

```
In [1]: # New Python Project - Breast Cancer - Walid Saleh
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [2]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
In [3]: df = pd.read_csv('/content/drive/MyDrive/Python/breast-cancer.csv')
```

```
In [4]: df.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
0	842302	M	17.99	10.38	122.80	1001.0
1	842517	M	20.57	17.77	132.90	1326.0
2	84300903	M	19.69	21.25	130.00	1203.0
3	84348301	M	11.42	20.38	77.58	386.1
4	84358402	M	20.29	14.34	135.10	1297.0

5 rows × 7 columns

```
In [5]: df.dtypes
```

Out[5]:

0

id	int64
diagnosis	object
radius_mean	float64
texture_mean	float64
perimeter_mean	float64
area_mean	float64
smoothness_mean	float64
compactness_mean	float64
concavity_mean	float64
concave points_mean	float64
symmetry_mean	float64
fractal_dimension_mean	float64
radius_se	float64
texture_se	float64
perimeter_se	float64
area_se	float64
smoothness_se	float64
compactness_se	float64
concavity_se	float64
concave points_se	float64
symmetry_se	float64
fractal_dimension_se	float64
radius_worst	float64
texture_worst	float64
perimeter_worst	float64
area_worst	float64
smoothness_worst	float64
compactness_worst	float64
concavity_worst	float64
concave points_worst	float64
symmetry_worst	float64
fractal_dimension_worst	float64

dtype: object

In [6]: `df.isnull().sum()`

Out[6]:

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

```
In [7]: # Encode the 'diagnosis' column (M=1, B=0)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Drop the 'id' column
df_processed = df.drop('id', axis=1)

# Separate features (X) and target (y)
X = df_processed.drop('diagnosis', axis=1)
y = df_processed['diagnosis']

print("Shape of features (X):", X.shape)
print("Shape of target (y):", y.shape)
print("First 5 rows of encoded diagnosis:")
display(y.head())
```

Shape of features (X): (569, 30)
Shape of target (y): (569,)
First 5 rows of encoded diagnosis:

diagnosis	
0	1
1	1
2	1
3	1
4	1

dtype: int64

```
In [8]: from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

X_train shape: (398, 30)
X_test shape: (171, 30)
y_train shape: (398,)
y_test shape: (171,)

```
In [12]: from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and tes
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
print("Shape of scaled X_train:", X_train_scaled.shape)
print("Shape of scaled X_test:", X_test_scaled.shape)
print("First 5 rows of scaled X_train (features):\n", X_train_scaled[:5])
```

Shape of scaled X_train: (398, 30)

Shape of scaled X_test: (171, 30)

First 5 rows of scaled X_train (features):

```
[[ 1.70581999  1.04974466  2.12129465  1.66933676  2.27645789  4.54429486
   3.53913939  2.84839268  4.03238962  2.63977264  1.88658218  1.20005314
   2.87275048  1.74921741  1.19777257  2.41558743  2.3247057  2.29784843
   6.94476933  0.82740765  1.45951037  0.94958949  1.86817555  1.3068397
   1.33538327  2.31385505  2.34642696  2.04939907  3.94428119  0.86655367]
 [ 0.53638811 -0.90994477  0.5690952  0.39031555 -0.06586788  0.64494486
   0.39354254  0.55700173 -0.11678098 -0.40152959 -0.25041371 -0.84442436
  -0.11857378 -0.16149227 -0.37614552  0.40854639  0.14771626  0.48165509
  -0.68243869 -0.23809862  0.50829368 -0.64531469  0.50191659  0.33008672
   0.47701253  1.24209126  1.05257355  1.25584994  0.16797759  0.40524244]
 [-0.13145612 -0.99333581 -0.14884823 -0.21154347 -0.91616898 -0.50857386
  -0.56752982 -0.60664497 -0.07986666 -0.51245428 -0.95807347 -1.2145298
  -0.87783349 -0.66531468 -0.59346476  0.26263059 -0.23715248 -0.51677077
  -0.26595611  0.41139177 -0.36245131 -1.02848469 -0.28041517 -0.39938703
  -0.19621941  0.36866098 -0.08079455 -0.14001353  0.74198821  0.68752098]
 [ 0.92807734  1.41805508  0.92540484  0.82854593  0.32973419  0.22374766
   0.98318775  0.44394722 -1.86652019 -0.55598169  0.44143242  0.80819795
   0.44689582  0.41909869  8.38975683  3.37717812  3.37462278  4.43435806
   0.15764583  3.27108269  0.36592069  0.35190979  0.38608159  0.24353898
  -0.3308658  -0.46472119  0.03031066 -0.07225317 -2.04071547 -0.86116674]
 [ 0.14469888 -0.97248805  0.15831525 -0.00969706  1.20575985  0.58504125
   0.13770066  0.56285816  1.11615761  0.1179911  0.37638091 -0.86311837
   0.47025002  0.05663157 -0.49087067  1.06097449  0.36586906  0.74431445
   0.45665897  0.40135385  0.00895651 -1.2577317  0.05936749 -0.14045035
  -0.18780401  0.35912571 -0.06417625  0.37647101  0.3738727  0.0477262
  3]]
```

```
In [13]: from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Define models and their parameter grids
models = {
    'LogisticRegression': {
        'model': LogisticRegression(solver='liblinear', random_state=42),
        'params': {
            'C': [0.001, 0.01, 0.1, 1, 10, 100]
        }
    },
    'SVC': {
        'model': SVC(probability=True, random_state=42),
        'params': {
            'C': [0.1, 1, 10],
            'kernel': ['linear', 'rbf']
        }
    },
    'RandomForestClassifier': {
        'model': RandomForestClassifier(random_state=42),
        'params': {
            'n_estimators': [100, 200, 300],
```

```

        'max_depth': [None, 10, 20]
    }
}

# Dictionary to store best models and their performance
best_models = {}
model_performance = {}

```

```

In [14]: for model_name, config in models.items():
    print(f"\n--- Training and evaluating {model_name} ---")
    grid_search = GridSearchCV(config['model'], config['params'], cv=5, s
    grid_search.fit(X_train_scaled, y_train)

    best_model = grid_search.best_estimator_
    best_models[model_name] = best_model

    y_pred = best_model.predict(X_test_scaled)
    y_proba = best_model.predict_proba(X_test_scaled)[: , 1] if hasattr(be

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba) if y_proba is not None else

    model_performance[model_name] = {
        'best_params': grid_search.best_params_,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'roc_auc_score': roc_auc
    }

    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"ROC AUC Score: {roc_auc:.4f}")

print("\n--- Model Performance Summary ---")
for model_name, metrics in model_performance.items():
    print(f"\n{model_name}:")
    for metric, value in metrics.items():
        print(f"    {metric}: {value}")

```

```

--- Training and evaluating LogisticRegression ---
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for LogisticRegression: {'C': 1}
Accuracy: 0.9708
Precision: 0.9836
Recall: 0.9375
F1-Score: 0.9600
ROC AUC Score: 0.9975

--- Training and evaluating SVC ---
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for SVC: {'C': 10, 'kernel': 'rbf'}
Accuracy: 0.9708
Precision: 0.9836
Recall: 0.9375
F1-Score: 0.9600
ROC AUC Score: 0.9955

--- Training and evaluating RandomForestClassifier ---
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best parameters for RandomForestClassifier: {'max_depth': None, 'n_estimators': 100}
Accuracy: 0.9649
Precision: 1.0000
Recall: 0.9062
F1-Score: 0.9508
ROC AUC Score: 0.9968

--- Model Performance Summary ---

LogisticRegression:
  best_params: {'C': 1}
  accuracy: 0.9707602339181286
  precision: 0.9836065573770492
  recall: 0.9375
  f1_score: 0.96
  roc_auc_score: 0.997517523364486

SVC:
  best_params: {'C': 10, 'kernel': 'rbf'}
  accuracy: 0.9707602339181286
  precision: 0.9836065573770492
  recall: 0.9375
  f1_score: 0.96
  roc_auc_score: 0.9954731308411214

RandomForestClassifier:
  best_params: {'max_depth': None, 'n_estimators': 100}
  accuracy: 0.9649122807017544
  precision: 1.0
  recall: 0.90625
  f1_score: 0.9508196721311475
  roc_auc_score: 0.9967873831775702

```

```

In [15]: # Get the best Random Forest model from GridSearchCV
rf_model = best_models['RandomForestClassifier']

# Get feature importances
feature_importances = rf_model.feature_importances_

```

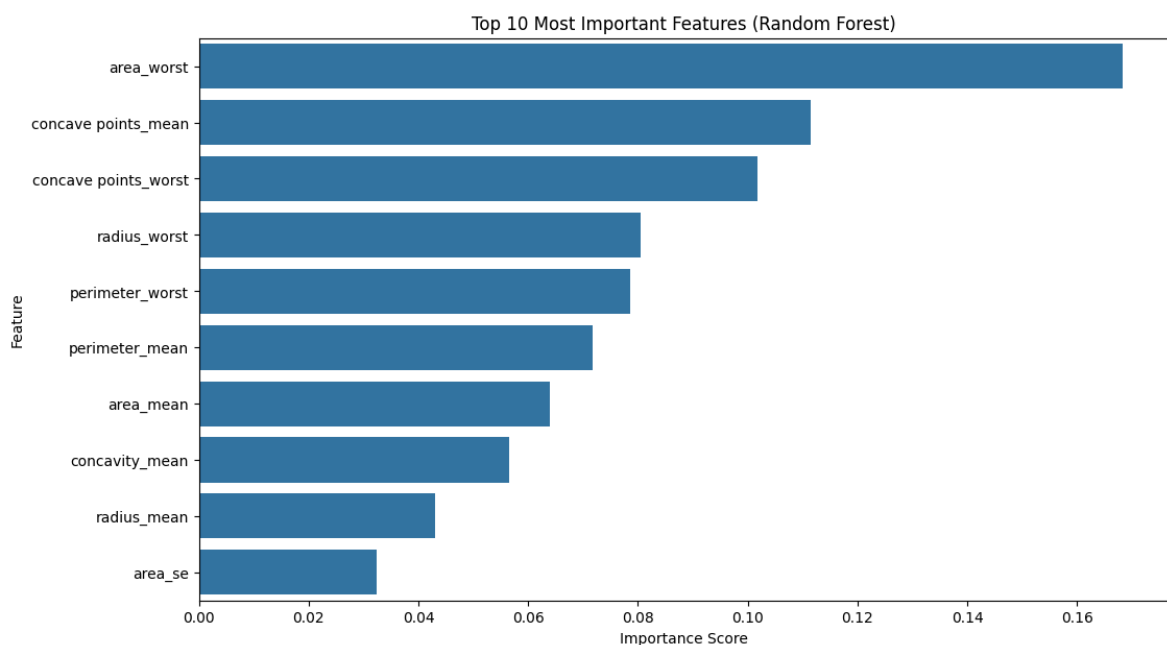
```
# Create a pandas Series for better visualization
features_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_i
features_df = features_df.sort_values(by='Importance', ascending=False)

print("Top 10 features by Random Forest Importance:")
display(features_df.head(10))

# Visualize feature importances
plt.figure(figsize=(12, 7))
sns.barplot(x='Importance', y='Feature', data=features_df.head(10))
plt.title('Top 10 Most Important Features (Random Forest)')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()
```

Top 10 features by Random Forest Importance:

	Feature	Importance
23	area_worst	0.168343
7	concave points_mean	0.111414
27	concave points_worst	0.101730
20	radius_worst	0.080520
22	perimeter_worst	0.078514
2	perimeter_mean	0.071684
3	area_mean	0.063877
6	concavity_mean	0.056572
0	radius_mean	0.043004
13	area_se	0.032464



```
In [16]: from sklearn.feature_selection import SelectKBest, f_classif

# Select the top 10 features based on f_classif scores
```

```

selector = SelectKBest(score_func=f_classif, k=10)
X_new = selector.fit_transform(X_train_scaled, y_train)

# Get the selected feature names
selected_features_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_features_indices]

# Get the scores for selected features
feature_scores = selector.scores_[selected_features_indices]

# Create a DataFrame for better visualization
kbest_features_df = pd.DataFrame({'Feature': selected_features, 'Score':
kbest_features_df = kbest_features_df.sort_values(by='Score', ascending=F

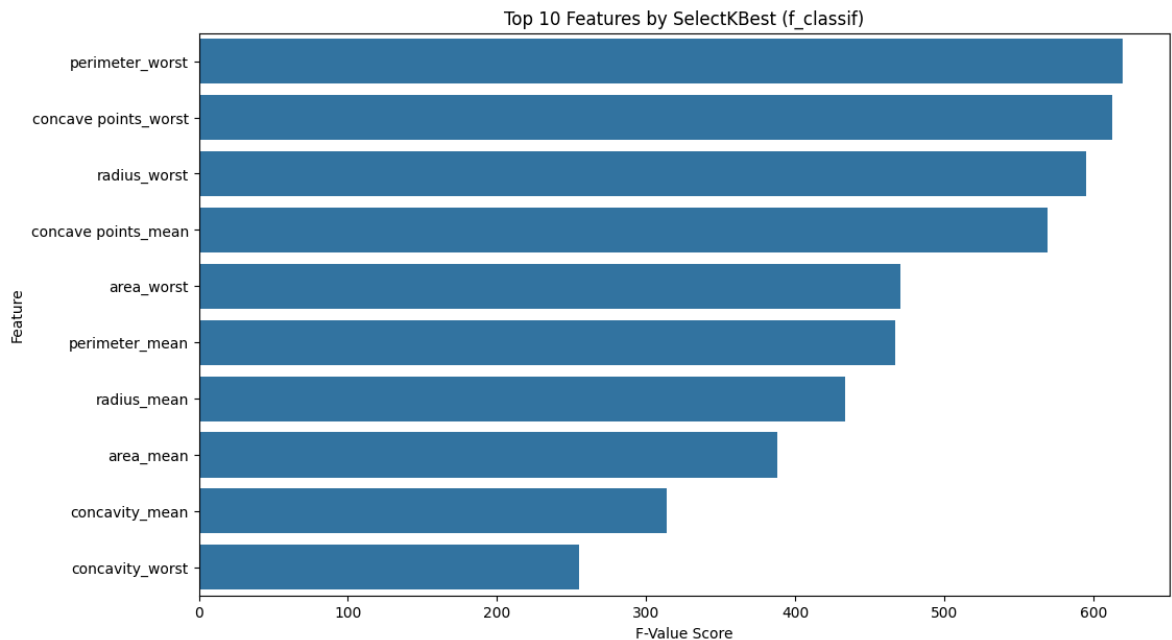
print("Top 10 features by SelectKBest (f_classif scores):")
display(kbest_features_df)

# Visualize selected features by score
plt.figure(figsize=(12, 7))
sns.barplot(x='Score', y='Feature', data=kbest_features_df)
plt.title('Top 10 Features by SelectKBest (f_classif)')
plt.xlabel('F-Value Score')
plt.ylabel('Feature')
plt.show()

```

Top 10 features by SelectKBest (f_classif scores):

	Feature	Score
6	perimeter_worst	619.823294
9	concave points_worst	612.841093
5	radius_worst	595.042825
4	concave points_mean	569.210242
7	area_worst	470.875421
1	perimeter_mean	467.003005
0	radius_mean	433.308866
2	area_mean	387.891499
3	concavity_mean	313.904115
8	concavity_worst	254.857117



```
In [17]: # Get the list of selected features from kbest_features_df
selected_features = kbest_features_df['Feature'].tolist()

# Create new training and testing sets with only the selected features
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

print("Shape of X_train with selected features:", X_train_selected.shape)
print("Shape of X_test with selected features:", X_test_selected.shape)
print("Selected features:", selected_features)
```

```
Shape of X_train with selected features: (398, 10)
Shape of X_test with selected features: (171, 10)
Selected features: ['perimeter_worst', 'concave points_worst', 'radius_worst', 'concave points_mean', 'area_worst', 'perimeter_mean', 'radius_mean', 'area_mean', 'concavity_mean', 'concavity_worst']
```

```
In [18]: # Re-initialize and fit the StandardScaler on the selected features
scaler_selected = StandardScaler()

X_train_scaled_selected = scaler_selected.fit_transform(X_train_selected)
X_test_scaled_selected = scaler_selected.transform(X_test_selected)

print("Shape of scaled X_train with selected features:", X_train_scaled_selected.shape)
print("Shape of scaled X_test with selected features:", X_test_scaled_selected.shape)
```

```
Shape of scaled X_train with selected features: (398, 10)
Shape of scaled X_test with selected features: (171, 10)
```

```
In [19]: # Dictionary to store best models and their performance with selected features
best_models_selected_features = {}
model_performance_selected_features = {}

for model_name, config in models.items():
    print(f"\n--- Training and evaluating {model_name} with selected features")
    grid_search = GridSearchCV(config['model'], config['params'], cv=5, scoring='f1')
    grid_search.fit(X_train_scaled_selected, y_train)

    best_model = grid_search.best_estimator_
    best_models_selected_features[model_name] = best_model
```

```

y_pred = best_model.predict(X_test_scaled_selected)
y_proba = best_model.predict_proba(X_test_scaled_selected)[:, 1] if h

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_proba) if y_proba is not None else

model_performance_selected_features[model_name] = {
    'best_params': grid_search.best_params_,
    'accuracy': accuracy,
    'precision': precision,
    'recall': recall,
    'f1_score': f1,
    'roc_auc_score': roc_auc
}

print(f"Best parameters for {model_name}: {grid_search.best_params_}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC AUC Score: {roc_auc:.4f}")

print("\n--- Model Performance Summary with Selected Features ---")
for model_name, metrics in model_performance_selected_features.items():
    print(f"\n{model_name}:")
    for metric, value in metrics.items():
        print(f"    {metric}: {value}")

```

```

--- Training and evaluating LogisticRegression with selected features ---
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for LogisticRegression: {'C': 10}
Accuracy: 0.9766
Precision: 1.0000
Recall: 0.9375
F1-Score: 0.9677
ROC AUC Score: 0.9988

--- Training and evaluating SVC with selected features ---
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for SVC: {'C': 10, 'kernel': 'linear'}
Accuracy: 0.9766
Precision: 1.0000
Recall: 0.9375
F1-Score: 0.9677
ROC AUC Score: 0.9987

--- Training and evaluating RandomForestClassifier with selected features
---
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best parameters for RandomForestClassifier: {'max_depth': None, 'n_estimators': 100}
Accuracy: 0.9532
Precision: 0.9667
Recall: 0.9062
F1-Score: 0.9355
ROC AUC Score: 0.9923

--- Model Performance Summary with Selected Features ---

LogisticRegression:
  best_params: {'C': 10}
  accuracy: 0.9766081871345029
  precision: 1.0
  recall: 0.9375
  f1_score: 0.967741935483871
  roc_auc_score: 0.9988317757009346

SVC:
  best_params: {'C': 10, 'kernel': 'linear'}
  accuracy: 0.9766081871345029
  precision: 1.0
  recall: 0.9375
  f1_score: 0.967741935483871
  roc_auc_score: 0.9986857476635513

RandomForestClassifier:
  best_params: {'max_depth': None, 'n_estimators': 100}
  accuracy: 0.9532163742690059
  precision: 0.9666666666666667
  recall: 0.90625
  f1_score: 0.9354838709677419
  roc_auc_score: 0.9923335280373832

```

In [20]: !jupyter nbconvert --to html /content/drive/MyDrive/Python/WalidSaleh_Bre

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Python/WalidSaleh_Breast_Cancer_Proj.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 2 image(s).
[NbConvertApp] Writing 456934 bytes to /content/drive/MyDrive/Python/WalidSaleh_Breast_Cancer_Proj.html
```

In [22]: **import** os

```
notebook_path = '/content/drive/MyDrive/Python/WalidSaleh_Breast_Cancer_P
output_dir = os.path.dirname(notebook_path)
```

```
!jupyter nbconvert --to html {notebook_path} --output-dir="{output_dir}"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Python/WalidSaleh_Breast_Cancer_Proj.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 2 image(s).
[NbConvertApp] Writing 463388 bytes to /content/drive/MyDrive/Python/WalidSaleh_Breast_Cancer_Proj.html
```

In []: