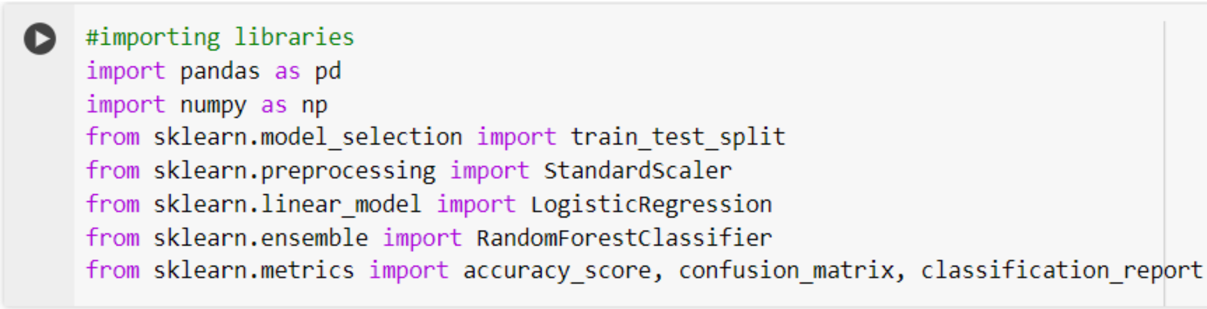
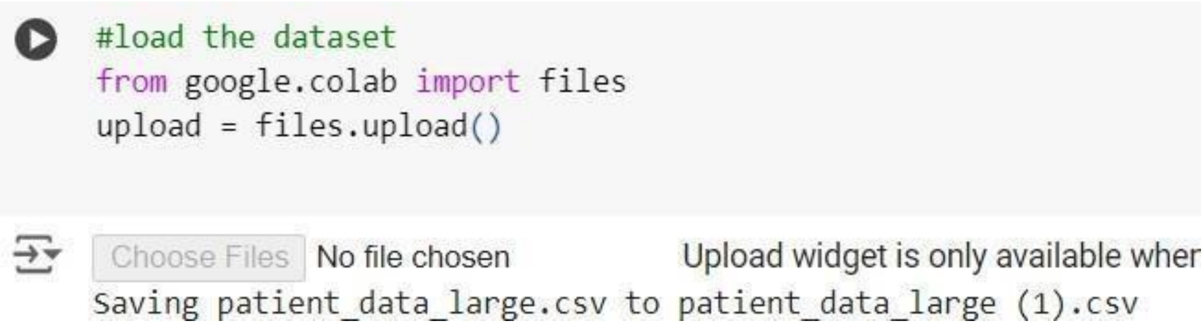
Practical 06

Aim - Healthcare Analytics using logistic regression. To predict the likelihood of patient readmission based on key health indicator.

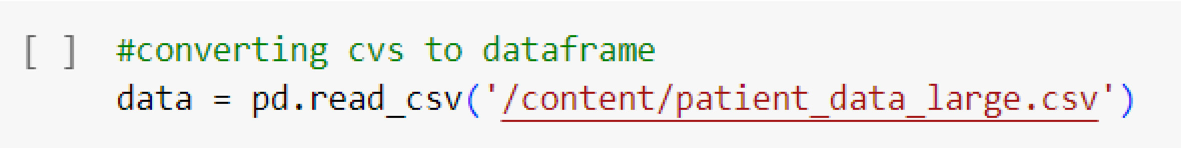


```python
#importing libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

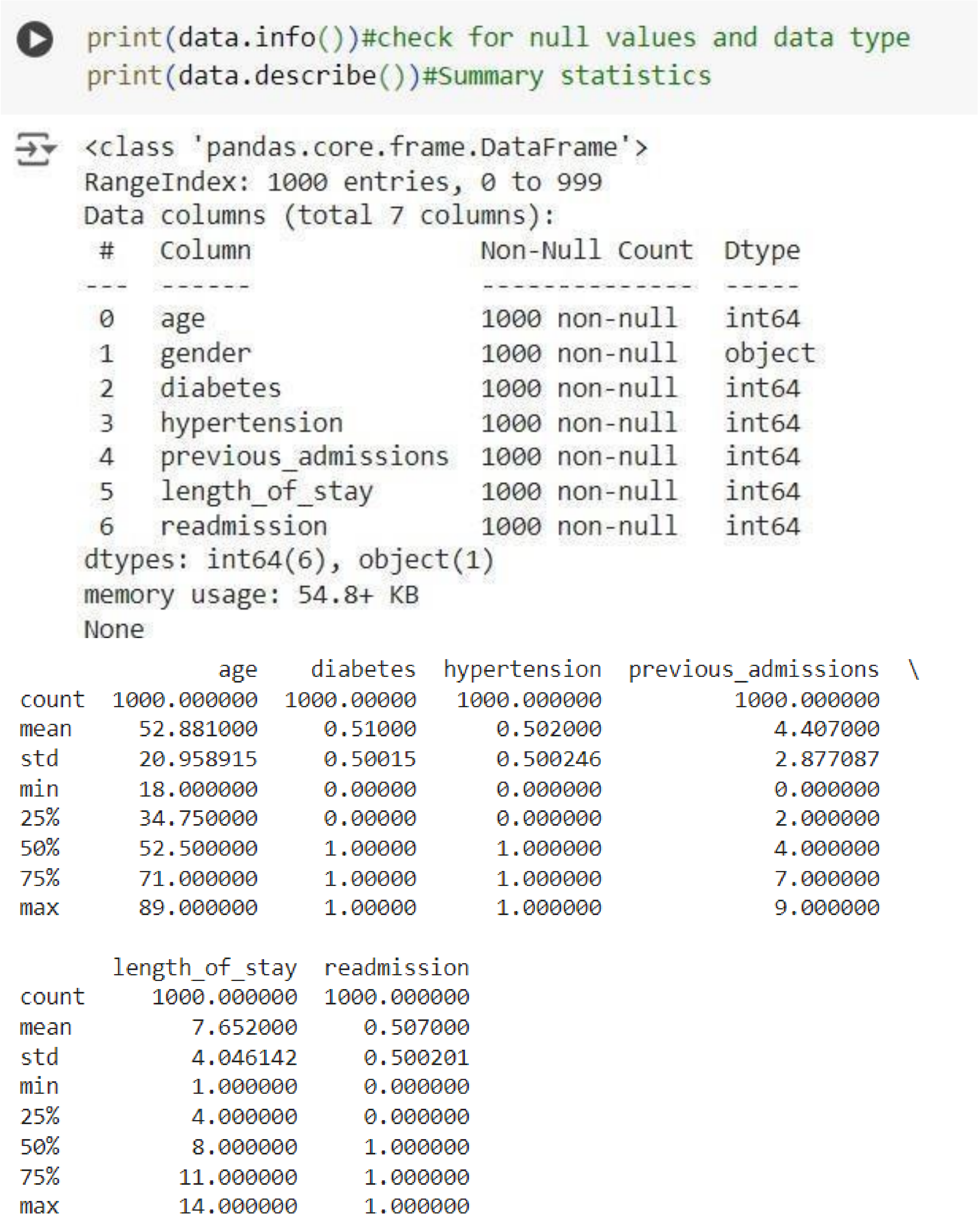


```python
#load the dataset
from google.colab import files
upload = files.upload()
```

```
Choose Files  No file chosen          Upload widget is only available wher
Saving patient_data_large.csv to patient_data_large (1).csv
```

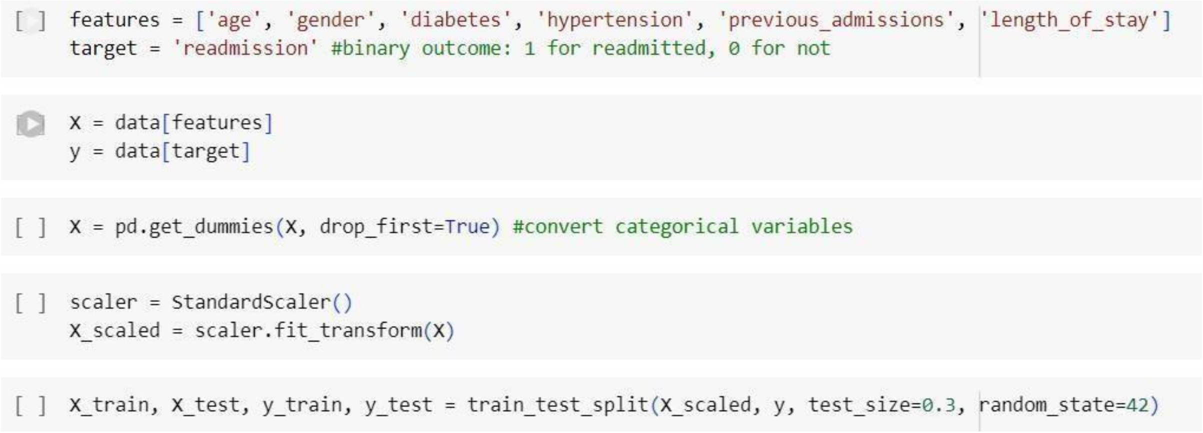


```python
#converting cvs to dataframe
data = pd.read_csv('/content/patient_data_large.csv')
```



```
print(data.info())#check for null values and data type
print(data.describe())#Summary statistics
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   age                 1000 non-null   int64 
 1   gender              1000 non-null   object
 2   diabetes            1000 non-null   int64 
 3   hypertension        1000 non-null   int64 
 4   previous_admissions 1000 non-null   int64 
 5   length_of_stay      1000 non-null   int64 
 6   readmission         1000 non-null   int64 
dtypes: int64(6), object(1)
memory usage: 54.8+ KB
None
               age    diabetes  hypertension  previous_admissions  \
count  1000.000000  1000.00000   1000.000000          1000.000000   
mean     52.881000     0.51000      0.502000             4.407000   
std      20.958915     0.50015      0.500246             2.877087   
min      18.000000     0.00000      0.000000             0.000000   
25%      34.750000     0.00000      0.000000             2.000000   
50%      52.500000     1.00000      1.000000             4.000000   
75%      71.000000     1.00000      1.000000             7.000000   
max      89.000000     1.00000      1.000000             9.000000   

       length_of_stay  readmission  
count     1000.000000  1000.000000  
mean         7.652000     0.507000  
std          4.046142     0.500201  
min          1.000000     0.000000  
25%          4.000000     0.000000  
50%          8.000000     1.000000  
75%         11.000000     1.000000  
max         14.000000     1.000000  
```



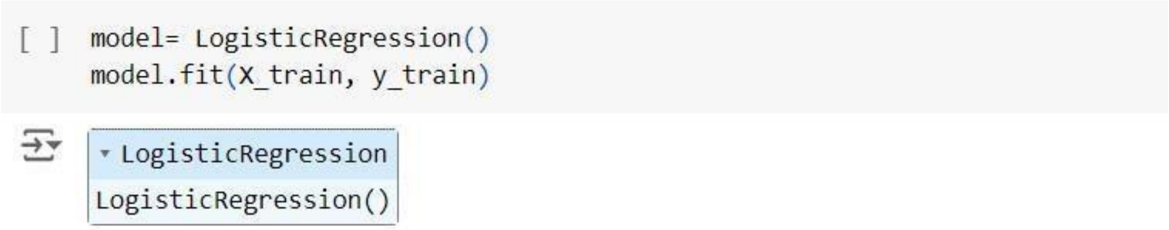
```
[ ] features = ['age', 'gender', 'diabetes', 'hypertension', 'previous_admissions', 'length_of_stay']
    target = 'readmission' #binary outcome: 1 for readmitted, 0 for not
```

```
X = data[features]
y = data[target]
```

```
[ ] X = pd.get_dummies(X, drop_first=True) #convert categorical variables
```

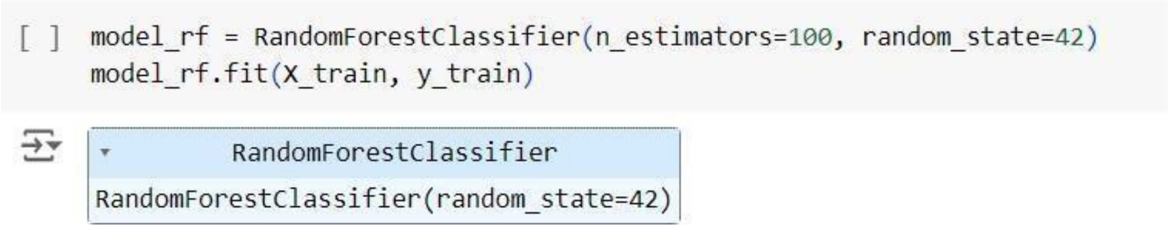
```
[ ] scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```



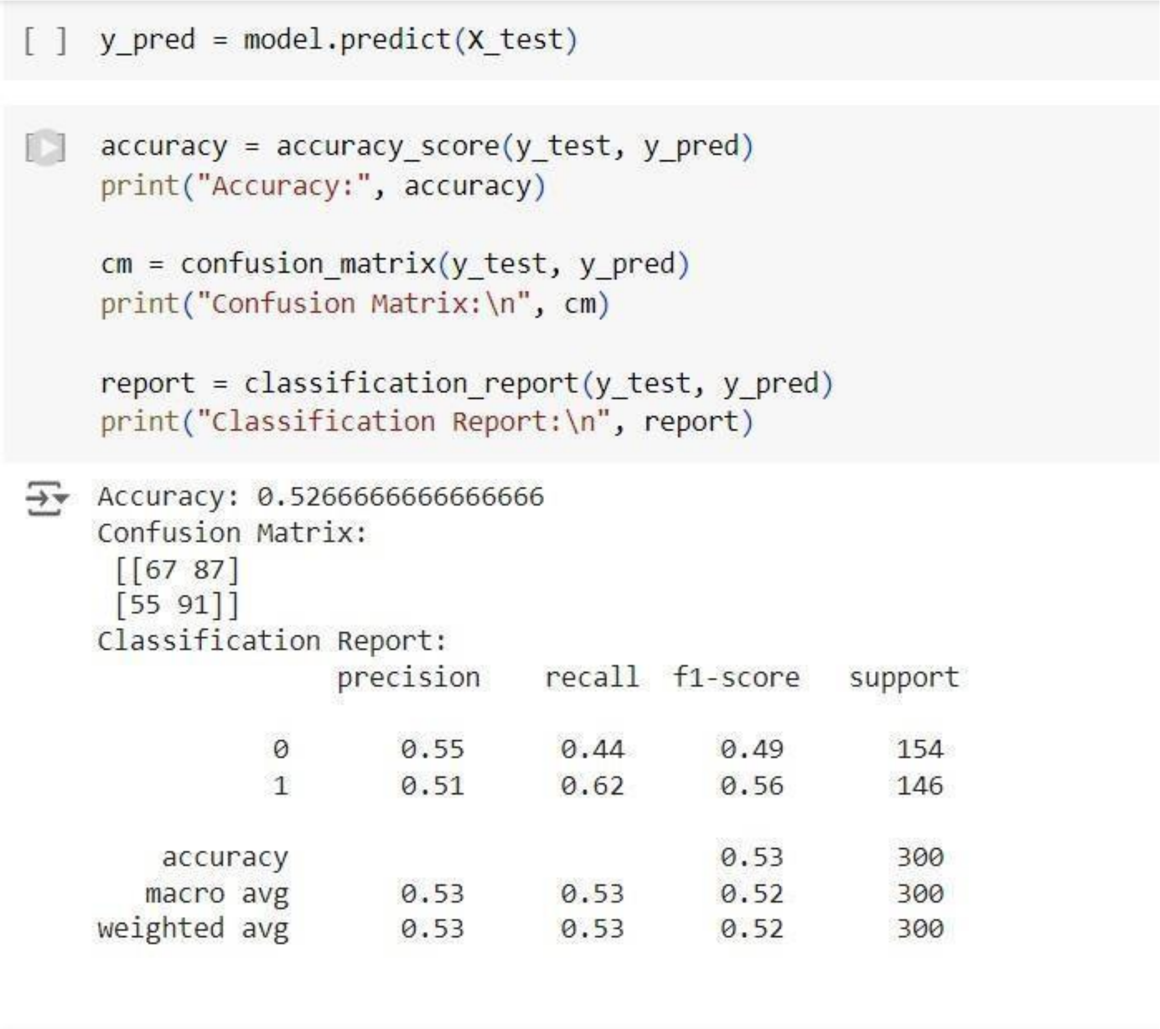
```
[ ] model= LogisticRegression()
    model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```



```
[ ] model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
    model_rf.fit(X_train, y_train)
```

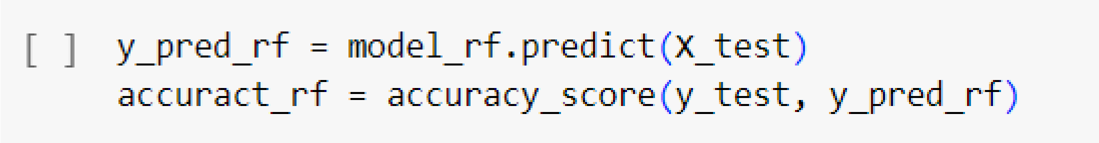
```
▾          RandomForestClassifier
RandomForestClassifier(random_state=42)
```



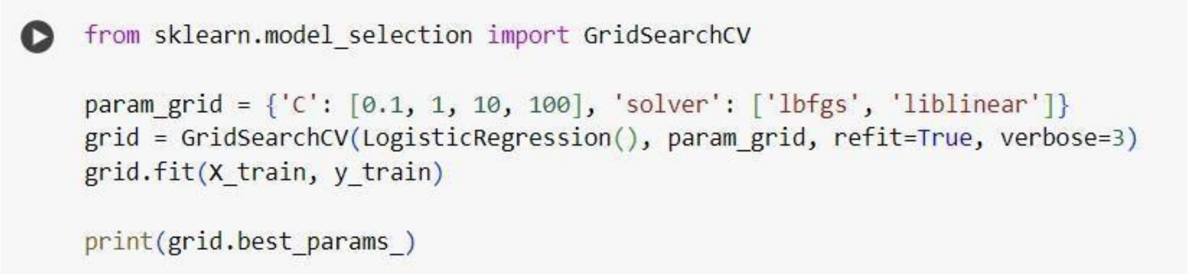
```
[ ] y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

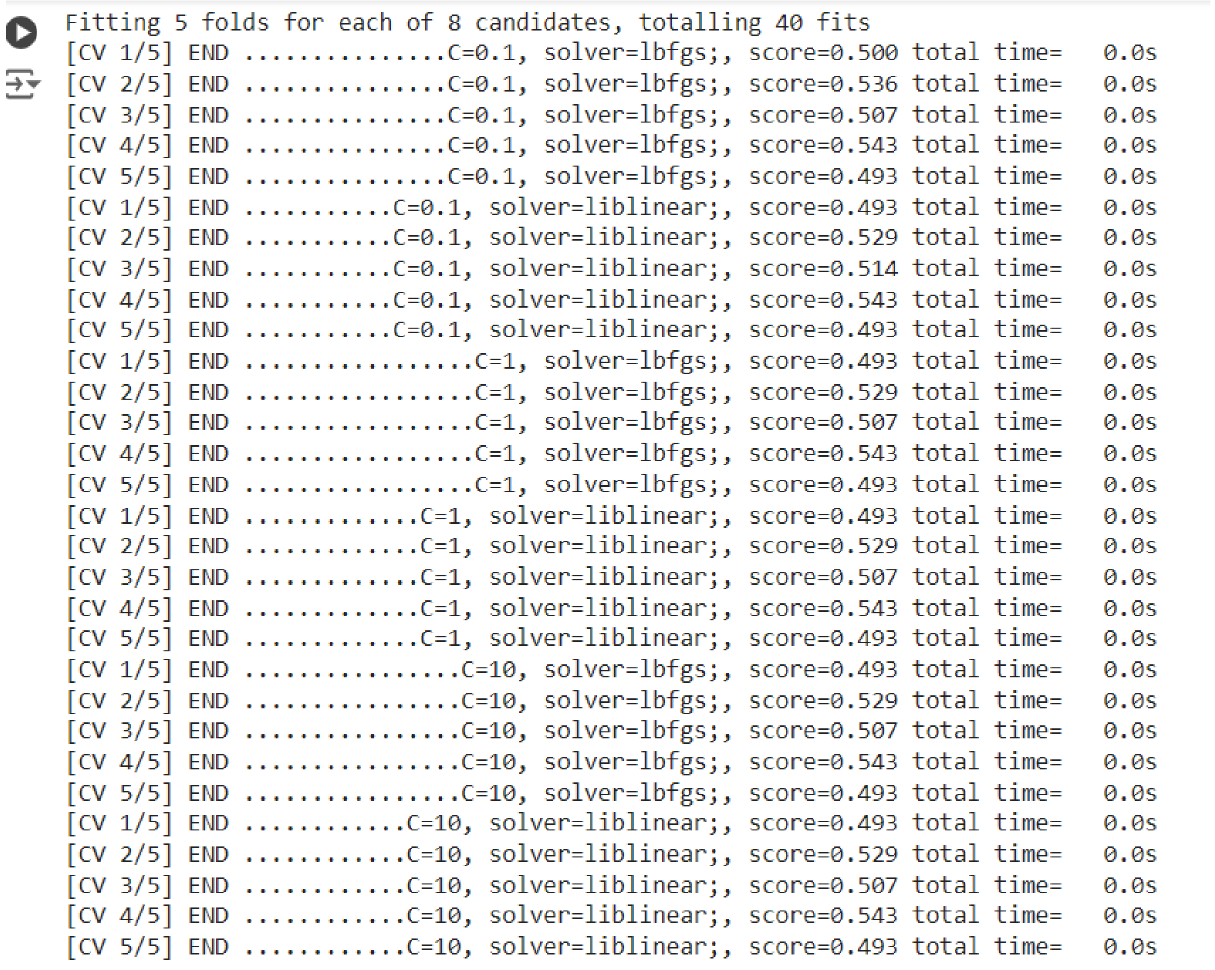
```
Accuracy: 0.5266666666666666
Confusion Matrix:
 [[67 87]
 [55 91]]
Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.44      0.49       154
           1       0.51      0.62      0.56       146

    accuracy                           0.53       300
   macro avg       0.53      0.53      0.52       300
weighted avg       0.53      0.53      0.52       300
```



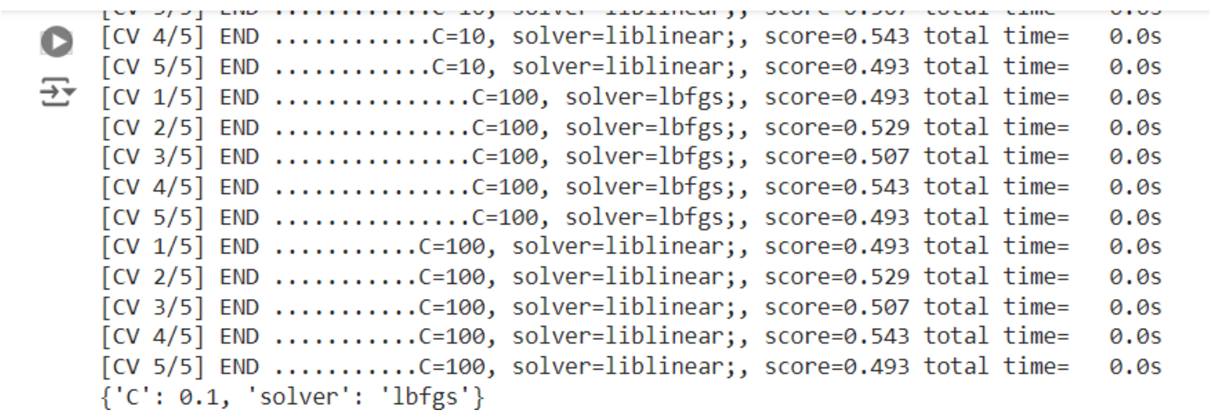
```
[ ] y_pred_rf = model_rf.predict(X_test)
    accuract_rf = accuracy_score(y_test, y_pred_rf)
```



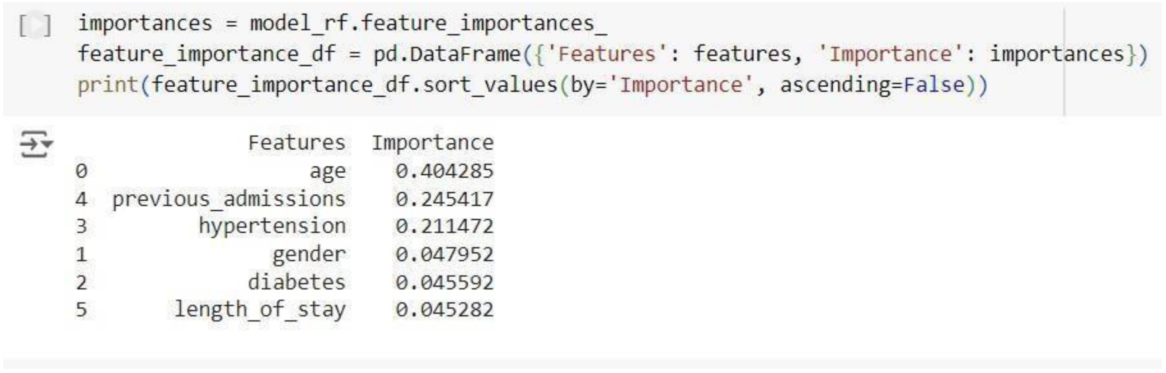
```
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100], 'solver': ['lbfgs', 'liblinear']}
grid = GridSearchCV(LogisticRegression(), param_grid, refit=True, verbose=3)
grid.fit(X_train, y_train)

print(grid.best_params_)
```



```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV 1/5] END ..............C=0.1, solver=lbfgs;, score=0.500 total time=   0.0s
[CV 2/5] END ..............C=0.1, solver=lbfgs;, score=0.536 total time=   0.0s
[CV 3/5] END ..............C=0.1, solver=lbfgs;, score=0.507 total time=   0.0s
[CV 4/5] END ..............C=0.1, solver=lbfgs;, score=0.543 total time=   0.0s
[CV 5/5] END ..............C=0.1, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 1/5] END ..........C=0.1, solver=liblinear;, score=0.493 total time=   0.0s
[CV 2/5] END ..........C=0.1, solver=liblinear;, score=0.529 total time=   0.0s
[CV 3/5] END ..........C=0.1, solver=liblinear;, score=0.514 total time=   0.0s
[CV 4/5] END ..........C=0.1, solver=liblinear;, score=0.543 total time=   0.0s
[CV 5/5] END ..........C=0.1, solver=liblinear;, score=0.493 total time=   0.0s
[CV 1/5] END ................C=1, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 2/5] END ................C=1, solver=lbfgs;, score=0.529 total time=   0.0s
[CV 3/5] END ................C=1, solver=lbfgs;, score=0.507 total time=   0.0s
[CV 4/5] END ................C=1, solver=lbfgs;, score=0.543 total time=   0.0s
[CV 5/5] END ................C=1, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 1/5] END ............C=1, solver=liblinear;, score=0.493 total time=   0.0s
[CV 2/5] END ............C=1, solver=liblinear;, score=0.529 total time=   0.0s
[CV 3/5] END ............C=1, solver=liblinear;, score=0.507 total time=   0.0s
[CV 4/5] END ............C=1, solver=liblinear;, score=0.543 total time=   0.0s
[CV 5/5] END ............C=1, solver=liblinear;, score=0.493 total time=   0.0s
[CV 1/5] END ...............C=10, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 2/5] END ...............C=10, solver=lbfgs;, score=0.529 total time=   0.0s
[CV 3/5] END ...............C=10, solver=lbfgs;, score=0.507 total time=   0.0s
[CV 4/5] END ...............C=10, solver=lbfgs;, score=0.543 total time=   0.0s
[CV 5/5] END ...............C=10, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 1/5] END ...........C=10, solver=liblinear;, score=0.493 total time=   0.0s
[CV 2/5] END ...........C=10, solver=liblinear;, score=0.529 total time=   0.0s
[CV 3/5] END ...........C=10, solver=liblinear;, score=0.507 total time=   0.0s
[CV 4/5] END ...........C=10, solver=liblinear;, score=0.543 total time=   0.0s
[CV 5/5] END ...........C=10, solver=liblinear;, score=0.493 total time=   0.0s
```



```
[CV 4/5] END ...........C=10, solver=liblinear;, score=0.543 total time=   0.0s
[CV 5/5] END ...........C=10, solver=liblinear;, score=0.493 total time=   0.0s
[CV 1/5] END ..............C=100, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 2/5] END ..............C=100, solver=lbfgs;, score=0.529 total time=   0.0s
[CV 3/5] END ..............C=100, solver=lbfgs;, score=0.507 total time=   0.0s
[CV 4/5] END ..............C=100, solver=lbfgs;, score=0.543 total time=   0.0s
[CV 5/5] END ..............C=100, solver=lbfgs;, score=0.493 total time=   0.0s
[CV 1/5] END ..........C=100, solver=liblinear;, score=0.493 total time=   0.0s
[CV 2/5] END ..........C=100, solver=liblinear;, score=0.529 total time=   0.0s
[CV 3/5] END ..........C=100, solver=liblinear;, score=0.507 total time=   0.0s
[CV 4/5] END ..........C=100, solver=liblinear;, score=0.543 total time=   0.0s
[CV 5/5] END ..........C=100, solver=liblinear;, score=0.493 total time=   0.0s
{'C': 0.1, 'solver': 'lbfgs'}
```



```
importances = model_rf.feature_importances_
feature_importance_df = pd.DataFrame({'Features': features, 'Importance': importances})
print(feature_importance_df.sort_values(by='Importance', ascending=False))
```

```
             Features  Importance
0                 age    0.404285
4  previous_admissions   0.245417
3        hypertension    0.211472
1              gender    0.047952
2            diabetes    0.045592
5      length_of_stay    0.045282
```