```
import pandas as pd
import numpy as np
```

```
df=pd.read_csv("/content/breast_cancer.csv")
df
```

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | c points |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0 |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0 |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0 |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0 |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **564** | 926424 | M | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 | 0.24390 | 0 |
| **565** | 926682 | M | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 | 0.14400 | 0 |
| **566** | 926954 | M | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 | 0.09251 | 0 |
| **567** | 927241 | M | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 | 0.35140 | 0 |
| **568** | 92751 | B | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 | 0.00000 | 0 |

569 rows × 32 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 |

```
df.drop(columns=["id"], inplace=True)
```

```
df_copy=df.copy()
```

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

```
X = df_copy.drop(columns=['diagnosis'])
y = df_copy['diagnosis']
```

```
X
```

| | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean | concavity_mean | concave points_mean | symmetry_mean |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0.14710 | 0.2419 |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0.07017 | 0.1812 |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0.12790 | 0.2069 |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0.10520 | 0.2597 |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0.10430 | 0.1809 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **564** | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 | 0.24390 | 0.13890 | 0.1726 |
| **565** | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 | 0.14400 | 0.09791 | 0.1752 |
| **566** | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 | 0.09251 | 0.05302 | 0.1590 |
| **567** | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 | 0.35140 | 0.15200 | 0.2397 |
| **568** | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 | 0.00000 | 0.00000 | 0.1587 |

569 rows × 30 columns

```
y
```

| | diagnosis |
|---|---|
| **0** | 1 |
| **1** | 1 |
| **2** | 1 |
| **3** | 1 |
| **4** | 1 |
| **...** | ... |
| **564** | 1 |
| **565** | 1 |
| **566** | 1 |
| **567** | 1 |
| **568** | 0 |

569 rows × 1 columns

## Task1:

### Informaion Gain for Feature Selection

```
from sklearn.feature_selection import mutual_info_classif
```

```
# Compute Information Gain using Mutual Information
info_gain = mutual_info_classif(X, y, random_state=42)
feature_importance = pd.Series(info_gain, index=X.columns)
```

```
# Sort feature importance in descending order
feature_importance = feature_importance.sort_values(ascending=False)
```

```
feature_importance
```

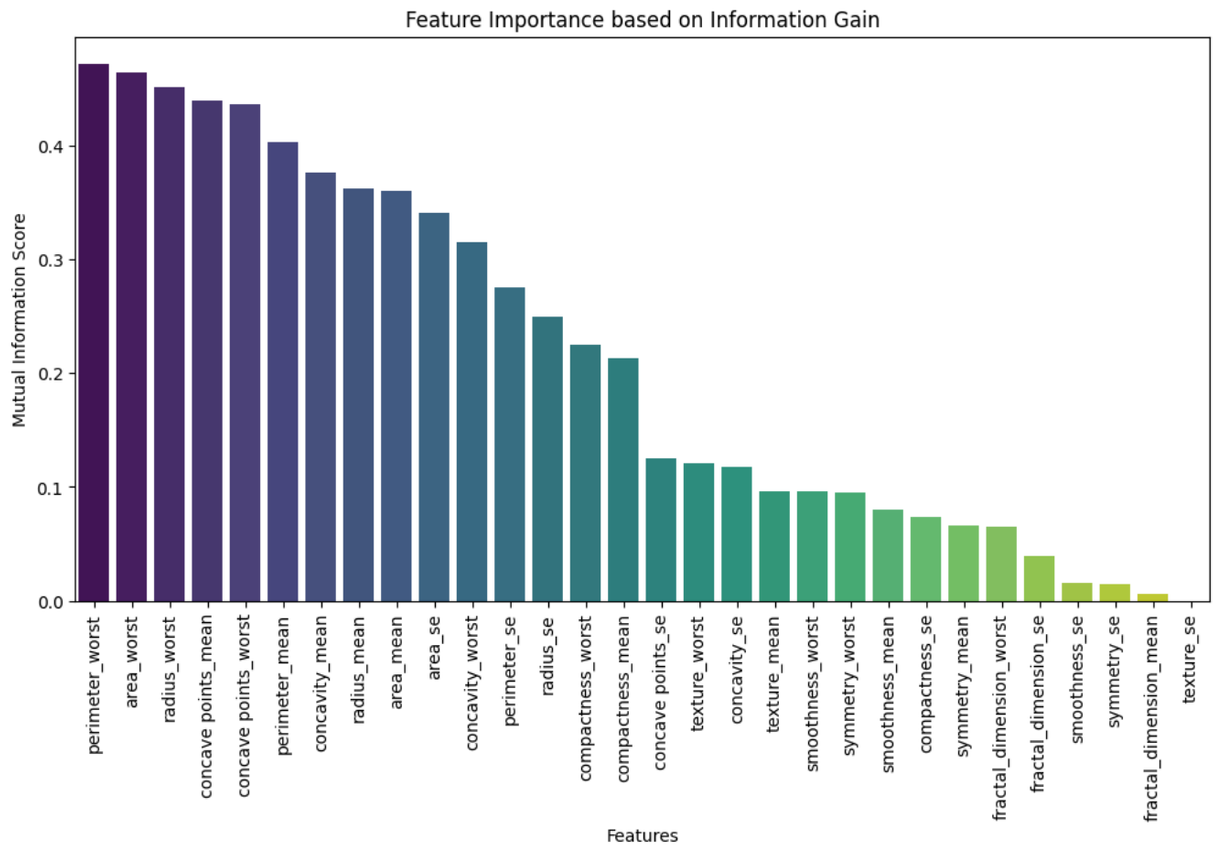
| | 0 |
|---|---|
| perimeter_worst | 0.471842 |
| area_worst | 0.464313 |
| radius_worst | 0.451230 |
| concave points_mean | 0.438806 |
| concave points_worst | 0.436255 |
| perimeter_mean | 0.402361 |
| concavity_mean | 0.375447 |
| radius_mean | 0.362276 |
| area_mean | 0.360023 |
| area_se | 0.340759 |
| concavity_worst | 0.315259 |
| perimeter_se | 0.275614 |
| radius_se | 0.249301 |
| compactness_worst | 0.225211 |
| compactness_mean | 0.213439 |
| concave points_se | 0.125415 |
| texture_worst | 0.120331 |
| concavity_se | 0.117440 |
| texture_mean | 0.096540 |
| smoothness_worst | 0.095697 |
| symmetry_worst | 0.095435 |
| smoothness_mean | 0.079740 |
| compactness_se | 0.073390 |
| symmetry_mean | 0.065721 |
| fractal_dimension_worst | 0.065041 |
| fractal_dimension_se | 0.039235 |
| smoothness_se | 0.015651 |
| symmetry_se | 0.014216 |
| fractal_dimension_mean | 0.005888 |
| texture_se | 0.000000 |

```
import matplotlib.pyplot as plt
import seaborn as sns
# Visualize feature importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importance.index, y=feature_importance.values, palette="viridis")
plt.xticks(rotation=90)
plt.xlabel("Features")
plt.ylabel("Mutual Information Score")
plt.title("Feature Importance based on Information Gain")
plt.show()
```

```
<ipython-input-20-6acca2b7fa38>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x=feature_importance.index, y=feature_importance.values, palette="viridis")
```




```
# Select top 5 most important features
top_5_features = feature_importance.index[:5]
X_top5 = X[top_5_features]
```

```
X_top5
```

| | perimeter_worst | area_worst | radius_worst | concave points_mean | concave points_worst |
|---|---|---|---|---|---|
| **0** | 184.60 | 2019.0 | 25.380 | 0.14710 | 0.2654 |
| **1** | 158.80 | 1956.0 | 24.990 | 0.07017 | 0.1860 |
| **2** | 152.50 | 1709.0 | 23.570 | 0.12790 | 0.2430 |
| **3** | 98.87 | 567.7 | 14.910 | 0.10520 | 0.2575 |
| **4** | 152.20 | 1575.0 | 22.540 | 0.10430 | 0.1625 |
| **...** | ... | ... | ... | ... | ... |
| **564** | 166.10 | 2027.0 | 25.450 | 0.13890 | 0.2216 |
| **565** | 155.00 | 1731.0 | 23.690 | 0.09791 | 0.1628 |
| **566** | 126.70 | 1124.0 | 18.980 | 0.05302 | 0.1418 |
| **567** | 184.60 | 1821.0 | 25.740 | 0.15200 | 0.2650 |
| **568** | 59.16 | 268.6 | 9.456 | 0.00000 | 0.0000 |

569 rows × 5 columns

Next steps: Generate code with X_top5 | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

```
X_train, X_test, y_train, y_test=train_test_split(X_top5, y, test_size=0.2, random_state=42)
```

```
logreg=LogisticRegression()
```

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

```
/usr/local/lib/python3.11/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(
```

▾ LogisticRegression

LogisticRegression()

```
y_predict = logreg.predict(X_test)
```

```
ac=accuracy_score(y_test, y_predict)
```

```
print(f"Model Accuracy using top 5 features: {ac*100:.4f}")
```

```
Model Accuracy using top 5 features: 95.6140
```

## Task2:

### Variance Threshold for Feature selection

```
from sklearn.feature_selection import VarianceThreshold
```

```
var_thresh = VarianceThreshold(threshold=0.01)  # Remove features with near-zero variance
X_var_selected = var_thresh.fit_transform(X)
```

```
# Get retained feature names
retained_features = X.columns[var_thresh.get_support()]
print(len(retained_features))
print("Retained Features after Variance Thresholding:", list(retained_features))
```

```
14
Retained Features after Variance Thresholding: ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'radius_se', 'texture_se',
```

```
X_train_var, X_test_var, y_train_var, y_test_var = train_test_split(X_var_selected, y, test_size=0.2, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
model_var = RandomForestClassifier(random_state=42)
```

```
model_var.fit(X_train_var, y_train_var)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred_var = model_var.predict(X_test_var)
```

```
accuracy_var = accuracy_score(y_test_var, y_pred_var)
print(f"Model Accuracy using Variance Threshold selected features: {accuracy_var*100:.4f}")
```

```
Model Accuracy using Variance Threshold selected features: 97.3684
```

## Task3:

### Forward Feature Selection using Logistic Regression

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

```
log_reg = LogisticRegression(max_iter=5000, random_state=42)
```

```
sfs = SFS(log_reg,
          k_features='best',
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=5)
```

```
sfs.fit(X, y)
```

```
SequentialFeatureSelector
estimator: LogisticRegression
LogisticRegression
```

```
# Selected features
selected_features = list(sfs.k_feature_names_)
print("Selected Features from Forward Selection:", selected_features)
```

```
Selected Features from Forward Selection: ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness
```

```
print(len(selected_features))
```

```
25
```

```
X_sfs_selected = X[selected_features]
```

```
X_train_sfs, X_test_sfs, y_train_sfs, y_test_sfs = train_test_split(X_sfs_selected, y, test_size=0.2, random_state=42)
```

```
log_reg.fit(X_train_sfs, y_train_sfs)
```

```
LogisticRegression
LogisticRegression(max_iter=5000, random_state=42)
```

```
y_pred_sfs = log_reg.predict(X_test_sfs)
```

```
accuracy_sfs = accuracy_score(y_test_sfs, y_pred_sfs)
print(f"Model Accuracy using Forward Feature Selection: {accuracy_sfs*100:.4f}")
```

```
Model Accuracy using Forward Feature Selection: 96.4912
```

## Task4:

### Recursive Feature Elimination (RFE) for Feature Selection

```
from sklearn.feature_selection import RFE
```

```
rfe = RFE(log_reg, n_features_to_select=7)
rfe.fit(X, y)
```

```
RFE
estimator: LogisticRegression
LogisticRegression
```

```
rfe_selected_features = X.columns[rfe.support_]
print(len(rfe_selected_features))
print("Selected Features from RFE:", list(rfe_selected_features))
```

```
7
Selected Features from RFE: ['radius_mean', 'texture_se', 'radius_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst'
```

```
X_rfe_selected=X[rfe_selected_features]
```

```
X_train_rfe, X_test_rfe, y_train_rfe, y_test_rfe = train_test_split(X_rfe_selected, y, test_size=0.2, random_state=42)
```

```
log_reg.fit(X_train_rfe, y_train_rfe)
```

```
LogisticRegression
LogisticRegression(max_iter=5000, random_state=42)
```

```
y_pred_rfe = log_reg.predict(X_test_rfe)
```

```
accuracy_rfe = accuracy_score(y_test_rfe, y_pred_rfe)
print(f"Model Accuracy using RFE-selected features: {accuracy_rfe*100:.4f}")
```

```
Model Accuracy using RFE-selected features: 97.3684
```

## Task5:

### Feature Importance using Random Forest

```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X, y)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Extract feature importance scores
rf_feature_importance = pd.Series(rf_model.feature_importances_, index=X.columns)
rf_feature_importance = rf_feature_importance.sort_values(ascending=False)
```

```
# Visualize feature importance
plt.figure(figsize=(12, 6))
sns.barplot(x=rf_feature_importance.index, y=rf_feature_importance.values, palette="coolwarm")
plt.xticks(rotation=90)
plt.xlabel("Features")
plt.ylabel("Feature Importance Score")
plt.title("Feature Importance using Random Forest")
plt.show()
```

```
plt.show()
```

```
<ipython-input-69-22bf2a0e4d6a>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.barplot(x=rf_feature_importance.index, y=rf_feature_importance.values, palette="coolwarm")
```