Titanic survivors, a guide for your first Data Science project

CLASSIFICATION DATA EXPLORATION DATA SCIENCE DATA VISUALIZATION MACHINE LEARNING PANDAS PROJECT STRUCTURED DATA SUPERVISED

Introduction

In this article, we are going to go through the popular Titanic dataset and try to predict whether a person survived the shipwreck. You can get this dataset from Kaggle, linked here. This article will be focused on how to think about these projects, rather than the implementation. A lot of the beginners are confused as to how to start when to end and everything in between, I hope this article acts as a beginner's handbook for you. I suggest you practice the project in Kaggle itself.

The Goal: Predict whether a passenger survived or not. **0** for not surviving, **1** for surviving.

Describing the data

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Image Source: Kaggle

In this article, we will do some basic data analysis, then some feature engineering, and in the end-use some of the popular models for prediction. Let's get started.

Data Analysis

Step 1: Importing basic libraries

```
import numpy as np import pandas as pd import seaborn as sns import matplotlib.pyplot as plt %matplotlib inline
```

Step 2: Reading the data

```
training = pd.read_csv('/kaggle/input/titanic/train.csv') test = pd.read_csv('/kaggle/input/titanic/test.csv')
```

```
training['train_test'] = 1 test['train_test'] = 0 test['Survived'] = np.NaN all_data = pd.concat([training,test])
```

```
all_data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'train_test'],
      dtype='object')
```

Step 3: Data Exploration

In this section we will try to draw insights from the Data, and get familiar with it, so we can create more efficient models.

```
training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 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 
 12  train_test   891 non-null    int64  
dtypes: float64(2), int64(6), object(5)
memory usage: 90.6+ KB
```

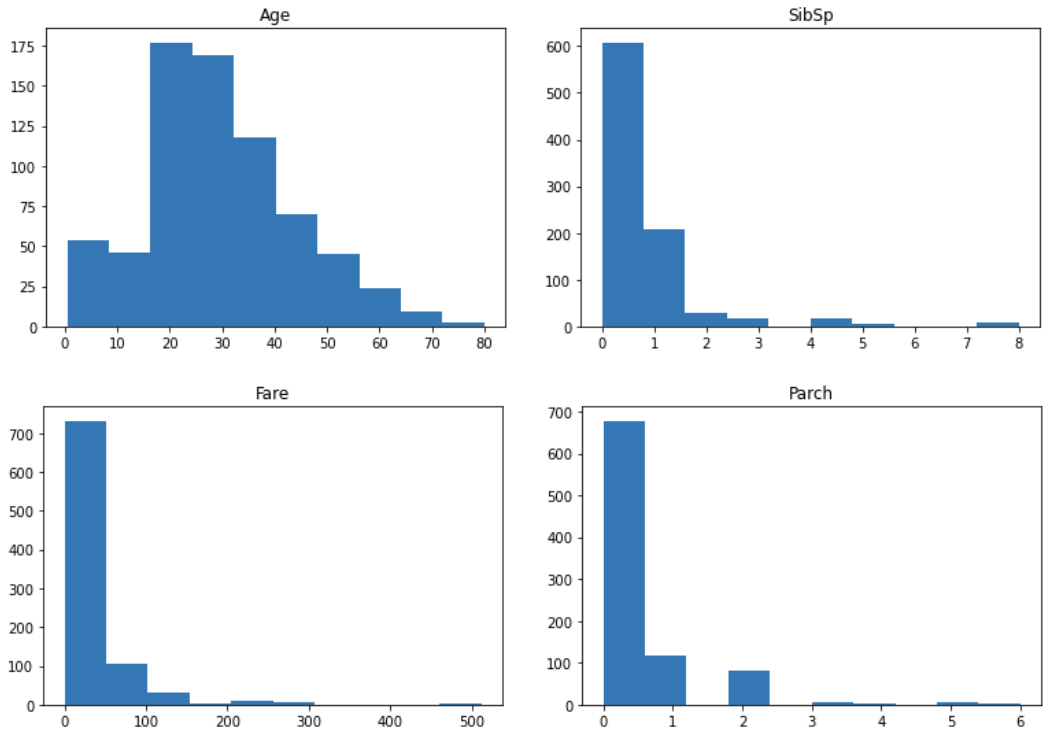
```
training.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	train_test
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	891.0
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208	1.0
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429	0.0
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	1.0
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400	1.0
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200	1.0
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000	1.0
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	1.0

```
# seperate the data into numeric and categorical df_num = training[['Age','SibSp','Parch','Fare']] df_cat = training[['Survived','Pclass','Sex','Ticket','Cabin','Embarked']]
```

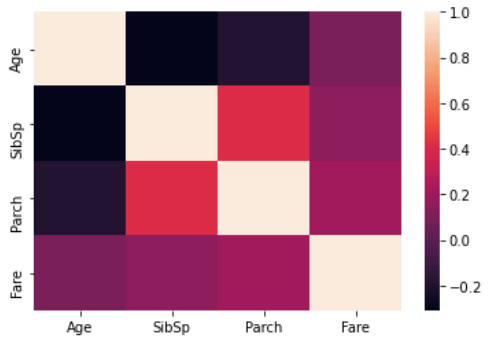
Now let’s make plots of the numeric data:

```
for i in df_num.columns: plt.hist(df_num[i]) plt.title(i) plt.show()
```



So as you can see, most of the distributions are scattered, except Age, it's pretty normalized. We might consider normalizing them later on. Next, we plot a correlation heatmap between the numeric columns:

```
sns.heatmap(df_num.corr())
```



Here we can see that Parch and SibSp has a higher correlation, which generally makes sense since Parents are more likely to travel with their multiple kids and spouses tend to travel together. Next, let us compare survival rates across the numeric variables. This might reveal some interesting insights:

```
pd.pivot_table(training, index = 'Survived', values = ['Age','SibSp','Parch','Fare'])
```

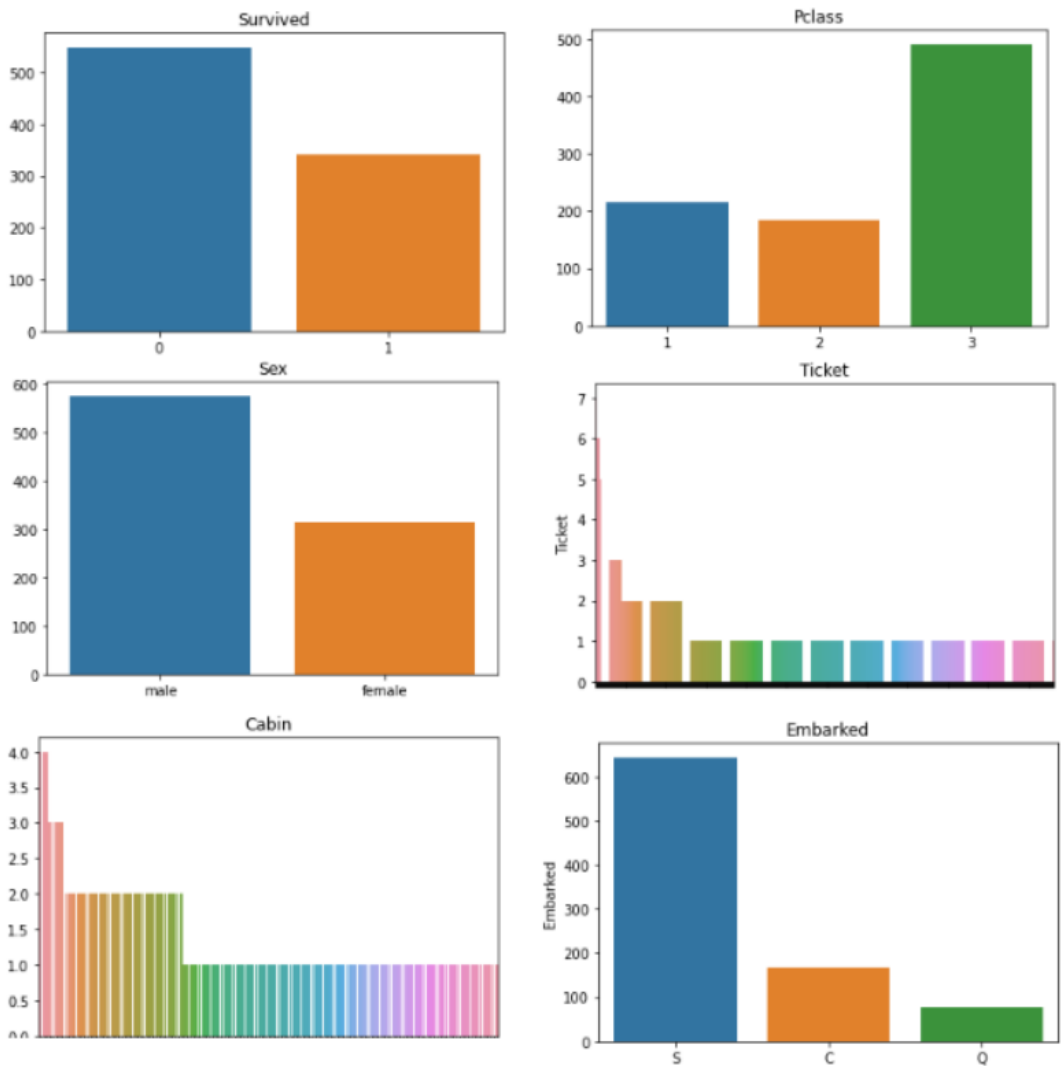
	Age	Fare	Parch	SibSp
Survived				
0	30.626179	22.117887	0.329690	0.553734
1	28.343690	48.395408	0.464912	0.473684

The inference we can draw from this table is:

1. The average age of survivors is 28, so young people tend to survive more.
2. People who paid higher fare rates were more likely to survive, more than double. This might be the people traveling in first-class. Thus the rich survived, which is kind of a sad story in this scenario.
3. In the 3rd column, If you have parents, you had a higher chance of surviving. So the parents might've saved the kids before themselves, thus explaining the rates
4. And if you are a child, and have siblings, you have less of a chance of surviving

Now we do a similar thing with our categorical variables:

```
for  i  in  df_cat.columns:  sns.barplot(df_cat[i].value_counts().index,df_cat[i].value_counts()).set_title(i)
plt.show()
```



The Ticket and Cabin graphs look very messy, we might have to feature engineer them! Other than that, the rest of the graphs tells us:

1. Survived: Most of the people died in the shipwreck, only around 300 people survived.
2. Pclass: The majority of the people traveling, had tickets to the 3rd class.
3. Sex: There were more males than females aboard the ship, roughly double the amount.

4. Embarked: Most of the passengers boarded the ship from Southampton.

Now we will do something similar to the pivot table above, but with our categorical variables, and compare them against our dependent variable, which is if people survived:

```
print(pd.pivot_table(training, index = 'Survived', columns = 'Pclass',                    values = 'Ticket' ,aggfunc ='count')) print() print(pd.pivot_table(training, index = 'Survived', columns = 'Sex',
           values = 'Ticket' ,aggfunc ='count')) print() print(pd.pivot_table(training, index = 'Survived', columns = 'Embarked',                    values = 'Ticket' ,aggfunc ='count'))
```

```
Pclass      1   2    3
Survived
0          80  97  372
1         136  87  119

Sex       female  male
Survived
0             81   468
1            233   109

Embarked   C   Q    S
Survived
0         75  47  427
1         93  30  217
```

1. Pclass: Here we can see a lot more people survived from the First class than the Second or the Third class, even though the total number of passengers in the First class was much much less than the Third class. Thus our previous assumption that the rich survived is confirmed here, which might be relevant to model building.
2. Sex: Most of the women survived, and the majority of the male died in the shipwreck. So it looks like the saying “Woman and children first” actually applied in this scenario.
3. Embarked: This doesn’t seem much relevant, maybe if someone was from “*Cherbourg*” had a higher chance of surviving.

Step 4: Feature Engineering

We saw that our ***ticket*** and ***cabin*** data don’t really make sense to us, and this might hinder the performance of our model, so we have to simplify some of this data with feature engineering.

If we look at the actual cabin data, we see that there’s basically a letter and then a number. The letters might signify what type of cabin it is, where on the ship it is, which floor, which Class it is for, etc. And the numbers might signify the Cabin number. Let us first split them into individual cabins and see whether someone owned more than a single cabin.

```
df_cat.Cabin training['cabin_multiple'] = training.Cabin.apply(lambda x: 0 if pd.isna(x)  else len(x.split(' '))) training['cabin_multiple'].value_counts()
```

```
0    687
1    180
2     16
3      6
4      2
Name: cabin_multiple, dtype: int64
```

It looks like the vast majority did not have individual cabins, and only a few people owned more than one cabins. Now let's see whether the survival rates depend on this:

```
pd.pivot_table(training, index = 'Survived', columns = 'cabin_multiple', values = 'Ticket' ,aggfunc ='count')
```

cabin_multiple	0	1	2	3	4
Survived					
0	481.0	58.0	7.0	3.0	NaN
1	206.0	122.0	9.0	3.0	2.0

Next, let us look at the actual letter of the cabin they were in. So you could expect that the cabins with the same letter are roughly in the same locations, or on the same floors, and logically if a cabin was near the lifeboats, they had a better chance of survival. Let us look into that:

```
# n stands for null # in this case we will treat null values like it's own category training['cabin_adv'] = training.Cabin.apply(lambda x: str(x)[0]) #comparing survival rates by cabin print(training.cabin_adv.value_counts()) pd.pivot_table(training,index='Survived',columns='cabin_adv', values = 'Name', aggfunc='count')
```

```
n    687
C     59
B     47
D     33
E     32
A     15
F     13
G      4
T      1
Name: cabin_adv, dtype: int64
```

I did some future engineering on the *ticket* column and it did not yield many significant insights, which we don't already know, so I'll be skipping that part to keep the article concise. We will just divide the tickets into numeric and non-numeric for efficient usage:

```
training['numeric_ticket'] = training.Ticket.apply(lambda x: 1 if x.isnumeric() else 0) training['ticket_letters'] = training.Ticket.apply(lambda x: ''.join(x.split(' ')[:-1]) .replace('.','').replace('/','') .lower() if len(x.split(' ')[:-1]) >0 else 0)
```

Another interesting thing we can look at is the title of individual passengers. And whether it played any role in them getting a seat in the lifeboats.

```
training.Name.head(50) training['name_title'] = training.Name.apply(lambda x: x.split(',')[1] .split('.') [0].strip()) training['name_title'].value_counts()
```

```
Mr              517
Miss            182
Mrs             125
Master           40
Dr                7
Rev               6
Col               2
Mlle              2
Major             2
Jonkheer          1
Don               1
Lady              1
Capt              1
the Countess      1
Sir               1
Mme               1
Ms                1
Name: name_title, dtype: int64
```

As you can see, the ship was boarded by people of many different classes, this might be useful for us in our model.

Step 5: Data preprocessing for model

In this segment, we make our data, model-ready. The objectives we have to fulfill are listed below:

1. Drop the null values from the Embarked column
2. Include only relevant data
3. Categorically transform all of the data, using something called a transformer.
4. Impute data with the central tendencies for age and fare.
5. Normalize the *fare* column to have a more normal distribution.
6. using standard scaler scale data 0-1

Step 6: Model Deployment

Here we will simply deploy the various models with default parameters and see which one yields the best result. The models can further be tuned for better performance but are not in the scope of this one article. The models we will run are:

- Logistic regression
- K Nearest Neighbour
- Support Vector classifier

First, we import the necessary models

```
from sklearn.model_selection import cross_val_score from sklearn.linear_model import LogisticRegression from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import SVC
```

1) Logistic Regression

```
lr = LogisticRegression(max_iter = 2000) cv = cross_val_score(lr,X_train_scaled,y_train,cv=5) print(cv)
print(cv.mean())
```

```
[0.8258427  0.80898876 0.80337079 0.82022472 0.85310734]
0.8223068621849807
```

2) K Nearest Neighbour

```
knn = KNeighborsClassifier() cv = cross_val_score(knn,X_train_scaled,y_train,cv=5) print(cv) print(cv.mean())
```

```
[0.79775281 0.79213483 0.83146067 0.79775281 0.85310734]
0.8144416936456548
```

3) Support Vector Classifier

```
svc = SVC(probability = True) cv = cross_val_score(svc,X_train_scaled,y_train,cv=5) print(cv)
print(cv.mean())
```

```
[0.85393258 0.82022472 0.8258427  0.80337079 0.86440678]
0.8335555132355742
```

Therefore the accuracy of the models are:

- Logistic regression: 82.2%
- K Nearest Neighbour: 81.4%
- **SVC: 83.3%**

As you can see we get decent accuracy with all our models, but the best one is SVC. And voila, just like that you've completed your first data science project! Though there is so much more one can do to get better results, this is more than enough to help you get started and see how you think like a data scientist. I hope this walkthrough helped you, I had a great time doing the project myself and hope you enjoy it too. Cheers!!

Article Url - https://www.analyticsvidhya.com/blog/2021/05/titanic-survivors-a-guide-for-your-first-data-science-project/

[Sion]