ML LAB 5

Explore and implement Logistic Regression by Stochastic Gradient Descent in a given business scenario and comment on its efficiency and performance.

In [1]:
```
#Imports for data analysis, data wrangling and visualization
import pandas as pd
import numpy as np
import random as rand
import seaborn as sns
import matplotlib.pyplot as plt

#Machine learning imports
from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import Perceptron, SGDClassifier
```

In [2]:
```
#Loading the data
train_df = pd.read_csv('C:/Users/user/Downloads/titanicpredictions-main/titanicpr
test_df = pd.read_csv('C:/Users/user/Downloads/titanicpredictions-main/titanicpre
combine = [train_df,test_df]
```

In [3]:
```
#Checking the column names
print(train_df.columns.values)

#Categorical variables - Survived, Sex, Embarked, Pclass
#Numerical variables - Age, Fare, SibSP, Parch
#Ticket is a mix of numeric and alphanumeric data types and Cabin is Alphanumeric
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

In [4]:
```
#Checking the training DF
train_df.tail(15)
#Cabin and Age contain null values
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
876	877	0	3	Gustafsson, Mr. Alfred Ossian	male	20.0	0	0	7534	9.8458
877	878	0	3	Petroff, Mr. Nedelio	male	19.0	0	0	349212	7.8958
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230433	26.0000
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

In [5]:
```
#Checking the test DF
test_df.tail(15)
#Cabin and Age contain null values
```

Out[5]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
403	1295	1	Carrau, Mr. Jose Pedro	male	17.0	0	0	113059	47.1000	NaN
404	1296	1	Frauenthal, Mr. Isaac Gerald	male	43.0	1	0	17765	27.7208	D40
405	1297	2	Nourney, Mr. Alfred (Baron von Drachstedt")"	male	20.0	0	0	SC/PARIS 2166	13.8625	D38
406	1298	2	Ware, Mr. William Jeffery	male	23.0	1	0	28666	10.5000	NaN
407	1299	1	Widener, Mr. George Dunton	male	50.0	1	1	113503	211.5000	C80
408	1300	3	Riordan, Miss. Johanna Hannah""	female	NaN	0	0	334915	7.7208	NaN
409	1301	3	Peacock, Miss. Treasteall	female	3.0	1	1	SOTON/O.Q. 3101315	13.7750	NaN
410	1302	3	Naughton, Miss. Hannah	female	NaN	0	0	365237	7.7500	NaN
411	1303	1	Minahan, Mrs. William Edward (Lillian E Thorpe)	female	37.0	1	0	19928	90.0000	C78
412	1304	3	Henriksson, Miss. Jenny Lovisa	female	28.0	0	0	347086	7.7750	NaN
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN

In [6]:
```
#Checking the data types of the features (7 features are integers or floats (6 in
train_df.info()
print('-'*40)
test_df.info()
```

```
<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
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

In [7]:
```
#Checking the numerical distribution of numerical features across the samples
train_df.describe()
#891 samples of 2224 that were aboard
#Around 38% survived, compared to 32% of the actual rate
```

Out[7]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [8]:
```
train_df.describe(include=['O'])
# Names are unique
# 65% are male (577/891)
# A lot of the cabins are shared (147 cabins), also duplicate values
# 3 possible embarked values, S is the most popular (644/889)
# Ticket feature has a high ratio of duplicate values (681/891)
```

Out[8]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Montvila, Rev. Juozas	male	347082	G6	S
freq	1	577	7	4	644

Assumptions based on the data analysis so far:

1) Correlating: We need to know how each of the features correlate with survival.

2) Completing: We need to complete the age and embarked features as they are probably related to survival.

3) Correcting: Ticket (high ratio of duplicates), Cabin (highly incomplete with many missing values) and passangerID (does not contribute to survival) should be dropped

4) Creating: We may need to create a new feature called 'Family' based on Parch and SibSp to get total count of family members. We may want to manipulate the name feature to extract title as a new feature. We may want to group age into bands as this turns the numerical feature into an ordinal categorical feature. We may also want to create a fare range to see if it correlates with survival.

5) Classifying: Based on the problem description we can check for some assumptions -> Woman (sex=female), Children and Upper Class Passengers (pclass=1) are more likely to have survived

In [9]:
```
#To confirm some of our assumptions we can analyze feature correlation by pivoting
train_df[['Pclass','Survived']].groupby(['Pclass'], as_index = False).mean().sort_
```

Out[9]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

In [10]:
```
train_df[['Sex','Survived']].groupby(['Sex'], as_index = False).mean().sort_value
```

Out[10]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

In [11]:
```
train_df[['SibSp','Survived']].groupby(['SibSp'], as_index = False).mean().sort_v
```

Out[11]:

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

In [12]:
```
train_df[['Parch','Survived']].groupby(['Parch'], as_index = False).mean().sort_v
```

Out[12]:

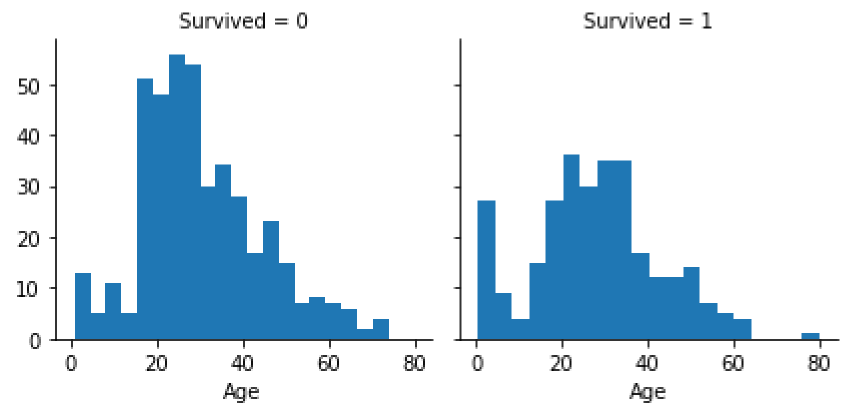
	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

Analyze by visualizing data

1) Correlating Numerical Features

In [13]:
```
graph = sns.FacetGrid(train_df,col = 'Survived')
graph.map(plt.hist, 'Age', bins = 20)
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x1f0e4ccf4f0>



Observations:

```
1) Babies (age<4) had a high survival rate
2) Oldest passanger survived (age=80)
3) A lot with passangers age 15 to 25 did not survived
4) Most passangers are in the 15-35 age range
```

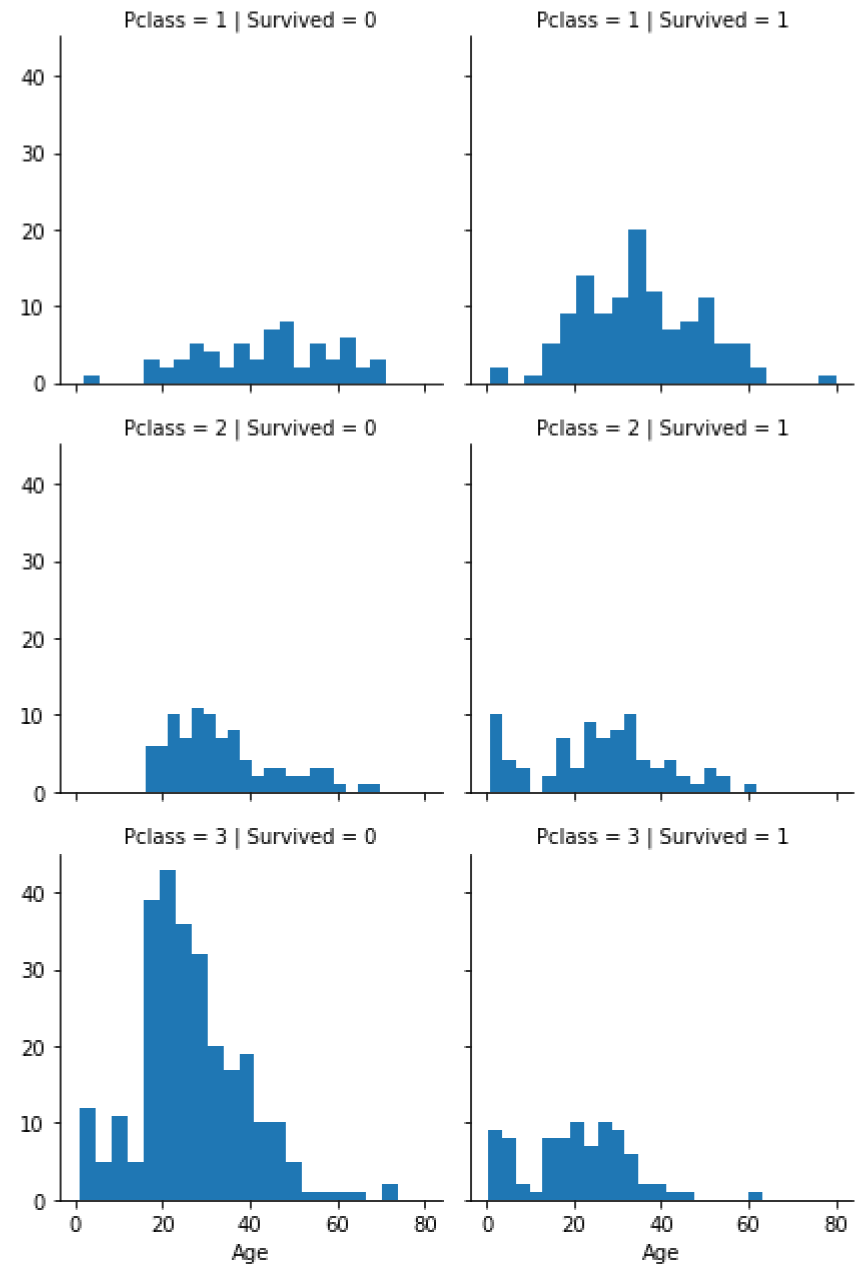
Decisions

```
1) We should consider age in our model training
2) We should complete the age feature for null values
3) We should band age groups to perform a better analysis
```

2) Correlating Numerical and Ordinal Features

In [14]:
```
graph = sns.FacetGrid(train_df, col = 'Survived',row='Pclass')
graph.map(plt.hist, 'Age', bins = 20)
graph.add_legend()
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x1f0e5520670>



Observations:

```
    1) Pclass=3 had the higher number of passangers but most of them didn
t survive
    2) Babies in pclass = 2 and 3 mostly survived so it further qualifies
our assumption about it
    3) Most passengers in pclass = 1 survived
    4) Pclass varies in terms of age distribution
```

Decisions:

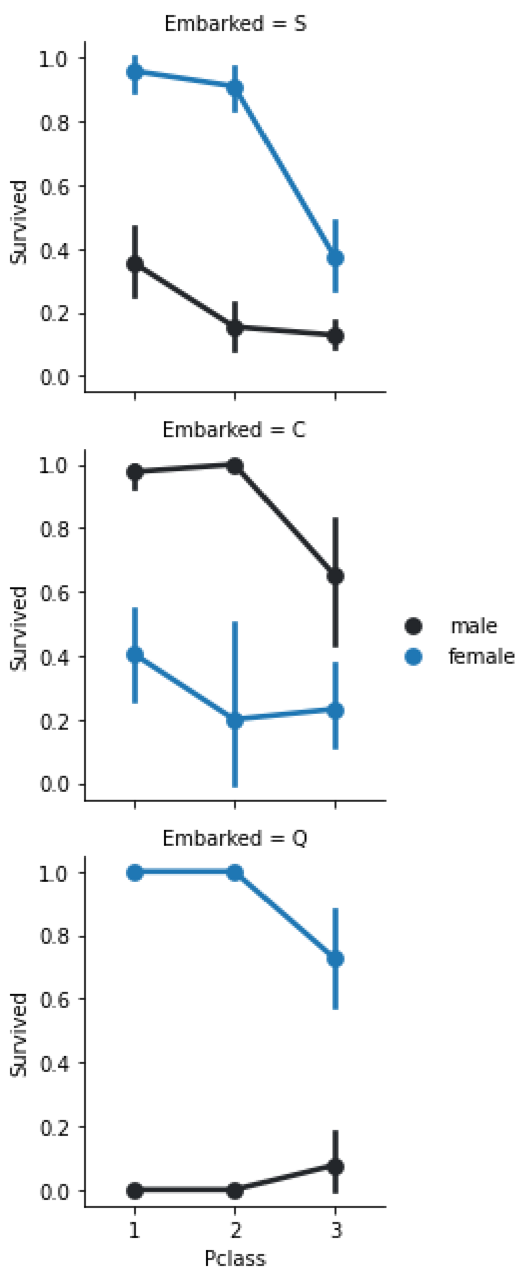
```
    1) Consider pclass for training
```

3) Correlating Categorical Features

In [15]:
```
graph = sns.FacetGrid(train_df, row = 'Embarked')
graph.map(sns.pointplot, 'Pclass','Survived','Sex')
graph.add_legend()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:643: UserWarning:
Using the pointplot function without specifying `order` is likely to produce an
incorrect plot.
  warnings.warn(warning)
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:648: UserWarning:
Using the pointplot function without specifying `hue_order` is likely to produc
e an incorrect plot.
  warnings.warn(warning)
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1f0e54cdc10>



Observations:

```
    1) Female passengers had a much better survival rate
    2) Exception is embarked = C where males had a higher survival rate
    3) Males had a higher survival rate in pclass=3 when compared do pcla
ss=2 for C and Q ports
```

Decisions:

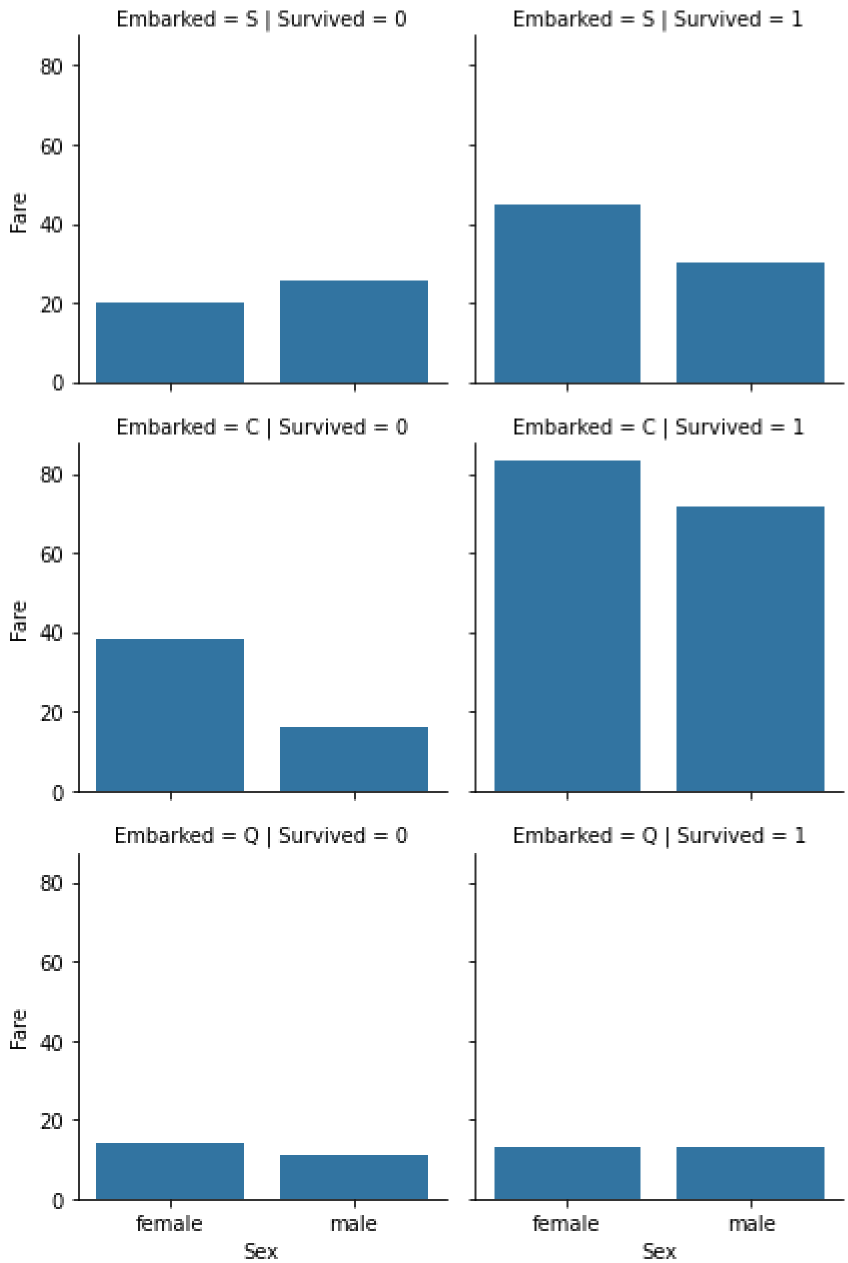
```
    1) Add sex feature to the model training
    2) Complete and add embarked feature to the model training
```

4) Correlating Categorical and Numerical Features

In [16]:
```
graph = sns.FacetGrid(train_df, col = 'Survived',row='Embarked')
graph.map(sns.barplot, 'Sex','Fare', ci = None)
graph.add_legend()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:643: UserWarning:
Using the barplot function without specifying `order` is likely to produce an i
ncorrect plot.
  warnings.warn(warning)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1f0e5c35f70>



Observations:

```
1) Higher fare rates had higher survival rates
2) Port of embarkation correlates with the survival rates
```

Decisions:

```
1) We should band the fare rates and consider them in the model
```

Wrangle the data

In [17]:
```
#dropping unnecessary features to speed up the training
IDs = test_df['PassengerId']
train_df.drop(['Ticket','Cabin','PassengerId'], inplace=True, axis = 1)
test_df.drop(['Ticket','Cabin','PassengerId'], inplace=True, axis = 1)
combine = [train_df,test_df]
```

In [18]:
```
#creating new feature from existing - 'name' - extracting the characters of the s
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract('([A-Za-z]+)\.', expand = False)

pd.crosstab(train_df['Title'], train_df['Sex'])
```

Out[18]:

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

In [19]:
```
#we can group the uncommon titles on a category named other
for dataset in combine:
    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')

train_df.groupby('Title').mean()
```

Out[19]:

	Survived	Pclass	Age	SibSp	Parch	Fare
Title						
Master	0.575000	2.625000	4.574167	2.300000	1.375000	34.703125
Miss	0.702703	2.291892	21.845638	0.702703	0.540541	43.800092
Mr	0.156673	2.410058	32.368090	0.288201	0.152805	24.441560
Mrs	0.793651	1.992063	35.788991	0.690476	0.825397	45.330290
Other	0.347826	1.347826	45.545455	0.347826	0.086957	37.169748

In [20]:
```
#Then we can convert the categorical titles to ordinal
title_dict = {'Mr': 1,
              'Miss': 2,
              'Mrs': 3,
              'Master': 4,
              'Other': 5 }
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_dict)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```

Out[20]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

In [21]:
```
#now we can also drop the name feature
train_df.drop(['Name'], axis = 1 , inplace = True)
train_df.head()
```

Out[21]:

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

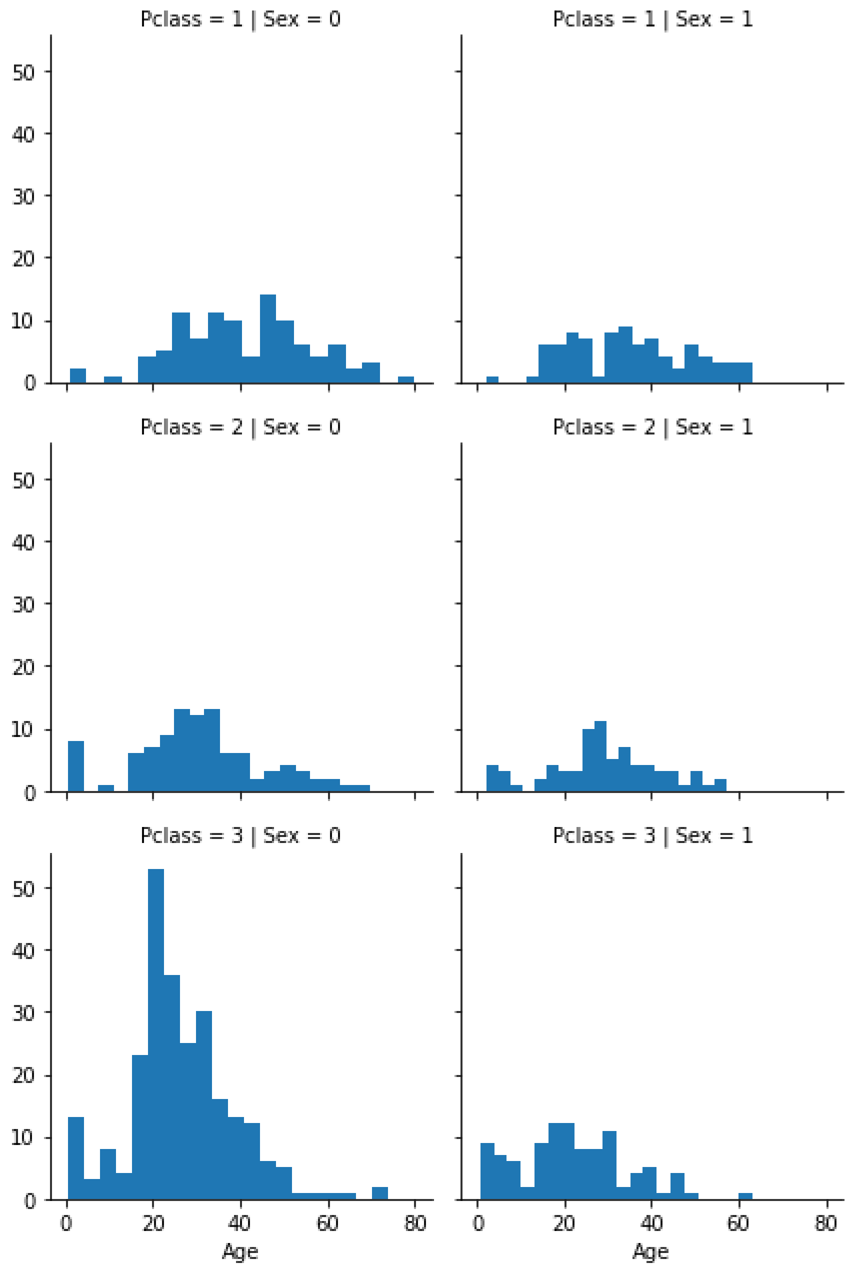
In [22]:
```
#converting the categorical feature (sex) into ordinal
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map({'female':1,
                                        'male':0})
train_df.head()
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

In [23]:
```
#now we should estimate or complete the feature with missing or null values, we'l
#we will guess the missing values for age by using other correlated features like
graph = sns.FacetGrid(train_df, row = 'Pclass', col = 'Sex')
graph.map(plt.hist, 'Age', bins = 20)
graph.add_legend()
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x1f0e5bf5bb0>



In [24]:
```
#Lets prepare an empty array to contain the guessed age values for all the 6 pcla
guess_ages = np.zeros((2,3))
guess_ages
```

Out[24]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [25]:
```
#Now we iterate to get the median of each combination of pclass and sex, and use

for dataset in combine:
    for i in range (0,2):
        for j in range(0,3):
           guess_df = dataset[(dataset['Sex'] == i) & \
                                  (dataset['Pclass'] == j+1)]['Age'].dropna()
           age_guess = guess_df.median()
           guess_ages[i,j] = int(age_guess/.5 +.5)*.5 #convert random age float t

    for i in range (0,2):
        for j in range (0,3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.P
                    'Age'] = guess_ages[i,j]

    dataset['Age'] = dataset['Age'].astype(int)

train_df.head()
```

Out[25]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22	1	0	7.2500	S	1
1	1	1	1	38	1	0	71.2833	C	3
2	1	3	1	26	0	0	7.9250	S	2
3	1	1	1	35	1	0	53.1000	S	3
4	0	3	0	35	0	0	8.0500	S	1

In [26]:
```
#Checking to see how the ages split in 5 different bands (in absolut numbers not (
train_df['AgeBand'] = pd.cut(train_df['Age'],5)
train_df[['AgeBand','Survived']].groupby(['AgeBand'], as_index = False).mean().so
```

Out[26]:

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

In [27]:
```
#Attributing a number to each of the agebands
for dataset in combine:
    dataset.loc[dataset['Age'] <= 16,'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32),'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48),'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64),'Age'] = 3
    dataset.loc[dataset['Age'] > 64,'Age'] = 4
train_df.head()
```

Out[27]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	AgeBand
0	0	3	0	1	1	0	7.2500	S	1	(16.0, 32.0]
1	1	1	1	2	1	0	71.2833	C	3	(32.0, 48.0]
2	1	3	1	1	0	0	7.9250	S	2	(16.0, 32.0]
3	1	1	1	2	1	0	53.1000	S	3	(32.0, 48.0]
4	0	3	0	2	0	0	8.0500	S	1	(32.0, 48.0]

In [28]:
```
train_df.drop(columns='AgeBand', inplace =True, axis = 1)
combine = [train_df,test_df]
train_df
```

Out[28]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	1	1	0	7.2500	S	1
1	1	1	1	2	1	0	71.2833	C	3
2	1	3	1	1	0	0	7.9250	S	2
3	1	1	1	2	1	0	53.1000	S	3
4	0	3	0	2	0	0	8.0500	S	1
...	...	...	...	...	...	...	...	...	...
886	0	2	0	1	0	0	13.0000	S	5
887	1	1	1	1	0	0	30.0000	S	2
888	0	3	1	1	1	2	23.4500	S	2
889	1	1	0	1	0	0	30.0000	C	1
890	0	3	0	1	0	0	7.7500	Q	1

891 rows × 9 columns

In [29]:
```
#Here we are aggregating the number of partners (parch) with simblings (SibSp) an
for dataset in combine:
    dataset['FamilySize'] = dataset['Parch'] + dataset['SibSp'] + 1
    dataset.drop(columns=['SibSp','Parch'],inplace=True,axis=1)
train_df[['FamilySize','Survived']].groupby(['FamilySize'], as_index=False).mean(
```

Out[29]:

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

In [30]:
```
#Creating a feature 'isAlone' will help us to correlate the fact of being alone w
for dataset in combine:
    dataset['isAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'isAlone'] = 1
    dataset.drop('FamilySize',inplace=True,axis=1) #we can also drop family size 
train_df[['isAlone','Survived']].groupby(['isAlone']).mean().sort_values(by=['Sur
#Its possible to see that the alone people had a higher survival mean rate
```

Out[30]:

	Survived
isAlone	
0	0.505650
1	0.303538

In [31]:
```
#We can create a new feature multiplying age and the pclass, so in theory the Low
for dataset in combine:
    dataset['AgeClass'] = dataset['Age'] * dataset['Pclass']

train_df[['Age','Pclass','AgeClass']].head(10)
```

Out[31]:

	Age	Pclass	AgeClass
0	1	3	3
1	2	1	2
2	1	3	3
3	2	1	2
4	2	3	6
5	1	3	3
6	3	1	3
7	0	3	0
8	1	3	3
9	0	2	0

In [32]:
```
#Checking the embarked feature we can see that S is the most common port, so we'l
train_df.Embarked.describe()
```

Out[32]:
```
count      889
unique       3
top          S
freq       644
Name: Embarked, dtype: object
```

In [33]:
```
freq_port = 'S'
for dataset in combine:
    dataset['Embarked'].fillna(freq_port,inplace=True)
train_df[['Embarked','Survived']].groupby(['Embarked']).mean().sort_values(by=['S
#Its possible to see that the S port had the lower mean survival rate and C had th
```

Out[33]:

Embarked	Survived
S	0.339009
Q	0.389610
C	0.553571

In [34]:
```
#Mapping the ports
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map({'S':0,
    'C':1,
    'Q':2}).astype(int)
train_df.head(10)
```

Out[34]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6
5	0	3	0	1	8.4583	2	1	1	3
6	0	1	0	3	51.8625	0	1	1	3
7	0	3	0	0	21.0750	0	4	0	0
8	1	3	1	1	11.1333	0	3	0	3
9	1	2	1	0	30.0708	1	3	0	0

In [35]:
```
#Complete fare for the single missing value on the test DF using the mode
test_df['Fare'].fillna(test_df['Fare'].dropna().median(),inplace=True)
test_df
```

Out[35]:

	Pclass	Name	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	3	Kelly, Mr. James	0	2	7.8292	2	1	1	6
1	3	Wilkes, Mrs. James (Ellen Needs)	1	2	7.0000	0	3	0	6
2	2	Myles, Mr. Thomas Francis	0	3	9.6875	2	1	1	6
3	3	Wirz, Mr. Albert	0	1	8.6625	0	1	1	3
4	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	1	1	12.2875	0	3	0	3
...	...	...	...	...	...	...	...	...	...
413	3	Spector, Mr. Woolf	0	1	8.0500	0	1	1	3
414	1	Oliva y Ocana, Dona. Fermina	1	2	108.9000	1	5	1	2
415	3	Saether, Mr. Simon Sivertsen	0	2	7.2500	0	1	1	6
416	3	Ware, Mr. Frederick	0	1	8.0500	0	1	1	3
417	3	Peter, Master. Michael J	0	1	22.3583	1	4	0	3

418 rows × 9 columns

In [36]:
```
#We can now create the bands for the fare, but as we did for the age we have to c
train_df['FareBand'] = pd.qcut(train_df['Fare'],4) #qcut divides into 4 quantiles
train_df[['FareBand','Survived']].groupby(['FareBand'],as_index=False).mean().sor
#We can see that the higher the band the higher the survival mean rate
```

Out[36]:

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

In [37]:
```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] =
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']  =
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df.drop(['FareBand'], axis=1, inplace=True)
combine = [train_df, test_df]

train_df
```

Out[37]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
...	...	...	...	...	...	...	...	...	...
886	0	2	0	1	1	0	5	1	2
887	1	1	1	1	2	0	2	1	1
888	0	3	1	1	2	0	2	0	3
889	1	1	0	1	2	1	1	1	1
890	0	3	0	1	0	2	1	1	3

891 rows × 9 columns

In [38]:
```
#And now both our datasets are ready
test_df.drop(columns=['Name'],inplace=True,axis=1)
test_df.head(100)
```

Out[38]:

	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	3	0	2	0	2	1	1	6
1	3	1	2	0	0	3	0	6
2	2	0	3	1	2	1	1	6
3	3	0	1	1	0	1	1	3
4	3	1	1	1	0	3	0	3
...	...	...	...	...	...	...	...	...
95	3	0	1	0	0	1	1	3
96	1	1	4	3	0	3	0	4
97	3	0	1	1	0	1	1	3
98	3	1	1	0	0	2	1	3
99	3	0	2	1	0	1	1	6

100 rows × 8 columns

In [39]:
```
X_train = train_df.drop('Survived',axis=1)
Y_train = train_df['Survived']
X_test = test_df.copy()
X_train.shape,Y_train.shape,X_test.shape
```

Out[39]: ((891, 8), (891,), (418, 8))

In [40]:
```
#Logistic Regression
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(acc_log,'%')
```

```
81.37 %
```

In [41]:
```
coeff = pd.DataFrame(train_df.columns.delete(0))
coeff.columns = ['Feature']
coeff['Correlation'] = pd.Series(logreg.coef_[0])
coeff.sort_values(by = 'Correlation', ascending = False)
```

Out[41]:

	Feature	Correlation
1	Sex	2.201057
5	Title	0.406027
4	Embarked	0.276628
6	isAlone	0.185986
7	AgeClass	-0.050260
3	Fare	-0.071665
2	Age	-0.469638
0	Pclass	-1.200309

In [42]:
```
#Stochastic Gradient Descent
sgd = SGDClassifier()
sgd.fit(X_train,Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train,Y_train)*100,2)
print(acc_sgd,'%')
```

```
69.81 %
```

In [43]:
```
models = pd.DataFrame({'Model':
['SGD','Logistic Regression'],
'Scores':
[acc_sgd,acc_log]}
)
models = models.sort_values(by='Scores',ascending=False).reset_index(drop=True)
models
```

Out[43]:

	Model	Scores
0	Logistic Regression	81.37
1	SGD	69.81

In [ ]: