EC3066D Artificial Intelligence: Theory and Practice

PROGRAMMING ASSIGNMENT

ABHINAV RAJ V

B210759EC

I. DATA VISUALIZATION

Write and execute Python scripts to do the followings:

(i) Read CSV file & display information on the dataframe.

Hints: read_csv(), info() method

CODE

```
import pandas as pd
import matplotlib.pyplot as plt
titanic_data = pd.read_csv(r"C:\Users\HP\Desktop\AI ASSIGMNET\titanic.csv")
titanic_data.info()
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

(ii) Display first 10 rows of the data.

CODE

```
titanic_data.head(10)
```

OUTPUT

```
In [4]: titanic_data.head(10)
Out[4]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

(iii) Display first 5 rows of the data having the given columns only.

'PassengerID', 'Name', 'Age', 'Sex'I.

CODE

```
titanic_data[['PassengerId','Name','Age','Sex']].head(5)
```

OUTPUT

```
In [5]: titanic_data[['PassengerId','Name','Age','Sex']].head(5)
Out[5]:
```

	PassengerId	Name	Age	Sex
0	1	Braund, Mr. Owen Harris	22.0	male
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	female
2	3	Heikkinen, Miss. Laina	26.0	female
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	female
4	5	Allen, Mr. William Henry	35.0	male

II. DATA ANALYSIS

For data visualization, the popular packages are Matplotlib and Seaborn. More advanced functionality is available with Seaborn.

Write and execute Python scripts to do the followings:

(i) Plot the count of survived passengers

CODE

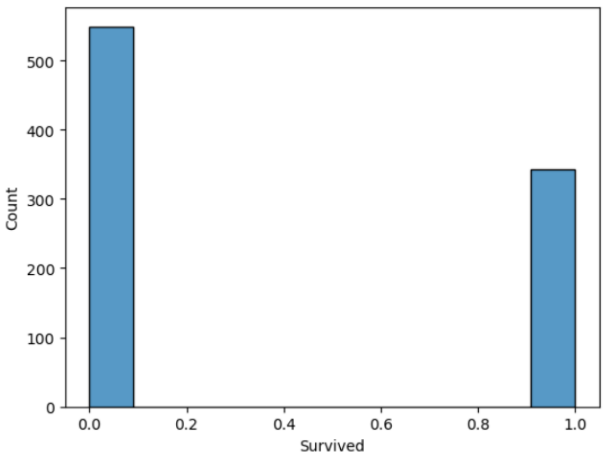
```
import seaborn as sns
sns.histplot(titanic_data['Survived'])
```

OUTPUT

```
In [6]: import seaborn as sns

        sns.histplot(titanic_data['Survived'])

Out[6]: <Axes: xlabel='Survived', ylabel='Count'>
```



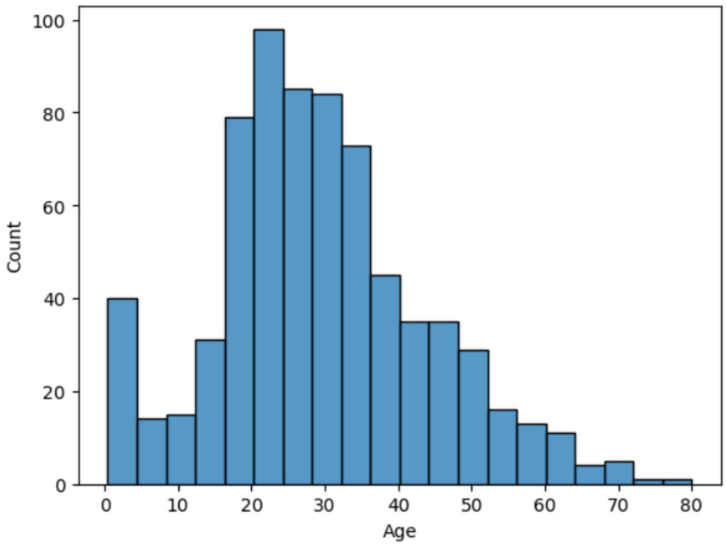
(ii) Plot histogram of ‘Age’ column Hints: hist() method

CODE

```
sns.histplot(titanic_data['Age'])
```

<u>**OUTPUT**</u>

```
In [7]: sns.histplot(titanic_data['Age'])
Out[7]: <Axes: xlabel='Age', ylabel='Count'>
```



II. DATAWRANGLING & FEATURE SELECTION

You can easily understand that all the columns (features) in the dataset are not significant for a binary classification problem to classify 'survived' or 'not'. Also, you can see NaN values in the dataset. So, data pre-processing is required here.

Write and execute Python scripts to do the followings:

(i) Drop the following unnecessary columns.

'PassengerID','Name', 'Ticket', 'Cabin', 'Embarked'

<u>**CODE**</u>

```
print('Data after preprocessing')
titanic_data.columns
unnecessary_columns = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
titanic_data.drop(unnecessary_columns, axis=1, inplace=True)
titanic_data.head(10)
```

OUTPUT

```
In [9]: print('Data after preprocessing')
        titanic_data.columns
        unnecessary_columns = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
        titanic_data.drop(unnecessary_columns, axis=1, inplace=True)
        titanic_data.head(10)

        Data after preprocessing
```

Out[9]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
5	0	3	male	NaN	0	0	8.4583
6	0	1	male	54.0	0	0	51.8625
7	0	3	male	2.0	3	1	21.0750
8	1	3	female	27.0	0	2	11.1333
9	1	2	female	14.0	1	0	30.0708

(ii) How many 'NaN' entries in 'Age' column? Replace all 'NaN' values in the 'Age' column with mean value of the 'Age' column vector. (Mean value replacement is a popular choice. It will not make a considerable damage to the data distribution in the column vector!). Please round off the mean value to two decimals.

CODE

```
n_count=titanic_data['Age'].isna().sum()
print("Number of NaN entries in Age column is",n_count)

mean_age=titanic_data['Age'].mean()
mean_age=round(mean_age,2)
print('Mean age is',mean_age)

titanic_data['Age'].fillna(mean_age,inplace=True)
titanic_data.head(10)
```

OUTPUT

```
In [10]: n_count=titanic_data['Age'].isna().sum()
         print("Number of NaN entries in Age column is",n_count)

         Number of NaN entries in Age column is 177

In [11]: mean_age=titanic_data['Age'].mean()
         mean_age=round(mean_age,2)
         print('Mean age is',mean_age)

         Mean age is 29.7

In [12]: titanic_data['Age'].fillna(mean_age,inplace=True)
         titanic_data.head(10)

Out[12]:
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
5	0	3	male	29.7	0	0	8.4583
6	0	1	male	54.0	0	0	51.8625
7	0	3	male	2.0	3	1	21.0750
8	1	3	female	27.0	0	2	11.1333
9	1	2	female	14.0	1	0	30.0708

(iii) The entries in the 'Sex' column are 'Male' or 'Female'. 'Pclass' can have 1st,2nd,3rd.

We should convert them to numerical values.

CODE

```
titanic_sex=pd.get_dummies(titanic_sex, columns=['Sex'])
titanic_pclass=pd.get_dummies(titanic_pclass, columns=['Pclass'])
titanic_sex.head(5)
titanic_pclass.head(5)
titanic_data2=titanic_data[['Sex','Pclass']]
titanic_data2 = pd.get_dummies(titanic_data2, columns=['Sex', 'Pclass'], drop_first=True)
titanic_data2.head(5)
```

OUTPUT

```
In [16]: titanic_sex=pd.get_dummies(titanic_sex, columns=['Sex'])
         titanic_pclass=pd.get_dummies(titanic_pclass, columns=['Pclass'])
         titanic_sex.head(5)

Out[16]:
```

	Sex_female	Sex_male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

```
In [17]: titanic_pclass.head(5)

Out[17]:
```

	Pclass_1	Pclass_2	Pclass_3
0	0	0	1
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1

```
In [18]: titanic_data2=titanic_data[['Sex','Pclass']]
         titanic_data2 = pd.get_dummies(titanic_data2, columns=['Sex', 'Pclass'], drop_first=True)

In [19]: titanic_data2.head(5)
```

Out[19]:

	Sex_male	Pclass_2	Pclass_3
0	1	0	1
1	0	0	0
2	0	0	1
3	0	0	0
4	1	0	1

(iv) Concatenate the results of ‘Sex’ and ‘Pclass’ from previous step to get the following pre-processed dataset.

CODE

```
conc_titanic_data = pd.concat([titanic_data, titanic_data2], axis=1)
conc_titanic_data.head(5)
```

OUTPUT

```
In [20]: conc_titanic_data = pd.concat([titanic_data, titanic_data2], axis=1)
         conc_titanic_data.head(5)
```

Out[20]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3
0	0	3	male	22.0	1	0	7.2500	1	0	1
1	1	1	female	38.0	1	0	71.2833	0	0	0
2	1	3	female	26.0	0	0	7.9250	0	0	1
3	1	1	female	35.0	1	0	53.1000	0	0	0
4	0	3	male	35.0	0	0	8.0500	1	0	1

(v) Next, drop ‘Pclass’ and ‘Sex’ from the data frame to obtain the following:

CODE

```
conc_titanic_data.drop(['Pclass', 'Sex'], axis=1, inplace=True)
conc_titanic_data.head(5)
```

OUTPUT

```
In [21]: conc_titanic_data.drop(['Pclass', 'Sex'], axis=1, inplace=True)
         conc_titanic_data.head(5)
```

Out[21]:

	Survived	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3
0	0	22.0	1	0	7.2500	1	0	1
1	1	38.0	1	0	71.2833	0	0	0
2	1	26.0	0	0	7.9250	0	0	1
3	1	35.0	1	0	53.1000	0	0	0
4	0	35.0	0	0	8.0500	1	0	1

(vi) We can rename the column names as shown below (for convenience):

CODE

```
conc_titanic_data.columns = ['Survived', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex', 'Pclass_1',
'Pclass_2']
conc_titanic_data.head(5)
```

OUTPUT

```
In [22]: conc_titanic_data.columns = ['Survived', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex', 'Pclass_1', 'Pclass_2']
         conc_titanic_data.head(5)
```

Out[22]:

	Survived	Age	SibSp	Parch	Fare	Sex	Pclass_1	Pclass_2
0	0	22.0	1	0	7.2500	1	0	1
1	1	38.0	1	0	71.2833	0	0	0
2	1	26.0	0	0	7.9250	0	0	1
3	1	35.0	1	0	53.1000	0	0	0
4	0	35.0	0	0	8.0500	1	0	1

CODE

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
zscaling = scaler.fit_transform(titanic_data[['Age','Fare']])
titanic_data[['Age','Fare']] = zscaling
titanic_data.head(5)
```

OUTPUT

(vii) Apply Z-score scaling with StandardScalar if mean and standard deviation are 0 and 1, respectively (optional in this assignment)

```
In [22]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         zscaling = scaler.fit_transform(titanic_data[['Age','Fare']])
         titanic_data[['Age','Fare']] = zscaling
         titanic_data.head(5)
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	-0.592494	1	0	-0.502445
1	1	1	female	0.638776	1	0	0.786845
2	1	3	female	-0.284677	0	0	-0.488854
3	1	1	female	0.407912	1	0	0.420730
4	0	3	male	0.407912	0	0	-0.486337

(vii) Apply Z-score scaling with StandardScalar if mean and standard deviation are 0 and 1, respectively

CODE

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
zscaling = scaler.fit_transform(titanic_data[['Age','Fare']])
titanic_data[['Age','Fare']] = zscaling
titanic_data.head(5)
```

OUTPUT

In [22]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
zscaling = scaler.fit_transform(titanic_data[['Age','Fare']])
titanic_data[['Age','Fare']] = zscaling
titanic_data.head(5)
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	-0.592494	1	0	-0.502445
1	1	1	female	0.638776	1	0	0.786845
2	1	3	female	-0.284677	0	0	-0.488854
3	1	1	female	0.407912	1	0	0.420730
4	0	3	male	0.407912	0	0	-0.486337

IV. TRAINING & TESTING

Write and execute Python scripts to do the followings:

(i) Make a ratio of 30% and 70% for test and train dataset.

