Chapter 3 - Regressoin Models

Segment 1 - Simple linear regression

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn

from pylab import rcParams

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import scale
```

In [2]:
```
%matplotlib inline
rcParams['figure.figsize'] = 10,8
```

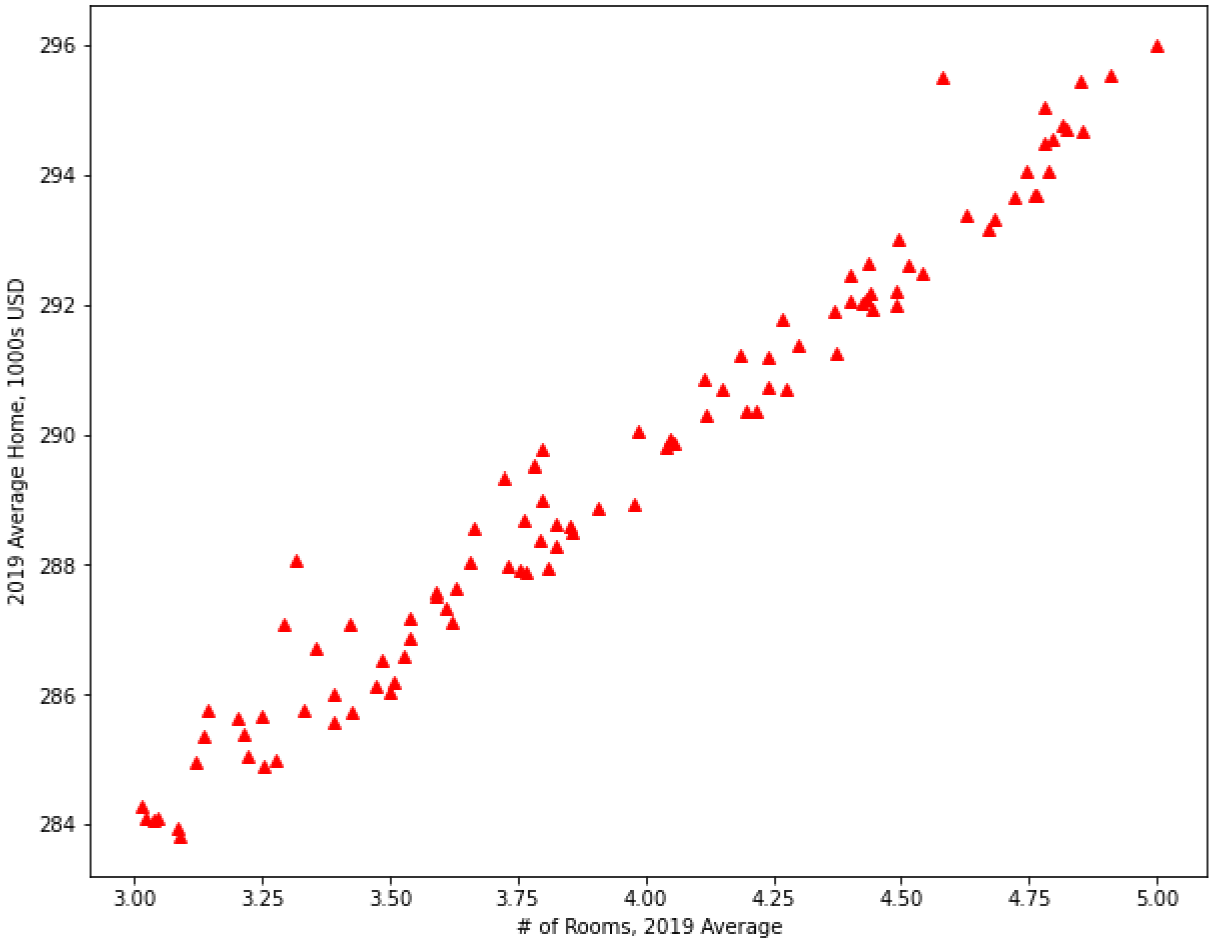
In [3]:
```
rooms = 2*np.random.rand(100, 1)+3
rooms[1:10]
```

Out[3]:
```
array([[4.78064306],
       [3.73240623],
       [4.36739615],
       [4.21742064],
       [3.98518742],
       [4.78965688],
       [4.9989641 ],
       [3.4234276 ],
       [4.42320375]])
```

In [4]:
```
price = 265 + 6*rooms +abs(np.random.randn(100, 1))
price[1:10]
```

Out[4]:
```
array([[294.48912037],
       [287.98682969],
       [291.9115306 ],
       [290.35172943],
       [290.05630896],
       [294.07397989],
       [296.01408121],
       [287.08271623],
       [292.01932983]])
```

In [5]:
```
plt.plot(rooms, price, 'r^')
plt.xlabel("# of Rooms, 2019 Average")
plt.ylabel("2019 Average Home, 1000s USD")
plt.show()
```



In [6]:

```
X = rooms
y = price

LinReg = LinearRegression()
LinReg.fit(X,y)
print(LinReg.intercept_, LinReg.coef_)
```

```
[266.66650943] [[5.77725468]]
```

Simple Algebra

- y = mx + b
- b = intercept = 265.7

Estimated Coefficients

- LinReg.coef_ = [5.99] Estimated coefficients for the terms in the linear regression problem.

In [7]:

```
print(LinReg.score(X,y))
```

```
0.9691297809079072
```

Segment 2 - Multiple linear regression

In [8]:

```
import seaborn as sb
```

```
sb.set_style('whitegrid')
from collections import Counter
```

(Multiple) linear regression on the enrollment data

In [9]:

```
address = './Data/enrollment_forecast.csv'

