Python for Data Scientists L15 (2) : Dataset Investigation with Python

Titanic Dataset Investigation with Python

Information about the Titanic dataset

We will use the Titanic data set, stored as CSV. The data consists of the following data columns:
PassengerId: Id of every passenger.
Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
Name: Name of passenger.
Sex: Gender of passenger.
Age: Age of passenger.
SibSp: Indication that passenger have siblings and spouse.
Parch: Whether a passenger is alone or have family.
Ticket: Ticket number of passenger.
Fare: Indicating the fare.
Cabin: The cabin of passenger.
Embarked: The embarked category.

Link: https://www.kaggle.com/c/titanic/data

Reading and checking the dataset

```
df = pd.read_csv("Data/train.csv")
print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S
5	6	0	3	...	8.4583	NaN	Q
6	7	0	1	...	51.8625	E46	S
7	8	0	3	...	21.0750	NaN	S
8	9	1	3	...	11.1333	NaN	S
9	10	1	2	...	30.0708	NaN	C

Reading and checking the dataset

```
df = pd.read_csv("Data/train.csv")
print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
```

[10 rows x 12 columns]

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
881	882	0	3	...	7.8958	NaN	S
882	883	0	3	...	10.5167	NaN	S
883	884	0	2	...	10.5000	NaN	S
884	885	0	3	...	7.0500	NaN	S
885	886	0	3	...	29.1250	NaN	Q
886	887	0	2	...	13.0000	NaN	S
887	888	1	1	...	30.0000	B42	S
888	889	0	3	...	23.4500	NaN	S
889	890	1	1	...	30.0000	C148	C
890	891	0	3	...	7.7500	NaN	Q

Reading and checking the dataset

```
df = pd.read_csv("Data/train.csv")
print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
```

[10 rows x 12 columns]

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

Reading and checking the dataset

```
df = pd.read_csv("Data/train.csv")
print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
```

```
[8 rows x 7 columns]
<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
```

Data Cleaning

```
missingValues= df.isnull().sum()
print(missingValues)
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

There are lots of dealing ways with missing values. In this project, we are going to use “ignore the tuple” and “fill it with median”.

 # We are going to ignore the “Cabin” column since %70 of that column is missing.
 # And we are going to fill the missing Ages with median value of that column (same for Embarked)

Data Cleaning

```
df1 = df.drop("Cabin", axis='columns')

df1['Age'] = df1['Age'].fillna((df1['Age'].median()))
print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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          891 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  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

Data Cleaning

```
missing_ports = df1[df1['Embarked'].isnull()]
print(missing_ports)
df1['Embarked'].fillna('S',inplace=True)
print(df1.info())
```

```
 PassengerId  Survived  Pclass  ...  Ticket  Fare  Embarked
61         62         1       1  ...  113572  80.0       NaN
829        830        1       1  ...  113572  80.0       NaN

[2 rows x 11 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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          891 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  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

Analysis of Data: General questions

What is the highest fares paid by passengers?

```
d1= df1.sort_values("Fare", ascending = False).head(10)
print(d1)
```

	PassengerId	Survived	Pclass	...	Ticket	Fare	Embarked
258	259	1	1	...	PC 17755	512.3292	C
737	738	1	1	...	PC 17755	512.3292	C
679	680	1	1	...	PC 17755	512.3292	C
88	89	1	1	...	19950	263.0000	S
27	28	0	1	...	19950	263.0000	S
341	342	1	1	...	19950	263.0000	S
438	439	0	1	...	19950	263.0000	S
311	312	1	1	...	PC 17608	262.3750	C
742	743	1	1	...	PC 17608	262.3750	C
118	119	0	1	...	PC 17558	247.5208	C

Analysis of Data: General questions

How many female and male were in this trip?

```
d2= df1['Sex'].value_counts()
print(d2)
```

[10 rows x 11 columns]
male 577
female 314
Name: Sex, dtype: int64

Analysis of Data: General questions

how many passengers are older than 18 years (How do I filter specific rows from a DataFrame?)

```
above_18 = df1[df1["Age"] > 18]
print(above_18.shape)
```

(752, 11)

Analysis of Data: General questions

What is the number of passengers in each cabin classes?

```
classes = df1["Pclass"].value_counts()
print(classes)
```

```
3   491
1   216
2   184
Name: Pclass, dtype: int64
```

Analysis of Data: General questions

Show only the names from the dataset

```
names = df1['Name']
print(names)
```

```
[5 rows x 11 columns]
0                                  Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                  Heikkinen, Miss. Laina
3            Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                                  Allen, Mr. William Henry
                                  ...
886                                Montvila, Rev. Juozas
887                         Graham, Miss. Margaret Edith
888             Johnston, Miss. Catherine Helen "Carrie"
889                                Behr, Mr. Karl Howell
890                                  Dooley, Mr. Patrick
```

Analysis of Data: General questions

Show only the names from the dataset

```
surnames = df1["Name"].str.split(",").str.get(0)
print(surnames)
```

```
Name: Name, Length: 891, dtype: object
0        Braund
1      Cumings
2     Heikkinen
3      Futrelle
4         Allen
       ...
886    Montvila
887      Graham
888    Johnston
889         Behr
890       Dooley
```

Analysis of Data: Investigating the survivors

Which gender had a better chance of survival?

```
table = pd.crosstab(df1['Survived'],df1['Sex'])
print (table)

print (df1.groupby('Sex').Survived.mean())
```

```
Sex      female  male
Survived
0          81   468
1         233   109
```

```
Sex
female    0.742038
male      0.188908
```

Analysis of Data: Investigating the survivors

Which social class had a better chance of survival?

```
table = pd.crosstab(df1['Survived'],df1['Pclass'])
print (table)

survivedPerClass = df1.groupby('Pclass').Survived.mean()
print((survivedPerClass))
```

```
Name: Survived, dtype: float64
Pclass     1   2    3
Survived
0         80  97  372
1        136  87  119


1    0.629630
2    0.472826
3    0.242363
```

Analysis of Data: Investigating the survivors

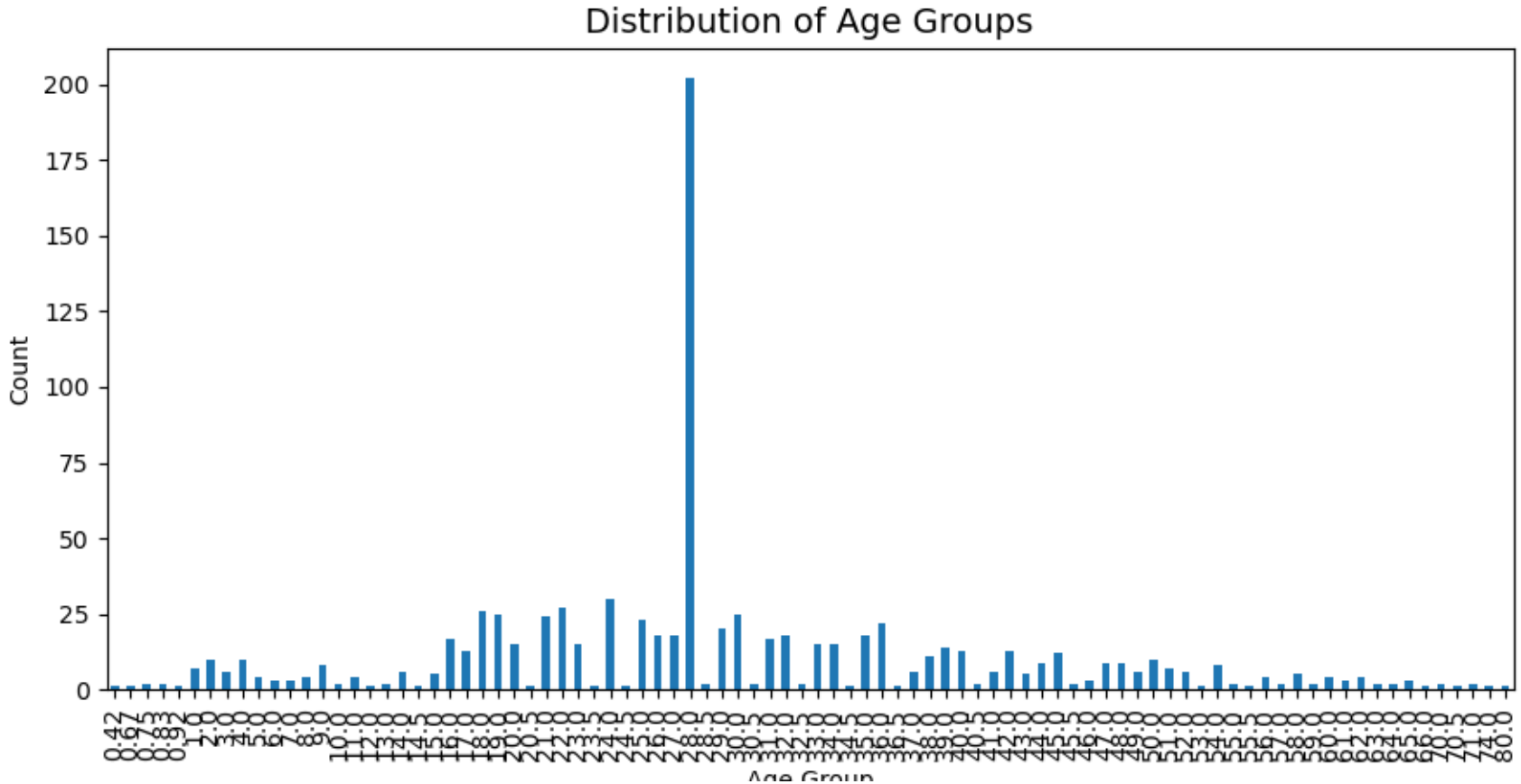
Which age group had a better chance of survival?

- First, we need to know the distribution of age groups:

```
#Distribution of Age Groups
df1.groupby(['Age']).size().plot(kind='bar',stacked=True)
plt.title("Distribution of Age Groups",fontsize=14)
plt.ylabel('Count')
plt.xlabel('Age Group')
plt.show()
```

Analysis of Data: Investigating the survivors

Distribution of age groups:



Analysis of Data: Investigating the survivors

Which age group had a better chance of survival?

- Now we can answer this question:


```
survivedPerAge = df1.groupby(['Age']).Survived.mean()
print(survivedPerAge)
```

```
Age
0.42    1.0
0.67    1.0
0.75    1.0
0.83    1.0
0.92    1.0
       ...
70.00   0.0
70.50   0.0
71.00   0.0
74.00   0.0
80.00   1.0
Name: Survived, Length: 88, dtype: float64
```

Analysis of Data: Investigating the survivors

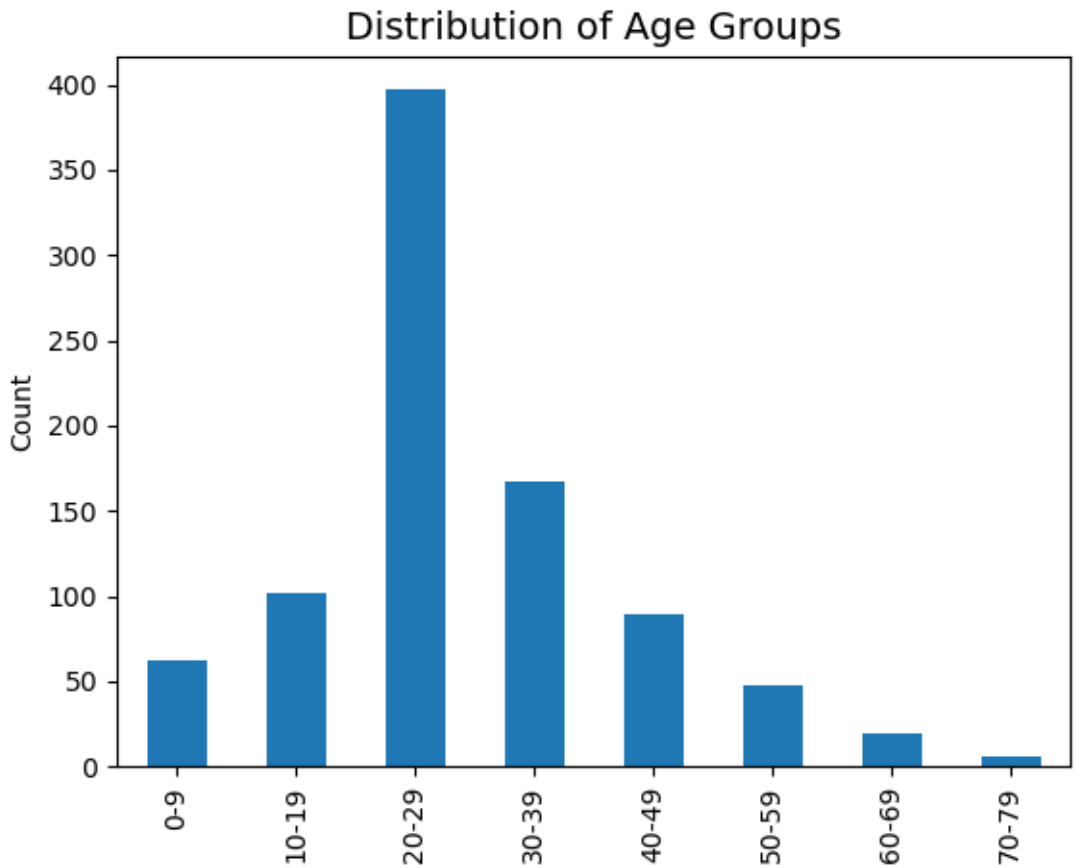
Which age group had a better chance of survival?

- Solution: Binning Ages

```
# Grouping / Binning Ages
# To make the ages easier to analyze, it would be a good idea to group
/ bin the ages.
# This way we can compare groups of ages instead of individual ages.
age_labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-
69', '70-79']
df1['Age'] = pd.cut(df1.Age, range(0, 81, 10), right=False,
labels=age_labels)
```

Analysis of Data: Investigating the survivors

Which age group had a better chance of survival?



Analysis of Data: Investigating the survivors

Which age group had a better chance of survival?

```
survivalPerAgeGroup =
df1.groupby(['Age']).Survived.mean()
print(survivalPerAgeGroup)
```

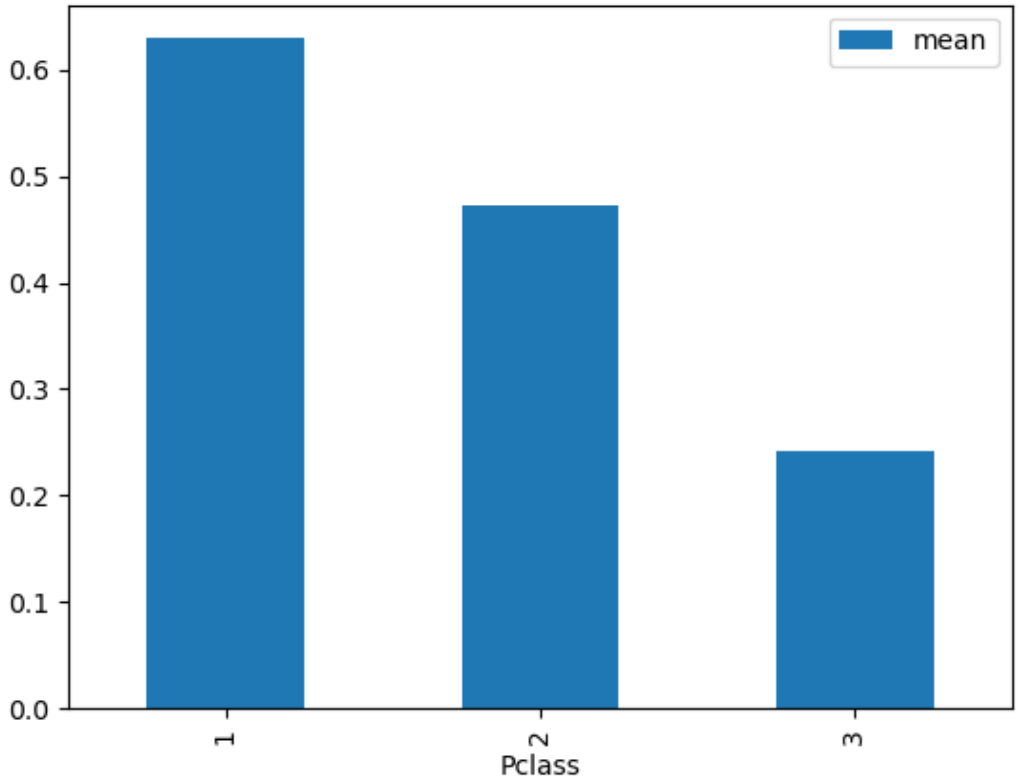
```
Age
0-9     0.612903
10-19   0.401961
20-29   0.324937
30-39   0.437126
40-49   0.382022
50-59   0.416667
60-69   0.315789
70-79   0.000000
Name: Survived, dtype: float64
```

Analysis of Data: Matplotlib report

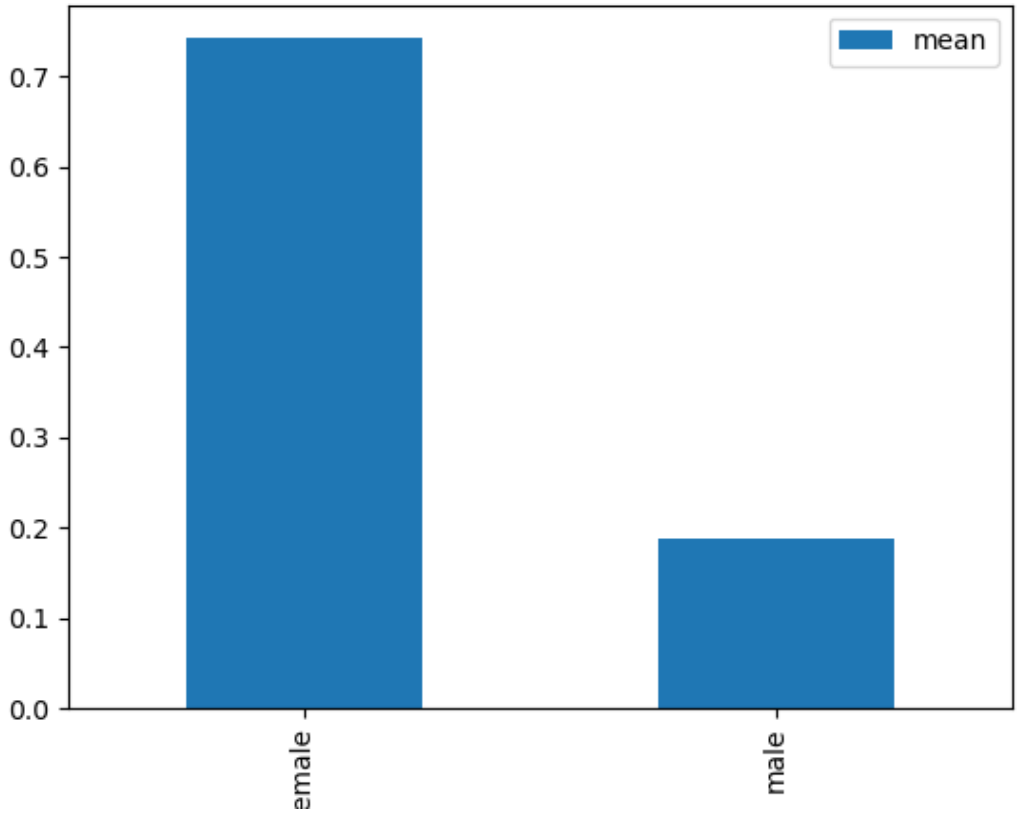
```
for column in
['Pclass','Sex','SibSp','Embarked']:

    (df1
        .groupby(column)['Survived']
        .agg(['count'])
        ).plot.bar()
    plt.show()
```

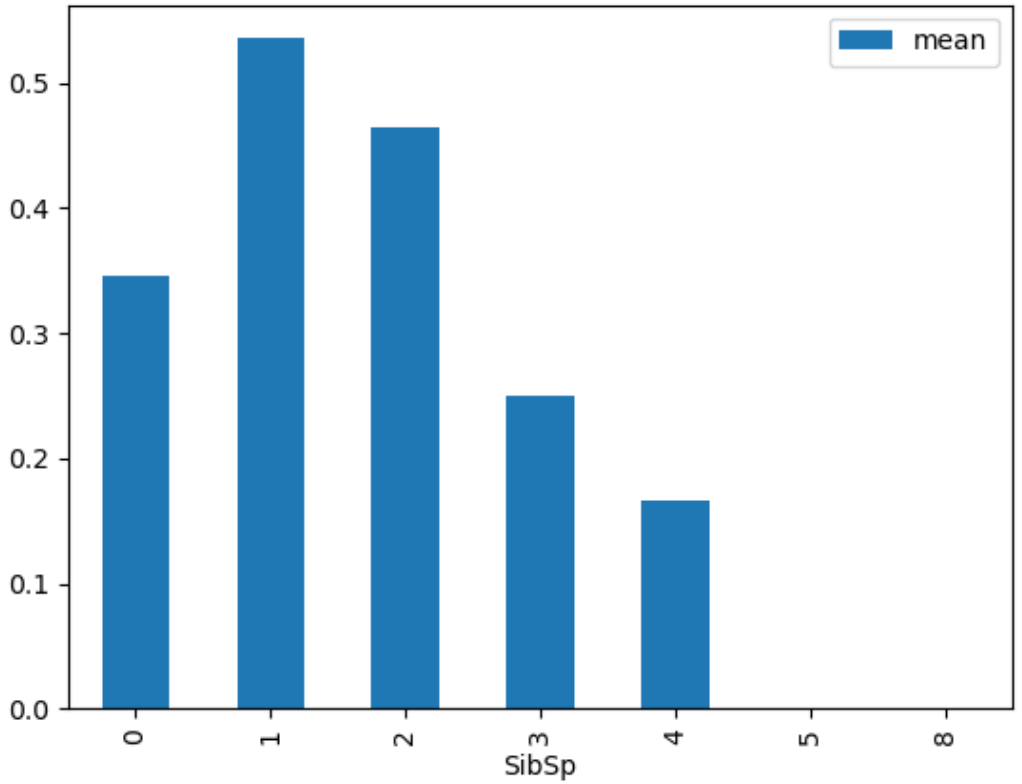
Analysis of Data: Matplotlib report



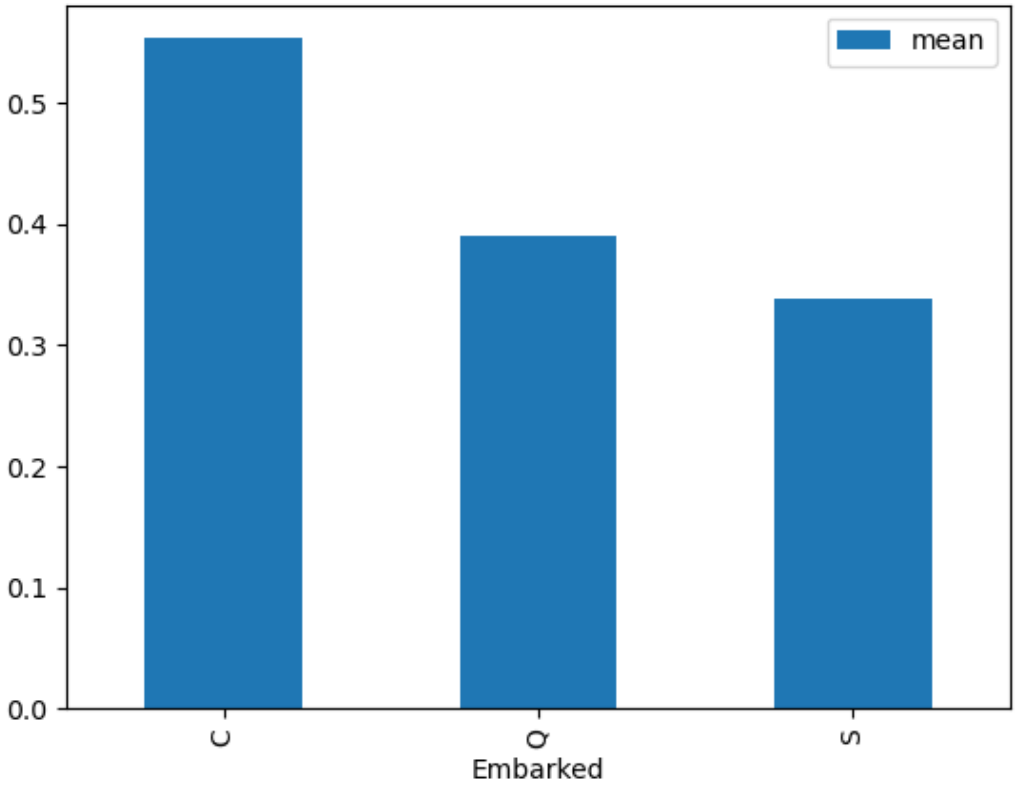
Analysis of Data: Matplotlib report



Analysis of Data: Matplotlib report



Analysis of Data: Matplotlib report



Conclusion

Persons who had better chance of survival:

- Children
- Females
- First and second classes
- Having one and two siblings

Conclusion

Limitations of the data set:

- Missing data
- Size of the sample data could also impact the results (2,435 passengers in total)
- Missing information : crew, life boat number, …