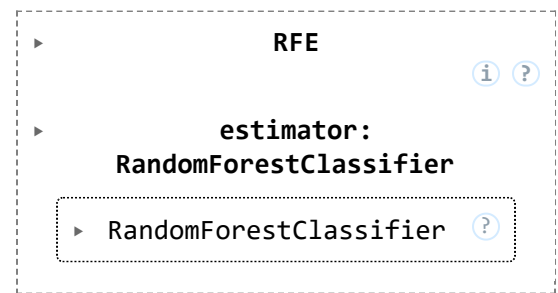
```
Selected Features: [ True False False False  True  True  True  True  True False False  True
 False False False False False False  True  True  True  True False
  True  True  True  True  True False]
Ranking: [ 1  2  4 16  1  1  1  1  1 12  7  1  9  3 13 15 11  6  5 14  1  1  1  8
 1  1  1  1  1 10]
```

```
from sklearn.datasets import load_iris
import pandas as pd

data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

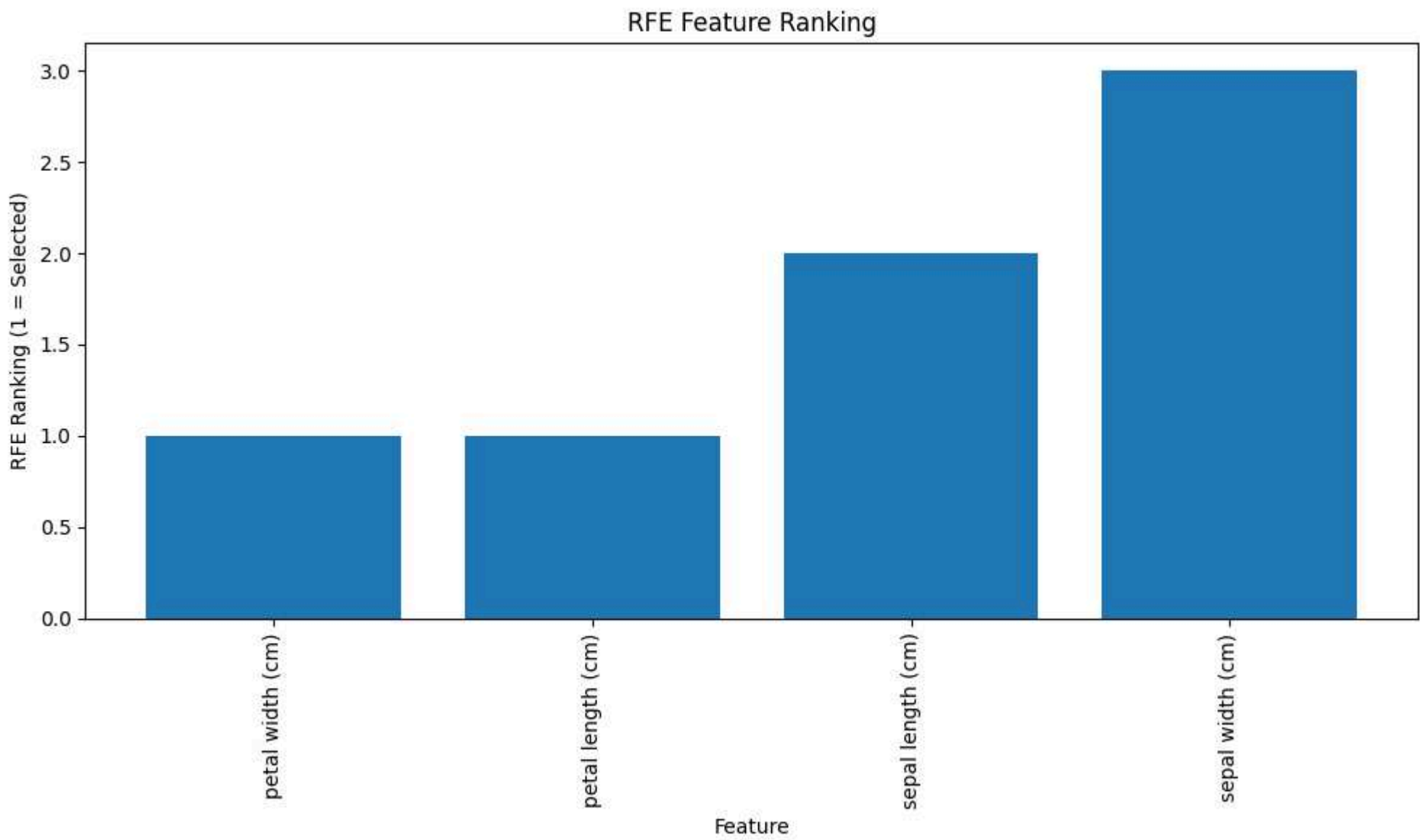
model = RandomForestClassifier()
rfe = RFE(estimator=model, n_features_to_select=2)
rfe.fit(X, y)
```



```
import pandas as pd
import matplotlib.pyplot as plt

feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
feature_ranking = feature_ranking.sort_values(by='Ranking', ascending=True)

plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



```
rf = RandomForestClassifier(n_estimators=200, random_state=42)

rfe = RFE(estimator=rf, n_features_to_select=15)
rfe.fit(X, y)
print("Selected Features Mask:", rfe.get_support())
print("Ranking of Features:", rfe.ranking_)
```

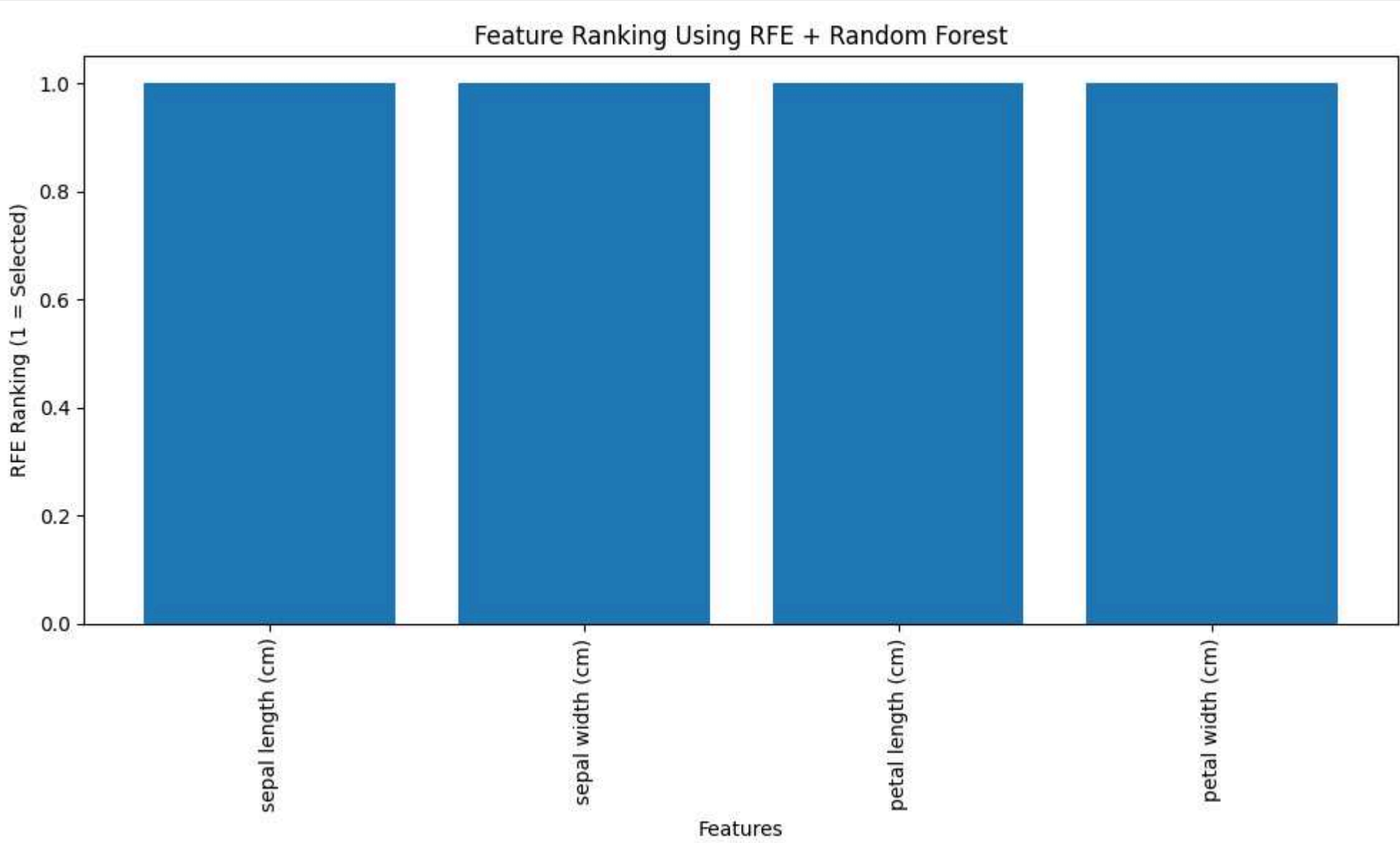
```
Selected Features Mask: [ True  True  True  True]
Ranking of Features: [1 1 1 1]
```

```
import pandas as pd
import matplotlib.pyplot as plt

ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')

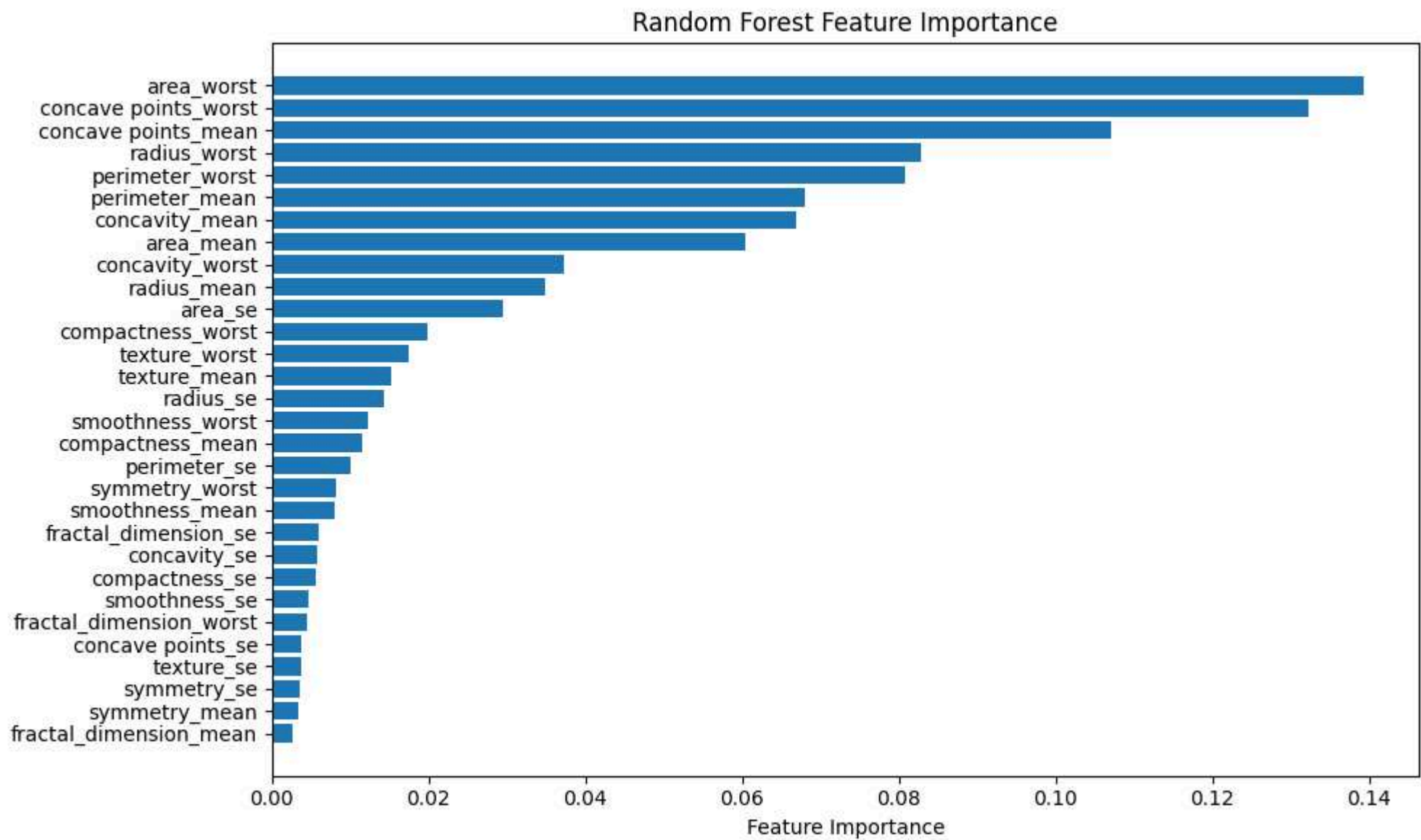
plt.figure(figsize=(10, 6))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'])
plt.xlabel("Features")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking Using RFE + Random Forest")
plt.xticks(rotation=90)
```

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



```
df = pd.read_csv('/content/drive/MyDrive/First project /data.csv')
# Explicitly drop the 'Unnamed: 32' column which caused issues
if 'Unnamed: 32' in df.columns:
    df = df.drop(columns=['Unnamed: 32'])

le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']
# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
# Extract importances
importances = rf.feature_importances_
# Use feature names from X (correct and safe)
feature_names = X.columns if hasattr(X, "columns") else [f"Feature{i}" for i in range(X.shape[1])]
# Create DataFrame
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
# Sort by importance
df_importance = df_importance.sort_values(by='Importance', ascending=False)
# Plot
plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1], df_importance['Importance'][::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer

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

imputer = SimpleImputer(strategy='mean')
X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X.columns)
X_test_imputed = pd.DataFrame(imputer.transform(X_test), columns=X.columns)

# chi2  $0 \leq$  القيم
X_train_imputed = X_train_imputed - X_train_imputed.min()
X_test_imputed = X_test_imputed - X_test_imputed.min()

k = 15
selector = SelectKBest(chi2, k=k)
X_train_k = selector.fit_transform(X_train_imputed, y_train)
X_test_k = selector.transform(X_test_imputed)
selected_features = X_train_imputed.columns[selector.get_support()]
print("Selected Features:", list(selected_features))

# تعيين البيانات قبل SVM
scaler = StandardScaler()
X_train_k = scaler.fit_transform(X_train_k)
X_test_k = scaler.transform(X_test_k)

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
```

```
metrics = []

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
```

Selected Features: ['radius\_mean', 'texture\_mean', 'perimeter\_mean', 'area\_mean', 'concavity\_mean', 'radius\_se', 'perimeter\_se', 'area\_se', 'radius\_worst', 'texture\_worst', 'perimeter\_worst', 'area\_worst', 'compactness\_worst', 'concavity\_worst', 'con

Model Performance:

|   | Model         | Accuracy | Precision | Recall   |
|---|---------------|----------|-----------|----------|
| 0 | Random Forest | 0.973684 | 0.973719  | 0.973684 |
| 1 | Decision Tree | 0.938596 | 0.938457  | 0.938596 |
| 2 | SVM           | 0.973684 | 0.973719  | 0.973684 |

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
import pandas as pd

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

imputer = SimpleImputer(strategy='mean')
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns=X.columns)
X_test = pd.DataFrame(imputer.transform(X_test), columns=X.columns)

k = 15
rf_selector = RandomForestClassifier(n_estimators=200, random_state=42)
rfe = RFE(estimator=rf_selector, n_features_to_select=k)
rfe.fit(X_train, y_train)

X_train_k = rfe.transform(X_train)
X_test_k = rfe.transform(X_test)

selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}

metrics = []

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

results_df = pd.DataFrame(metrics)
```