titanic

September 23, 2023

[6]:
```
# Linear algebra
import numpy as np

# Data manipulation and analysis
import pandas as pd

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

[8]:
```
train_df = pd.read_csv(r'C:\Users\HP\Downloads\train.csv')
test_df = pd.read_csv(r'C:\Users\HP\Downloads\test.csv')

train_df['train_test'] = 1
test_df['train_test'] = 0
# test_df['Survived'] = np.NaN
all_data = pd.concat([train_df,test_df])

%matplotlib inline
all_data.columns
```

[8]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'train_test'],
      dtype='object')
```

[9]:
```
train_df.head(10)
```

[9]:

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

[10]:
```
test_df.head(10)
```

[10]:

	PassengerId	Pclass	Name	Sex
0	892	3	Kelly, Mr. James	male
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female
2	894	2	Myles, Mr. Thomas Francis	male
3	895	3	Wirz, Mr. Albert	male
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female
5	897	3	Svensson, Mr. Johan Cervin	male

```
6          898       3                          Connolly, Miss. Kate  female
7          899       2                 Caldwell, Mr. Albert Francis    male
8          900       3    Abrahim, Mrs. Joseph (Sophie Halaut Easu)  female
9          901       3                        Davies, Mr. John Samuel    male

    Age  SibSp  Parch     Ticket     Fare Cabin Embarked  train_test
0  34.5      0      0     330911   7.8292   NaN        Q           0
1  47.0      1      0     363272   7.0000   NaN        S           0
2  62.0      0      0     240276   9.6875   NaN        Q           0
3  27.0      0      0     315154   8.6625   NaN        S           0
4  22.0      1      1    3101298  12.2875   NaN        S           0
5  14.0      0      0       7538   9.2250   NaN        S           0
6  30.0      0      0     330972   7.6292   NaN        Q           0
7  26.0      1      1     248738  29.0000   NaN        S           0
8  18.0      0      0       2657   7.2292   NaN        C           0
9  21.0      2      0  A/4 48871  24.1500   NaN        S           0
```

[11]:
```
train_df.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
```

[12]:
```
train_df.describe()
```

[12]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
```

```
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

```
            Parch        Fare  train_test
count  891.000000  891.000000       891.0
mean     0.381594   32.204208         1.0
std      0.806057   49.693429         0.0
min      0.000000    0.000000         1.0
25%      0.000000    7.910400         1.0
50%      0.000000   14.454200         1.0
75%      0.000000   31.000000         1.0
max      6.000000  512.329200         1.0
```

[13]:
```
total = train_df.isnull().sum().sort_values(ascending=False)
percent_1 = train_df.isnull().sum()/train_df.isnull().count()*100
percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
missing_data.head(13)
```

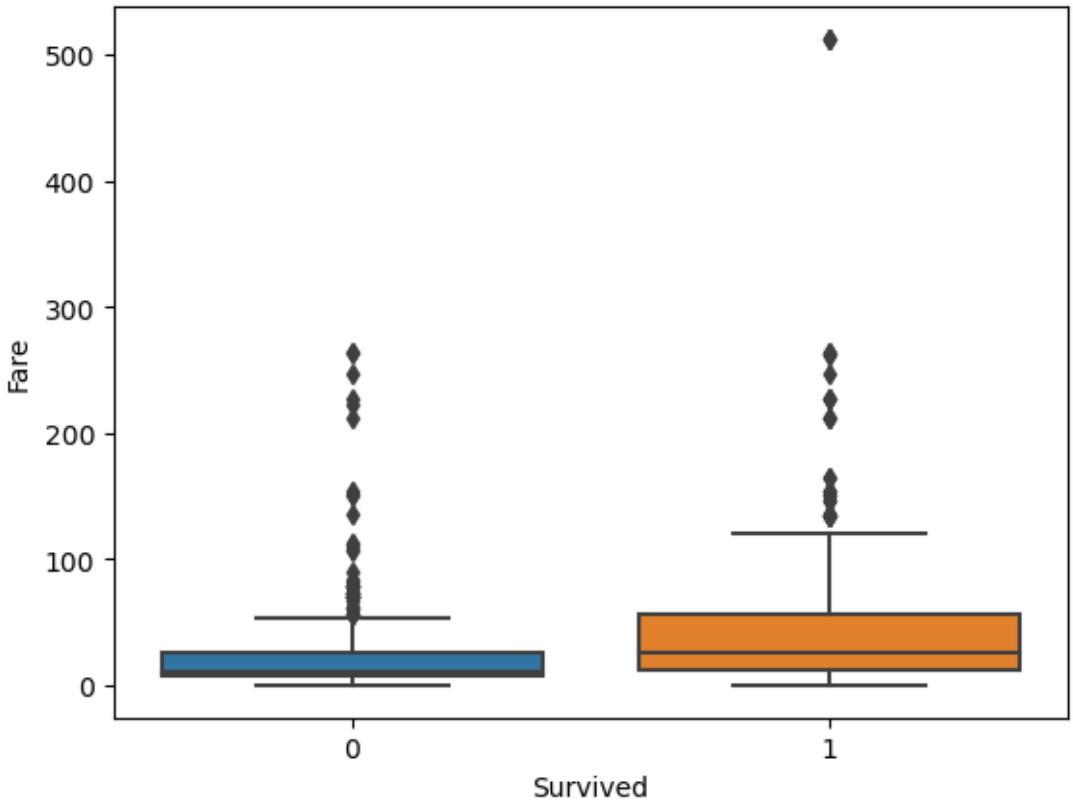
[13]:
```
             Total     %
Cabin          687  77.1
Age            177  19.9
Embarked         2   0.2
PassengerId      0   0.0
Survived         0   0.0
Pclass           0   0.0
Name             0   0.0
Sex              0   0.0
SibSp            0   0.0
Parch            0   0.0
Ticket           0   0.0
Fare             0   0.0
train_test       0   0.0
```

[14]:
```
train_df.columns.values
```

[14]:
```
array(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'train_test'],
      dtype=object)
```

[15]:
```
sns.boxplot(x="Survived",y="Fare",data=train_df)
```

[15]:
```
<AxesSubplot:xlabel='Survived', ylabel='Fare'>
```



[16]:
```
train_df[train_df['Fare']>300]
```

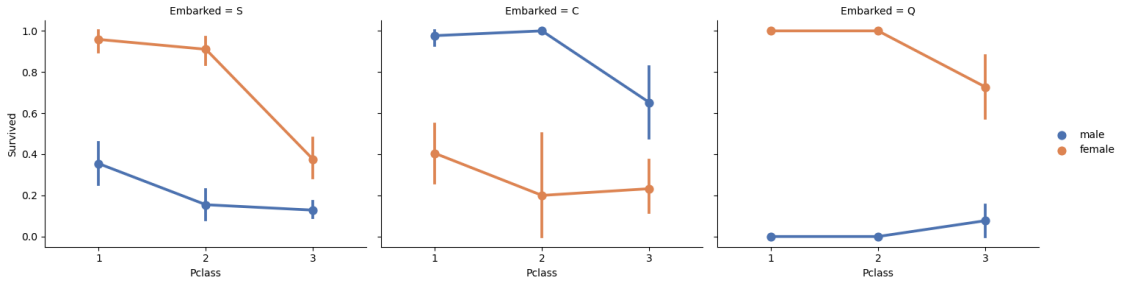
[16]:
```
     PassengerId  Survived  Pclass                                Name  \
258          259         1       1                   Ward, Miss. Anna
679          680         1       1  Cardeza, Mr. Thomas Drake Martinez
737          738         1       1              Lesurer, Mr. Gustave J

        Sex   Age  SibSp  Parch    Ticket      Fare        Cabin Embarked  \
258  female  35.0      0      0  PC 17755  512.3292          NaN        C
679    male  36.0      0      1  PC 17755  512.3292  B51 B53 B55        C
737    male  35.0      0      0  PC 17755  512.3292         B101        C

     train_test
258           1
679           1
737           1
```

[17]:
```
train_df[train_df['Name'].str.contains("Capt")]
```

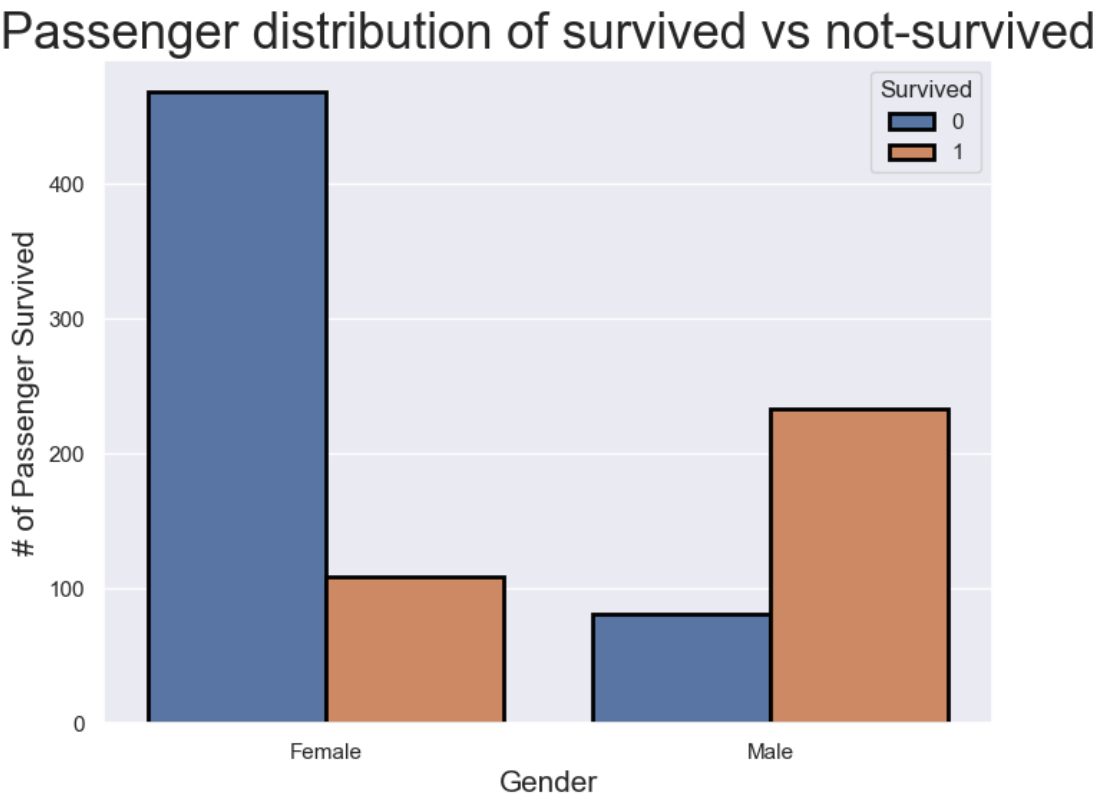
[17]:
```
     PassengerId  Survived  Pclass                          Name   Sex   Age  \
745          746         0       1  Crosby, Capt. Edward Gifford  male  70.0
```

```
     SibSp  Parch     Ticket  Fare Cabin Embarked  train_test
745      1      1  WE/P 5735  71.0   B22        S           1
```

[18]:
```
FacetGrid = sns.FacetGrid(train_df, col='Embarked', height=4, aspect=1.2)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', ci=95.0,␣
 ↪palette='deep', order=None, hue_order=None)
FacetGrid.add_legend();
```



[19]:
```
sns.set(style='darkgrid')
plt.subplots(figsize = (8,6))
ax=sns.countplot(x='Sex', data = train_df, hue='Survived', edgecolor=(0,0,0),␣
 ↪linewidth=2)

# Fixing title, xlabel and ylabel
plt.title('Passenger distribution of survived vs not-survived', fontsize=25)
plt.xlabel('Gender', fontsize=15)
plt.ylabel("# of Passenger Survived", fontsize = 15)
labels = ['Female', 'Male']

# Fixing xticks.
plt.xticks(sorted(train_df.Survived.unique()),labels);
```



[20]:
```
train_df.groupby(['Sex']).mean()
```

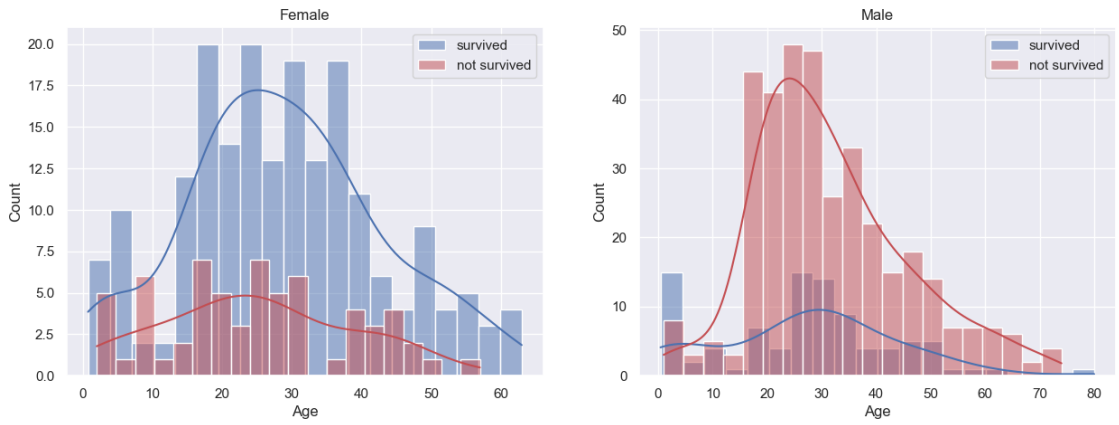
[20]:
```
        PassengerId  Survived    Pclass        Age     SibSp     Parch  \
Sex
female   431.028662  0.742038  2.159236  27.915709  0.694268  0.649682
male     454.147314  0.188908  2.389948  30.726645  0.429809  0.235702

              Fare  train_test
Sex
female  44.479818         1.0
male    25.523893         1.0
```

[21]:
```
train_df.groupby(['Sex','Pclass']).mean()
```

[21]:
```
                PassengerId  Survived        Age     SibSp     Parch  \
Sex    Pclass
female 1         469.212766  0.968085  34.611765  0.553191  0.457447
       2         443.105263  0.921053  28.722973  0.486842  0.605263
       3         399.729167  0.500000  21.750000  0.895833  0.798611
male   1         455.729508  0.368852  41.281386  0.311475  0.278689
       2         447.962963  0.157407  30.740707  0.342593  0.222222
```

```
       3        455.515850  0.135447  26.507589  0.498559  0.224784

                      Fare  train_test
Sex    Pclass
female 1       106.125798         1.0
       2        21.970121         1.0
       3        16.118810         1.0
male   1        67.226127         1.0
       2        19.741782         1.0
       3        12.661633         1.0
```

[22]:
```
survived = 'survived'
not_survived = 'not survived'

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

women = train_df[train_df['Sex']=='female']
men = train_df[train_df['Sex']=='male']

# Plot Female Survived vs Not-Survived distribution
ax = sns.histplot(women[women['Survived']==1].Age.dropna(), bins=20, label =␣
 ↪survived, ax = axes[0],color='b', kde=True)
ax = sns.histplot(women[women['Survived']==0].Age.dropna(), bins=20, label =␣
 ↪not_survived, ax = axes[0],color='r', kde=True)
ax.legend()
ax.set_title('Female')

# Plot Male Survived vs Not-Survived distribution
ax = sns.histplot(men[men['Survived']==1].Age.dropna(), bins=20, label =␣
 ↪survived, ax = axes[1],color='b', kde=True)
ax = sns.histplot(men[men['Survived']==0].Age.dropna(), bins=20, label =␣
 ↪not_survived, ax = axes[1],color='r', kde=True)
ax.legend()
ax.set_title('Male');
```

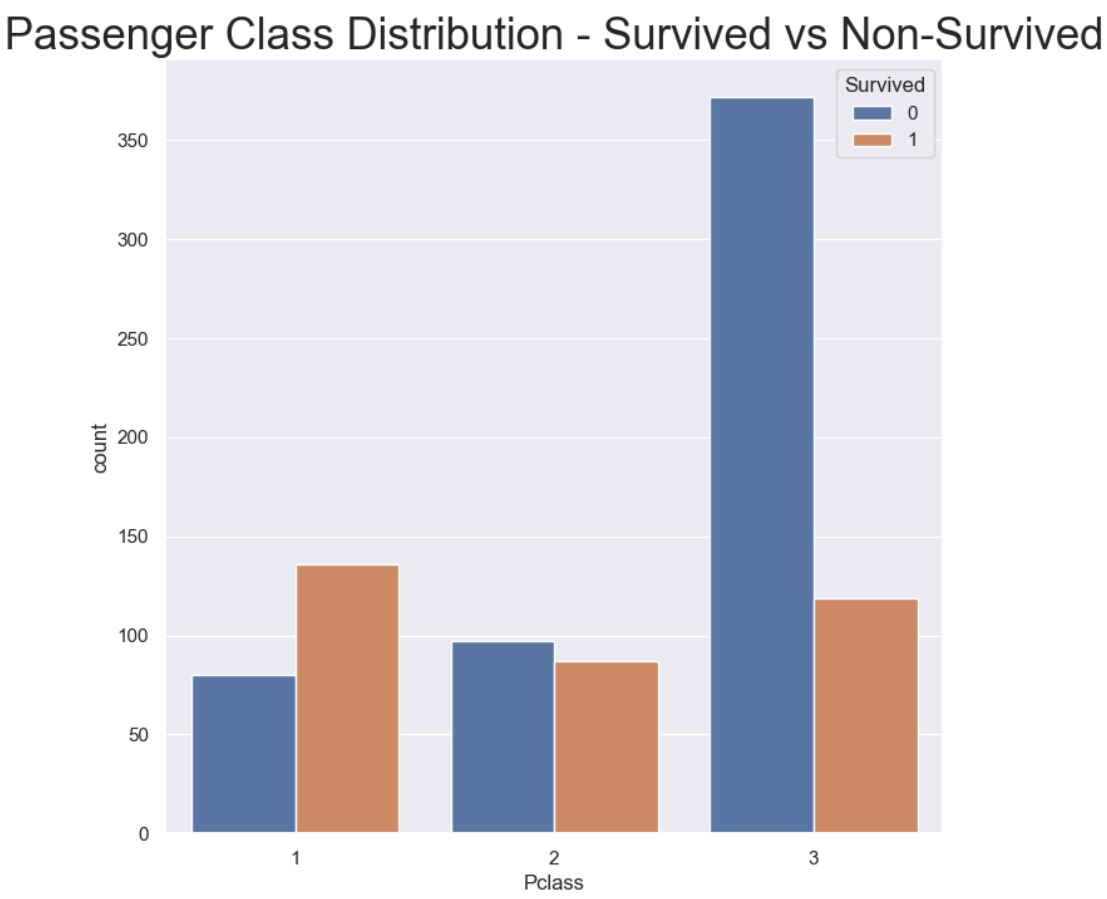


[23]:
```
train_df[train_df['Age']<18].groupby(['Sex','Pclass']).mean()
```

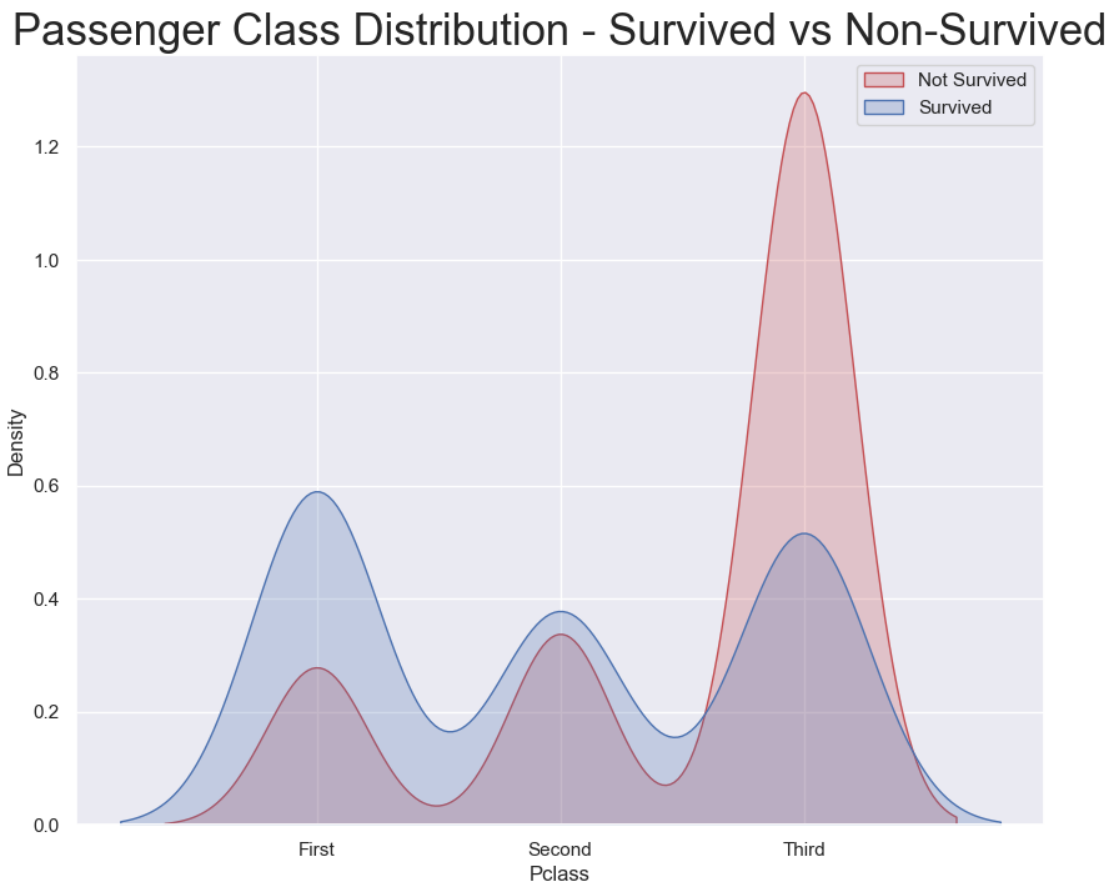
[23]:

Sex	Pclass	PassengerId	Survived	Age	SibSp	Parch	Fare	train_test
female	1	525.375000	0.875000	14.125000	0.500000	0.875000	104.083337	1.0
	2	369.250000	1.000000	8.333333	0.583333	1.083333	26.241667	1.0
	3	374.942857	0.542857	8.428571	1.571429	1.057143	18.727977	1.0
male	1	526.500000	1.000000	8.230000	0.500000	2.000000	116.072900	1.0
	2	527.818182	0.818182	4.757273	0.727273	1.000000	25.659473	1.0
	3	437.953488	0.232558	9.963256	2.069767	1.000000	22.752523	1.0

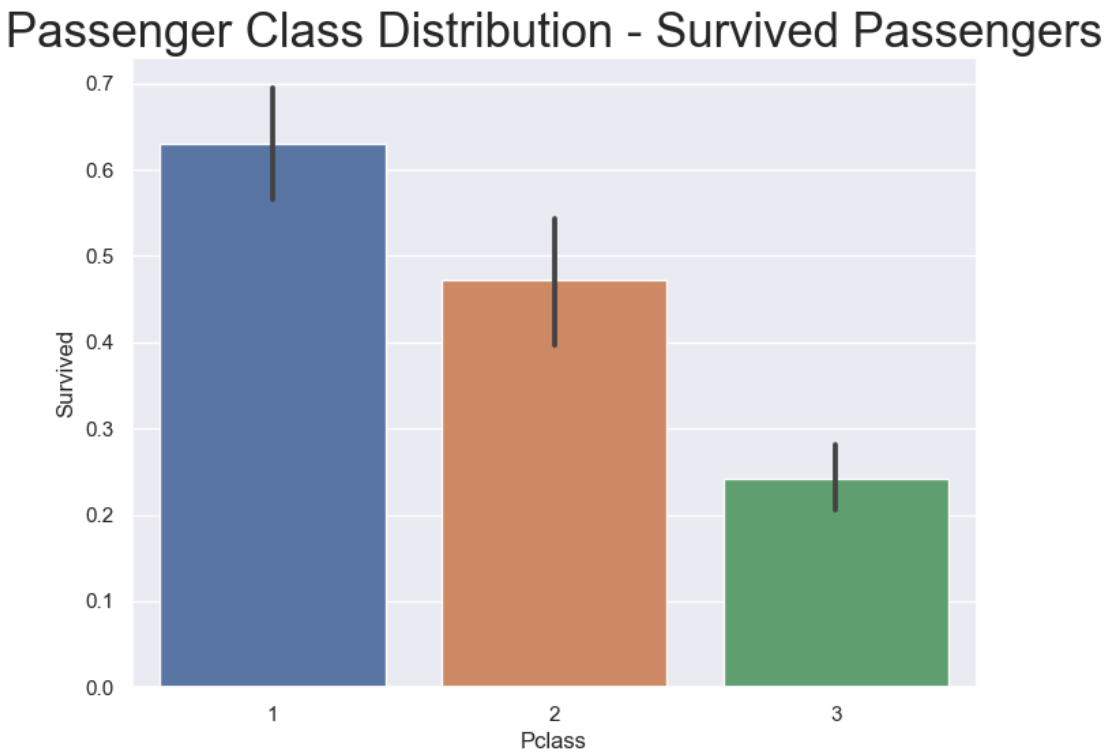
[24]:
```
plt.subplots(figsize = (8,8))
ax=sns.countplot(x='Pclass',hue='Survived',data=train_df)
plt.title("Passenger Class Distribution - Survived vs Non-Survived", fontsize =␣
↪25);
```



[25]:
```
plt.subplots(figsize=(10,8))
ax=sns.kdeplot(train_df.loc[(train_df['Survived'] ==␣
 ↪0),'Pclass'],shade=True,color='r',label='Not Survived')
ax.legend()
ax=sns.kdeplot(train_df.loc[(train_df['Survived'] ==␣
 ↪1),'Pclass'],shade=True,color='b',label='Survived')
ax.legend()

plt.title("Passenger Class Distribution - Survived vs Non-Survived", fontsize =␣
 ↪25)
labels = ['First', 'Second', 'Third']
plt.xticks(sorted(train_df.Pclass.unique()),labels);
```

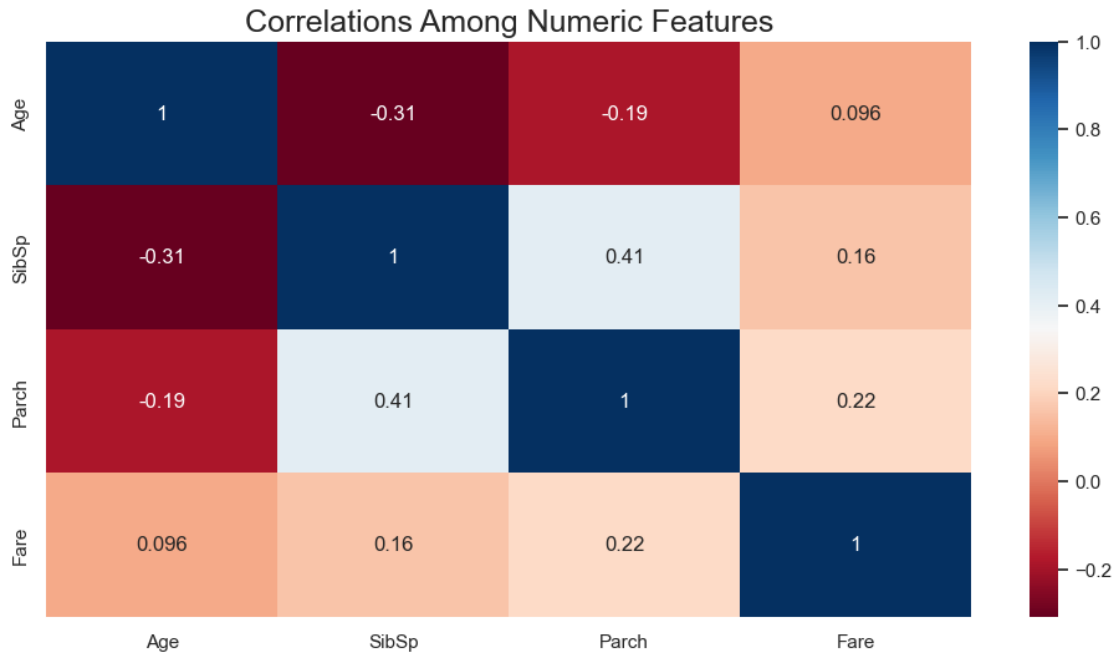


[26]:
```
plt.subplots(figsize = (8,6))
sns.barplot(x='Pclass', y='Survived', data=train_df);
plt.title("Passenger Class Distribution - Survived Passengers", fontsize = 25);
```



[27]:
```
# Look at numeric and categorical values separately
df_num = train_df[['Age','SibSp','Parch','Fare']]
df_cat = train_df[['Survived','Pclass','Sex','Ticket','Cabin','Embarked']]
```

[28]:
```
plt.subplots(figsize = (12,6))
sns.heatmap(df_num.corr(), annot=True,cmap="RdBu")
plt.title("Correlations Among Numeric Features", fontsize = 18);
```



[29]:
```
train_df = train_df.drop(['PassengerId'], axis=1)
train_df.head()
```

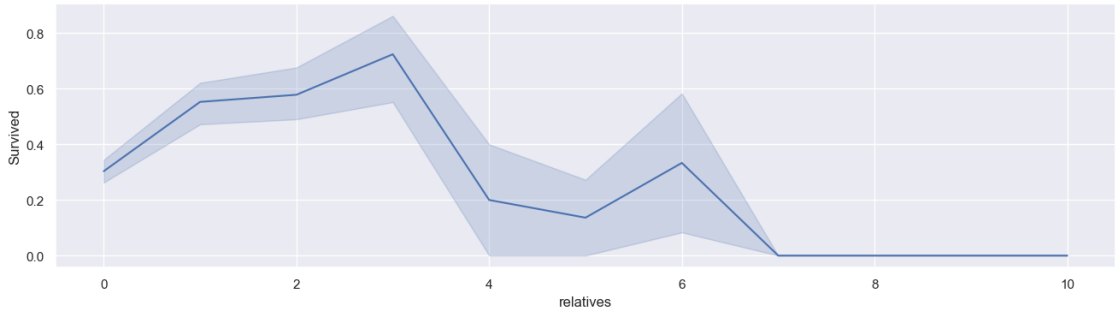
[29]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	train_test
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	1
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C	1
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	1
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	1

[30]:
```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] = dataset['not_alone'].astype(int)
train_df['not_alone'].value_counts()
```

[30]:
```
1    537
0    354
Name: not_alone, dtype: int64
```

[31]:
```
plt.subplots(figsize = (16,4))
ax = sns.lineplot(x='relatives',y='Survived', data=train_df)
```



[32]:
```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").
 ↪search(x).group())
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
```

[33]:
```
# We can now drop the Cabin feature
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)
```

[34]:
```
data = [train_df, test_df]

for dataset in data:
    mean = train_df["Age"].mean()
```

```
    std = test_df["Age"].std()
    is_null = dataset["Age"].isnull().sum()

    # Compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)

    # Fill NaN values in Age column with random values generated
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = train_df["Age"].astype(int)
```

[35]:
```
train_df["Age"].isnull().sum()
```

[35]:
```
0
```

[36]:
```
train_df['Embarked'].describe()
```

[36]:
```
count     889
unique      3
top         S
freq      644
Name: Embarked, dtype: object
```

[37]:
```
common_value = 'S'
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)
```

[38]:
```
train_df['Embarked'].isnull().sum()
```

[38]:
```
0
```

[39]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Name        891 non-null    object
 3   Sex         891 non-null    object
 4   Age         891 non-null    int32
 5   SibSp       891 non-null    int64
```

```
 6   Parch       891 non-null    int64
 7   Ticket      891 non-null    object
 8   Fare        891 non-null    float64
 9   Embarked    891 non-null    object
 10  train_test  891 non-null    int64
 11  relatives   891 non-null    int64
 12  not_alone   891 non-null    int32
 13  Deck        891 non-null    int32
dtypes: float64(1), int32(3), int64(6), object(4)
memory usage: 87.1+ KB
```

[40]:
```
data = [train_df, test_df]

for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)
```

[41]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Name        891 non-null    object
 3   Sex         891 non-null    object
 4   Age         891 non-null    int32
 5   SibSp       891 non-null    int64
 6   Parch       891 non-null    int64
 7   Ticket      891 non-null    object
 8   Fare        891 non-null    int32
 9   Embarked    891 non-null    object
 10  train_test  891 non-null    int64
 11  relatives   891 non-null    int64
 12  not_alone   891 non-null    int32
 13  Deck        891 non-null    int32
dtypes: int32(4), int64(6), object(4)
memory usage: 83.7+ KB
```

[42]:
```
data = [train_df, test_df]
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Other": 5}

for dataset in data:
    # Extract titles
    dataset['Title'] = dataset.Name.str.extract('([A-Za-z]+)\.', expand=False)
```

```
    # Replace titles with a more common title or as Other
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt',␣
↪'Col','Don', 'Dr','Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Other')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

    # Convert titles into numbers
    dataset['Title'] = dataset['Title'].map(titles)

    # Filling NaN with 0 just to be safe
    dataset['Title'] = dataset['Title'].fillna(0)
```

[43]:
```
train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
```

[44]:
```
# Checking results
train_df.head()
```

[44]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	train_test	relatives	not_alone	Deck	Title
0	0	3	male	22	1	0	A/5 21171	7	S	1	1	0	8	1
1	1	1	female	38	1	0	PC 17599	71	C	1	1	0	3	3
2	1	3	female	26	0	0	STON/O2. 3101282	7	S	1	0	1	8	2
3	1	1	female	35	1	0	113803	53	S	1	1	0	3	3
4	0	3	male	35	0	0	373450	8	S	1	0	1	8	1

[45]:
```
genders = {"male": 0, "female": 1}
data = [train_df, test_df]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

[46]:
```
train_df.head()
```

[46]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	train_test	relatives	not_alone	Deck	Title
0	0	3	0	22	1	0	A/5 21171	7	S	1	1	0	8	1
1	1	1	1	38	1	0	PC 17599	71	C	1	1	0	3	3
2	1	3	1	26	0	0	STON/O2. 3101282	7	S	1	0	1	8	2
3	1	1	1	35	1	0	113803	53	S	1	1	0	3	3

```
4         0       3    0   35      0      0           373450     8        S

   train_test  relatives  not_alone  Deck  Title
0           1          1          0     8      1
1           1          1          0     3      3
2           1          0          1     8      2
3           1          1          0     3      3
4           1          0          1     8      1
```

[47]:
```
train_df['Ticket'].describe()
```

[47]:
```
count        891
unique       681
top       347082
freq           7
Name: Ticket, dtype: object
```

[48]:
```
train_df = train_df.drop(['Ticket'], axis=1)
test_df = test_df.drop(['Ticket'], axis=1)
```

[49]:
```
train_df.head()
```

[49]:
```
   Survived  Pclass  Sex  Age  SibSp  Parch  Fare Embarked  train_test  \
0         0       3    0   22      1      0     7        S           1
1         1       1    1   38      1      0    71        C           1
2         1       3    1   26      0      0     7        S           1
3         1       1    1   35      1      0    53        S           1
4         0       3    0   35      0      0     8        S           1

   relatives  not_alone  Deck  Title
0          1          0     8      1
1          1          0     3      3
2          0          1     8      2
3          1          0     3      3
4          0          1     8      1
```

[50]:
```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

[51]:
```
train_df.head()
```

[51]:
```
   Survived  Pclass  Sex  Age  SibSp  Parch  Fare  Embarked  train_test  \
0         0       3    0   22      1      0     7         0           1
1         1       1    1   38      1      0    71         1           1
```

```
2         1       3    1   26      0      0     7         0           1
3         1       1    1   35      1      0    53         0           1
4         0       3    0   35      0      0     8         0           1

   relatives  not_alone  Deck  Title
0          1          0     8      1
1          1          0     3      3
2          0          1     8      2
3          1          0     3      3
4          0          1     8      1
```

[52]:
```
data = [train_df, test_df]
for dataset in data:
    dataset['Age'] = dataset['Age'].astype(int)
    dataset.loc[ dataset['Age'] <= 11, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 11) & (dataset['Age'] <= 18), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 18) & (dataset['Age'] <= 22), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 22) & (dataset['Age'] <= 27), 'Age'] = 3
    dataset.loc[(dataset['Age'] > 27) & (dataset['Age'] <= 33), 'Age'] = 4
    dataset.loc[(dataset['Age'] > 33) & (dataset['Age'] <= 40), 'Age'] = 5
    dataset.loc[(dataset['Age'] > 40) & (dataset['Age'] <= 66), 'Age'] = 6
    dataset.loc[ dataset['Age'] > 66, 'Age'] = 6
```

[53]:
```
# Checking the distribution
train_df['Age'].value_counts()
```

[53]:
```
4    162
6    161
5    153
3    136
2    116
1     95
0     68
Name: Age, dtype: int64
```

[54]:
```
train_df.head()
```

[54]:
```
   Survived  Pclass  Sex  Age  SibSp  Parch  Fare  Embarked  train_test  \
0         0       3    0    2      1      0     7         0           1
1         1       1    1    5      1      0    71         1           1
2         1       3    1    3      0      0     7         0           1
3         1       1    1    5      1      0    53         0           1
4         0       3    0    5      0      0     8         0           1

   relatives  not_alone  Deck  Title
0          1          0     8      1
1          1          0     3      3
```

```
2           0           1     8      2
3           1           0     3      3
4           0           1     8      1
```

[55]:
```
pd.qcut(train_df['Fare'], q=6)
```

[55]:
```
0      (-0.001, 7.0]
1      (52.0, 512.0]
2      (-0.001, 7.0]
3      (52.0, 512.0]
4         (7.0, 8.0]
             ...
886      (8.0, 14.0]
887     (26.0, 52.0]
888     (14.0, 26.0]
889     (26.0, 52.0]
890    (-0.001, 7.0]
Name: Fare, Length: 891, dtype: category
Categories (6, interval[float64, right]): [(-0.001, 7.0] < (7.0, 8.0] < (8.0,
14.0] < (14.0, 26.0] < (26.0, 52.0] < (52.0, 512.0]]
```

[56]:
```
#Using the values from pd.qcut() to create bins for Fare
data = [train_df, test_df]

for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7) & (dataset['Fare'] <= 8), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 8) & (dataset['Fare'] <= 14), 'Fare']   = 2
    dataset.loc[(dataset['Fare'] > 14) & (dataset['Fare'] <= 26), 'Fare']   = 3
    dataset.loc[(dataset['Fare'] > 26) & (dataset['Fare'] <= 52), 'Fare']   = 4
    dataset.loc[dataset['Fare'] > 52, 'Fare'] = 5
    dataset['Fare'] = dataset['Fare'].astype(int)
```

[57]:
```
# Checking the dataset
train_df.head(10)
```

[57]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	train_test	\
0	0	3	0	2	1	0	0	0	1	
1	1	1	1	5	1	0	5	1	1	
2	1	3	1	3	0	0	0	0	1	
3	1	1	1	5	1	0	5	0	1	
4	0	3	0	5	0	0	1	0	1	
5	0	3	0	5	0	0	1	2	1	
6	0	1	0	6	0	0	4	0	1	
7	0	3	0	0	3	1	3	0	1	
8	1	3	1	3	0	2	2	0	1	
9	1	2	1	1	1	0	4	1	1	

	relatives	not_alone	Deck	Title
0	1	0	8	1
1	1	0	3	3
2	0	1	8	2
3	1	0	3	3
4	0	1	8	1
5	0	1	8	1
6	0	1	5	1
7	4	0	8	4
8	2	0	8	3
9	1	0	8	3

[58]:
```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
```

[59]:
```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)

Y_pred = sgd.predict(X_test)

sgd.score(X_train, Y_train)
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_sgd,2,), "%")
```

```
82.15 %
```

[60]:
```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_random_forest,2,), "%")
```

```
93.15 %
```

[61]:
```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)
```

```
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_log,2,), "%")
```

```
81.82 %
```

[62]:
```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)

Y_pred = knn.predict(X_test)

acc_knn = round(knn.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_knn,2,), "%")
```

```
85.63 %

C:\Users\HP\anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\HP\anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[63]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)

Y_pred = gaussian.predict(X_test)

acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_gaussian,2,), "%")
```

```
78.45 %
```

[64]:
```
perceptron = Perceptron(max_iter=1000)
perceptron.fit(X_train, Y_train)

Y_pred = perceptron.predict(X_test)

acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_perceptron,2,), "%")
```

```
69.14 %
```

[65]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

Y_pred = decision_tree.predict(X_test)

acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)

# Print score
print(round(acc_decision_tree,2,), "%")
```

```
93.15 %
```

[66]:
```
Which one is the best model?
results = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree]})

result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

```
Object `model` not found.
```

[66]:

Score	Model
93.15	Random Forest
93.15	Decision Tree
85.63	KNN
82.15	Stochastic Gradient Decent
81.82	Logistic Regression
78.45	Naive Bayes

69.14 Perceptron

[67]:
```
from sklearn.model_selection import cross_val_score

rf = RandomForestClassifier(n_estimators=100)
scores = cross_val_score(rf, X_train, Y_train, cv=10, scoring = "accuracy")
```

[68]:
```
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

```
Scores: [0.8        0.85393258 0.76404494 0.83146067 0.86516854 0.84269663
 0.7752809  0.7752809  0.86516854 0.84269663]
Mean: 0.821573033707865
Standard Deviation: 0.037281704822275025
```

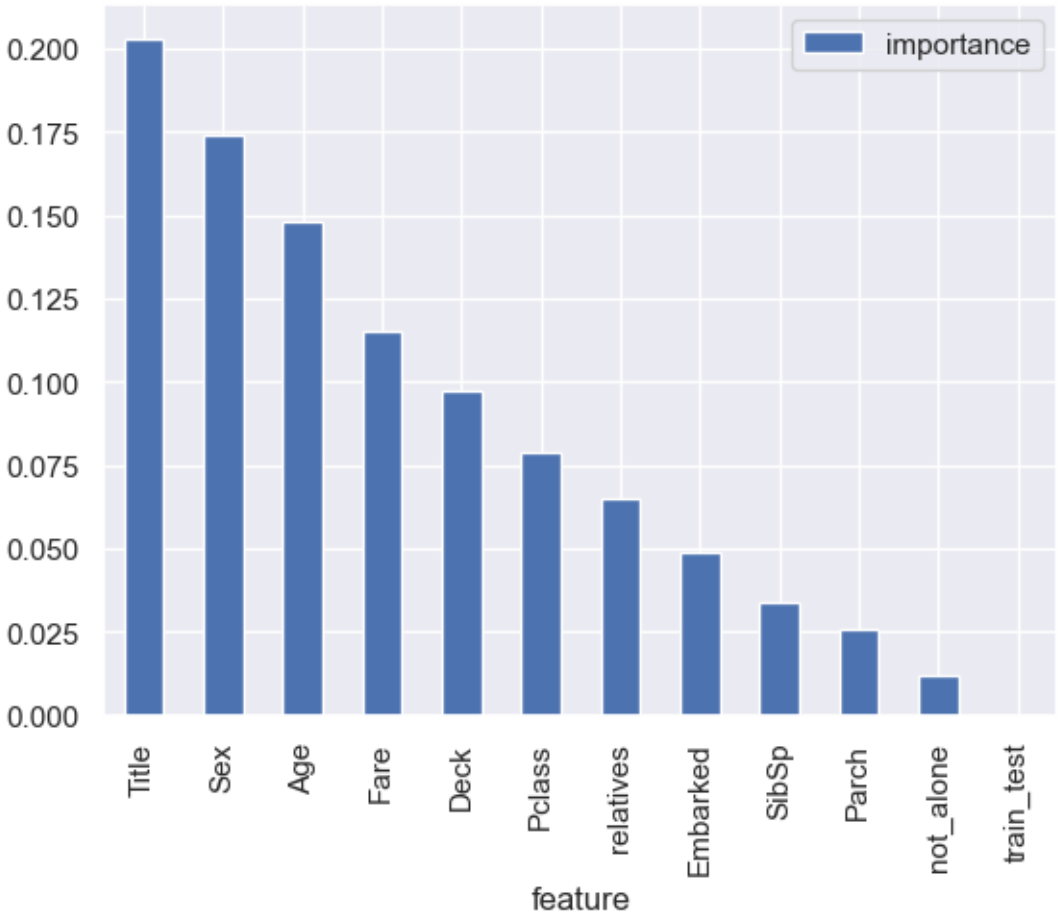
[69]:
```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.
 ↪round(random_forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).
 ↪set_index('feature')
```

[70]:
```
importances.head(12)
```

[70]:

feature	importance
Title	0.203
Sex	0.174
Age	0.148
Fare	0.115
Deck	0.097
Pclass	0.079
relatives	0.065
Embarked	0.049
SibSp	0.034
Parch	0.026
not_alone	0.012
train_test	0.000

[71]:
```
importances.plot.bar();
```



[72]:
```
# Dropping not_alone
train_df  = train_df.drop("not_alone", axis=1)
test_df   = test_df.drop("not_alone", axis=1)

# Dropping Parch
train_df  = train_df.drop("Parch", axis=1)
test_df   = test_df.drop("Parch", axis=1)
```

[73]:
```
# # Reassigning features
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
```

[74]:
```
random_forest = RandomForestClassifier(n_estimators=100, oob_score = True)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)
```

```
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)

# Print scores
print(round(acc_random_forest,2,), "%")
```

```
93.15 %
```

[75]:
```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.
 ↪round(random_forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).
 ↪set_index('feature')
```

[76]:
```
importances.head(12)
```

[76]:

feature	importance
Title	0.220
Sex	0.160
Age	0.145
Fare	0.114
Deck	0.091
Pclass	0.084
relatives	0.082
SibSp	0.055
Embarked	0.049
train_test	0.000

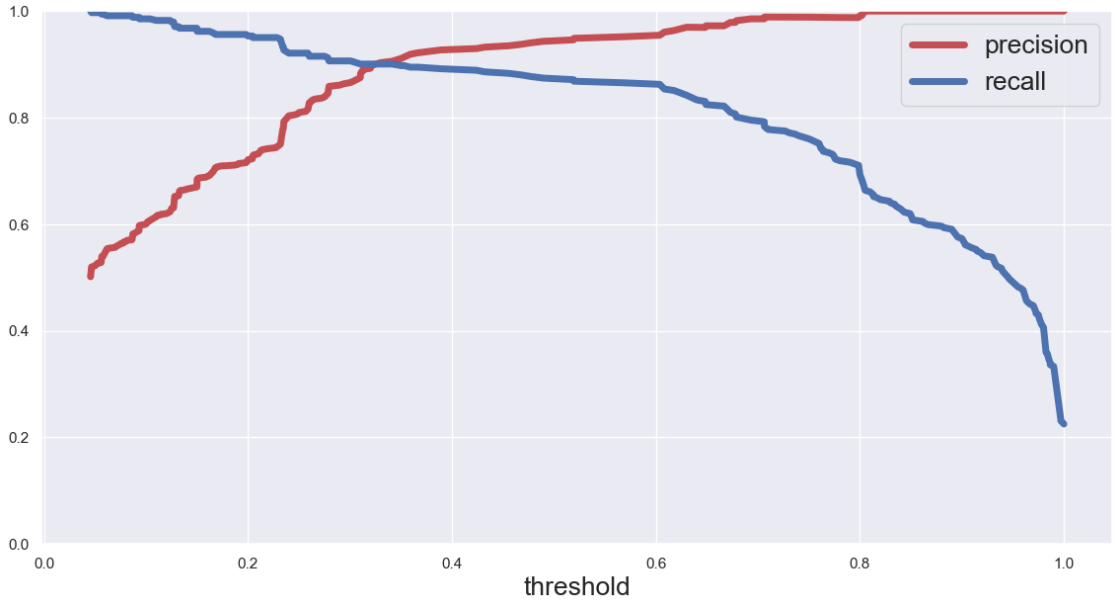
[77]:
```
print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
```

```
oob score: 81.82000000000001 %
```

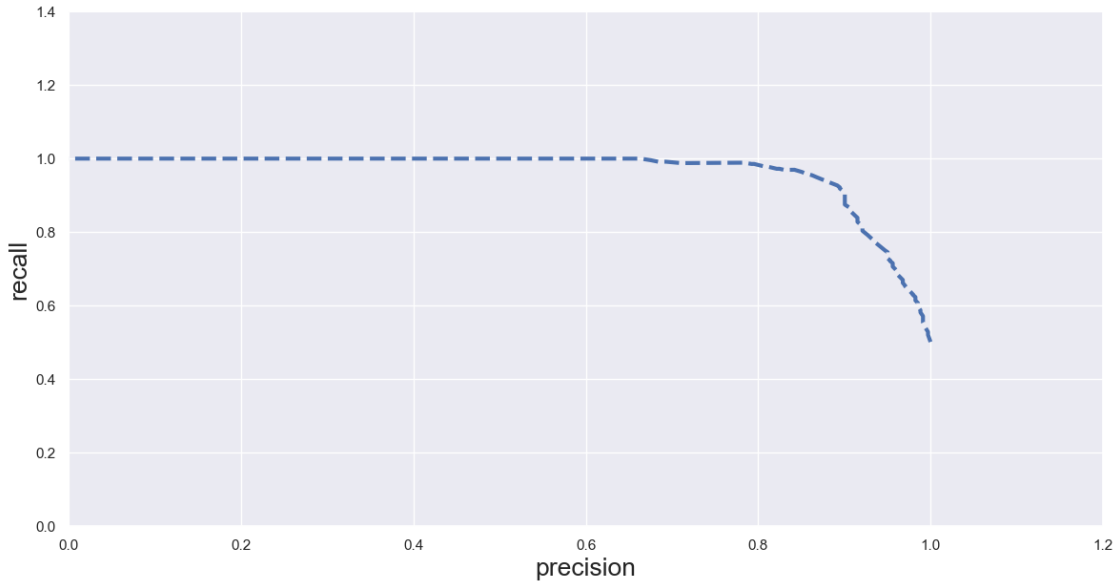
[78]:
```
# Simple performance reporting function
def clf_performance(classifier, model_name):
    print(model_name)
    print('Best Score: ' + str(classifier.best_score_))
    print('Best Parameters: ' + str(classifier.best_params_))
```

[79]:
```
from sklearn.metrics import precision_recall_curve

# Getting the probabilities of our predictions
y_scores = random_forest.predict_proba(X_train)
y_scores = y_scores[:,1]

precision, recall, threshold = precision_recall_curve(Y_train, y_scores)
def plot_precision_and_recall(precision, recall, threshold):
    plt.plot(threshold, precision[:-1], "r", label="precision", linewidth=5)
```

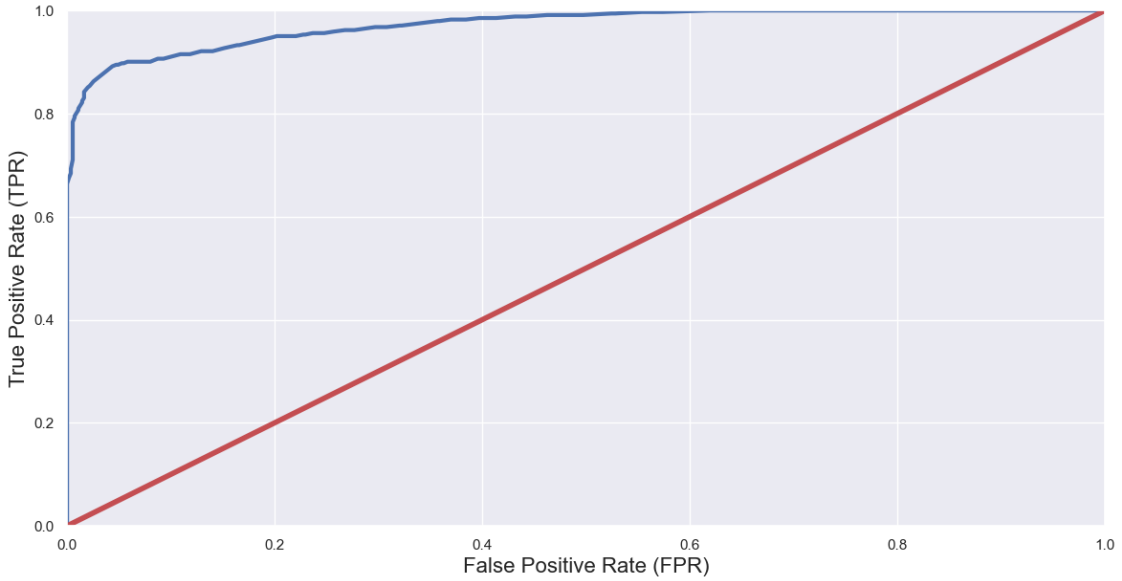
```
    plt.plot(threshold, recall[:-1], "b", label="recall", linewidth=5)
    plt.xlabel("threshold", fontsize=19)
    plt.legend(loc="upper right", fontsize=19)
    plt.ylim([0, 1])

plt.figure(figsize=(14, 7))
plot_precision_and_recall(precision, recall, threshold)
plt.show()
```



[80]:
```
def plot_precision_vs_recall(precision, recall):
    plt.plot(recall, precision, "b--", linewidth=3)
    plt.xlabel("precision", fontsize=19)
    plt.ylabel("recall", fontsize=19)
    plt.axis([0, 1.2, 0, 1.4])

plt.figure(figsize=(14, 7))
plot_precision_vs_recall(precision, recall)
plt.show()
```



[81]:
```
from sklearn.metrics import roc_curve

# Compute true positive rate and false positive rate
false_positive_rate, true_positive_rate, thresholds = roc_curve(Y_train,␣
 ↪y_scores)

# Plotting them against each other
def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=3, label=label)
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```



[82]:
```
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(Y_train, y_scores)
print("ROC-AUC-Score:", r_a_score)
```

```
ROC-AUC-Score: 0.9717641858136536
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: