# Lab-05 Logistic Regression with Titanic data set[¶](#Logistic-Regression-with-Titanic-data-s)

## Import packages and dataset[¶](#Import-packages-and-dataset)

train = pd.read\_csv('titanic\_train.csv') # Training set is already available

train.tail()

### Check basic info about the data set including missing value[¶](#Check-basic-info-about-the-data-set-inc)

train.info(verbose=True)

d=train.describe()

d

## Exploratory analysis and plots[¶](#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.

dT=d.T

dT.plot.bar(y='count')

plt.title("Bar plot of the count of numeric features",fontsize=17)

**Check the relative size of survived and not-survived**

sns.set\_style('whitegrid')

sns.countplot(x='Survived',data=train,palette='RdBu\_r')

sns.pairplot(train)

**Is there a pattern for the survivability based on sex?**

It looks like more female survived than males!

sns.set\_style('whitegrid')

sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu\_r')

**What about any pattern related to passenger class?**

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

sns.set\_style('whitegrid')

sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')

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

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)

**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

sns.set\_style('whitegrid')

sns.countplot(x='Survived',hue='SibSp',data=train,palette='rainbow')

**How does the overall age distribution look like?**

plt.xlabel("Age of the passengers",fontsize=)

plt.ylabel("Count",fontsize=)

plt.title("Histogram of the passengers",fontsize=)

#train['Age'].hist(bins=30,color='darkred',alpha=0.7,figsize=(10,6))

train['Age'].hist()

**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.

plt.figure(figsize=(12, 10))

plt.xlabel("Passenger Class",fontsize=)

plt.ylabel("Age",fontsize=)

sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')

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=)

plt.ylabel("Age (years)", fontsize=)

plt.xlabel("Passenger class", fontsize=)

## Data wrangling (impute and drop)[¶](#Data-wrangling-(impute-and-drop))

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

### Define a function to impute (fill-up missing values) age feature[¶](#Define-a-function-to-impute-(fill-up-mi)

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**

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)

### Drop the 'Cabin' feature and any other null value[¶](#Drop-the-'Cabin'-feature-and-any-other-)

train.drop('Cabin',axis=1,inplace=True)

train.dropna(inplace=True)

train.head()

### Drop other unnecessary features[¶](#Drop-other-unnecessary-features)

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

train.drop(['PassengerId','Name','Ticket'],axis=1,

inplace=True)

train.dropna(inplace=True)

train.head()

### Convert categorial feature like 'Sex'[¶](#Convert-categorial-feature-like-'Sex')

and 'Embarked' to dummy variables

**Use pandas 'get\_dummies()' function**

sex = pd.get\_dummies(train['Sex'],drop\_first=True)

embark = pd.get\_dummies(train['Embarked'],drop\_first=True)

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

train.drop(['Sex','Embarked'],axis=1,inplace=True)

train = pd.concat([train,sex,embark],axis=1)

train.head()

This data set is now ready for logistic regression analysis!

## Logistic Regression model fit and prediction[¶](#Logistic-Regression-model-fit-and-predi)

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).

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

train.drop('Survived',axis=1),train['Survived'],

test\_size=0.40,random\_state=121)

### F1-score as a function of regularization (penalty) parameter[¶](#F1-score-as-a-function-of-regularizatio)

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)

### F1-score as a function of test set size (fraction)[¶](#F1-score-as-a-function-of-test-set-size)

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)

### F1-score as a function of random seed of test/train split[¶](#F1-score-as-a-function-of-random-seed-o)

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)