CS:311-Data Mining

Data Mining Lab Assignment

Name: Saurabh Chaturvedi

Exam Roll No: 21419MCA050

Importing Required Libraries

In [86]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import statsmodels.api as sm
import pylab as py
from scipy import stats
from sklearn.model_selection import train_test_split
from mlxtend.frequent_patterns import association_rules, apriori
from mlxtend.preprocessing import TransactionEncoder
import pyfpgrowth
import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

Qus-1: Data preprocessing and Visualization for data mining

For this problem, I have used the titanic dataset.

In [2]:
```
df=pd.read_csv("C:/Users/Mycomputer/Documents/MCA_BHU/data mining/DM_Assignment/datasets/titanic_dataset.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarke
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

In [4]:
```
df.describe()
```

Out[4]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [5]:
```
# Names of features or preditors in the dataset
df.columns
```

Out[5]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [6]:
```
columns = ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
```

In [7]:
```
#cheking rows and columns
df.shape
```

Out[7]:
```
(418, 12)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

In [9]:
```
# datatypes of each columns or features
df.dtypes
```

Out[9]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

- Numerical columns - PassengerId,Survived,Pclass,Age,Sibsp,Parch,Ticket,Fare.
- Categorial Columns - Name,Sex,cabin,Embarked

In [10]:
```
# number of unique values in each columns
df.nunique()
```

Out[10]:
```
PassengerId    418
Survived         2
Pclass           3
Name           418
Sex              2
Age             79
SibSp            7
Parch            8
Ticket         363
Fare           169
Cabin           76
Embarked         3
dtype: int64
```

Handling Missing Values

In [11]:
```
# missing values
df.isna()
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
413	False	False	False	False	False	True	False	False	False	False	True	False
414	False	False	False	False	False	False	False	False	False	False	False	False
415	False	False	False	False	False	False	False	False	False	False	True	False
416	False	False	False	False	False	True	False	False	False	False	True	False
417	False	False	False	False	False	True	False	False	False	False	True	False

418 rows × 12 columns

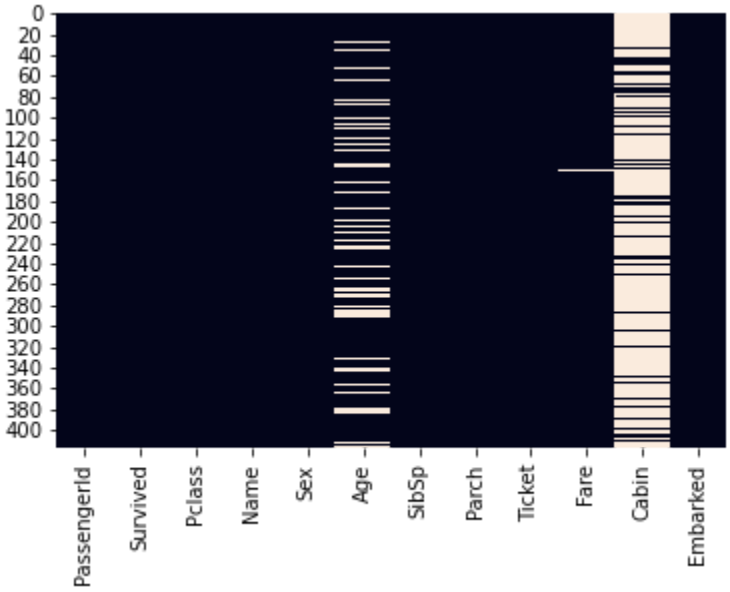
In [12]:
```
df.isna().sum()
```

Out[12]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

- since this data contains 400 records and cabin has 327 missing values so it's better to drop this columns

In [13]:
```
### Visualizing the missing values using heatmap
sns.heatmap(df.isnull(),cbar=False)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e5589730>
```



Insights

- 'Survived' is the target column/variable.
- 'PassengerId', 'Name' and 'Ticket' doesn't contribute to the target variable 'Survived'. So, we can remove it from the data.
- 'Age' and 'Embarked' has less number of missing value. We have to impute them using different techniques.
- As there are a lot of missing values in the column 'Cabin', we can remove it from the training data.
- As there are a lot of missing values in the column 'Cabin', we can remove it from the training data.
- We can also create new variable like 'total size of the family' from the columns 'SibSp' and 'Parch'.

In [14]:
```
# droping the cabin column since it contains very large number of missing value
s
df.drop(['Cabin'],axis=1,inplace=True)
```

In [15]:
```
df.isna().sum()
```

Out[15]:
```
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age            86
SibSp           0
Parch           0
Ticket          0
Fare            1
Embarked        0
dtype: int64
```

For Age columns , we can fill missing values with their mean or median value.

In [16]:
```
# filling missing values with its mean
df['Age']=df['Age'].fillna(df['Age'].mean())
```

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

```
<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   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          418 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 36.0+ KB
```

In [18]:
```
df.isna().sum()
```

Out[18]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           1
Embarked       0
dtype: int64
```

In [19]:
```
### And, For fair column,simply remove the row with missing values as it contai
ns only one missing values
df=df.dropna()
```

In [20]:
```
df.shape
```

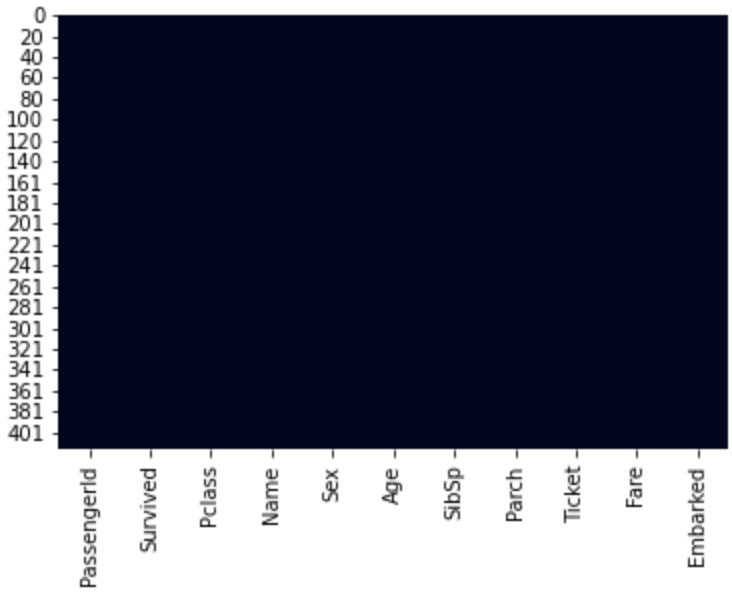
Out[20]:
```
(417, 11)
```

In [21]:
```
df.isna().sum()
```

Out[21]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [22]:
```
sns.heatmap(df.isnull(),cbar=False)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e5551730>
```



Find the mean, median, mode, max, min, quantiles, outliers, standard deviation and variance of the data

In [23]:
```
df.head()
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Q
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	S

Mean

In [24]:
```
df.mean()
```

Out[24]:
```
PassengerId    1100.635492
Survived          0.364508
Pclass            2.263789
Age              30.200103
SibSp             0.448441
Parch             0.393285
Fare             35.627188
dtype: float64
```

Median

In [25]:
```
df.median()
```

Out[25]:
```
PassengerId    1101.00000
Survived          0.00000
Pclass            3.00000
Age              30.27259
SibSp             0.00000
Parch             0.00000
Fare             14.45420
dtype: float64
```

Mode

In [26]:
```
df.mode()
```

Out[26]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	892	0.0	3.0	Abbott, Master. Eugene Joseph	male	30.27259	0.0	0.0	PC 17608	7.75	S
1	893	NaN	NaN	Abelseth, Miss. Karen Marie	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	894	NaN	NaN	Abelseth, Mr. Olaus Jorgensen	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	895	NaN	NaN	Abrahamsson, Mr. Abraham August Johannes	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	896	NaN	NaN	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
412	1305	NaN	NaN	de Brito, Mr. Jose Joaquim	NaN	NaN	NaN	NaN	NaN	NaN	NaN
413	1306	NaN	NaN	de Messemaeker, Mr. Guillaume Joseph	NaN	NaN	NaN	NaN	NaN	NaN	NaN
414	1307	NaN	NaN	del Carlo, Mrs. Sebastiano (Argenia Genovesi)	NaN	NaN	NaN	NaN	NaN	NaN	NaN
415	1308	NaN	NaN	van Billiard, Master. James William	NaN	NaN	NaN	NaN	NaN	NaN	NaN
416	1309	NaN	NaN	van Billiard, Master. Walter John	NaN	NaN	NaN	NaN	NaN	NaN	NaN

417 rows × 11 columns

Variance

In [27]:
```
df.var()
```

Out[27]:
```
PassengerId    14622.559122
Survived           0.232199
Pclass             0.709094
Age              157.813531
SibSp              0.805629
Parch              0.965147
Fare            3125.657074
dtype: float64
```

Standard deviation

In [28]:
```
df.std()
```

Out[28]:
```
PassengerId    120.923774
Survived         0.481870
Pclass           0.842077
Age             12.562386
SibSp            0.897568
Parch            0.982419
Fare            55.907576
dtype: float64
```

Maximum values

In [29]:
```
df.max()
```

Out[29]:
```
PassengerId                                 1309
Survived                                       1
Pclass                                         3
Name           van Billiard, Master. Walter John
Sex                                         male
Age                                           76
SibSp                                          8
Parch                                          9
Ticket                                W.E.P. 5734
Fare                                      512.329
Embarked                                        S
dtype: object
```

Minimum

In [30]:
```
min = df.min()
min
```

Out[30]:
```
PassengerId                                   892
Survived                                        0
Pclass                                          1
Name           Abbott, Master. Eugene Joseph
Sex                                        female
Age                                          0.17
SibSp                                           0
Parch                                           0
Ticket                                     110469
Fare                                            0
Embarked                                        C
dtype: object
```

Find the first quartile (Q1), the third quartile (Q3) of the data and Inter-quartile range.

Converting categorial features

In [31]:
```
df1=df.copy()
```

In [32]:
```
sex = pd.get_dummies(df1['Sex'],drop_first=True)
embark = pd.get_dummies(df1['Embarked'],drop_first=True)
```

In [33]:
```
df1.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [34]:
```
df1 = pd.concat([df1,sex,embark],axis=1)
df1
```

Out[34]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	892	0	3	34.50000	0	0	7.8292	1	1	0
1	893	1	3	47.00000	1	0	7.0000	0	0	1
2	894	0	2	62.00000	0	0	9.6875	1	1	0
3	895	0	3	27.00000	0	0	8.6625	1	0	1
4	896	1	3	22.00000	1	1	12.2875	0	0	1
...	...	...	...	...	...	...	...	...	...	...
413	1305	0	3	30.27259	0	0	8.0500	1	0	1
414	1306	1	1	39.00000	0	0	108.9000	0	0	0
415	1307	0	3	38.50000	0	0	7.2500	1	0	1
416	1308	0	3	30.27259	0	0	8.0500	1	0	1
417	1309	0	3	30.27259	1	1	22.3583	1	0	0

417 rows × 10 columns

In [35]:
```
np.quantile(df1, 0.50,axis=0)
```

Out[35]:
```
array([1.10100000e+03, 0.00000000e+00, 3.00000000e+00, 3.02725904e+01,
       0.00000000e+00, 0.00000000e+00, 1.44542000e+01, 1.00000000e+00,
       0.00000000e+00, 1.00000000e+00])
```

First Quartile Q1

In [36]:
```
q1 = np.quantile(df1, 0.25,axis=0)
q1
```

Out[36]:
```
array([996.    ,   0.    ,   1.    ,  23.    ,   0.    ,   0.    ,
         7.8958,   0.    ,   0.    ,   0.    ])
```

Third Quartile Q3

In [37]:
```
q3 = np.quantile(df1, 0.75,axis=0)
q3
```

Out[37]:
```
array([1.205e+03, 1.000e+00, 3.000e+00, 3.500e+01, 1.000e+00, 0.000e+00,
       3.150e+01, 1.000e+00, 0.000e+00, 1.000e+00])
```

Inter-Quartile Range

In [38]:
```
iqr = q3-q1
iqr
```

Out[38]:
```
array([209.    ,   1.    ,   2.    ,  12.    ,   1.    ,   0.    ,
        23.6042,   1.    ,   0.    ,   1.    ])
```

Five number summary of data

In [39]:
```
print("Five number summary of the data")
des=df.describe()
des
des.iloc[3:8,]
```

```
Five number summary of the data
```

Out[39]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
min	892.0	0.0	1.0	0.17000	0.0	0.0	0.0000
25%	996.0	0.0	1.0	23.00000	0.0	0.0	7.8958
50%	1101.0	0.0	3.0	30.27259	0.0	0.0	14.4542
75%	1205.0	1.0	3.0	35.00000	1.0	0.0	31.5000
max	1309.0	1.0	3.0	76.00000	8.0	9.0	512.3292

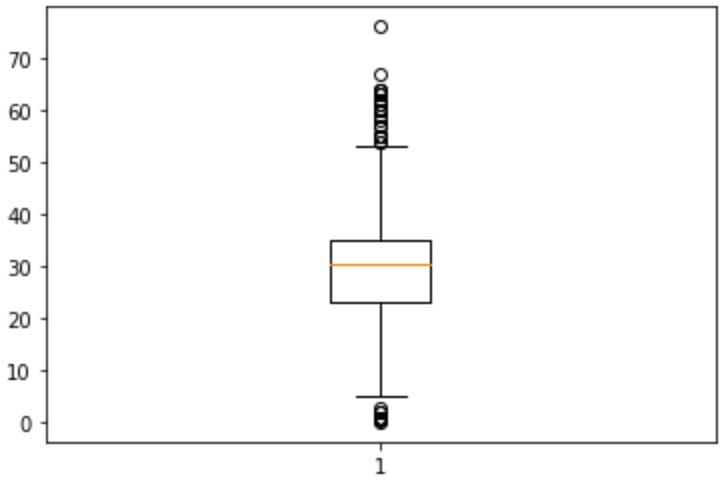
Outlier Detection

- Outliers can be detected using visualization like using box plot and scatter plot, or using Inter-Quartile Range and z-sore(mathematial approach).

1.Using Box plot

In [40]:
```
plt.boxplot(df['Age'])
```

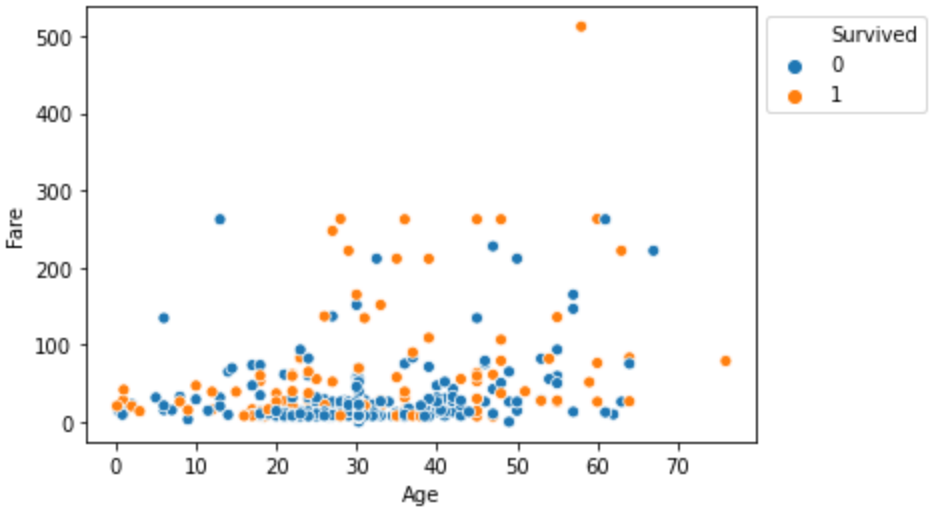
Out[40]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x201e5df7d00>,
  <matplotlib.lines.Line2D at 0x201e5e020a0>],
 'caps': [<matplotlib.lines.Line2D at 0x201e5e02400>,
  <matplotlib.lines.Line2D at 0x201e5e02760>],
 'boxes': [<matplotlib.lines.Line2D at 0x201e5df79a0>],
 'medians': [<matplotlib.lines.Line2D at 0x201e5e02ac0>],
 'fliers': [<matplotlib.lines.Line2D at 0x201e5e02dc0>],
 'means': []}
```



- In the above graph, it's clearly visible that values above 60 are acting as Outliers.

2.Using Scatter plot

In [41]:
```
## scatter plot between Age and PassengerId columns
sns.scatterplot(x='Age', y='Fare',
                hue='Survived', data=df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```



- By Looking at the graph we can summarize that most of the data points are in the bottom of the graph and the points that are up-right corners are acting as outliers.

.Using Inter-quartile Range

In [42]:
```
# IQR
Q1=np.percentile(df['Age'], 25, interpolation = 'midpoint')
Q3=np.percentile(df['Age'], 75, interpolation = 'midpoint')
IQR = Q3-Q1
IQR
```

Out[42]:
```
12.0
```

In [43]:
```
# Above Upper bound
upper =(Q3+1.5*IQR)

print("Upper bound:",upper)
print(np.where(df['Age']>=upper))

# Below Lower bound
lower =(Q1-1.5*IQR)
print("Lower bound:",lower)
print(np.where(df['Age']<=lower))
```

```
Upper bound: 53.0
(array([  2,  13,  20,  48,  69,  77,  81,  96, 114, 131, 142, 176, 178,
       192, 212, 216, 223, 235, 239, 292, 304, 307, 313, 315, 342, 355,
       373, 377, 386], dtype=int64),)
Lower bound: 5.0
(array([ 89, 117, 200, 249, 262, 280, 283, 295, 306, 353, 378, 408],
      dtype=int64),)
```

- This points are acting as outliers.

4.Z-score

In [44]:
```
z = np.abs(stats.zscore(df['Age']))
print(z)
```

```
[0.34269466 1.33892384 2.53439885 0.25504284 0.65353451 1.29112119
 0.01594784 0.33474118 0.97232785 0.73323285 0.00577716 1.25922551
 0.57383618 2.61409719 1.33892384 0.49413785 0.38254383 0.73323285
 0.25504284 1.17952717 1.97651051 1.68961286 0.00577716 0.73323285
 1.41862217 1.57801884 0.65353451 0.61368535 0.86073383 0.00577716
 1.57801884 0.49413785 0.22314716 0.00577716 0.01594784 0.93247868
 0.00577716 0.73323285 0.41443951 0.00577716 0.70133717 0.00577716
 0.86073383 0.01594784 1.17952717 0.41443951 1.17952717 0.00577716
 2.37500218 0.46224216 0.49413785 0.25504284 0.81293118 0.17534451
 0.00577716 1.60991453 0.38254383 0.41443951 0.00577716 0.46224216
 1.05202619 0.14344883 0.97232785 0.65353451 1.37081952 0.00577716
 0.97232785 1.33892384 0.06375049 2.37500218 0.49413785 0.73323285
 0.09564618 0.13549534 0.38254383 0.18329799 0.00577716 1.97651051
 0.01594784 0.49413785 1.92870786 2.93289052 1.49832051 0.00577716
 0.00577716 0.00577716 0.25504284 0.97232785 0.00577716 2.2475012
 0.65353451 0.00577716 0.25504284 0.00577716 0.41443951 0.41443951
 3.65017553 0.09564618 0.81293118 0.22314716 1.0201305  0.25504284
 0.00577716 0.33474118 1.13172452 0.17534451 0.73323285 0.00577716
 0.00577716 0.93247868 0.86073383 0.00577716 0.46224216 0.93247868
 2.61409719 0.97232785 0.00577716 2.32719953 0.46224216 0.09564618
 1.45051786 0.00577716 0.38254383 0.17534451 0.00577716 1.05202619
 0.65353451 0.00577716 0.94043217 0.49413785 0.14344883 1.81711384
 0.00577716 0.00577716 1.0201305  0.49413785 0.29489201 0.33474118
 0.57383618 0.7810355  1.60991453 0.22314716 2.45470052 0.17534451
 0.94043217 0.06375049 0.00577716 0.65353451 0.00577716 0.01594784
 0.57383618 0.00577716 0.46224216 1.37081952 0.49413785 0.09564618
 0.57383618 0.94043217 0.33474118 0.00577716 1.84900953 0.33474118
 0.00577716 0.86073383 0.33474118 1.41862217 0.97232785 0.00577716
 0.65353451 0.00577716 0.25504284 0.57383618 0.00577716 0.7810355
 1.21142285 0.81293118 1.89681218 0.46224216 2.69379552 0.01594784
 0.5419405  0.97232785 0.00577716 0.25504284 0.7810355  0.73323285
 1.05202619 0.00577716 0.7810355  0.3028455  0.00577716 1.49036702
 2.45470052 1.76931119 0.22314716 1.92870786 0.97232785 0.57383618
 0.00577716 0.00577716 2.38059742 1.33892384 1.76931119 0.41443951
 0.00577716 0.38254383 0.49413785 0.22314716 0.41443951 0.14344883
 0.00577716 1.05202619 2.37500218 0.62163883 0.94043217 0.00577716
 2.13590718 1.57801884 0.00577716 0.01594784 0.73323285 0.65353451
 0.73323285 1.81711384 0.00577716 0.57383618 0.00577716 0.82088467
 0.46224216 1.29112119 0.73323285 0.73323285 0.00577716 0.70133717
 0.81293118 2.69379552 0.81293118 0.97232785 1.41862217 1.97651051
 1.17952717 1.17952717 0.00577716 0.00577716 0.86073383 0.65353451
 0.94043217 0.09564618 0.00577716 2.3335754  0.81293118 0.25504284
 0.49413785 0.18329799 0.00577716 0.00577716 0.17534451 0.89262952
 0.73323285 0.50209133 0.73323285 0.09564618 2.32719953 0.01594784
 0.00577716 0.00577716 0.00577716 0.00577716 1.05202619 1.25922551
 0.00577716 0.33474118 0.00577716 0.00577716 0.81293118 0.17534451
 0.7810355  0.01594784 0.65353451 0.57383618 2.34712412 0.00577716
 1.68961286 2.2475012  0.46224216 0.00577716 0.49413785 0.00577716
 0.00577716 0.00577716 0.01594784 0.00577716 1.81711384 0.46224216
 0.33474118 2.32719953 0.00577716 0.01594784 0.09564618 0.14344883
 0.00577716 1.0201305  0.49413785 0.00577716 2.69379552 0.01594784
 2.34074825 1.97651051 1.17952717 0.97232785 0.65353451 0.00577716
 0.5419405  1.97651051 1.05202619 2.13590718 0.89262952 0.25504284
 0.65353451 0.33474118 0.41443951 0.33474118 0.22314716 0.70133717
 0.57383618 1.45051786 1.25922551 0.09564618 0.73323285 1.41862217
 0.70133717 0.00577716 0.89262952 0.25504284 0.01594784 0.14344883
 0.70133717 0.41443951 0.00577716 0.97232785 0.14344883 0.00577716
 2.21560552 0.00577716 1.13172452 0.33474118 0.62163883 0.49413785
 0.06375049 1.17952717 0.41443951 0.97232785 1.49832051 2.39334915
 1.57801884 2.29530385 0.00577716 0.00577716 0.01594784 1.25127202
 0.49413785 0.06375049 0.25504284 0.41443951 0.00577716 0.00577716
```

```
0.65353451 1.17952717 0.09564618 0.73323285 0.06375049 1.49832051
1.09982884 1.89681218 1.17952717 0.65353451 0.73323285 1.97651051
2.0084062  0.00577716 0.33474118 0.00577716 0.89262952 0.00577716
0.49413785 0.49413785 2.13590718 0.73323285 1.92870786 0.57383618
1.65771718 1.37081952 1.33892384 0.09564618 0.97232785 0.49413785
1.41862217 0.65353451 0.06375049 0.01594784 0.62163883 0.65353451
1.05202619 1.0201305  0.81293118 0.57383618 1.57801884 0.00577716
2.16780286 0.00577716 0.5419405  0.17534451 0.00577716 0.70133717
0.661488   0.00577716 0.00577716]
```

- To Define an outlier threshold value is hoosen whcih is generally 3.

In [45]:
```
threshold=2.5
# Position of outlier
print(np.where(z>2.5))
```

```
(array([  2,  13,  81,  96, 114, 178, 235, 304], dtype=int64),)
```

- This points are acting as outliers.

Data Visualization

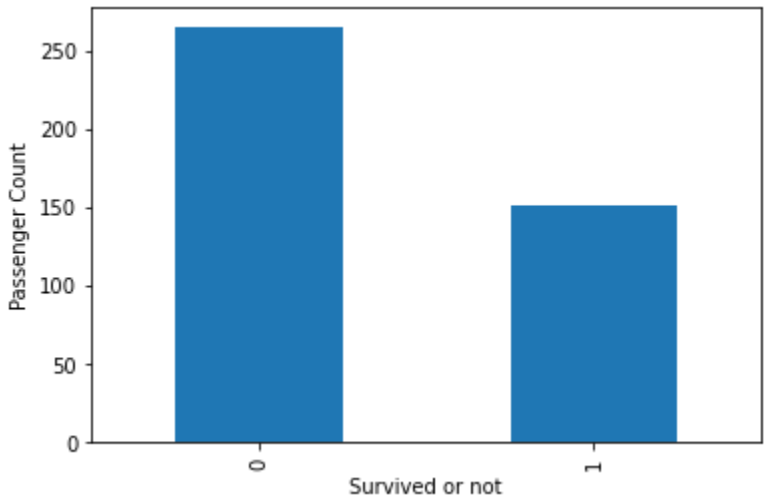
Visualization of 'Survived'[Target column]

In [46]:
```
df.Survived.value_counts()
```

Out[46]:
```
0    265
1    152
Name: Survived, dtype: int64
```

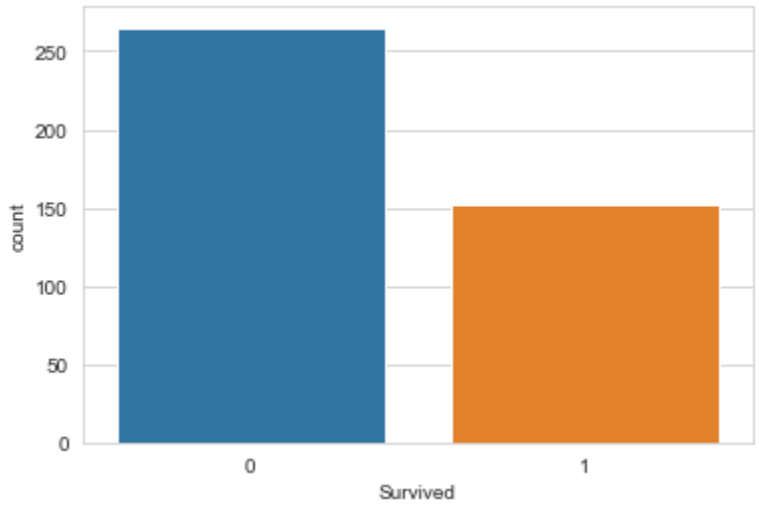
In [47]:
```
plt = df.Survived.value_counts().plot(kind='bar')
plt.set_xlabel('Survived or not')
plt.set_ylabel('Passenger Count')
```

Out[47]:
```
Text(0, 0.5, 'Passenger Count')
```



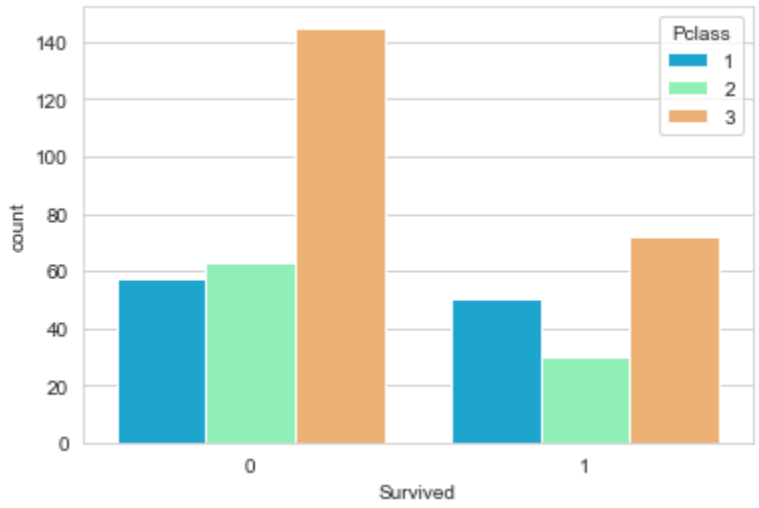
In [48]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=df)
```

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x201e5e1bcd0>



In [49]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=df,palette='rainbow')
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x201e5f1c520>

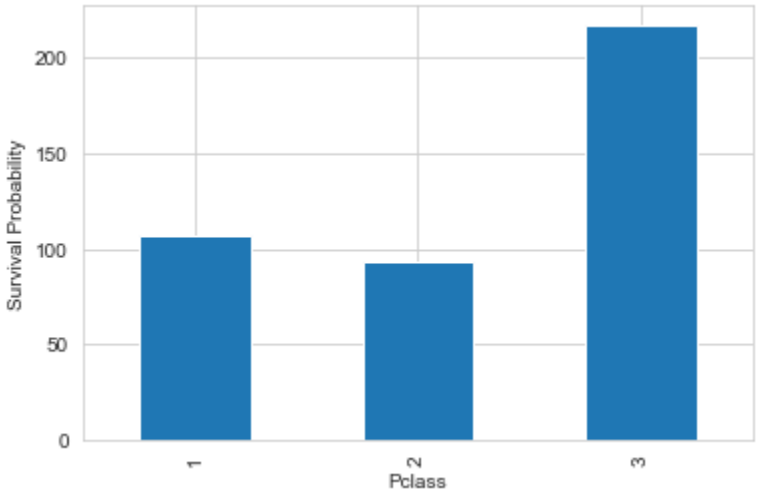


Plass

- Majority of them are from 3rd class

In [50]:
```
plt = df.Pclass.value_counts().sort_index().plot(kind='bar', title='')
plt.set_xlabel('Pclass')
plt.set_ylabel('Survival Probability')
```

Out[50]:
```
Text(0, 0.5, 'Survival Probability')
```



In [51]:
```
df[['Pclass', 'Survived']].groupby('Pclass').count()
```

Out[51]:

Pclass	Survived
1	107
2	93
3	217

In [52]:
```
df[['Pclass', 'Survived']].groupby('Pclass').sum()
```

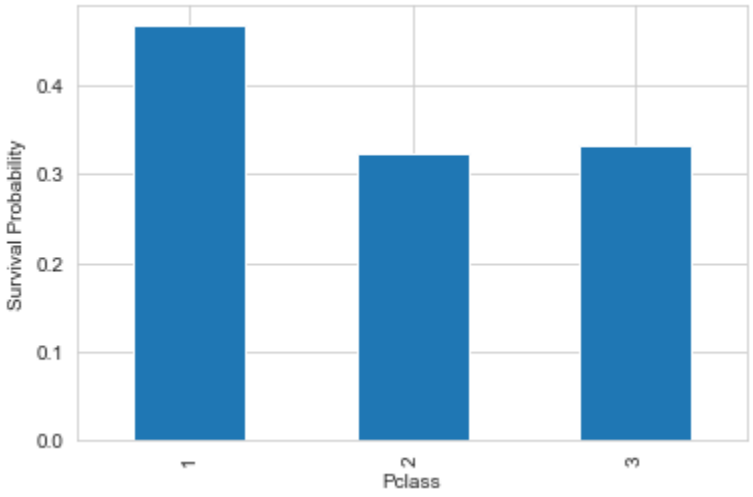
Out[52]:

Pclass	Survived
1	50
2	30
3	72

Plass survival probability

In [53]:
```
plt = df[['Pclass', 'Survived']].groupby('Pclass').mean().Survived.plot(kind='bar')
plt.set_xlabel('Pclass')
plt.set_ylabel('Survival Probability')
```

Out[53]:
```
Text(0, 0.5, 'Survival Probability')
```



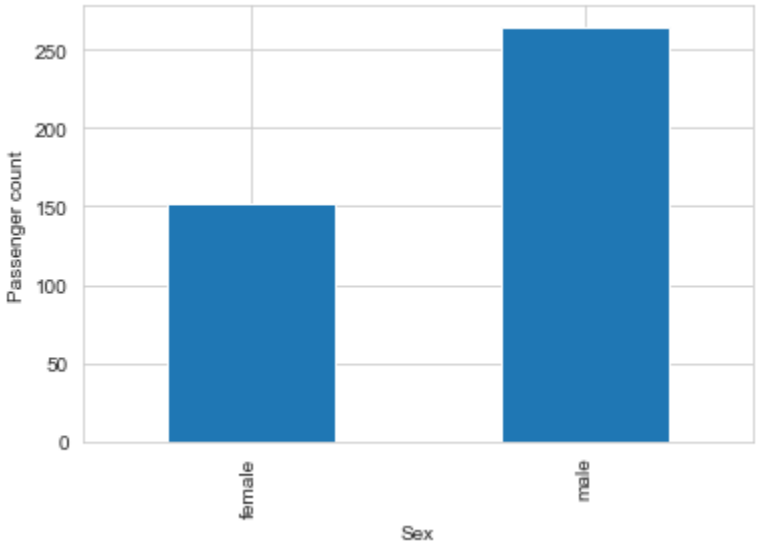
- From the above results, we can say that, 1st class has high chance of surviving than the other two classes.

Sex

- majority of them are male

In [54]:
```
plt = df.Sex.value_counts().sort_index().plot(kind='bar')
plt.set_xlabel('Sex')
plt.set_ylabel('Passenger count')
```

Out[54]:
```
Text(0, 0.5, 'Passenger count')
```

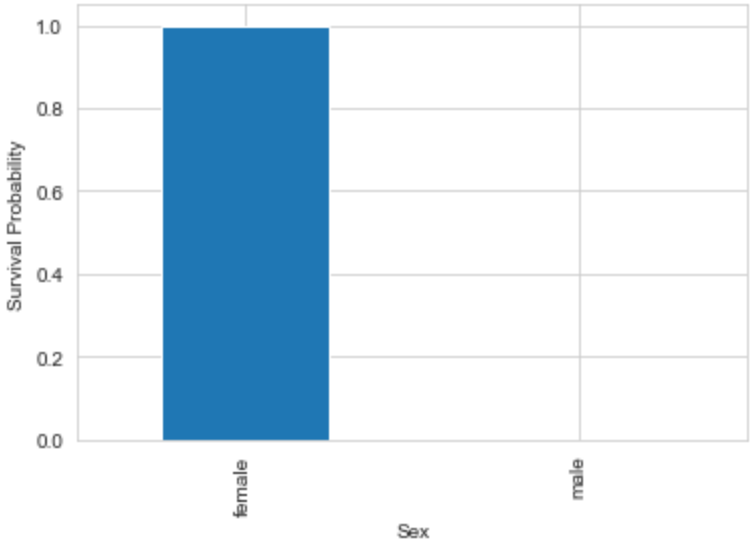


Sex Survival Probability

- As we see, the survival probaility for Female is more. They might have given more priority to female than male.

In [55]:
```
plt = df[['Sex', 'Survived']].groupby('Sex').mean().Survived.plot(kind='bar')
plt.set_xlabel('Sex')
plt.set_ylabel('Survival Probability')
```

Out[55]:
```
Text(0, 0.5, 'Survival Probability')
```

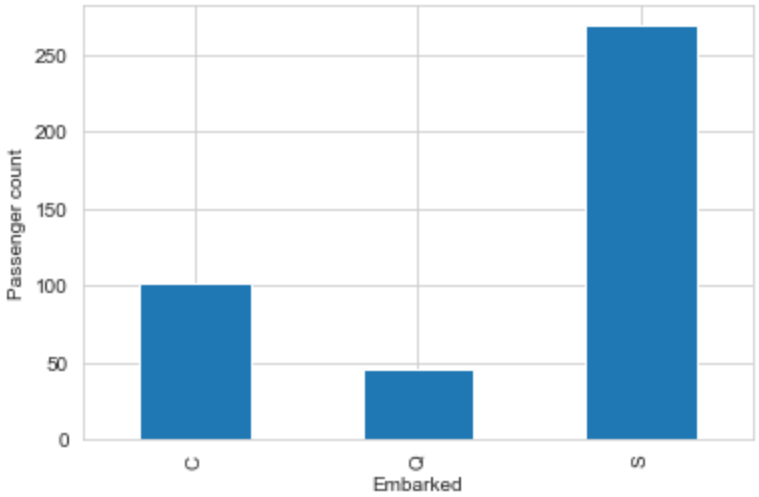


Embarked

- Most of them are from Southampton(S).

In [56]:
```
plt = df.Embarked.value_counts().sort_index().plot(kind='bar')
plt.set_xlabel('Embarked')
plt.set_ylabel('Passenger count')
```

Out[56]:
```
Text(0, 0.5, 'Passenger count')
```

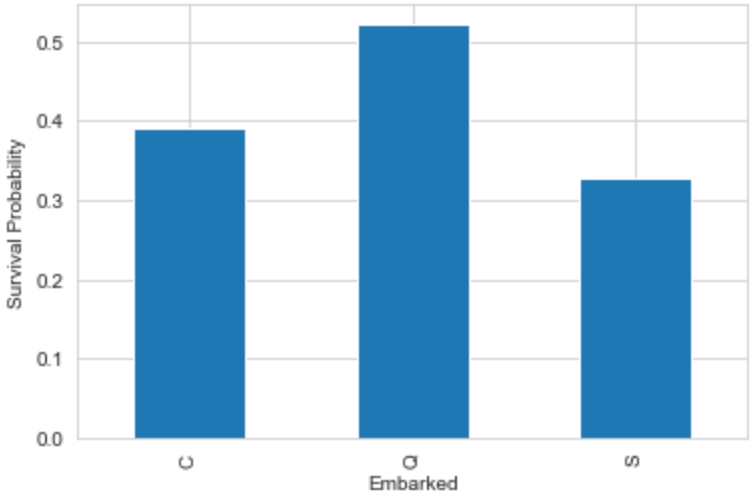


Embarked-Survival Probability

- Survival probability: C > Q > S

In [57]:
```
plt = df[['Embarked', 'Survived']].groupby('Embarked').mean().Survived.plot(kind='bar')
plt.set_xlabel('Embarked')
plt.set_ylabel('Survival Probability')
```

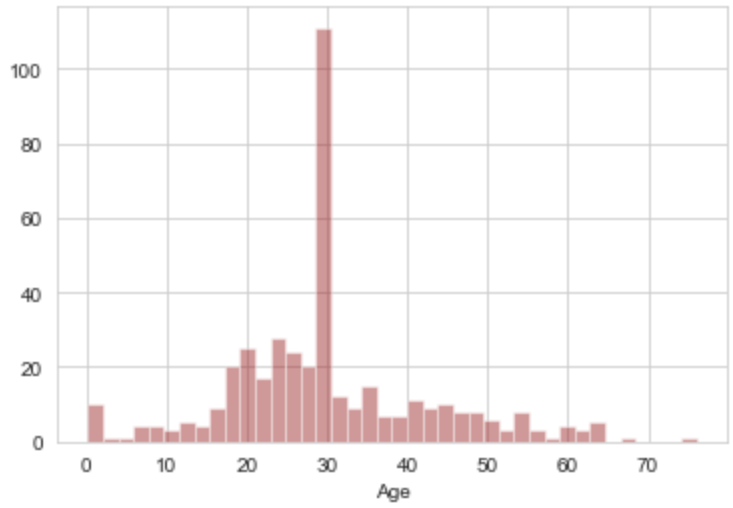
Out[57]:
```
Text(0, 0.5, 'Survival Probability')
```



Age

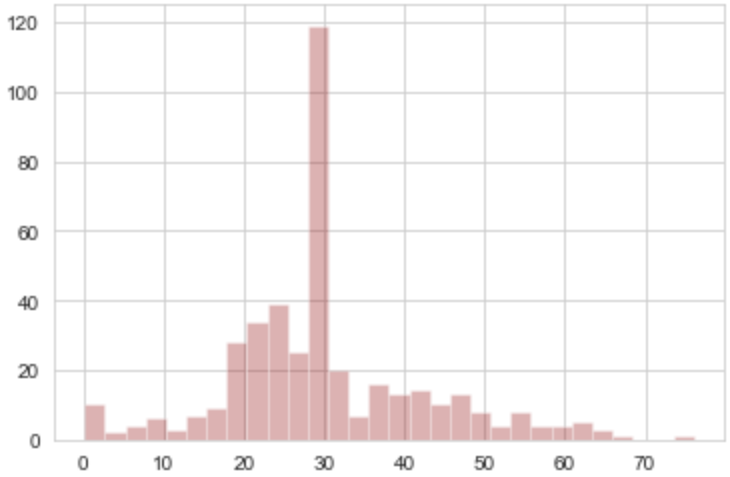
In [58]:
```
sns.distplot(df['Age'].dropna(),kde=False,color='darkred',bins=40)
```

Out[58]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e71495e0>
```



In [59]:
```
df['Age'].hist(bins=30,color='darkred',alpha=0.3)
```

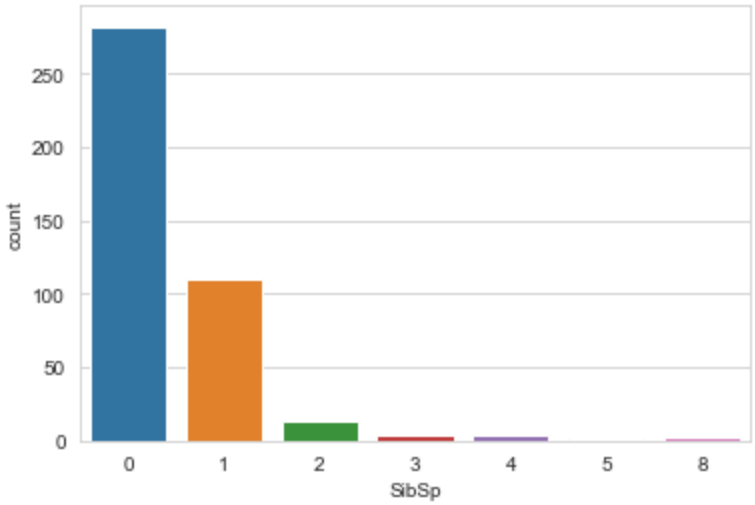
Out[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e71e5280>
```



SibSp - Siblings/Spouse

In [60]:
```
sns.countplot(x='SibSp',data=df)
```

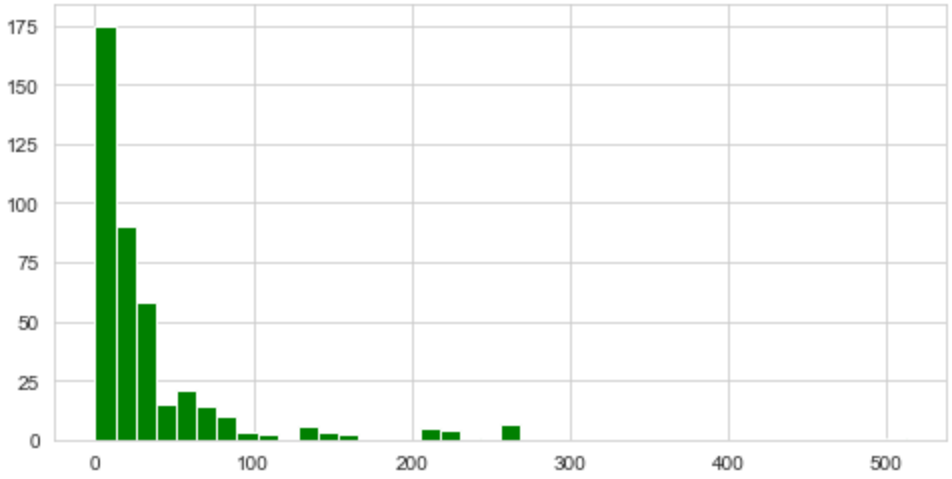
Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e70026a0>
```



Fare

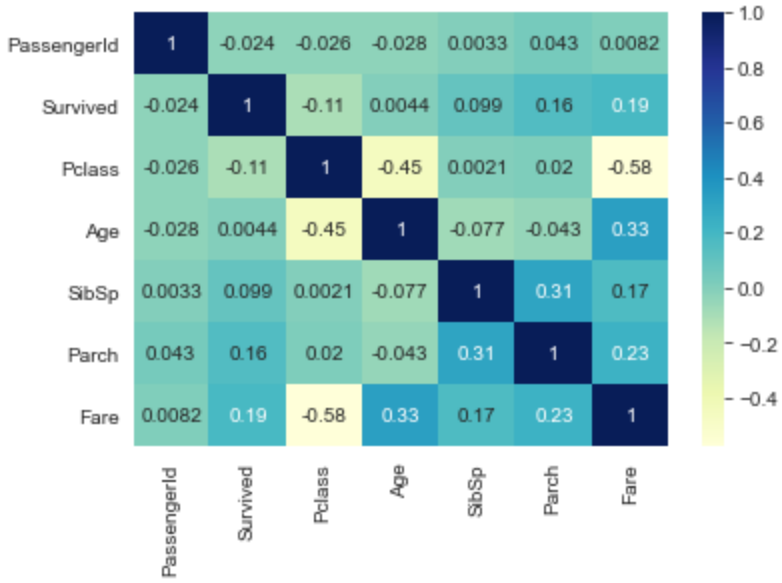
In [61]:
```
df['Fare'].hist(color='green',bins=40,figsize=(8,4),label='Fare')
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e7301280>
```



Correlation between columns

In [62]:
```
dataplot = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```



- There are no very highly correlated columns.

Box Plot

In [63]:
```
plt.boxplot(df['Age'])
```

Out[63]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x201e73bbc70>,
  <matplotlib.lines.Line2D at 0x201e71d36d0>],
 'caps': [<matplotlib.lines.Line2D at 0x201e73c8910>,
  <matplotlib.lines.Line2D at 0x201e73cfc70>],
 'boxes': [<matplotlib.lines.Line2D at 0x201e738d460>],
 'medians': [<matplotlib.lines.Line2D at 0x201e73cfee0>],
 'fliers': [<matplotlib.lines.Line2D at 0x201e7494280>],
 'means': []}
```

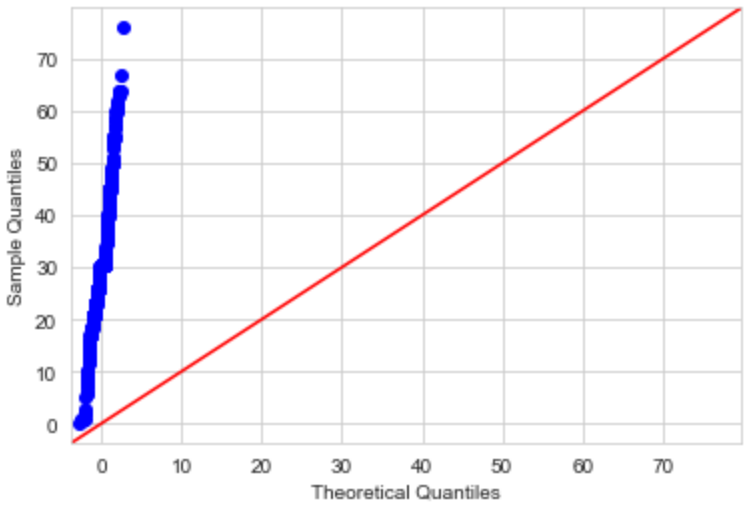
In [64]:
```
plt.boxplot(df['PassengerId'])
```

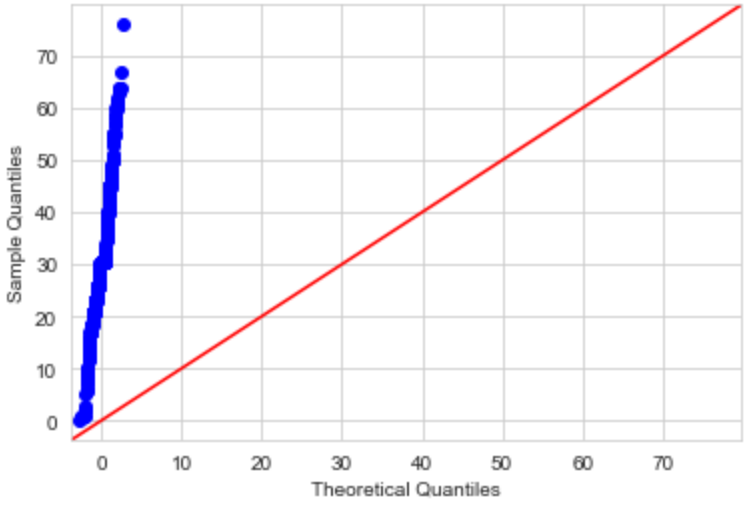
Out[64]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x201e7494bb0>,
  <matplotlib.lines.Line2D at 0x201e7494dc0>],
 'caps': [<matplotlib.lines.Line2D at 0x201e74a0100>,
  <matplotlib.lines.Line2D at 0x201e74a0460>],
 'boxes': [<matplotlib.lines.Line2D at 0x201e7494940>],
 'medians': [<matplotlib.lines.Line2D at 0x201e74a07c0>],
 'fliers': [<matplotlib.lines.Line2D at 0x201e74a0ac0>],
 'means': []}
```

Quantile-Quantile plot

In [65]:
```
## Quantile plot
sm.qqplot(df['Age'],line='45')
```

Out[65]:





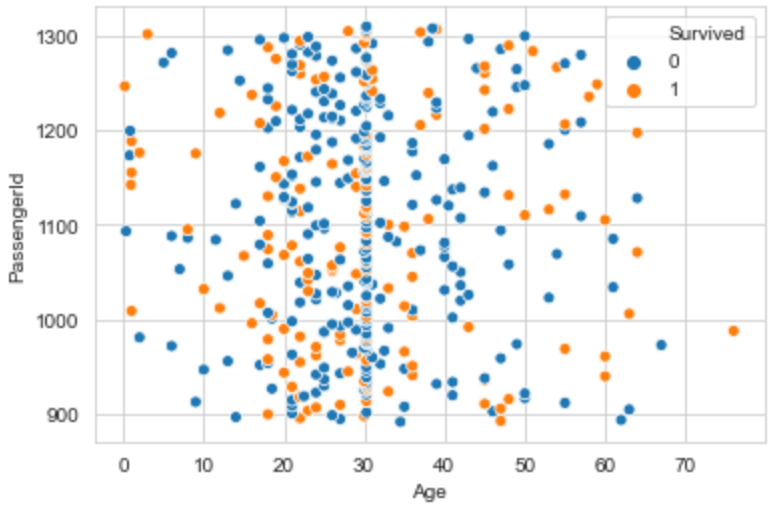
Scatter Plot

In [66]:
```
## scatter plot between Age and PassengerId columns
plt.scatter(df['Age'],df['PassengerId'])
```

Out[66]:
```
<matplotlib.collections.PathCollection at 0x201e73ab3a0>
```

In [67]:
```
## scatter plot between Age and PassengerId columns
sns.scatterplot(x='Age', y='PassengerId',
                hue='Survived', data=df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-67-3b155aa0304c> in <module>
      6 plt.legend(bbox_to_anchor=(1, 1), loc=2)
      7
----> 8 plt.show()

AttributeError: 'AxesSubplot' object has no attribute 'show'
```



In [ ]:
```
## scatter plot between Age and Fare columns
sns.scatterplot(x='Age', y='Fare',
                hue='Survived', data=df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

In [ ]:
```
## scatter plot between Age and Fare columns
plt.scatter(df['Age'],df['Fare'])
```

Qus.2 : Implementation of Apriori algorithm for finding Frequent Itemsets and Association Rules.

- For this Question, I have used the bakery data with the list of items bought by customers.
- And, By using Apriori Algorithm, I will try to find the frequent itemsets and Association rules between the items which are mostly buy together by the customers

In [68]:
```
bakery_data=pd.read_csv("C:/Users/Mycomputer/Documents/MCA_BHU/data mining/DM_Assignment/datasets/breadbasket_dataset.csv",encoding='utf-8-sig')
```

In [69]:
```
bakery_data.head()
```

Out[69]:

	Transaction	Item	date_time	period_day	weekday_weekend
0	1	Bread	30-10-2016 09:58	morning	weekend
1	2	Scandinavian	30-10-2016 10:05	morning	weekend
2	2	Scandinavian	30-10-2016 10:05	morning	weekend
3	3	Hot chocolate	30-10-2016 10:07	morning	weekend
4	3	Jam	30-10-2016 10:07	morning	weekend

In [70]:
```
bakery_data.describe()
```

Out[70]:

	Transaction
count	20507.000000
mean	4976.202370
std	2796.203001
min	1.000000
25%	2552.000000
50%	5137.000000
75%	7357.000000
max	9684.000000

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20507 entries, 0 to 20506
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Transaction      20507 non-null  int64 
 1   Item             20507 non-null  object
 2   date_time        20507 non-null  object
 3   period_day       20507 non-null  object
 4   weekday_weekend  20507 non-null  object
dtypes: int64(1), object(4)
memory usage: 801.2+ KB
```

In [72]:
```
 bakery_data.dtypes
```

Out[72]:
```
Transaction         int64
Item               object
date_time          object
period_day         object
weekday_weekend    object
dtype: object
```

- There is no missing value in the data.

In [73]:
```
bakery_data.nunique()
```

Out[73]:
```
Transaction        9465
Item                 94
date_time          9182
period_day            4
weekday_weekend       2
dtype: int64
```

In [74]:
```
bakery_data['Transaction'].nunique()
```

Out[74]:
```
9465
```

- There are 9465 unique customers.

In [75]:
```
# Extracting date
bakery_data['date'] = pd.to_datetime(bakery_data['date_time'],errors='coerce')
```

In [76]:
```
# Extracting time
bakery_data['time'] = bakery_data['date'].dt.time
```

In [77]:
```
# Extracting month and replacing it with text
bakery_data['month'] = bakery_data['date'].dt.month
bakery_data['month'] = bakery_data['month'].replace((1,2,3,4,5,6,7,8,9,10,11,12
),
                                                    ('January','February','March','April'
,'May','June','July','August',
                                                    'September','October','November','Dec
ember'))
```

In [78]:
```
# Extracting hour
bakery_data['hour'] = bakery_data['date'].dt.hour
# Replacing hours with text
hour_in_num = (1,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)
hour_in_obj = ('1-2','7-8','8-9','9-10','10-11','11-12','12-13','13-14','14-15'
,
               '15-16','16-17','17-18','18-19','19-20','20-21','21-22','22-23',
'23-24')
bakery_data['hour'] = bakery_data['hour'].replace(hour_in_num, hour_in_obj)
```

In [79]:
```
# Extracting weekday and replacing it with text
bakery_data['weekday'] = bakery_data['date'].dt.weekday
bakery_data['weekday'] = bakery_data['weekday'].replace((0,1,2,3,4,5,6),
                                                        ('Monday','Tuesday','Wednesday','Thur
sday','Friday','Saturday','Sunday'))
```

In [80]:
```
# dropping date_time column
bakery_data.drop('date_time', axis = 1, inplace = True)
```

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

Out[81]:

	Transaction	Item	period_day	weekday_weekend	date	time	month	hour	weekday
0	1	Bread	morning	weekend	2016-10-30 09:58:00	09:58:00	October	9-10	Sunday
1	2	Scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
2	2	Scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
3	3	Hot chocolate	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday
4	3	Jam	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday

In [82]:
```
# cleaning the item column
bakery_data['Item'] = bakery_data['Item'].str.strip()
bakery_data['Item'] = bakery_data['Item'].str.lower()
```

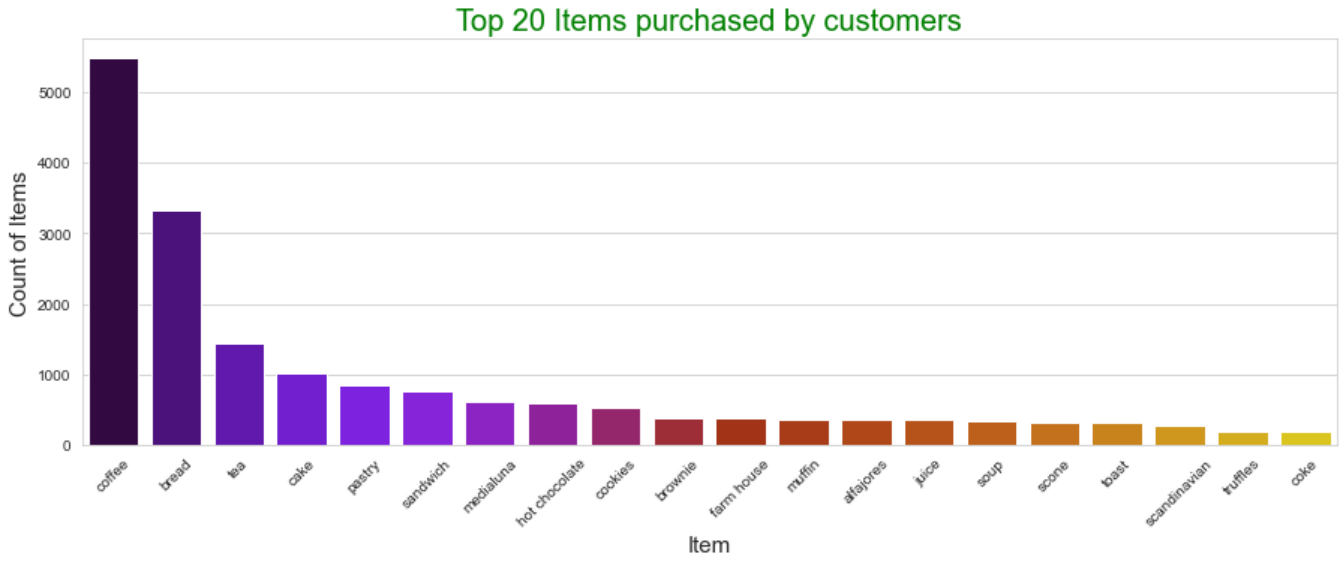
In [87]:
```
bakery_data.head()
```

Out[87]:

	Transaction	Item	period_day	weekday_weekend	date	time	month	hour	weekday
0	1	bread	morning	weekend	2016-10-30 09:58:00	09:58:00	October	9-10	Sunday
1	2	scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
2	2	scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
3	3	hot chocolate	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday
4	3	jam	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday

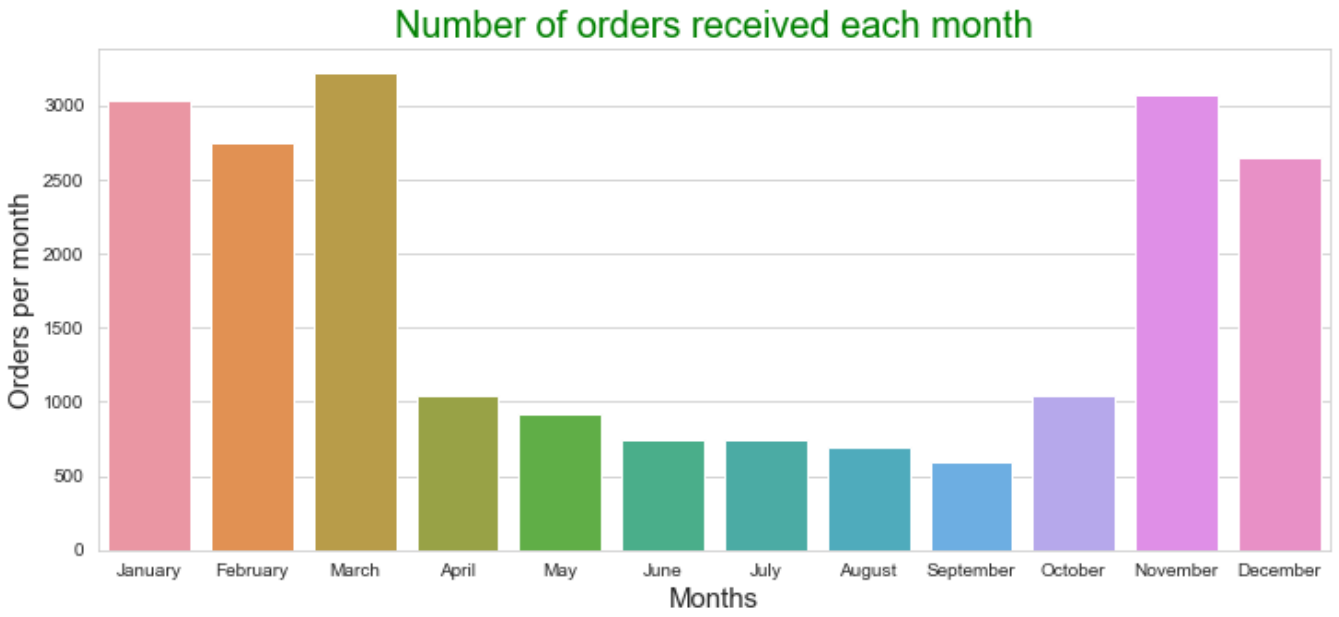
Data Visualization

In [88]:
```
plt.figure(figsize=(15,5))
sns.barplot(x = bakery_data.Item.value_counts().head(20).index, y = bakery_data
.Item.value_counts().head(20).values, palette='gnuplot')
plt.xlabel('Item', size = 15)
plt.xticks(rotation=45)
plt.ylabel('Count of Items', size = 15)
plt.title('Top 20 Items purchased by customers', color = 'green', size = 20)
plt.show()
```



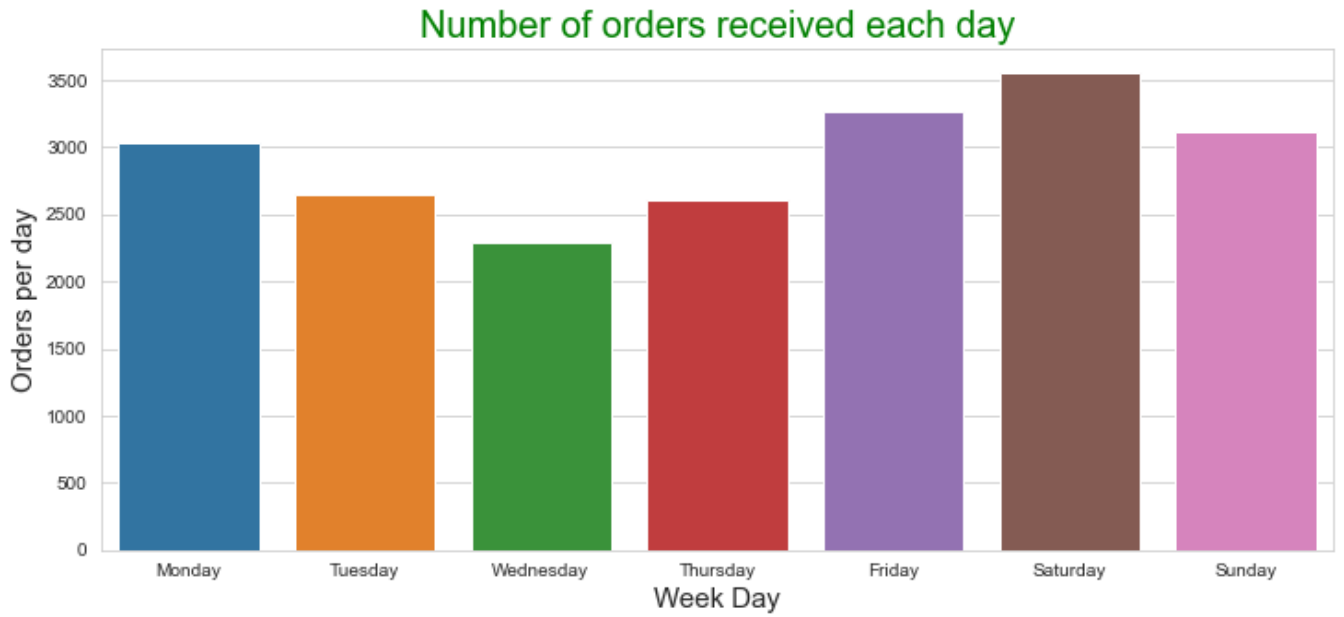
- Coffee has the highest transactions.
- Coke is the 20th most buyed product.

In [89]:
```
monthTran = bakery_data.groupby('month')['Transaction'].count().reset_index()
monthTran.loc[:,"monthorder"] = [4,8,12,2,1,7,6,3,5,11,10,9]
monthTran.sort_values("monthorder",inplace=True)

plt.figure(figsize=(12,5))
sns.barplot(data = monthTran, x = "month", y = "Transaction")
plt.xlabel('Months', size = 15)
plt.ylabel('Orders per month', size = 15)
plt.title('Number of orders received each month', color = 'green', size = 20)
plt.show()


plt.show()
```



- Most transactions were in **March, January, February, November, December**

In [90]:
```
weekTran = bakery_data.groupby('weekday')['Transaction'].count().reset_index()
weekTran.loc[:,"weekorder"] = [4,0,5,6,3,1,2]
weekTran.sort_values("weekorder",inplace=True)

plt.figure(figsize=(12,5))
sns.barplot(data = weekTran, x = "weekday", y = "Transaction")
plt.xlabel('Week Day', size = 15)
plt.ylabel('Orders per day', size = 15)
plt.title('Number of orders received each day', color = 'green', size = 20)
plt.show()


plt.show()
```

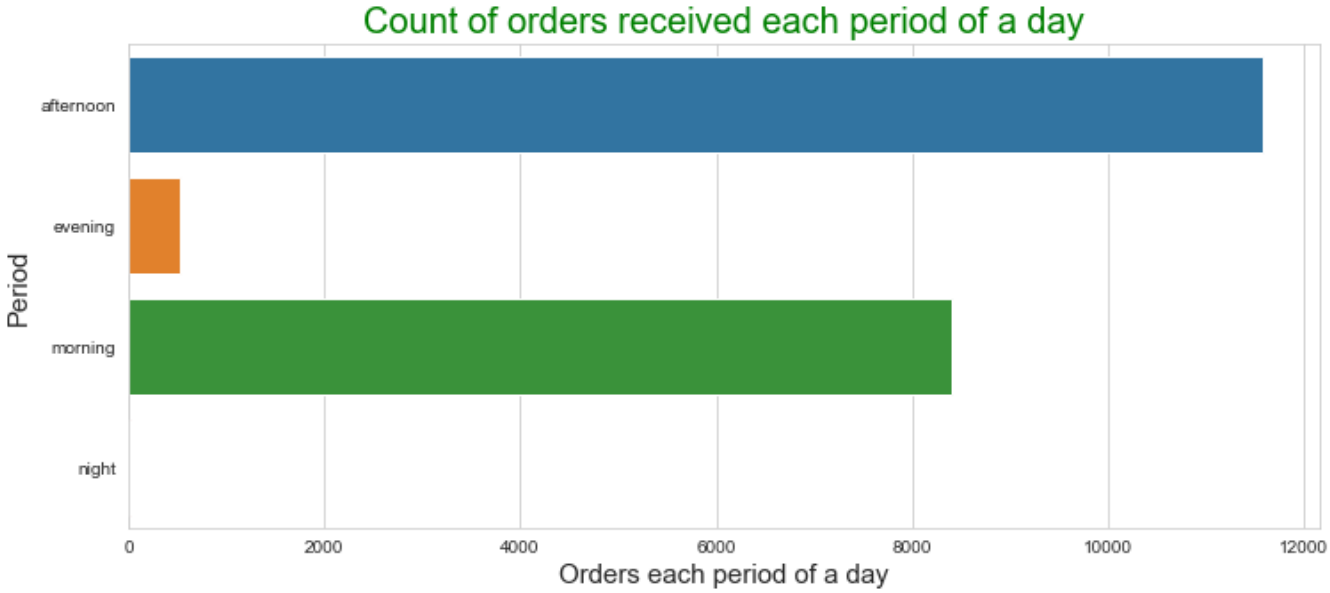


People order more on weekends.

In [91]:
```
hourTran = bakery_data.groupby('hour')['Transaction'].count().reset_index()
hourTran.loc[:,"hourorder"] = [1,10,11,12,13,14,15,16,17,18,19,20,21,22,23,7,8,
9]
hourTran.sort_values("hourorder",inplace=True)

plt.figure(figsize=(12,5))
sns.barplot(data = hourTran, x = "Transaction", y = "hour")
plt.ylabel('Hours', size = 15)
plt.xlabel('Orders each hour', size = 15)
plt.title('Count of orders received each hour', color = 'green', size = 20)
plt.show()
```

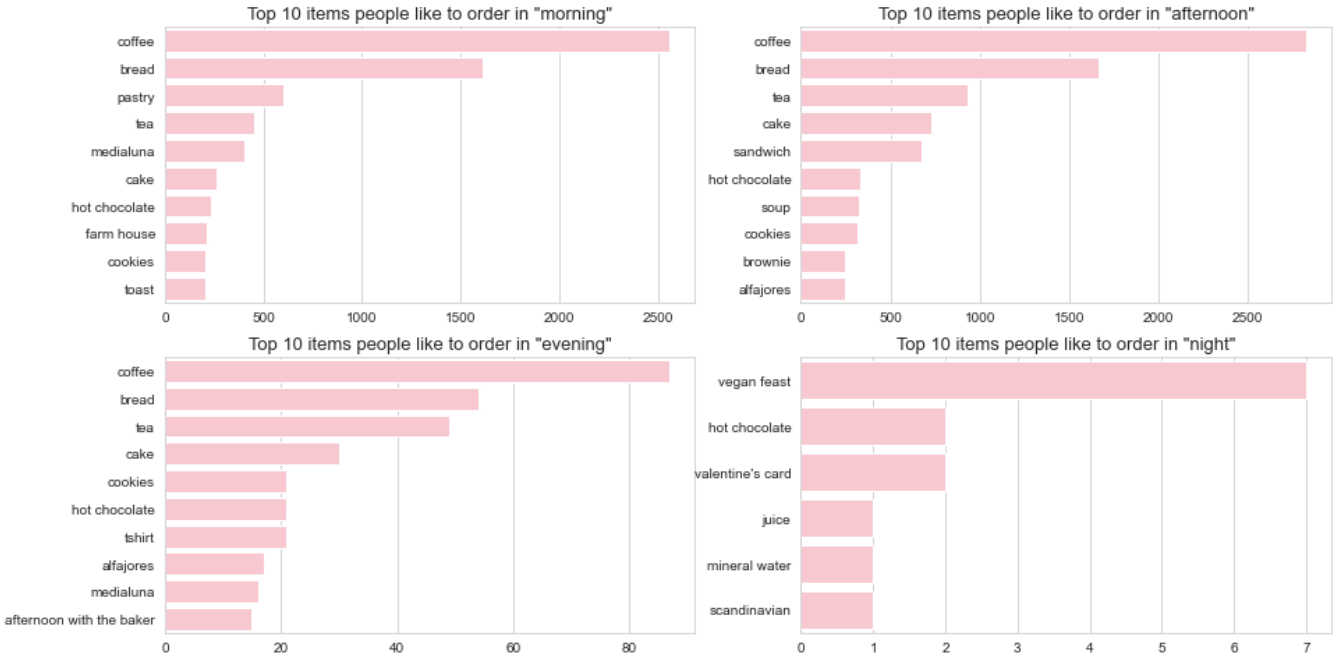
People order more during the afternoon, since there are a lot of maximum order percentage between 12-5.

In [92]:
```
dayTran = bakery_data.groupby('period_day')['Transaction'].count().reset_index
()
# dayTran.loc[:,"hourorder"] = [1,10,11,12,13,14,15,16,17,18,19,20,21,22,23,7,
8,9]
# dayTran.sort_values("hourorder",inplace=True)

plt.figure(figsize=(12,5))
sns.barplot(data = dayTran, x = "Transaction", y = "period_day")
plt.ylabel('Period', size = 15)
plt.xlabel('Orders each period of a day', size = 15)
plt.title('Count of orders received each period of a day', color = 'green', siz
e = 20)
plt.show()
```



People prefer to order in the morning and afternoon.

In [93]:
```
data = bakery_data.groupby(['period_day','Item'])['Transaction'].count().reset_
index().sort_values(['period_day','Transaction'],ascending=False)
day = ['morning','afternoon','evening','night']

plt.figure(figsize=(15,8))
for i,j in enumerate(day):
    plt.subplot(2,2,i+1)
    df1 = data[data.period_day==j].head(10)
    sns.barplot(data=df1, y=df1.Item, x=df1.Transaction, color='pink')
    plt.xlabel('')
    plt.ylabel('')
    plt.title('Top 10 items people like to order in "{}"'.format(j), size=13)

plt.show()
```



Apriori Algorithm

In [94]:
```
transactions_str = bakery_data.groupby(['Transaction', 'Item'])['Item'].count()
.reset_index(name ='Count')
transactions_str
```

Out[94]:

	Transaction	Item	Count
0	1	bread	1
1	2	scandinavian	2
2	3	cookies	1
3	3	hot chocolate	1
4	3	jam	1
...	...	...	...
18882	9682	tacos/fajita	1
18883	9682	tea	1
18884	9683	coffee	1
18885	9683	pastry	1
18886	9684	smoothies	1

18887 rows × 3 columns

- *Making a mxn matrice where m=transaction and n=items and each row represents whether the item was in the transaction or not*

In [95]:
```
my_basket = transactions_str.pivot_table(index='Transaction', columns='Item', v
alues='Count', aggfunc='sum').fillna(0)

my_basket.head()
```

Out[95]:

Item	adjustment	afternoon with the baker	alfajores	argentina night	art tray	bacon	baguette	bakewell	bare popcorn	basket
Transaction										
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 94 columns

- making a function which returns 0 or 1
- 0 means item was not in that transaction, 1 means item present in that transaction

In [96]:
```
def encode(x):
    if x<=0:
        return 0
    if x>=1:
        return 1

# applying the function to the dataset

my_basket_sets = my_basket.applymap(encode)
my_basket_sets.head()
```

Out[96]:

Item	adjustment	afternoon with the baker	alfajores	argentina night	art tray	bacon	baguette	bakewell	bare popcorn	basket
Transaction										
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0

5 rows × 94 columns

- **Generating frequent_item sets with min_support=0.01**

Using the 'apriori algorithm' with **min_support=0.01 (1% of 9465)** It means the item should be present in atleast 94 transaction out of 9465 transactions only when we considered that item in frequent itemset

In [97]:
```
frequent_items1 = apriori(my_basket_sets, min_support = 0.01,use_colnames = True)
frequent_items1
```

Out[97]:

	support	itemsets
0	0.036344	(alfajores)
1	0.016059	(baguette)
2	0.327205	(bread)
3	0.040042	(brownie)
4	0.103856	(cake)
...	...	...
56	0.023666	(toast, coffee)
57	0.014369	(sandwich, tea)
58	0.010037	(coffee, cake, bread)
59	0.011199	(coffee, pastry, bread)
60	0.010037	(tea, coffee, cake)

61 rows × 2 columns

- **Generating Association Rules from Frequent Itemsets-1.**

now making the rules from frequent itemset generated above

In [98]:
```
rules1 = association_rules(frequent_items1, metric = "lift", min_threshold = 1)
rules1.sort_values('confidence', ascending = False, inplace = True)
rules1
```

Out[98]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
30	(toast)	(coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.76458
28	(spanish brunch)	(coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.002189	1.30023
18	(medialuna)	(coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.21087
23	(pastry)	(coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.16468
0	(alfajores)	(coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.002264	1.13564
16	(juice)	(coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.002154	1.11991
24	(sandwich)	(coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.003877	1.11538
7	(cake)	(coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.10266
27	(scone)	(coffee)	0.034548	0.478394	0.018067	0.522936	1.093107	0.001539	1.09336
12	(cookies)	(coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.002179	1.08317
14	(hot chocolate)	(coffee)	0.058320	0.478394	0.029583	0.507246	1.060311	0.001683	1.05855
5	(brownie)	(coffee)	0.040042	0.478394	0.019651	0.490765	1.025860	0.000495	1.02429
21	(muffin)	(coffee)	0.038457	0.478394	0.018806	0.489011	1.022193	0.000408	1.02077
2	(pastry)	(bread)	0.086107	0.327205	0.029160	0.338650	1.034977	0.000985	1.01730
11	(cake)	(tea)	0.103856	0.142631	0.023772	0.228891	1.604781	0.008959	1.11186
38	(tea, coffee)	(cake)	0.049868	0.103856	0.010037	0.201271	1.937977	0.004858	1.12196
32	(sandwich)	(tea)	0.071844	0.142631	0.014369	0.200000	1.402222	0.004122	1.07171
8	(hot chocolate)	(cake)	0.058320	0.103856	0.011410	0.195652	1.883874	0.005354	1.11412
39	(coffee, cake)	(tea)	0.054728	0.142631	0.010037	0.183398	1.285822	0.002231	1.04992
10	(tea)	(cake)	0.142631	0.103856	0.023772	0.166667	1.604781	0.008959	1.07537
37	(pastry)	(coffee, bread)	0.086107	0.090016	0.011199	0.130061	1.444872	0.003448	1.04603
36	(coffee, bread)	(pastry)	0.090016	0.086107	0.011199	0.124413	1.444872	0.003448	1.04374
6	(coffee)	(cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.01190
34	(coffee, bread)	(cake)	0.090016	0.103856	0.010037	0.111502	1.073621	0.000688	1.00860
9	(cake)	(hot chocolate)	0.103856	0.058320	0.011410	0.109868	1.883874	0.005354	1.05791
33	(tea)	(sandwich)	0.142631	0.071844	0.014369	0.100741	1.402222	0.004122	1.03213
22	(coffee)	(pastry)	0.478394	0.086107	0.047544	0.099382	1.154168	0.006351	1.01474
35	(cake)	(coffee, bread)	0.103856	0.090016	0.010037	0.096643	1.073621	0.000688	1.00733
41	(cake)	(tea, coffee)	0.103856	0.049868	0.010037	0.096643	1.937977	0.004858	1.05177
3	(bread)	(pastry)	0.327205	0.086107	0.029160	0.089119	1.034977	0.000985	1.00330
25	(coffee)	(sandwich)	0.478394	0.071844	0.038246	0.079947	1.112792	0.003877	1.00880
19	(coffee)	(medialuna)	0.478394	0.061807	0.035182	0.073542	1.189878	0.005614	1.01266
40	(tea)	(coffee, cake)	0.142631	0.054728	0.010037	0.070370	1.285822	0.002231	1.01682

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
15	(coffee)	(hot chocolate)	0.478394	0.058320	0.029583	0.061837	1.060311	0.001683	1.00374
13	(coffee)	(cookies)	0.478394	0.054411	0.028209	0.058966	1.083723	0.002179	1.00484
31	(coffee)	(toast)	0.478394	0.033597	0.023666	0.049470	1.472431	0.007593	1.01669
17	(coffee)	(juice)	0.478394	0.038563	0.020602	0.043065	1.116750	0.002154	1.00470
1	(coffee)	(alfajores)	0.478394	0.036344	0.019651	0.041078	1.130235	0.002264	1.00493
4	(coffee)	(brownie)	0.478394	0.040042	0.019651	0.041078	1.025860	0.000495	1.00108
20	(coffee)	(muffin)	0.478394	0.038457	0.018806	0.039311	1.022193	0.000408	1.00088
26	(coffee)	(scone)	0.478394	0.034548	0.018067	0.037765	1.093107	0.001539	1.00334
29	(coffee)	(spanish brunch)	0.478394	0.018172	0.010882	0.022747	1.251766	0.002189	1.00468

- **Arranging the data from highest to lowest with respect to 'confidence' for Frequent Itemset-1**

In [99]:
```
rules1.sort_values('confidence', ascending=False)
```

Out[99]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
30	(toast)	(coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.76458
28	(spanish brunch)	(coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.002189	1.30023
18	(medialuna)	(coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.21087
23	(pastry)	(coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.16468
0	(alfajores)	(coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.002264	1.13564
16	(juice)	(coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.002154	1.11991
24	(sandwich)	(coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.003877	1.11538
7	(cake)	(coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.10266
27	(scone)	(coffee)	0.034548	0.478394	0.018067	0.522936	1.093107	0.001539	1.09336
12	(cookies)	(coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.002179	1.08317
14	(hot chocolate)	(coffee)	0.058320	0.478394	0.029583	0.507246	1.060311	0.001683	1.05855
5	(brownie)	(coffee)	0.040042	0.478394	0.019651	0.490765	1.025860	0.000495	1.02429
21	(muffin)	(coffee)	0.038457	0.478394	0.018806	0.489011	1.022193	0.000408	1.02077
2	(pastry)	(bread)	0.086107	0.327205	0.029160	0.338650	1.034977	0.000985	1.01730
11	(cake)	(tea)	0.103856	0.142631	0.023772	0.228891	1.604781	0.008959	1.11186
38	(tea, coffee)	(cake)	0.049868	0.103856	0.010037	0.201271	1.937977	0.004858	1.12196
32	(sandwich)	(tea)	0.071844	0.142631	0.014369	0.200000	1.402222	0.004122	1.07171
8	(hot chocolate)	(cake)	0.058320	0.103856	0.011410	0.195652	1.883874	0.005354	1.11412
39	(coffee, cake)	(tea)	0.054728	0.142631	0.010037	0.183398	1.285822	0.002231	1.04992
10	(tea)	(cake)	0.142631	0.103856	0.023772	0.166667	1.604781	0.008959	1.07537
37	(pastry)	(coffee, bread)	0.086107	0.090016	0.011199	0.130061	1.444872	0.003448	1.04603
36	(coffee, bread)	(pastry)	0.090016	0.086107	0.011199	0.124413	1.444872	0.003448	1.04374
6	(coffee)	(cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.01190
34	(coffee, bread)	(cake)	0.090016	0.103856	0.010037	0.111502	1.073621	0.000688	1.00860
9	(cake)	(hot chocolate)	0.103856	0.058320	0.011410	0.109868	1.883874	0.005354	1.05791
33	(tea)	(sandwich)	0.142631	0.071844	0.014369	0.100741	1.402222	0.004122	1.03213
22	(coffee)	(pastry)	0.478394	0.086107	0.047544	0.099382	1.154168	0.006351	1.01474
35	(cake)	(coffee, bread)	0.103856	0.090016	0.010037	0.096643	1.073621	0.000688	1.00733
41	(cake)	(tea, coffee)	0.103856	0.049868	0.010037	0.096643	1.937977	0.004858	1.05177
3	(bread)	(pastry)	0.327205	0.086107	0.029160	0.089119	1.034977	0.000985	1.00330
25	(coffee)	(sandwich)	0.478394	0.071844	0.038246	0.079947	1.112792	0.003877	1.00880
19	(coffee)	(medialuna)	0.478394	0.061807	0.035182	0.073542	1.189878	0.005614	1.01266
40	(tea)	(coffee, cake)	0.142631	0.054728	0.010037	0.070370	1.285822	0.002231	1.01682

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
15	(coffee)	(hot chocolate)	0.478394	0.058320	0.029583	0.061837	1.060311	0.001683	1.00374
13	(coffee)	(cookies)	0.478394	0.054411	0.028209	0.058966	1.083723	0.002179	1.00484
31	(coffee)	(toast)	0.478394	0.033597	0.023666	0.049470	1.472431	0.007593	1.01669
17	(coffee)	(juice)	0.478394	0.038563	0.020602	0.043065	1.116750	0.002154	1.00470
1	(coffee)	(alfajores)	0.478394	0.036344	0.019651	0.041078	1.130235	0.002264	1.00493
4	(coffee)	(brownie)	0.478394	0.040042	0.019651	0.041078	1.025860	0.000495	1.00108
20	(coffee)	(muffin)	0.478394	0.038457	0.018806	0.039311	1.022193	0.000408	1.00088
26	(coffee)	(scone)	0.478394	0.034548	0.018067	0.037765	1.093107	0.001539	1.00334
29	(coffee)	(spanish brunch)	0.478394	0.018172	0.010882	0.022747	1.251766	0.002189	1.00468

- **Generating frequent_item sets with min_support=0.02**

In [100]:
```
frequent_items2 = apriori(my_basket_sets, min_support = 0.02,use_colnames = True)
frequent_items2
```

Out[100]:

	support	itemsets
0	0.036344	(alfajores)
1	0.327205	(bread)
2	0.040042	(brownie)
3	0.103856	(cake)
4	0.478394	(coffee)
5	0.054411	(cookies)
6	0.039197	(farm house)
7	0.058320	(hot chocolate)
8	0.038563	(juice)
9	0.061807	(medialuna)
10	0.038457	(muffin)
11	0.086107	(pastry)
12	0.071844	(sandwich)
13	0.029054	(scandinavian)
14	0.034548	(scone)
15	0.034443	(soup)
16	0.142631	(tea)
17	0.033597	(toast)
18	0.020285	(truffles)
19	0.023349	(cake, bread)
20	0.090016	(coffee, bread)
21	0.029160	(pastry, bread)
22	0.028104	(tea, bread)
23	0.054728	(coffee, cake)
24	0.023772	(tea, cake)
25	0.028209	(cookies, coffee)
26	0.029583	(hot chocolate, coffee)
27	0.020602	(juice, coffee)
28	0.035182	(medialuna, coffee)
29	0.047544	(coffee, pastry)
30	0.038246	(sandwich, coffee)
31	0.049868	(tea, coffee)
32	0.023666	(toast, coffee)

- **Generating Association Rules from Frequent Itemsets-1.**

In [101]:
```
rules2 = association_rules(frequent_items2, metric = "lift", min_threshold = 1)
rules2.sort_values('confidence', ascending = False, inplace = True)
rules2
```

Out[101]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
18	(toast)	(coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.76458
12	(medialuna)	(coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.21087
15	(pastry)	(coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.16468
10	(juice)	(coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.002154	1.11991
16	(sandwich)	(coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.003877	1.11538
3	(cake)	(coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.10266
6	(cookies)	(coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.002179	1.08317
8	(hot chocolate)	(coffee)	0.058320	0.478394	0.029583	0.507246	1.060311	0.001683	1.05855
0	(pastry)	(bread)	0.086107	0.327205	0.029160	0.338650	1.034977	0.000985	1.01730
5	(cake)	(tea)	0.103856	0.142631	0.023772	0.228891	1.604781	0.008959	1.11186
4	(tea)	(cake)	0.142631	0.103856	0.023772	0.166667	1.604781	0.008959	1.07537
2	(coffee)	(cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.01190
14	(coffee)	(pastry)	0.478394	0.086107	0.047544	0.099382	1.154168	0.006351	1.01474
1	(bread)	(pastry)	0.327205	0.086107	0.029160	0.089119	1.034977	0.000985	1.00330
17	(coffee)	(sandwich)	0.478394	0.071844	0.038246	0.079947	1.112792	0.003877	1.00880
13	(coffee)	(medialuna)	0.478394	0.061807	0.035182	0.073542	1.189878	0.005614	1.01266
9	(coffee)	(hot chocolate)	0.478394	0.058320	0.029583	0.061837	1.060311	0.001683	1.00374
7	(coffee)	(cookies)	0.478394	0.054411	0.028209	0.058966	1.083723	0.002179	1.00484
19	(coffee)	(toast)	0.478394	0.033597	0.023666	0.049470	1.472431	0.007593	1.01669
11	(coffee)	(juice)	0.478394	0.038563	0.020602	0.043065	1.116750	0.002154	1.00470

- **Arranging the data from highest to lowest with respect to 'confidence' for Frequent Itemset-2**

In [102]:
```
rules2.sort_values('confidence', ascending=False)
```

Out[102]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
18	(toast)	(coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.007593	1.76458
12	(medialuna)	(coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.005614	1.21087
15	(pastry)	(coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.006351	1.16468
10	(juice)	(coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.002154	1.11991
16	(sandwich)	(coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.003877	1.11538
3	(cake)	(coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.005044	1.10266
6	(cookies)	(coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.002179	1.08317
8	(hot chocolate)	(coffee)	0.058320	0.478394	0.029583	0.507246	1.060311	0.001683	1.05855
0	(pastry)	(bread)	0.086107	0.327205	0.029160	0.338650	1.034977	0.000985	1.01730
5	(cake)	(tea)	0.103856	0.142631	0.023772	0.228891	1.604781	0.008959	1.11186
4	(tea)	(cake)	0.142631	0.103856	0.023772	0.166667	1.604781	0.008959	1.07537
2	(coffee)	(cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.005044	1.01190
14	(coffee)	(pastry)	0.478394	0.086107	0.047544	0.099382	1.154168	0.006351	1.01474
1	(bread)	(pastry)	0.327205	0.086107	0.029160	0.089119	1.034977	0.000985	1.00330
17	(coffee)	(sandwich)	0.478394	0.071844	0.038246	0.079947	1.112792	0.003877	1.00880
13	(coffee)	(medialuna)	0.478394	0.061807	0.035182	0.073542	1.189878	0.005614	1.01266
9	(coffee)	(hot chocolate)	0.478394	0.058320	0.029583	0.061837	1.060311	0.001683	1.00374
7	(coffee)	(cookies)	0.478394	0.054411	0.028209	0.058966	1.083723	0.002179	1.00484
19	(coffee)	(toast)	0.478394	0.033597	0.023666	0.049470	1.472431	0.007593	1.01669
11	(coffee)	(juice)	0.478394	0.038563	0.020602	0.043065	1.116750	0.002154	1.00470

Qus-3: Implementation of FP-growth algorithm for finding Frequent Itemsets and Association Rules

For this problem, I will use the same bakery datasets that I have used in the previous question and will try to find the frequent itemsets and Assoiations rules between the items that are mostly buys together by the customers USING FP-Growth algorithm. And, also the data preprocessing is already being done above

- After doing Data Preproessing and EDA on the bakery dataset, we now have data bakery_data

In [103]:
```
bakery_data.head()
```

Out[103]:

	Transaction	Item	period_day	weekday_weekend	date	time	month	hour	weekday
0	1	bread	morning	weekend	2016-10-30 09:58:00	09:58:00	October	9-10	Sunday
1	2	scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
2	2	scandinavian	morning	weekend	2016-10-30 10:05:00	10:05:00	October	10-11	Sunday
3	3	hot chocolate	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday
4	3	jam	morning	weekend	2016-10-30 10:07:00	10:07:00	October	10-11	Sunday

In [104]:
```
transactions = bakery_data.groupby(['Transaction', 'Item'])['Item'].count().reset_index(name ='Count')
transactions
```

Out[104]:

	Transaction	Item	Count
0	1	bread	1
1	2	scandinavian	2
2	3	cookies	1
3	3	hot chocolate	1
4	3	jam	1
...	...	...	...
18882	9682	tacos/fajita	1
18883	9682	tea	1
18884	9683	coffee	1
18885	9683	pastry	1
18886	9684	smoothies	1

18887 rows × 3 columns

- Making a mxn matrice where m=transaction and n=items and each row represents whether the item was in the transaction or not

In [105]:
```
mybasket = transactions.pivot_table(index='Transaction', columns='Item', values
='Count', aggfunc='sum').fillna(0)

mybasket.head()
```

Out[105]:

Item	adjustment	afternoon with the baker	alfajores	argentina night	art tray	bacon	baguette	bakewell	bare popcorn	basket
Transaction										
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 94 columns

- making a function which returns 0 or 1
- 0 means item was not in that transaction, 1 means item present in that transaction

In [106]:
```
def encode(x):
    if x<=0:
        return 0
    if x>=1:
        return 1

# applying the function to the dataset

my_basket_sets = my_basket.applymap(encode)
my_basket_sets.head()
```

Out[106]:

Item	adjustment	afternoon with the baker	alfajores	argentina night	art tray	bacon	baguette	bakewell	bare popcorn	basket
Transaction										
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0

5 rows × 94 columns

- **Generating frequent_item sets with min_support=0.03**

In [107]:
```
pip install pyfpgrowth
```

```
Requirement already satisfied: pyfpgrowth in c:\users\mycomputer\anaconda3\lib\site-packages (1.0)Note: you may need to restart the kernel to use updated packages.
```

In [109]:
```
freq_item = fpgrowth(my_basket_sets,min_support=0.03,use_colnames = True)
freq_item
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-109-8da370f382b1> in <module>
----> 1 freq_item = fpgrowth(my_basket_sets,min_support=0.03,use_colnames = True
)
      2 freq_item

NameError: name 'fpgrowth' is not defined
```

In [110]:
```
rules_1 = association_rules(freq_item, metric = "lift", min_threshold = 1)
rules_1.sort_values('confidence', ascending = False, inplace = True)
rules_1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-110-a37f0cf7ff57> in <module>
----> 1 rules_1 = association_rules(freq_item, metric = "lift", min_threshold =
1)
      2 rules_1.sort_values('confidence', ascending = False, inplace = True)
      3 rules_1

NameError: name 'freq_item' is not defined
```

- **Arranging the data from highest to lowest with respect to 'confidence'**

In [111]:
```
rules_1.sort_values('confidence', ascending=False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-111-30b1cc965b77> in <module>
----> 1 rules_1.sort_values('confidence', ascending=False)

NameError: name 'rules_1' is not defined
```

Qus-4: Construct the decision tree using CART algorithm and evaluate the performance.

- Select a suitable data set and do the required preprocessing.
- Construct the tree using CART algorithm.
- Perform cross validation and tune the parameters to improve model's overall performance.
- Use different attribute selection measures and compare the model performance.
- Visualize the constructed decision tree.
- Extract rules form the constructed tree

In [203]:
```
car_df=pd.read_csv("C:/Users/Mycomputer/Documents/MCA_BHU/data mining/DM_Assignment/datasets/car_evaluation_dataset.csv")
```

In [204]:
```
car_df.head()
```

Out[204]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [205]:
```
car_df.columns
```

Out[205]:
```
Index(['vhigh', 'vhigh.1', '2', '2.1', 'small', 'low', 'unacc'], dtype='object')
```

In [206]:
```
car_df.describe()
```

Out[206]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
count	1727	1727	1727	1727	1727	1727	1727
unique	4	4	4	3	3	3	4
top	med	med	4	more	big	med	unacc
freq	432	432	432	576	576	576	1209

In [207]:
```
car_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   vhigh    1727 non-null   object
 1   vhigh.1  1727 non-null   object
 2   2        1727 non-null   object
 3   2.1      1727 non-null   object
 4   small    1727 non-null   object
 5   low      1727 non-null   object
 6   unacc    1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Renaming the columns

In [208]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'clas
s']


car_df.columns = col_names

car_df.columns
```

Out[208]:
```
Index(['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class'], dt
ype='object')
```

In [209]:
```
car_df.head()
```

Out[209]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [210]:
```
car_df.nunique()
```

Out[210]:
```
buying      4
maint       4
doors       4
persons     3
lug_boot    3
safety      3
class       4
dtype: int64
```

In [211]:
```
car_df.isna().sum()
```

Out[211]:
```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```

- We can see that there are no missing values in the dataset.

Frequency distribution of values in variable

- frequency counts of categorical variables.

In [212]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'clas
s']


for col in col_names:

    print(car_df[col].value_counts())
```

```
med      432
low      432
high     432
vhigh    431
Name: buying, dtype: int64
med      432
low      432
high     432
vhigh    431
Name: maint, dtype: int64
4        432
5more    432
3        432
2        431
Name: doors, dtype: int64
more    576
4       576
2       575
Name: persons, dtype: int64
big      576
med      576
small    575
Name: lug_boot, dtype: int64
med     576
high    576
low     575
Name: safety, dtype: int64
unacc    1209
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

- We can see that the doors and persons are categorical in nature. So, I will treat them as categorical variables.

Summary of variables

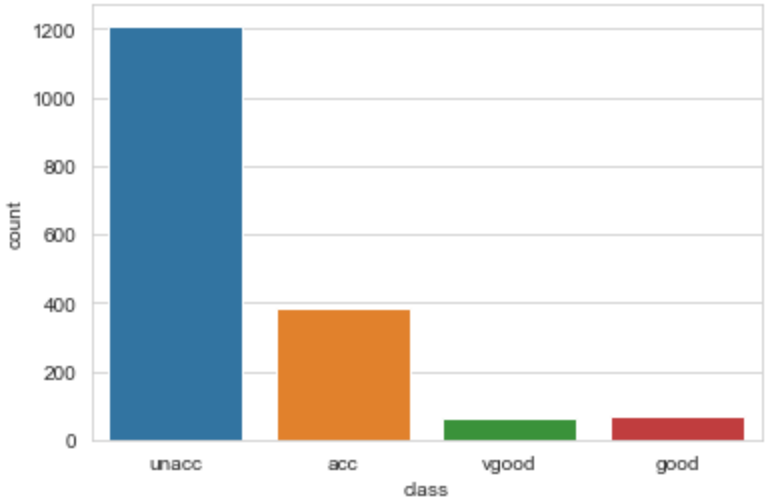
- There are 7 variables in the dataset. All the variables are of categorical data type.
- These are given by buying, maint, doors, persons, lug_boot, safety and class.
- class is the target variable.

In [213]:
```
car_df['class'].value_counts()
```

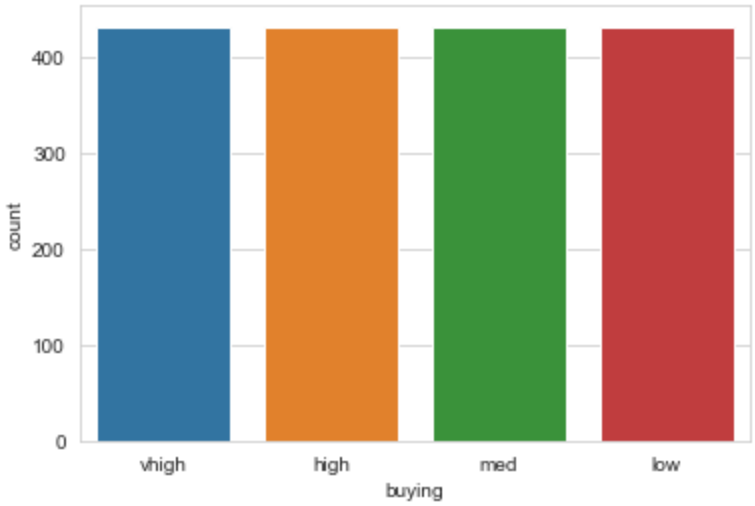
Out[213]:
```
unacc    1209
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

Data Visualization

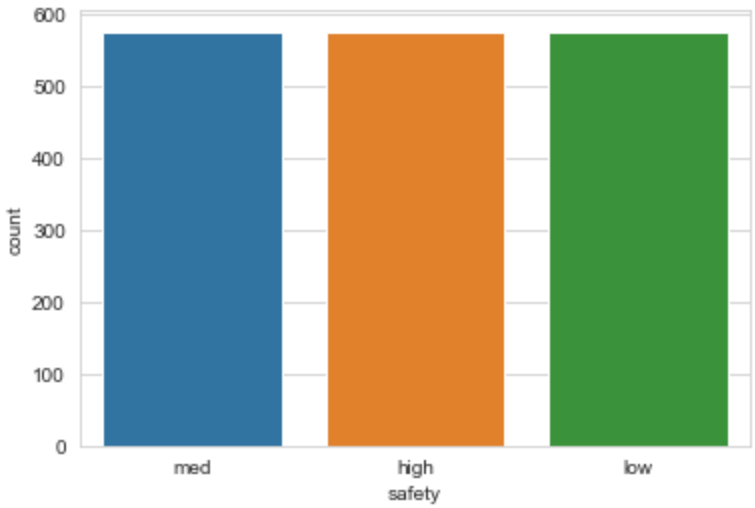
In [214]:
```
sns.countplot(x='class', data=car_df, )
plt.show()
```



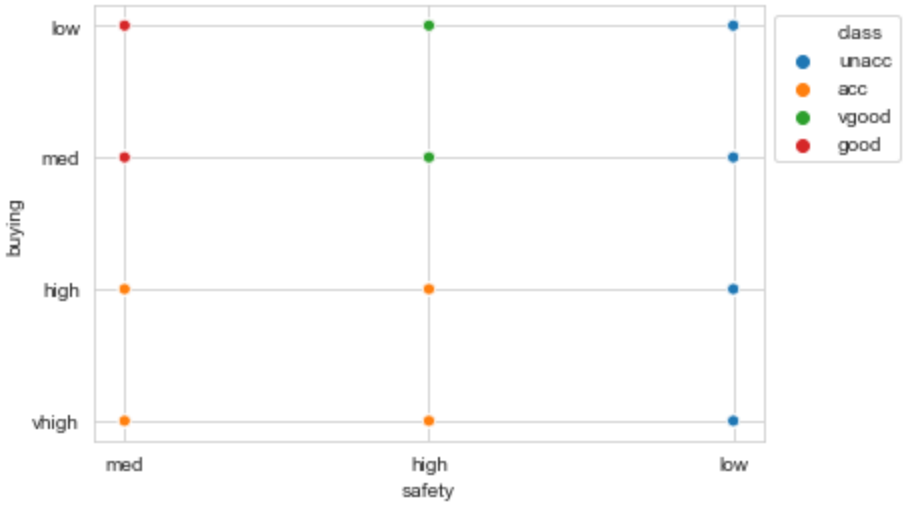
In [215]:
```
sns.countplot(x='buying', data=car_df, )
plt.show()
```



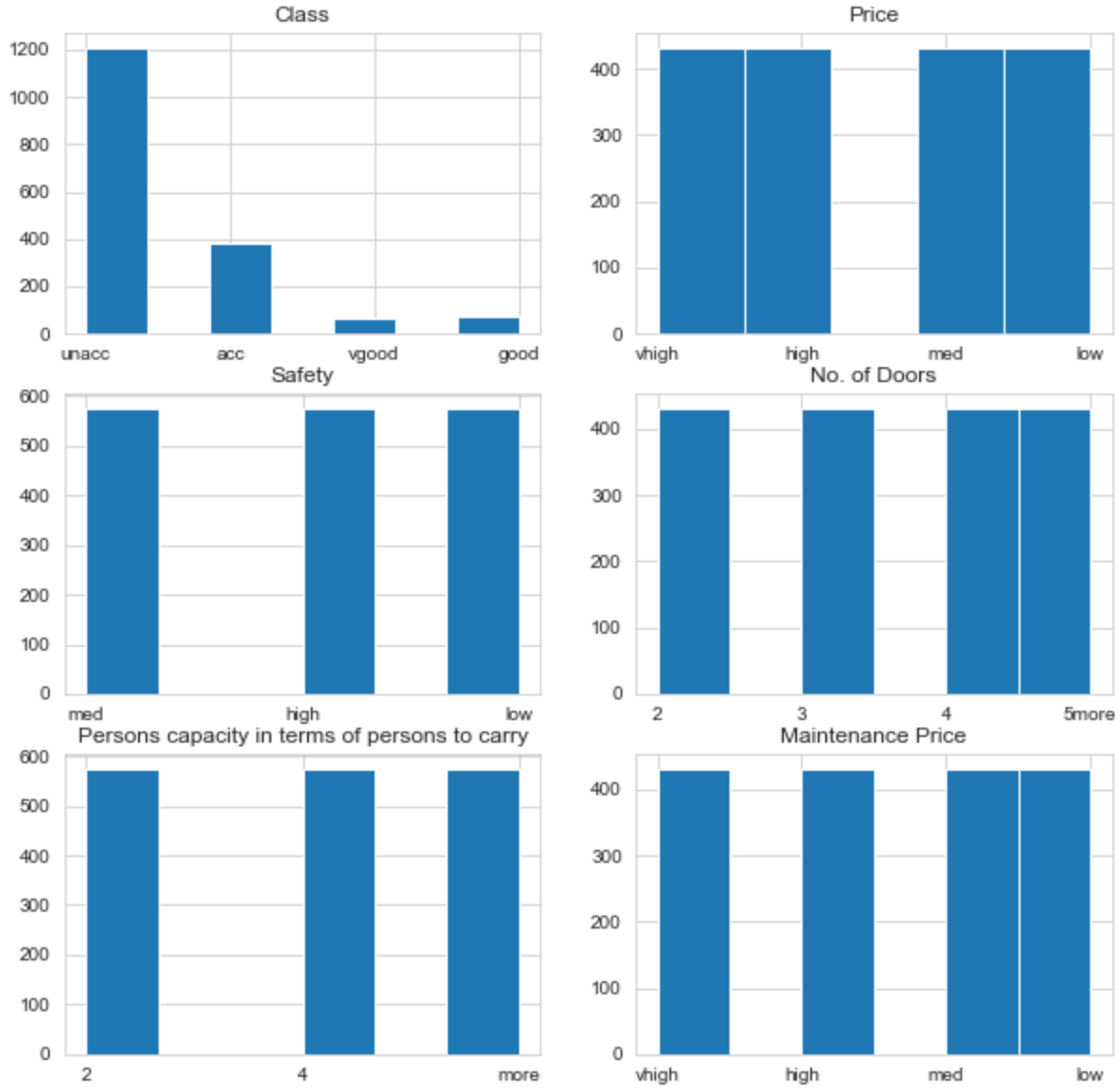
In [216]:
```
sns.countplot(x='safety', data=car_df, )
plt.show()
```



In [217]:
```
sns.scatterplot(x='safety', y='buying',
                hue='class', data=car_df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```



In [218]:
```
fig, axes = plt.subplots(3, 2, figsize=(10,10))

axes[0,0].set_title("Class")
axes[0,0].hist(car_df['class'], bins=7)

axes[0,1].set_title("Price")
axes[0,1].hist(car_df['buying'], bins=5);

axes[1,0].set_title("Safety")
axes[1,0].hist(car_df['safety'], bins=6);

axes[1,1].set_title("No. of Doors")
axes[1,1].hist(car_df['doors'], bins=6);

axes[2,0].set_title("Persons capacity in terms of persons to carry")
axes[2,0].hist(car_df['persons'], bins=6);

axes[2,1].set_title("Maintenance Price")
axes[2,1].hist(car_df['maint'], bins=6);
```



- Target Variable

In [219]:
```
X = car_df.drop(['class'], axis=1)

y = car_df['class']
```

In [220]:
```
y.head()
```

Out[220]:
```
0    unacc
1    unacc
2    unacc
3    unacc
4    unacc
Name: class, dtype: object
```

- split X and y into training and testing sets

In [221]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, ran
dom_state = 42)
```

In [222]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[222]:
```
((1157, 6), (570, 6))
```

In [223]:
```
X_train.dtypes
```

Out[223]:
```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
dtype: object
```

Encode categorical variables

In [224]:
```
X_train.head()
```

Out[224]:

	buying	maint	doors	persons	lug_boot	safety
83	vhigh	vhigh	5more	2	med	low
48	vhigh	vhigh	3	more	med	med
468	high	vhigh	3	4	small	med
155	vhigh	high	3	more	med	low
1043	med	high	4	more	small	low

In [225]:
```
# import category encoders

import category_encoders as ce
```

In [226]:
```
# encode variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [227]:
```
X_train.head()
```

Out[227]:

	buying	maint	doors	persons	lug_boot	safety
83	1	1	1	1	1	1
48	1	1	2	2	1	2
468	2	1	2	3	2	2
155	1	2	2	2	1	1
1043	3	2	3	2	2	1

In [228]:
```
X_test.head()
```

Out[228]:

	buying	maint	doors	persons	lug_boot	safety
599	2	2	3	1	3	1
932	3	1	3	3	3	1
628	2	2	1	1	3	3
1497	4	2	1	3	1	2
1262	3	4	3	2	1	1

Decision Tree Classifier with criterion gini index(CART Algorithm)

In [229]:
```
# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
```

In [230]:
```
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)


# fit the model
clf_gini.fit(X_train, y_train)
```

Out[230]:

```
▼              DecisionTreeClassifier

DecisionTreeClassifier(max_depth=3, random_state=0)
```

Prediction on the Test data

In [231]:
```
y_pred_gini = clf_gini.predict(X_test)
```

- Check accuracy score with criterion gini index

In [232]:
```
print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accura
cy_score(y_test, y_pred_gini)))
```

```
Model accuracy score with criterion gini index: 0.8053
```

Here, y_test are the true class labels and y_pred_gini are the predicted class labels in the test-set.

Compare the train-set and test-set accuracy

In [233]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[233]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [234]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y
_pred_train_gini)))
```

```
Training-set accuracy score: 0.7848
```

In [235]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

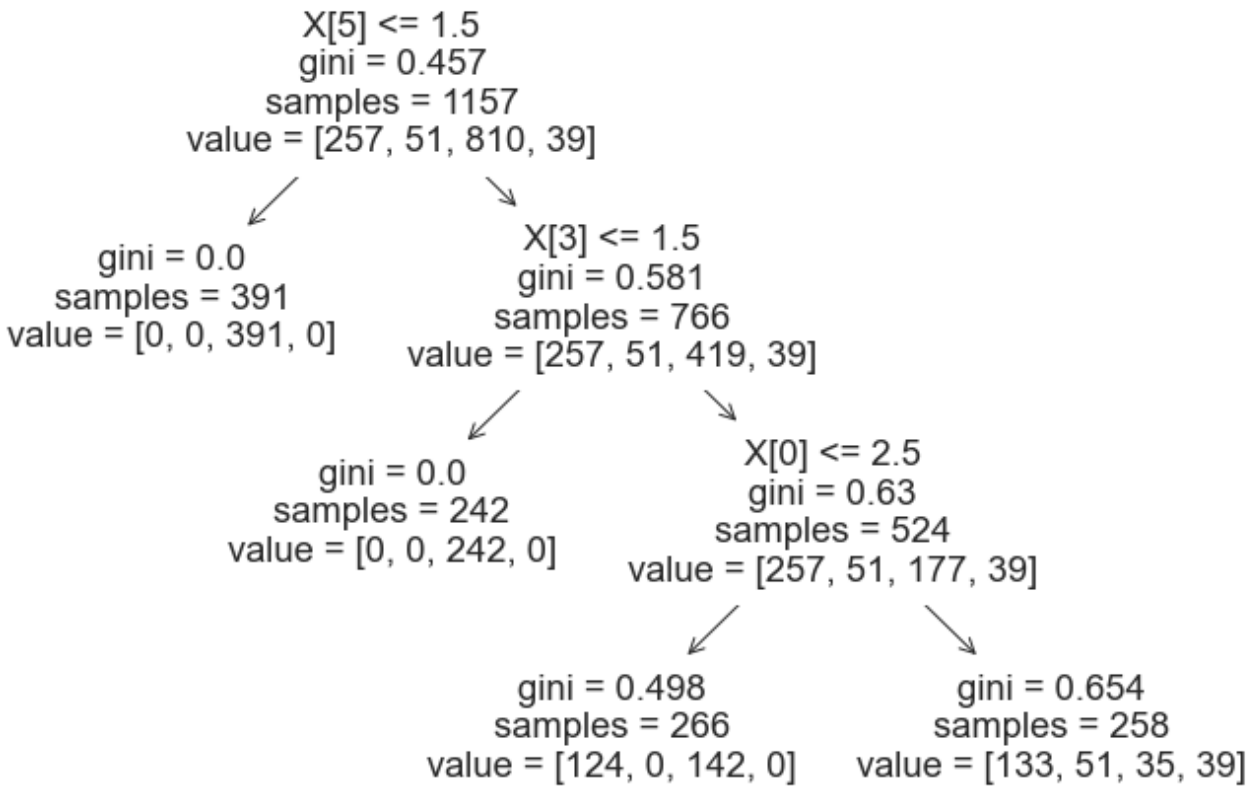
```
Training set score: 0.7848
Test set score: 0.8053
```

Here, the training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.

Visualizing the constructed decision tree

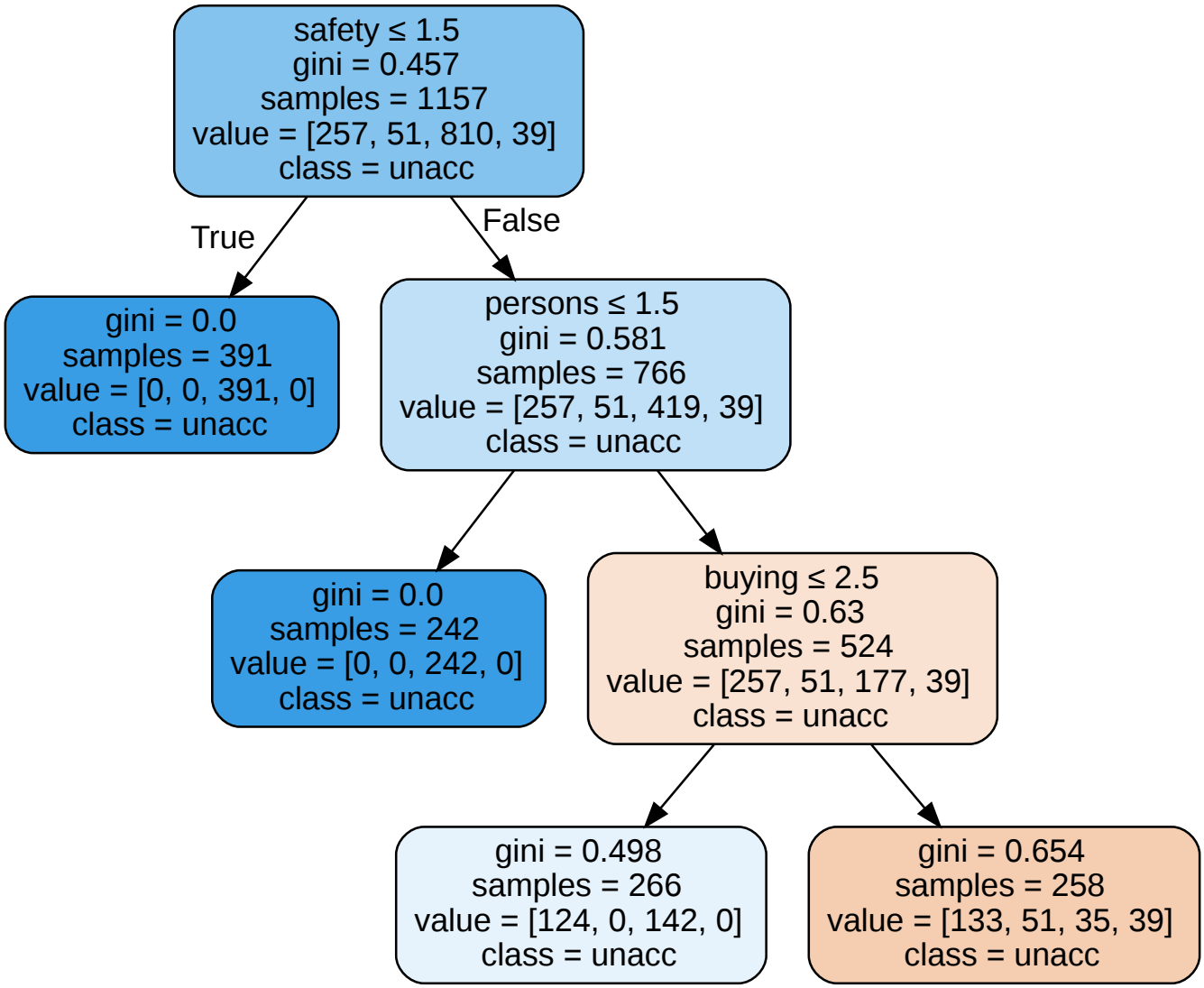
In [236]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[236]:
```
[Text(0.3333333333333333, 0.875, 'X[5] <= 1.5\ngini = 0.457\nsamples = 1157\nval
ue = [257, 51, 810, 39]'),
 Text(0.1666666666666666, 0.625, 'gini = 0.0\nsamples = 391\nvalue = [0, 0, 39
1, 0]'),
 Text(0.5, 0.625, 'X[3] <= 1.5\ngini = 0.581\nsamples = 766\nvalue = [257, 51, 4
19, 39]'),
 Text(0.3333333333333333, 0.375, 'gini = 0.0\nsamples = 242\nvalue = [0, 0, 242,
0]'),
 Text(0.6666666666666666, 0.375, 'X[0] <= 2.5\ngini = 0.63\nsamples = 524\nvalue
= [257, 51, 177, 39]'),
 Text(0.5, 0.125, 'gini = 0.498\nsamples = 266\nvalue = [124, 0, 142, 0]'),
 Text(0.8333333333333334, 0.125, 'gini = 0.654\nsamples = 258\nvalue = [133, 51,
35, 39]')]
```



In [237]:
```
import graphviz
dot_data = tree.export_graphviz(clf_gini, out_file=None,
                                feature_names=X_train.columns,
                                class_names=y_train,
                                filled=True, rounded=True,
                                special_characters=True)

graph = graphviz.Source(dot_data)

graph
```

Out[237]:



Decision Tree Classifier with criterion entropy

In [238]:
```
# instantiate the DecisionTreeClassifier model with criterion entropy

clf_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=
0)


# fit the model
clf_en.fit(X_train, y_train)
```

Out[238]:
```
▼                          DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

In [239]:
```
#Prediction
y_pred_en = clf_en.predict(X_test)
```

In [240]:
```
# Checking Accuracy
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_
score(y_test, y_pred_en)))
```

```
Model accuracy score with criterion entropy: 0.8053
```

Compare the train-set and test-set accuracy

In [241]:
```
y_pred_train_en = clf_en.predict(X_train)

y_pred_train_en
```

Out[241]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [242]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y
_pred_train_en)))
```

```
Training-set accuracy score: 0.7848
```

Check for overfitting and underfitting

In [243]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

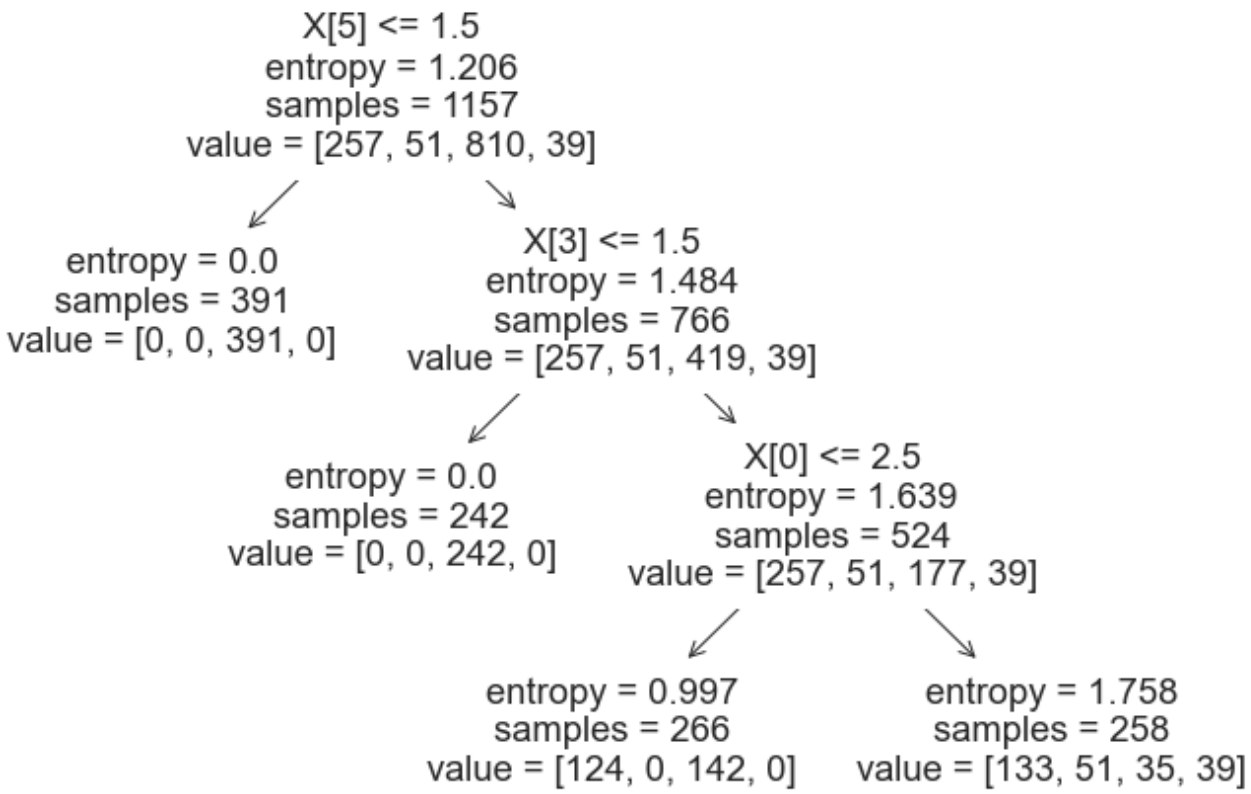
```
Training set score: 0.7848
Test set score: 0.8053
```

We can see that the training-set score and test-set score is same as above. The training-set accuracy score is 0.7865 while the test-set accuracy to be 0.8021. These two values are quite comparable. So, there is no sign of overfitting.

- **Visualizing the constructed decision tree**

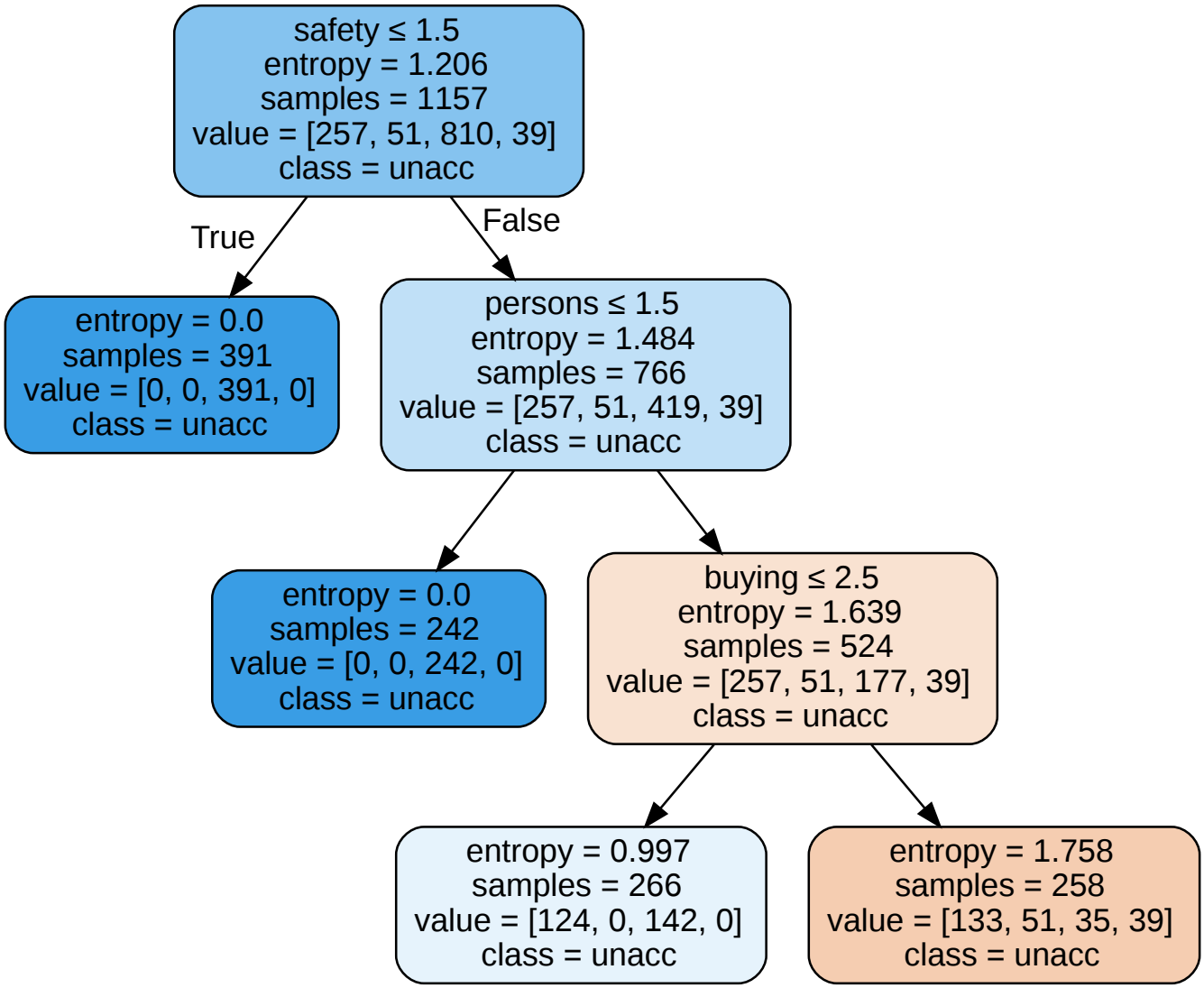
In [244]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[244]:
```
[Text(0.3333333333333333, 0.875, 'X[5] <= 1.5\nentropy = 1.206\nsamples = 1157\n
value = [257, 51, 810, 39]'),
 Text(0.16666666666666666, 0.625, 'entropy = 0.0\nsamples = 391\nvalue = [0, 0,
391, 0]'),
 Text(0.5, 0.625, 'X[3] <= 1.5\nentropy = 1.484\nsamples = 766\nvalue = [257, 5
1, 419, 39]'),
 Text(0.3333333333333333, 0.375, 'entropy = 0.0\nsamples = 242\nvalue = [0, 0, 2
42, 0]'),
 Text(0.6666666666666666, 0.375, 'X[0] <= 2.5\nentropy = 1.639\nsamples = 524\nv
alue = [257, 51, 177, 39]'),
 Text(0.5, 0.125, 'entropy = 0.997\nsamples = 266\nvalue = [124, 0, 142, 0]'),
 Text(0.8333333333333334, 0.125, 'entropy = 1.758\nsamples = 258\nvalue = [133,
51, 35, 39]')]
```



In [245]:
```
import graphviz
dot_data = tree.export_graphviz(clf_en, out_file=None,
                                feature_names=X_train.columns,
                                class_names=y_train,
                                filled=True, rounded=True,
                                special_characters=True)

graph = graphviz.Source(dot_data)

graph
```

Out[245]:



Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called Type I error.

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called Type II error.

These four outcomes are summarized in a confusion matrix given below.

In [246]:
```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[ 71   0  56   0]
 [ 18   0   0   0]
 [ 11   0 388   0]
 [ 26   0   0   0]]
```

Classification Report

In [247]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_en))
```

```
              precision    recall  f1-score   support

         acc       0.56      0.56      0.56       127
        good       0.00      0.00      0.00        18
       unacc       0.87      0.97      0.92       399
       vgood       0.00      0.00      0.00        26

    accuracy                           0.81       570
   macro avg       0.36      0.38      0.37       570
weighted avg       0.74      0.81      0.77       570
```

Qus-9: Implementation of linear regression.

1. Selecting a suitable data set and performing the required preprocessing.

In [113]:
```
# Import Dataset from sklearn
from sklearn.datasets import load_iris
# Load Iris Data
iris = load_iris()
```

In [114]:
```
# Creating pd DataFrames
iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
target_df = pd.DataFrame(data= iris.target, columns= ['species'])
def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'
target_df['species'] = target_df['species'].apply(converter)
# Concatenate the DataFrames
iris_df = pd.concat([iris_df, target_df], axis= 1)
```

In [115]:
```
iris_df.head()
```

Out[115]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [116]:
```
target_df.head()
```

Out[116]:

	species
0	setosa
1	setosa
2	setosa
3	setosa
4	setosa

In [117]:
```
iris_df.shape
```

Out[117]:
```
(150, 5)
```

In [118]:
```
iris_df.columns
```

Out[118]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

- Renaming columns

In [119]:
```
iris_df.columns = ['sepalLength','sepalWidth','petalLength','petalWidth','species']
```

iris dataset consists of 150 rows and 6 columns.

In [120]:
```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sepalLength  150 non-null    float64
 1   sepalWidth   150 non-null    float64
 2   petalLength  150 non-null    float64
 3   petalWidth   150 non-null    float64
 4   species      150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [121]:
```
iris_df.describe()
```

Out[121]:

	sepalLength	sepalWidth	petalLength	petalWidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [122]:
```
iris_df.dtypes
```

Out[122]:
```
sepalLength    float64
sepalWidth     float64
petalLength    float64
petalWidth     float64
species         object
dtype: object
```

iris data have-

- Numerical columns: SepalLengthCm, SepalWidthCm, PetalLengthCm,PetalWidthCm
- Categorial columns: Species

In [123]:
```
# Number of uniquevalues in each columns
iris_df.nunique()
```

Out[123]:
```
sepalLength    35
sepalWidth     23
petalLength    43
petalWidth     22
species         3
dtype: int64
```

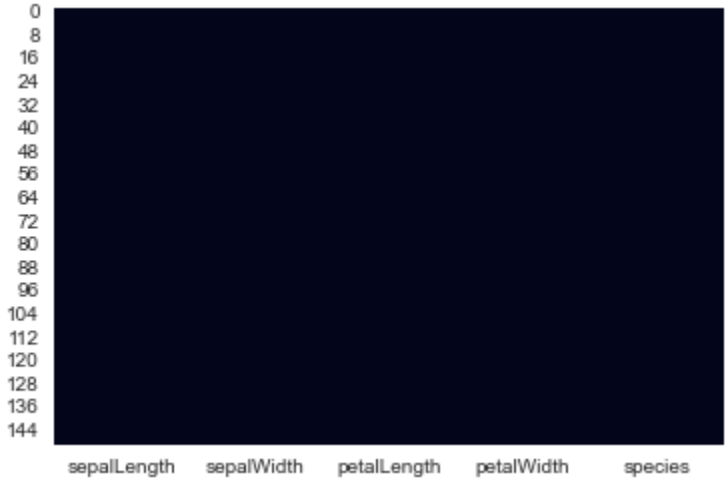
In [124]:
```
#Missing values
iris_df.isna().sum()
```

Out[124]:
```
sepalLength    0
sepalWidth     0
petalLength    0
petalWidth     0
species        0
dtype: int64
```

- There is no missing values in the data.

In [125]:
```
### Visualizing the missing values using heatmap
sns.heatmap(iris_df.isnull(),cbar=False)
```

Out[125]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e7f5ce20>
```



In [126]:
```
iris_df["species"].value_counts()
```

Out[126]:
```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

In [127]:
```
# data["column_name"].sum()

sum_data = iris_df["sepalLength"].sum()
mean_data = iris_df["sepalLength"].mean()
median_data = iris_df["sepalLength"].median()

print("Sum:",sum_data, "\nMean:", mean_data, "\nMedian:",median_data)
```

```
Sum: 876.5
Mean: 5.843333333333335
Median: 5.8
```

In [128]:
```
min_data=iris_df["sepalLength"].min()
max_data=iris_df["sepalLength"].max()

print("Minimum:",min_data, "\nMaximum:", max_data)
```

```
Minimum: 4.3
Maximum: 7.9
```

- **Five Number Summary of data**

In [129]:
```
print("Five number summary of the data")
des=iris_df.describe()
des
des.iloc[3:8,]
```

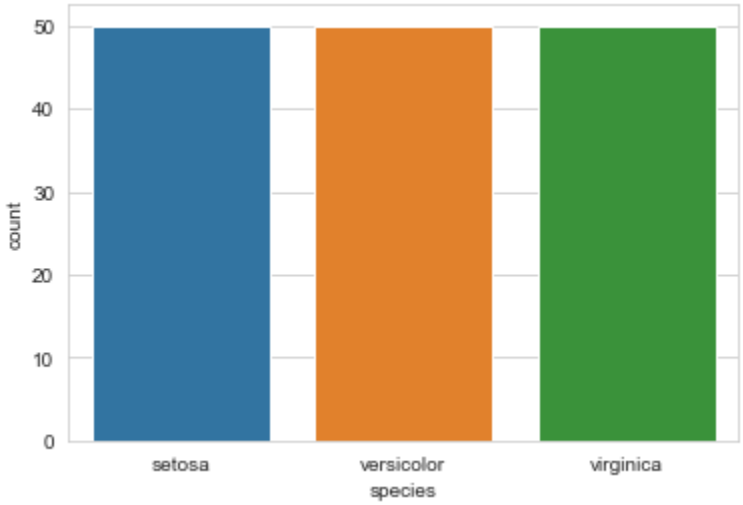
```
Five number summary of the data
```

Out[129]:

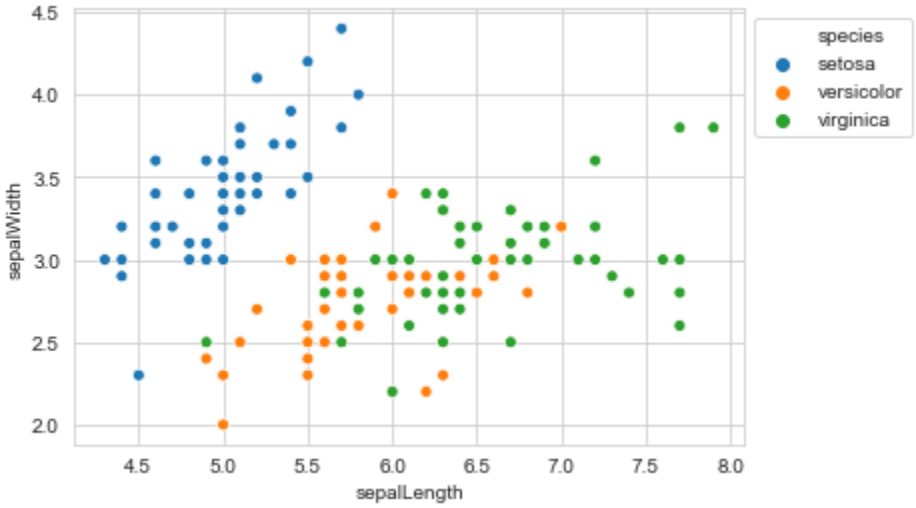
	sepalLength	sepalWidth	petalLength	petalWidth
min	4.3	2.0	1.00	0.1
25%	5.1	2.8	1.60	0.3
50%	5.8	3.0	4.35	1.3
75%	6.4	3.3	5.10	1.8
max	7.9	4.4	6.90	2.5

Data Visualization

In [130]:
```
sns.countplot(x='species', data=iris_df, )
plt.show()
```



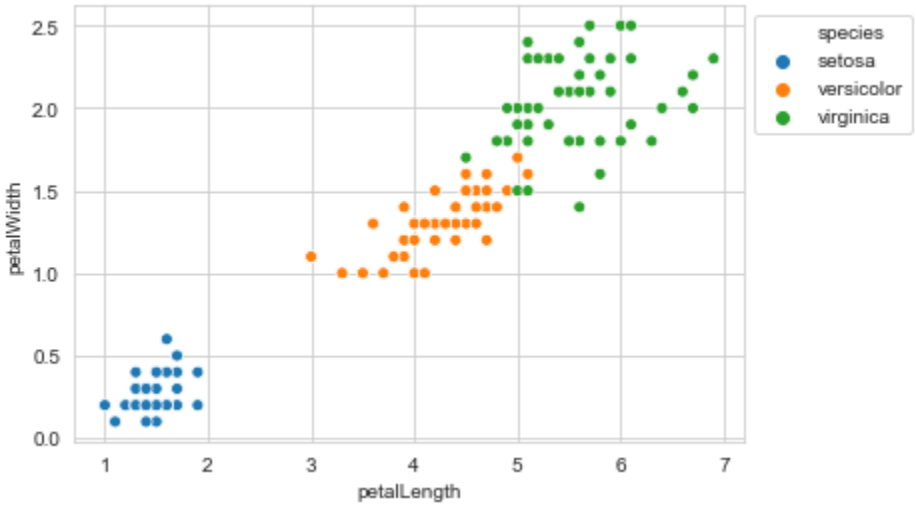
In [131]:
```
sns.scatterplot(x='sepalLength', y='sepalWidth',
                hue='species', data=iris_df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```



From the above plot, we can infer that –

- Species Setosa has smaller sepal lengths but larger sepal widths.
- Versicolor Species lies in the middle of the other two species in terms of sepal length and width
- Species Virginica has larger sepal lengths but smaller sepal widths.

In [132]:
```
sns.scatterplot(x='petalLength', y='petalWidth',
                hue='species', data=iris_df, )

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

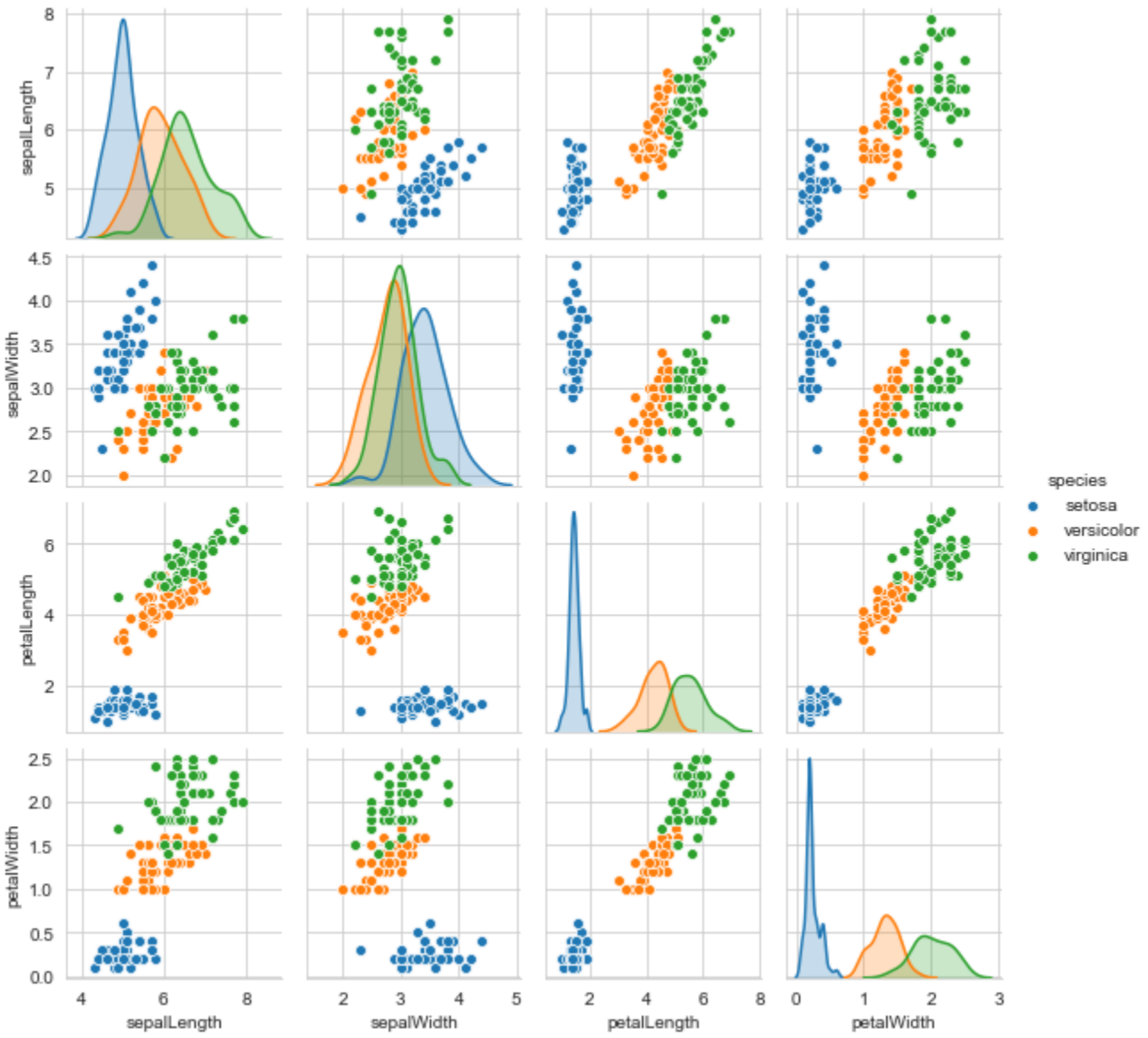


From the above plot, we can infer that –

- Species Setosa has smaller petal lengths and widths.
- Versicolor Species lies in the middle of the other two species in terms of petal length and width
- Species Virginica has the largest of petal lengths and widths.

```
In [133]: sns.pairplot(iris_df, hue='species', height=2)
```

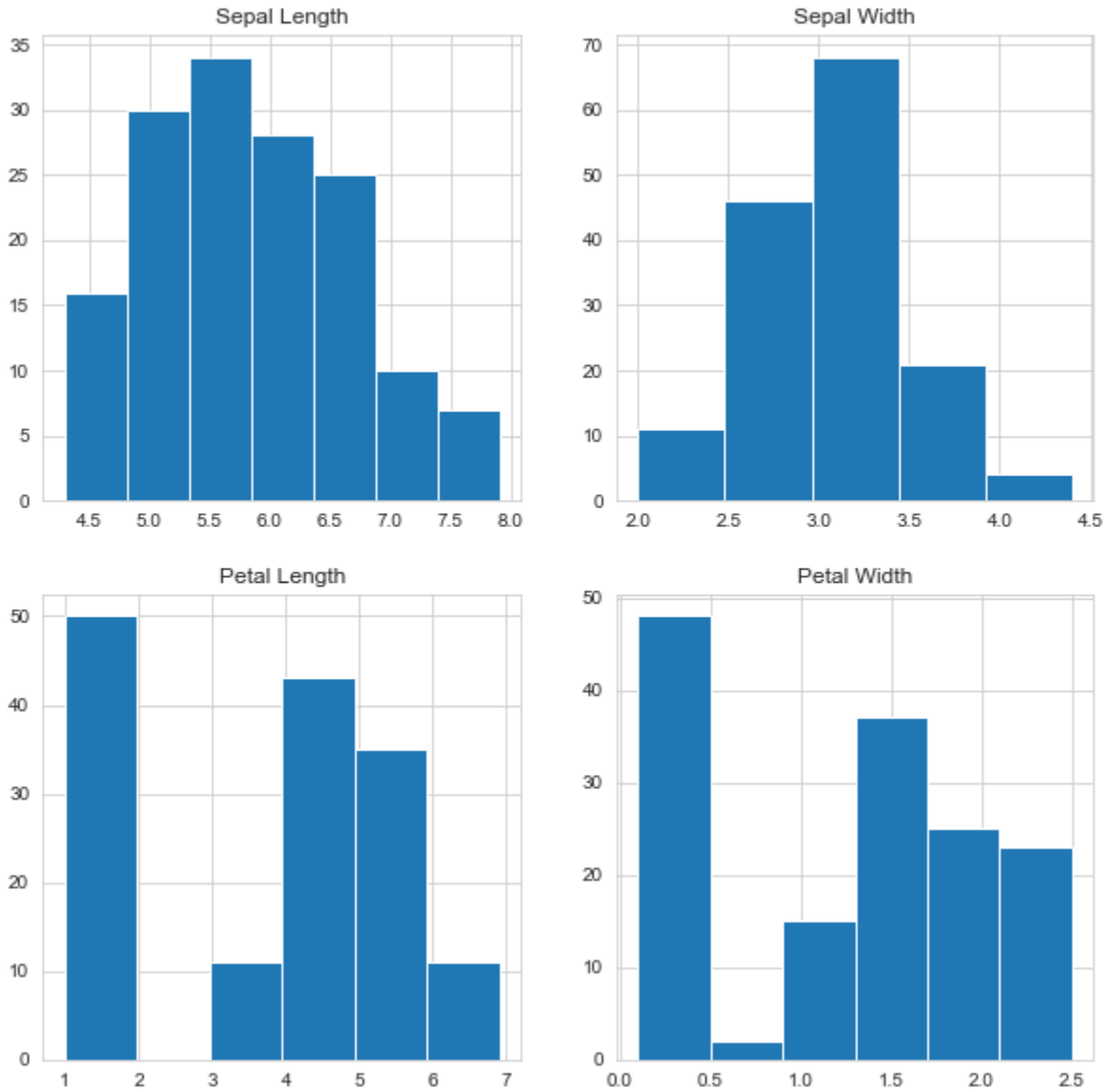
```
Out[133]: <seaborn.axisgrid.PairGrid at 0x201e7f7ce20>
```



We can see many types of relationships from this plot such as the species Seotsa has the smallest of petals widths and lengths. It also has the smallest sepal length but larger sepal widths. Such information can be gathered about any other species.

Histograms

In [134]:
```
fig, axes = plt.subplots(2, 2, figsize=(10,10))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(iris_df['sepalLength'], bins=7)

axes[0,1].set_title("Sepal Width")
axes[0,1].hist(iris_df['sepalWidth'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(iris_df['petalLength'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(iris_df['petalWidth'], bins=6);
```



From the above plot, we can see that –

- The highest frequency of the sepal length is between 30 and 35 which is between 5.5 and 6
- The highest frequency of the sepal Width is around 70 which is between 3.0 and 3.5
- The highest frequency of the petal length is around 50 which is between 1 and 2
- The highest frequency of the petal width is between 40 and 50 which is between 0.0 and 0.5

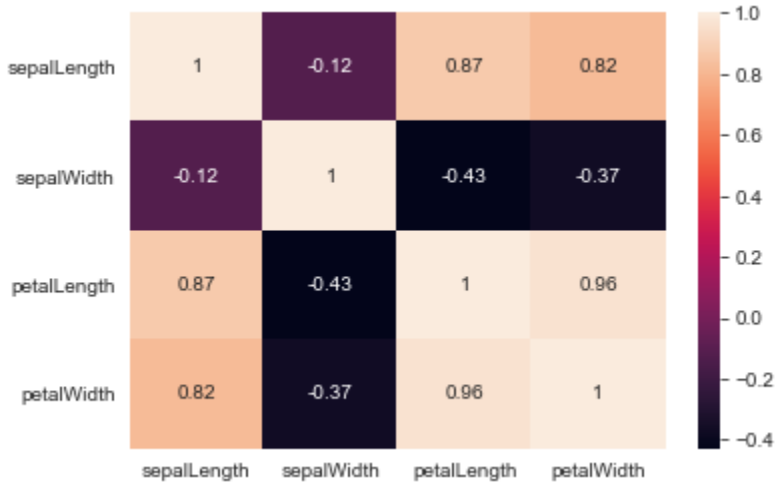
Handling Correlation

In [135]:
```
iris_df.corr(method='pearson')
```

Out[135]:

	sepalLength	sepalWidth	petalLength	petalWidth
sepalLength	1.000000	-0.117570	0.871754	0.817941
sepalWidth	-0.117570	1.000000	-0.428440	-0.366126
petalLength	0.871754	-0.428440	1.000000	0.962865
petalWidth	0.817941	-0.366126	0.962865	1.000000

In [136]:
```
sns.heatmap(iris_df.corr(method='pearson'), annot = True);

plt.show()
```



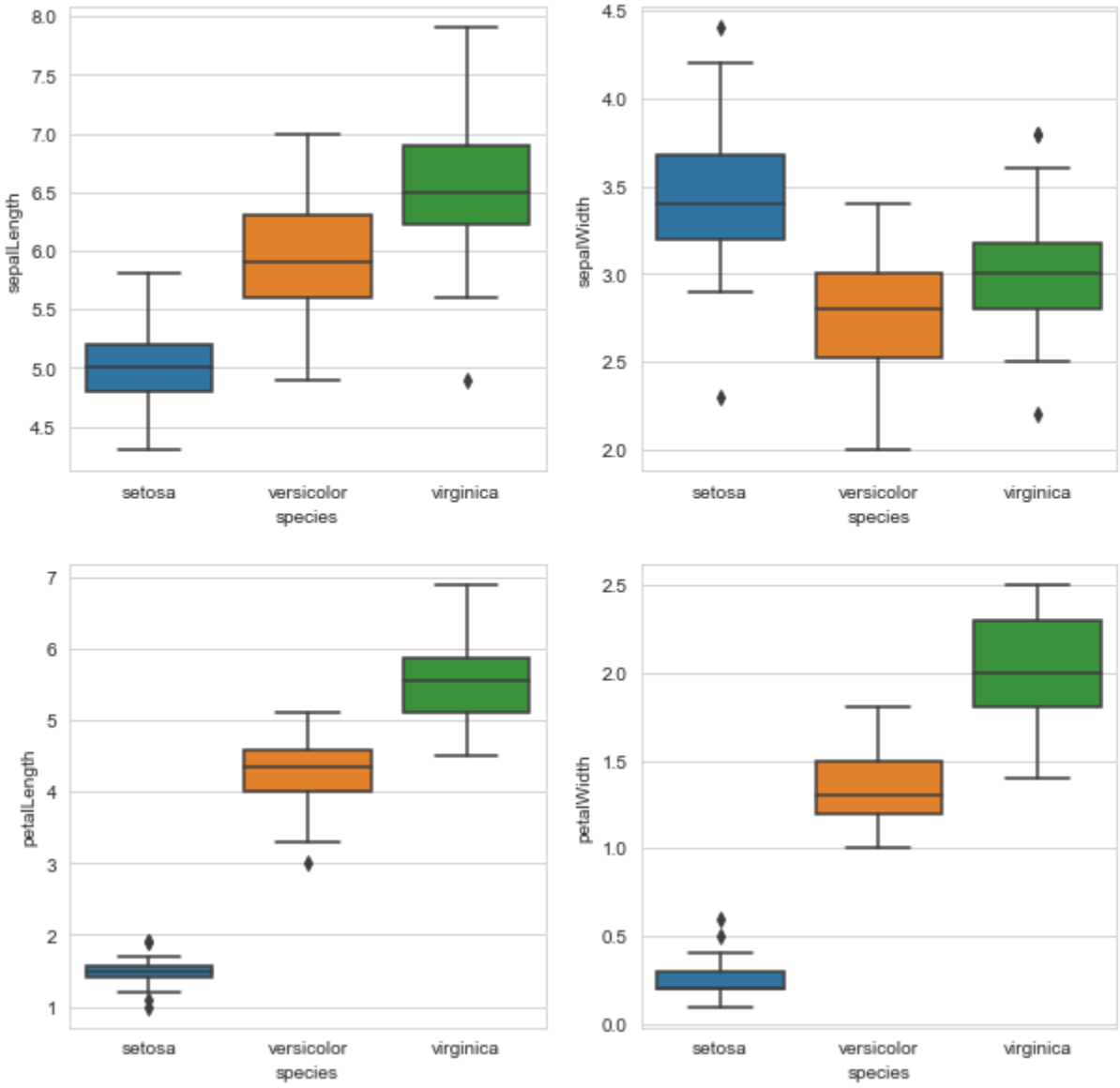
From the above graph, we can see that –

- Petal width and petal length have high correlations.
- Petal length and sepal width have good correlations.
- Petal Width and Sepal length have good correlations.

Box Plots

In [137]:

```
def graph(y):
    sns.boxplot(x="species", y=y, data=iris_df)

plt.figure(figsize=(10,10))

# Adding the subplot at the specified
# grid position
plt.subplot(221)
graph('sepalLength')

plt.subplot(222)
graph('sepalWidth')

plt.subplot(223)
graph('petalLength')

plt.subplot(224)
graph('petalWidth')

plt.show()
```



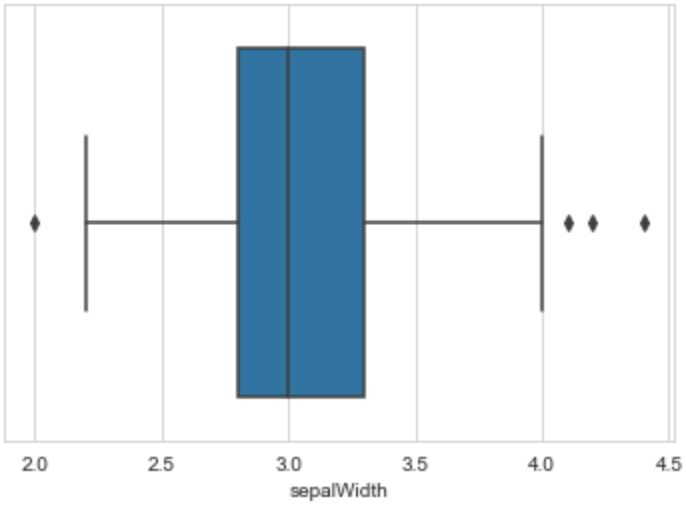
From the above graph, we can see that –

- Species Setosa has the smallest features and less distributed with some outliers.
- Species Versicolor has the average features.
- Species Virginica has the highest features

Handling Outliers

```
In [138]: sns.boxplot(x='sepalWidth', data=iris_df)
```

```
Out[138]: <matplotlib.axes._subplots.AxesSubplot at 0x201e9509880>
```

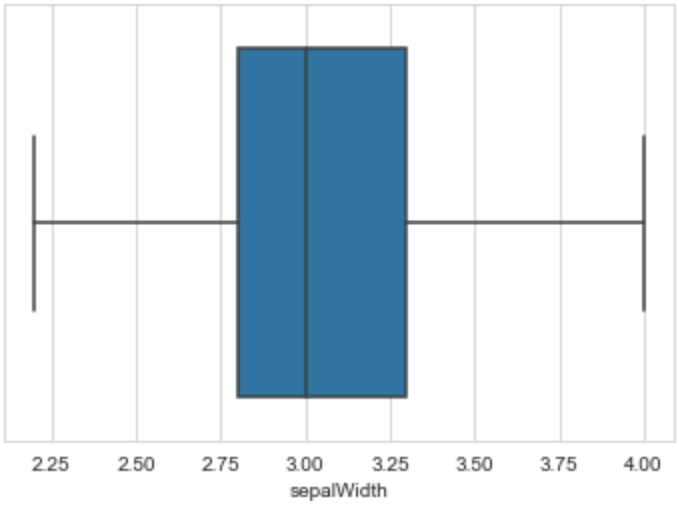


- In the above graph, the values above 4 and below 2 are acting as outliers.

In [139]:
```
# IQR
Q1 = np.percentile(iris_df['sepalWidth'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(iris_df['sepalWidth'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", iris_df.shape)

# Upper bound
upper = np.where(iris_df['sepalWidth'] >= (Q3+1.5*IQR))

# Lower bound
lower = np.where(iris_df['sepalWidth'] <= (Q1-1.5*IQR))

# Removing the Outliers
iris_df.drop(upper[0], inplace = True)
iris_df.drop(lower[0], inplace = True)

print("New Shape: ",iris_df.shape)

sns.boxplot(x='sepalWidth', data=iris_df)
```

```
Old Shape:  (150, 5)
New Shape:  (146, 5)
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x201e9e07be0>
```



Linear Regression

- **Prediction of sepal length (cm) of the iris flowers using Linear Regression**

- Converting Objects to Numerical dtype

In [140]:
```
iris_df.head()
```

Out[140]:

	sepalLength	sepalWidth	petalLength	petalWidth	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [141]:
```
iris_df.drop('species', axis= 1, inplace= True)
target_df = pd.DataFrame(columns= ['species'], data= iris.target)
iris_df = pd.concat([iris_df, target_df], axis= 1)
iris_df.head()
```

Out[141]:

	sepalLength	sepalWidth	petalLength	petalWidth	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [142]:
```
# Variables
X= iris_df.drop(labels= 'sepalLength', axis= 1)
y= iris_df['sepalLength']
y
```

Out[142]:
```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepalLength, Length: 150, dtype: float64
```

In [143]:
```
X.isna().sum()
```

Out[143]:
```
sepalWidth     4
petalLength    4
petalWidth     4
species        0
dtype: int64
```

In [151]:
```
X=X.dropna()
```

In [185]:
```
y.isna().sum()
```

Out[185]:
```
4
```

In [187]:
```
y=y.dropna()
```

In [188]:
```
# Splitting the Dataset
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size= 0.33,
random_state= 101)
```

In [189]:
```
# rescale the features
scaler = MinMaxScaler()

# apply scaler() to all the numeric columns
iris_df = scaler.fit_transform(iris_df)
X_train1.head()
```

Out[189]:

	sepalWidth	petalLength	petalWidth	species
106	2.5	4.5	1.7	2
78	2.9	4.5	1.5	1
40	3.5	1.3	0.3	0
42	3.2	1.3	0.2	0
90	2.6	4.4	1.2	1

In [190]:
```
X_train1.isna().sum()
```

Out[190]:
```
sepalWidth     0
petalLength    0
petalWidth     0
species        0
dtype: int64
```

In [191]:
```
X_train1=X_train1.dropna()
y_train1=y_train1.dropna()
```

In [192]:
```
# LinearRegression() Model
lr = LinearRegression()
```

In [193]:
```
# Training/Fitting the Model
lr.fit(X_train1, y_train1)
lr.fit
```

Out[193]:
```
<bound method LinearRegression.fit of LinearRegression()>
```

In [194]:
```
d = {'sepal length (cm)' : [4.6],
    'sepal width (cm)' : [3.4],
    'petal length (cm)' : [1.4],
    'petal width (cm)' : [0.3],
    'species' : 0}
test_df = pd.DataFrame(data= d)
test_df
```

Out[194]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	4.6	3.4	1.4	0.3	0

In [195]:
```
X_test1=X_test1.dropna()
```

In [196]:
```
# Making Predictions
lr.predict(X_test1)
pred = lr.predict(X_test1)
pred
```

Out[196]:
```
array([5.61267319, 4.85521812, 4.96881093, 5.42884626, 4.63896992,
       6.03102483, 5.71233845, 6.29901344, 5.00662378, 6.44748426,
       5.89616538, 6.17268997, 6.81582579, 6.67850961, 4.66127918,
       6.04217947, 5.41365887, 6.58250564, 5.80084908, 6.88899476,
       5.92769829, 6.86914222, 5.55113698, 5.58182795, 4.94215272,
       6.35982322, 4.8184479 , 8.01567206, 6.67383679, 7.87819395,
       6.28419752, 6.47865336, 5.43581387, 5.01919251, 5.80834242,
       4.76267473, 4.96027496, 6.41732668, 6.3986787 , 4.60377572,
       4.46629761, 6.22427611, 6.47917909, 7.21851192, 6.12634114,
       5.66860828, 5.37411572, 5.09932143, 6.37882617])
```

In [197]:
```
y_test1=y_test1.dropna()
```

In [198]:
```
# Evaluating Model's Performance
print('Mean Absolute Error:', mean_absolute_error(y_test1, pred))
print('Mean Squared Error:', mean_squared_error(y_test1, pred))
print('Mean Root Squared Error:', np.sqrt(mean_squared_error(y_test1, pred)))
```

```
Mean Absolute Error: 0.2597782623771217
Mean Squared Error: 0.10674838509376988
Mean Root Squared Error: 0.32672371369977093
```

Evaluate the model with the following metrics using cross validation:

- a) R Square/Adjusted R Square.
- b) Mean Square Error(MSE)/Root Mean Square Error(RMSE)
- c) Mean Absolute Error(MAE)

In [199]:
```
#sklearn.metrics.get_scorer_names()
```

In [200]:
```
# k-fold CV using R Square/Adjusted R Square.
lm = LinearRegression()
np.mean(cross_val_score(lm, X_train1, y_train1, scoring='r2', cv=5))
```

Out[200]:
```
0.7773059219671189
```

In [201]:
```
# k-fold CV using Mean Square Error(MSE)/Root Mean Square Error(RMSE)
np.mean(-1*cross_val_score(lm, X_train1, y_train1, scoring='neg_mean_squared_er
ror', cv=5))
```

Out[201]: 0.10970147355456739

In [202]:
```
# k-fold CV using Mean Absolute Error(MAE)
np.mean(-1*cross_val_score(lm, X_train1, y_train1, scoring='neg_mean_squared_er
ror', cv=5))
```

Out[202]: 0.10970147355456739

In [ ]:

In [ ]:

In [ ]: