Titanic survival prediction by Logistic regression

In [215…
```
## Importing important libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [216…
```
## Loading seaborn sample dataset - Titanic
df = sns.load_dataset("titanic")
```

In [217…
```
df.head() ## There are some unneccessary column which we can drop
```

Out[217…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [218…
```
df.shape
```

Out[218… (891, 15)

In [219…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

In [220…
```
df.describe()
```

Out[220…

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [221…
```
df.isnull().sum() ## There are null values to be taken care
```

Out[221…
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [222…
```
df.drop(columns=["deck", "embark_town", "alive", "class", "who", ],inplace=True)
```

In [223…
```
df.head()
```

Out[223…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	adult_male	alone
0	0	3	male	22.0	1	0	7.2500	S	True	False
1	1	1	female	38.0	1	0	71.2833	C	False	False
2	1	3	female	26.0	0	0	7.9250	S	False	True
3	1	1	female	35.0	1	0	53.1000	S	False	False
4	0	3	male	35.0	0	0	8.0500	S	True	True

In [224…
```
## Filling null values in age column with mean
df["age"].fillna(df["age"].mean(),inplace=True)
```

In [225…
```
## Dropping null values of emarked column
df["embarked"].dropna(inplace=True)
```

In [226…
```
df
```

Out[226…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	adult_male	alone
0	0	3	male	22.000000	1	0	7.2500	S	True	False
1	1	1	female	38.000000	1	0	71.2833	C	False	False
2	1	3	female	26.000000	0	0	7.9250	S	False	True
3	1	1	female	35.000000	1	0	53.1000	S	False	False
4	0	3	male	35.000000	0	0	8.0500	S	True	True
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.000000	0	0	13.0000	S	True	True
887	1	1	female	19.000000	0	0	30.0000	S	False	True
888	0	3	female	29.699118	1	2	23.4500	S	False	False
889	1	1	male	26.000000	0	0	30.0000	C	True	True
890	0	3	male	32.000000	0	0	7.7500	Q	True	True

891 rows × 10 columns

In [227…
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [228…
```
## doing label encoding for sex & embarked column
df['sex'] = le.fit_transform(df['sex'])
df["embarked"] = le.fit_transform(df["embarked"]) # S=2, C=0, Q=1
```

In [229…
```
df = df.astype(int)
```

In [230…
```
df
```

Out[230…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	adult_male	alone
0	0	3	1	22	1	0	7	2	1	0
1	1	1	0	38	1	0	71	0	0	0
2	1	3	0	26	0	0	7	2	0	1
3	1	1	0	35	1	0	53	2	0	0
4	0	3	1	35	0	0	8	2	1	1
...	...	...	...	...	...	...	...	...	...	...
886	0	2	1	27	0	0	13	2	1	1
887	1	1	0	19	0	0	30	2	0	1
888	0	3	0	29	1	2	23	2	0	0
889	1	1	1	26	0	0	30	0	1	1
890	0	3	1	32	0	0	7	1	1	1

891 rows × 10 columns

In [231…
```
df
```

Out[231…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	adult_male	alone
0	0	3	1	22	1	0	7	2	1	0
1	1	1	0	38	1	0	71	0	0	0
2	1	3	0	26	0	0	7	2	0	1
3	1	1	0	35	1	0	53	2	0	0
4	0	3	1	35	0	0	8	2	1	1
...	...	...	...	...	...	...	...	...	...	...
886	0	2	1	27	0	0	13	2	1	1
887	1	1	0	19	0	0	30	2	0	1
888	0	3	0	29	1	2	23	2	0	0
889	1	1	1	26	0	0	30	0	1	1
890	0	3	1	32	0	0	7	1	1	1

891 rows × 10 columns

In [232…
```
X = df.drop("survived",axis=1)
y = df["survived"]
```

In [233…
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [234…
```
## Splitting the data into train & test data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [235…
```
## Model training
model.fit(X_train,y_train)
```

Out[235…

▾ LogisticRegression

LogisticRegression()

In [236…
```
## Model prediction
y_pred = model.predict(X_test)
```

In [237…
```
y_pred
```

Out[237…
```
array([0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 1])
```

In [238…
```
y_test
```

Out[238…
```
709    1
439    0
840    0
720    1
39     1
      ..
433    0
773    0
25     1
84     1
10     1
Name: survived, Length: 179, dtype: int64
```

In [239…
```
from sklearn.metrics import accuracy_score, confusion_matrix , classification_report
```

In [240…
```
## Model evaluation using accuracy, precision, recall & f1-Score
accuracy_score(y_test,y_pred)
```

Out[240…
```
0.8156424581005587
```

In [241…
```
confusion_matrix(y_test,y_pred)
```

Out[241…
```
array([[91, 14],
       [19, 55]])
```

In [242…
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       105
           1       0.80      0.74      0.77        74

    accuracy                           0.82       179
   macro avg       0.81      0.80      0.81       179
weighted avg       0.81      0.82      0.81       179
```

Conclusion

We have an built an efficienct regression model with Accuracy - 0.81, Precision - 0.83, Recall - 0.87, f1 - score - 0.85