enroll = pd.read_csv(address)
enroll.columns = ['year', 'roll', 'unem', 'hgrad', 'inc']
enroll.head()
```

Out[9]:

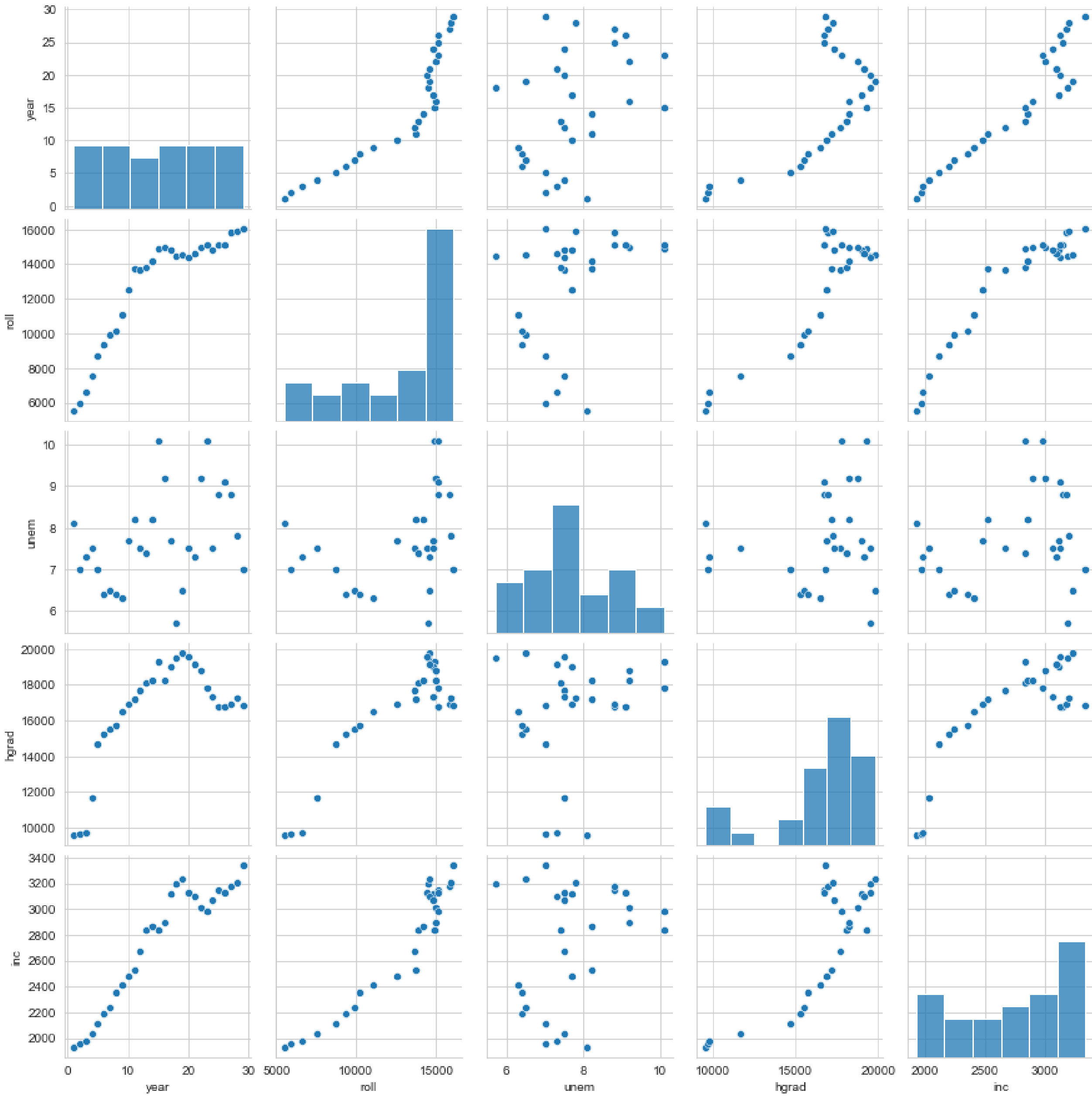
	year	roll	unem	hgrad	inc
0	1	5501	8.1	9552	1923
1	2	5945	7.0	9680	1961
2	3	6629	7.3	9731	1979
3	4	7556	7.5	11666	2030
4	5	8716	7.0	14675	2112

In [10]:

```
sb.pairplot(enroll)
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x1fd1680e9d0>
```



In [11]:
```
print(enroll.corr())
```

	year	roll	unem	hgrad	inc
year	1.000000	0.900934	0.378305	0.670300	0.944287
roll	0.900934	1.000000	0.391344	0.890294	0.949876
unem	0.378305	0.391344	1.000000	0.177376	0.282310
hgrad	0.670300	0.890294	0.177376	1.000000	0.820089
inc	0.944287	0.949876	0.282310	0.820089	1.000000

In [12]:
```
enroll_data = enroll[['unem', 'hgrad']].values

enroll_target = enroll[['roll']].values

enroll_data_names = ['unem', 'hgrad']

X, y =scale(enroll_data), enroll_target
```

Checking for missing values

In [13]:
```
missing_values = X==np.NAN
X[missing_values == True]
```

Out[13]:
```
array([], dtype=float64)
```

In [14]:
```
LinReg = LinearRegression(normalize=True)

LinReg.fit(X, y)

print(LinReg.score(X, y))
```

```
0.8488812666133723
```

Segment 3 - Logistic regression

In [15]:
```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import sklearn

from pandas import Series, DataFrame
from pylab import rcParams
from sklearn import preprocessing
```

In [16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
```

In [17]:
```
%matplotlib inline
rcParams['figure.figsize'] = 5, 4
sb.set_style('whitegrid')
```

In [18]:
```
address = './Data/titanic-training-data.csv'
titanic_training = pd.read_csv(address)
titanic_training.columns = ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
print(titanic_training.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [19]:

```
print(titanic_training.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

VARIABLE DESCRIPTIONS

Survived - Survival (0 = No; 1 = Yes)

Pclass - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)

Name - Name

Sex - Sex

Age - Age

SibSp - Number of Siblings/Spouses Aboard

Parch - Number of Parents/Children Aboard

Ticket - Ticket Number

Fare - Passenger Fare (British pound)

Cabin - Cabin

Embarked - Port of Embarkation (C = Cherbourg, France; Q = Queenstown, UK; S = Southampton - Cobh, Ireland)

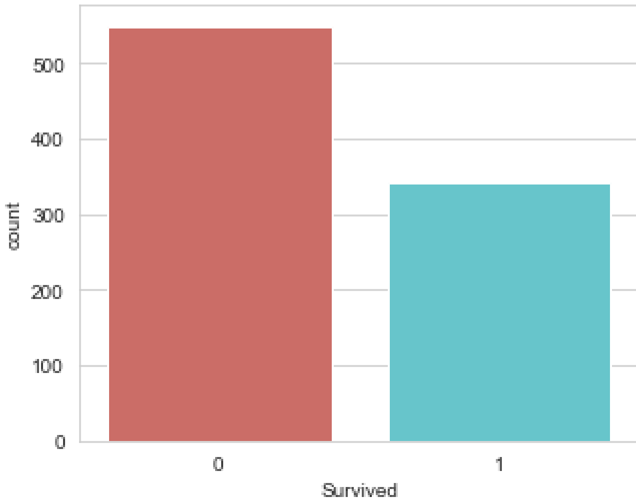
Checking that your target variable is binary

In [20]:

```
sb.countplot(x='Survived', data=titanic_training, palette='hls')
```

Out[20]:

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



Checking for missing values

In [21]:

```
titanic_training.isnull().sum()
```

Out[21]:

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

titanic_training.describe()

Taking care of missing values

Dropping missing values

So let's just go ahead and drop all the variables that aren't relevant for predicting survival. We should at least keep the following:

- Survived - This variable is obviously relevant.
- Pclass - Does a passenger's class on the boat affect their survivability?
- Sex - Could a passenger's gender impact their survival rate?
- Age - Does a person's age impact their survival rate?
- SibSp - Does the number of relatives on the boat (that are siblings or a spouse) affect a person survivability? Probability

- Parch - Does the number of relatives on the boat (that are children or parents) affect a person survivability? Probability
- Fare - Does the fare a person paid effect his survivability? Maybe - let's keep it.
- Embarked - Does a person's point of embarkation matter? It depends on how the boat was filled... Let's keep it.

What about a person's name, ticket number, and passenger ID number? They're irrelavant for predicting survivability. And as you recall, the cabin variable is almost all missing values, so we can just drop all of these.

In [22]:

```
titanic_data = titanic_training.drop(['Name', 'Ticket', 'Cabin'], axis=1)
titanic_data.head()
```

Out[22]:

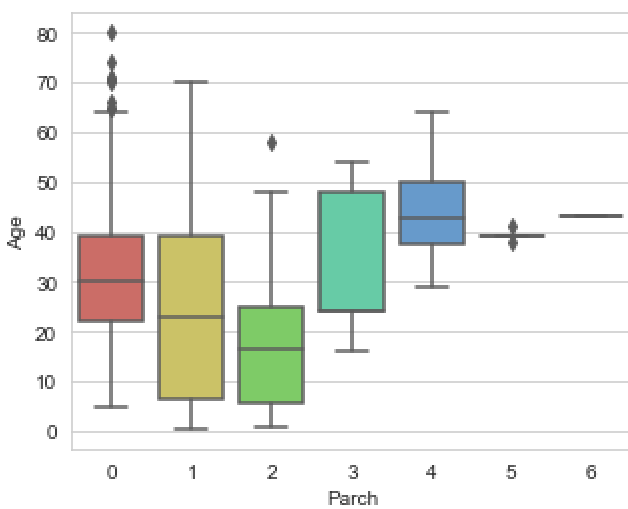
	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

Imputing missing values

In [23]:

```
sb.boxplot(x='Parch', y='Age', data=titanic_data, palette='hls')
```

Out[23]: <AxesSubplot:xlabel='Parch', ylabel='Age'>



In [24]:

```
Parch_groups = titanic_data.groupby(titanic_data['Parch'])
Parch_groups.mean()
```

Out[24]:

	PassengerId	Survived	Pclass	Age	SibSp	Fare
Parch						

Parch	PassengerId	Survived	Pclass	Age	SibSp	Fare
0	445.255162	0.343658	2.321534	32.178503	0.237463	25.586774
1	465.110169	0.550847	2.203390	24.422000	1.084746	46.778180
2	416.662500	0.500000	2.275000	17.216912	2.062500	64.337604
3	579.200000	0.600000	2.600000	33.200000	1.000000	25.951660
4	384.000000	0.000000	2.500000	44.500000	0.750000	84.968750
5	435.200000	0.200000	3.000000	39.200000	0.600000	32.550000
6	679.000000	0.000000	3.000000	43.000000	1.000000	46.900000

In [25]:

```
def age_approx(cols):
    Age = cols[0]
    Parch = cols[1]

    if pd.isnull(Age):
        if Parch == 0:
            return 32
        elif Parch == 1:
            return 24
        elif Parch == 2:
            return 17
        elif Parch == 3:
            return 33
        elif Parch == 4:
            return 45
        else:
            return 30

    else:
        return Age
```

In [26]:

```
titanic_data['Age']= titanic_data[['Age', 'Parch']].apply(age_approx, axis=1)
titanic_data.isnull().sum()
```

Out[26]:

```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       2
dtype: int64
```

In [27]:

```
titanic_data.dropna(inplace=True)
titanic_data.reset_index(inplace=True, drop=True)

print(titanic_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 889 entries, 0 to 888
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Sex          889 non-null    object 
 4   Age          889 non-null    float64
 5   SibSp        889 non-null    int64  
 6   Parch        889 non-null    int64  
 7   Fare         889 non-null    float64
 8   Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(2)
memory usage: 62.6+ KB
None
```

Converting categorical variables to a dummy indicators

In [28]:

```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
gender_cat = titanic_data['Sex']
gender_encoded = label_encoder.fit_transform(gender_cat)
gender_encoded[0:5]
```

Out[28]:

```
array([1, 0, 0, 0, 1])
```

In [29]:

```python
titanic_data.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

In [30]:

```python
# 1 = male / 0 = female
gender_DF = pd.DataFrame(gender_encoded, columns=['male_gender'])
gender_DF.head()
```

Out[30]:

	male_gender
0	1
1	0
2	0
3	0
4	1

In [31]:
```
embarked_cat = titanic_data['Embarked']
embarked_encoded = label_encoder.fit_transform(embarked_cat)
embarked_encoded[0:100]
```

Out[31]:
```
array([2, 0, 2, 2, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 2, 2,
       1, 2, 2, 2, 0, 2, 1, 2, 0, 0, 1, 2, 0, 2, 0, 2, 2, 0, 2, 2, 0, 0,
       1, 2, 1, 1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 2, 2, 0, 2, 2, 0, 0, 2,
       2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2])
```

In [32]:
```
from sklearn.preprocessing import OneHotEncoder
binary_encoder = OneHotEncoder(categories='auto')
embarked_1hot = binary_encoder.fit_transform(embarked_encoded.reshape(-1,1))
embarked_1hot_mat = embarked_1hot.toarray()
embarked_DF = pd.DataFrame(embarked_1hot_mat, columns = ['C', 'Q', 'S'])
embarked_DF.head()
```

Out[32]:

	C	Q	S
0	0.0	0.0	1.0
1	1.0	0.0	0.0
2	0.0	0.0	1.0
3	0.0	0.0	1.0
4	0.0	0.0	1.0

In [33]:
```
titanic_data.drop(['Sex', 'Embarked'], axis=1, inplace=True)
titanic_data.head()
```

Out[33]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	1	0	3	22.0	1	0	7.2500
1	2	1	1	38.0	1	0	71.2833
2	3	1	3	26.0	0	0	7.9250
3	4	1	1	35.0	1	0	53.1000
4	5	0	3	35.0	0	0	8.0500

In [34]:
```
titanic_dmy = pd.concat([titanic_data, gender_DF, embarked_DF], axis=1, verify_integrit
titanic_dmy[0:5]
```

Out[34]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male_gender	C	Q	S
0	1.0	0.0	3.0	22.0	1.0	0.0	7.2500	1.0	0.0	0.0	1.0
1	2.0	1.0	1.0	38.0	1.0	0.0	71.2833	0.0	1.0	0.0	0.0
2	3.0	1.0	3.0	26.0	0.0	0.0	7.9250	0.0	0.0	0.0	1.0
3	4.0	1.0	1.0	35.0	1.0	0.0	53.1000	0.0	0.0	0.0	1.0

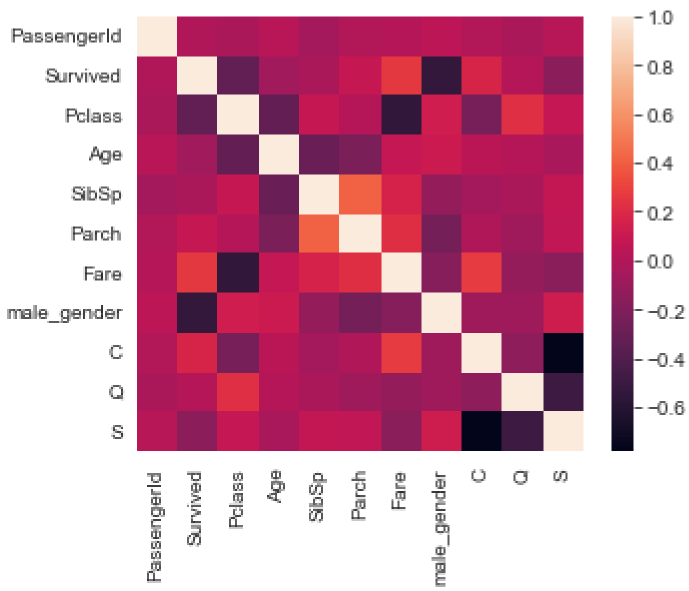
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male_gender	C	Q	S
4	5.0	0.0	3.0	35.0	0.0	0.0	8.0500	1.0	0.0	0.0	1.0

Checking for independence between features

In [35]:

```
sb.heatmap(titanic_dmy.corr())
```

Out[35]: <AxesSubplot:>



In [36]:

```
titanic_dmy.drop(['Fare','Pclass'], axis=1, inplace=True)
titanic_dmy.head()
```

Out[36]:

	PassengerId	Survived	Age	SibSp	Parch	male_gender	C	Q	S
0	1.0	0.0	22.0	1.0	0.0	1.0	0.0	0.0	1.0
1	2.0	1.0	38.0	1.0	0.0	0.0	1.0	0.0	0.0
2	3.0	1.0	26.0	0.0	0.0	0.0	0.0	0.0	1.0
3	4.0	1.0	35.0	1.0	0.0	0.0	0.0	0.0	1.0
4	5.0	0.0	35.0	0.0	0.0	1.0	0.0	0.0	1.0

Checking that your dataset size is sufficient

In [38]:

```
titanic_dmy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 889 entries, 0 to 888
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    float64
 1   Survived     889 non-null    float64
```

```
 2   Age          889 non-null    float64
 3   SibSp        889 non-null    float64
 4   Parch        889 non-null    float64
 5   male_gender  889 non-null    float64
 6   C            889 non-null    float64
 7   Q            889 non-null    float64
 8   S            889 non-null    float64
dtypes: float64(9)
memory usage: 62.6 KB
```

In [39]:

```
X_train, X_test, y_train, y_test = train_test_split(titanic_dmy.drop('Survived', axis=1
                                                    titanic_dmy['Survived'], test_size=0
                                                    random_state=200)
```

In [40]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(711, 8)
(711,)
```

In [41]:

```
X_train[0:5]
```

Out[41]:

	PassengerId	Age	SibSp	Parch	male_gender	C	Q	S
719	721.0	6.0	0.0	1.0	0.0	0.0	0.0	1.0
165	167.0	24.0	0.0	1.0	0.0	0.0	0.0	1.0
879	882.0	33.0	0.0	0.0	1.0	0.0	0.0	1.0
451	453.0	30.0	0.0	0.0	1.0	1.0	0.0	0.0
181	183.0	9.0	4.0	2.0	1.0	0.0	0.0	1.0

Deploying and evaluating the model

In [42]:

```
LogReg = LogisticRegression(solver='liblinear')
LogReg.fit(X_train, y_train)
```

Out[42]:

```
LogisticRegression(solver='liblinear')
```

In [43]:

```
y_pred = LogReg.predict(X_test)
```

Model Evaluation

Classification report without cross-validation

In [44]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.88      0.85       109
```

```
         1.0       0.79      0.71      0.75        69

    accuracy                           0.81       178
   macro avg       0.81      0.80      0.80       178
weighted avg       0.81      0.81      0.81       178
```

K-fold cross-validation & confusion matrices

In [45]:

```
y_train_pred = cross_val_predict(LogReg, X_train, y_train, cv=5)
confusion_matrix(y_train, y_train_pred)
```

Out[45]:

```
array([[377,  63],
       [ 91, 180]], dtype=int64)
```

In [46]:

```
precision_score(y_train, y_train_pred)
```

Out[46]:

```
0.7407407407407407
```

Make a test prediction

In [47]:

```
titanic_dmy[863:864]
```

Out[47]:

	PassengerId	Survived	Age	SibSp	Parch	male_gender	C	Q	S
863	866.0	1.0	42.0	0.0	0.0	0.0	0.0	0.0	1.0

In [48]:

```
test_passenger = np.array([866, 40, 0, 0, 0, 0, 0, 1]).reshape(1,-1)

print(LogReg.predict(test_passenger))
print(LogReg.predict_proba(test_passenger))
```

```
[1.]
[[0.26351831 0.73648169]]
```

Chapter 4 - Clustering Models

Segment 1 - K-means method

Setting up for clustering analysis

In [49]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import sklearn
from sklearn.preprocessing import scale
import sklearn.metrics as sm
from sklearn.metrics import confusion_matrix, classification_report
```

In [50]:
```
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
```

In [51]:
```
%matplotlib inline
plt.figure(figsize=(7,4))
```

Out[51]:
```
<Figure size 504x288 with 0 Axes>
```

```
<Figure size 504x288 with 0 Axes>
```

In [52]:
```
iris = datasets.load_iris()

X = scale(iris.data)
y = pd.DataFrame(iris.target)
variable_names = iris.feature_names
X[0:10]
```

Out[52]:
```
array([[-0.90068117,  1.01900435, -1.34022653, -1.3154443 ],
       [-1.14301691, -0.13197948, -1.34022653, -1.3154443 ],
       [-1.38535265,  0.32841405, -1.39706395, -1.3154443 ],
       [-1.50652052,  0.09821729, -1.2833891 , -1.3154443 ],
       [-1.02184904,  1.24920112, -1.34022653, -1.3154443 ],
       [-0.53717756,  1.93979142, -1.16971425, -1.05217993],
       [-1.50652052,  0.78880759, -1.34022653, -1.18381211],
       [-1.02184904,  0.78880759, -1.2833891 , -1.3154443 ],
       [-1.74885626, -0.36217625, -1.34022653, -1.3154443 ],
       [-1.14301691,  0.09821729, -1.2833891 , -1.44707648]])
```

Building and running your model

In [54]:
```
clustering = KMeans(n_clusters=3, random_state=5)

clustering.fit(X)
```

Out[54]:
```
KMeans(n_clusters=3, random_state=5)
```

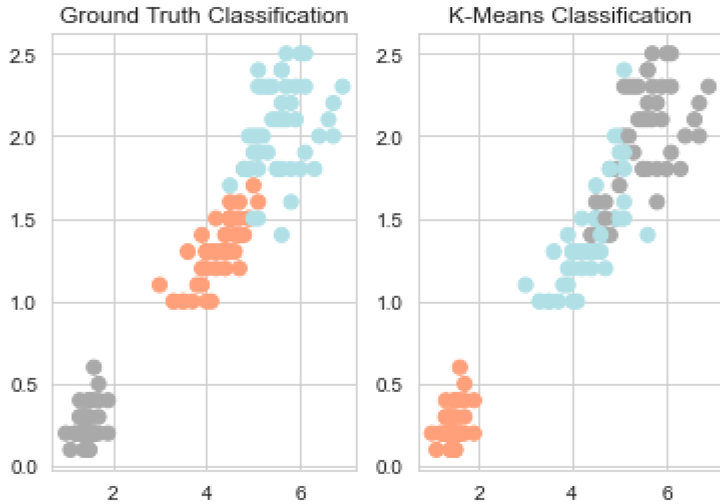
Plotting your model outputs

In [56]:
```
iris_df = pd.DataFrame(iris.data)
iris_df.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
y.columns = ['Targets']
```

In [57]:
```
color_theme = np.array(['darkgray', 'lightsalmon', 'powderblue'])

plt.subplot(1,2,1)

plt.scatter(x=iris_df.Petal_Length, y=iris_df.Petal_Width, c=color_theme[iris.target],
plt.title('Ground Truth Classification')

plt.subplot(1,2,2)
```

```
plt.scatter(x=iris_df.Petal_Length, y=iris_df.Petal_Width, c=color_theme[clustering.lab
plt.title('K-Means Classification')
```

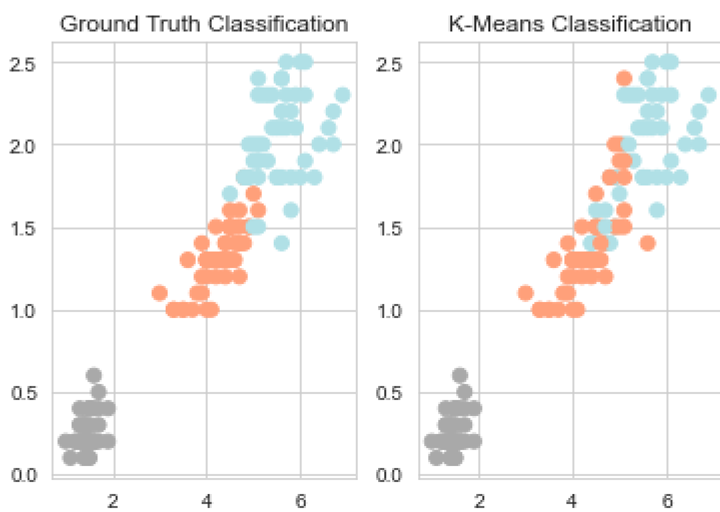
Out[57]: Text(0.5, 1.0, 'K-Means Classification')



In [58]:

```
relabel = np.choose(clustering.labels_, [2, 0, 1]).astype(np.int64)

plt.subplot(1,2,1)

plt.scatter(x=iris_df.Petal_Length, y=iris_df.Petal_Width, c=color_theme[iris.target],
plt.title('Ground Truth Classification')

plt.subplot(1,2,2)

plt.scatter(x=iris_df.Petal_Length, y=iris_df.Petal_Width, c=color_theme[relabel], s=50
plt.title('K-Means Classification')
```

Out[58]: Text(0.5, 1.0, 'K-Means Classification')



Evaluate your clustering results

In [59]:

```
print(classification_report(y, relabel))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.74      0.78      0.76        50
           2       0.77      0.72      0.74        50

    accuracy                           0.83       150
   macro avg       0.83      0.83      0.83       150
weighted avg       0.83      0.83      0.83       150
```

Segment 2 - Hierarchial methods

Setting up for clustering analysis

In [60]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from pylab import rcParams
import seaborn as sb

import sklearn
import sklearn.metrics as sm
```

In [61]:

```
from sklearn.cluster import AgglomerativeClustering

import scipy
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist
```

In [ ]:

```
np.set_printoptions(precision=4, suppress=True)
plt.figure(figsize=(10, 3))
%matplotlib inline
plt.style.use('seaborn-whitegrid')
```

In [62]:

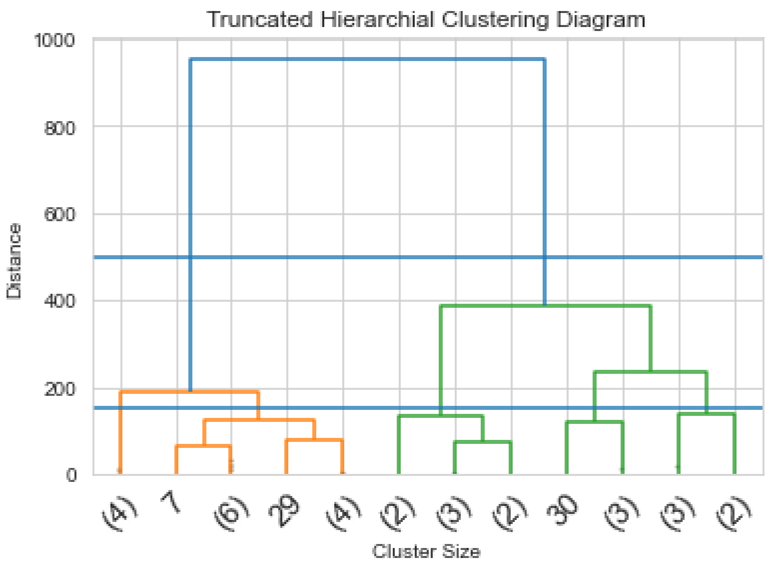
```
address = './Data/mtcars.csv'

cars = pd.read_csv(address)
cars.columns = ['car_names','mpg','cyl','disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',

X = cars[['mpg', 'disp', 'hp', 'wt']].values

y = cars.iloc[:,(9)].values
```

Using scipy to generate dendrograms

In [63]:

```
Z = linkage(X, 'ward')
```

In [64]:
```
dendrogram(Z, truncate_mode='lastp', p=12, leaf_rotation=45., leaf_font_size=15, show_c

plt.title('Truncated Hierarchial Clustering Diagram')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')

plt.axhline(y=500)
plt.axhline(y=150)
plt.show()
```



Generating hierarchical clusters

In [65]:
```
k=2

Hclustering = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage='ward
Hclustering.fit(X)

sm.accuracy_score(y, Hclustering.labels_)
```

Out[65]: 0.78125

In [66]:
```
Hclustering = AgglomerativeClustering(n_clusters=k, affinity='euclidean', linkage='aver
Hclustering.fit(X)

sm.accuracy_score(y, Hclustering.labels_)
```

Out[66]: 0.78125

In [67]:
```
Hclustering = AgglomerativeClustering(n_clusters=k, affinity='manhattan', linkage='aver
Hclustering.fit(X)

sm.accuracy_score(y, Hclustering.labels_)
```

Out[67]: 0.71875

Segment 3 - DBSCan clustering to identify outliers

In [68]:
```
import pandas as pd

import matplotlib.pyplot as plt
from pylab import rcParams
import seaborn as sb

import sklearn
from sklearn.cluster import DBSCAN
from collections import Counter
```

In [69]:
```
%matplotlib inline
rcParams['figure.figsize'] = 5, 4
sb.set_style('whitegrid')
```

DBSCan clustering to identify outliers

Train your model and identify outliers

In [72]:
```
# with this example, we're going to use the same data that we used for the rest of this
# paste in the code.
address = './Data/iris.data.csv'
df = pd.read_csv(address, header=None, sep=',')

df.columns=['Sepal Length','Sepal Width','Petal Length','Petal Width', 'Species']

data = df.iloc[:,0:4].values
target = df.iloc[:,4].values

df[:5]
```

Out[72]:

	Sepal Length	Sepal Width	Petal Length	Petal Width	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 [73]:
```
model = DBSCAN(eps=0.8,  min_samples=19).fit(data)
print(model)
```

```
DBSCAN(eps=0.8, min_samples=19)
```

In [74]:
```
outliers_df = pd.DataFrame(data)

print(Counter(model.labels_))

print(outliers_df[model.labels_ ==-1])
```

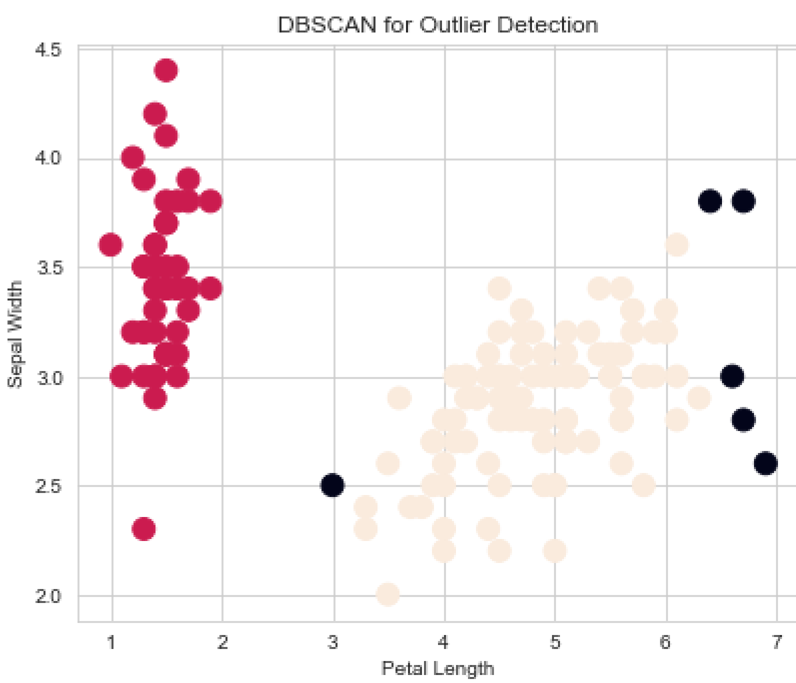
```
Counter({1: 94, 0: 50, -1: 6})
       0    1    2    3
98   5.1  2.5  3.0  1.1
105  7.6  3.0  6.6  2.1
117  7.7  3.8  6.7  2.2
118  7.7  2.6  6.9  2.3
122  7.7  2.8  6.7  2.0
131  7.9  3.8  6.4  2.0
```

In [75]:

```
fig = plt.figure()
ax = fig.add_axes([.1, .1, 1, 1])

colors = model.labels_

ax.scatter(data[:,2], data[:,1], c=colors, s=120)
ax.set_xlabel('Petal Length')
ax.set_ylabel('Sepal Width')
plt.title('DBSCAN for Outlier Detection')
```

Out[75]:

```
Text(0.5, 1.0, 'DBSCAN for Outlier Detection')
```



Chapter 5 - Dimensionality Reduction Methods

Segment 1 - Explanatory factor analysis

In [76]:

```
import pandas as pd
import numpy as np

import sklearn
from sklearn.decomposition import FactorAnalysis

from sklearn import datasets
```

In [77]:
```
iris =  datasets.load_iris()

X = iris.data
variable_names = iris.feature_names

X[0:10,]
```

Out[77]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [78]:
```
factor = FactorAnalysis().fit(X)

DF = pd.DataFrame(factor.components_, columns=variable_names)
print(DF)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0           0.706989         -0.158005           1.654236           0.70085
1           0.115161          0.159635          -0.044321          -0.01403
2          -0.000000          0.000000           0.000000           0.00000
3          -0.000000          0.000000           0.000000          -0.00000
```

Chapter 6 - Other Popular Machine Learning Methods

Segment 1 - Association Rule Mining Using Apriori Algorithm

Import the required libraries

In [25]:
```
import sys
# !{sys.executable} -m pip install mlxtend
```

Out[25]:
```
['C:\\Users\\aadar\\Documents\\TERM2\\BDM 1034 - Application Design for Big Data\\Week10
\\Assignment',
 'S:\\Anaconda\\python39.zip',
 'S:\\Anaconda\\DLLs',
 'S:\\Anaconda\\lib',
 'S:\\Anaconda',
 '',
 'S:\\Anaconda\\lib\\site-packages',
 'S:\\Anaconda\\lib\\site-packages\\locket-0.2.1-py3.9.egg',
 'S:\\Anaconda\\lib\\site-packages\\win32',
 'S:\\Anaconda\\lib\\site-packages\\win32\\lib',
 'S:\\Anaconda\\lib\\site-packages\\Pythonwin',
```

```
 'S:\\Anaconda\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\aadar\\.ipython',
 'C:\\Users\\aadar\\Documents\\TERM2\\BDM 1034 - Application Design for Big Data\\Week1
0']
```

In [20]:

```
import pandas as pd
# import mlxtend
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Data Format

In [21]:

```
address = './Data/groceries.csv'
data = pd.read_csv(address)
```

In [22]:

```
data.head()
```

Out[22]:

	1	2	3	4	5	6	7	8	9
0	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN
1	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN
2	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN
4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN

Data Coversion

In [23]:

```
basket_sets = pd.get_dummies(data)
```

In [24]:

```
basket_sets.head()
```

Out[24]:

	1_Instant food products	1_UHT-milk	1_artif. sweetener	1_baby cosmetics	1_bags	1_baking powder	1_bathroom cleaner	1_beef	1_berries	1_bevera
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

5 rows × 1113 columns

Support Calculation

In [26]:
```
apriori(basket_sets, min_support=0.02)
```

Out[26]:

	support	itemsets
0	0.030421	(7)
1	0.034951	(17)
2	0.029126	(23)
3	0.049191	(26)
4	0.064401	(47)
5	0.044660	(83)
6	0.024272	(90)
7	0.040453	(92)
8	0.038835	(99)
9	0.033981	(100)
10	0.076052	(105)
11	0.028803	(111)
12	0.044984	(123)
13	0.073463	(130)
14	0.022977	(131)
15	0.028803	(159)
16	0.058900	(217)
17	0.022977	(224)
18	0.040129	(232)
19	0.036893	(233)
20	0.031068	(243)
21	0.034628	(256)
22	0.062136	(263)
23	0.028479	(264)
24	0.045955	(351)
25	0.033010	(366)
26	0.024272	(378)
27	0.057929	(397)

	support	itemsets
28	0.023301	(398)
29	0.020712	(479)
30	0.024595	(497)
31	0.024272	(510)
32	0.033333	(531)
33	0.023301	(532)
34	0.020065	(631)
35	0.021036	(217, 397)

In [27]:

```
apriori(basket_sets, min_support=0.02, use_colnames=True)
```

Out[27]:

	support	itemsets
0	0.030421	(1_beef)
1	0.034951	(1_canned beer)
2	0.029126	(1_chicken)
3	0.049191	(1_citrus fruit)
4	0.064401	(1_frankfurter)
5	0.044660	(1_other vegetables)
6	0.024272	(1_pip fruit)
7	0.040453	(1_pork)
8	0.038835	(1_rolls/buns)
9	0.033981	(1_root vegetables)
10	0.076052	(1_sausage)
11	0.028803	(1_soda)
12	0.044984	(1_tropical fruit)
13	0.073463	(1_whole milk)
14	0.022977	(1_yogurt)
15	0.028803	(2_citrus fruit)
16	0.058900	(2_other vegetables)
17	0.022977	(2_pip fruit)
18	0.040129	(2_rolls/buns)
19	0.036893	(2_root vegetables)
20	0.031068	(2_soda)
21	0.034628	(2_tropical fruit)

	support	itemsets
22	0.062136	(2_whole milk)
23	0.028479	(2_yogurt)
24	0.045955	(3_other vegetables)
25	0.033010	(3_rolls/buns)
26	0.024272	(3_soda)
27	0.057929	(3_whole milk)
28	0.023301	(3_yogurt)
29	0.020712	(4_other vegetables)
30	0.024595	(4_rolls/buns)
31	0.024272	(4_soda)
32	0.033333	(4_whole milk)
33	0.023301	(4_yogurt)
34	0.020065	(5_rolls/buns)
35	0.021036	(3_whole milk, 2_other vegetables)

In [28]:

```
df = basket_sets

frequent_itemsets = apriori(basket_sets, min_support=0.002, use_colnames=True)

frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

Out[28]:

	support	itemsets	length
0	0.006472	(1_UHT-milk)	1
1	0.030421	(1_beef)	1
2	0.011974	(1_berries)	1
3	0.008414	(1_beverages)	1
4	0.014887	(1_bottled beer)	1
...	...	...	...
844	0.002265	(5_other vegetables, 6_whole milk, 3_pip fruit)	3
845	0.002589	(5_whole milk, 3_root vegetables, 4_other vege...	3
846	0.002913	(3_whole milk, 4_curd, 5_yogurt)	3
847	0.003236	(4_root vegetables, 5_other vegetables, 6_whol...	3
848	0.002265	(5_other vegetables, 7_butter, 6_whole milk)	3

849 rows × 3 columns

In [29]:
```
frequent_itemsets[frequent_itemsets['length'] >= 3]
```

Out[29]:

	support	itemsets	length
820	0.002589	(1_beef, 3_other vegetables, 2_root vegetables)	3
821	0.002589	(3_whole milk, 1_chicken, 2_other vegetables)	3
822	0.002589	(3_whole milk, 2_other vegetables, 1_citrus fr...	3
823	0.003236	(2_tropical fruit, 3_pip fruit, 1_citrus fruit)	3
824	0.002589	(4_whole milk, 1_citrus fruit, 3_other vegetab...	3
825	0.002265	(5_other vegetables, 6_whole milk, 1_frankfurter)	3
826	0.002265	(4_whole milk, 1_pork, 3_other vegetables)	3
827	0.003560	(3_whole milk, 2_other vegetables, 1_root vege...	3
828	0.002589	(1_sausage, 3_soda, 2_rolls/buns)	3
829	0.002265	(4_whole milk, 1_sausage, 3_other vegetables)	3
830	0.002265	(5_whole milk, 1_sausage, 4_other vegetables)	3
831	0.002913	(3_whole milk, 2_other vegetables, 1_tropical ...	3
832	0.002265	(5_whole milk, 2_citrus fruit, 4_other vegetab...	3
833	0.002265	(3_whole milk, 2_other vegetables, 4_butter)	3
834	0.003560	(3_whole milk, 2_other vegetables, 4_curd)	3
835	0.003883	(3_whole milk, 2_other vegetables, 4_yogurt)	3
836	0.002265	(3_whole milk, 2_other vegetables, 6_rolls/buns)	3
837	0.003236	(4_whole milk, 2_pip fruit, 3_other vegetables)	3
838	0.005825	(4_whole milk, 3_other vegetables, 2_root vege...	3
839	0.002265	(4_other vegetables, 2_tropical fruit, 3_pip f...	3
840	0.003560	(4_whole milk, 5_butter, 3_other vegetables)	3
841	0.002913	(4_whole milk, 3_other vegetables, 5_yogurt)	3
842	0.003560	(4_whole milk, 6_yogurt, 3_other vegetables)	3
843	0.002265	(4_root vegetables, 5_other vegetables, 3_pip ...	3
844	0.002265	(5_other vegetables, 6_whole milk, 3_pip fruit)	3
845	0.002589	(5_whole milk, 3_root vegetables, 4_other vege...	3
846	0.002913	(3_whole milk, 4_curd, 5_yogurt)	3
847	0.003236	(4_root vegetables, 5_other vegetables, 6_whol...	3
848	0.002265	(5_other vegetables, 7_butter, 6_whole milk)	3

Association Rules

Confidence

In [30]:

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
rules.head()
```

Out[30]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conv
0	(2_sausage)	(1_frankfurter)	0.011327	0.064401	0.011327	1.000000	15.527638	0.010597	
1	(7_pastry)	(1_frankfurter)	0.005178	0.064401	0.002589	0.500000	7.763819	0.002256	1.8
2	(2_ham)	(1_sausage)	0.007120	0.076052	0.004531	0.636364	8.367505	0.003989	2.5
3	(2_meat)	(1_sausage)	0.006796	0.076052	0.004854	0.714286	9.392097	0.004338	3.2
4	(3_beef)	(1_sausage)	0.004854	0.076052	0.002589	0.533333	7.012766	0.002220	1.9

Lift

In [32]:

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()
```

Out[32]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convict
0	(1_beef)	(2_citrus fruit)	0.030421	0.028803	0.005502	0.180851	6.278986	0.004625	1.185
1	(2_citrus fruit)	(1_beef)	0.028803	0.030421	0.005502	0.191011	6.278986	0.004625	1.198
2	(1_beef)	(2_other vegetables)	0.030421	0.058900	0.003236	0.106383	1.806173	0.001444	1.053
3	(2_other vegetables)	(1_beef)	0.058900	0.030421	0.003236	0.054945	1.806173	0.001444	1.025
4	(1_beef)	(2_root vegetables)	0.030421	0.036893	0.005502	0.180851	4.902016	0.004379	1.175

Lift and Confidence

In [34]:

```
rules[(rules['lift'] >= 5) & (rules['confidence']>= 0.5)]
```

Out[34]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	co
93	(2_sausage)	(1_frankfurter)	0.011327	0.064401	0.011327	1.000000	15.527638	0.010597	
137	(7_pastry)	(1_frankfurter)	0.005178	0.064401	0.002589	0.500000	7.763819	0.002256	

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	co
239	(2_ham)	(1_sausage)	0.007120	0.076052	0.004531	0.636364	8.367505	0.003989	2
243	(2_meat)	(1_sausage)	0.006796	0.076052	0.004854	0.714286	9.392097	0.004338	3
259	(3_beef)	(1_sausage)	0.004854	0.076052	0.002589	0.533333	7.012766	0.002220	1
...	...	...	...	...	...	...	...	...	
958	(4_root vegetables, 5_other vegetables)	(6_whole milk)	0.005178	0.009385	0.003236	0.625000	66.594828	0.003188	2
959	(4_root vegetables, 6_whole milk)	(5_other vegetables)	0.003883	0.012621	0.003236	0.833333	66.025641	0.003187	5
964	(5_other vegetables, 7_butter)	(6_whole milk)	0.002589	0.009385	0.002265	0.875000	93.232759	0.002241	7
966	(7_butter, 6_whole milk)	(5_other vegetables)	0.002913	0.012621	0.002265	0.777778	61.623932	0.002229	4
968	(7_butter)	(5_other vegetables, 6_whole milk)	0.004207	0.007443	0.002265	0.538462	72.341137	0.002234	2

76 rows × 9 columns

Chapter 6 - Other Popular Machine Learning Methods

Segment 2 - A neural network with a Perceptron

In [35]:

```
import numpy as np
import pandas as pd
import sklearn

from pandas import Series, DataFrame
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

In [36]:

```
from sklearn.linear_model import Perceptron
```

In [37]:

```
iris = datasets.load_iris()
```

```
X = iris.data
y = iris.target

X[0:10,]
```

Out[37]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [39]:
```
standardize = StandardScaler()

standardized_X_test = standardize.fit_transform(X_test)

standardized_X_train = standardize.fit_transform(X_train)
```

In [40]:
```
standardized_X_test[0:10,]
```

Out[40]:
```
array([[-0.49621415,  1.4716667 , -1.04648076, -0.93812902],
       [ 0.3881279 , -1.89402845,  0.92313132,  0.65765746],
       [-1.23316587, -1.30008224,  0.62470525,  0.94780045],
       [ 0.53551825, -0.50815397,  0.74407568,  0.51258596],
       [-1.97011758, -0.50815397, -1.22553641, -1.22827201],
       [-0.93838518,  1.27368463, -1.10616598, -1.22827201],
       [ 1.12507962, -0.70613604,  0.68439046,  0.65765746],
       [ 0.09334722, -0.90411811,  0.38596439, -0.06770003],
       [ 0.83029893, -0.70613604,  0.98281654,  0.65765746],
       [ 0.09334722, -1.10210018,  0.32627917,  0.22244296]])
```

In [41]:
```
perceptron = Perceptron(max_iter=50, eta0=0.15, tol=1e-3, random_state=15)

perceptron.fit(standardized_X_train, y_train.ravel())
```

Out[41]:
```
Perceptron(eta0=0.15, max_iter=50, random_state=15)
```

In [42]:
```
y_pred = perceptron.predict(standardized_X_test)
```

In [43]:
```
print(y_test)
```

```
[0 2 2 1 0 0 1 1 2 1 1 2 2 2 0 0 0 2 0 0 0 1 1 1 2 1 0 2 0 0]
```

In [44]:
```
print(y_pred)
```

```
[0 1 1 1 0 0 2 1 2 1 1 2 2 2 0 0 0 2 0 0 0 1 1 2 2 1 0 2 0 0]
```

In [45]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.78      0.78      0.78         9
           2       0.78      0.78      0.78         9

    accuracy                           0.87        30
   macro avg       0.85      0.85      0.85        30
weighted avg       0.87      0.87      0.87        30
```

Segment 3 - Instance-based learning w/ k-Nearest Neighbor

Setting up for classification analysis

In [46]:

```
import numpy as np
import pandas as pd
import scipy
import urllib
import sklearn

import matplotlib.pyplot as plt
from pylab import rcParams

from sklearn import neighbors
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [47]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [48]:

```
np.set_printoptions(precision=4, suppress=True)
%matplotlib inline
rcParams['figure.figsize'] = 7, 4
plt.style.use('seaborn-whitegrid')
```

In [49]:

```
address = './Data/mtcars.csv'

cars = pd.read_csv(address)
cars.columns = ['car_names','mpg','cyl','disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',

X_prime = cars[['mpg', 'disp', 'hp', 'wt']].values
y = cars.iloc[:,9].values
```

In [50]:

```
X = preprocessing.scale(X_prime)
```

In [51]:
```
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=.2, random_state=17)
```

Building and training your model with training data

In [53]:
```
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train, y_train)
print(clf)
```

```
KNeighborsClassifier()
```

Evaluating your model's predictions

In [55]:
```
y_pred= clf.predict(X_test)
y_expect = y_test

print(metrics.classification_report(y_expect, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89         4
           1       1.00      0.67      0.80         3

    accuracy                           0.86         7
   macro avg       0.90      0.83      0.84         7
weighted avg       0.89      0.86      0.85         7
```

Segment 5 - Naive Bayes Classifiers

In [56]:
```
import numpy as np
import pandas as pd
import urllib
import sklearn

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

In [57]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

Naive Bayes

Using Naive Bayes to predict spam

In [58]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data

import urllib.request
```

```
raw_data = urllib.request.urlopen(url)
dataset = np.loadtxt(raw_data, delimiter=',')
print(dataset[0])
```

```
[  0.      0.64    0.64    0.      0.32    0.      0.      0.      0.
   0.      0.      0.64    0.      0.      0.      0.32    0.      1.29
   1.93    0.      0.96    0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.      0.      0.      0.      0.      0.      0.778   0.      0.
   3.756  61.    278.      1.    ]
```

In [59]:

```
X = dataset[:,0:48]

y = dataset[:,-1]
```

In [60]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=17
```

In [61]:

```
BernNB = BernoulliNB(binarize=True)
BernNB.fit(X_train, y_train)
print(BernNB)

y_expect = y_test
y_pred = BernNB.predict(X_test)

print(accuracy_score(y_expect, y_pred))
```

```
BernoulliNB(binarize=True)
0.8577633007600435
```

In [62]:

```
MultiNB = MultinomialNB()
MultiNB.fit(X_train, y_train)
print(MultiNB)


y_pred = MultiNB.predict(X_test)

print(accuracy_score(y_expect, y_pred))
```

```
MultinomialNB()
0.8816503800217155
```

In [63]:

```
GausNB = GaussianNB()
GausNB.fit(X_train, y_train)
print(GausNB)


y_pred = GausNB.predict(X_test)

print(accuracy_score(y_expect, y_pred))
```

```
GaussianNB()
0.8197611292073833
```

In [64]:

```
BernNB = BernoulliNB(binarize=0.1)
```

```
BernNB.fit(X_train, y_train)
print(BernNB)

y_expect = y_test
y_pred = BernNB.predict(X_test)

print(accuracy_score(y_expect, y_pred))
```

```
BernoulliNB(binarize=0.1)
0.9109663409337676
```

Segment 6 - Ensemble methods with random forest

This is a classification problem, where in we will be estimating the species label for iris flowers.

In [65]:

```
import numpy as np
import pandas as pd

import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [66]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [67]:

```
iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.DataFrame(iris.target)

y.columns = ['labels']

print(df.head())
y[0:5]
```

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

Out[67]:

	labels
0	0
1	0
2	0
3	0
4	0

The data set contains information on the:

- sepal length (cm)

- sepal width (cm)
- petal length (cm)
- petal width (cm)
- species type

In [69]:
```
df.isnull().any()==True
```

Out[69]:
```
sepal length (cm)    False
sepal width (cm)     False
petal length (cm)    False
petal width (cm)     False
dtype: bool
```

In [70]:
```
print(y.labels.value_counts())
```

```
0    50
1    50
2    50
Name: labels, dtype: int64
```

Preparing the data for training the model

In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=.2, random_state=1
```

Build a Random Forest model

In [74]:
```
classifier = RandomForestClassifier(n_estimators=200, random_state=0)

y_train_array = np.ravel(y_train)

classifier.fit(X_train, y_train_array)

y_pred = classifier.predict(X_test)
```

Evaluating the model on the test data

In [75]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.92      1.00      0.96        11
           2       1.00      0.92      0.96        12

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [76]:
```
y_test_array = np.ravel(y_test)
print(y_test_array)
```

```
[0 1 2 1 2 2 1 2 1 2 2 0 1 0 2 0 0 2 2 2 2 0 2 1 1 1 1 1 0 1]
```

In [77]:
```
print(y_pred)
```

```
[0 1 2 1 2 2 1 2 1 2 2 0 1 0 2 0 0 2 2 2 1 0 2 1 1 1 1 1 0 1]
```

In [ ]: