CS5785 HW 1 -- Applied Machine Learning

Student: Shangjing Tang(st787), Katherine Wu(Kw634)

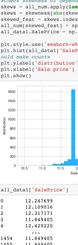
Due Date: September 13th

PROGRAMMING EXERCISES

Part I. The Housing Prices

1. Join the House Prices - Advanced Regression Techniques competition on Kaggle. Download the training and test data.
2. Give3 examples of continuous and categorical features in the dataset; choose one feature of each type and plot the histogram to illustrate the distribution.

In [455]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os
import warnings
import seaborn as sns
from scipy.stats import skew
from matplotlib import pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import OneHotEncoder
from sklearn import datasets
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, make_scorer

pd.options.mode.chained_assignment = None  # default='warn'
warnings.filterwarnings('ignore')
%matplotlib inline
workingDirectory = os.getcwd()
trainData = pd.read_csv(workingDirectory+'/train.csv')
testData = pd.read_csv(workingDirectory+'/test.csv')
all_data = pd.concat((trainData.iloc[:, 1:testData], testData.iloc[:, 1:]))#merge both trainig

con_feat1 = all_data['SalePrice']
con_feat2 = all_data['1stFlrSF']
con_feat3 = all_data['LotArea']

cat_feat4 = all_data['SaleCondition']
cat_feat5 = all_data['MSZoning']
cat_feat6 = all_data['LotShape']

plt.style.use('seaborn-whitegrid')
plt.hist(con_feat1, bins=10,facecolor = '#2ab0ff', edgecolor='#169acf', linewidth=0.5)  # density=False would make cou
nts
plt.ylabel('distribution')
plt.xlabel('Sale price')

plt.show()

plt.style.use('seaborn-whitegrid')
plt.hist(con_feat2, bins=10,facecolor = '#2ab0ff', edgecolor='#169acf', linewidth=0.5)  # density=False would make cou
nts
plt.ylabel('distribution')
plt.xlabel('sale condition')

plt.show()
```



Answer: The distribution for continuous feature in the graph is the sale price from training dataset. This feature has a left skewed histogram. Each bar in the histogram represents how many occurance in each price range. The second graph has the categorical data of "sales condition". In this graph the data has 6 different categories and the heigh of the bar represents the number of occurence of each category.

1. Pre-process your data, explain your pre-processing steps, and the reasons why you need them. (Hint: data pre-processing steps can include but are not restricted to: dealing with missing values, normalizing numerical values, dealing with categorical values etc.)

In this pre-crocessing stage, we mainly perform 4 steps.

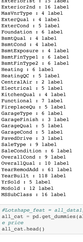
- Drop columns where they have greater than 75% of NA values. When we perform data cleaning process, we found that there are several columns such as "Alley" has mainly NA values which are not significant to our study.
- For remaining columns that don't have greater than 75% of NA values but do have small amount of NA values in the column. We will then fill mean values for those numeric features who has NA, and mode for those categorical values who has NA. Check if there is any additional NA values in the data set
- We then divided dataset into categorical features and continuous features. Also manually exclude some categorical features who has numeric values from the continuous feature sets and extend those numeric categorical feature into the categorical feature sets.
- Finally, we will check for the skewness of the continuous features. When we plot the histogram for "Sale price". We found that the graph is left skewed, which may mean there are a bunch of outliers in the dataset. So we take a log function to normalize those feature.

In [456]:
```
#Drop columns with NA value greater than 75%
all_data1 = all_data.dropna(axis='columns', thresh=0.25*len(all_data))
salesprice = all_data['SalePrice']

#Fill the remaining data with mean in numerical, mode in categorical: https://stackoverflow.com/questions/32617811/imp
utation-of-missing-values-for-categorical-variables-in-pandas
all_data1.fillna(all_data1.apply(lambda x: x.value_counts().index[0]), inplace = True)

#Check if there are still na in the data set
def find_nan_cols(data):
    cols = []
    lens = len(data)
    for col in data.columns:
        if data[col].count() < lens:
            cols.append(col)
    return cols
len(find_nan_cols(all_data1))==0
```

Out[457]:
```
True
```

In [458]:
```
# distinguish data between categorical and continuous
col_numeric = list(all_data1.select_dtypes(exclude = 'object').columns)

for cols in ['OverallCond','OverallQual','YearRemodAdd','YearBuilt','TrBsId','Modold','MSSubClass']:
    col_numeric.remove(cols)

col_category= list(all_data1.select_dtypes(include = 'object').columns)
col_category.extend(['OverallCond','OverallQual','YearRemodAdd','YearBuilt','TrBsId','Modold','MSSubClass'])
all_num = all_data1[col_numeric]
all_cat = all_data1[col_category]
all_num = all_num.drop('SalePrice',axis=1)
```

In [459]:
```
#Check skewness of dependent variable and independent variables, take log to normalize data
skewd = all_num.apply(lambda x: skew(x))
skews = skewness[abs(skewness) > 0.5]
skewed_feat = skews.index
all_num[skewed_feat] = np.log1p(all_num[skewed_feat])
all_data1.SalePrice = np.log1p(all_data1.SalePrice)

plt.style.use('seaborn-whitegrid')
plt.hist(all_data1['SalePrice'], bins=30,facecolor = '#2ab0ff', edgecolor='#169acf', linewidth=0.5)  # density=False w
ould make counts
plt.ylabel('distribution')
plt.xlabel('Sale price')
plt.show()
```



In [460]:
```
all_data1['SalePrice']
```

Out[460]:
```
0       12.247699
1       12.109016
2       12.317171
3       11.849405
4       12.429220
       ...
1454    11.849405
1455    11.849405
1456    11.849405
1457    11.849405
1458    11.849405
Name: SalePrice, Length: 2919, dtype: float64
```

1. One common method of pre-processing categorical features is to use a one-hot encoding (OHE). Suppose that we start with a categorical feature x_j taking three possible values: $x_j \in \{R, G, B\}$. A one-hot encoding of this feature replaces x_j with three new features: x_j^R, x_j^G, x_j^B. Each feature contains a binary value of 0 or 1, depending on the value taken by x_j. For example, if $x_j = G$, then $x_j^G = 1$ and $x_j^R = x_j^B = 0$.

   Give some examples of features that you think should use a one-hot encoding and explain why. Convert at least one feature to a one-hot encoding (you can use your own implementation, or in pandas or scikit-learn) and visualize the results by plotting feature histograms of the original feature and its new one-hot encoding.

Answer:The features that can be using one-hot encoding are 'LotShape', 'Sales Conditions', 'LotConfig', and 'MSZoning', etc. In this example, I will pick all categorical features to perform the one-hot encoding.

In [461]:
```
#Check unique features within each variable
for col in all_data1[col_category].columns:
    print(col,':',len(all_data1[col_category][col].unique()),'label')

MSZoning : 5 label
Street : 2 label
LotShape : 4 label
LandContour : 4 label
Utilities : 2 label
LotConfig : 5 label
LandSlope : 3 label
Neighborhood : 25 label
Condition1 : 9 label
Condition2 : 8 label
BldgType : 5 label
HouseStyle : 8 label
RoofStyle : 6 label
RoofMatl : 8 label
Exterior1st : 15 label
Exterior2nd : 16 label
MasVnrType : 4 label
ExterQual : 4 label
ExterCond : 5 label
Foundation : 6 label
BsmtQual : 4 label
BsmtCond : 4 label
BsmtExposure : 4 label
BsmtFinType1 : 6 label
BsmtFinType2 : 6 label
Heating : 6 label
HeatingQC : 5 label
CentralAir : 2 label
Electrical : 5 label
KitchenQual : 4 label
Functional : 7 label
FireplaceQu : 5 label
GarageType : 6 label
GarageFinish : 3 label
GarageQual : 5 label
GarageCond : 5 label
PavedDrive : 3 label
SaleType : 9 label
SaleCondition : 6 label
OverallCond : 9 label
OverallQual : 10 label
YearRemodAdd : 61 label
YearBuilt : 118 label
TrBsId : 5 label
Modold : 12 label
MSSubClass : 16 label
```

In [462]:
```
#Exchange_feat = all_data[col_category][['LotShape']
all_cat = pd.get_dummies(all_cat.astype(str), drop_first = True) #one hot encoding using panda methods without the sal
e price
all_cat.head()
```

Out[462]:

	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave	LotShape_IR2	LotShape_IR3	LotShape_Reg	LandContour_HLS	LandContour_Low
0	0	0	1	0	1	0	0	1	0	0
1	0	0	1	0	1	0	0	1	0	0
2	0	0	1	0	1	1	0	0	0	0
3	0	0	1	0	1	1	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0

5 rows × 424 columns

1. Using ordinary least squares(OLS),try to predict house prices on this dataset.Choose the features (or combinations of features) you would like to use or ignore, provided you justify your choice. Evaluate your predictions on the training set using the MSE and the R2 score. For this question, you need to implement OLS from scratch without using any external libraries or packages.

In [463]:
```
# Join both numerical and categorical data without sale price
join_data = pd.concat([all_num,all_cat],axis=1)
#Training_df = join_data.iloc[:
train_row = trainData.shape[0]#returns row number
training = join_data.iloc[:train_row] # we only want cleansed data with the original training data size
x_test = join_data.iloc[train_row:]
y = all_data1[:train_row].SalePrice
x_train, x_valid,y_train,y_valid = train_test_split(training, y, test_size = 0.3, random_state = 0)
```

Based on the course lecture, for any parameter vector θ, the fitted value is $X\theta$, and thus the term in the exponent is the function showing as below:

$$S(\theta) = (y - X\theta)^T + (y - X\theta)$$

This function is a measure of goodness of fit. If Sθ is small, then y and the fitted value will be calcoses located to yield a smaller MSE. The minimum value of S(θ) is obtained when

$$X^T X\theta = X^T y$$

Therefore we can get the OLS equation of

$$\theta^* = (X^T X)^{-1} X^T y$$

In [464]:
```
#define the theta function.
def theta(x_train, y_train):
    return np.linalg.pinv(x_train.T.dot(x_train)).dot(x_train.T).dot(y_train)
```

$$f(x) = \theta^T X$$

Since the x_train matrix is a singular square matrix that cannot be inverted, we compute the pseudo-inverse of the matrix. Once we have the theta value, we can then calculate the MSE.

In [465]:
```
theta1 = theta(x_train, y_train)
mse = mean_squared_error(x_valid.dot(theta1),y_valid)
r2 = r2_score(x_valid.dot(theta1), y_valid)
print('theta_train, x_train, y_train).reshape((5,1))
y_pred = x_train.dot(theta1)
```

In [466]:
```
mse, r2
```

Out[466]:
```
(0.016468773486476957, 0.5362345578749224)
```

1. Train your model using all of the training data (all data points,but not necessarily all the features), and test it using the testing data. Submit your results to Kaggle.

In [467]:
```
y_pred_test = x_test.dot(theta(x_train, y_train))
y_pred_test = np.expm1(y_pred_test)
```

In [468]:
```
y_pred_test
```

Out[468]:
```
0       131764.329268
1       156695.208818
2       193009.441106
3       201673.866413
4       182524.631302
              ...
1454     84250.612875
1455     84654.611393
1456    165958.051713
1457    111546.692319
1458    185505.593657
Length: 1459, dtype: float64
```

In [469]:
```
submission = testData.copy()

submission['SalePrice'] = y_pred_test
submission = submission[['Id', 'SalePrice']]
submission.to_csv('./submission.csv',index=False)
```

In [470]:
```
submission
```

Out[470]:

	Id	SalePrice
0	1461	131764.329268
1	1462	156695.208818
2	1463	193009.441106
3	1464	201673.866413
4	1465	182524.631302
...	...	...
1454	2915	86250.612875
1455	2916	84654.611393
1456	2917	165958.051713
1457	2918	111546.692319
1458	2919	185505.593657

1459 rows × 2 columns

Below is our submission score in Kaggle. Our overall score is 0.62918

Markdown Monster icon Since our pitures don't show when convert Jupyter HTML to PDF, we will include both pictures of Kaggle scores in the last page of the pdf file.

In [ ]:
```

```

In [ ]:
```

```

Problem 2 Titanic

1. Join the Titanic: Machine Learning From Disaster competition on Kaggle. Download and pre-process the data.
2. Implement logistic regression (it's OK to use sklearn or similar software packages), try to predict whether a passenger survived the disaster with your model. Choose the features (or combinations of features) you would like to use or ignore, provided you justify your choice.
3. Train your classifier using all of the training data, and test it using the testing data. Submit your results to Kaggle.

In [65]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
train_data_path = './titanic/train.csv'
test_data_path = './titanic/test.csv'
train_data_pd = pd.read_csv(train_data_path)
test_data_pd = pd.read_csv(test_data_path)
all_data_pd = pd.concat([train_data_pd, test_data_pd]).reset_index(drop=True)
```

In [36]:
```
print(train_data_pd.shape, test_data_pd.shape, all_data_pd.shape)
```

```
(891, 12) (418, 11) (1309, 12)
```

In [37]:
```
#To see if data has been fetched and if it can be shown
train_data_pd.head(10)
```

Out[37]:

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

In [66]:
```
# First drop the name column since it should be unrelated to survival
train_data_pd.drop("Name", axis = 1, inplace = True)
test_data_pd.drop("Name", axis = 1, inplace = True)
```

In [67]:
```
# Observe the features by info()
print(train_data_pd.info())
# Find and show the null values :
cols_with_miss = []
for col in all_data_pd.columns.tolist():
    num_missing = all_data_pd[col].isnull().sum()
    if num_missing > 0:
        cols_with_miss.append(col)

print(cols_with_miss)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
 9   Cabin        204 non-null    object
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
None
['Survived', 'Age', 'Fare', 'Cabin', 'Embarked']
```

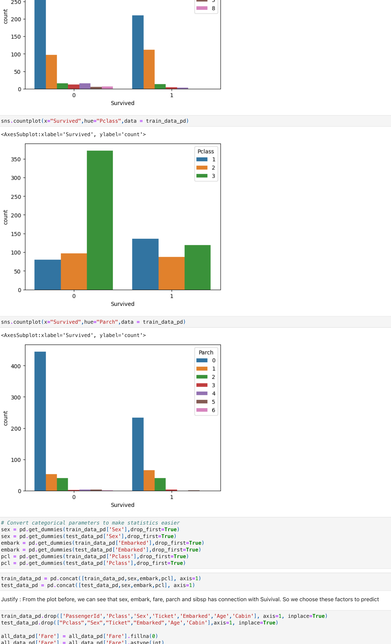
In [68]:
```
#To see the connection between survival and ages
corr_bet = all_data_pd.corr().abs().unstack().sort_values(kind="quicksort",ascending=False).reset_index()
corr_bet[corr_bet["level_0"] == 'Age']
```

Out[68]:

	level_0	level_1	0
5	Age	Age	1.000000
9	Age	Pclass	0.408106
18	Age	SibSp	0.243699
22	Age	Fare	0.178740
25	Age	Parch	0.150917
30	Age	Survived	0.077221
42	Age	Passengerid	0.028814

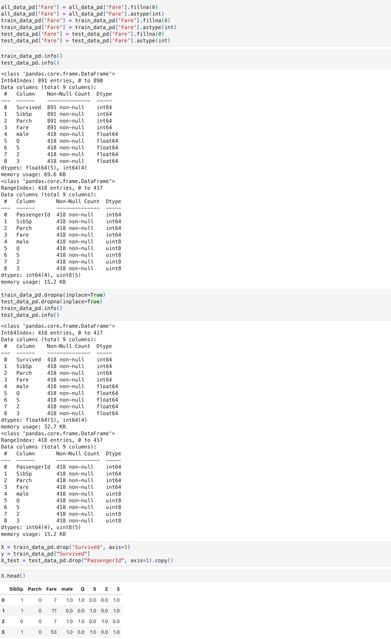
In [69]:
```
sns.countplot(x="Survived",hue="SibSp",data = train_data_pd)
```

Out[69]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



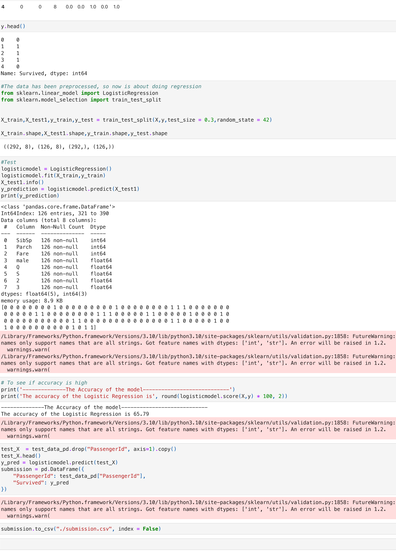
In [71]:
```
sns.countplot(x="Survived",hue="Pclass",data = train_data_pd)
```

Out[71]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [72]:
```
sns.countplot(x="Survived",hue="Parch",data = train_data_pd)
```

Out[72]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [73]:
```
# Convert categorical parameters to make statistics easier
sex = pd.get_dummies(train_data_pd['Sex'],drop_first=True)
sex = pd.get_dummies(test_data_pd['Sex'],drop_first=True)
embark = pd.get_dummies(train_data_pd['Embarked'],drop_first=True)
embark = pd.get_dummies(test_data_pd['Embarked'],drop_first=True)
pcl = pd.get_dummies(train_data_pd['Pclass'],drop_first=True)
pcl = pd.get_dummies(test_data_pd['Pclass'],drop_first=True)
```

In [74]:
```
train_data_pd = pd.concat([train_data_pd,sex,embark,pcl], axis=1)
test_data_pd = pd.concat([test_data_pd,sex,embark,pcl], axis=1)
```

Justify : From the plot before, we can see that sex, embark, fare, parch and sibsp has connection with Suivival. So we choose these factors to predict

In [75]:
```
train_data_pd.drop(['PassengerId','Pclass','Sex','Ticket','Embarked','Age','Cabin'], axis=1, inplace=True)
test_data_pd.drop(['Pclass","Sex","Ticket","Embarked", 'Age','Cabin',axis=1, inplace=True)

all_data_pd['Fare'] = all_data_pd['Fare'].fillna(0)
all_data_pd['Fare'] = all_data_pd['Fare'].astype(int)
train_data_pd['Fare'] = train_data_pd['Fare'].fillna(0)
train_data_pd['Fare'] = train_data_pd['Fare'].astype(int)
test_data_pd['Fare'] = test_data_pd['Fare'].fillna(0)
test_data_pd['Fare'] = test_data_pd['Fare'].astype(int)
```

In [77]:
```
train_data_pd.info()
test_data_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   SibSp     891 non-null    int64
 2   Parch     891 non-null    int64
 3   Fare      891 non-null    int64
 4   male      418 non-null    float64
 5   Q         418 non-null    float64
 6   S         418 non-null    float64
 7   2         418 non-null    float64
 8   3         418 non-null    float64
dtypes: float64(5), int64(4)
memory usage: 69.6 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   PassengerId  418 non-null    int64
 1   SibSp     418 non-null    int64
 2   Parch     418 non-null    int64
 3   Fare      418 non-null    int64
 4   male      418 non-null    float64
 5   Q         418 non-null    uint8
 6   S         418 non-null    uint8
 7   2         418 non-null    uint8
 8   3         418 non-null    uint8
dtypes: int64(4), uint8(5)
memory usage: 15.2 KB
```

In [78]:
```
train_data_pd.dropna(inplace=True)
test_data_pd.dropna(inplace=True)
train_data_pd.info()
test_data_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 0 to 417
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  418 non-null    int64
 1   SibSp     418 non-null    int64
 2   Parch     418 non-null    int64
 3   Fare      418 non-null    int64
 4   male      418 non-null    float64
 5   Q         418 non-null    float64
 6   S         418 non-null    float64
 7   2         418 non-null    float64
 8   3         418 non-null    float64
dtypes: float64(5), int64(4)
memory usage: 32.7 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   PassengerId  418 non-null    int64
 1   SibSp     418 non-null    int64
 2   Parch     418 non-null    int64
 3   Fare      418 non-null    int64
 4   male      418 non-null    float64
 5   Q         418 non-null    uint8
 6   S         418 non-null    uint8
 7   2         418 non-null    uint8
 8   3         418 non-null    uint8
dtypes: int64(4), uint8(5)
memory usage: 15.2 KB
```

In [79]:
```
X = train_data_pd.drop('Survived', axis=1)
y = train_data_pd["Survived"]
X_test = test_data_pd.drop("PassengerId", axis=1).copy()
```

In [80]:
```
X.head()
```

Out[80]:

	SibSp	Parch	Fare	male	Q	S	2	3
0	1	0	7	1.0	1.0	0.0	0.0	1.0
1	1	0	71	0.0	0.0	1.0	0.0	1.0
2	0	0	7	1.0	1.0	0.0	1.0	0.0
3	1	0	53	1.0	0.0	1.0	0.0	1.0
4	0	0	8	0.0	0.0	1.0	0.0	1.0

In [81]:
```
y.head()
```

Out[81]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [106]:
```
#The data has been preprocessed, so now is about doing regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split


X_train,X_test1,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state = 42)

X_train.shape,X_test1.shape,y_train.shape,y_test.shape
```

Out[106]:
```
((292, 8), (126, 8), (292,), (126,))
```

In [107]:
```
#Test
logisticmodel = LogisticRegression()
logisticmodel.fit(X_train,y_train)
X_test1.info()
y_prediction = logisticmodel.predict(X_test1)
print(y_prediction)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 126 entries, 321 to 390
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   SibSp     126 non-null    int64
 1   Parch     126 non-null    int64
 2   Fare      126 non-null    int64
 3   male      126 non-null    float64
 4   Q         126 non-null    float64
 5   S         126 non-null    float64
 6   2         126 non-null    float64
 7   3         126 non-null    float64
dtypes: float64(5), int64(3)
memory usage: 8.9 KB
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1]
```

In [108]:
```
#To see if accuracy is high
print('--------------The Accuracy of the model----------------------------')
print('The accuracy of the Logistic Regression is', round(logisticmodel.score(X_y) * 100, 2))
```

```
--------------The Accuracy of the model----------------------------
The accuracy of the Logistic Regression is 65.79
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

In [115]:
```
test_X  = test_data_pd.drop("PassengerId", axis=1).copy()
test_X.head()
y_pred = logisticmodel.predict(test_X)
submission = pd.DataFrame({
    "PassengerId": test_data_pd["PassengerId"],
    "Survived": y_pred
})
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

In [116]:
```
submission.to_csv("./submission.csv", index = False)
```

In [ ]:

YOUR RECENT SUBMISSION

submission.csv

Submitted by katttwu · Submitted 13 hours ago

Score: 0.62918

Jump to your leaderboard position

1 submissions for **katttwu**

Sort by Select...

All Successful Selected

Submission and Description	Public Score
submission.csv 13 hours ago by katttwu add submission details	0.62918

2 submissions for **katttwu**

Sort by Select...

All Successful Selected

Submission and Description	Public Score
housesubmission.csv just now by katttwu add submission details	0.28966