Titanic-Survival-Prediction/Titanic Predication.ipynb

In [1]:

```
#Importing All Required Libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from warnings import filterwarnings
filterwarnings(action='ignore')
```

In [9]:

```
#Loading Datasets
pd.set_option('display.max_columns',10,'display.width',1000)
train = pd.read_csv('https://raw.githubusercontent.com/training-ml/Files/main/titanic_train
test = pd.read_csv('https://raw.githubusercontent.com/amberkakkar01/Titanic-Survival-Predic
train.head()
```

Out[9]:

	Unnamed: 0	PassengerId	Survived	Pclass	Name	...	Parch	Ticket	Fare	Cabin
0	0	1	0	3	Braund, Mr. Owen Harris	...	0	A/5 21171	7.2500	NaN
1	1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	...	0	PC 17599	71.2833	C85
2	2	3	1	3	Heikkinen, Miss. Laina	...	0	STON/O2. 3101282	7.9250	NaN
3	3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	...	0	113803	53.1000	C123
4	4	5	0	3	Allen, Mr. William Henry	...	0	373450	8.0500	NaN

5 rows × 13 columns

In [10]:

```
#Display shape
train.shape
```

Out[10]:

```
(891, 13)
```

In [11]:

```
test.shape
```

Out[11]:

```
(418, 11)
```

In [12]:

```
#Checking for Null values
train.isnull().sum()
```

Out[12]:

```
Unnamed: 0       0
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [13]:

```
test.isnull().sum()
```

Out[13]:

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [14]:

```
#Description of dataset
train.describe(include="all")
```

Out[14]:

	Unnamed: 0	PassengerId	Survived	Pclass	Name	...	Parch	Ticket	
count	891.000000	891.000000	891.000000	891.000000	891	...	891.000000	891	891
unique	NaN	NaN	NaN	NaN	891	...	NaN	681	
top	NaN	NaN	NaN	NaN	Braund, Mr. Owen Harris	...	NaN	347082	
freq	NaN	NaN	NaN	NaN	1	...	NaN	7	
mean	445.000000	446.000000	0.383838	2.308642	NaN	...	0.381594	NaN	32
std	257.353842	257.353842	0.486592	0.836071	NaN	...	0.806057	NaN	49
min	0.000000	1.000000	0.000000	1.000000	NaN	...	0.000000	NaN	0
25%	222.500000	223.500000	0.000000	2.000000	NaN	...	0.000000	NaN	7
50%	445.000000	446.000000	0.000000	3.000000	NaN	...	0.000000	NaN	14
75%	667.500000	668.500000	1.000000	3.000000	NaN	...	0.000000	NaN	31
max	890.000000	891.000000	1.000000	3.000000	NaN	...	6.000000	NaN	512

11 rows × 13 columns

In [15]:

```
train.groupby('Survived').mean()
```

Out[15]:

Survived	Unnamed: 0	PassengerId	Pclass	Age	SibSp	Parch	Fare
0	446.016393	447.016393	2.531876	30.626179	0.553734	0.329690	22.117887
1	443.368421	444.368421	1.950292	28.343690	0.473684	0.464912	48.395408

In [16]:

```
train.corr()
```

Out[16]:

	Unnamed: 0	PassengerId	Survived	Pclass	Age	SibSp	Parch	
Unnamed: 0	1.000000	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.(
PassengerId	1.000000	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.(
Survived	-0.005007	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.2
Pclass	-0.035144	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.5
Age	0.036847	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.(
SibSp	-0.057527	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.1
Parch	-0.001652	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.2
Fare	0.012658	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.(

In [17]:

```
male_ind = len(train[train['Sex'] == 'male'])
print("No of Males in Titanic:",male_ind)
```

```
No of Males in Titanic: 577
```

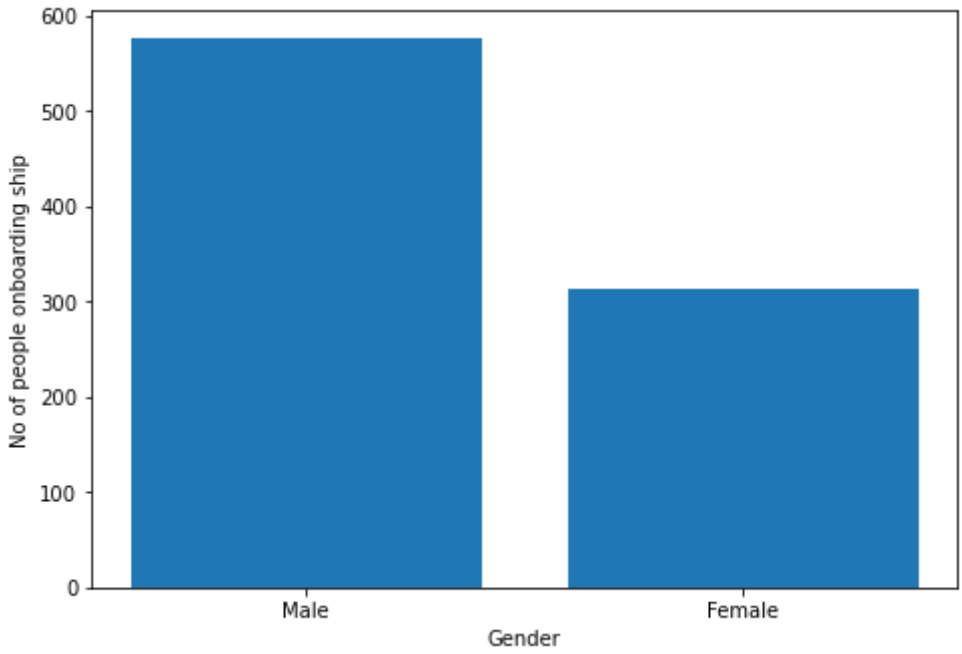
In [18]:

```
female_ind = len(train[train['Sex'] == 'female'])
print("No of Females in Titanic:",female_ind)
```

```
No of Females in Titanic: 314
```

In [19]:

```
#Plotting
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
gender = ['Male','Female']
index = [577,314]
ax.bar(gender,index)
plt.xlabel("Gender")
plt.ylabel("No of people onboarding ship")
plt.show()
```



In [20]:

```
alive = len(train[train['Survived'] == 1])
dead = len(train[train['Survived'] == 0])
```

In [21]:

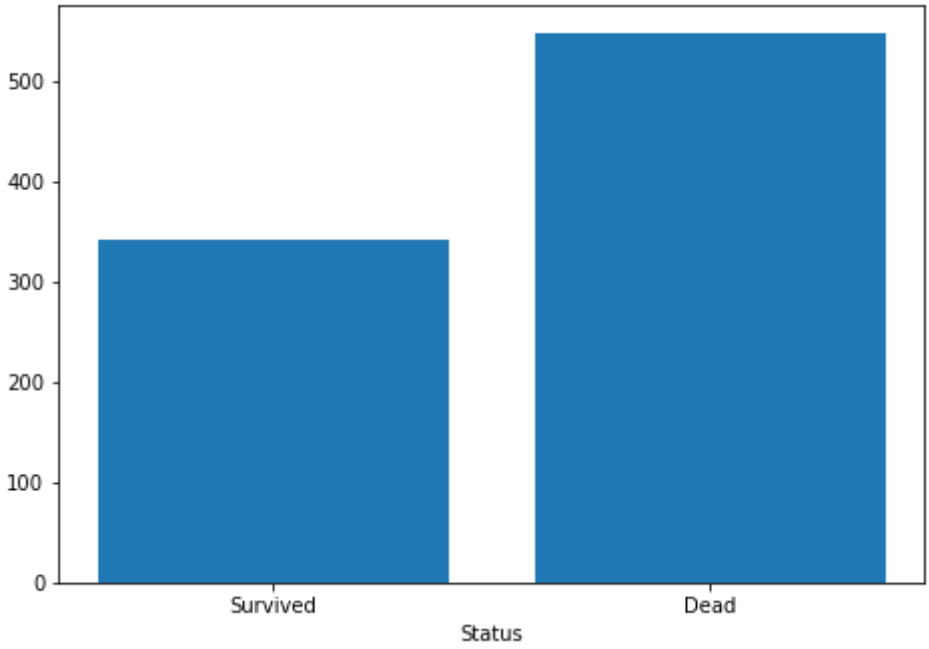
```
train.groupby('Sex')[['Survived']].mean()
```

Out[21]:

Sex	Survived
female	0.742038
male	0.188908

In [22]:

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
status = ['Survived','Dead']
ind = [alive,dead]
ax.bar(status,ind)
plt.xlabel("Status")
plt.show()
```



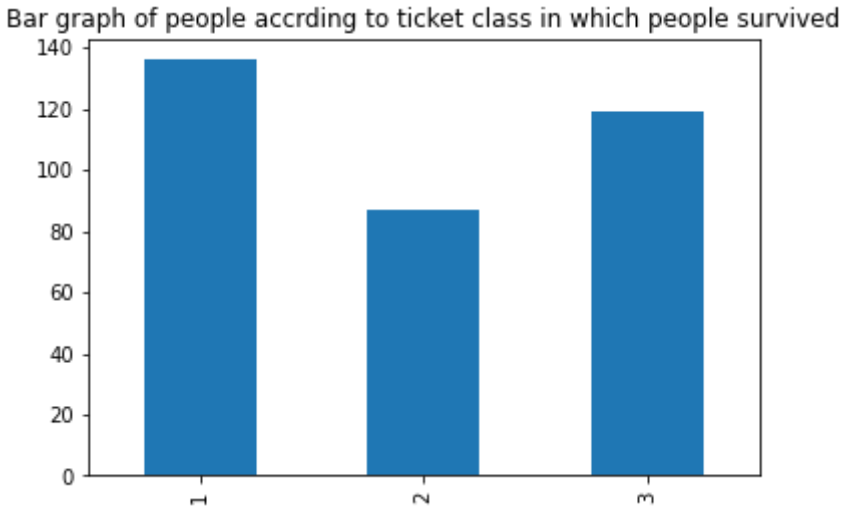
In [23]:

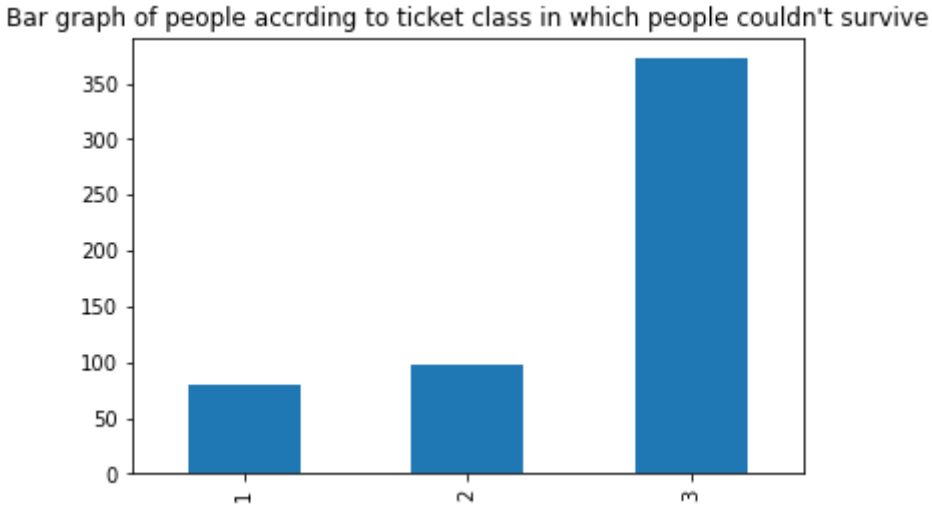
```
plt.figure(1)
train.loc[train['Survived'] == 1, 'Pclass'].value_counts().sort_index().plot.bar()
plt.title('Bar graph of people accrding to ticket class in which people survived')


plt.figure(2)
train.loc[train['Survived'] == 0, 'Pclass'].value_counts().sort_index().plot.bar()
plt.title('Bar graph of people accrding to ticket class in which people couldn\'t survive')
```

Out[23]:

```
Text(0.5, 1.0, "Bar graph of people accrding to ticket class in which people
couldn't survive")
```





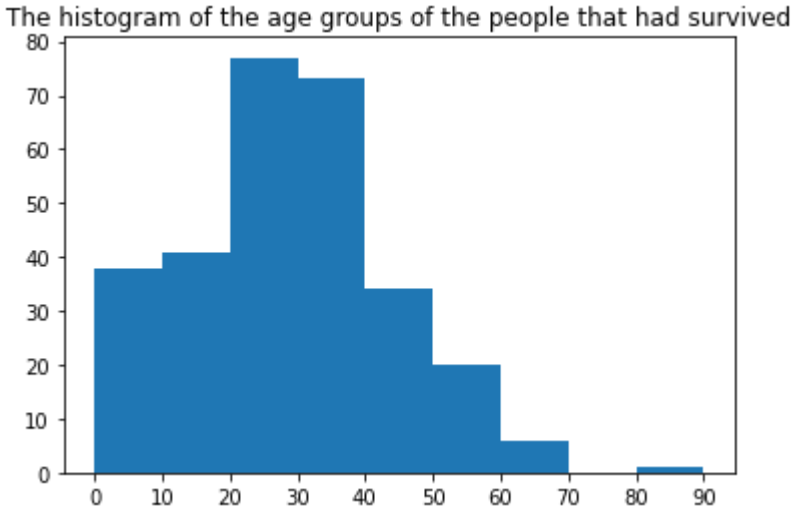
In [24]:

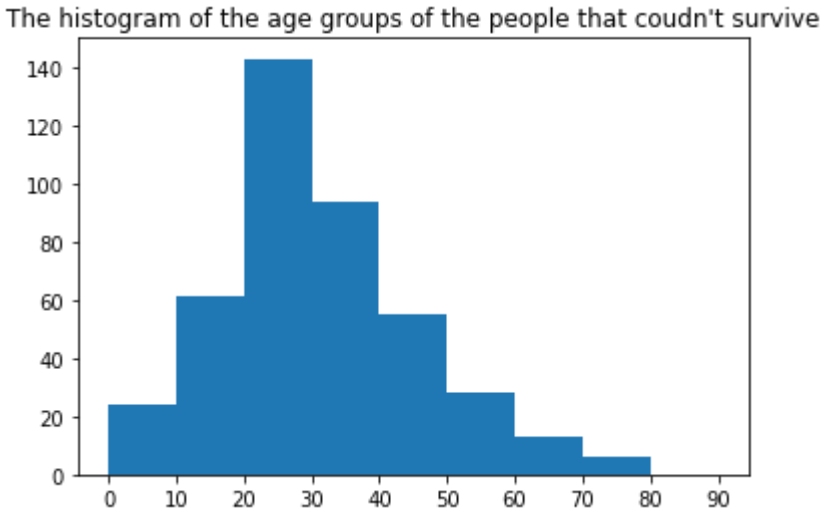
```
plt.figure(1)
age  = train.loc[train.Survived == 1, 'Age']
plt.title('The histogram of the age groups of the people that had survived')
plt.hist(age, np.arange(0,100,10))
plt.xticks(np.arange(0,100,10))


plt.figure(2)
age  = train.loc[train.Survived == 0, 'Age']
plt.title('The histogram of the age groups of the people that coudn\'t survive')
plt.hist(age, np.arange(0,100,10))
plt.xticks(np.arange(0,100,10))
```

Out[24]:

```
([<matplotlib.axis.XTick at 0x205ecca7670>,
  <matplotlib.axis.XTick at 0x205ecca7640>,
  <matplotlib.axis.XTick at 0x205ecc9dca0>,
  <matplotlib.axis.XTick at 0x205ecced400>,
  <matplotlib.axis.XTick at 0x205eccedb50>,
  <matplotlib.axis.XTick at 0x205eccf52e0>,
  <matplotlib.axis.XTick at 0x205ecced5b0>,
  <matplotlib.axis.XTick at 0x205eccf5910>,
  <matplotlib.axis.XTick at 0x205eccfd160>,
  <matplotlib.axis.XTick at 0x205eccfd7f0>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```





In [25]:

```
train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Surv
```

Out[25]:

	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 [26]:

```
train[["Pclass", "Survived"]].groupby(['Pclass'], as_index=False).mean().sort_values(by='Su
```

Out[26]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

In [27]:

```
train[["Age", "Survived"]].groupby(['Age'], as_index=False).mean().sort_values(by='Age', as
```

Out[27]:

	Age	Survived
0	0.42	1.0
1	0.67	1.0
2	0.75	1.0
3	0.83	1.0
4	0.92	1.0
...	...	...
83	70.00	0.0
84	70.50	0.0
85	71.00	0.0
86	74.00	0.0
87	80.00	1.0

88 rows × 2 columns

In [28]:

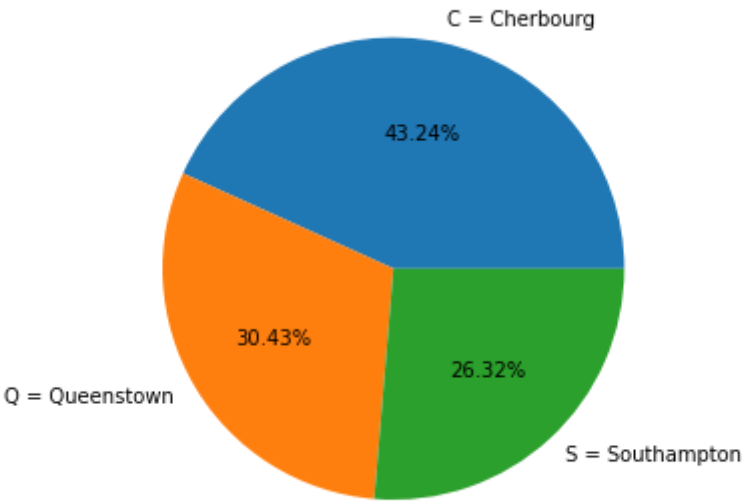
```
train[["Embarked", "Survived"]].groupby(['Embarked'], as_index=False).mean().sort_values(by
```

Out[28]:

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.336957

In [29]:

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
l = ['C = Cherbourg', 'Q = Queenstown', 'S = Southampton']
s = [0.553571,0.389610,0.336957]
ax.pie(s, labels = l,autopct='%1.2f%%')
plt.show()
```



In [30]:

```
test.describe(include="all")
```

Out[30]:

	PassengerId	Pclass	Name	Sex	Age	...	Parch	Ticket	Fare
count	418.000000	418.000000	418	418	332.000000	...	418.000000	418	417.000000
unique	NaN	NaN	418	2	NaN	...	NaN	363	NaN
top	NaN	NaN	Kelly, Mr. James	male	NaN	...	NaN	PC 17608	NaN
freq	NaN	NaN	1	266	NaN	...	NaN	5	NaN
mean	1100.500000	2.265550	NaN	NaN	30.272590	...	0.392344	NaN	35.627188
std	120.810458	0.841838	NaN	NaN	14.181209	...	0.981429	NaN	55.907576
min	892.000000	1.000000	NaN	NaN	0.170000	...	0.000000	NaN	0.000000
25%	996.250000	1.000000	NaN	NaN	21.000000	...	0.000000	NaN	7.895800
50%	1100.500000	3.000000	NaN	NaN	27.000000	...	0.000000	NaN	14.454200
75%	1204.750000	3.000000	NaN	NaN	39.000000	...	0.000000	NaN	31.500000
max	1309.000000	3.000000	NaN	NaN	76.000000	...	9.000000	NaN	512.329200

11 rows × 11 columns

In [31]:

```
#Droping Useless Columns
train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)
```

In [32]:

```
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)
```

In [33]:

```
train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)
```

In [34]:

```
#Feature Selection
column_train=['Age','Pclass','SibSp','Parch','Fare','Sex','Embarked']
#training values
X=train[column_train]
#target value
Y=train['Survived']
```

In [35]:

```
X['Age'].isnull().sum()
X['Pclass'].isnull().sum()
X['SibSp'].isnull().sum()
X['Parch'].isnull().sum()
X['Fare'].isnull().sum()
X['Sex'].isnull().sum()
X['Embarked'].isnull().sum()
```

Out[35]:

```
2
```

In [36]:

```
#now we have to fill all the missing values
#age have 177  missing values
#either we fill missing values with mean or median form existing values
X['Age']=X['Age'].fillna(X['Age'].median())
X['Age'].isnull().sum()
```

Out[36]:

```
0
```

In [37]:

```
X['Embarked'] = train['Embarked'].fillna(method ='pad')
X['Embarked'].isnull().sum()
```

Out[37]:

```
0
```

In [38]:

```
#now we need to convert sex into integer value
d={'male':0, 'female':1}
X['Sex']=X['Sex'].apply(lambda x:d[x])
X['Sex'].head()
```

Out[38]:

```
0    0
1    1
2    1
3    1
4    0
Name: Sex, dtype: int64
```

In [39]:

```
e={'C':0, 'Q':1 ,'S':2}
X['Embarked']=X['Embarked'].apply(lambda x:e[x])
X['Embarked'].head()
```

Out[39]:

```
0    2
1    0
2    2
3    2
4    2
Name: Embarked, dtype: int64
```

In [40]:

```
#Training Testing and Spliting the model
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3,random_state=7)
```

In [41]:

```
#Using LogisticRegression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,Y_pred))
```

```
Accuracy Score: 0.7574626865671642
```

In [42]:

```
#Confusion Matrix
from sklearn.metrics import accuracy_score,confusion_matrix
confusion_mat = confusion_matrix(Y_test,Y_pred)
print(confusion_mat)
```

```
[[130  26]
 [ 39  73]]
```

In [43]:

```
#Using Support Vector
from sklearn.svm import SVC
model1 = SVC()
model1.fit(X_train,Y_train)

pred_y = model1.predict(X_test)

from sklearn.metrics import accuracy_score
print("Acc=",accuracy_score(Y_test,pred_y))
```

```
Acc= 0.6604477611940298
```

In [44]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,pred_y)
print(confusion_mat)
print(classification_report(Y_test,pred_y))
```

```
[[149   7]
 [ 84  28]]
              precision    recall  f1-score   support

           0       0.64      0.96      0.77       156
           1       0.80      0.25      0.38       112

    accuracy                           0.66       268
   macro avg       0.72      0.60      0.57       268
weighted avg       0.71      0.66      0.61       268
```

In [45]:

```
#Using KNN Neighbors
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=5)
model2.fit(X_train,Y_train)
y_pred2 = model2.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred2))
```

```
Accuracy Score: 0.6604477611940298
```

In [48]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred2)
print(confusion_mat)
print(classification_report(Y_test,y_pred2))
```

```
[[126  30]
 [ 61  51]]
              precision    recall  f1-score   support

           0       0.67      0.81      0.73       156
           1       0.63      0.46      0.53       112

    accuracy                           0.66       268
   macro avg       0.65      0.63      0.63       268
weighted avg       0.66      0.66      0.65       268
```

In [50]:

```
#Using GaussianNB
from sklearn.naive_bayes import GaussianNB
model3 = GaussianNB()
model3.fit(X_train,Y_train)
y_pred3 = model3.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred3))
```

```
Accuracy Score: 0.7686567164179104
```

In [51]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred3)
print(confusion_mat)
print(classification_report(Y_test,y_pred3))
```

```
[[129  27]
 [ 35  77]]
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       156
           1       0.74      0.69      0.71       112

    accuracy                           0.77       268
   macro avg       0.76      0.76      0.76       268
weighted avg       0.77      0.77      0.77       268
```

In [52]:

```
#Using Decision Tree
from sklearn.tree import DecisionTreeClassifier
model4 = DecisionTreeClassifier(criterion='entropy',random_state=7)
model4.fit(X_train,Y_train)
y_pred4 = model4.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred4))
```

```
Accuracy Score: 0.7425373134328358
```

In [53]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred4)
print(confusion_mat)
print(classification_report(Y_test,y_pred4))
```

```
[[132  24]
 [ 45  67]]
              precision    recall  f1-score   support

           0       0.75      0.85      0.79       156
           1       0.74      0.60      0.66       112

    accuracy                           0.74       268
   macro avg       0.74      0.72      0.73       268
weighted avg       0.74      0.74      0.74       268
```

In [54]:

```
results = pd.DataFrame({
    'Model': ['Logistic Regression','Support Vector Machines', 'Naive Bayes','KNN' ,'Decisi
    'Score': [0.75,0.66,0.76,0.66,0.74]})

result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

Out[54]:

Score	Model
0.76	Naive Bayes
0.75	Logistic Regression
0.74	Decision Tree
0.66	Support Vector Machines
0.66	KNN

Sales Prediction (Simple Linear Regression)

```
Sales Prediction
( Simple Linear Regression)
Problem Statement
Build a model which predicts sales based on the money spent on different platforms for
marketing.

Data
Use the advertising dataset given in ISLR and analyse the relationship between 'TV
advertising' and 'sales' using a simple linear regression model.

In this notebook, we'll build a linear regression model to predict Sales using an
appropriate predictor variable.
```

In [55]:

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')

# Import the numpy and pandas package

import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
```

In [57]:

```
advertising = pd.DataFrame(pd.read_csv('https://raw.githubusercontent.com/training-ml/Files
advertising.head()
```

Out[57]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [58]:

```
advertising.shape
```

Out[58]:

```
(200, 5)
```

In [59]:

```
advertising.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  200 non-null    int64  
 1   TV          200 non-null    float64
 2   radio       200 non-null    float64
 3   newspaper   200 non-null    float64
 4   sales       200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

In [60]:

```
advertising.describe()
```

Out[60]:

	Unnamed: 0	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	100.500000	147.042500	23.264000	30.554000	14.022500
std	57.879185	85.854236	14.846809	21.778621	5.217457
min	1.000000	0.700000	0.000000	0.300000	1.600000
25%	50.750000	74.375000	9.975000	12.750000	10.375000
50%	100.500000	149.750000	22.900000	25.750000	12.900000
75%	150.250000	218.825000	36.525000	45.100000	17.400000
max	200.000000	296.400000	49.600000	114.000000	27.000000

In [61]:

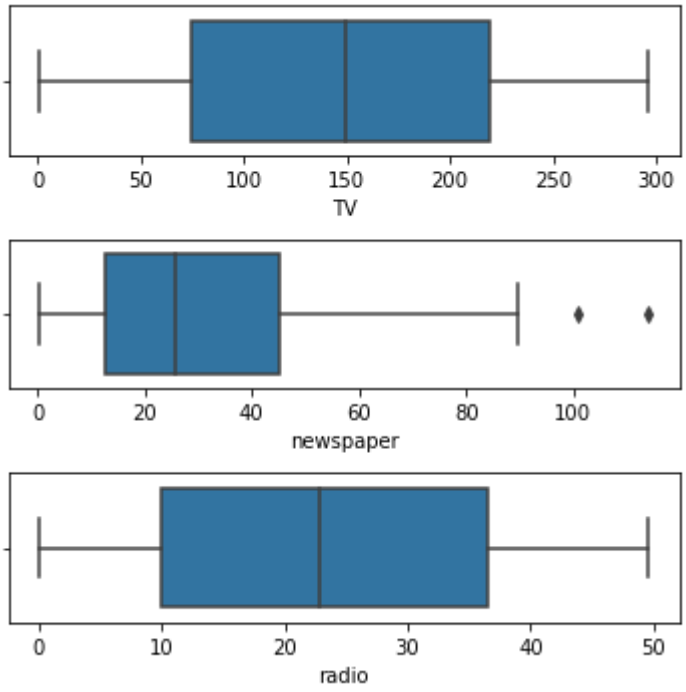
```
# Checking Null values
advertising.isnull().sum()*100/advertising.shape[0]
# There are no NULL values in the dataset, hence it is clean.
```

Out[61]:

```
Unnamed: 0    0.0
TV            0.0
radio         0.0
newspaper     0.0
sales         0.0
dtype: float64
```

In [65]:

```
# Outlier Analysis
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(advertising['TV'], ax = axs[0])
plt2 = sns.boxplot(advertising['newspaper'], ax = axs[1])
plt3 = sns.boxplot(advertising['radio'], ax = axs[2])
plt.tight_layout()
```



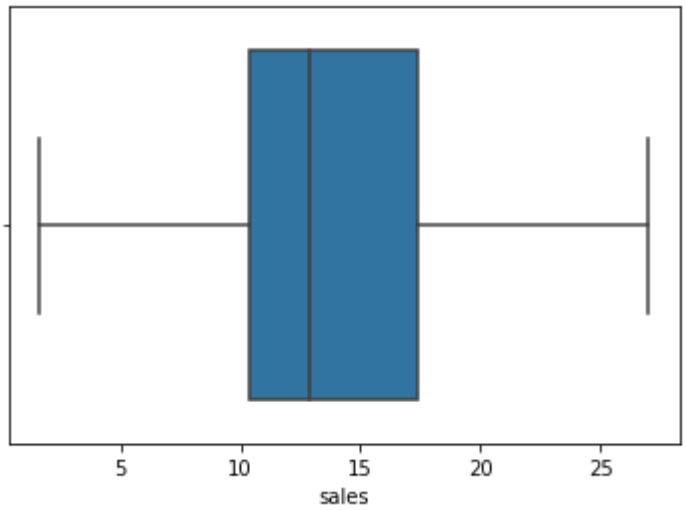
In [66]:

```
# There are no considerable outliers present in the data.
```

```
Exploratory Data Analysis
Univariate Analysis
Sales (Target Variable)
```

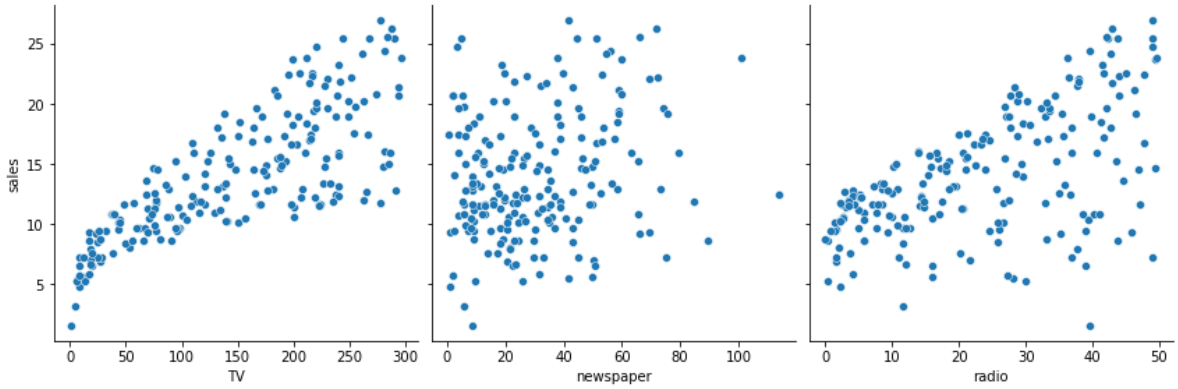
In [68]:

```
sns.boxplot(advertising['sales'])
plt.show()
```



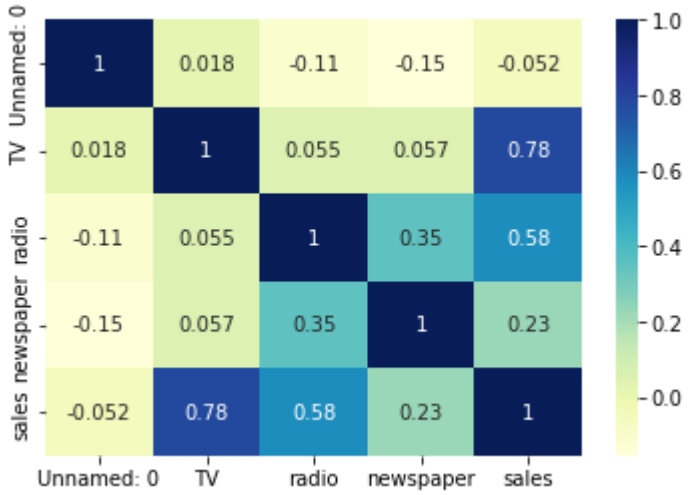
In [71]:

```
# Let's see how Sales are related with other variables using scatter plot.
sns.pairplot(advertising, x_vars=['TV', 'newspaper', 'radio'], y_vars='sales', height=4, as
plt.show()
```



In [72]:

```
# Let's see the correlation between different variables.
sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



As is visible from the pairplot and the heatmap, the variable TV seems to be most correlated with Sales. So let's go ahead and perform simple linear regression using TV as our feature variable.

Model Building
Performing Simple Linear Regression
Equation of linear regression
y=c+m1x1+m2x2+...+mnxn
y is the response
c is the intercept
m1 is the coefficient for the first feature
mn is the coefficient for the nth feature
In our case:

y=c+m1×TV
The m values are called the model coefficients or model parameters.

Generic Steps in model building using statsmodels
We first assign the feature variable, TV, in this case, to the variable X and the response variable, Sales, to the variable y.

In [74]:

```
X = advertising['TV']
y = advertising['sales']
```

Train-Test Split

You now need to split our variable into training and testing sets. You'll perform this by importing train_test_split from the sklearn.model_selection library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

In [75]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3
```

In [76]:

```
# Let's now take a look at the train dataset

X_train.head()
```

Out[76]:

```
74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
Name: TV, dtype: float64
```

In [77]:

```
y_train.head()
```

Out[77]:

```
74     17.0
3      18.5
185    22.6
26     15.0
90     11.2
Name: sales, dtype: float64
```

```
Building a Linear Model
You first need to import the statsmodel.api library using which you'll perform the linear
regression.
```

In [78]:

```
import statsmodels.api as sm
```

In [79]:

```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

In [80]:

```
# Print the parameters, i.e. the intercept and the slope of the regression line fitted
lr.params
```

Out[80]:

```
const    6.989666
TV       0.046497
dtype: float64
```

In [81]:

```
# Performing a summary operation lists out all the different parameters of the regression l
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                   sales   R-squared:                       0.6
13
Model:                           OLS   Adj. R-squared:                  0.6
11
Method:                Least Squares   F-statistic:                     21
9.0
Date:               Mon, 13 Jun 2022   Prob (F-statistic):           2.84e-
30
Time:                        23:04:38   Log-Likelihood:                -370.
62
No. Observations:                 140   AIC:                             74
5.2
Df Residuals:                     138   BIC:                             75
1.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          6.9897      0.548     12.762      0.000       5.907       8.0
73
TV             0.0465      0.003     14.798      0.000       0.040       0.0
53
==============================================================================
==
Omnibus:                        0.995   Durbin-Watson:                   1.9
83
Prob(Omnibus):                  0.608   Jarque-Bera (JB):                0.9
70
Skew:                          -0.008   Prob(JB):                        0.6
16
Kurtosis:                       2.593   Cond. No.                         32
8.
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

```
Looking at some key statistics from the summary
The values we are concerned with are -

The coefficients and significance (p-values)
R-squared
F statistic and its significance
1. The coefficient for TV is 0.054, with a very low p value
The coefficient is statistically significant. So the association is not purely by chance.
```

2. R - squared is 0.816
Meaning that 81.6% of the variance in Sales is explained by TV

This is a decent R-squared value.

3. F statistic has a very low p value (practically low)
Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

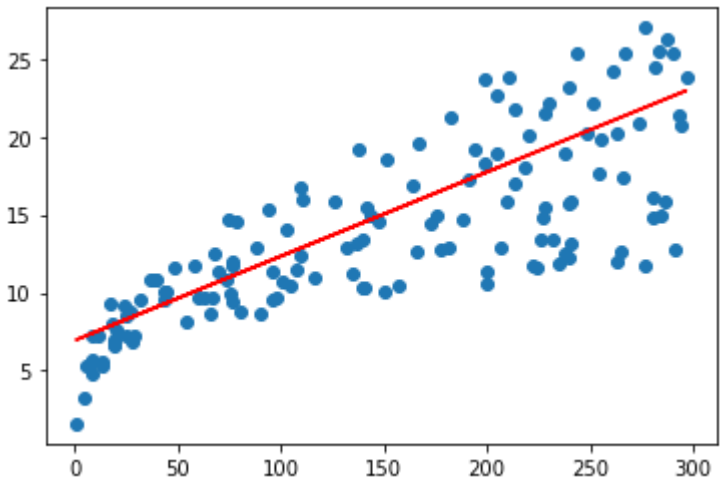
The fit is significant. Let's visualize how well the model fit the data.

From the parameters that we get, our linear regression equation becomes:

Sales=6.948+0.054×TV

In [82]:

```
plt.scatter(X_train, y_train)
plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
plt.show()
```



Model Evaluation
Residual analysis
To validate assumptions of the model, and hence the reliability for inference

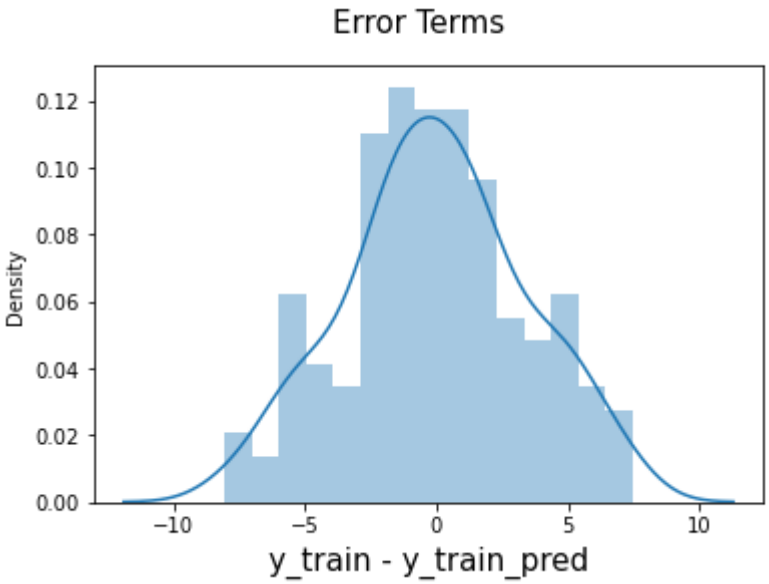
Distribution of the error terms
We need to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

In [83]:

```
y_train_pred = lr.predict(X_train_sm)
res = (y_train - y_train_pred)
```

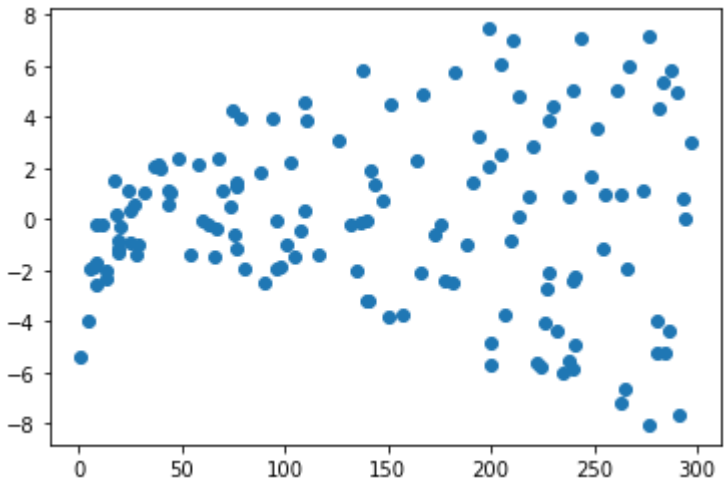
In [84]:

```
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.xlabel('y_train - y_train_pred', fontsize = 15)         # X-label
plt.show()
```



In [85]:

```
plt.scatter(X_train,res)
plt.show()
```



In [86]:

```
# Add a constant to X_test
X_test_sm = sm.add_constant(X_test)

# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

In [87]:

```
y_pred.head()
```

Out[87]:

```
126     7.352345
104    18.065337
99     13.276109
92     17.112141
111    18.228077
dtype: float64
```

In [88]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [89]:

```
#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[89]:

```
2.8241456288327016
```

In [90]:

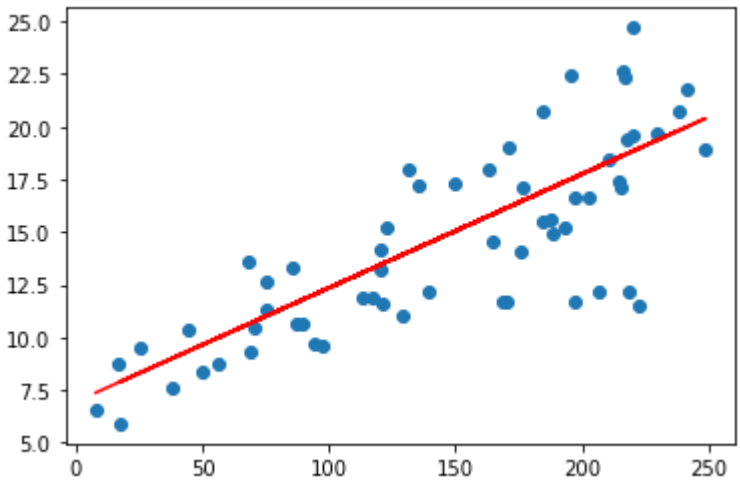
```
r_squared = r2_score(y_test, y_pred)
r_squared
```

Out[90]:

```
0.59429872677833
```

In [91]:

```
plt.scatter(X_test, y_test)
plt.plot(X_test, 6.948 + 0.054 * X_test, 'r')
plt.show()
```



Big-Mart-Sales/Bigmart_XGBoost3.ipynb

In [92]:

```
# importing required libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [94]:

```
# reading train and test data
train_data = pd.read_csv('https://raw.githubusercontent.com/akki8087/Big-Mart-Sales/master/
test_data = pd.read_csv('https://raw.githubusercontent.com/akki8087/Big-Mart-Sales/master/t
```

In [95]:

```
train_data.head()
```

Out[95]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	...	Outlet_Estab
0	FDA15	9.30	Low Fat	0.016047	Dairy	...	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	...	
2	FDN15	17.50	Low Fat	0.016760	Meat	...	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	...	
4	NCD19	8.93	Low Fat	0.000000	Household	...	

5 rows × 12 columns

In [96]:

```
test_data.apply(lambda x: sum(x.isnull()))
```

Out[96]:

```
Item_Identifier                 0
Item_Weight                   976
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  1606
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64
```

In [97]:

```
test_data['Item_Fat_Content'].unique()
```

Out[97]:

```
array(['Low Fat', 'reg', 'Regular', 'LF', 'low fat'], dtype=object)
```

In [98]:

```
# combining Item_Fat_Content misspelled
train_data['Item_Fat_Content'].replace(['low fat','LF','reg'],['Low Fat','Low Fat','Regular
test_data['Item_Fat_Content'].replace(['low fat','LF','reg'],['Low Fat','Low Fat','Regular'
```

In [99]:

```
# creating new column num_years
train_data['num_years'] = train_data['Outlet_Establishment_Year'].apply(lambda x: 2013 - x)
test_data['num_years'] = test_data['Outlet_Establishment_Year'].apply(lambda x: 2013 - x)
```

In [100]:

```
train_data['Item_Type'].unique()
```

Out[100]:

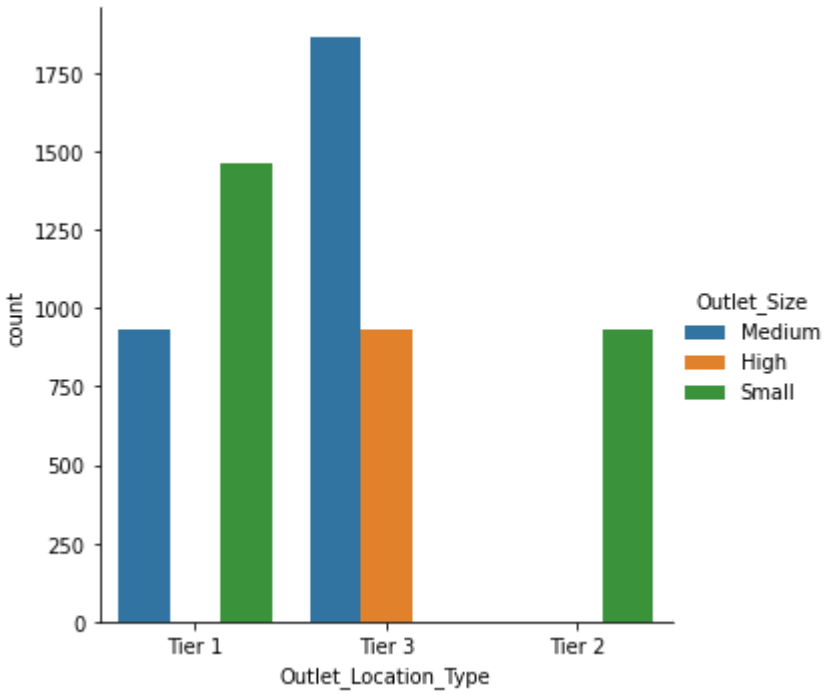
```
array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
       'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
       'Breakfast', 'Health and Hygiene', 'Hard Drinks', 'Canned',
       'Breads', 'Starchy Foods', 'Others', 'Seafood'], dtype=object)
```

In [101]:

```
sns.factorplot('Outlet_Location_Type',data = train_data,hue ='Outlet_Size' ,kind='count')
```

Out[101]:

```
<seaborn.axisgrid.FacetGrid at 0x205f0e017c0>
```

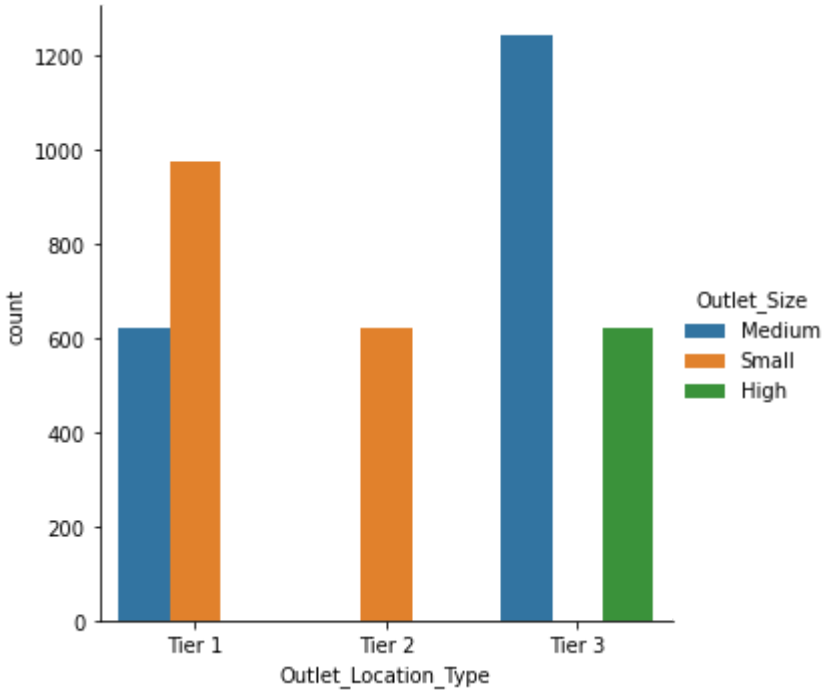


In [102]:

```
sns.factorplot('Outlet_Location_Type',data = test_data,hue ='Outlet_Size' ,kind='count')
```

Out[102]:

```
<seaborn.axisgrid.FacetGrid at 0x205f1f7bbe0>
```

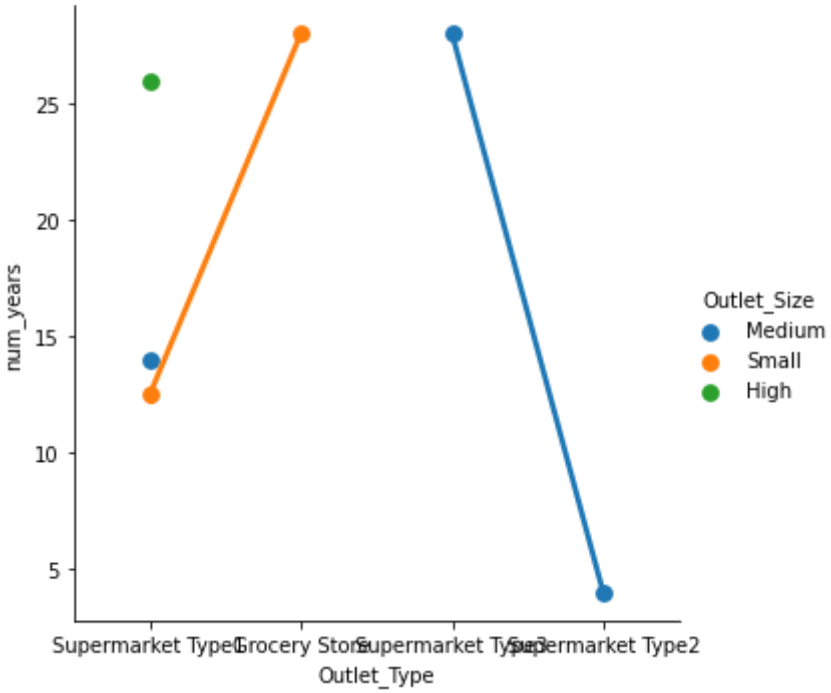


In [103]:

```
sns.factorplot('Outlet_Type','num_years',data = test_data,hue='Outlet_Size' )
```

Out[103]:

```
<seaborn.axisgrid.FacetGrid at 0x205efc3aeb0>
```

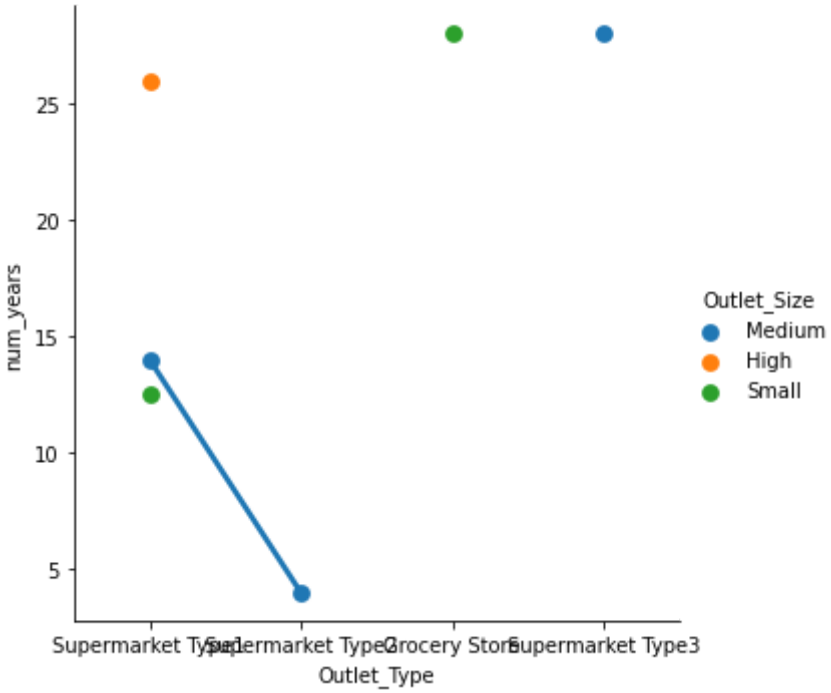


In [104]:

```
sns.factorplot('Outlet_Type','num_years',data = train_data,hue='Outlet_Size' )
```

Out[104]:

```
<seaborn.axisgrid.FacetGrid at 0x205f1f82bb0>
```



In [105]:

```
full_data = [train_data, test_data]
```

In [106]:

```
# filling null values
for data in full_data:
    data['Item_Weight'].fillna(data['Item_Weight'].mean(),inplace = True)
    data['Outlet_Size'].fillna('Medium',inplace = True)
```

In [107]:

```
col = ['Item_Fat_Content', 'Item_Type', 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type
```

In [108]:

```
# handling catagorical variables
train_datar = pd.get_dummies(train_data, columns = col, drop_first = True)
test_datar = pd.get_dummies(test_data, columns = col,drop_first = True)
```

In [109]:

```
feat_cols = ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'num_years',
       'Item_Fat_Content_Regular', 'Item_Type_Breads', 'Item_Type_Breakfast',
       'Item_Type_Canned', 'Item_Type_Dairy', 'Item_Type_Frozen Foods',
       'Item_Type_Fruits and Vegetables', 'Item_Type_Hard Drinks',
       'Item_Type_Health and Hygiene', 'Item_Type_Household', 'Item_Type_Meat',
       'Item_Type_Others', 'Item_Type_Seafood', 'Item_Type_Snack Foods',
       'Item_Type_Soft Drinks', 'Item_Type_Starchy Foods',
       'Outlet_Size_Medium', 'Outlet_Size_Small',
       'Outlet_Location_Type_Tier 2', 'Outlet_Location_Type_Tier 3',
       'Outlet_Type_Supermarket Type1', 'Outlet_Type_Supermarket Type2',
       'Outlet_Type_Supermarket Type3']
```

In [110]:

```
X = train_datar[feat_cols]
y = train_datar['Item_Outlet_Sales']
```

In [111]:

```
# splitting data as X_train and X_test
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,random_state = 0)
```

In [112]:

```
# creating XGBoost model
from xgboost.sklearn import XGBRegressor
XGB = XGBRegressor()
XGB.fit(X_train, y_train)
y_pred = XGB.predict(X_test)
```

In [113]:

```
# calculating RMSE
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
```

In [114]:

```
rmse
```

Out[114]:

```
1199.8410346504536
```

In [115]:

```
# predicting on actual test data
X_t = test_datar[feat_cols]
y_result = XGB.predict(X_t)
```

In [116]:

```
y_result
```

Out[116]:

```
array([1747.2274 , 1265.3572 ,  202.73059, ..., 1714.7622 , 5602.5938 ,
       1564.9109 ], dtype=float32)
```

In [117]:

```
#creating results .csv file
result = pd.DataFrame()
result['Item_Identifier'] = test_datar['Item_Identifier']
result['Outlet_Identifier'] = test_datar['Outlet_Identifier']

result["Item_Outlet_Sales"] = y_result
result = result.sort_index()
result.to_csv('Bigmart_XGBoost3.csv',index = False)
```

In [ ]:

```

```