In [15]:
```
import numpy as nm
import pandas as pd
import os

# loc will be used ahead. Loc is used to get details of a perticular col
# example is we have name, age salary, and we write a=df.loc['peter'], t
```

In [6]:
```
train_df=pd.read_csv('train.csv', index_col='PassengerId')
test_df= pd.read_csv('test.csv', index_col='PassengerId')
```

In [7]:
```
type(train_df)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 18 columns):
Survived      891 non-null int64
Age           891 non-null float64
Fare          891 non-null float64
IsMale        891 non-null int64
Deck_A        891 non-null int64
Deck_B        891 non-null int64
Deck_C        891 non-null int64
Deck_D        891 non-null int64
Deck_E        891 non-null int64
Deck_F        891 non-null int64
Deck_G        891 non-null int64
Deck_z        891 non-null int64
Pclass_1      891 non-null int64
Pclass_2      891 non-null int64
Pclass_3      891 non-null int64
Embarked_C    891 non-null int64
Embarked_Q    891 non-null int64
Embarked_S    891 non-null int64
dtypes: float64(2), int64(16)
memory usage: 132.3 KB
```

In [9]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 892 to 1309
Data columns (total 17 columns):
Age           418 non-null float64
Fare          418 non-null float64
IsMale        418 non-null int64
Deck_A        418 non-null int64
Deck_B        418 non-null int64
Deck_C        418 non-null int64
Deck_D        418 non-null int64
Deck_E        418 non-null int64
Deck_F        418 non-null int64
Deck_G        418 non-null int64
Deck_z        418 non-null int64
Pclass_1      418 non-null int64
Pclass_2      418 non-null int64
Pclass_3      418 non-null int64
Embarked_C    418 non-null int64
Embarked_Q    418 non-null int64
Embarked_S    418 non-null int64
dtypes: float64(2), int64(15)
memory usage: 58.8 KB
```

In [10]:
```
##there is no survived column in test, because this is what we are suppo

# also there are lot of missing values because total values in each shou
```

In [11]:
```
test_df['Survived']=-888 # adding surevived with a default value
```

In [12]:
```
df=pd.concat((train_df, test_df), axis=0, sort=False)  #making a new dat
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 18 columns):
Survived      1309 non-null int64
Age           1309 non-null float64
Fare          1309 non-null float64
IsMale        1309 non-null int64
Deck_A        1309 non-null int64
Deck_B        1309 non-null int64
Deck_C        1309 non-null int64
Deck_D        1309 non-null int64
Deck_E        1309 non-null int64
Deck_F        1309 non-null int64
Deck_G        1309 non-null int64
Deck_z        1309 non-null int64
Pclass_1      1309 non-null int64
Pclass_2      1309 non-null int64
Pclass_3      1309 non-null int64
Embarked_C    1309 non-null int64
Embarked_Q    1309 non-null int64
Embarked_S    1309 non-null int64
dtypes: float64(2), int64(16)
memory usage: 194.3 KB
```

In [14]:
```
df[['Name','Age']]
```

In [15]:
```
df.describe()
```

Out[15]:

	Survived	Age	Fare	IsMale	Deck_A	Deck_B
count	1309.000000	1309.000000	1309.000000	1309.000000	1309.000000	1309.000000
mean	-283.301757	29.881138	33.276193	0.644003	0.016807	0.049656
std	414.337413	12.883193	51.743584	0.478997	0.128596	0.217317
min	-888.000000	0.170000	0.000000	0.000000	0.000000	0.000000
25%	-888.000000	22.000000	7.895800	0.000000	0.000000	0.000000
50%	0.000000	29.881138	14.454200	1.000000	0.000000	0.000000
75%	1.000000	35.000000	31.275000	1.000000	0.000000	0.000000
max	1.000000	80.000000	512.329200	1.000000	1.000000	1.000000

In [16]:
```
print(df.Fare.min())
```

```
0.0
```

In [17]:
```
print(df.Fare.max())
```

```
512.3292
```

In [18]:
```
df.Sex.value_counts()
```

```
#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
```

In [19]:
```
df.Sex.value_counts(normalize=True)   # the data is normalized, that mea

#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
```

In [20]:
```
df.Fare.plot(kind='box')
```

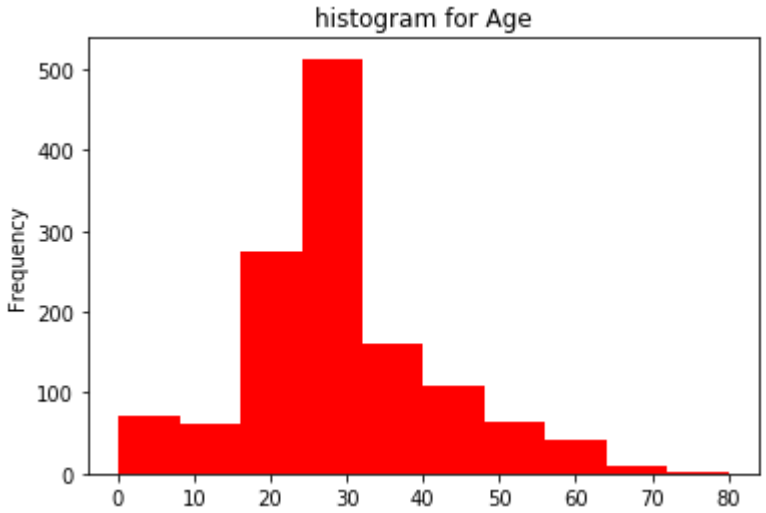
Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb633f43850>
```

In [21]:
```
df.Pclass.value_counts().plot(kind='bar')

#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
```

In [22]:
```
df.Pclass.value_counts().plot(kind='bar', rot=0, title='Class wise passe

#rot is for rotating the labels
#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
```

In [23]:
```
df.Age.plot(kind='hist', title='histogram for Age', color='r')
```

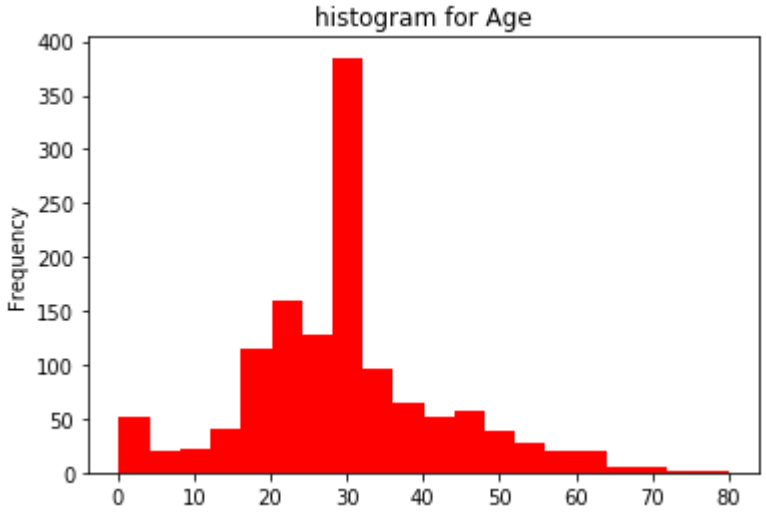
Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb633a5ac50>
```



In [24]:
```
df.Age.plot(kind='hist', title='histogram for Age', color='r', bins=20)
```

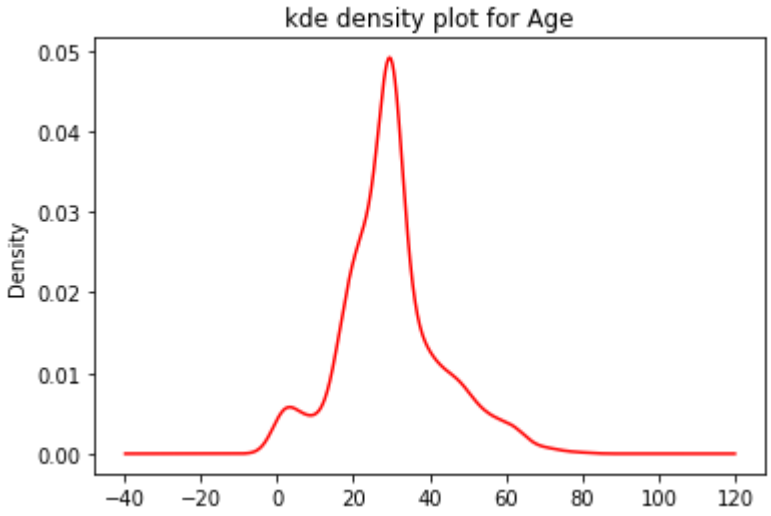
```
#bins are used to create range for yaxis, like 0-20, 20-40 bin is 20
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb633788810>



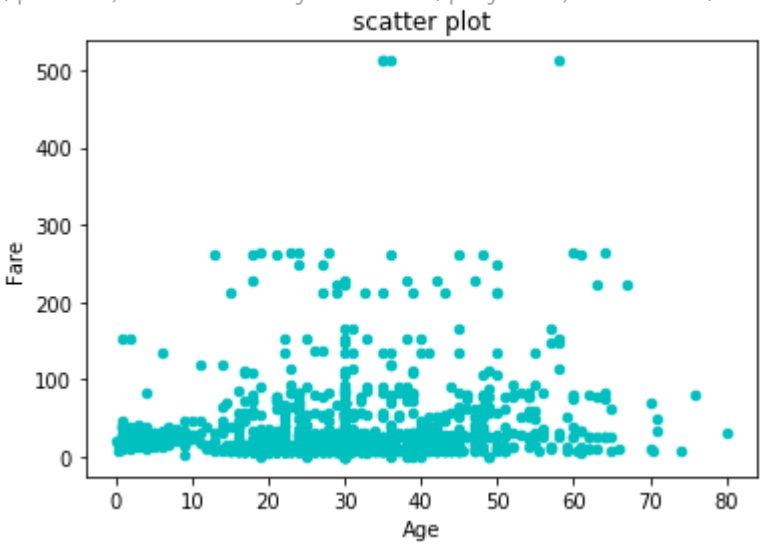
In [25]:
```
df.Age.plot(kind='kde', title='kde density plot for Age', color='r')
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb63339d450>



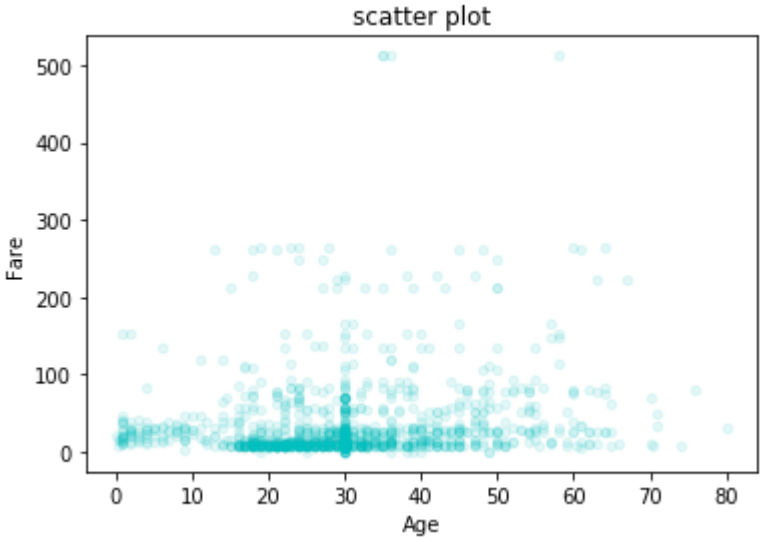
In [26]:
```
df.plot.scatter(x='Age', y='Fare', color='c', title='scatter plot')
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb633312f50>

