Mir Mohammed Zain 20BDS0211 - Assignment: 3

Q1 - Download the dataset

Q2 - load the dataset into the tool

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('Housing.csv')
```

```
data
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airc
0	13300000	7420	4	2	3	yes	no	no	no	
1	12250000	8960	4	4	4	yes	no	no	no	
2	12250000	9960	3	2	2	yes	no	yes	no	
3	12215000	7500	4	2	2	yes	no	yes	no	
4	11410000	7420	4	1	2	yes	yes	yes	no	
				...	...	...	...	...	...	
				1	1	yes	no	yes	no	
541	1767150	2400	3	1	1	no	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	no	
543	1750000	2910	3	1	1	no	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	no	

545 rows × 12 columns

Saved successfully!

```
data.info()
```

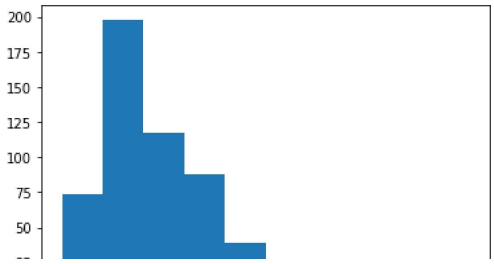
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 541 entries, 0 to 544
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             541 non-null    int64
 1   area              541 non-null    int64
 2   bedrooms          541 non-null    int64
 3   bathrooms         541 non-null    int64
 4   stories           541 non-null    int64
 5   mainroad          541 non-null    int32
 6   guestroom         541 non-null    int32
 7   basement          541 non-null    int32
 8   hotwaterheating   541 non-null    int32
 9   airconditioning   541 non-null    int32
 10  parking           541 non-null    int64
 11  furnishingstatus  541 non-null    int32
dtypes: int32(6), int64(6)
memory usage: 42.3 KB
```

q3 - data visualization

Q3 - Univariate visualization

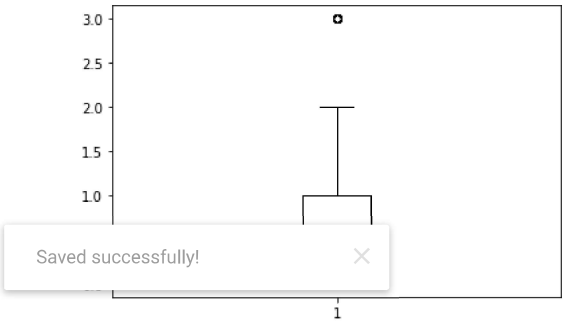
```
plt.hist(data['area'])
```

```
(array([ 73., 198., 117.,  88.,  39.,  15.,   8.,   5.,   0.,   2.]),
 array([ 1650.,  3105.,  4560.,  6015.,  7470.,  8925., 10380., 11835.,
        13290., 14745., 16200.]),
 <a list of 10 Patch objects>)
```




```
plt.boxplot(data['parking'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1d2a23f9790>,
  <matplotlib.lines.Line2D at 0x1d2a23f9af0>],
 'caps': [<matplotlib.lines.Line2D at 0x1d2a23f9e50>,
  <matplotlib.lines.Line2D at 0x1d2a24071f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1d2a23f9430>],
 'medians': [<matplotlib.lines.Line2D at 0x1d2a2407550>],
 'fliers': [<matplotlib.lines.Line2D at 0x1d2a2407850>],
 'means': []}
```

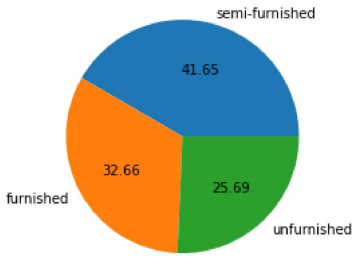



```
furnished = data['furnishingstatus'].value_counts()
furnished
```

```
semi-furnished    227
unfurnished       178
furnished         140
Name: furnishingstatus, dtype: int64
```

