Exploratory Data Analysis with Titanic Data Set

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog

Enter your authorization code:
··········
Mounted at /content/gdrive
```

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

```
sns.set(style="whitegrid", font_scale=1.75)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the funct
  import pandas.util.testing as tm
```

```
train = pd.read_csv('/content/gdrive/My Drive/titanic/train.csv')
test = pd.read_csv('/content/gdrive/My Drive/titanic/test.csv')

print(train.shape)
print(test.shape)
```

```
(891, 12)
(418, 11)
```

```
train.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

```
test_Id = test['PassengerId']
```

```
all_features = pd.concat((train, test), sort = False)
```

```
all_features.drop('PassengerId', axis =1, inplace = True)
```

Missing Value treatment

```
missing_values = (all_features.isnull().sum()/all_features.shape[0]  *100).sort_values(ascending = False)
missing_values
```

```
Cabin       77.463713
Survived    31.932773
Age         20.091673
Embarked     0.152788
Fare         0.076394
Ticket       0.000000
Parch        0.000000
SibSp        0.000000
Sex          0.000000
Name         0.000000
```

```
Pclass       0.000000
dtype: float64
```

```
all_features['Age'].fillna(all_features['Age'].mean(), inplace = True)
```

Feature Engineering

```
def expand_embark_acronym(embarked):
    result = []
    mapping = {
            "C": "Cherbourg",
            "S": "Southampton",
            "Q": "Queenstown"
    }
    for each in embarked.values:
        if len(str(each)) > 1:
            result.append(each)
        else:
            if each in mapping:
                result.append(mapping[each])
            else:
                result.append("Unknown")
    return result

def expand_pclass_acronym(pclass):
    result = []
    mapping = {
            1: "1st class",
            2: "2nd class",
            3: "3rd class"
    }
    for each in pclass.values:
        if len(str(each)) > 1:
            result.append(each)
        else:
            if each in mapping:
                result.append(mapping[each])
            else:
                result.append("Unknown")
    return result

def is_a_minor(age):

    if age < 18:
        return "Under 18 (minor)"

    return "Adult"

# See https://help.healthycities.org/hc/en-us/articles/219556208-How-are-the-different-age-groups-defined-
def apply_age_groups(age):
    result = []
    mapping = {
            1: "Infant",      # Infants: <1
           13: "Child",       # Children: <18, <11 or K - 7th grade
           18: "Teen",        # Teens: 13-17 (Teens, who are not Adults)
           66: "Adult",       # Adults: 20+ (includes adult teens: 18+)
           123: "Elderly"     # Elderly: 65+ (123 is the oldest age known till date)
    }
    for each_age in age.values:
        if type(each_age) == str:
            result.append(category)
        else:
            category = "Unknown"
            if each_age != np.nan:
                for each_age_range in mapping:
                    if  each_age < each_age_range:
                        category = mapping[each_age_range]
                        break
            result.append(category)
    return result

def apply_age_ranges(age):
    result = []
    mapping = {
```

```
                6: "00-05 years",
               12: "06-11 years",
               19: "12-18 years",
               31: "19-30 years",
               41: "31-40 years",
               51: "41-50 years",
               61: "51-60 years",
               71: "61-70 years",
               81: "71-80 years",
               91: "81-90 years",
               124: "91+ years",  # (123 is the oldest age known till date)
        }

        for each_age in age.values:
            if type(each_age) == str:
                result.append(category)
            else:
                category = "Unknown"
                if each_age != np.nan:
                    for each_age_range in mapping:
                        if  each_age < each_age_range:
                            category = mapping[each_age_range]
                            break
                result.append(category)
        return result

    def is_married_of_single(names, ages, sexes):
        result = []
        for name, age, sex in zip(names.values, ages.values, sexes.values):
            if age < 18:
                result.append("Not of legal age")
            else:
                if ('Mrs.' in name) or ('Mme.' in name):
                    result.append("Married")
                elif ('Miss.' in name) or ('Ms.' in name) or ('Lady' in name) or ('Mlle.' in name):
                    result.append("Single")
                else:
                    result.append("Unknown")

        return result

    def apply_travel_companions(siblings_spouse, parent_children):
        result = []
        for siblings_spouse_count, parent_children_count in zip(siblings_spouse.values, parent_children.values):
            if (siblings_spouse_count > 0) and (parent_children_count > 0):
                result.append("Parent/Children & Sibling/Spouse")
            else:
                if (siblings_spouse_count > 0):
                    result.append("Sibling/Spouse")
                elif (parent_children_count > 0):
                    result.append("Parent/Children")
                else:
                    result.append("Alone")

        return result

    def apply_fare_ranges(fare):
        result = []
        mapping = {
               11: "£000 - 010",
               21: "£011 - 020",
               41: "£020 - 040",
               81: "£041 - 080",
              101: "£081 - 100",
              201: "£101 - 200",
              301: "£201 - 300",
              401: "£301 - 400",
              515: "£401 & above"  # in this case the max fare is around £512
        }
        for each_fare in fare.values:
            if type(each_fare) == str:
                result.append(category)
            else:
                category = "Unknown"
                if each_fare != np.nan:
                    for each_fare_range in mapping:
                        if  each_fare < each_fare_range:
```

```
                        category = mapping[each_fare_range]
                        break
            result.append(category)

    return result

def were_in_a_cabin_or_not(row):
    if type(row) is str:
        return "In a Cabin"
    return "Not in a Cabin"


## Embarked: Place of embarkation
all_features['Embarked'] = expand_embark_acronym(all_features['Embarked'])

# Pclass: Passenger Class
all_features['Pclass'] = expand_pclass_acronym(all_features['Pclass'])

# Age
all_features['Adult_or_minor'] = all_features['Age'].apply(is_a_minor)
females_filter = all_features['Sex'] == 'female'
adult_filter = all_features['Adult_or_minor'] == '2. Adult'

all_features['Marital_status'] = is_married_of_single(all_features['Name'], all_features['Age'], all_features['Sex'])
all_features['Age_group'] = apply_age_groups(all_features['Age'])
all_features['Age_ranges'] = apply_age_ranges(all_features['Age'])

# SibSp and Parch: Sibling/Spouse counts, Parent/Children counts
all_features['Travel_companion'] = apply_travel_companions(all_features['SibSp'], all_features['Parch'])

# Fare: ticket fare across the different classes
all_features['Fare_range'] = apply_fare_ranges(all_features['Fare'])

# Cabin: ticket holder has a cabin or not
all_features['In_Cabin'] = all_features['Cabin'].apply(were_in_a_cabin_or_not)
all_features['Cabin'] = all_features['Cabin'].fillna('No cabin')


all_features.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Adult_or_minor	Marital_status	Age_grou
0	0.0	3rd class	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	No cabin	Southampton	Adult	Unknown	Adu
1	1.0	1st class	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	Cherbourg	Adult	Married	Adu
2	1.0	3rd class	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	No cabin	Southampton	Adult	Single	Adu
3	1.0	1st class	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	Southampton	Adult	Married	Adu
4	0.0	3rd class	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	No cabin	Southampton	Adult	Unknown	Adu

```
missing_values = (all_features.isnull().sum()/all_features.shape[0]  *100).sort_values(ascending = False)
missing_values
```

```
Survived            31.932773
Embarked             0.152788
Fare                 0.076394
Age_ranges           0.000000
Age_group            0.000000
Marital_status       0.000000
Adult_or_minor       0.000000
Travel_companion     0.000000
```

```
Cabin               0.000000
Fare_range          0.000000
Ticket              0.000000
Parch               0.000000
SibSp               0.000000
Age                 0.000000
Sex                 0.000000
Name                0.000000
Pclass              0.000000
In_Cabin            0.000000
dtype: float64
```

```
def passenger_stats(dataset):
    total_ticket_holders = dataset.shape[0]
    siblings_count = dataset['SibSp'].sum()
    parents_children_count = dataset['Parch'].sum()

    print("total_ticket_holders:", total_ticket_holders)
    print("siblings_count:", siblings_count)
    print("parents_children_count:", parents_children_count)
    print("total (siblings, parents and children count):", siblings_count + parents_children_count)

    grand_total = total_ticket_holders + siblings_count + parents_children_count
    print("grand total (ticket holders, siblings, parents, children count):", grand_total)

    return grand_total

training_dataset_passengers_count = passenger_stats(all_features)
```

```
total_ticket_holders: 1309
siblings_count: 653
parents_children_count: 504
total (siblings, parents and children count): 1157
grand total (ticket holders, siblings, parents, children count): 2466
```

Creating the test & train dataset again.

```
train = all_features[: 891]
test = all_features[891:]
```

```
print(train.shape)
print(test.shape)
```

```
(891, 18)
(418, 18)
```

```
missing_values = (test.isnull().sum()/test.shape[0]  *100).sort_values(ascending = False)
missing_values
```

```
Survived            100.000000
Fare                  0.239234
Travel_companion      0.000000
Age_ranges            0.000000
Age_group             0.000000
Marital_status        0.000000
Adult_or_minor        0.000000
Embarked              0.000000
Cabin                 0.000000
Fare_range            0.000000
Ticket                0.000000
Parch                 0.000000
SibSp                 0.000000
Age                   0.000000
Sex                   0.000000
Name                  0.000000
Pclass                0.000000
In_Cabin              0.000000
dtype: float64
```

Remove the Survived Variable from the test dataset since it's empty ans the test dataset shouldn't contain the Target Variable.

```
test.drop('Survived',axis = 1, inplace = True)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3997: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

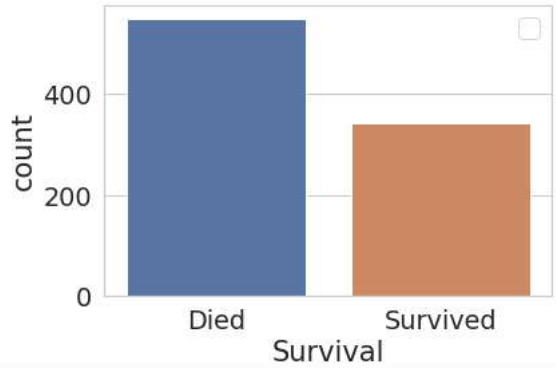
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  errors=errors,
```

Analysis

Distribution of the dataset

```
g = sns.countplot(train['Survived'])
plt.legend(loc='upper right')
g.set(xlabel="Survival", xticklabels=["Died", "Survived"]) # "0=Died", "1=Survived"
```

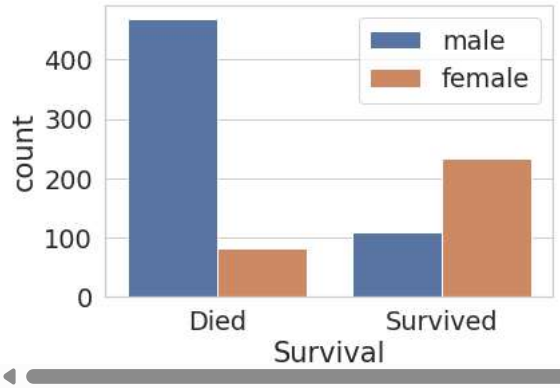
```
No handles with labels found to put in legend.
[[Text(0, 0, 'Died'), Text(0, 0, 'Survived')], Text(0.5, 0, 'Survival')]
```



Sex of the passenger associated with Survival

```
g = sns.countplot(train['Survived'], hue = train['Sex'])
plt.legend(loc='upper right')
g.set(xlabel="Survival", xticklabels=["Died", "Survived"])
```

```
[[Text(0, 0, 'Died'), Text(0, 0, 'Survived')], Text(0.5, 0, 'Survival')]
```



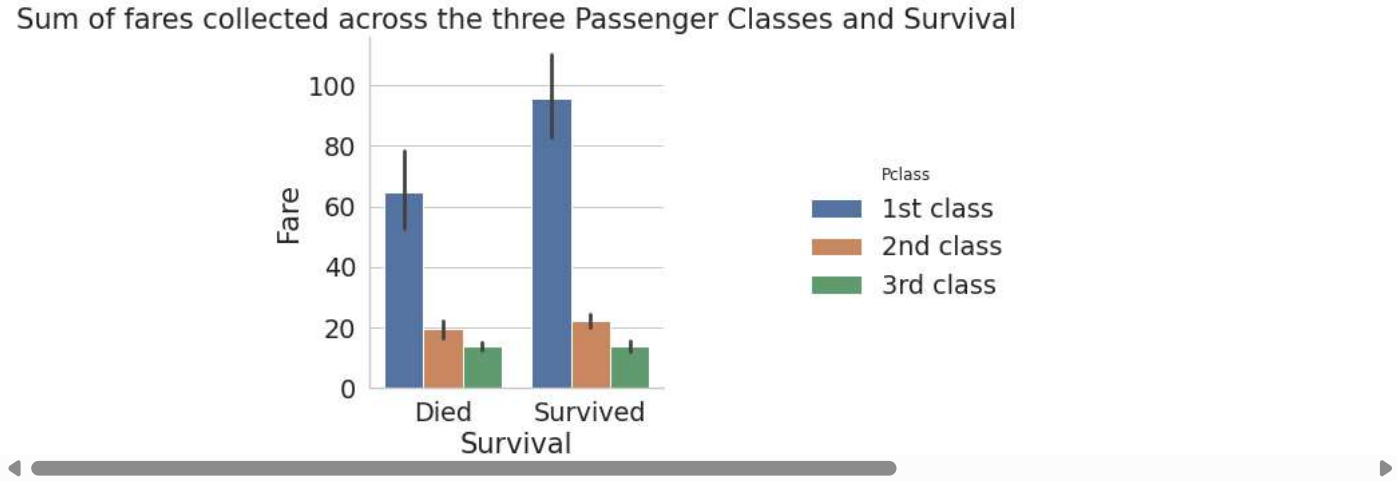
Passenger Class associated with survival

[ ] ↳ 4 cells hidden

Fares Paid associated with Survival

```
g = sns.catplot(x="Survived", y="Fare", hue="Pclass", data=train.sort_values(by='Pclass'), kind="bar");
g.set(xticklabels=['Died', 'Survived'], xlabel="Survival", title="Sum of fares collected across the three Passenger Classes and Survival")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fea55d7a2e8>
```




```
g = sns.countplot(train['Pclass'])
plt.legend(loc='upper right')
g.set(xlabel="Passenger Class")
```

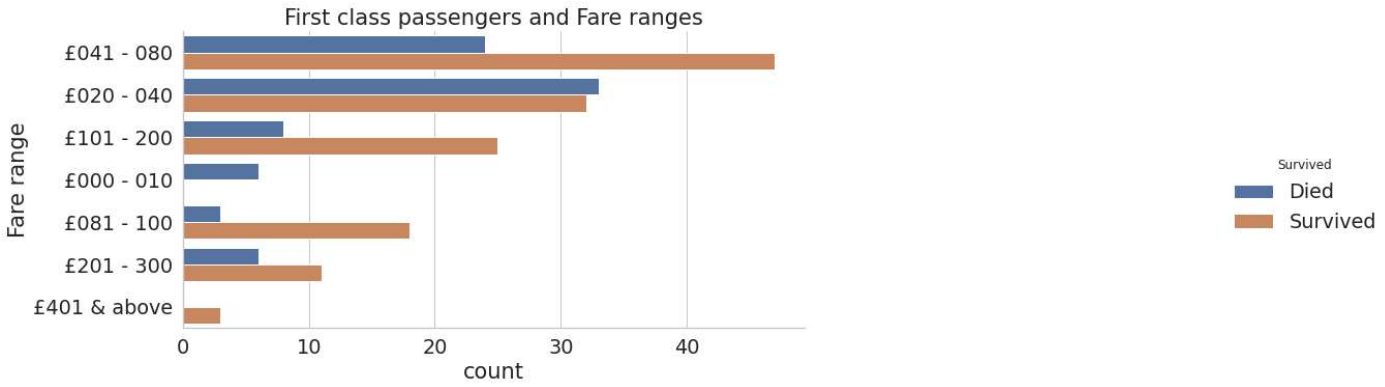
```
No handles with labels found to put in legend.
[Text(0.5, 0, 'Passenger Class')]
```

Though the 3rd Class passengers forms the largest group but it's the 1st Class passengers who survived the most. This shows Rescue services were provided to wealthy passengers.

Passenger fare range with survival

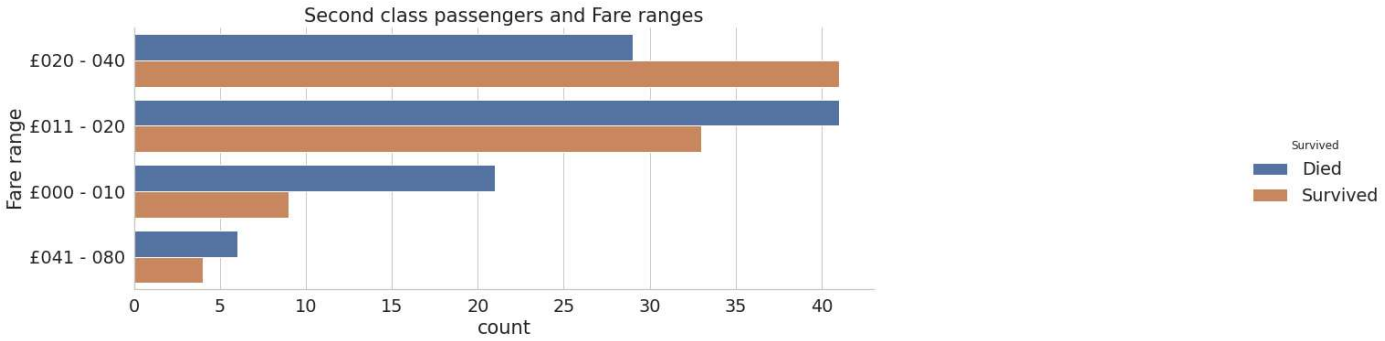
```
def fare_range_with_survival( passenger_class, title):
    dataset = train.copy()
    class_filter = dataset['Pclass'] == passenger_class
    dataset = dataset[class_filter]

    dataset[class_filter]
    g = sns.catplot(y="Fare_range", hue="Survived", data=dataset.sort_values(by='Pclass'), kind="count")
    g.set(ylabel="Fare range", title=title)

    new_labels = ['Died', 'Survived']
    for t, l in zip(g._legend.texts, new_labels):
      t.set_text(l)
    g.fig.set_figwidth(35)


fare_range_with_survival('1st class', "First class passengers and Fare ranges")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning: Boolean Series key will be reindexed to match DataFrame inc
```



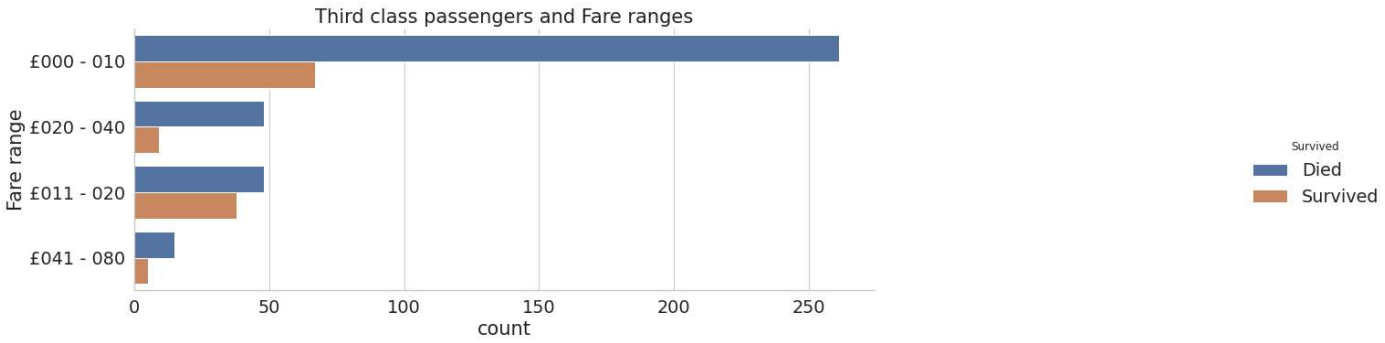

```
fare_range_with_survival('2nd class', "Second class passengers and Fare ranges")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning: Boolean Series key will be reindexed to match DataFrame inc
```



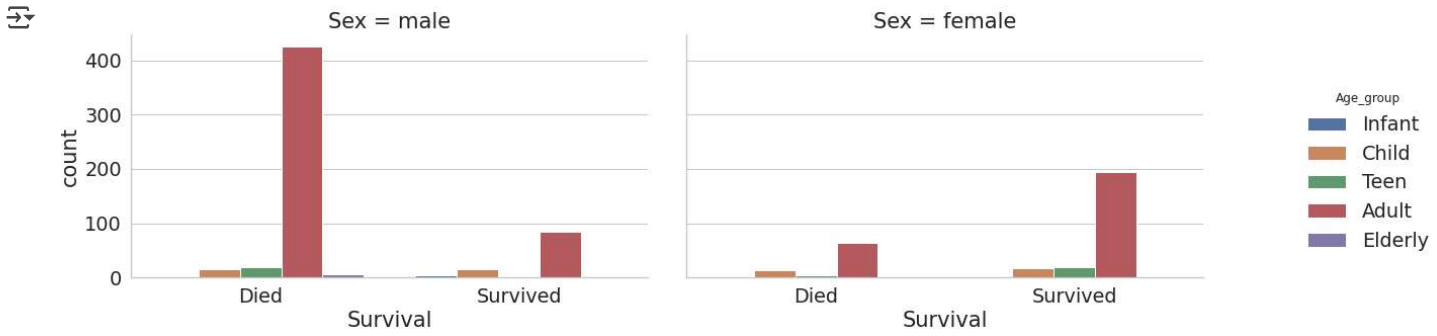

```
fare_range_with_survival('3rd class', "Third class passengers and Fare ranges")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning: Boolean Series key will be reindexed to match DataFrame inc
```

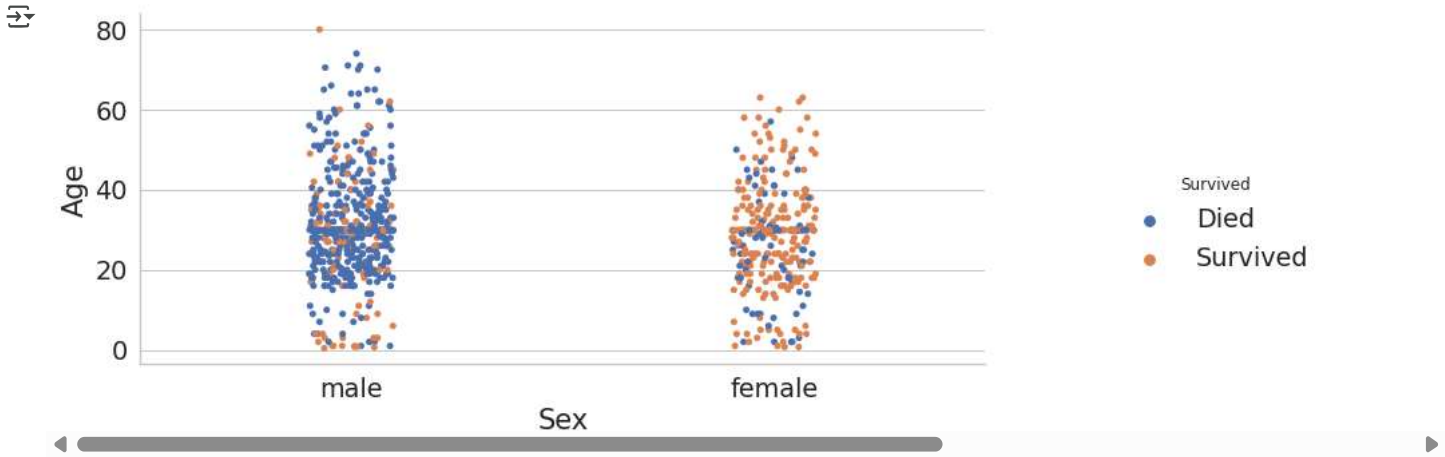


Survival with age

```
g = sns.catplot(col="Sex", x="Survived", hue="Age_group", data=train.sort_values(by='Age'), kind='count')
g.set(xlabel="Survival", xticklabels=['Died', 'Survived'])
g.fig.set_figwidth(20)
```



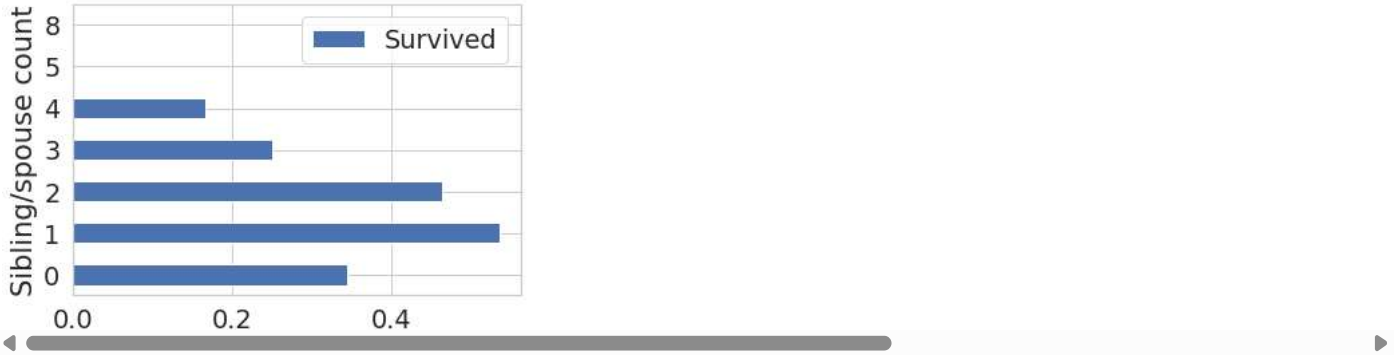

```
g = sns.catplot(y="Age", x="Sex", hue="Survived", data=train)
new_labels = ['Died', 'Survived']
for t, l in zip(g._legend.texts, new_labels):
    t.set_text(l)
g.fig.set_figwidth(16)
```



Survival with Travel Companion

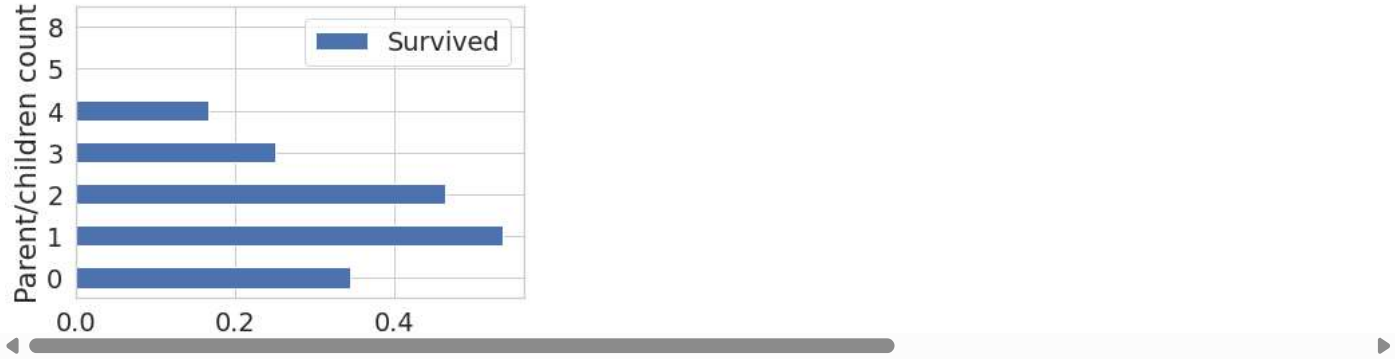
```
sibling_spouse_pivot_table = train.pivot_table(values = ['Survived'],index = 'SibSp')
sibling_spouse_pivot_table
sibling_spouse_pivot_table.plot(kind='barh')
plt.ylabel('Sibling/spouse count')
```

```
Text(0, 0.5, 'Sibling/spouse count')
```



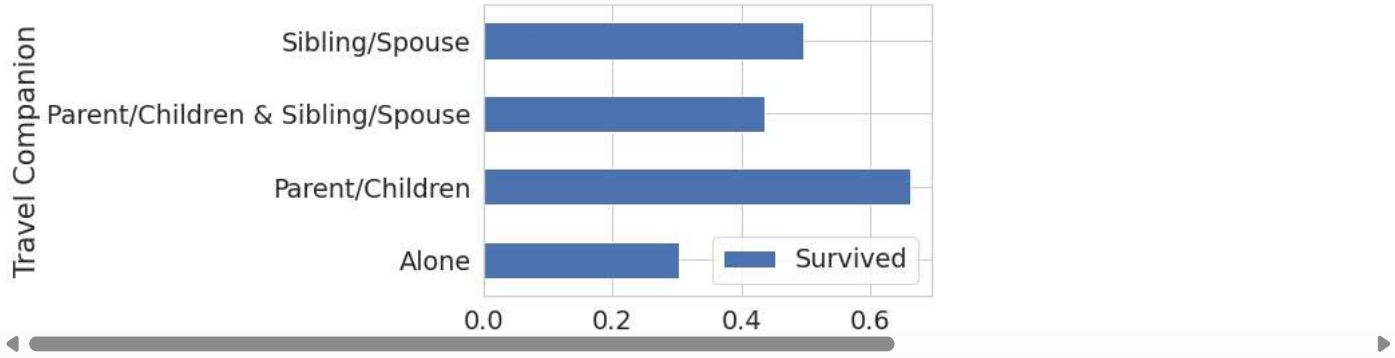

```
parent_children_pivot_table = train.pivot_table(values = ['Survived'],index = 'SibSp')
parent_children_pivot_table
sibling_spouse_pivot_table.plot(kind='barh')
plt.ylabel('Parent/children count')
```

```
Text(0, 0.5, 'Parent/children count')
```




```
travel_companion_pivot_table = train.pivot_table(values = ['Survived'],index = 'Travel_companion')
travel_companion_pivot_table
travel_companion_pivot_table.plot(kind='barh')
plt.ylabel('Travel Companion')
```

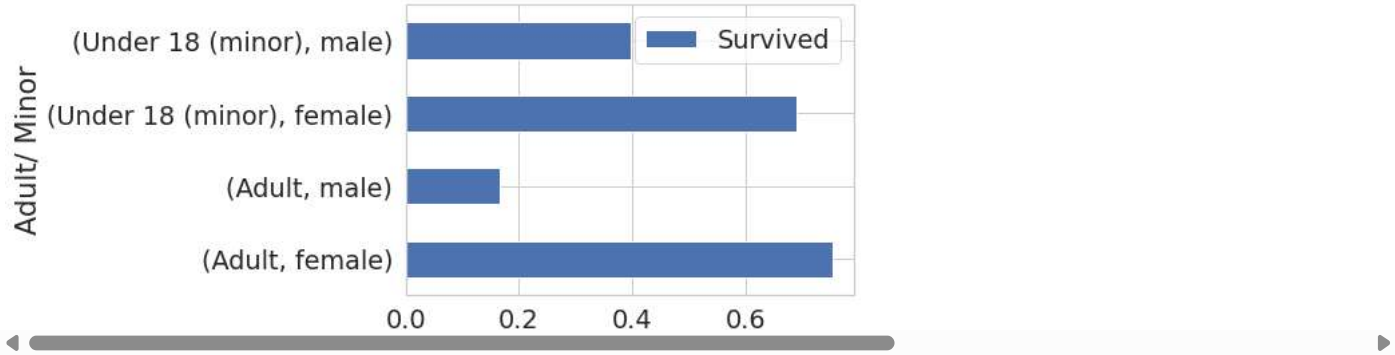
```
Text(0, 0.5, 'Travel Companion')
```



Double-click (or enter) to edit

```
adult_or_minor_pivot_table = train.pivot_table(values=['Survived'], index=['Adult_or_minor', 'Sex'])
adult_or_minor_pivot_table
adult_or_minor_pivot_table.plot(kind='barh')
plt.ylabel('Adult/ Minor')
```

```
Text(0, 0.5, 'Adult/ Minor')
```



Embarked with survival

```
embarked_pivot_table=train.pivot_table(values=['Survived'], index='Embarked')
embarked_pivot_table
```

Embarked	Survived
Cherbourg	0.553571
Queenstown	0.389610
Southampton	0.336957

It looks like, Passengers who boarded the Titanic from Southampton were the least fornunate. We need to analyze more. Let's do it with Passenger Class.

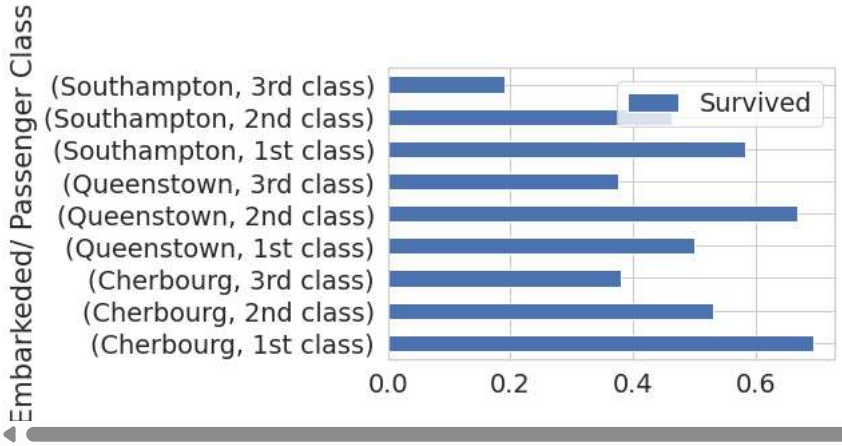
```
embarked_passenger_class_pivot_table = train.pivot_table(values=['Survived'], index=['Embarked', 'Pclass'])
embarked_passenger_class_pivot_table
```

Embarked	Pclass	Survived
Cherbourg	1st class	0.694118
	2nd class	0.529412
	3rd class	0.378788
Queenstown	1st class	0.500000
	2nd class	0.666667
	3rd class	0.375000
Southampton	1st class	0.582677
	2nd class	0.463415
	3rd class	0.189802

3rd Class passengers who boarded from Souththampon, were the least fortunate.

```
embarked_passenger_class_pivot_table.plot(kind = 'barh')
plt.ylabel('Embarkeded/ Passenger Class')
```

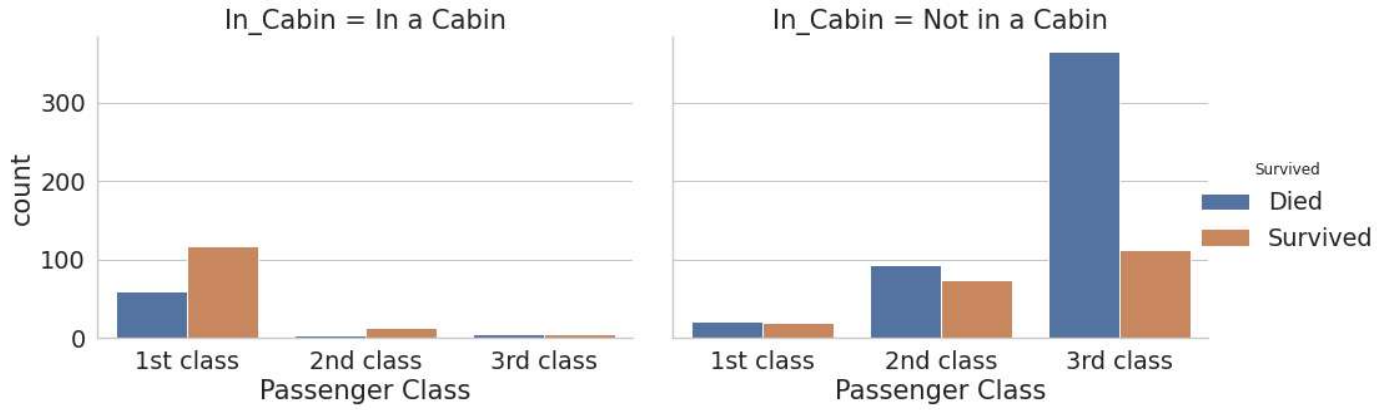
```
Text(0, 0.5, 'Embarkeded/ Passenger Class')
```



Cabin & Passenger Class with Survival

```
g = sns.catplot(col="In_Cabin", x='Pclass', hue="Survived", kind="count", data=train.sort_values(by='Pclass'));
new_labels = ['Died', 'Survived']
for t, l in zip(g._legend.texts, new_labels):
    t.set_text(l)
g.fig.set_figwidth(16)
g.set(xlabel="Passenger Class")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fea512efba8>
```



It looks like being in a cabin, somehow helped with Survival

Tickets

```
train['Ticket'].describe()
```

```
count      891
unique     681
top       1601
freq         7
Name: Ticket, dtype: object
```

There are 681 unique tickets out of 891 passengers. It seems some passengers shared a single ticket.

```
train['Ticket'].value_counts()
```

```
1601        7
```