lab05-aiml

April 26, 2024

1 Logistic Regression with Titanic data set

1.1 Import packages and dataset

[1]:
```
#import nbconvert #recode the dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

train = pd.read_csv('titanic_train.csv') # Training set is already available
train.head()
```

[1]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

1.1.1 Check basic info about the data set including missing value

[3]:
```
train.info(verbose=True)
```

```
<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
```

[4]:
```
d=train.describe()
d
```

[4]:

	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
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200
75%	0.000000	31.000000
max	6.000000	512.329200

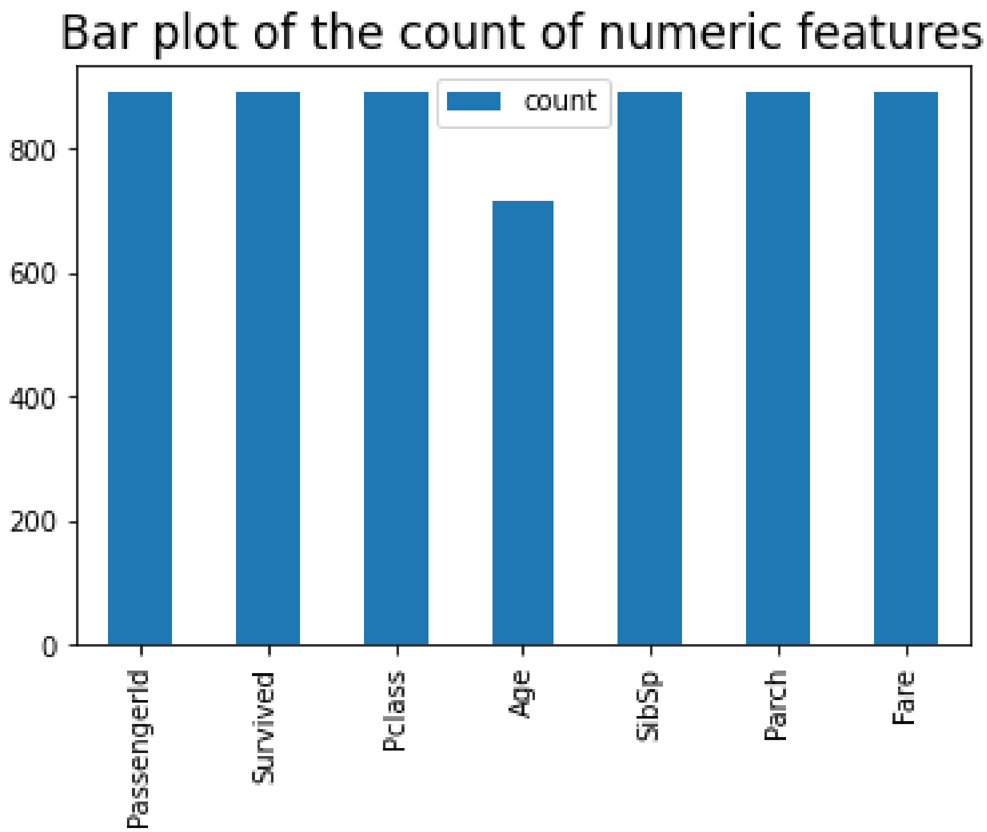
1.2 Exploratory analysis and plots

Plot a bar diagram to check the number of numeric entries

From the bar diagram, it shows that there are some age entries missing as the number of count for 'Age' is less than the other counts. We can do some impute/transformation of the data to fill-up the missing entries.

[5]:
```
dT=d.T
dT.plot.bar(y='count')
plt.title("Bar plot of the count of numeric features",fontsize=17)
```

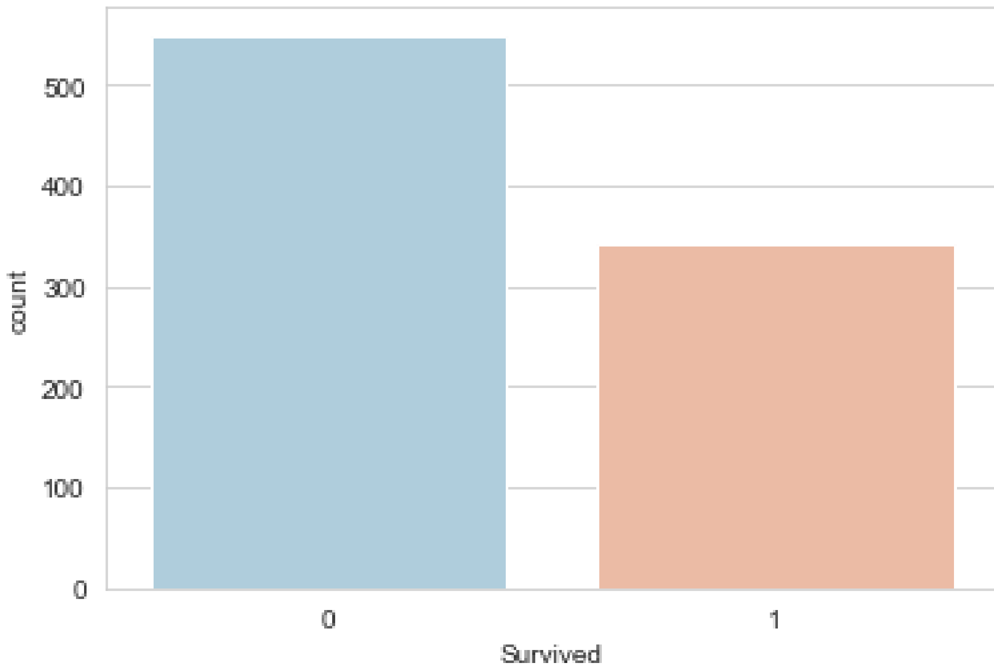
[5]: `Text(0.5, 1.0, 'Bar plot of the count of numeric features')`



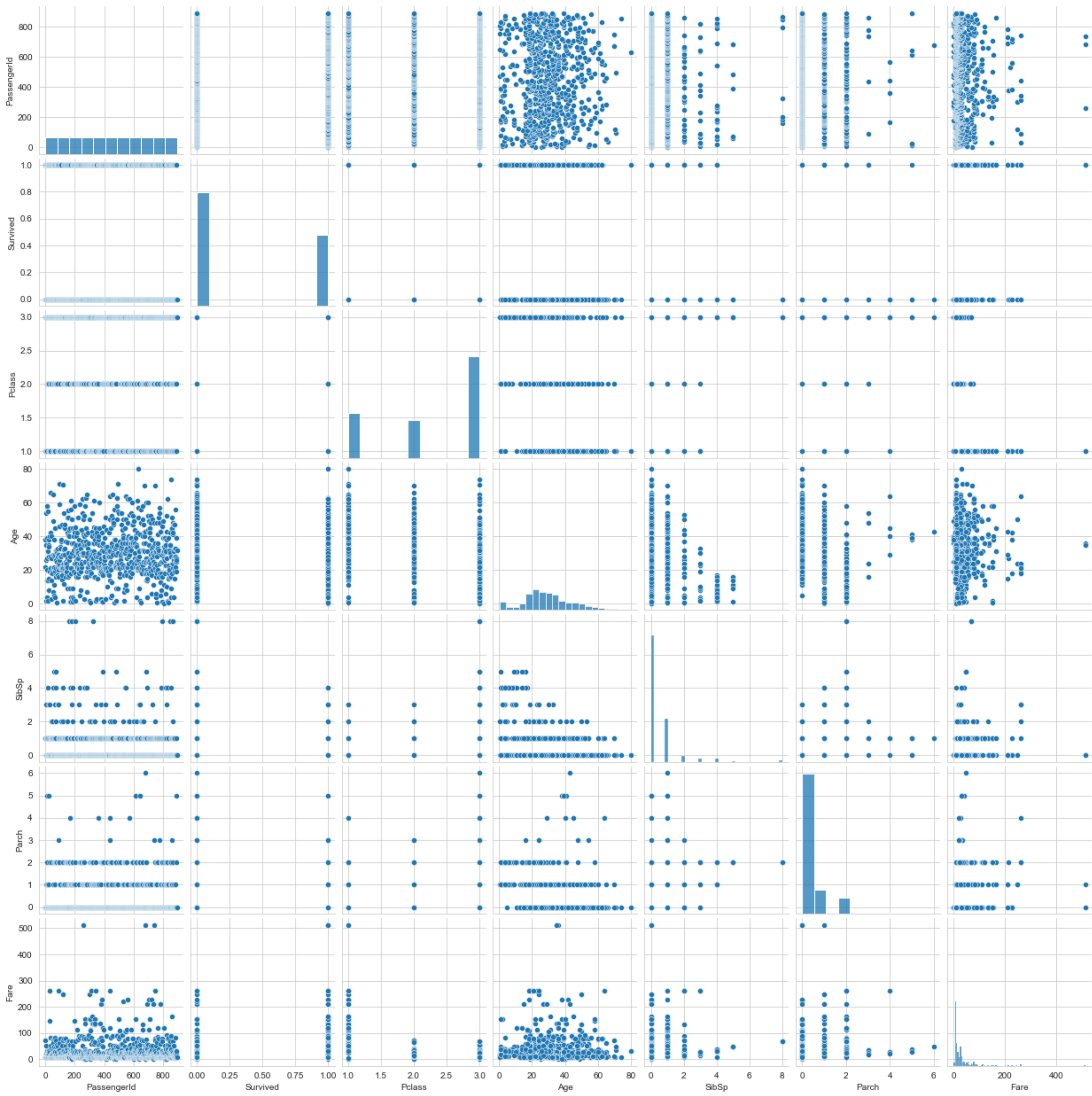
Check the relative size of survived and not-survived

[6]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
sns.pairplot(train)
```

[6]: `<seaborn.axisgrid.PairGrid at 0x11fcc09a0>`



500
400
300
200
100
0
count
0
1
Survived

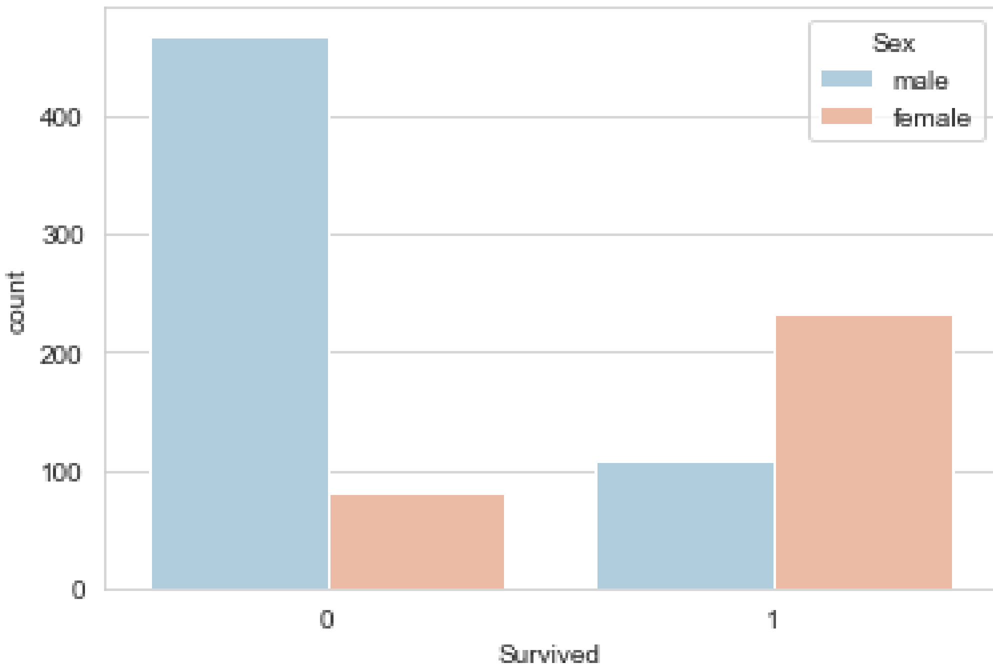


Is there a pattern for the survivability based on sex?

It looks like more female survived than males!

[3]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

[3]: <AxesSubplot:xlabel='Survived', ylabel='count'>

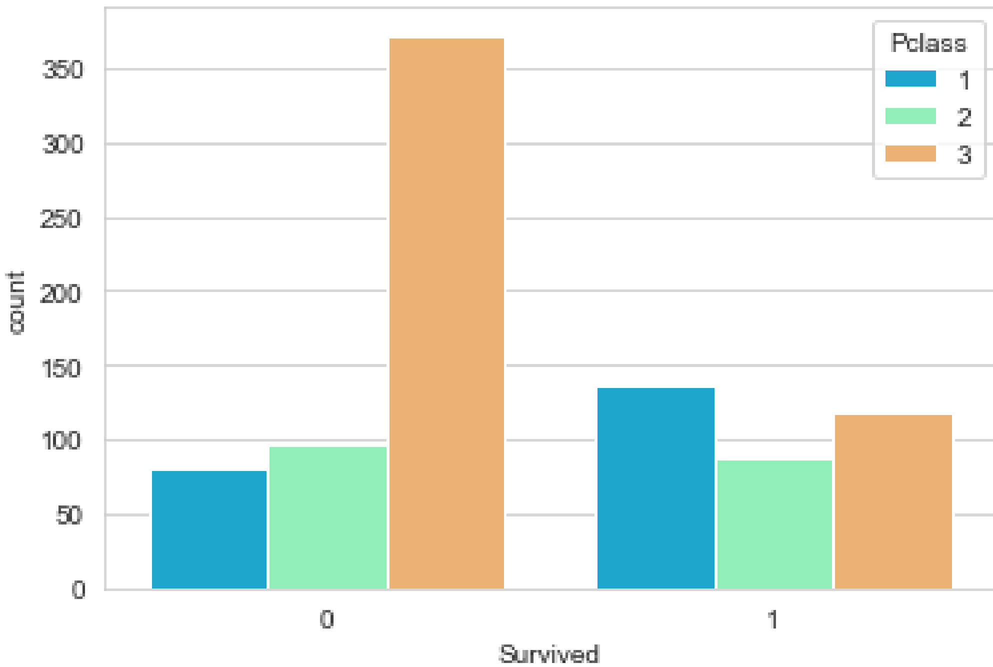


What about any pattern related to passenger class?

It looks like disproportionately large number of 3rd class passengers died!

[9]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

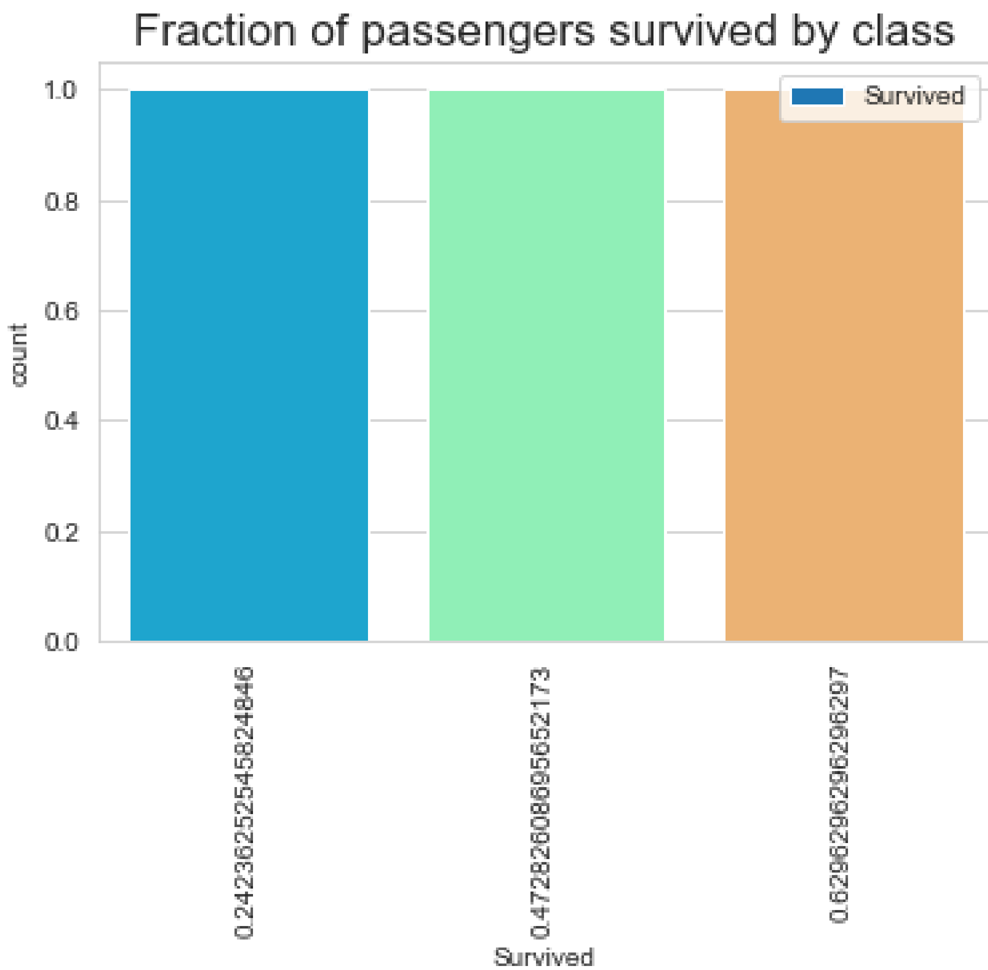
[9]: <AxesSubplot:xlabel='Survived', ylabel='count'>



Following code extracts and plots the fraction of passenger count that survived, by each class

[10]:
```
f_class_survived=train.groupby('Pclass')['Survived'].mean()
f_class_survived = pd.DataFrame(f_class_survived)
f_class_survived
f_class_survived.plot.bar(y='Survived')
sns.countplot(x='Survived',data=f_class_survived,palette='rainbow')
plt.title("Fraction of passengers survived by class",fontsize=17)
```

[10]:
```
Text(0.5, 1.0, 'Fraction of passengers survived by class')
```

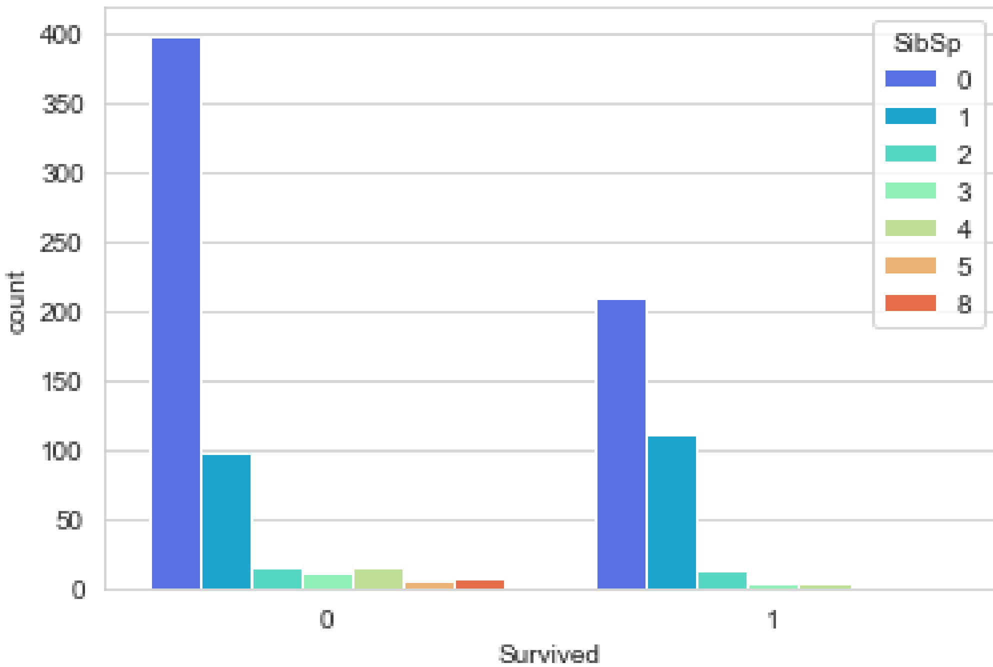


What about any pattern related to having sibling and spouse?

It looks like there is a weak trend that chance of survibility increased if there were more number of sibling or spouse

[9]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='SibSp',data=train,palette='rainbow')
```

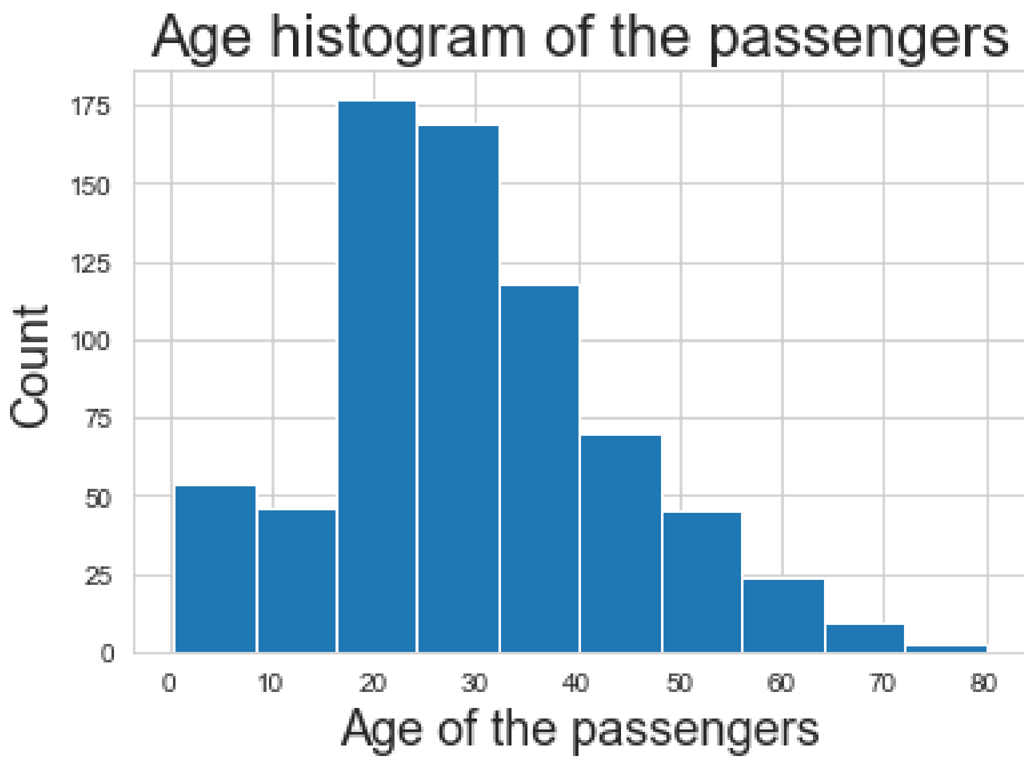
[9]: <AxesSubplot:xlabel='Survived', ylabel='count'>



How does the overall age distribution look like?

[10]:
```
plt.xlabel("Age of the passengers",fontsize=18)
plt.ylabel("Count",fontsize=18)
plt.title("Age histogram of the passengers",fontsize=22)
#train['Age'].hist(bins=30,color='darkred',alpha=0.7,figsize=(10,6))
train['Age'].hist()
```

[10]:
```
<AxesSubplot:title={'center':'Age histogram of the passengers'}, xlabel='Age of
the passengers', ylabel='Count'>
```

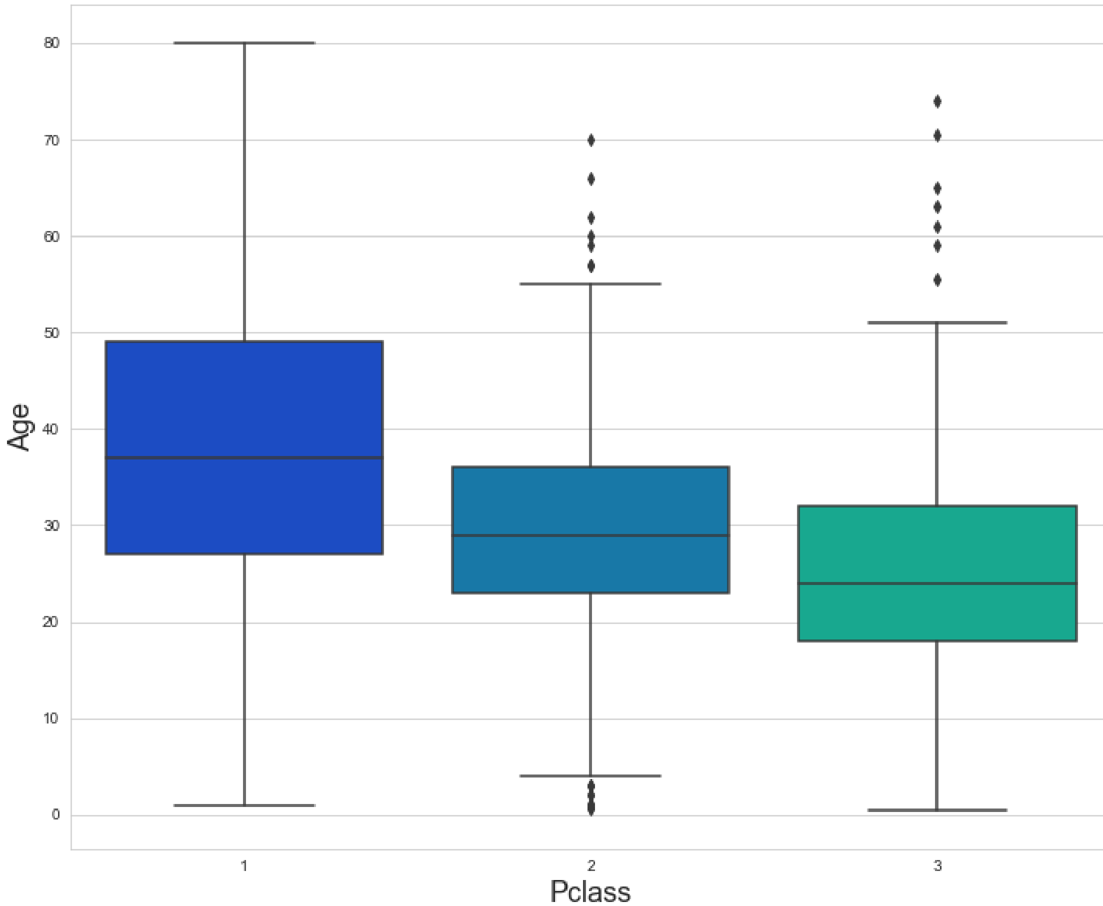


How does the age distribution look like across passenger class?

It looks like that the average age is different for three classes and it generally decreases from 1st class to 3rd class.

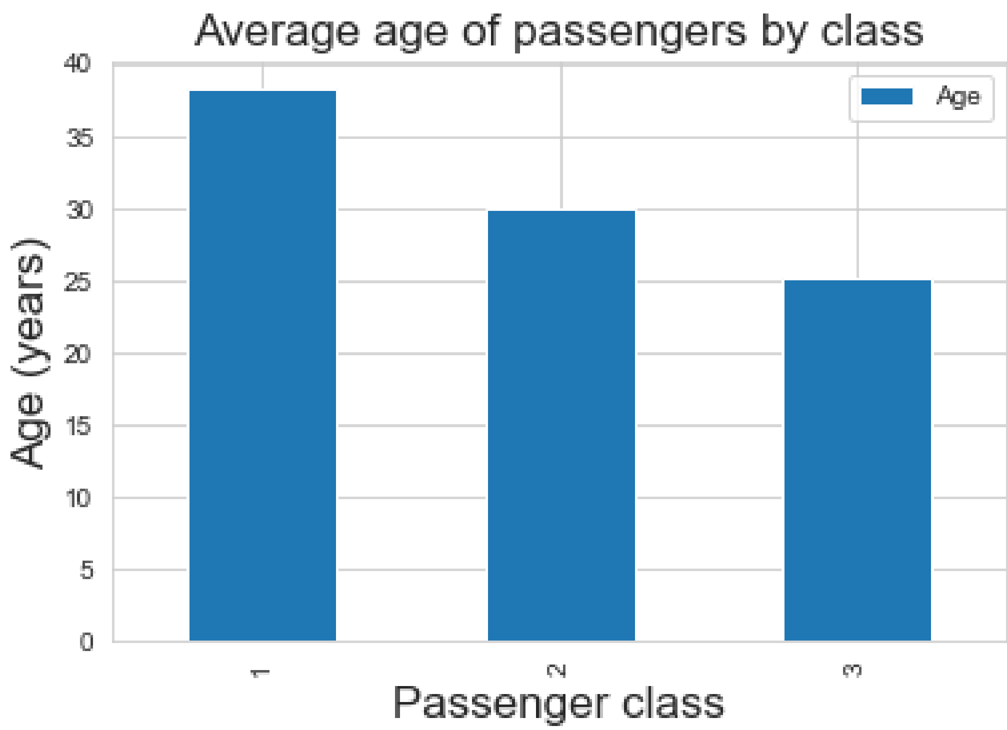
[11]:
```
plt.figure(figsize=(12, 10))
plt.xlabel("Passenger Class",fontsize=18)
plt.ylabel("Age",fontsize=18)
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

[11]: `<AxesSubplot:xlabel='Pclass', ylabel='Age'>`




```
[5]: f_class_Age=train.groupby('Pclass')['Age'].mean()
     f_class_Age = pd.DataFrame(f_class_Age)

     f_class_Age.plot.bar(y='Age')
     plt.title("Average age of passengers by class",fontsize=17)
     plt.ylabel("Age (years)", fontsize=17)
     plt.xlabel("Passenger class", fontsize=17)
```

```
[5]: Text(0.5, 0, 'Passenger class')
```



1.3 Data wrangling (impute and drop)

- Impute age (by averaging)
- Drop unncessary features
- Convert categorical features to dummy variables

1.3.1 Define a function to impute (fill-up missing values) age feature

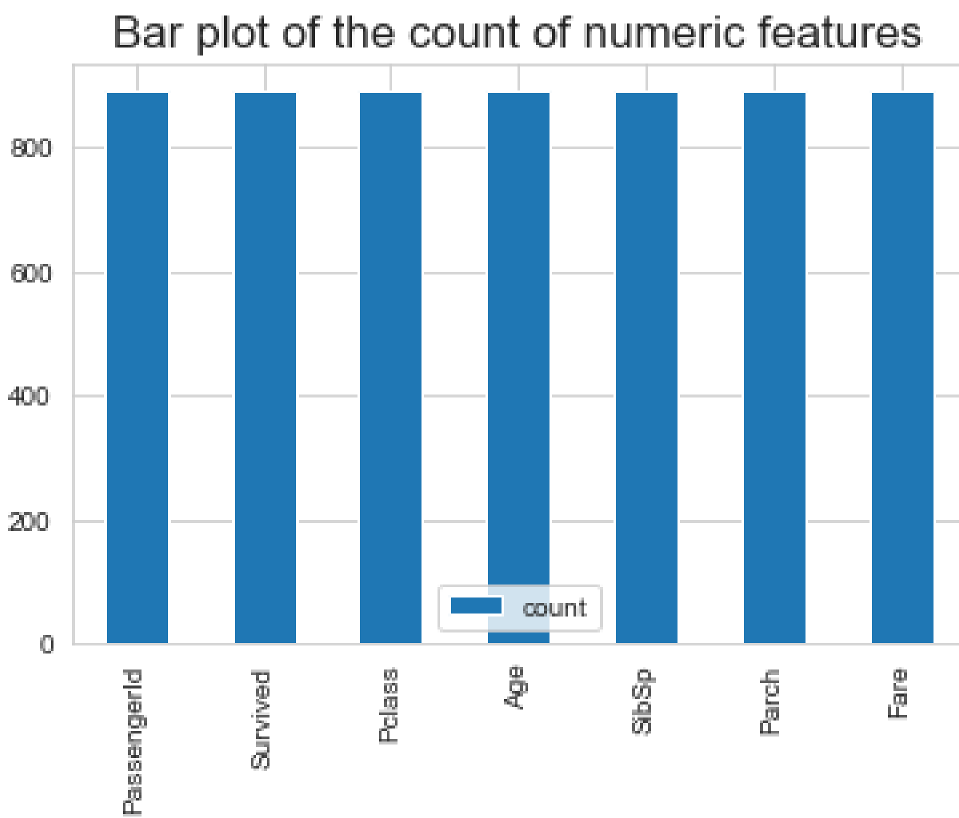
[6]:
```
a=list(f_class_Age['Age'])

def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return a[0]

        elif Pclass == 2:
            return a[1]

        else:
```

```
            return a[2]

    else:
        return Age
```

Apply the above-defined function and plot the count of numeric features

[7]:
```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
d=train.describe()

dT=d.T
dT.plot.bar(y='count')
plt.title("Bar plot of the count of numeric features",fontsize=17)
```

[7]:
```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```



1.3.2 Drop the 'Cabin' feature and any other null value

[16]:
```
train.drop('Cabin',axis=1,inplace=True)
train.dropna(inplace=True)
train.head()
```

[16]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Embarked
0      0         A/5 21171   7.2500        S
1      0          PC 17599  71.2833        C
2      0  STON/O2. 3101282   7.9250        S
3      0            113803  53.1000        S
4      0            373450   8.0500        S
```

1.3.3 Drop other unnecessary features

like 'Cabin','PassengerId', 'Name', 'Ticket'

[8]:
```
train.drop(['Cabin','PassengerId','Name','Ticket'],axis=1,
           inplace=True)
train.dropna(inplace=True)
train.head()
```

[8]:
```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         0       3    male  22.0      1      0   7.2500        S
1         1       1  female  38.0      1      0  71.2833        C
2         1       3  female  26.0      0      0   7.9250        S
3         1       1  female  35.0      1      0  53.1000        S
4         0       3    male  35.0      0      0   8.0500        S
```

1.3.4 Convert categorial feature like 'Sex'

and 'Embarked' to dummy variables

Use pandas 'get_dummies()' function

[15]:
```python
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.9/site-packages/pandas/core/indexes/base.py:3621, in
↪Index.get_loc(self, key, method, tolerance)
   3620 try:
-> 3621     return self._engine.get_loc(casted_key)
   3622 except KeyError as err:

File /usr/local/lib/python3.9/site-packages/pandas/_libs/index.pyx:136, in
↪pandas._libs.index.IndexEngine.get_loc()

File /usr/local/lib/python3.9/site-packages/pandas/_libs/index.pyx:163, in
↪pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5198, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5206, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'Sex'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 sex = pd.get_dummies(train['Sex'],drop_first=True)
      2 embark = pd.get_dummies(train['Embarked'],drop_first=True)

File /usr/local/lib/python3.9/site-packages/pandas/core/frame.py:3505, in
↪DataFrame.__getitem__(self, key)
   3503 if self.columns.nlevels > 1:
   3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
   3506 if is_integer(indexer):
   3507     indexer = [indexer]

File /usr/local/lib/python3.9/site-packages/pandas/core/indexes/base.py:3623, in
↪Index.get_loc(self, key, method, tolerance)
   3621     return self._engine.get_loc(casted_key)
   3622 except KeyError as err:
-> 3623     raise KeyError(key) from err
   3624 except TypeError:
   3625     # If we have a listlike key, _check_indexing_error will raise
```

```
3626     #  InvalidIndexError. Otherwise we fall through and re-raise
3627     #  the TypeError.
3628     self._check_indexing_error(key)

KeyError: 'Sex'
```

Now drop the 'Sex' and 'Embarked' columns and concatenate the new dummy variables

[14]:
```
train.drop(['Sex','Embarked'],axis=1,inplace=True)
train = pd.concat([train,sex,embark],axis=1)
train.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [14], in <cell line: 1>()
----> 1 train.drop(['Sex','Embarked'],axis=1,inplace=True)
      2 train = pd.concat([train,sex,embark],axis=1)
      3 train.head()

File /usr/local/lib/python3.9/site-packages/pandas/util/_decorators.py:311, in␣
 ↪deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args,␣
 ↪**kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File /usr/local/lib/python3.9/site-packages/pandas/core/frame.py:4954, in␣
 ↪DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4806 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self",␣
 ↪"labels"])
   4807 def drop(
   4808     self,
   (…)
   4815     errors: str = "raise",
   4816 ):
   4817     """
   4818     Drop specified labels from rows or columns.
   4819
   (…)
   4952             weight  1.0     0.8
   4953     """
-> 4954     return super().drop(
```

```
   4955         labels=labels,
   4956         axis=axis,
   4957         index=index,
   4958         columns=columns,
   4959         level=level,
   4960         inplace=inplace,
   4961         errors=errors,
   4962     )

File /usr/local/lib/python3.9/site-packages/pandas/core/generic.py:4267, in␣
 ↪NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4265 for axis, labels in axes.items():
   4266     if labels is not None:
-> 4267         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4269 if inplace:
   4270     self._update_inplace(obj)

File /usr/local/lib/python3.9/site-packages/pandas/core/generic.py:4311, in␣
 ↪NDFrame._drop_axis(self, labels, axis, level, errors, consolidate, only_slice
   4309         new_axis = axis.drop(labels, level=level, errors=errors)
   4310     else:
-> 4311         new_axis = axis.drop(labels, errors=errors)
   4312     indexer = axis.get_indexer(new_axis)
   4314 # Case for non-unique axis
   4315 else:

File /usr/local/lib/python3.9/site-packages/pandas/core/indexes/base.py:6644, i␣
 ↪Index.drop(self, labels, errors)
   6642 if mask.any():
   6643     if errors != "ignore":
-> 6644         raise KeyError(f"{list(labels[mask])} not found in axis")
   6645     indexer = indexer[~mask]
   6646 return self.delete(indexer)

KeyError: "['Sex', 'Embarked'] not found in axis"
```

This data set is now ready for logistic regression analysis!

1.4 Logistic Regression model fit and prediction

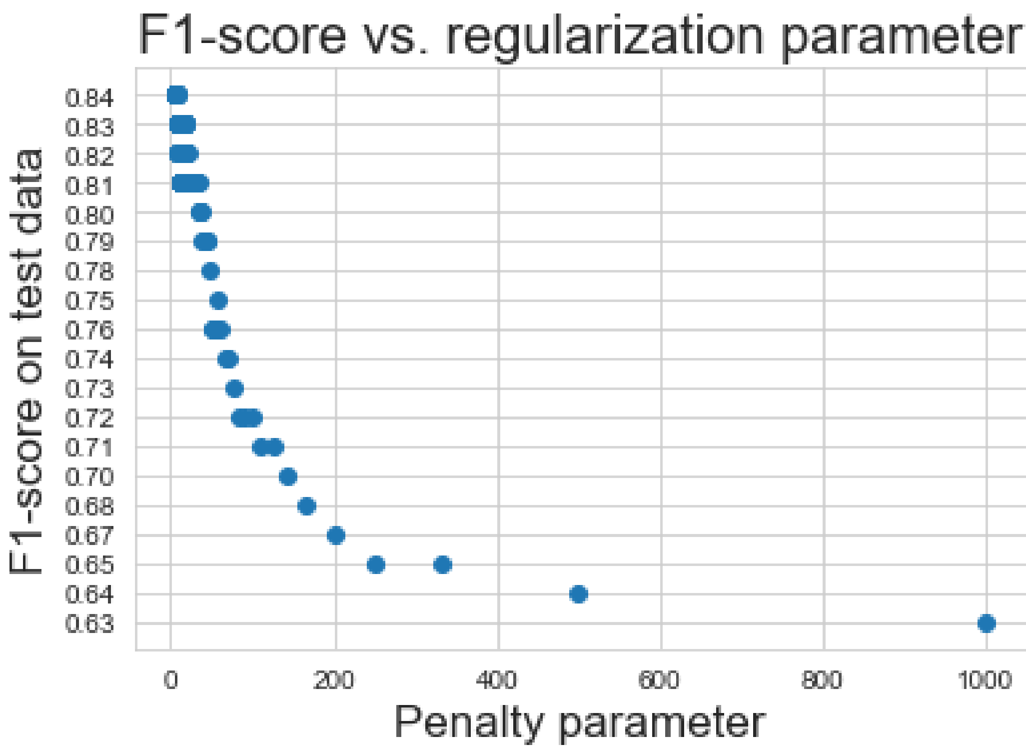
Let's start by splitting our data into a training set and test set (there is another test.csv file that you can play around with in case you want to use all this data for training).

[22]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    train.drop('Survived',axis=1),train['Survived'],
```

```
    test_size=0.30,random_state=111)
```

1.4.1 F1-score as a function of regularization (penalty) parameter

[21]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
nsimu=201
penalty=[0]*nsimu
logmodel=[0]*nsimu
predictions =[0]*nsimu
class_report = [0]*nsimu
f1=[0]*nsimu

for i in range(1,nsimu):
        logmodel[i] =(LogisticRegression(C=i/1000,tol=1e-4, max_iter=int(1e6),
                                         n_jobs=4))
        logmodel[i].fit(X_train,y_train)
        predictions[i] = logmodel[i].predict(X_test)
        class_report[i] = classification_report(y_test,predictions[i])
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        penalty[i]=1000/i

plt.scatter(penalty[1:len(penalty)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. regularization parameter",fontsize=20)
plt.xlabel("Penalty parameter",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```



1.4.2 F1-score as a function of test set size (fraction)

[ ]:
```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu
test_fraction =[0]*nsimu
for i in range(1,nsimu):
        X_train, X_test, y_train, y_test = train_test_split(train.
 ↪drop('Survived',axis=1),
                                                            train['Survived'],␣
 ↪test_size=0.1+(i-1)*0.007,
                                                            random_state=111)
        logmodel =(LogisticRegression(C=1,tol=1e-4, max_iter=1000,n_jobs=4))
        logmodel.fit(X_train,y_train)
        predictions = logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        test_fraction[i]=0.1+(i-1)*0.007

plt.plot(test_fraction[1:len(test_fraction)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. test set size (fraction)",fontsize=20)
```

```
plt.xlabel("Test set size (fraction)",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```

1.4.3 F1-score as a function of random seed of test/train split

[ ]:
```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu
random_init =[0]*nsimu
for i in range(1,nsimu):
        X_train, X_test, y_train, y_test = train_test_split(train.
 ↪drop('Survived',axis=1),
                                                    train['Survived'],␣
 ↪test_size=0.3,
                                                    random_state=i+100)
        logmodel =(LogisticRegression(C=1,tol=1e-5, max_iter=1000,n_jobs=4))
        logmodel.fit(X_train,y_train)
        predictions = logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
        l=class_report[i].split()
        f1[i] = l[len(l)-2]
        random_init[i]=i+100

plt.plot(random_init[1:len(random_init)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. random initialization seed",fontsize=20)
plt.xlabel("Random initialization seed",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17)
plt.show()
```