Sardar Patel Institute of Technology,Mumbai
Department of Electronics and Telecommunication Engineering
B.E. Sem-VII (2022-2023) Data Analytics

Experiment: Exploratory Data Analysis (EDA)

Name: Rahul Alshi **UID:** 2019110001 **BE ETRX** **DA LAB 1**

Aim: Perform Exploratory Data Analysis (EDA) on Titanic Crash Passengers and Survivors dataset

Dataset Overview

The dataset 'train (1).csv' contains 12 columns :

- PassengerId : The Id of the passenger
- Survived : No of survived people or not
- Pclass : The class of the seat
- Name : Name of the Passenger
- Sex : Gender of the Passenger
- Age : Age of the Passenger
- SibSp : Family relations
- Parch : No of Parents/No of children aboard
- Ticket : The ticket No.
- Fare : The fare price
- Cabin : The cabin No.
- Embarked : from where the traveller started from

```
[37] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Our dataset is given as a CSV file. Pandas provides an easy way to read our file with `read_csv`. The path of the file to read is relative to our notebook file. The path can also be an URL, supporting HTTP, FTP and also S3 if your data is stored inside an AWS S3 Bucket!

```
[38] data = pd.read_csv('/content/sample_data/train (1).csv')
```

The first thing we will check is the size of our dataset. We can use `info()` to get the number of entries of each column.

Successfully imported the necessary libraries and the dataset into the notebook

The first thing we will check is the size of our dataset. We can use `info()` to get the number of entries of each column.

```
data.info()
```

```
<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
```

Now we know how many data is inside our file. Pandas is smart enough to parse the column titles by itself and estimate the data types of each column.

You may be curious how the data looks like. Let's see by using `head()`, which will print the first 5 rows.

```
[40] data.head()
```

	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

The dataset has 891 rows and 12 columns to work with for EDA.

We can access a column of our dataset by using bracket notation and the name of the column.

```
[41] data['PassengerId']
```

```
0        1
1        2
2        3
3        4
4        5
      ...
886    887
887    888
888    889
889    890
890    891
Name: PassengerId, Length: 891, dtype: int64
```

For categorical features like `sex`, you can also get the distributions of each value by using `value_counts()`.

```
[42] data.value_counts()
```

```
PassengerId  Survived  Pclass  Name                                                Sex     Age   SibSp  Parch  Ticket
Fare      Cabin  Embarked
2            1         1       Cumings, Mrs. John Bradley (Florence Briggs Thayer)  female  38.0  1      0      PC 17599
71.2833   C85    C           1
572          1         1       Appleton, Mrs. Edward Dale (Charlotte Lamson)        female  53.0  2      0      11769
51.4792   C101   S           1
578          1         1       Silvey, Mrs. William Baird (Alice Munger)            female  39.0  1      0      13507
55.9000   E44    S           1
582          1         1       Thayer, Mrs. John Borland (Marian Longstreth Morris) female  39.0  1      1      17421
110.8833  C68    C           1
584          0         1       Ross, Mr. John Hugo                                  male    36.0  0      0      13049
40.1250   A10    C           1

..
328          1         2       Ball, Mrs. (Ada E Hall)                              female  36.0  0      0      28551
13.0000   D      S           1
330          1         1       Hippach, Miss. Jean Gertrude                         female  16.0  0      1      111361
57.9792   B18    C           1
332          0         1       Partner, Mr. Austen                                  male    45.5  0      0      113043
28.5000   C124   S           1
333          0         1       Graham, Mr. George Edward                            male    38.0  0      1      PC 17582
153.4625  C91    S           1
890          1         1       Behr, Mr. Karl Howell                                male    26.0  0      0      111369
30.0000   C148   C           1
Length: 183, dtype: int64
```

But what about numerical values? It definitly makes no sense to count each distinct value. Therefore, we can use `describe()`.

```
[43] data.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

This works for the whole dataframe as well. Pandas knows which values are numerical based on the datatype and hides the categorical features for you.

```
[44] data.nunique()
```

```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

```
[45] student = data.drop(['Survived'],axis=1)
```

The above statement returns a new dataframe (not a copy, modifying this data will modify the original as well), which can be accessed like before. Let's see how the numerical distribution is for our females.

```
[46] student.head
```

```
<bound method NDFrame.head of      PassengerId  Pclass                                               Name  \
0              1       3                            Braund, Mr. Owen Harris
1              2       1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2              3       3                             Heikkinen, Miss. Laina
3              4       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4              5       3                           Allen, Mr. William Henry
..           ...     ...                                                ...
886          887       2                              Montvila, Rev. Juozas
887          888       1                       Graham, Miss. Margaret Edith
888          889       3           Johnston, Miss. Catherine Helen "Carrie"
889          890       1                              Behr, Mr. Karl Howell
890          891       3                                Dooley, Mr. Patrick

        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0      male  22.0      1      0         A/5 21171   7.2500   NaN        S
1    female  38.0      1      0          PC 17599  71.2833   C85        C
2    female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3    female  35.0      1      0            113803  53.1000  C123        S
4      male  35.0      0      0            373450   8.0500   NaN        S
..      ...   ...    ...    ...               ...      ...   ...      ...
886    male  27.0      0      0            211536  13.0000   NaN        S
887  female  19.0      0      0            112053  30.0000   B42        S
888  female   NaN      1      2        W./C. 6607  23.4500   NaN        S
```

```
[78] student.head
```

```
<bound method NDFrame.head of      PassengerId  Pclass                                               Name  \
0              1       3                            Braund, Mr. Owen Harris
1              2       1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2              3       3                             Heikkinen, Miss. Laina
3              4       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4              5       3                           Allen, Mr. William Henry
..           ...     ...                                                ...
886          887       2                              Montvila, Rev. Juozas
887          888       1                       Graham, Miss. Margaret Edith
888          889       3           Johnston, Miss. Catherine Helen "Carrie"
889          890       1                              Behr, Mr. Karl Howell
890          891       3                                Dooley, Mr. Patrick

        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0      male  22.0      1      0         A/5 21171   7.2500   NaN        S
1    female  38.0      1      0          PC 17599  71.2833   C85        C
2    female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3    female  35.0      1      0            113803  53.1000  C123        S
4      male  35.0      0      0            373450   8.0500   NaN        S
..      ...   ...    ...    ...               ...      ...   ...      ...
886    male  27.0      0      0            211536  13.0000   NaN        S
887  female  19.0      0      0            112053  30.0000   B42        S
888  female   NaN      1      2        W./C. 6607  23.4500   NaN        S
889    male  26.0      0      0            111369  30.0000  C148        C
890    male  32.0      0      0            370376   7.7500   NaN        Q

[891 rows x 11 columns]>
```

Next, We will explore numbers of NULL values or missing values the dataset has.

We can also create new rows. Specify the new column name in brackets and provide a function to set the data. We will create a new column containing True or False, wheather or not the person is below 30.

```
[47] data.isna().any()
```

```
PassengerId    False
Survived       False
Pclass         False
Name           False
Sex            False
Age             True
SibSp          False
Parch          False
Ticket         False
Fare           False
Cabin           True
Embarked        True
dtype: bool
```

```
[48] data.isna().sum().sort_values(ascending=False)
```

```
Cabin          687
Age            177
Embarked         2
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
SibSp            0
Parch            0
```

We see that the column Video Count has 866 null values lets drop those values.

```
[49] data.dropna(axis=0,inplace=True)
```

```
[51] data.shape

(183, 12)
```

After dropping the rows with missing values , the dataset has 183 rows and 12 columns to work upon.

Now, Let us import seaborn for data visualization

Visualize Data

To visualize our data, we will use Seaborn, a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. Let's import it.

```
[83] import seaborn as sns
```

To see our charts directly in our notebook, we have to execute the following:

```
[84] %matplotlib inline
     sns.set()
     sns.set_context('talk')
```

Seaborn together with Pandas makes it pretty easy to create charts to analyze our data. We can pass our Dataframes and Series directly into Seaborn methods. We will see how in the following sections.

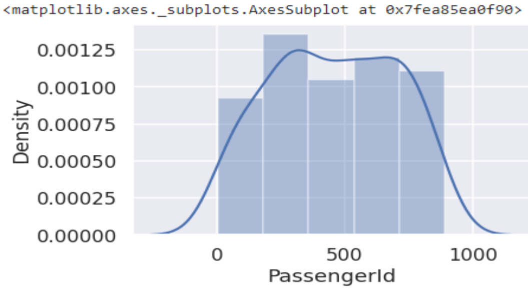
Let us visualize the displot:

Univariate Plotting

Let's start by visualizing the distribution of the age our our people. We can achieve this with a simple method called `distplot` by passing our series of ages as argument.

```
[85] sns.distplot(data['PassengerId'],bins=5)

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function a
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fea85ea0f90>
```



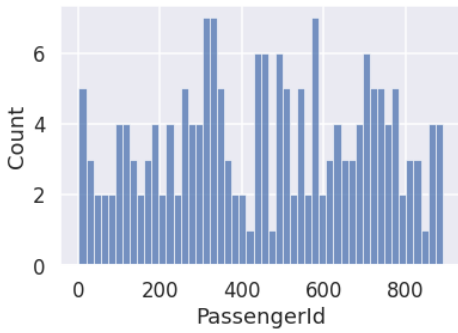
As we can see the density is highest for the passenger Id between 0 to 500 and it decreases as we go to 1000.

Now let us see the histogram

The chart above calculates a kernel density as well. To get a real histogram, we have to disable the `kde` feature. We can also increase to number of buckets for our histogram by setting `bins` to 50.

```
[86] sns.histplot(data['PassengerId'],bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea84ec9450>
```



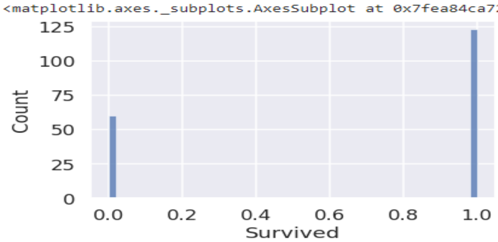
Interesting! The ages of the people in this dataset seem to end with two or seven.

We can do the same for every numerical column, e.g. the years of marriage.

As the bins are set to 50 , the histogram shows that the count goes to highest for the passenger Id’s between 200 to 600. We can plot the histogram for other attributes as well.

```
sns.histplot(data['Survived'],bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea84ca72d0>
```



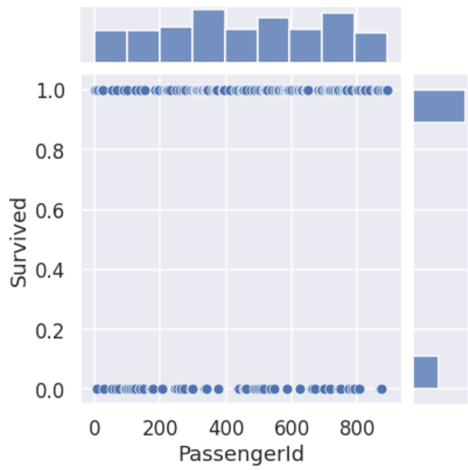
The average age of our people is around 32, but the most people are married for more than 14 years!

Now let us see some examples of bivariate plotting by plotting the join plot.

Numbers get even more interesting when we can compare them to other numbers! Lets start comparing the number of years married vs the number of affairs. Seaborn provides us with a method called `jointplot` for this use case.

```
[88] sns.jointplot(data['PassengerId'],data['Survived'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword ar
  FutureWarning
<seaborn.axisgrid.JointGrid at 0x7fea84ba7ed0>
```

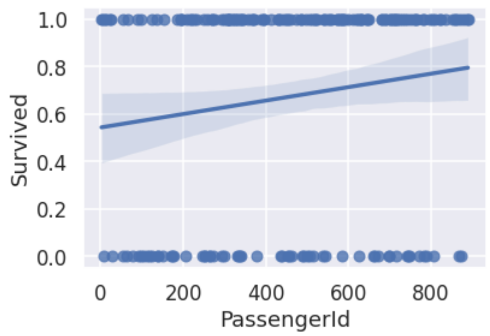


The above joined plot gives us the relation between the passenger Id and survived people in the crash.

To get a better feeling of how the number of affairs is affected by the number of years married, we can use a regression model by specifying `kind` as `reg`.

```
[89] sns.regplot(x='PassengerId',y='Survived',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea84abd6d0>
```



We can also use a kernel to kompare the density of two columns against each other, e.g. `age` and `ym`.

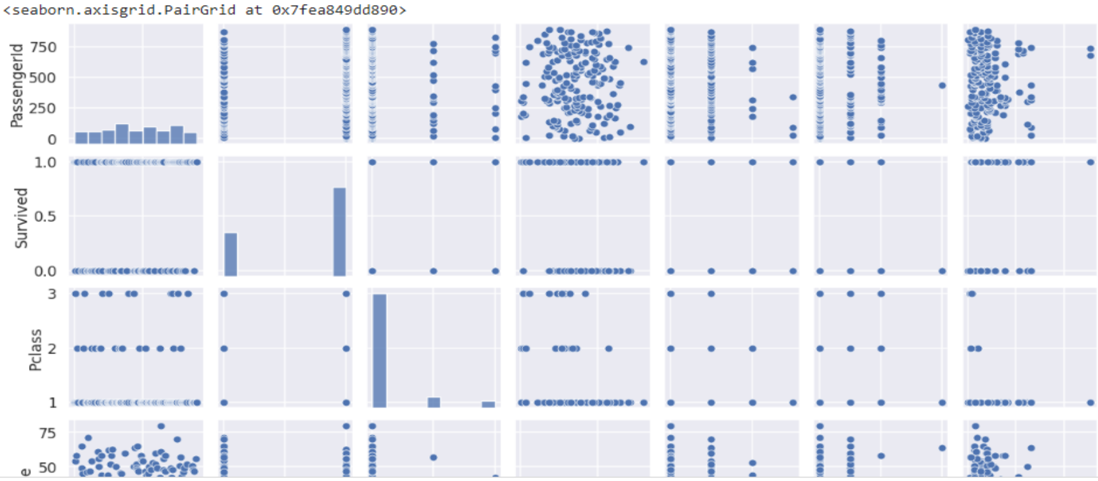
The above regression gives the relation between passenger Id and survived and the coefficient is minimum at 0.5 which goes to to 0.7 for passeneger Id's upto 1000.

Now let us visualize the pairplot

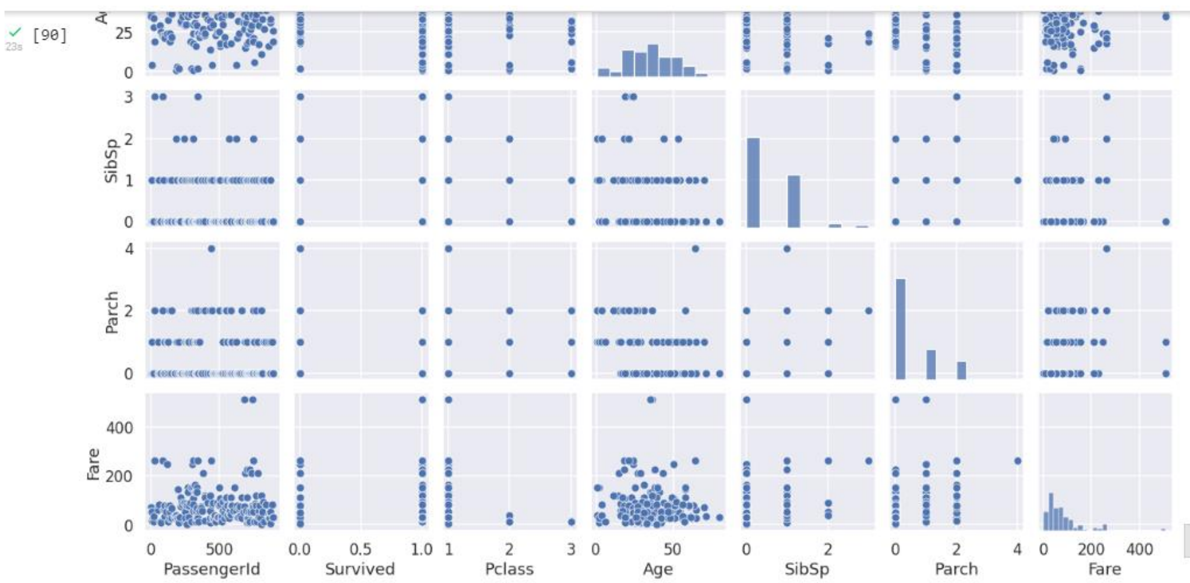
We can get an even better comparison by plotting everything vs everything! Seaborn provides this with the `pairplot` method.

```
[90] correlation=data.corr()
     sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7fea849dd890>
```



1s completed at 12:24 PM



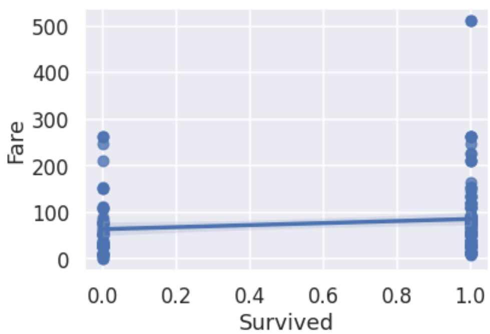
You won't see any special in this data. We need to separate them by some kind of criteria. We can use our categorical values to do this! Seaborn uses a parameter called `hue` to do this. Let's separate our data by `sex` first. To make things even more interesting, let's create a regression for every plot, too!

As we can see the jointplot for several attributes of the dataset , for the age attribue the density is high whereas for survived and Pclass is low.

You won't see any special in this data. We need to separate them by some kind of criteria. We can use our categorical values to do this! Seaborn uses a parameter called `hue` to do this. Let's separate our data by `sex` first. To make things even more interesting, let's create a regression for every plot, too!

```
sns.regplot(x='Survived',y='Fare',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea831d6590>
```



The above regression plot is for two different attrobutes Fare and Survived.

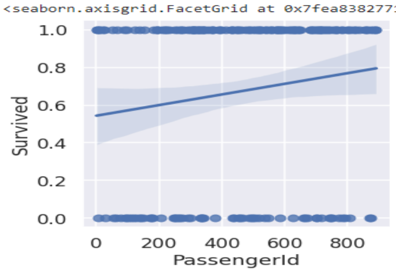
Now let us understand the lmplot which compares the two attributes and give the relation between them.

To get even better separation, we can use `lmplot` to compare just the fields we need.

Let's say we're interested in the number of affairs vs years married. We also whant to separate them by `sex`, `child` and `religious`. We will use `sns.lmplot(x="ym", y="nbaffairs", hue="sex", col="child", row="religious", data=affairs)` to achieve this.

```
[92] sns.lmplot(x="PassengerId", y="Survived",data=data)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fea83827710>
```



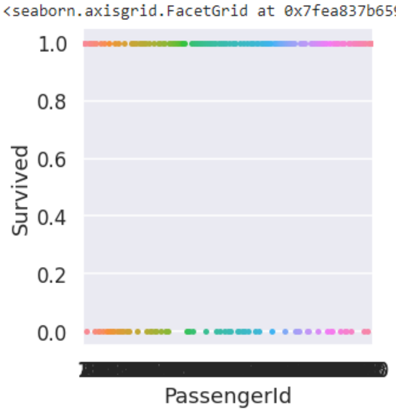
The lmplot for the above data set gives the comparision between survived and passenger Id for the folowing titanic crash.

Here are some categorical plots to explore the dataset even further.

```
[ ]
```

```
sns.catplot(x='PassengerId',y='Survived',data=data)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fea837b6590>
```



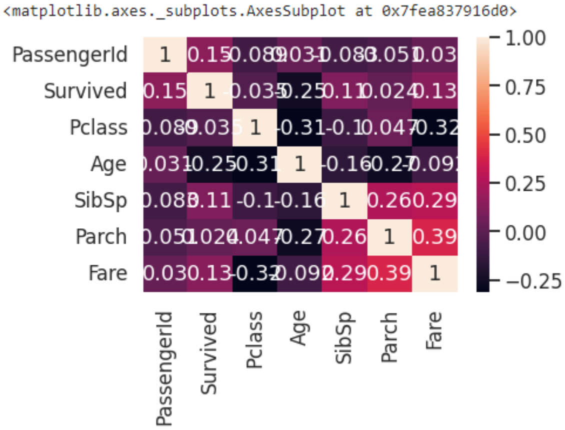
We can also get the correlations between the values by using Pandas builtin method `corr()`.

```
[94] correlation=data.corr()
```

Feed these stats into Seaborns `heatmap` method will provide us with the visual representation.

```
[95] sns.heatmap(correlation,xticklabels=correlation.columns, yticklabels=correlation.columns, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea837916d0>
```



✓ 1s completed at 3:22 PM

The above plots are catplot and the heatmap which even explore the data further.The catplot shows frequencies of the categories of one , two or three categorical variables for eg in our data set the Passenger and the survived.The heat map is coloured map which basically shows the relationship between variables one plotted on each axis.

Conclusion:

1. Performed EDA for Titanic Crash Passengers Data set .
2. Exploratory Data Analysis refers to the critical process of performing critical investigations on data so as to discover patterns or to spot anomalities.
3. Few insights we found from the dataset:
    - For passenger Id between 400 to 500 the kernel density was the highest.
    - Th count goes the highest for passenger Id’s between 200 to 400 and 600
    - The regression coefficient of survived increases the passenger Id’s increases from 200 to 800 , with a minimum of 0.5
    - The survival count is highest of 125 at a coefficient of 1.0