

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

```
import pandas as pd
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
```

```
import os
print(os.listdir())
```

```
['.config', 'diabetes.csv', 'amazon.csv', 'drive', 'sample_data']
```

```
import pandas as pd

df = pd.read_csv("diabetes.csv")

df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPec
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

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

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

dtype: int64

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

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	148	72	35	0	33.6	0.627
1	85	66	29	0	26.6	0.351
2	183	64	0	0	23.3	0.672
3	89	66	23	94	28.1	0.167
4	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...
763	101	76	48	180	32.9	0.178
764	122	70	27	0	36.8	0.351
765	121	72	23	112	26.2	0.178
766	126	60	0	0	30.1	0.351
767	93	70	31	0	30.4	0.351

768 rows × 8 columns

```
print(df.head())
```

```

Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0             6      148             72             35      0  33.6
1             1       85             66             29      0  26.6
2             8      183             64              0      0  23.3
3             1       89             66             23     94  28.1
4             0      137             40             35    168  43.1

DiabetesPedigreeFunction  Age  Outcome
0             0.627     50         1
1             0.351     31         0
2             0.672     32         1
3             0.167     21         0
4             2.288     33         1

```

```
print(df.columns)
```

```

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

```

```

X = df.drop("Pregnancies", axis=1)
y = df["Pregnancies"]

```

```

from sklearn.ensemble import RandomForestRegressor
X_train, X_test, y_train, y_test = train_test_split(

```

```
X, y, test_size=0.25, random_state=42
)
```

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

```
array([[ 0.84832379,  0.14964075,  0.90726993, ...,  0.46849198,
         1.4259954 ,  1.36589591],
       [-1.12339636, -0.16054575,  0.53090156, ..., -0.36506078,
        -0.19067191, -0.73212021],
       [ 1.94372388, -0.26394125, -1.28821221, ...,  0.60439732,
        -0.10558415,  1.36589591],
       ...,
       [ 0.00330087,  0.14964075,  0.15453319, ..., -0.68519336,
        -0.27575966, -0.73212021],
       [ 0.1597866 , -0.47073225, -1.28821221, ..., -0.37110101,
         1.17073215,  1.36589591],
       [-0.8730192 ,  0.04624525,  0.65635768, ..., -0.47378505,
        -0.87137393, -0.73212021]])
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
```

```
def evaluate_model(model_name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
```

```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
```

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
rf =RandomForestRegressor(random_state=42)
```

```
print(X_train.dtypes)
```

```
Glucose          int64
BloodPressure    int64
SkinThickness    int64
Insulin          int64
BMI              float64
DiabetesPedigreeFunction float64
Age              int64
Outcome          int64
dtype: object
```

```
from sklearn.model_selection import train_test_split
```

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

```
print(df.shape)
print(df.columns)
```

```
(768, 9)
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
print(df["Pregnancies"].head())
```

```
0    6
1    1
2    8
3    1
4    0
Name: Pregnancies, dtype: int64
```

```
X = df.select_dtypes(include=["number"])
```

```
print("X shape:", X.shape)
print(X.head())
```

```
X shape: (768, 9)
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0             6     148             72             35         0  33.6
1             1      85             66             29         0  26.6
2             8     183             64              0         0  23.3
3             1      89             66             23        94  28.1
```

4	0	137	40	35	168	43.1
---	---	-----	----	----	-----	------

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

```
X = pd.get_dummies(X)
```

```
print(X.shape)
```

```
(768, 9)
```

```
y = df["Pregnancies"].astype(str).str.replace(",", "")
y = y.astype(float)
```

```
X = pd.get_dummies(df.drop("Pregnancies", axis=1))
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

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

```
rf = RandomForestRegressor(random_state=42)
```

```
# GridSearch
grid_search = GridSearchCV(
    estimator=rf,
    param_grid={
        "n_estimators": [100, 200],
        "max_depth": [10, None]
    },
    cv=3,
    n_jobs=-1
)
```

```
y = df["Pregnancies"].astype(str).str.replace(",", "")
y = y.astype(float)
```

```
X = pd.get_dummies(df.drop("Pregnancies", axis=1))
```

```
print(X_train.shape)
print(y_train.head())
```

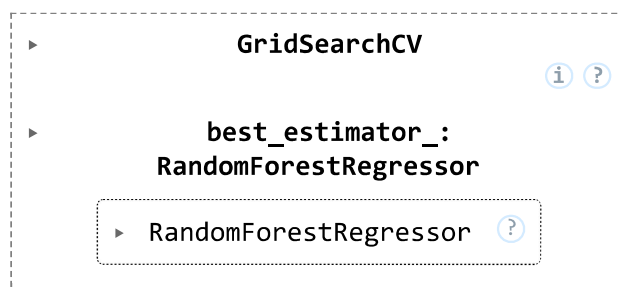
```
(614, 8)
60      2.0
618      9.0
346      1.0
294      0.0
231      6.0
Name: Pregnancies, dtype: float64
```

```
rf = RandomForestRegressor(random_state=42, n_jobs=-1)

grid_search = GridSearchCV(
    rf,
    param_grid={'n_estimators':[100], 'max_depth':[None]},
    cv=3,
    n_jobs=-1
)
```

```
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
# Train the model
rf_regressor.fit(X_train, y_train)
# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
evaluation_results
```

```
grid_search.fit(X_train, y_train)
```



```
print(hasattr(grid_search, "best_estimator_"))
```

```
True
```

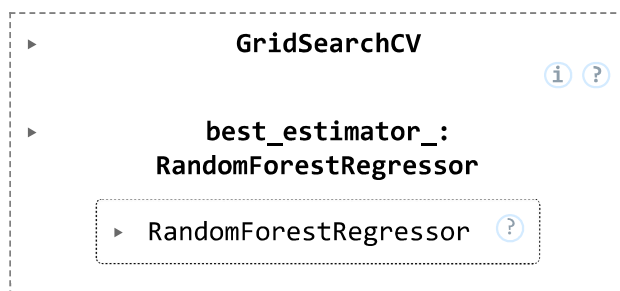
```
y_pred = best_rf.predict(X_test)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(
    rf,
    param_grid={
        "n_estimators": [100],
        "max_depth": [None]
    },
    cv=3,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
```



```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
```

```
def evaluate_model(model_name, y_test, y_pred):
    evaluation_results = {
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred)),
        "R2": r2_score(y_test, y_pred)
    }

    return evaluation_results
```

```
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
```

```

best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

print(hasattr(grid_search, "best_estimator_"))

```

```

MAE: 2.0188
MSE: 7.6249
RMSE: 2.7613
R2: 0.3969
True

```

```

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
rf =RandomForestRegressor(random_state=42)
param_distributions = {
    'n_estimators': randint(100,500),
    'max_depth': [None, 10, 20,30],
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 5)
}

```

```

# Random Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_distributions,
    n_iter=20, # number of random combinations to try
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=2
)
random_search.fit(X_train, y_train)
# Evaluate best model
best_rf_random = random_search.best_estimator_
y_pred = best_rf_random.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
print("Best Parameters:", grid_search.best_params_)

```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
MAE: 2.0423
MSE: 7.8238
RMSE: 2.7971
R2: 0.3812
Best Parameters: {'max_depth': None, 'n_estimators': 100}
```

```
from sklearn.model_selection import train_test_split
rf = RandomForestRegressor(random_state=42)
# Define parameter grid (try all combinations)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
# Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)
# Evaluate best model
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
MAE: 2.0322
MSE: 7.7258
RMSE: 2.7795
R2: 0.3890
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2,
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

```

```

from sklearn.feature_selection import SelectKBest, chi2
# Create selector (choose number of features you want, e.g., k=5)
selector = SelectKBest(score_func=chi2, k=20)
# Fit and transform
selector.fit(X, y)
print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)

```

```

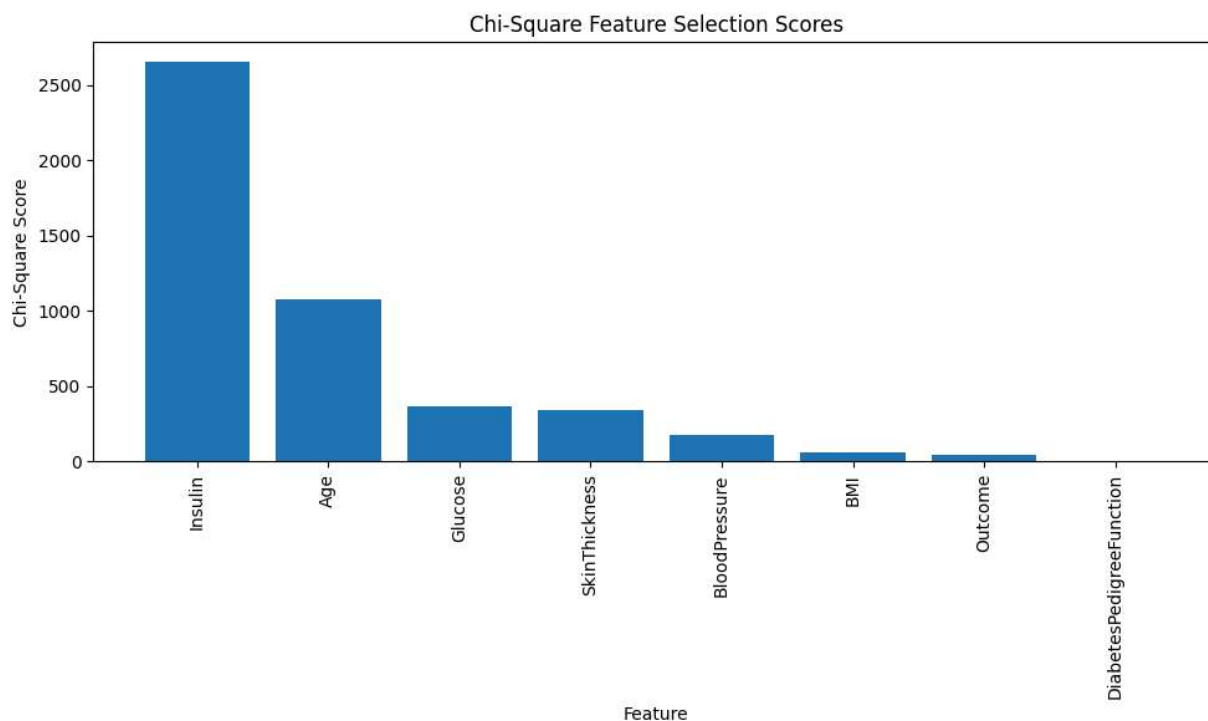
Selected Features: [ True  True  True  True  True  True  True  True]
Scores: [ 363.71569068  178.35474199  345.47449979 2652.53090823   63.0660698
         3.1325696   1078.53070143   42.05391191]
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:100: UserWarning:

```

```

import pandas as pd
import matplotlib.pyplot as plt
# Create DataFrame of features and chi-square scores
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
# Sort by score (optional, but makes plot clearer)
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
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 Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```



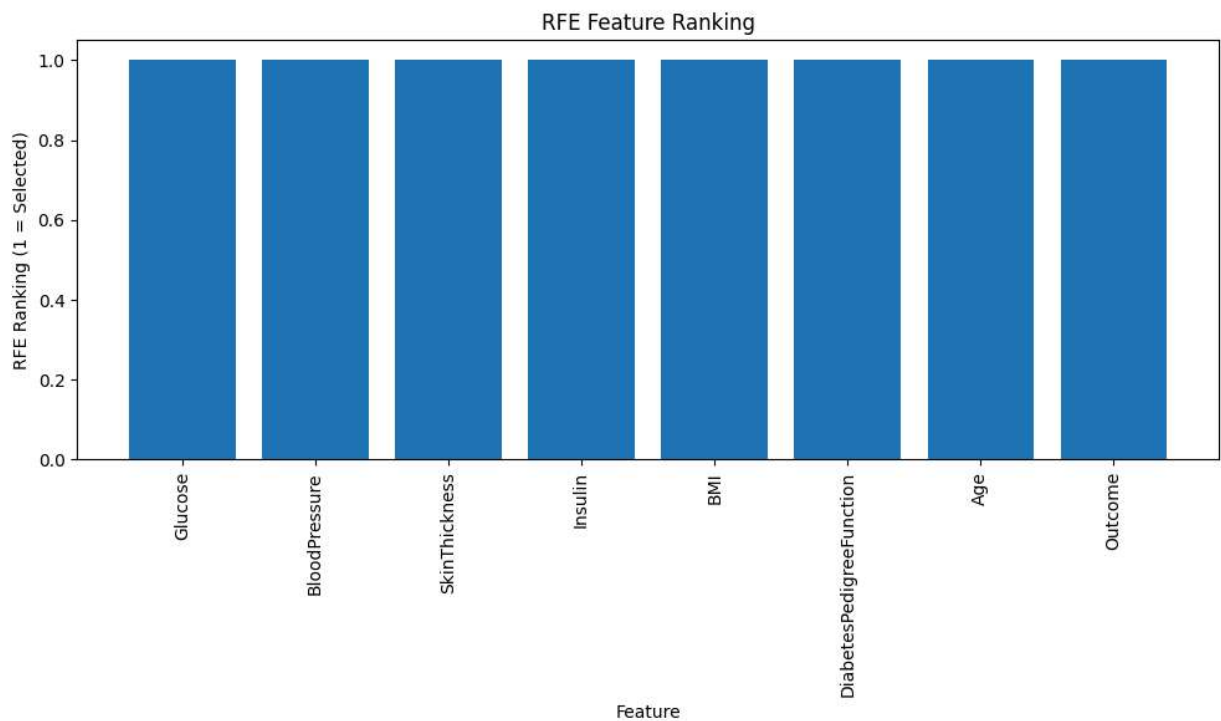
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=15)
rfe.fit(X, y)
print("Selected Features:", rfe.get_support())
print("Ranking:", rfe.ranking_)
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: U
warnings.warn(
Selected Features: [ True  True  True  True  True  True  True  True]
Ranking: [1 1 1 1 1 1 1 1]
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: C
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
 Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(

```
import pandas as pd
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()
```

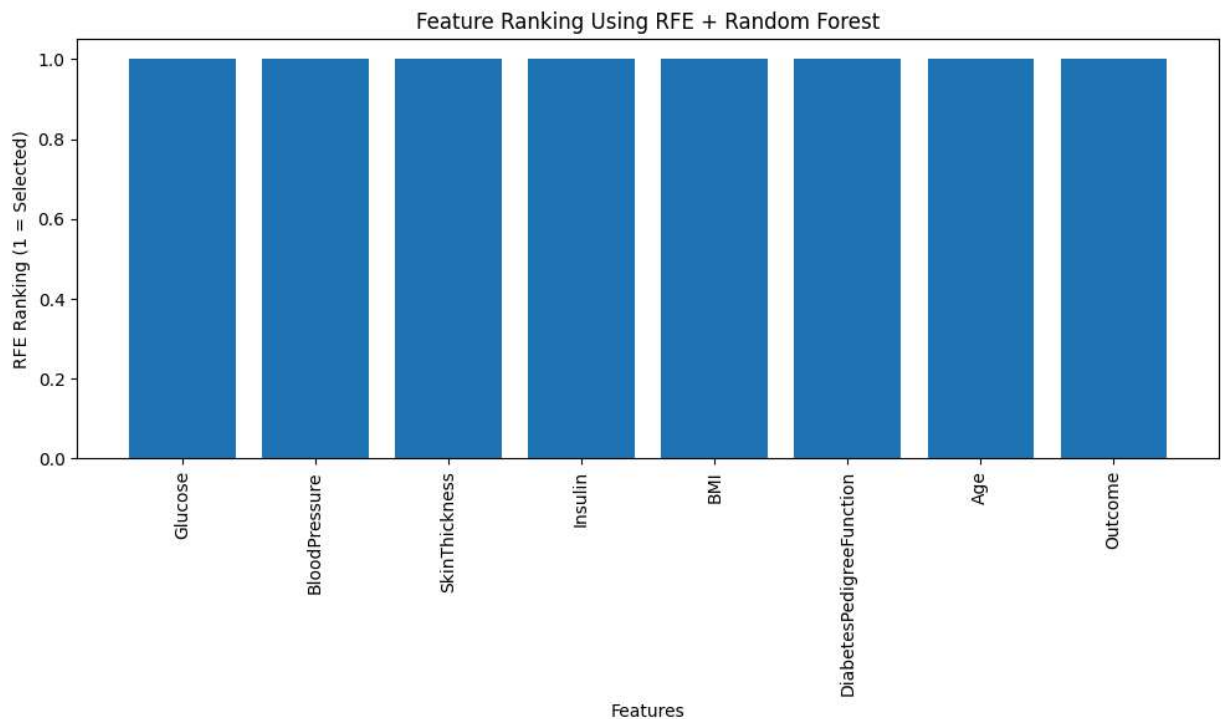


```
rf = RandomForestClassifier(n_estimators=200, random_state=42)
# Apply RFE (select top 5 features)
rfe = RFE(estimator=rf, n_features_to_select=15)
rfe.fit(X, y)
print("Selected Features Mask:", rfe.get_support())
print("Ranking of Features:", rfe.ranking_)
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: U
warnings.warn(
Selected Features Mask: [ True  True  True  True  True  True  True  True]
Ranking of Features: [1 1 1 1 1 1 1 1]
```

```
import pandas as pd
ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')
import matplotlib.pyplot as plt
```

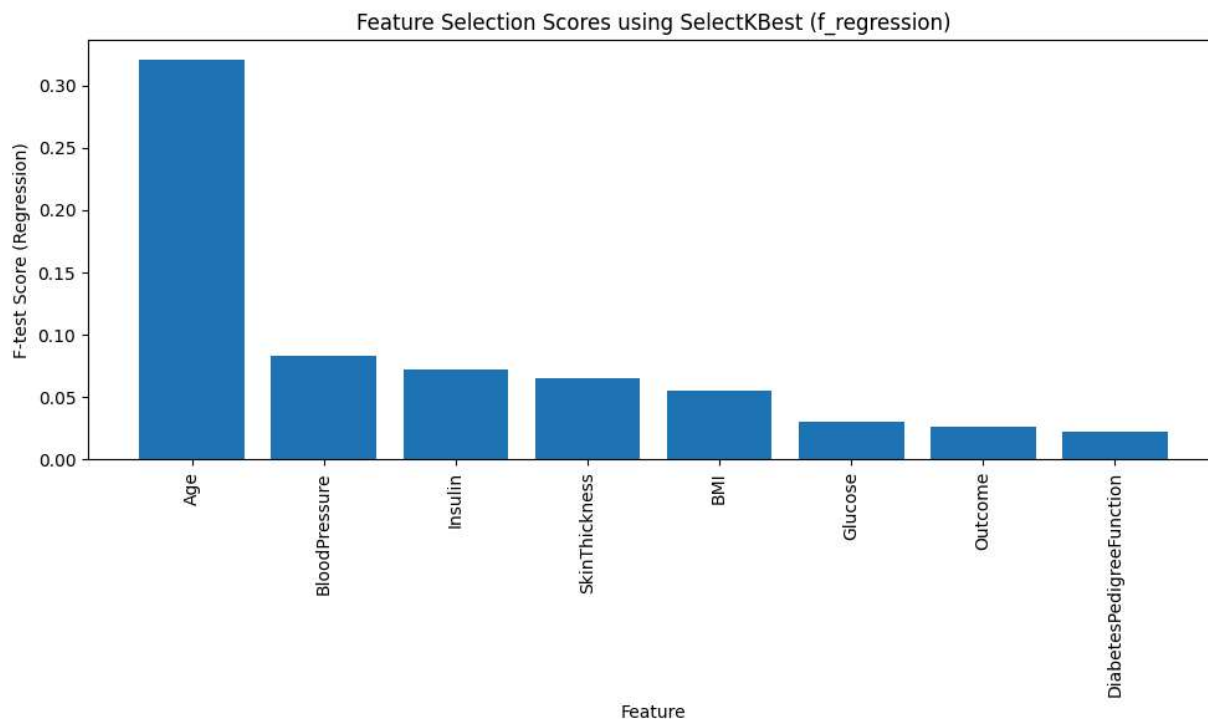
```
plt.figure(figsize=(10,6 ))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'])
plt.xlabel("Features")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking Using RFE + Random Forest")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, mutual_info_regression
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Select top k features (choose any k)
selector = SelectKBest(score_func=mutual_info_regression, k='all')
# Fit selector
selector.fit(X, y)
print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)
# Create DataFrame of features and F-test scores
data = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
})
# Sort by score for clearer plot
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("F-test Score (Regression)")
plt.title("Feature Selection Scores using SelectKBest (f_regression)")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Selected Features: [True True True True True True True True]
 Scores: [0.03023945 0.08302311 0.06486573 0.07236675 0.05570073 0.02281426
 0.32008738 0.02632625]



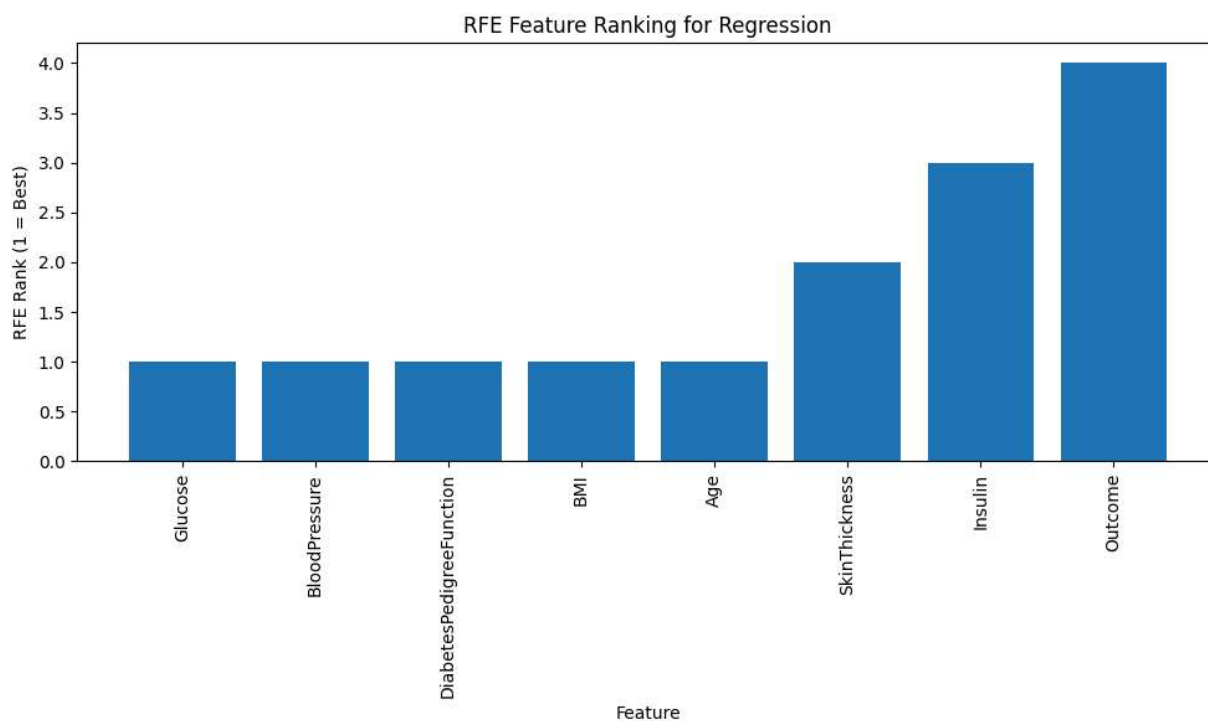
```
from sklearn.feature_selection import RFE
# Define regression model for RFE
model = RandomForestRegressor()
# RFE: choose number of features (or use all for ranking)
rfe = RFE(estimator=model, n_features_to_select=5)
# choose k features
rfe.fit(X, y)
# Get ranking (1 = selected, higher = less important)
ranking = rfe.ranking_
support = rfe.support_
print("Selected Features:", X.columns[support])
print("Feature Ranking:", ranking)
# Create DataFrame of features and RFE ranks
data = pd.DataFrame({
    'Feature': X.columns,
```

```

    'Rank': ranking
})
# Sort by rank (1 is best)
data = data.sort_values(by='Rank', ascending=True)
# Plot bar chart of feature ranks
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Rank'])
plt.xlabel("Feature")
plt.ylabel("RFE Rank (1 = Best)")
plt.title("RFE Feature Ranking for Regression")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```

Selected Features: Index(['Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFu
Feature Ranking: [1 1 2 3 1 1 4]



```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# -----
# 2. CHI-SQUARE FEATURE SELECTION
# -----
k = 15
selector = SelectKBest(chi2, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))
# 3. MODELS
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = [] # will store results
# -----
# 4. TRAIN + EVALUATE
# -----
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')
    })
# -----
# 5. RESULTS AS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

```

```

/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_se
warnings.warn(
Selected Features: ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI

Model Performance:
      Model  Accuracy  Precision    Recall

```

```

0 Random Forest 0.201299 0.216191 0.201299
1 Decision Tree 0.168831 0.176637 0.168831
2 SVM 0.129870 0.032311 0.129870
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

```

# Comparing models using features selected by REF
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# -----
# 2. RFE FEATURE SELECTION
# -----
k = 15 # number of features to select
rf_selector = RandomForestClassifier(n_estimators=200,
    random_state=42)
rfe = RFE(estimator=rf_selector, n_features_to_select=k)
# Fit RFE on training data
rfe.fit(X_train, y_train)
# Transform datasets
X_train_k = rfe.transform(X_train)
X_test_k = rfe.transform(X_test)
# Get selected features
selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))
# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = [] # to store accuracy, precision, recall
# -----
# 4. TRAIN + EVALUATE
# -----
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')
    })
# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RFE-Selected Features:")
results_df

```

```

/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: U
warnings.warn(
Selected Features by RFE: ['Glucose', 'BloodPressure', 'SkinThickness', 'Insul n

```

Model Performance Using RFE-Selected Features:

```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156 :
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156 :
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

	Model	Accuracy	Precision	Recall
0	Random Forest	0.201299	0.216191	0.201299
1	Decision Tree	0.168831	0.176637	0.168831



```
[ ] import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

```
[ ] import pandas as pd
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
```

```
[ ] import os
print(os.listdir())

[ ] ['.config', 'diabetes.csv', 'amazon.csv', 'drive', 'sample_data']
```

```
[ ] import pandas as pd

df = pd.read_csv("diabetes.csv")

df.head()
```

	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

```
[ ] df.isnull().sum()
```

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

dtype: int64

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

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...
763	101	76	48	180	32.9	0.171	63	0
764	122	70	27	0	36.8	0.340	27	0
765	121	72	23	112	26.2	0.245	30	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

```
print(df.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
      'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
X = df.drop("Pregnancies", axis=1)
y = df["Pregnancies"]
```

```
from sklearn.ensemble import RandomForestRegressor
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
```

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

```
array([[ 0.84832379,  0.14964075,  0.90726993, ...,  0.46849198,
         1.4259954 ,  1.36589591],
       [-1.12339636, -0.16054575,  0.53090156, ..., -0.36506078,
        -0.19067191, -0.73212021],
       [ 1.94372388, -0.26394125, -1.28821221, ...,  0.60439732,
        -0.10558415,  1.36589591],
       ...,
       [ 0.00330087,  0.14964075,  0.15453319, ..., -0.68519336,
        -0.27575966, -0.73212021],
       [ 0.1597866 , -0.47073225, -1.28821221, ..., -0.37110101,
         1.17073215,  1.36589591],
       [-0.8730192 ,  0.04624525,  0.65635768, ..., -0.47378505,
        -0.87137393, -0.73212021]])
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
```

```
def evaluate_model(model_name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
```

```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators=100,
                                    random_state=42)
```

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
rf = RandomForestRegressor(random_state=42)
```

```
print(X_train.dtypes)
```

```
Glucose          int64
BloodPressure    int64
SkinThickness    int64
Insulin          int64
BMI              float64
DiabetesPedigreeFunction float64
Age              int64
Outcome          int64
dtype: object
```

```
from sklearn.model_selection import train_test_split

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

```
print(df.shape)
print(df.columns)
```

```
1 1
2 8
3 1
4 0
Name: Pregnancies, dtype: int64
```

```
X = df.select_dtypes(include=["number"])

print("X shape:", X.shape)
print(X.head())
```

```
X shape: (768, 9)
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI \
0           6       148             72             35      0  33.6
1           1        85             66             29      0  26.6
2           8       183             64              0      0  23.3
3           1        89             66             23     94  28.1
4           0       137             40             35     168  43.1

DiabetesPedigreeFunction  Age  Outcome
0           0.627      50         1
1           0.351      31         0
2           0.672      32         1
3           0.167      21         0
4           2.288      33         1
```

```
X = pd.get_dummies(X)
```

```
print(X.shape)
```

```
(768, 9)
```

```
y = df["Pregnancies"].astype(str).str.replace(",", "")
y = y.astype(float)
```

```
X = pd.get_dummies(df.drop("Pregnancies", axis=1))
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

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

```
rf = RandomForestRegressor(random_state=42)
```

```
# GridSearch
grid_search = GridSearchCV(
    estimator=rf,
    param_grid={
        "n_estimators": [100, 200],
        "max_depth": [10, None]
    },
    cv=3,
    n_jobs=-1
)
```

```
y = df["Pregnancies"].astype(str).str.replace(",", "")
y = y.astype(float)
```

```
X = pd.get_dummies(df.drop("Pregnancies", axis=1))
```

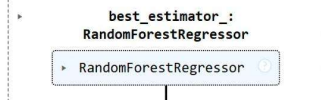
```
print(X_train.shape)
print(y_train.head())
```

```
(614, 8)
60      2.0
618     9.0
346     1.0
294     0.0
231     6.0
Name: Pregnancies, dtype: float64
```

```
rf = RandomForestRegressor(random_state=42, n_jobs=-1)

grid_search = GridSearchCV(
    rf,
    param_grid={'n_estimators':[100], 'max_depth':[None]},
    cv=3,
    n_jobs=-1
)
```

```
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
# Train the model
rf_regressor.fit(X_train, y_train)
# Make predictions
y_pred = rf_regressor.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
```



```
print(hasattr(grid_search, "best_estimator_"))
```

```
True
```

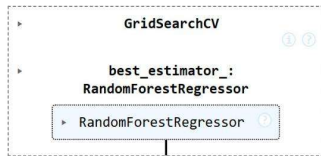
```
y_pred = best_rf.predict(X_test)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(
    rf,
    param_grid={
        "n_estimators": [100],
        "max_depth": [None]
    },
    cv=3,
    n_jobs=-1
)
```

```
grid_search.fit(X_train, y_train)
```



```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

def evaluate_model(model_name, y_test, y_pred):
    evaluation_results = {
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred)),
        "R2": r2_score(y_test, y_pred)
    }

    return evaluation_results
```

```
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)
```

```
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

print(hasattr(grid_search, "best_estimator_"))
```

```
MAE: 2.0188
MSE: 7.6249
RMSE: 2.7613
R2: 0.3969
True
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

rf = RandomForestRegressor(random_state=42)
param_distributions = {
    'n_estimators': randint(100, 500),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 5)
}
```

```
# Random Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_distributions,
    n_iter=20, # number of random combinations to try
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    random_state=42,
    verbose=2
)
random_search.fit(X_train, y_train)
# Evaluate best model
best_rf_random = random_search.best_estimator_
y_pred = best_rf_random.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
```

R2: 0.3812
Best Parameters: {'max_depth': None, 'n_estimators': 100}

```
from sklearn.model_selection import train_test_split
rf = RandomForestRegressor(random_state=42)
# Define parameter grid (try all combinations)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
# Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=3,
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)
# Evaluate best model
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

print("Best Parameters:", grid_search.best_params_)
```

Fitting 3 folds for each of 24 candidates, totalling 72 fits
MAE: 2.0322
MSE: 7.7258
RMSE: 2.7795
R2: 0.3890
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

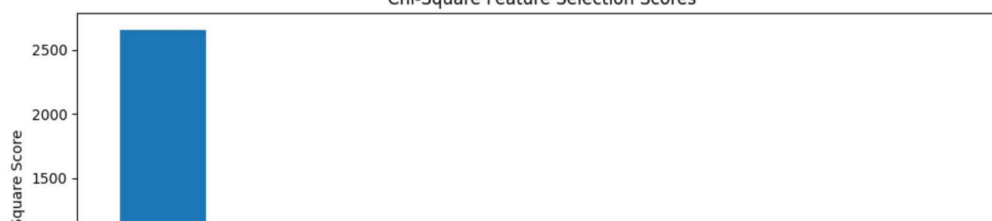
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

```
from sklearn.feature_selection import SelectKBest, chi2
# Create selector (choose number of features you want, e.g., k=5)
selector = SelectKBest(score_func=chi2, k=5)
# Fit and transform
selector.fit(X, y)
print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)
```

Selected Features: [True True True True True True True True]
Scores: [363.71569068 178.35474199 345.47449979 2652.53090823 63.0660698
3.1325696 1078.53070143 42.05391191]
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:783: UserWarning: k=20 is greater than n_features=8. All the features will be re
warnings.warn()

```
import pandas as pd
import matplotlib.pyplot as plt
# Create DataFrame of features and chi-square scores
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
# Sort by score (optional, but makes plot clearer)
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
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 Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Chi-Square Feature Selection Scores



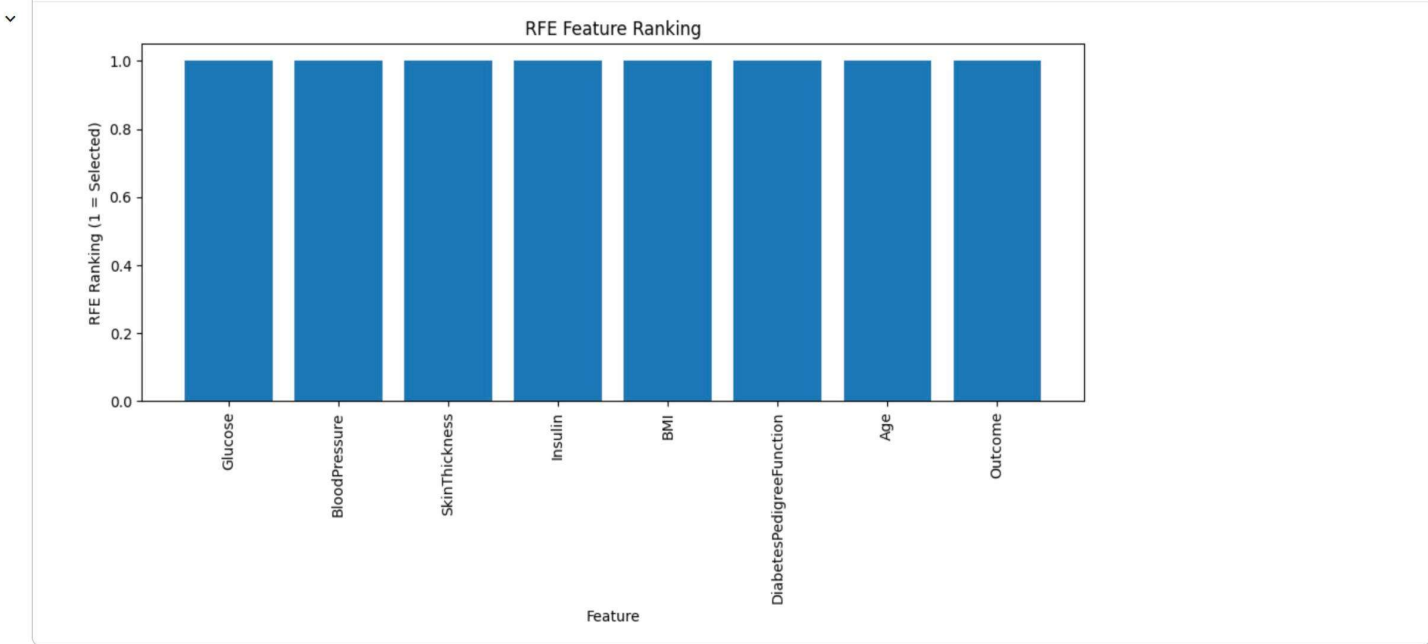
Insulin	Age	Glucose	SkinThickness	BloodPressure	BMI	Outcome	DiabetesPedigreeFunction
Feature							

```
[ ]
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_)

[ ]
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: UserWarning: Found n_features_to_select=15 > n_features=8. There will be no feature selection.
warnings.warn(
Selected Features: [ True  True  True  True  True  True  True  True]
Ranking: [1 1 1 1 1 1 1 1]
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
```

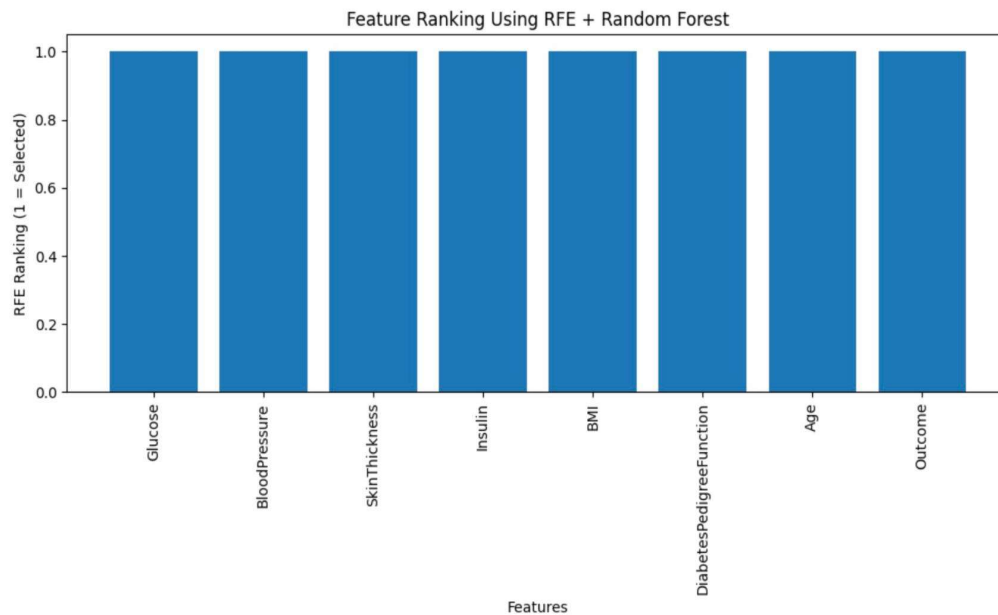
```
[ ]
import pandas as pd
feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
feature_ranking = feature_ranking.sort_values(by='Ranking',
ascending=True)
plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
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)
# Apply RFE (select top 5 features)
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_)

[ ]
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: UserWarning: Found n_features_to_select=15 > n_features=8. There will be no feature selection.
warnings.warn(
Selected Features Mask: [ True  True  True  True  True  True  True  True]
Ranking of Features: [1 1 1 1 1 1 1 1]
```

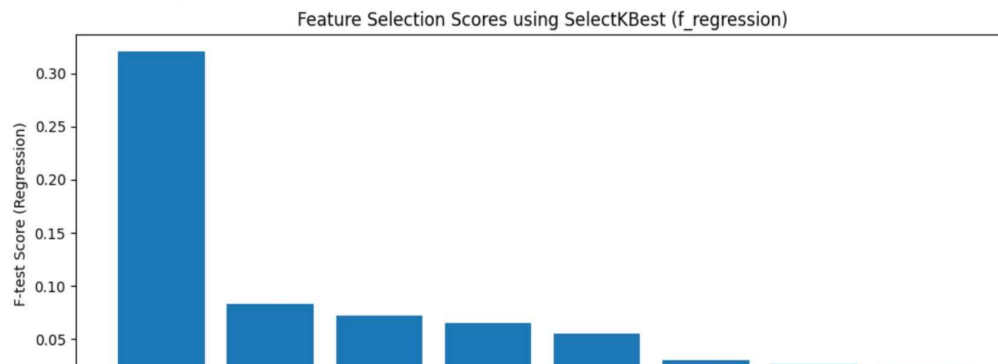
```
[ ]
import pandas as pd
ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

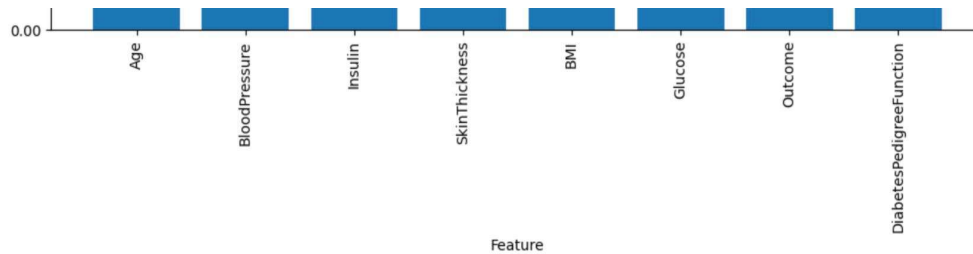


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, mutual_info_regression
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Select top k features (choose any k)
selector = SelectKBest(score_func=mutual_info_regression, k='all')
# Fit selector
selector.fit(X, y)
print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)
# Create DataFrame of features and F-test scores
data = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
})
# Sort by score for clearer plot
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("F-test Score (Regression)")
plt.title("Feature Selection Scores using SelectKBest (f_regression)")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Selected Features: [True True True True True True True True]

Scores: [0.03023945 0.08302311 0.06486573 0.07236675 0.05570073 0.02281426
0.32008738 0.02632625]

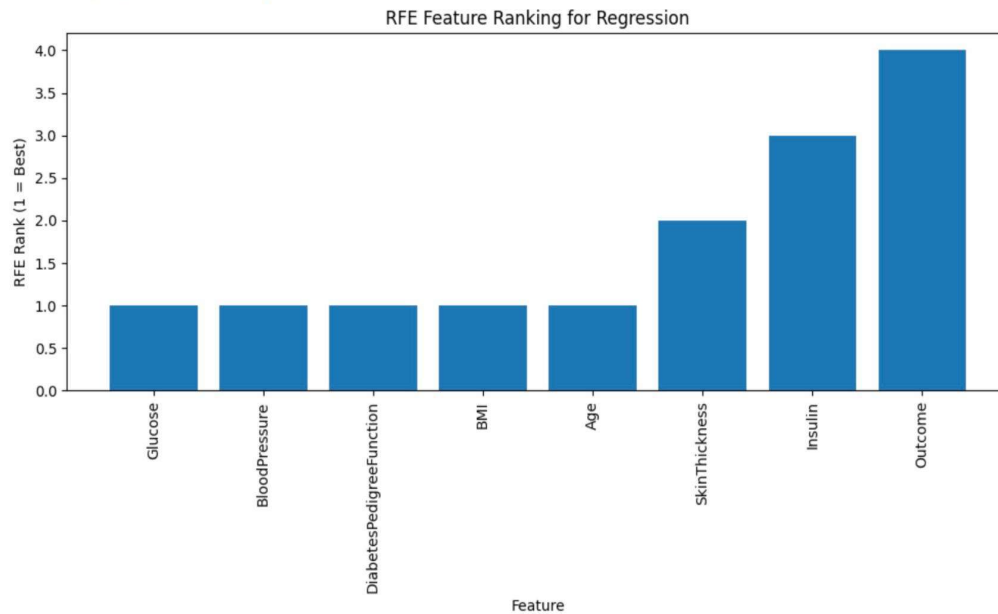




```
from sklearn.feature_selection import RFE
# Define regression model for RFE
model = RandomForestRegressor()
# RFE: choose number of features (or use all for ranking)
rfe = RFE(estimator=model, n_features_to_select=5)
# choose k features
rfe.fit(X, y)
# Get ranking (1 = selected, higher = less important)
ranking = rfe.ranking_
support = rfe.support_
print("Selected Features:", X.columns[support])
print("Feature Ranking:", ranking)
# Create DataFrame of features and RFE ranks
data = pd.DataFrame({
    'Feature': X.columns,
    'Rank': ranking
})
# Sort by rank (1 is best)
data = data.sort_values(by='Rank', ascending=True)
# Plot bar chart of feature ranks
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Rank'])
plt.xlabel("Feature")
plt.ylabel("RFE Rank (1 = Best)")
plt.title("RFE Feature Ranking for Regression")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Selected Features: Index(['Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'Age'], dtype='object')

Feature Ranking: [1 1 2 3 1 1 1 4]



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# -----
# 2. CHI-SQUARE FEATURE SELECTION
# -----
k = 15
selector = SelectKBest(chi2, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))
# 3. MODELS
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = [] # will store results
```

```

        "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')
    })

# -----
# 5. RESULTS AS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

```

/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:783: UserWarning: k=15 is greater than n_features=8. All the features will be re
warnings.warn(
Selected Features: ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.201299	0.216191	0.201299
1	Decision Tree	0.168831	0.176637	0.168831
2	SVM	0.129870	0.032311	0.129870

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

# Comparing models using features selected by REF
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# -----
# 2. RFE FEATURE SELECTION
# -----
k = 15 # number of features to select
rf_selector = RandomForestClassifier(n_estimators=200,
    random_state=42)
rfe = RFE(estimator=rf_selector, n_features_to_select=k)
# Fit RFE on training data
rfe.fit(X_train, y_train)
# Transform datasets
X_train_k = rfe.transform(X_train)
X_test_k = rfe.transform(X_test)
# Get selected features
selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))
# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = [] # to store accuracy, precision, recall
# -----
# 4. TRAIN + EVALUATE
# -----
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')
    })
# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RFE-Selected Features:")
results_df

```

/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_rfe.py:300: UserWarning: Found n_features_to_select=15 > n_features=8. There will be no feature selectio
warnings.warn(
Selected Features by RFE: ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

Model Performance Using RFE-Selected Features:

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

	Model	Accuracy	Precision	Recall
0	Random Forest	0.201299	0.216191	0.201299
1	Decision Tree	0.168831	0.176637	0.168831
2	SVM	0.129870	0.032311	0.129870

```

# Comparing models using features selected by the Random Forest Feature Importance
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

```

# Sort indices by importance (descending)
sorted_idx = np.argsort(importances[::-1])
# Select top-k feature indices
top_k_idx = sorted_idx[:k]
# Get feature names
selected_features = X.columns[top_k_idx]
print("Selected Features by RF Importance:", list(selected_features))
# Reduce datasets to selected features
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]
# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = []
# -----
# 4. TRAIN + EVALUATE
# -----
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')
    })
# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df

```

... Selected Features by RF Importance: ['Age', 'BMI', 'DiabetesPedigreeFunction', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'Outcome']

Model Performance Using RF-Selected Features:

```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no pr
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

	Model	Accuracy	Precision	Recall
0	Random Forest	0.175325	0.177193	0.175325
1	Decision Tree	0.175325	0.186338	0.175325
2	SVM	0.129870	0.032311	0.129870