```
plt.pie(furnished,autopct = '%.2f',labels=['semi-furnished','furnished','unfurnished'])
```

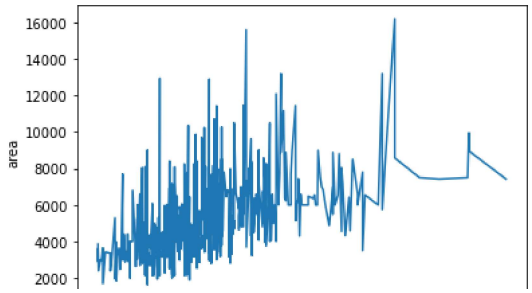
```
([<matplotlib.patches.Wedge at 0x1d2a2468100>,
  <matplotlib.patches.Wedge at 0x1d2a24685e0>,
  <matplotlib.patches.Wedge at 0x1d2a2468c70>],
 [Text(0.28521128309432414, 1.0623815340995388, 'semi-furnished'),
  Text(-0.9645476294288756, -0.5288174264934321, 'furnished'),
  Text(0.7608233961924185, -0.7944480850289933, 'unfurnished')],
 [Text(0.15556979077872224, 0.5794808367815666, '41.65'),
  Text(-0.5261168887793867, -0.2884458689964175, '32.66'),
  Text(0.4149945797413191, -0.43333531910672357, '25.69')])
```



Q3 - Bivariate visualization

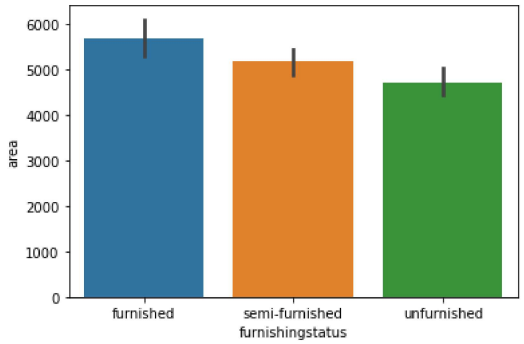
```
plt.plot(data['price'],data['area'])
plt.xlabel('price')
plt.ylabel('area')
```

```
Text(0, 0.5, 'area')
```




```
sns.barplot(data['furnishingstatus'],data['area'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2a24e83a0>
```



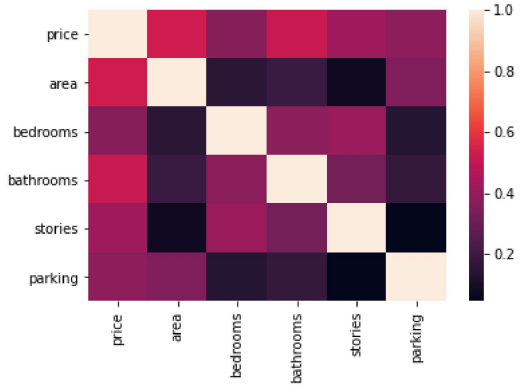
Q3 - Multivariate analysis

Saved successfully!

```
heat = data.corr()
```

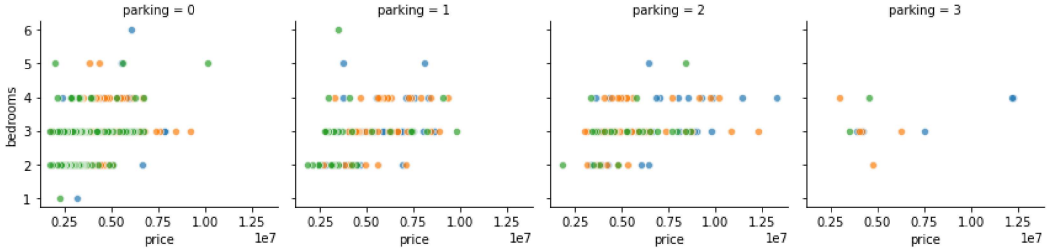
```
sns.heatmap(heat)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2a2565520>
```




```
ploting = sns.FacetGrid(data,col='parking',hue='furnishingstatus')
ploting.map(sns.scatterplot,'price','bedrooms',alpha=.7)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2a26270d0>
```



Q4 - Perform descriptive statistics on the dataset

```
print(data.describe())
```

```
print('median')
print(data.median())

print('mode')
print(data.mode())

print(data.kurt())

print('printing quartile')
quantile=data.quantile(q=[0.75,0.25])
print(quantile)
print(quantile.iloc[0])
print(data.quantile(0.5))
print(quantile.iloc[1])
```

```
            parking
count  545.000000
mean     0.693578
std      0.861586
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      3.000000
median
price        4340000.0
area            4600.0
bedrooms           3.0
bathrooms          1.0
stories            2.0
parking            0.0
dtype: float64
mode
      price    area  bedrooms  bathrooms  stories mainroad guestroom basement  \
                                      1.0      2.0      yes        no       no
                                      NaN      NaN      NaN       NaN      NaN

  hotwaterheating airconditioning  parking furnishingstatus
0              no              no      0.0   semi-furnished
1             NaN             NaN      NaN              NaN
price        1.960130
area         2.751480
bedrooms     0.728323
bathrooms    2.164856
stories      0.679404
parking     -0.573063
dtype: float64
printing quartile
          price    area  bedrooms  bathrooms  stories  parking
0.75  5740000.0  6360.0       3.0        2.0      2.0      1.0
0.25  3430000.0  3600.0       2.0        1.0      1.0      0.0
price        5740000.0
area            6360.0
bedrooms           3.0
bathrooms          2.0
stories            2.0
parking            1.0
Name: 0.75, dtype: float64
price        4340000.0
area            4600.0
bedrooms           3.0
bathrooms          1.0
stories            2.0
parking            0.0
Name: 0.5, dtype: float64
price        3430000.0
area            3600.0
bedrooms           2.0
bathrooms          1.0
stories            1.0
parking            0.0
Name: 0.25, dtype: float64
```

Saved successfully!

Q5 - check for missing values and deal with them

```
data.isnull().sum()
```

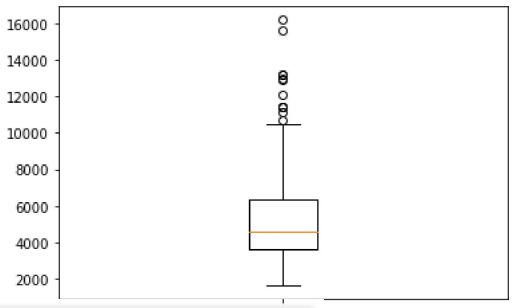
```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
```

```
basement            0
hotwaterheating     0
airconditioning     0
parking             0
furnishingstatus    0
dtype: int64
```

Q6 - find the outliers and deal with them

```
plt.boxplot(data['area'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1d2a27eb040>,
  <matplotlib.lines.Line2D at 0x1d2a27eb3a0>],
 'caps': [<matplotlib.lines.Line2D at 0x1d2a27eb700>,
  <matplotlib.lines.Line2D at 0x1d2a27eba60>],
 'boxes': [<matplotlib.lines.Line2D at 0x1d2a27deca0>],
 'medians': [<matplotlib.lines.Line2D at 0x1d2a27ebdc0>],
 'fliers': [<matplotlib.lines.Line2D at 0x1d2a27f7100>],
 'means': []}
```



Saved successfully!

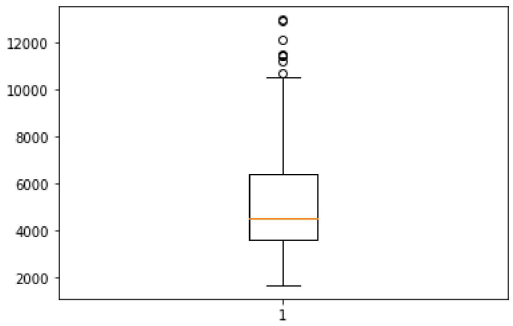
```
perc = data.area.quantile(0.995)
perc
```

```
13200.0
```

```
data = data[data.area<perc]
```

```
plt.boxplot(data['area'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1d2a248bbe0>,
  <matplotlib.lines.Line2D at 0x1d2a23ca040>],
 'caps': [<matplotlib.lines.Line2D at 0x1d2a23ca9d0>,
  <matplotlib.lines.Line2D at 0x1d2a253fc40>],
 'boxes': [<matplotlib.lines.Line2D at 0x1d2a2429bb0>],
 'medians': [<matplotlib.lines.Line2D at 0x1d2a2587f10>],
 'fliers': [<matplotlib.lines.Line2D at 0x1d2a2592700>],
 'means': []}
```




```
data
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

#**Q7 - Check for categorical columns and perform encoding**

Q7 - Check for categorical columns and perform encoding

543	1750000	2910	3	1	1	no	no	no

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
data['mainroad'] = le.fit_transform(data.mainroad)
data['guestroom'] = le.fit_transform(data.guestroom)
data['basement'] = le.fit_transform(data.basement)
data['furnishingstatus'] = le.fit_transform(data.furnishingstatus)
data['airconditioning'] = le.fit_transform(data.airconditioning)
data['hotwaterheating'] = le.fit_transform(data.hotwaterheating)
```

```
<ipython-input-23-a82b821baf85>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['mainroad'] = le.fit_transform(data.mainroad)
<ipython-input-23-a82b821baf85>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['guestroom'] = le.fit_transform(data.guestroom)
<ipython-input-23-a82b821baf85>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['basement'] = le.fit_transform(data.basement)
<ipython-input-23-a82b821baf85>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['furnishingstatus'] = le.fit_transform(data.furnishingstatus)
<ipython-input-23-a82b821baf85>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['airconditioning'] = le.fit_transform(data.airconditioning)
<ipython-input-23-a82b821baf85>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['hotwaterheating'] = le.fit_transform(data.hotwaterheating)
```

Saved successfully!

```
data
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airc
0	13300000	7420	4	2	3	1	0	0	0	
1	12250000	8960	4	4	4	1	0	0	0	
2	12250000	9960	3	2	2	1	0	1	0	
3	12215000	7500	4	2	2	1	0	1	0	

Q8 - split the data into dependent and independent variables

540 1820000 3000 2 1 1 1 0 1 0

```
x = data.drop(columns = 'price')
y = data['price']
```

542 1750000 3620 2 1 1 1 0 0 0

```
x
```

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditionin
0	7420	4	2	3	1	0	0	0	
1	8960	4	4	4	1	0	0	0	
2	9960	3	2	2	1	0	1	0	
3	7500	4	2	2	1	0	1	0	
4	7420	4	1	2	1	1	1	0	
...	...	...	...	...	...	...	...	...	.
540	3000	2	1	1	1	0	1	0	
541	2400	3	1	1	0	0	0	0	
542	3620	2	1	1	1	0	0	0	
543	2910	3	1	1	0	0	0	0	
544	3850	3	1	2	1	0	0	0	

541 rows × 11 columns

Saved successfully!

```
y
```

```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 541, dtype: int64
```

Q9 - scale the independent varaible

```
from sklearn.preprocessing import MinMaxScaler

scale = MinMaxScaler()

x_scaled = scale.fit_transform(x)
x_scaled
```

```
array([[0.51089074, 0.6       , 0.33333333, ..., 1.        , 0.66666667,
        0.        ],
       [0.64724633, 0.6       , 1.        , ..., 1.        , 1.        ,
        0.        ],
       [0.73578891, 0.4       , 0.33333333, ..., 0.        , 0.66666667,
        0.5       ],
       ...,
       [0.1744289 , 0.2       , 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.11156366, 0.4       , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.1947937 , 0.4       , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

```
X = pd.DataFrame(x_scaled,columns = x.columns)
X
```

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditi
0	0.510891	0.6	0.333333	0.666667	1.0	0.0	0.0	0.0	
1	0.647246	0.6	1.000000	1.000000	1.0	0.0	0.0	0.0	
2	0.735789	0.4	0.333333	0.333333	1.0	0.0	1.0	0.0	
3	0.517974	0.6	0.333333	0.333333	1.0	0.0	1.0	0.0	
4	0.510891	0.6	0.000000	0.333333	1.0	1.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	
536	0.119532	0.2	0.000000	0.000000	1.0	0.0	1.0	0.0	
537	0.066407	0.4	0.000000	0.000000	0.0	0.0	0.0	0.0	
538	0.174429	0.2	0.000000	0.000000	1.0	0.0	0.0	0.0	
539	0.111564	0.4	0.000000	0.000000	0.0	0.0	0.0	0.0	
540	0.194794	0.4	0.000000	0.333333	1.0	0.0	0.0	0.0	

541 rows × 11 columns

Q10 - split the data into testing and training

```
from sklearn.model_selection import train_test_split
```

Saved successfully!

```
n_test_split(X,y,test_size =0.2,random_state = 0)
```

Q11 - Build the model

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression


log = LogisticRegression()
df=DecisionTreeClassifier(criterion='entropy',random_state=0)
knn=KNeighborsClassifier()
lr=LinearRegression()
```

Q12 - Train the model

```
log.fit(x_train,y_train)
df.fit(x_train,y_train)
knn.fit(x_train,y_train)
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

Q13 - test the model

```
pred1 = lr.predict(x_test)
pred2 = log.predict(x_test)
pred3 = knn.predict(x_test)
pred4 = df.predict(x_test)


y_test
```

```
74     6650000
396    3500000
382    3570000
369    3675000
144    5600000
        ...
303    4200000
```

```
233    4620000
541    1767150
500    2660000
12     9310000
Name: price, Length: 109, dtype: int64
```

Q14 - Measure the performance using the metrics

```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
```

```
print(accuracy_score(y_test,pred2))
print(accuracy_score(y_test,pred3))
print(accuracy_score(y_test,pred4))
```

```
0.009174311926605505
0.0
0.027522935779816515
```

```
acc1=r2_score(pred1,y_test)
print(acc1)
acc2=r2_score(pred2,y_test)
print(acc2)
acc3=r2_score(pred3,y_test)
print(acc3)
acc4=r2_score(pred4,y_test)
print(acc4)
```

```
0.6070770253224096
 0.8988004647878108
```



```
print(confusion_matrix(y_test,pred2))
print(confusion_matrix(y_test,pred3))
print(confusion_matrix(y_test,pred4))
```

```
[[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 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]]
```

```
print(classification_report(y_test,pred2))
print(classification_report(y_test,pred3))
print(classification_report(y_test,pred4))
```

4970000	0.00	0.00	0.00	0
5005000	0.00	0.00	0.00	1
5033000	0.00	0.00	0.00	0
5040000	0.00	0.00	0.00	1
5145000	0.00	0.00	0.00	1
5215000	0.00	0.00	0.00	1
5250000	0.00	0.00	0.00	1
5320000	0.00	0.00	0.00	1
5383000	0.00	0.00	0.00	0
5460000	0.00	0.00	0.00	2
5495000	0.00	0.00	0.00	1
5530000	0.00	0.00	0.00	0
5600000	0.00	0.00	0.00	2
5652500	0.00	0.00	0.00	0
5740000	0.00	0.00	0.00	2
5866000	0.00	0.00	0.00	1
5950000	0.00	0.00	0.00	1
6020000	0.00	0.00	0.00	0
6090000	0.00	0.00	0.00	2
6107500	0.00	0.00	0.00	0
6160000	0.00	0.00	0.00	2
6195000	0.00	0.00	0.00	0
6265000	0.00	0.00	0.00	1
6300000	0.00	0.00	0.00	1
6440000	0.00	0.00	0.00	0
6475000	0.00	0.00	0.00	1
6615000	0.00	0.00	0.00	1
6650000	0.00	0.00	0.00	3
6790000	0.00	0.00	0.00	1
6895000	0.00	0.00	0.00	0
7420000	0.00	0.00	0.00	0
7455000	0.00	0.00	0.00	1
7490000	0.00	0.00	0.00	1
7560000	0.00	0.00	0.00	0
7875000	0.00	0.00	0.00	0
7910000	0.00	0.00	0.00	1
7980000	0.00	0.00	0.00	0
		00	0.00	0
		00	0.00	0

Saved successfully!

conclusion - best model is 'Logistic Regression'