

Project 2 Steps: Name is : Ghurmallah Salem Alqarni

1. Download a dataset from Kaggle.
2. Handle missing values appropriately to prepare the datasets for applying machine learning models.
3. Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.
4. Use two feature selection methods to identify the most important features from the datasets.
5. Apply machine learning models to the selected features and evaluate the models.
6. Save the file code as your full name and upload the code as pdf. Submission is available until 21/12/2025.

```
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).

1.Download a dataset from Kaggle. <https://www.kaggle.com/datasets/rahmasleam/breast-cancer>

```
# =====
# Imports
# =====
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, RFE, mutual_info_regression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
from xgboost import XGBRegressor
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```



```
concavity_se      0
concave points_se 0
symmetry_se       0
fractal dimension se 0
```

2. Handle missing values appropriately to prepare the datasets for applying machine learning models.

```
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns
num_imputer = SimpleImputer(strategy='median')
df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])
cat_imputer = SimpleImputer(strategy='most_frequent')
df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
df.isnull().sum()
```


3. Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.

```
0
id  0

X = df.drop("diagnosis", axis=1)
y = df["diagnosis"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
#Logistic Regression
pipe_lr = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression(max_iter=1000))
])

param_grid_lr = {
    'clf__C': [0.01, 0.1, 1, 10],
    'clf__solver': ['liblinear']
}

#Support Vector Machine (SVM)
pipe_svm = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', SVC())
])

param_grid_svm = {
    'clf__C': [0.1, 1, 10],
    'clf__kernel': ['linear', 'rbf']
}

#Random Forest
pipe_rf = Pipeline([
    ('clf', RandomForestClassifier(random_state=42))
])

param_grid_rf = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [None, 10, 20]
}

#GridSearchCV
models = {
    "Logistic Regression": (pipe_lr, param_grid_lr),
    "SVM": (pipe_svm, param_grid_svm),
    "Random Forest": (pipe_rf, param_grid_rf)
}

results = {}

for name, (model, params) in models.items():
    grid = GridSearchCV(
        model,
        params,
        cv=5,
```

```

        scoring='accuracy',
        n_jobs=-1
    )
    grid.fit(X_train, y_train)
    results[name] = grid

```

```

for name, grid in results.items():
    y_pred = grid.predict(X_test)
    print("="*50)
    print(name)
    print("Best Parameters:", grid.best_params_)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

```

```

=====

```

Logistic Regression

Best Parameters: {'clf__C': 0.1, 'clf__solver': 'liblinear'}

Accuracy: 0.9824561403508771

Classification Report:

	precision	recall	f1-score	support
B	0.97	1.00	0.99	72
M	1.00	0.95	0.98	42
accuracy			0.98	114
macro avg	0.99	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

```

=====

```

SVM

Best Parameters: {'clf__C': 10, 'clf__kernel': 'rbf'}

Accuracy: 0.9736842105263158

Classification Report:

	precision	recall	f1-score	support
B	0.96	1.00	0.98	72
M	1.00	0.93	0.96	42
accuracy			0.97	114
macro avg	0.98	0.96	0.97	114
weighted avg	0.97	0.97	0.97	114

```

=====

```

Random Forest

Best Parameters: {'clf__max_depth': None, 'clf__n_estimators': 100}

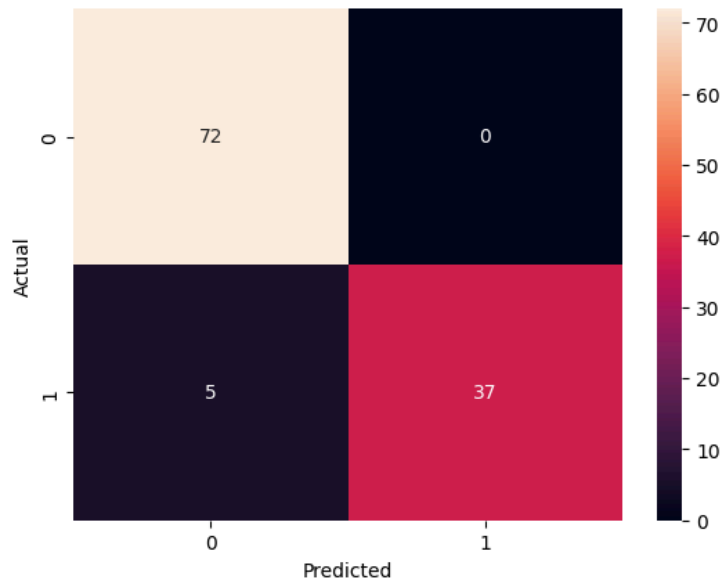
Accuracy: 0.956140350877193

Classification Report:

	precision	recall	f1-score	support
B	0.94	1.00	0.97	72
M	1.00	0.88	0.94	42
accuracy			0.96	114
macro avg	0.97	0.94	0.95	114
weighted avg	0.96	0.96	0.96	114

```
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, results["Random Forest"].predict(X_test))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



4. Use two feature selection methods to identify the most important features from the datasets.

```
import pandas as pd
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
#SelectKBest
from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(score_func=f_classif, k=10)
X_train_kbest = selector.fit_transform(X_train, y_train)

selected_features_kbest = X_train.columns[selector.get_support()]
print("Top 10 Features using SelectKBest:")
```

```
print(selected_features_kbest)
```

Top 10 Features using SelectKBest:

```
Index(['radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean',  
      'concave points_mean', 'radius_worst', 'perimeter_worst', 'area_worst',  
      'concavity_worst', 'concave points_worst'],  
      dtype='object')
```

```
#Random Forest Feature Importance  
from sklearn.ensemble import RandomForestClassifier  
import numpy as np  
import matplotlib.pyplot as plt  
  
rf = RandomForestClassifier(n_estimators=200, random_state=42)  
rf.fit(X_train, y_train)  
  
importances = rf.feature_importances_  
feature_importance_df = pd.DataFrame({  
    'Feature': X_train.columns,  
    'Importance': importances  
}).sort_values(by='Importance', ascending=False)  
  
print("Top 10 Features using Random Forest:")  
print(feature_importance_df.head(10))  
  
common_features = set(selected_features_kbest).intersection(  
    feature_importance_df.head(10)['Feature']  
)  
  
print("Common Important Features:")  
print(common_features)  
  
feature_importance_df.head(10).plot(  
    x='Feature',  
    y='Importance',  
    kind='bar',  
    legend=False  
)  
plt.title("Top 10 Feature Importances (Random Forest)")  
plt.ylabel("Importance Score")  
plt.show()
```

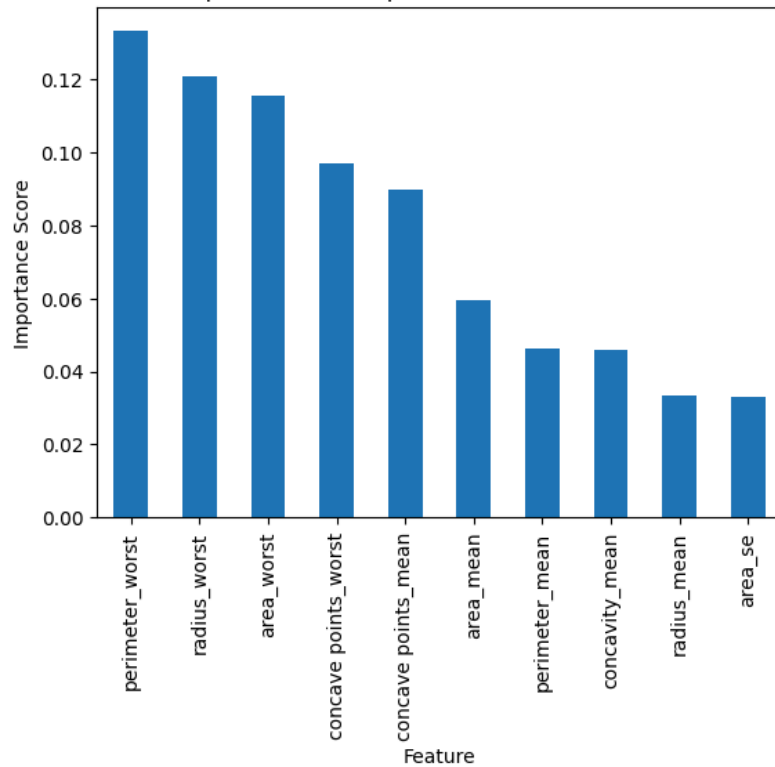
Top 10 Features using Random Forest:

	Feature	Importance
23	perimeter_worst	0.132898
21	radius_worst	0.120699
24	area_worst	0.115203
28	concave points_worst	0.097009
8	concave points_mean	0.089514
4	area_mean	0.059642
3	perimeter_mean	0.046239
7	concavity_mean	0.045974
1	radius_mean	0.033573
14	area_se	0.032980

Common Important Features:

{'concave points_worst', 'perimeter_mean', 'concavity_mean', 'perimeter_worst', 'area_mean', 'radius_worst', 'radius_mean', 'concave points_mean', 'area_worst'}

Top 10 Feature Importances (Random Forest)



5. Apply machine learning models to the selected features and evaluate the models.

```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
import pandas as pd

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

# SelectKBest
```

```

# SelectKBest
selector = SelectKBest(score_func=f_classif, k=10)
X_selected_array = selector.fit_transform(X, y)

selected_features = X.columns[selector.get_support()]

print("Selected Features:")
print(selected_features)

X_selected = X[selected_features]

X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y, test_size=0.2, random_state=42, stratify=y
)
# Logistic Regression
pipe_lr = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression(max_iter=1000))
])

pipe_lr.fit(X_train, y_train)
y_pred_lr = pipe_lr.predict(X_test)
# Support Vector Machine (SVM)
pipe_svm = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', SVC(kernel='rbf', C=1))
])

pipe_svm.fit(X_train, y_train)
y_pred_svm = pipe_svm.predict(X_test)
# Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
# evaluate
models_results = {
    "Logistic Regression": y_pred_lr,
    "SVM": y_pred_svm,
    "Random Forest": y_pred_rf
}

for model_name, y_pred in models_results.items():
    print("="*50)
    print(model_name)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
#
results_df = pd.DataFrame({
    "Model": ["Logistic Regression", "SVM", "Random Forest"],
    "Accuracy": [
        accuracy_score(y_test, y_pred_lr),
        accuracy_score(y_test, y_pred_svm),
        accuracy_score(y_test, y_pred_rf)
    ]
})

print(results_df)

```

```
Selected Features:
Index(['radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean',
      'concave points_mean', 'radius_worst', 'perimeter_worst', 'area_worst',
      'concavity_worst', 'concave points_worst'],
      dtype='object')
```

=====

Logistic Regression

Accuracy: 0.956140350877193

Classification Report:

	precision	recall	f1-score	support
B	0.95	0.99	0.97	72
M	0.97	0.90	0.94	42