Practical no: 7

Aim: 7. Model Evaluation and Hyperparameter Tuning

a. Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation.

Description:

Model Evaluation and Hyperparameter Tuning

Model evaluation is the process of testing how well a machine learning model performs on unseen data. It ensures that the model is not just memorizing the training data (overfitting) but can generalize well to new inputs. One of the most robust methods for evaluation is **cross-validation**, where the dataset is divided into multiple subsets (folds), and the model is trained and tested on different combinations of these subsets.

Cross-Validation Techniques

1. **K-Fold Cross-Validation**
    - The dataset is divided into *k* equal parts (folds).
    - The model is trained on *k-1* folds and tested on the remaining fold.
    - This process is repeated *k* times, and the average accuracy (or other metric) is taken as the final performance.
    - This method reduces bias from any single train-test split.
2. **Stratified K-Fold Cross-Validation**
    - Similar to K-Fold, but ensures that each fold maintains the same proportion of classes as in the overall dataset.
    - Useful in **classification tasks with imbalanced data** to avoid biased evaluation.

Key Evaluation Metrics

- **Accuracy** – percentage of correctly classified samples.
- **Precision & Recall** – important for imbalanced datasets.
- **R^2, MSE** – used for regression models.

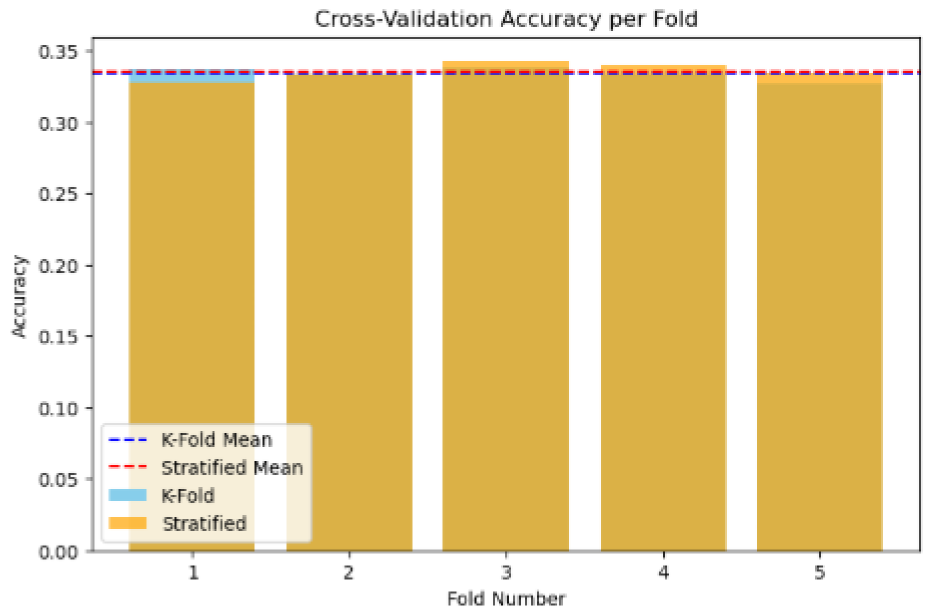
Code:

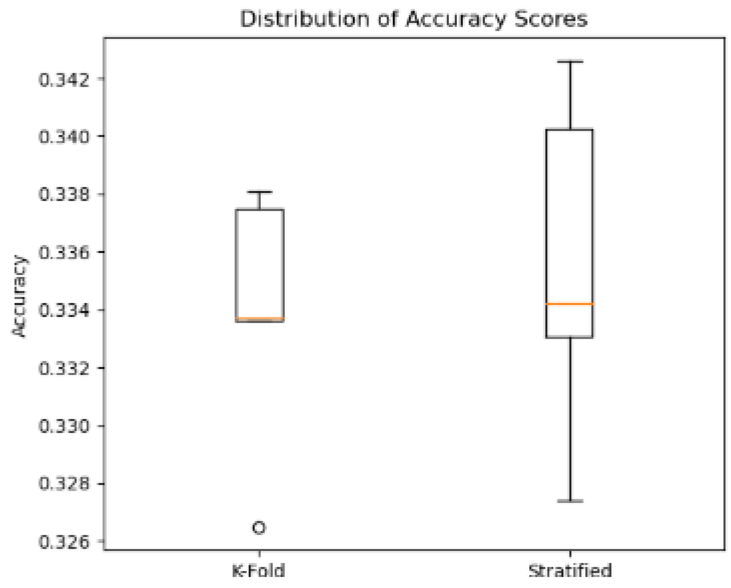
```
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

# Load dataset

file_path = "healthcare_dataset.csv"

df = pd.read_csv(file_path)

# Encode categorical variables if any

df_encoded = df.apply(lambda col: LabelEncoder().fit_transform(col) if col.dtypes ==
'object' else col)

# Separate features and target (assuming last column is target)

X = df_encoded.iloc[:, :-1]

y = df_encoded.iloc[:, -1]

# Define model

model = LogisticRegression(max_iter=1000)

# --- K-Fold Cross Validation ---

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

scores_kfold = cross_val_score(model, X, y, cv=kfold, scoring="accuracy")

# --- Stratified K-Fold Cross Validation ---

skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores_stratified = cross_val_score(model, X, y, cv=skfold, scoring="accuracy")


# --- Print Results ---

print("K-Fold Accuracy per Fold:", scores_kfold)

print("K-Fold Mean Accuracy:", np.mean(scores_kfold))

print("\nStratified K-Fold Accuracy per Fold:", scores_stratified)

print("Stratified K-Fold Mean Accuracy:", np.mean(scores_stratified))
```

```
# --- Visualization ---

# 1. Line Plot of Accuracy per Fold

plt.figure(figsize=(8,5))

plt.plot(range(1, len(scores_kfold)+1), scores_kfold, marker='o', color="blue", label="K-Fold")

plt.plot(range(1, len(scores_stratified)+1), scores_stratified, marker='s', color="orange", label="Stratified")

plt.axhline(np.mean(scores_kfold), color="blue", linestyle="--", label="K-Fold Mean")

plt.axhline(np.mean(scores_stratified), color="red", linestyle="--", label="Stratified Mean")

plt.title("Cross-Validation Accuracy per Fold (Line Plot)")

plt.xlabel("Fold Number")

plt.ylabel("Accuracy")

plt.legend()

plt.grid(True)

plt.show()


# 2. Violin Plot for Distribution

plt.figure(figsize=(6,5))

sns.violinplot(data=[scores_kfold, scores_stratified], palette="Set2")

plt.xticks([0, 1], ["K-Fold", "Stratified"])

plt.title("Distribution of Accuracy Scores (Violin Plot)")

plt.ylabel("Accuracy")

plt.show()
```

Output:

```
K-Fold Accuracy per Fold: [0.33747748 0.3336036  0.33810811 0.33369369 0.32648649]
K-Fold Mean Accuracy: 0.3338738738738385

Stratified K-Fold Accuracy per Fold: [0.32738739 0.33306306 0.34261261 0.34027027 0.33423423]
Stratified K-Fold Mean Accuracy: 0.3355135135135135
```





Learnings:

Understood the importance of **model evaluation** to check generalization.

Learned how to apply **K-Fold** and **Stratified K-Fold Cross-Validation** for robust evaluation.

Observed how cross-validation reduces the bias of a single train-test split.

Learned the role of **hyperparameter tuning** in improving model performance.

Aim : b.Systematically explore combinations of hyperparameters to optimize model performance. (use grid and randomized search)

Description:

Hyperparameter tuning is the process of systematically selecting the best hyperparameters for a machine learning model to maximize performance. Unlike model parameters (which are learned during training), hyperparameters are set before training (e.g., number of neighbors in KNN, max depth in Decision Tree, C in SVM).

Two common techniques are:

1. **Grid Search**
    - Tries all possible combinations of hyperparameters from a given search space.
    - Ensures the best combination is found but can be computationally expensive.
2. **Randomized Search**
    - Randomly samples combinations of hyperparameters from the search space.
    - Faster than grid search and often gives near-optimal results.

Both methods can be combined with **cross-validation** to ensure robust evaluation.

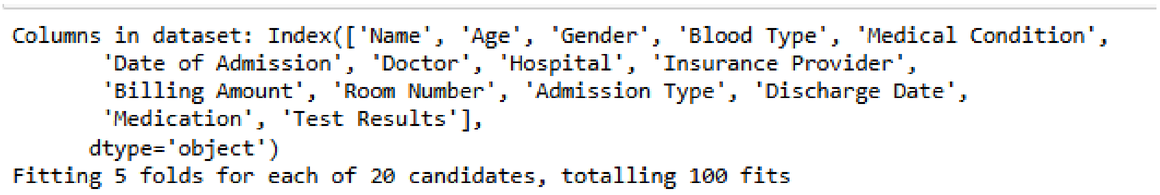
Code:

```
# ----------------------------

# Advanced Hyperparameter Tuning with Visualization

# ----------------------------


import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
```

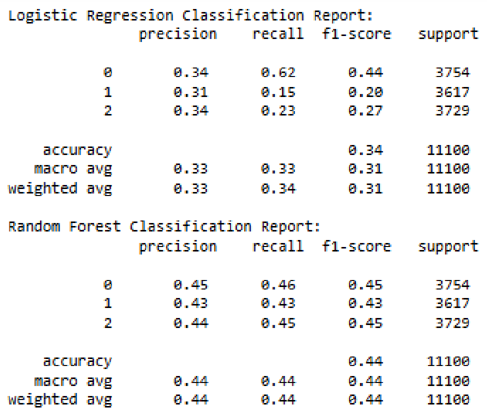
```
from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# -----------------------------

# 1. Load dataset

# -----------------------------

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


print("Columns in dataset:", df.columns)


# -----------------------------

# 2. Preprocessing

# -----------------------------

# Automatically encode categorical columns

for col in df.select_dtypes(include=['object']).columns:

    df[col] = LabelEncoder().fit_transform(df[col])


# Assume last column is target (update if needed)

target_column = df.columns[-1]


X = df.drop(columns=[target_column])

y = df[target_column]


# Scale features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

```
# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# -----------------------------
# 3. Logistic Regression with GridSearchCV
# -----------------------------
log_reg = LogisticRegression(max_iter=500)

param_grid_lr = {
   'C': [0.01, 0.1, 1, 10, 100],
   'solver': ['liblinear', 'saga'],
   'penalty': ['l1', 'l2']
}

grid_search_lr = GridSearchCV(log_reg, param_grid_lr, cv=5, scoring='accuracy',
verbose=1)
grid_search_lr.fit(X_train, y_train)

print("Best Parameters (Logistic Regression):", grid_search_lr.best_params_)
print("Best Score (Logistic Regression):", grid_search_lr.best_score_)

# -----------------------------
# 4. Random Forest with RandomizedSearchCV
# -----------------------------
rf = RandomForestClassifier()

param_dist_rf = {
    'n_estimators': np.arange(50, 300, 50),
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

random_search_rf = RandomizedSearchCV(rf, param_distributions=param_dist_rf,
                                n_iter=20, cv=5, scoring='accuracy',
                                random_state=42, verbose=1)
random_search_rf.fit(X_train, y_train)

print("Best Parameters (Random Forest):", random_search_rf.best_params_)
print("Best Score (Random Forest):", random_search_rf.best_score_)

# -----------------------------
# 5. Evaluate Best Models
# -----------------------------
best_lr = grid_search_lr.best_estimator_
best_rf = random_search_rf.best_estimator_

y_pred_lr = best_lr.predict(X_test)
y_pred_rf = best_rf.predict(X_test)

print("\nLogistic Regression Classification Report:\n", classification_report(y_test,
y_pred_lr))
print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf))

# -----------------------------
# 6. Visualization
# -----------------------------

# Confusion Matrix for both models
fig, axes = plt.subplots(1, 2, figsize=(12,5))

sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt="d", cmap="Blues",
ax=axes[0])
axes[0].set_title("Logistic Regression Confusion Matrix")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")

sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt="d", cmap="Greens",
ax=axes[1])
axes[1].set_title("Random Forest Confusion Matrix")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")
plt.show()
# Feature importance for Random Forest
importances = best_rf.feature_importances_
feat_importances = pd.Series(importances,
index=X.columns).sort_values(ascending=False)

plt.figure(figsize=(8,6))
sns.barplot(x=feat_importances, y=feat_importances.index, palette="viridis")
plt.title("Feature Importance - Random Forest")
plt.show()
```

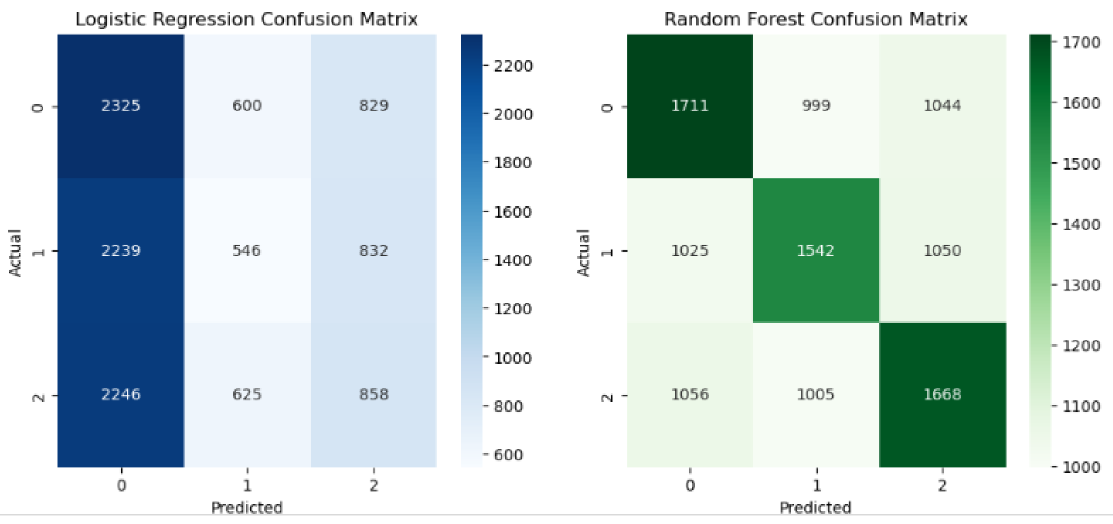
Output:

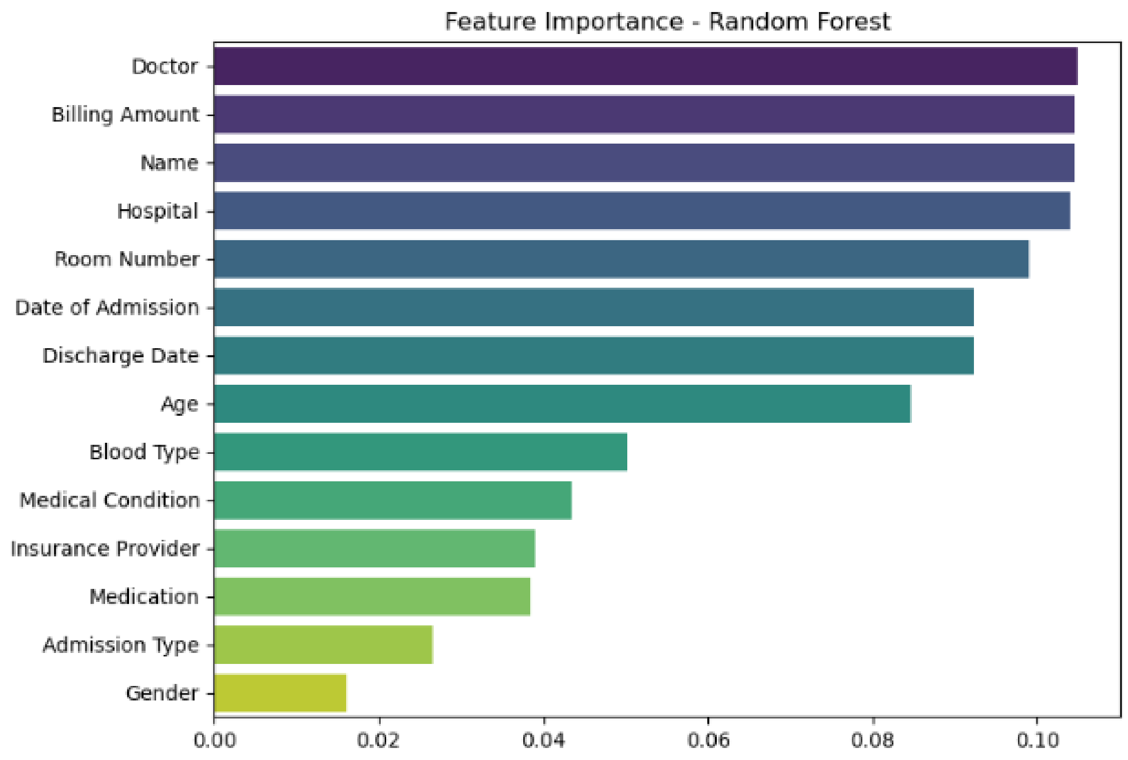


```
Columns in dataset: Index(['Name', 'Age', 'Gender', 'Blood Type', 'Medical Condition',
       'Date of Admission', 'Doctor', 'Hospital', 'Insurance Provider',
       'Billing Amount', 'Room Number', 'Admission Type', 'Discharge Date',
       'Medication', 'Test Results'],
      dtype='object')
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```



```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.34      0.62      0.44      3754
           1       0.31      0.15      0.20      3617
           2       0.34      0.23      0.27      3729

    accuracy                           0.34     11100
   macro avg       0.33      0.33      0.31     11100
weighted avg       0.33      0.34      0.31     11100

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.45      0.46      0.45      3754
           1       0.43      0.43      0.43      3617
           2       0.44      0.45      0.45      3729

    accuracy                           0.44     11100
   macro avg       0.44      0.44      0.44     11100
weighted avg       0.44      0.44      0.44     11100
```





Learnings:

Hyperparameter tuning improves model accuracy by finding the best settings (like depth, C, n_estimators).

Grid Search → tests all combinations of parameters (exhaustive but slow).

Randomized Search → tests random parameter combinations (faster, good for large search space).

Cross-validation ensures robust evaluation by splitting data into multiple folds.

Logistic Regression works well for linear problems, while **Random Forest** handles complex, non-linear data.

Confusion Matrix & Reports show precision, recall, F1, and accuracy to judge performance.

Feature Importance (RF) highlights which features contribute most to predictions.