CODE

```
import sklearn as sk
from sklearn.model_selection import train_test_split
X = conc_titanic_data.drop('Survived', axis=1)
y = conc_titanic_data['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
print("Train set size:", len(X_train))
print("Test set size:", len(X_test))
```

OUTPUT

```
In [23]: import sklearn as sk

In [24]: from sklearn.model_selection import train_test_split

In [25]: X = conc_titanic_data.drop('Survived', axis=1)
         y = conc_titanic_data['Survived']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
         print("Train set size:", len(X_train))
         print("Test set size:", len(X_test))

         Train set size: 623
         Test set size: 268
```

(ii) Apply the following models:

(a) Logistic regression

(b) Neural Networks classifier

CODE

```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train)
y_log_predicted=log_model.predict(X_test)
from sklearn.neural_network import MLPClassifier
nn_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000)
nn_model.fit(X_train, y_train)
y_nn_predicted=nn_model.predict(X_test)
```

OUTPUT

```
In [26]: from sklearn.linear_model import LogisticRegression

In [27]: log_model = LogisticRegression(max_iter=1000)
         log_model.fit(X_train, y_train)
         y_log_predicted=log_model.predict(X_test)

In [28]: from sklearn.neural_network import MLPClassifier
         nn_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000)
         nn_model.fit(X_train, y_train)
         y_nn_predicted=nn_model.predict(X_test)
```

V. PERFORMANCE STUDY

Write and execute Python scripts to do the followings:

(i) Plot confusion matrix.

Logistic Regression

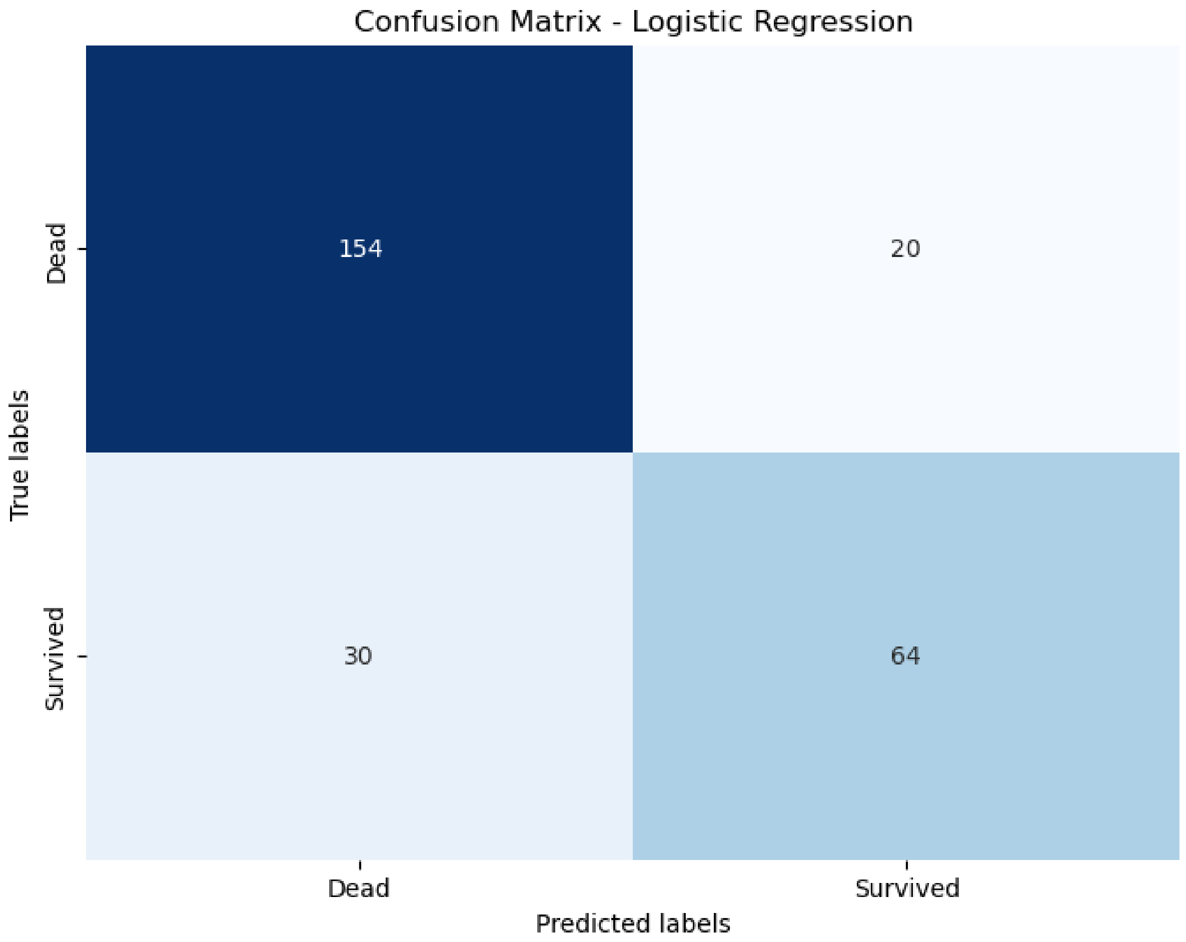
CODE

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
log_matrix = confusion_matrix(y_test, y_log_predicted)
plt.figure(figsize=(8, 6))
sns.heatmap(log_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,xticklabels=['Dead',
'Survived'], yticklabels=['Dead', 'Survived'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

OUTPUT

```
In [29]: from sklearn.metrics import confusion_matrix
         import matplotlib.pyplot as plt
         import seaborn as sns
```

```
In [30]: log_matrix = confusion_matrix(y_test, y_log_predicted)
         plt.figure(figsize=(8, 6))
         sns.heatmap(log_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,xticklabels=['Dead', 'Survived'],
                     yticklabels=['Dead', 'Survived'])
         plt.xlabel('Predicted labels')
         plt.ylabel('True labels')
         plt.title('Confusion Matrix - Logistic Regression')
         plt.show()
```



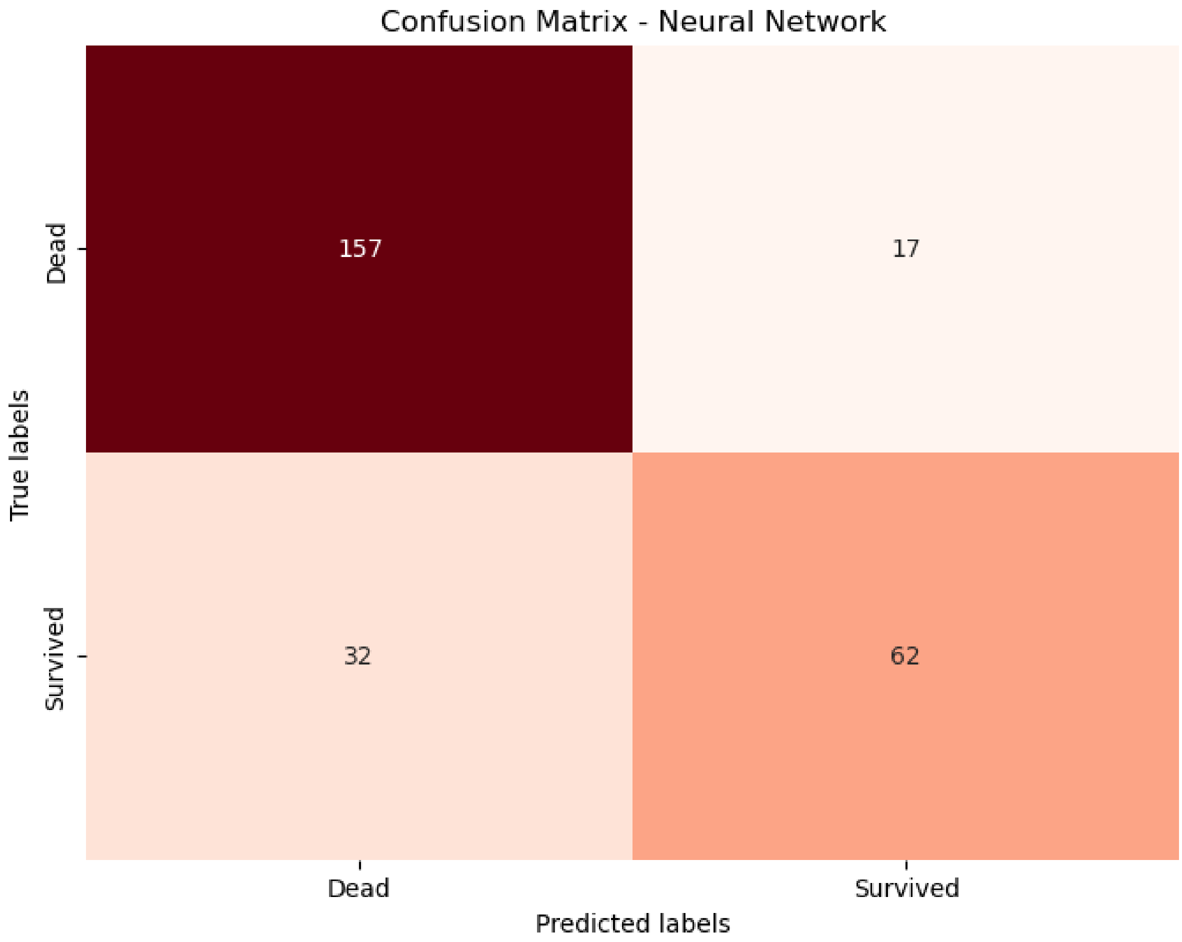
Neural Network

CODE

```
nn_matrix = confusion_matrix(y_test, y_nn_predicted)
plt.figure(figsize=(8, 6))
sns.heatmap(nn_matrix, annot=True, fmt='d', cmap='Reds', cbar=False,
            yticklabels=['Dead', 'Survived'], xticklabels=['Dead', 'Survived'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Neural Network')
plt.show()
```

OUTPUT

```
In [34]:
nn_matrix = confusion_matrix(y_test, y_nn_predicted)
plt.figure(figsize=(8, 6))
sns.heatmap(nn_matrix, annot=True, fmt='d', cmap='Reds', cbar=False,
            yticklabels=['Dead', 'Survived'], xticklabels=['Dead', 'Survived'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Neural Network')
plt.show()
```



(ii) Find Precision, Recall, F1score, and Accuracy.

<u>CODE</u>

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score
log_accuracy = accuracy_score(y_test, y_log_predicted)
log_precision = precision_score(y_test, y_log_predicted)
log_recall = recall_score(y_test, y_log_predicted)
log_f1 = f1_score(y_test, y_log_predicted)

nn_accuracy = accuracy_score(y_test, y_nn_predicted)
nn_precision = precision_score(y_test, y_nn_predicted)
nn_recall = recall_score(y_test, y_nn_predicted)
nn_f1 = f1_score(y_test, y_nn_predicted)

print("Logistic Regression:")
print("Accuracy:", round(log_accuracy, 3))
print("Precision:", round(log_precision, 3))
print("Recall:", round(log_recall, 3))
print("F1 Score:", round(log_f1, 3))
print("\nNeural Network:")
print("Accuracy:", round(nn_accuracy, 3))
print("Precision:", round(nn_precision, 3))
print("Recall:", round(nn_recall, 3))
print("F1 Score:", round(nn_f1, 3))
```

OUTPUT

```
In [32]: from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

```
In [38]: log_accuracy = accuracy_score(y_test, y_log_predicted)
         log_precision = precision_score(y_test, y_log_predicted)
         log_recall = recall_score(y_test, y_log_predicted)
         log_f1 = f1_score(y_test, y_log_predicted)

         nn_accuracy = accuracy_score(y_test, y_nn_predicted)
         nn_precision = precision_score(y_test, y_nn_predicted)
         nn_recall = recall_score(y_test, y_nn_predicted)
         nn_f1 = f1_score(y_test, y_nn_predicted)

         print("Logistic Regression:")
         print("Accuracy:", round(log_accuracy, 3))
         print("Precision:", round(log_precision, 3))
         print("Recall:", round(log_recall, 3))
         print("F1 Score:", round(log_f1, 3))
         print("\nNeural Network:")
         print("Accuracy:", round(nn_accuracy, 3))
         print("Precision:", round(nn_precision, 3))
         print("Recall:", round(nn_recall, 3))
         print("F1 Score:", round(nn_f1, 3))
```

```
Logistic Regression:
Accuracy: 0.813
Precision: 0.762
Recall: 0.681
F1 Score: 0.719

Neural Network:
Accuracy: 0.817
Precision: 0.785
Recall: 0.66
F1 Score: 0.717
```