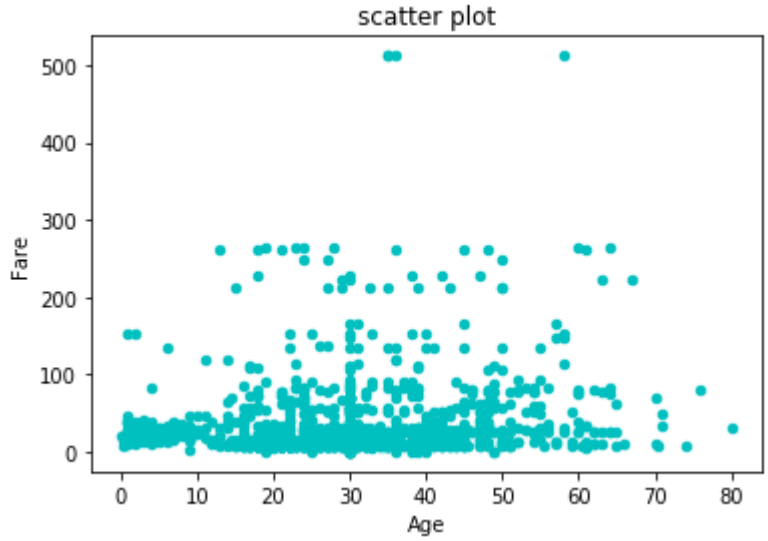


In [27]:
```
df.plot.scatter(x='Age', y='Fare', color='c', title='scatter plot', alph
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb633289d50>



In [28]:
```
df.plot.scatter(x='Age', y='Fare', color='c', title='scatter plot');
#semicolon for not getting the <matplotlib.axes._subplots.AxesSubplot at
```



In [29]:
```
df.info
```

Out[29]:
```
<bound method DataFrame.info of              Survived        Age
Fare  IsMale  Deck_A  Deck_B  Deck_C  \
PassengerId
1                   0  22.000000    7.2500       1       0       0
0
2                   1  38.000000   71.2833       0       0       0
1
3                   1  26.000000    7.9250       0       0       0
0
4                   1  35.000000   53.1000       0       0       0
1
5                   0  35.000000    8.0500       1       0       0
0
6                   0  29.881138    8.4583       1       0       0
0
7                   0  54.000000   51.8625       1       0       0
0
8                   0   2.000000   21.0750       1       0       0
0
```

In [30]:
```
#Grouping is used to group the available dataset into different group ar

# like grouping the whole dataset into male and female and then getting

df.groupby('Sex').Age.median()

#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
```

In [31]:
```
df.groupby(['Pclass']).Fare.median()
#commented out because this needs to run only one time and not after the
```

```
#else it will give error because later on we will be removin this columr
#we can see that the first class passengers pay more average fee than 2r
```

In [32]:
```
df.groupby(['Pclass'])['Fare', 'Age'].median()
#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
#
```

In [33]:
```
df.groupby(['Pclass']).agg({'Fare':'mean','Age':'median'})
#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
#here we have the mean of the fare and median of the age with Pclass as
```

In [34]:
```
## Crosstab

#Crosstabs are like 2D matrix of data, which helps us to get more inform
# We use two different data information and plot them in 2D matrix and s

#Like how many 1st class passengers were men and how many were women. Si
#how many 2nd class passenger were men and how many women
```

In [35]:
```
pd.crosstab(df.Sex, df.Pclass)
#commented out because this needs to run only one time and not after the
#else it will give error because later on we will be removin this columr
#here we have a matrix betweem two different features
```

In [36]:
```
pd.crosstab(df.Sex, df.Pclass).plot(kind='bar');

#here we plotted the matrix that we would have obtained
```

In [37]:
```
##Pivots (kind of natural extension of the crosstab)

#the difference between crosstab and pivot table is that in pivot table
# Like we apply mean age to the male and female passengers of each class
```

In [38]:
```
df.pivot_table(index='Sex', columns='Pclass', values='Age', aggfunc='mea

# here aggfunc means aggregate function. So 37.03 is the mean age of the
# In pivot table, we were getting there head count
```

In [39]:
```
#We can acheive the same result through other method also, below two mor

df.groupby(['Sex','Pclass']).Age.mean()
```

In [40]:
```
df.groupby(['Sex','Pclass']).Age.mean().unstack()
```

In [41]:
```
#using unstack for a better uncluttered view
```

In [41]:
```
## Data preparation for ML

train_df.info()
```

MAchine LEarning Starts Here !

In [43]:
```
# We will use a baseline model, which is just a basic model to compare w
```

In [44]:
```
# Our implemented model should score more than baseline model, like our
```

In [45]:
```
# first we have to refine our data
```

In [46]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 18 columns):
Survived      1309 non-null int64
Age           1309 non-null float64
Fare          1309 non-null float64
IsMale        1309 non-null int64
Deck_A        1309 non-null int64
Deck_B        1309 non-null int64
Deck_C        1309 non-null int64
Deck_D        1309 non-null int64
Deck_E        1309 non-null int64
Deck_F        1309 non-null int64
Deck_G        1309 non-null int64
Deck_z        1309 non-null int64
Pclass_1      1309 non-null int64
Pclass_2      1309 non-null int64
Pclass_3      1309 non-null int64
Embarked_C    1309 non-null int64
Embarked_Q    1309 non-null int64
Embarked_S    1309 non-null int64
dtypes: float64(2), int64(16)
memory usage: 194.3 KB
```

In [52]:
```
# first we treat embarked because it has least errors
```

```
df[df.Embarked.isnull()]

###commented out because this needs to run only one time and not after t
```

In [54]:
```
df.Embarked.value_counts()
#how many people boarded from each location, S(south hempton), so we wil

###commented out because this needs to run only one time and not after t
```

In [56]:
```
#now checking which embarkement point has more number of survivalsal
#-888 means no survival information
# we'll see south hemptop has most survival as 217

pd.crosstab(df[df.Survived != -888].Survived, df[df.Survived !=-888].Emb

###commented out because this needs to run only one time and not after t
```

In [57]:
```

# first we will extract(select from the data) the embarkement feature wi
# but the missing values people had fare 80, which is clossest to C, so
#we will use fillna function that is specifically used to fill missing v

df.Embarked.fillna('C', inplace=True)

# writting true because this will false will create a parallel df with s


###commented out because this needs to run only one time and not after t
```

In [58]:
```
df.info()
```

```
#we can see that the missing values have been filled
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 18 columns):
Survived      1309 non-null int64
Age           1309 non-null float64
Fare          1309 non-null float64
IsMale        1309 non-null int64
Deck_A        1309 non-null int64
Deck_B        1309 non-null int64
Deck_C        1309 non-null int64
Deck_D        1309 non-null int64
Deck_E        1309 non-null int64
Deck_F        1309 non-null int64
Deck_G        1309 non-null int64
Deck_z        1309 non-null int64
Pclass_1      1309 non-null int64
Pclass_2      1309 non-null int64
Pclass_3      1309 non-null int64
Embarked_C    1309 non-null int64
Embarked_Q    1309 non-null int64
Embarked_S    1309 non-null int64
dtypes: float64(2), int64(16)
memory usage: 194.3 KB
```

In [ ]:
```
df[df.Embarked.isnull()]

# we will see that no value is null now
```

In [ ]:
```
print(df.Embarked)
#all values filled
```

In [ ]:
```
# now will see the fare attribute and correct it

df[df.Fare.isnull()]
```

In [ ]:
```
# the passenger embarked from S and class 3, so we will use median of fa

#median_fare=df.loc[(df.Pclass==3) & (df.Embarked =='S'), 'Fare'].median

# we are extracting only fare attribute

print (median_fare)
```

In [ ]:
```
df.Fare.fillna(median_fare, inplace=True)
```

In [59]:
```
df.info()

# now fare is also treated
```

In [60]:
```
# now we will treat age
# this command will only display 15 rows at a time
```

```
# this command will only display 15 rows at a time
pd.options.display.max_rows=15
```

In [61]:
```
df[df.Age.isnull()]
```

Out[61]:

PassengerId	Survived	Age	Fare	IsMale	Deck_A	Deck_B	Deck_C	Deck_D	Deck_E

In [62]:
```
# we will replace all the age with mean age

df.Age.mean()
```

Out[62]:
```
29.881137667303985
```

In [63]:
```
df.Age.fillna(df.Age.mean(), inplace=True)
```

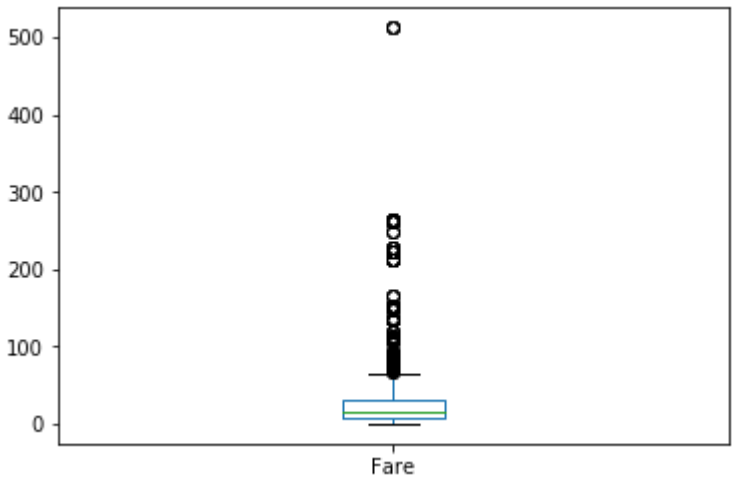
In [64]:
```
df.info()
# we have solved the age problem too
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 18 columns):
Survived      1309 non-null int64
Age           1309 non-null float64
Fare          1309 non-null float64
IsMale        1309 non-null int64
Deck_A        1309 non-null int64
Deck_B        1309 non-null int64
Deck_C        1309 non-null int64
Deck_D        1309 non-null int64
Deck_E        1309 non-null int64
Deck_F        1309 non-null int64
Deck_G        1309 non-null int64
Deck_z        1309 non-null int64
Pclass_1      1309 non-null int64
Pclass_2      1309 non-null int64
Pclass_3      1309 non-null int64
Embarked_C    1309 non-null int64
Embarked_Q    1309 non-null int64
Embarked_S    1309 non-null int64
dtypes: float64(2), int64(16)
memory usage: 194.3 KB
```

In [65]:
```
## OUTLIERS

# now we will handle outliers, these are extreme values that tend to aff
```

In [66]:
```
df.Fare.plot(kind='box');
```

```
#we can see that one point is significantly above others, an outlier. We
```



In [67]:
```
#binning.. the values are put in different bin depending on the range
```

In [68]:
```
pd.qcut(df.Fare,4)

# we canb see the ranges in the catogries section of the output
```

Out[68]:
```
PassengerId
1         (-0.001, 7.896]
2       (31.275, 512.329]
3         (7.896, 14.454]
4       (31.275, 512.329]
5         (7.896, 14.454]
6         (7.896, 14.454]
7       (31.275, 512.329]
                ...
1303    (31.275, 512.329]
1304      (-0.001, 7.896]
1305      (7.896, 14.454]
1306    (31.275, 512.329]
1307      (-0.001, 7.896]
1308      (7.896, 14.454]
1309     (14.454, 31.275]
Name: Fare, Length: 1309, dtype: category
Categories (4, interval[float64]): [(-0.001, 7.896] < (7.896, 14.454]
< (14.454, 31.275] < (31.275, 512.329]]
```

In [69]:
```
pd.qcut(df.Fare, 4, labels=['very_low','low','high','very_high'])
```

```
# this is called descritization.. converting numerical to categorial fea
```

Out[69]:
```
PassengerId
1         very_low
2        very_high
3              low
4        very_high
5              low
6              low
7        very_high
           ...
1303     very_high
1304      very_low
1305           low
1306     very_high
1307      very_low
1308           low
1309          high
Name: Fare, Length: 1309, dtype: category
Categories (4, object): [very_low < low < high < very_high]
```

In [70]:
```
# now we are creating a new feature into our df dataset as Fare_Bin

df['Fare_Bin']=pd.qcut(df.Fare, 4, labels=['very_low','low','high','very
```

In [71]:
```
df.info()

#fare_bin created
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 19 columns):
Survived      1309 non-null int64
Age           1309 non-null float64
Fare          1309 non-null float64
IsMale        1309 non-null int64
Deck_A        1309 non-null int64
Deck_B        1309 non-null int64
Deck_C        1309 non-null int64
Deck_D        1309 non-null int64
Deck_E        1309 non-null int64
Deck_F        1309 non-null int64
Deck_G        1309 non-null int64
Deck_z        1309 non-null int64
Pclass_1      1309 non-null int64
Pclass_2      1309 non-null int64
Pclass_3      1309 non-null int64
Embarked_C    1309 non-null int64
Embarked_Q    1309 non-null int64
Embarked_S    1309 non-null int64
Fare_Bin      1309 non-null category
dtypes: category(1), float64(2), int64(16)
memory usage: 195.8 KB
```

In [72]:
```
## Feature Engineering
```

```
# It means making new columns like name, salary etc
```

In [73]:
```
df['AgeState']=nm.where(df['Age']>=18, 'Adult', 'Child')

# add new feature/column, in which is age is above 18 or equal ,then tho
```

In [74]:
```
df['AgeState'].value_counts()
```

Out[74]:
```
Adult    1155
Child     154
Name: AgeState, dtype: int64
```

In [75]:
```
## Feature Cabin
```

In [76]:
```
df.Cabin
```

In [82]:
```
df.Cabin.unique()

# we have nan meaning some were not allocated any cabin, one of them is

###commented out because this needs to run only one time and not after t
```

In [83]:
```
# exploring cabins with T
df.loc[df.Cabin=='T']
#getting rows for which cabin value is T
# ONLY ONE, SO LETS ASSUME IT TO BE NAN
```

In [84]:
```
df.loc[df.Cabin == 'T','Cabin']=nm.NaN
#setting cabin to NaN because it is most probably error as only one pers

###commented out because this needs to run only one time and not after t
```

In [86]:
```
df.Cabin.unique()

###commented out because this needs to run only one time and not after t
```

In [51]:
```
# we are creating a new deck, to put everyone in one NaN into a new deck

def get_deck(cabin):
    return nm.where(pd.notnull(cabin),str(cabin)[0].upper(),'z')
df['Deck']= df['Cabin'].map(lambda x:get_deck(x))
```

In [ ]:
```
df.Deck.value_counts()
```

In [ ]:
```
df.info()
```

In [ ]:
```
## Data Science most algo will no work on strings or char, so we need to
```

In [ ]:
```
## ENCODING
```

In [ ]:
```
# Binary Encoding( converting something like gender to 0 and 1)
```

In [ ]:
```
# for multi encoding technique, we use label encoding

# like passengers fare low, medium and high as 1,2 and 3
```

In [ ]:
```
## ONE HOt ENCODING - used when we the orders dont make sence like not j
```

In [ ]:
```
# Encoding and making a new column/feature named IsMale

# commented out after the data preparation, because this needs to be exe

df['IsMale']=nm.where(df.Sex == 'male',1,0)
```

In [52]:
```
# Now using get_dummies function, we are label encoding all the columns

df=pd.get_dummies(df,columns=['Deck','Pclass','Embarked'])
###commented out because this needs to run only one time and not after t
```

In [ ]:
```
df.info()
```

In [ ]:
```
# now we will remove the columns that are non-numeric
```

In [ ]:
```
df.drop(['Cabin','Name','Ticket','Parch','SibSp','Sex'], axis=1, inplace

# axis=1 so that drop can work on columns

#inplace= True so that the actual data frame is changed and not a copy o

###commented out because this needs to run only one time and not after t
```

In [ ]:
```
#reordering the columns
columns=[column for column in df.columns if column!='Survived']
columns=['Survived']+columns # adding survived columns infront of the li
df=df[columns] #assignin the new rearranged data frame to the original d
```

In [ ]:
```
df.info()
```

In [ ]:
```
## SAVING THE PROCESSED DATASET
```

In [ ]:
```
## APPLYING MACHINE LEARNING
```

In [ ]:
```
df.drop(['AgeState'],axis=1, inplace=True)
```

In [ ]:
```
df.drop(['Fare_Bin'],axis=1, inplace=True)
```

In [ ]:
```
# Now creating test and training datasets from the df dataframe we refir

df.loc[df.Survived != -888].to_csv('train.csv')

#removing survived because in test, we dont need survived column, surviv
columns=[columns for columns in df.columns if columns !='Survived']

df.loc[df.Survived == -888, columns].to_csv('test.csv')
```

In [ ]:
```

```

Start Executing Cells from below after data Preparation

In [16]:
```
import numpy as nm
import pandas as pd
import os

train_df=pd.read_csv('train.csv', index_col='PassengerId')
test_df=pd.read_csv('test.csv', index_col='PassengerId')
```

In [17]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 18 columns):
Survived      891 non-null int64
Age           891 non-null float64
Fare          891 non-null float64
IsMale        891 non-null int64
Deck_A        891 non-null int64
Deck_B        891 non-null int64
Deck_C        891 non-null int64
Deck_D        891 non-null int64
Deck_E        891 non-null int64
Deck_F        891 non-null int64
Deck_G        891 non-null int64
Deck_z        891 non-null int64
Pclass_1      891 non-null int64
Pclass_2      891 non-null int64
Pclass_3      891 non-null int64
Embarked_C    891 non-null int64
Embarked_Q    891 non-null int64
Embarked_S    891 non-null int64
dtypes: float64(2), int64(16)
memory usage: 132.3 KB
```

In [18]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 892 to 1309
Data columns (total 17 columns):
Age           418 non-null float64
Fare          418 non-null float64
IsMale        418 non-null int64
Deck_A        418 non-null int64
Deck_B        418 non-null int64
Deck_C        418 non-null int64
Deck_D        418 non-null int64
Deck_E        418 non-null int64
Deck_F        418 non-null int64
Deck_G        418 non-null int64
Deck_z        418 non-null int64
Pclass_1      418 non-null int64
Pclass_2      418 non-null int64
Pclass_3      418 non-null int64
Embarked_C    418 non-null int64
Embarked_Q    418 non-null int64
Embarked_S    418 non-null int64
dtypes: float64(2), int64(15)
memory usage: 58.8 KB
```

Data Preparation

In [19]:
```
train_df.info
```

```
16          0       0       0       0       1       0
1
17          0       0       0       0       1       0
0
18          0       0       0       0       1       0
1
19          0       0       0       0       1       0
0
20          0       0       0       0       1       0
0
21          0       0       0       0       1       0
1
22          1       0       0       0       0       0
1
23          0       0       0       0       1       0
0
24          0       0       0       0       0       1
0
```

In [74]:
```
train_df.drop(['Fare_Bin','Age_State'], inplace=True)
```

In [21]:
```
X=train_df.loc[:,'Age':].as_matrix().astype('float')

# here we are taking everything and changing to a matrix of float from A
# X is a matrix of float values as input
# X is basically all columns except the output or result or here survive
```

```
/home/nbuser/anaconda2_501/lib/python2.7/site-packages/ipykernel/__mai
n__.py:1: FutureWarning: Method .as_matrix will be removed in a future
version. Use .values instead.
  if __name__ == '__main__':
```

In [22]:
```
y=train_df['Survived'].ravel()

#y is output. ravel will give a 1D array

#Y is our result containing column, y has the info whether someone survi

# when we are training, we send X and y both to train with respect to re
# and in test, y is used to cross check how many of them we got correct
```

In [23]:
```
print X.shape, y.shape
```

```
(891, 17) (891,)
```

In [ ]:
```

```

train test split

In [24]:
```
from sklearn.model_selection import train_test_split
```

```
x_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,rand
print X_train.shape, y_train.shape

print X_test.shape, y_test.shape
```

```
(712, 17) (712,)
(179, 17) (179,)
```

In [25]:
```
# PRINTING AVERAGE SURVIVAL IN TRAINA AND TEST

# this output will be positive cases ie the people who survived
# we want the two results to be nearly same, so that we know that both t

print 'Mean survival in train : {0:.3f}'.format(nm.mean(y_train))

print 'Mean survival in test : {0:.3f}'.format(nm.mean(y_test))
```

```
Mean survival in train : 0.383
Mean survival in test : 0.385
```

Baseline Model

In [26]:
```
# SEE THE BASELINE MODEL IN NEXT NOTEBOOK
```

In [27]:
```
from sklearn.dummy import DummyClassifier
```

In [28]:
```
model_dummy= DummyClassifier(strategy='most_frequent', random_state=0)
```

In [29]:
```
model_dummy.fit(X_train,y_train)

#giving inputs to the dummy algo or the baseline algo
```

Out[29]:
```
DummyClassifier(constant=None, random_state=0, strategy='most_frequen
t')
```

In [30]:
```
print 'Score for baseline model : {0: .2f}'.format(model_dummy.score(X_t

# in this the model will first get a prediction as output from dummy mod
# all these outputs will then be compared to the y_test which the actual

#as the answer is 0.61, this means that 61% of times the baseline models

# so without using ML, we are still getting .61 accuracy just by classif
```

```
Score for baseline model :  0.61
```

In [31]:
```
# now making a performance matrix so that we can compare the scores of b
```

In [32]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_
```

```
# all these are different kinds of performance matrixes
```

In [33]:
```
# accuracy score

print 'accuracy for baseline model : {0: .2f}'.format(accuracy_score(y_t

# this output will give us the accuracy score of the baseline function

# we have sent y_test is the actual result and model_dummy.predict(X_tes
# these both together will be compared to eachother and then accuracy wi
```

```
accuracy for baseline model :  0.61
```

In [34]:
```
# now showing the confusion matrix, the parameters is same for all the p

print 'confusion matrix for baseline model : \n {0}'.format(confusion_ma
```

```
confusion matrix for baseline model :
 [[110   0]
 [ 69   0]]
```

In [ ]:
```

```

In [35]:
```
# precision and recall scores

print 'precision score for baseline model : \n {0}'.format(precision_sco

print 'recall score for baseline model : \n {0}'.format(recall_score(y_t

# the warning is fine. Zero will answer to both
```

```
precision score for baseline model :
 0.0
recall score for baseline model :
 0.0

/home/nbuser/anaconda2_501/lib/python2.7/site-packages/sklearn/metric
s/classification.py:1135: UndefinedMetricWarning: Precision is ill-def
ined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Storing the output ie predicted values on a csv file(for submission or just reference)

In [36]:
```
test_X= test_df.as_matrix().astype('float')
```

```
# test_df is the dataframe for which we dont have answers
# we will use baseline model to predict, so first we are converting it t
```

```
/home/nbuser/anaconda2_501/lib/python2.7/site-packages/ipykernel/__mai
n__.py:1: FutureWarning: Method .as_matrix will be removed in a future
version. Use .values instead.
  if __name__ == '__main__':
```

In [37]:
```
# getting the predictions
# predictions will get the predicted values from dummy algo ie baseline

predictions=model_dummy.predict(test_X)
```

In [38]:
```
#now making a data frame that we will save/submit
# we are using passengerID as index and their predicted value
# so will have a file that will show whether that passenger ID person is

# remember this is prediction from baseline model. It can be wrong

df_submissions=pd.DataFrame({'PassengerID': test_df.index,'Survived': pr
```

In [39]:
```
df_submissions.head()

# showing the dataframe of predicted values that we created
```

Out[39]:

	PassengerID	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

In [40]:
```
df_submissions.to_csv('01_dummy.csv', index=False)

#we are setting index=False so that no other columns is formed separatel

# we can check after executing this line that a new file named 01_dummy.
```

In [41]:
```
# we can also make a function as below to write the outputs of all the d
```

```
# because the steps to write output from an algo is always the same
def get_submission_file(model, filename):
    #converting to matrix
    test_X=test_df.as_matrix().astype('float')
    #make predicitons
    predictions =model.predict(test_X)
    #submission file
    df_submission =pd.DataFrame({'PassengerId': test_df.index, 'Survived
    df_submission.to_csv(filename, index=False)
```

In [42]:
```
get_submission_file(model_dummy, '001_dummy.csv')

# we have just sent the model name and ouput file name to make a new out
```

```
/home/nbuser/anaconda2_501/lib/python2.7/site-packages/ipykernel/__mai
n__.py:7: FutureWarning: Method .as_matrix will be removed in a future
version. Use .values instead.
```

In [ ]:
```
```

Logistic Regression Model

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
#creating a model object

model_lr_1=LogisticRegression(random_state=0)
```

In [45]:
```
#training the model

model_lr_1.fit(X_train, y_train)

# we can see a big message after this is run saying c=1 and various othe

#these are various regularization features, we can change these values t

# we'll explore them after few steps under regularization steps
```

Out[45]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
```

In [46]:
```
# getting the model score on the test data
# we'll get a score higher than the baseline model  so LR is atleast bet
```

```
print 'score for logistic regression v 1 : {0:.2f}'.format(model_lr_1.sc
```

```
score for logistic regression v 1 : 0.81
```

In [47]:
```
print 'accuracy for LR model : {0: .2f}'.format(accuracy_score(y_test, m

print 'confusion matrix for LR model : \n {0}'.format(confusion_matrix(y

print 'precision score for LR model : \n {0}'.format(precision_score(y_t

print 'recall score for LR model : \n {0}'.format(recall_score(y_test,mo


#We can see that everything improved from baseline model
```

```
accuracy for LR model :  0.81
confusion matrix for LR model :
 [[95 15]
 [19 50]]
precision score for LR model :
 0.769230769231
recall score for LR model :
 0.724637681159
```

In [48]:
```
# Model Cofficients

# these are the values on coffecients of the regression line of u=mx+c s

#also called model weights or model parameters, these values define how

model_lr_1.coef_
```

Out[48]:
```
array([[-3.40946106e-02,  1.69659272e-03, -2.35816697e+00,
         2.53681326e-01,  8.83295630e-02, -3.81642345e-01,
         4.15549039e-01,  1.14475654e+00,  4.78454407e-01,
        -1.06125594e-01, -3.45796908e-01,  1.18314023e+00,
         7.50627887e-01, -3.86562083e-01,  6.21674744e-01,
         7.71763571e-01,  1.53767716e-01]])
```

Regularization

In [49]:
```
# LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
#          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
```

```
#          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
#          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
#          verbose=0, warm_start=False)

# Logistic regression has a lot of parameters that we can change and get
# above is the output after we run LR, it shows all the parameters set

# these are basically like Nobs, increasing and decreacing these has a c

# These are also called hyperparameter and techniques to optimize these

# one such Hyperparameter optimization is 'grid search'
```

Grid Search (Hyperparameter Optimization technique)

In [50]:
```
model_lr=LogisticRegression(random_state=0)
```

In [51]:
```
from sklearn.model_selection import GridSearchCV

#using gridsearchcv function for hyperparameter optimization
```

In [52]:
```
parameters = {'C':[1.0,10.0,50,100,1000], 'penalty' : ['l1','l2']}

# we are creatign a parameter dictionary to try during the grid operatic

# so we are trying 1.0, 10.0 are various other numbers for C and similar

clf=GridSearchCV(model_lr,param_grid=parameters, cv=3)

#first we mentioned the algo name on which we will be applying the optim

# param_grid will have all the different parameters that we want to try

#cv=3 means perform 3 fold cross validation

#clf is grid search object

# clf is the object of the hyperparametatized logistic regression

# so clf is like the optimized LR model with best parameters in it

# becasue GridsearchCV will return a LR model with parameters set to the
```

In [53]:
```
clf.fit(X_train, y_train)
```

```
#now here we are passing the training data into the grid search object
#the object when it was created above, we already sent the algo name,so

#so we only have to send the trianing and test data
```

Out[53]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'penalty': ['l1', 'l2'], 'C': [1.0, 10.0, 50, 100,
1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [54]:
```
clf.best_params_

#best_params is a function which will give us the best and optimized val
```

Out[54]:
```
{'C': 1.0, 'penalty': 'l1'}
```

In [55]:
```
print 'best score :{0:.2f}'.format(clf.best_score_)

#no significant difference observed, most advanced algo we get improveme
```

```
best score :0.80
```

In [56]:
```
#evalute model

print 'score for Logistic Regression version : {0:.2f}'.format(clf.score
```

```
score for Logistic Regression version : 0.81
```

In [ ]:
```

```

Feature Normalization

In [57]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [58]:
```
#feature normalization
```

```
scaler=MinMaxScaler()
X_train_scaled=scaler.fit_transform(X_train)

# this line is working in two parts, first part is sending X_train to fi
```

In [59]:
```
X_train_scaled[:,0].min(),X_train_scaled[:,0].max()

# this is givnig the minimum and maximum values of the scaled values
```

Out[59]:
```
(0.0, 1.0)
```

In [60]:
```
#normalization test data
# this is scaling the tst data also
X_test_scaled=scaler.transform(X_test)
```

feature standardization

In [61]:
```
scaler=StandardScaler()

X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [62]:
```
model_lr=LogisticRegression(random_state=0)
parameters={'C':[1.0,10.0,50.0,100.0,1000.0],'penalty':['l1','l2']}

clf=GridSearchCV(model_lr, param_grid=parameters,cv=3)
clf.fit(X_train,y_train)
```

Out[62]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'penalty': ['l1', 'l2'], 'C': [1.0, 10.0, 50.0, 10
0.0, 1000.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [63]:
```
clf.best_score_
```

Out[63]:
```
0.7963483146067416
```

In [64]:
```
#evaluation model
```

```
print 'score for logistic regression - v2 : {0:.2f} '.format(clf.score(X_
# we can see that their is no imporvement with feature standardiaztion

# this happens because standardized features dont have good affect on  L

#but still we apply to see if we get any improvement
```

```
score for logistic regression - v2 : 0.68
```

Model persistence

Our work on LR is done, now we are trying to save the model so that we can directly use this model, hence we are saving the model, this is called model persistence

In [65]:
```
import pickle
```

In [79]:
```
#creating a file and opening it in write mode


model_file_pickle=open('lr_model.pkl','wb')
scaler_file_pickle=open('scaler_model.pkl','wb')

# we need scaler model to save the standardized scalers we created in th
# so we created scaler_model.pkl also

# wb stands for writting in binary mode
```

In [80]:
```

pickle.dump(clf, model_file_pickle)
pickle.dump(scaler, scaler_file_pickle)

# model_file_pickle is the object name of the file

# clf is the object of the hyperparametatized logistic regression

# that is ... clf=GridSearchCV(model_lr,param_grid=parameters, cv=3)

# so clf is like the optimized LR model with best parameters in it

# becasue grid search CV will return a LR model with parameters set to t



# dump function in used to write the model and scalar objects
```

In [81]:
```
model_file_pickle.close()
scaler_file_pickle.close()
```

Loading the persistent file

In [82]:
```
#now opening persisted files as read

model_file_pickle=open('lr_model.pkl','r')
scaler_file_pickle=open('scaler_model.pkl', 'r')

#load files

clf_loaded=pickle.load(model_file_pickle)

scaler_loaded=pickle.load(scaler_file_pickle)

#close files

model_file_pickle.close()
scaler_file_pickle.close()
```

In [83]:
```
clf_loaded
```

Out[83]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'penalty': ['l1', 'l2'], 'C': [1.0, 10.0, 50.0, 10
0.0, 1000.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [84]:
```
scaler_loaded
```

Out[84]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [87]:
```
# transform the test data using loaded scaler object
```

```
x_test_scaled=scaler_loaded.transform(X_test)
#calculate the score using loaded model object

print 'score for persisited logistc regression : {0:.2f}'.format(clf_loa

print 'score for persisited logistc regression non scaled: {0:.2f}'.form

# so this perticular problem does not work well when using a scaled data

# also we can see that we saved the whole model and no need to retrain t

# we are just sending it the test data sets, and no train data set

# this is persisted model, a model which is alread learnt and does not r
```

```
score for persisited logistc regression : 0.68
score for persisited logistc regression non scaled: 0.81
```

In [ ]:

In [ ]:

In [ ]: