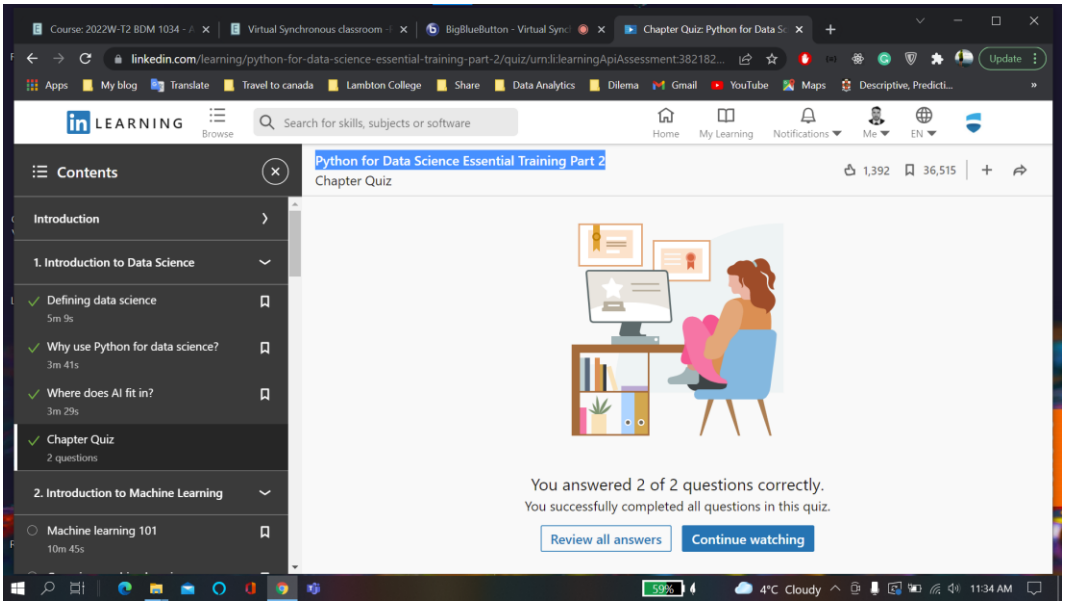
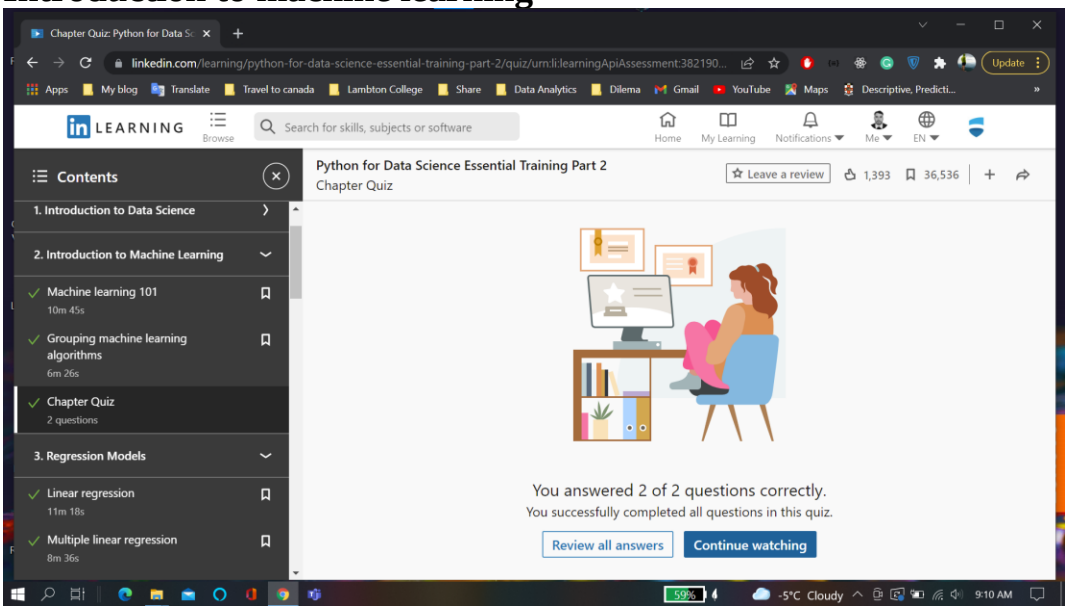
BDM 1034 - Application Design for Big Data
Week10
Submitted by: Aadarsha Chapagain
Student ID:C0825975
Submitted to: Prof. Teresa Zhu

Here I have attached the chapter quiz and exercise I have done from Linkedin learning for Course "**Python for Data Science Essential Training Part 2"** and **"Python function for Data Science**" along with the Screenshot of score I achieved.

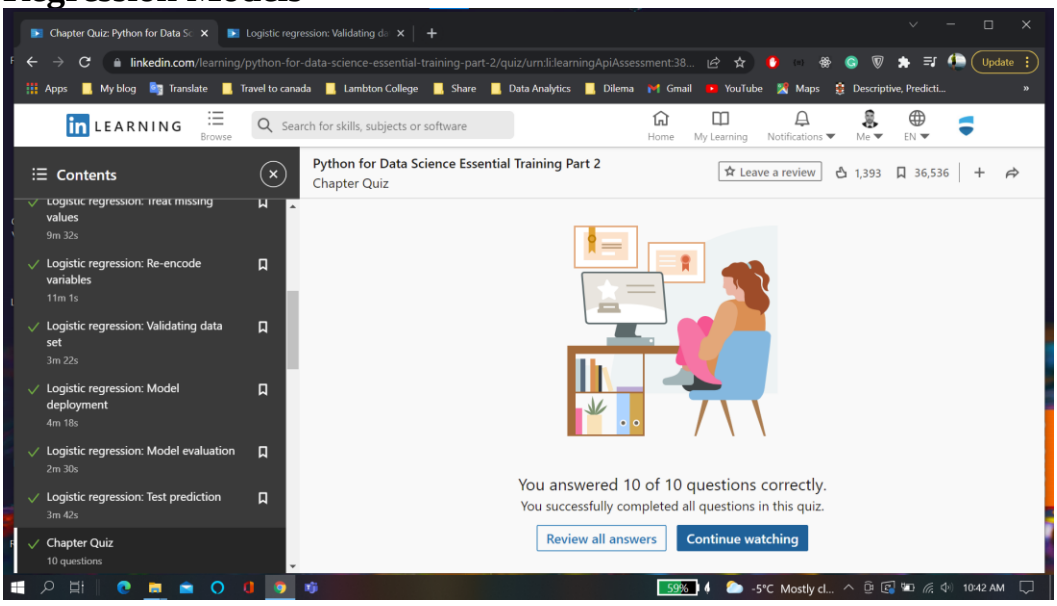
Chapter 1:
Introduction



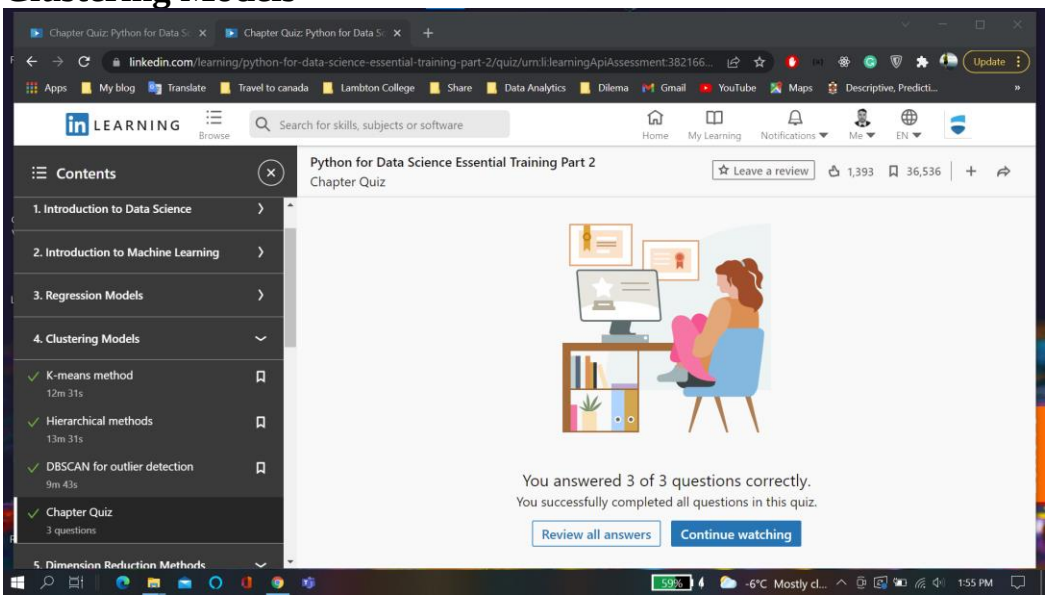
Chapter 2
Introduction to machine learning



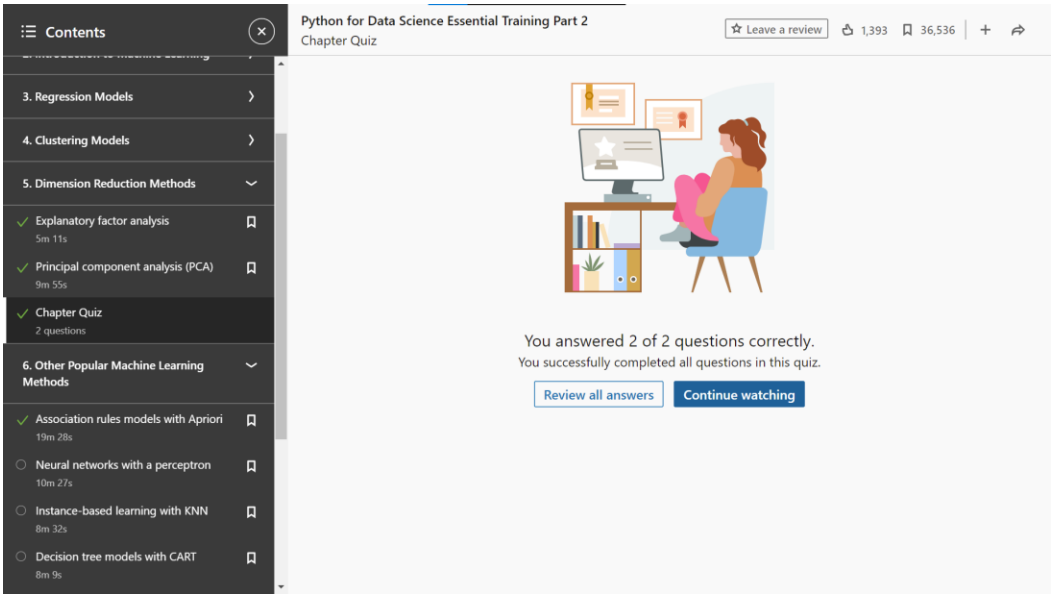
Chapter 3:
Regression Models



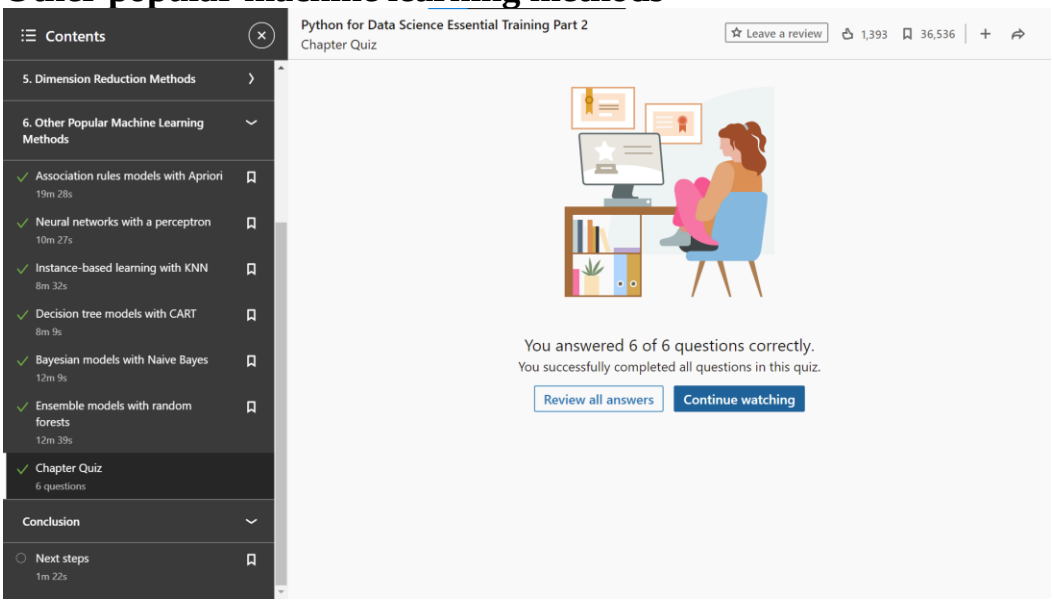
Chapter 4:
Clustering Models



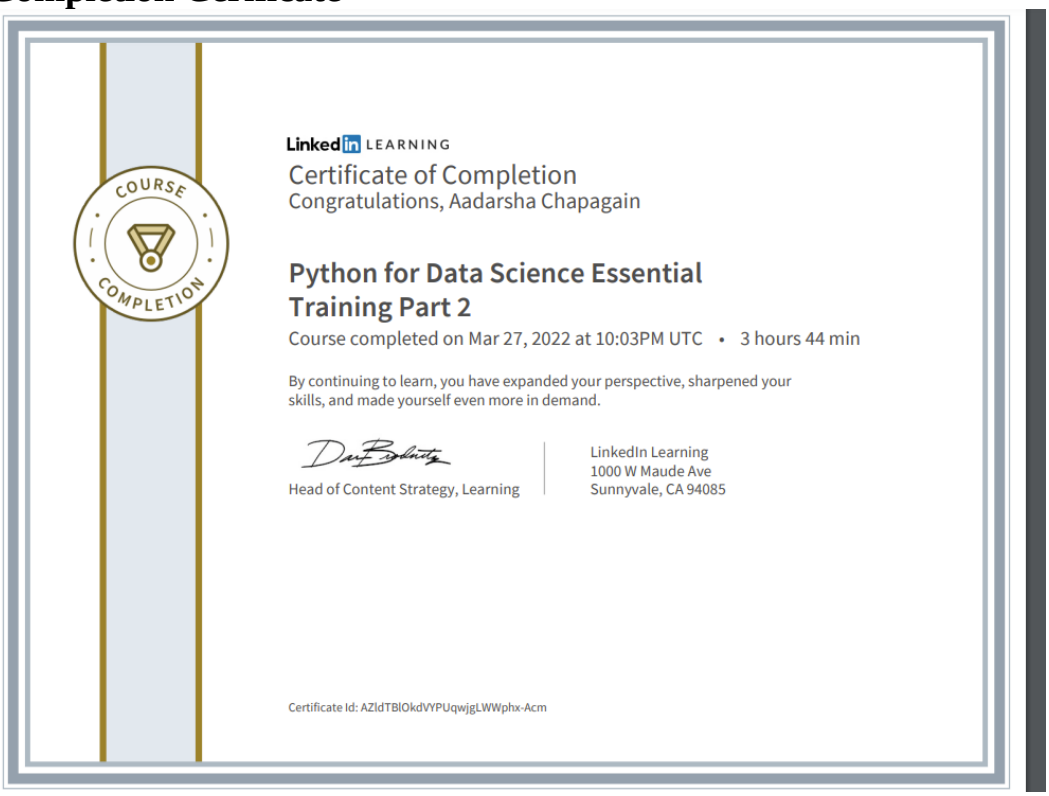
Chapter 5:
Dimension Reduction Method:



Chapter 6:
Other popular machine learning methods



Completion Cerificate



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
```

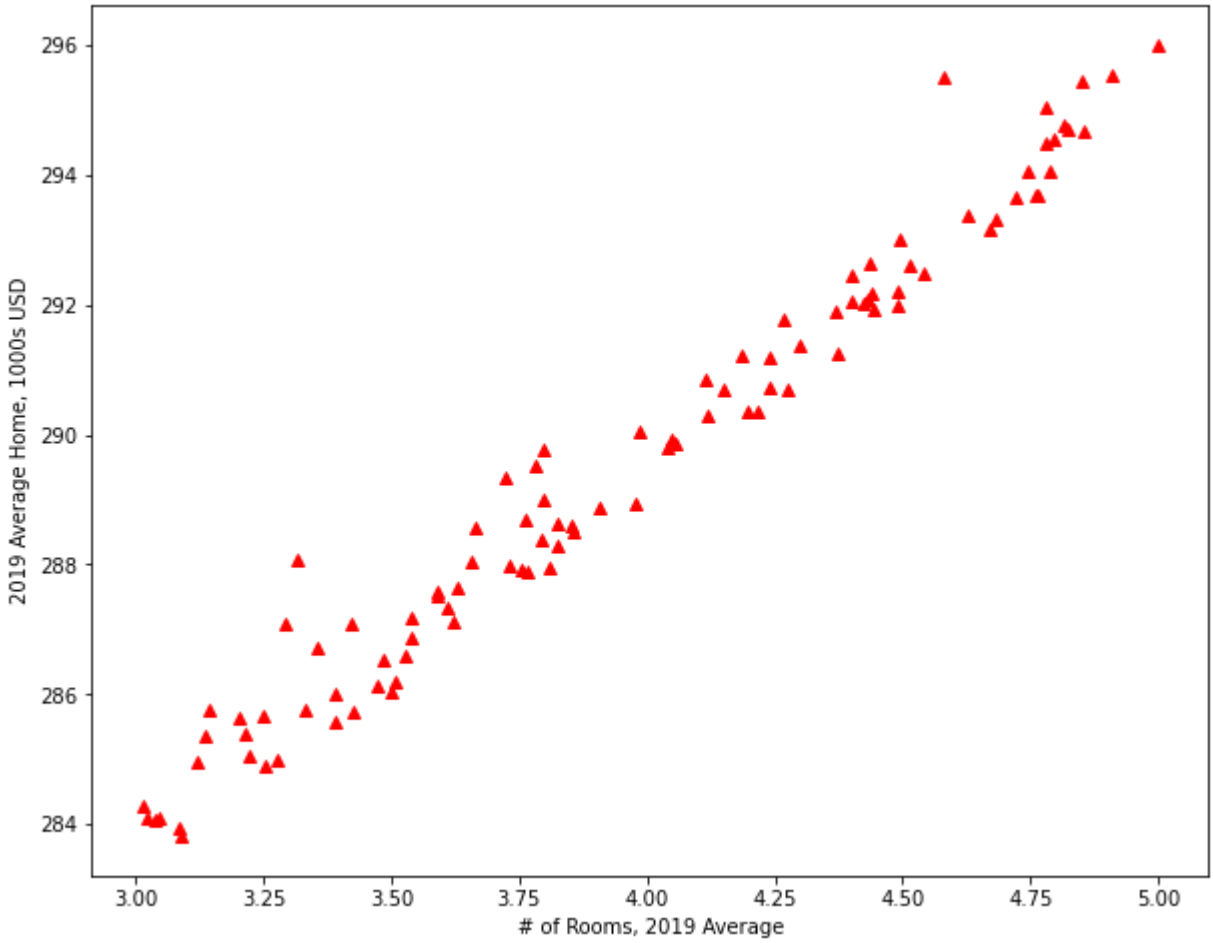
In [3]:
```
rooms = 2*np.random.rand(100, 1)+3
rooms[1:10]
```

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

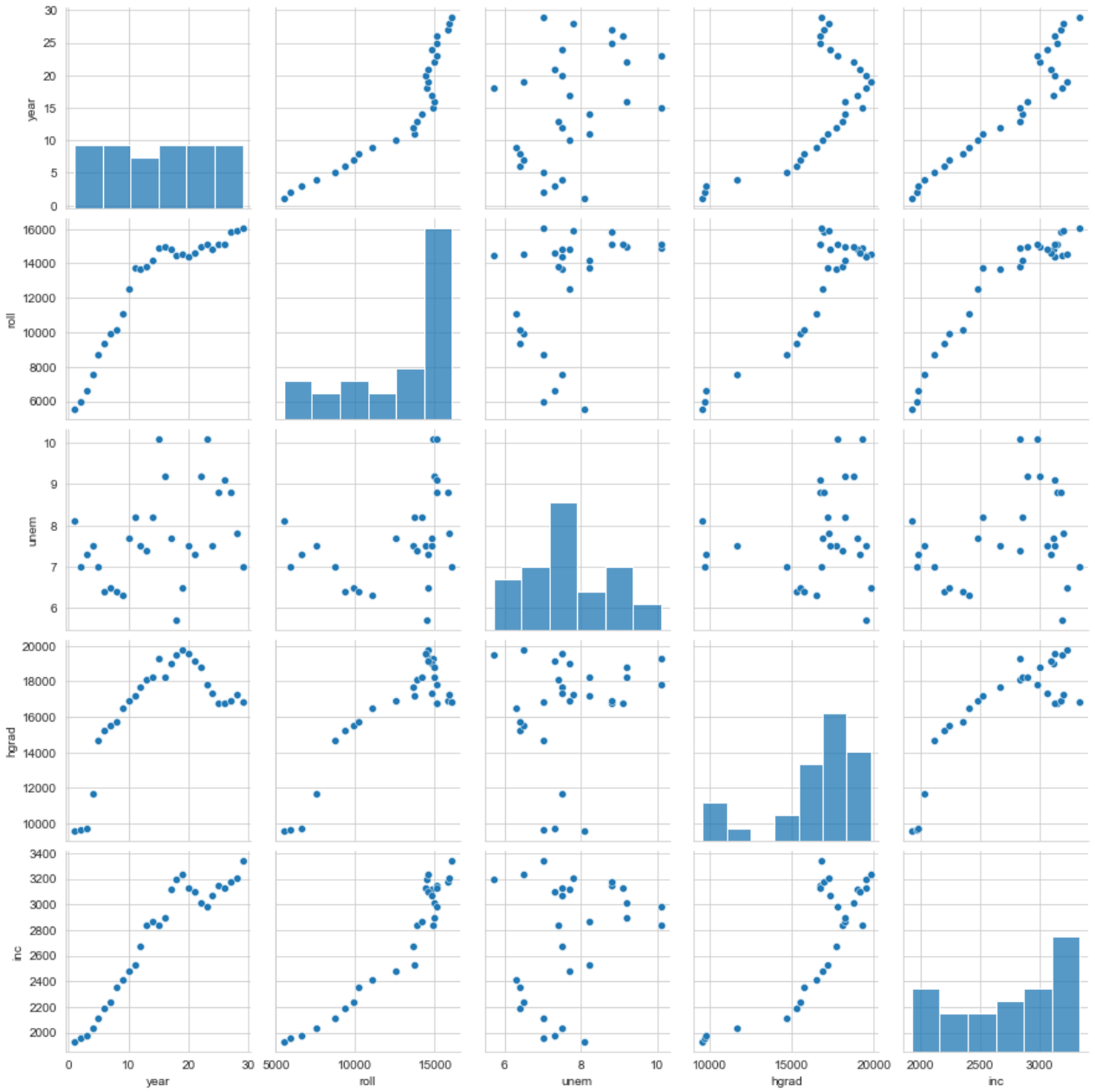
In [9]:
```
address = './Data/enrollment_forecast.csv'

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

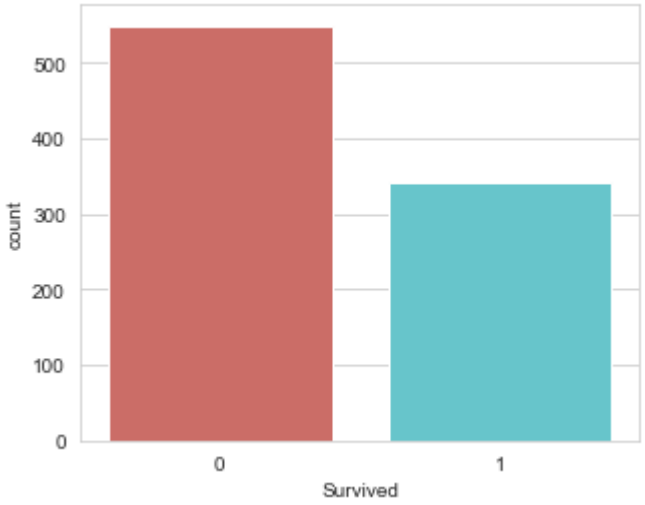
Embarked - Port of Embarkation (C = Cherbourg, France; Q = Queenstown, UK; S = Southampton - Cobh, Ireland)

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()
```

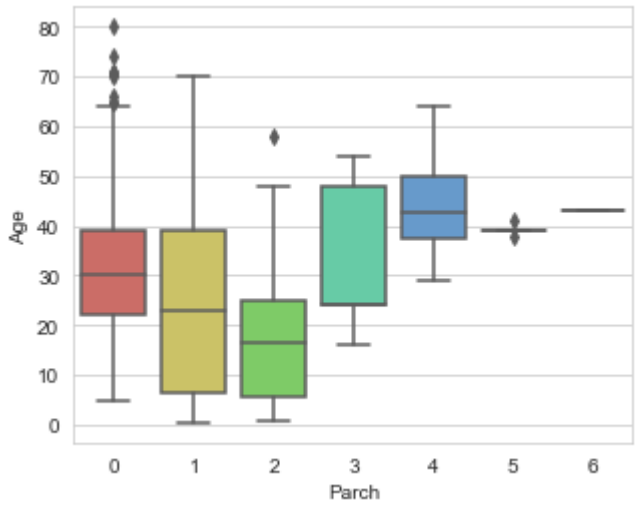
Out[22]:

	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]:
```
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]:
```
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]:
```
# 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]:

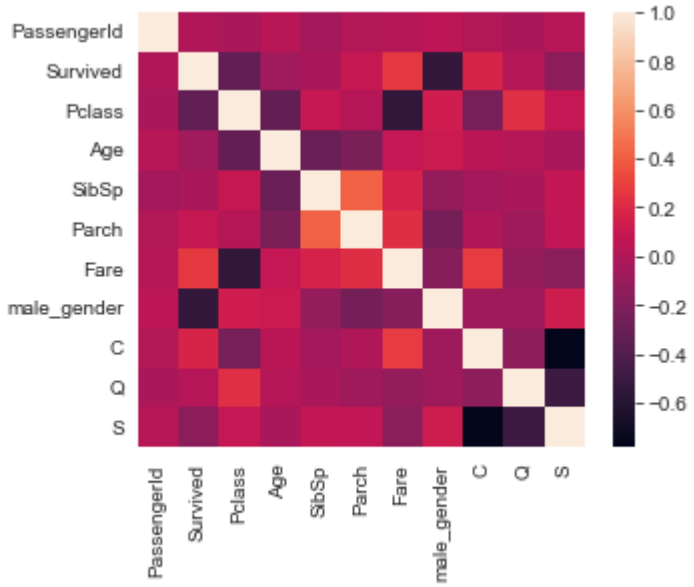
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male_gender	C	Q	S
0	1.0	0.0	3.0	22.0	1.0	0.0	7.2500	1.0	0.0	0.0	1.0
1	2.0	1.0	1.0	38.0	1.0	0.0	71.2833	0.0	1.0	0.0	0.0
2	3.0	1.0	3.0	26.0	0.0	0.0	7.9250	0.0	0.0	0.0	1.0
3	4.0	1.0	1.0	35.0	1.0	0.0	53.1000	0.0	0.0	0.0	1.0

	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)
```

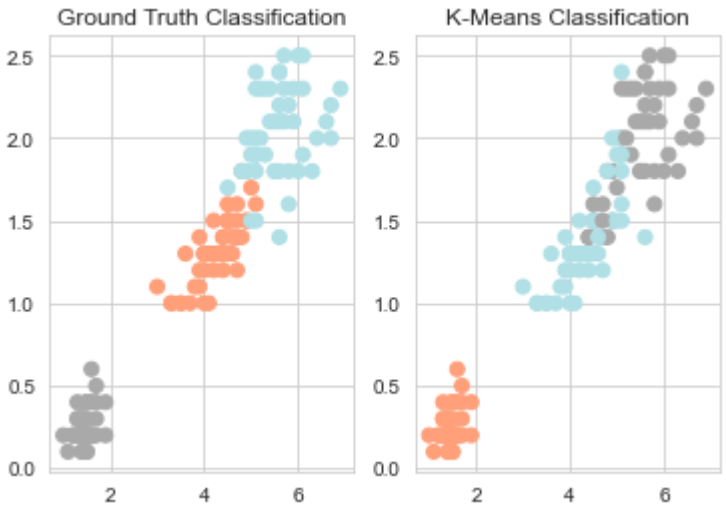
Plotting your model outputs

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)
```

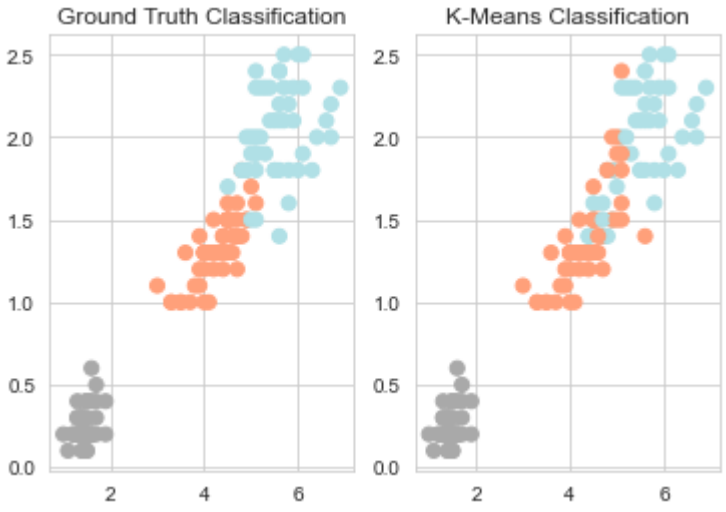
```
plt.scatter(x=iris_df.Petal_Length, y=iris_df.Petal_Width, c=color_theme[clustering.lab
plt.title('K-Means Classification')
```

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
```

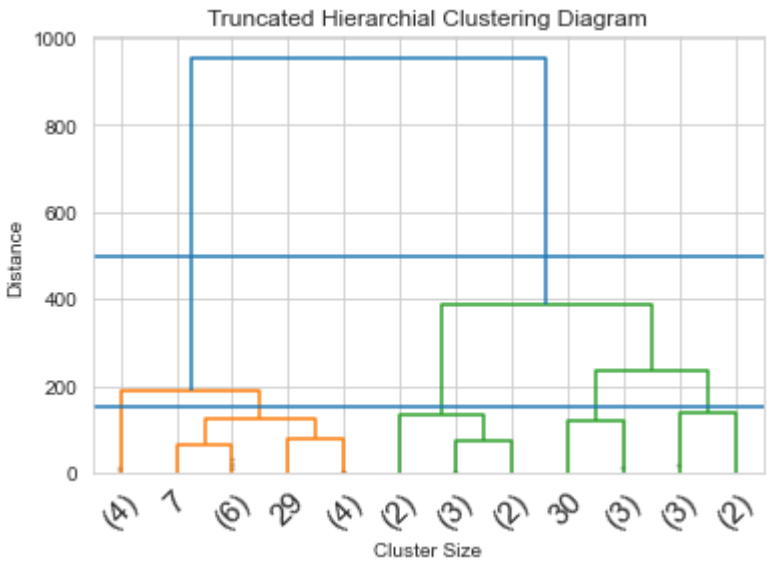
In [ ]:
```
np.set_printoptions(precision=4, suppress=True)
plt.figure(figsize=(10, 3))
%matplotlib inline
plt.style.use('seaborn-whitegrid')
```

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])
```

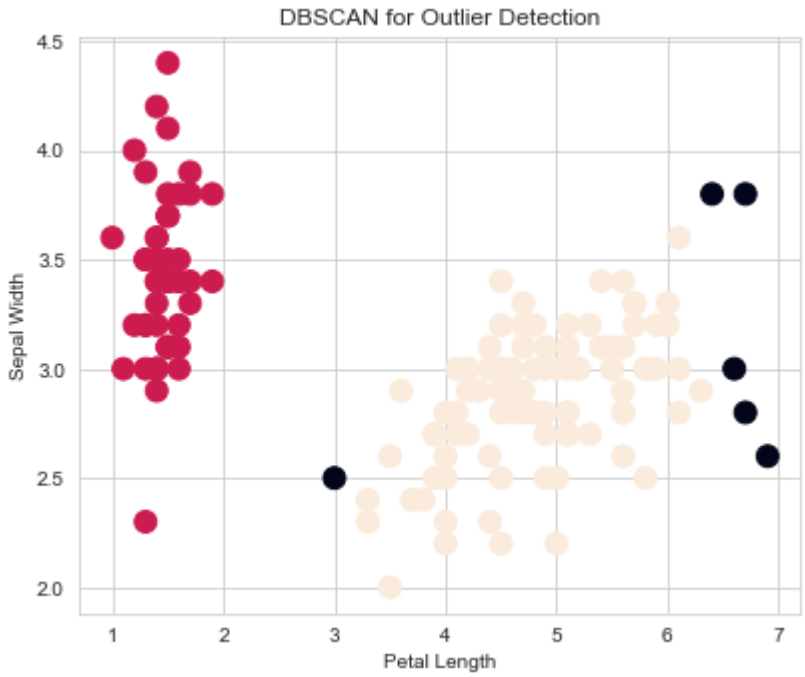
```
Counter({1: 94, 0: 50, -1: 6})
       0    1    2    3
98   5.1  2.5  3.0  1.1
105  7.6  3.0  6.6  2.1
117  7.7  3.8  6.7  2.2
118  7.7  2.6  6.9  2.3
122  7.7  2.8  6.7  2.0
131  7.9  3.8  6.4  2.0
```

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\\Week10']
```

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 [ ]:
```

```

In [1]:
```
# import relevant modules
import pandas as pd
```

In [2]:
```
# read grades dataset, save as a pandas dataframe
grades = pd.read_csv('./grades.csv')
```

In [3]:
```
# display first few rows of grades
grades.head()
```

Out[3]:

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [4]:
```
def lowest_grade(student_id):

    """Find lowest grade across all exams for student with given student_id.
    Treat missing exam grades as zeros."""

    return grades.loc[grades['student_id'] == student_id]['grade'].fillna(0).min()
```

In [5]:
```
# test lowest_grade on student_id 1
assert lowest_grade(1) == 0.0, 'test failed'
print('test passed')
```

```
test passed
```

In [6]:
```
# sequence containing all distinct student ids
student_ids = grades['student_id'].unique()
student_ids
```

Out[6]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int64)
```

In [7]:
```
# apply lowest_grade to each student id
list(map(lowest_grade, student_ids))
```

Out[7]:
```
[0.0, 0.0, 70.0, 0.0, 0.0, 0.0, 75.0, 56.0, 73.0, 75.0]
```

Zip

In [8]:
```
# points scored by each of five players in game1
points_game1 = [50, 40, 60, 70, 80]
```

In [9]:
```
# points scored by each of five players in game2
points_game2 = [76, 81, 53, 92, 67]
```

In [10]:
```
# sequence where points differences will be saved
diffs = []

# iterate through points_game1 and points_game2 at the same time
for x, y in zip(points_game1, points_game2):
    # compute absolute difference in points between game1 and game2
    # add to diffs
    diffs.append(abs(x - y))

diffs
```

Out[10]:
```
[26, 41, 7, 22, 13]
```

Filter

In [11]:
```
def mean_atleast_70(student_id):

    """Compute mean grade across all exams for student with given student_id.
    Treat missing exam grades as zeros.
    If mean grade is atleast 70, return True. Otherwise, return False."""

    mean_grade = grades.loc[grades['student_id'] == student_id]['grade'].fillna(0).mean
    return mean_grade >= 70
```

In [12]:
```
# test mean_grade on student_id 1
assert mean_atleast_70(1) == False, 'test failed'
print('test passed')
```

```
test passed
```

In [13]:
```
# sequence containing all distinct student ids
student_ids = grades['student_id'].unique()
student_ids
```

Out[13]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int64)
```

In [14]:
```
list(filter(mean_atleast_70, student_ids))
```

Out[14]:
```
[3, 5, 7, 8, 9, 10]
```

Numpy

In [16]:
```
# import relevant libraries
import numpy as np
```

In [17]:
```
# create an empty numpy array & save in a variable
array0 = np.array([])
array0
```

Out[17]:
```
array([], dtype=float64)
```

In [18]:
```
# initialize list1 as a Python list
list1 = [1, 2, 3, 4, 5]
```

In [19]:
```
# create a one-dimensional numpy array from list1 & save in a variable
array1 = np.array(list1)
array1
```

Out[19]:
```
array([1, 2, 3, 4, 5])
```

In [20]:
```
# initialize list2 as a nested Python list
list2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [21]:
```
# create a two-dimensional numpy array from list2
array2 = np.array(list2)
array2
```

Out[21]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [22]:
```
# create a numpy array containing integers 0 to 4 including 0 and 4
np.arange(5)
```

Out[22]:
```
array([0, 1, 2, 3, 4])
```

In [23]:
```
# create a numpy array containing all even integers between 0 and 10 including 0 and 10
np.arange(0, 11, 2)
```

Out[23]:
```
array([ 0,  2,  4,  6,  8, 10])
```

In [24]:
```
# create a one-dimenstional numpy array containing 5 zeros
np.zeros(5)
```

Out[24]:
```
array([0., 0., 0., 0., 0.])
```

In [25]:
```
# create a two-dimensional numpy array of zeros having 4 rows and 5 columns
np.zeros((4, 5))
```

Out[25]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [26]:
```
# create a one-dimenstional numpy array containing 6 ones
np.ones(6)
```

Out[26]:
```
array([1., 1., 1., 1., 1., 1.])
```

In [27]:
```
# create a two-dimensional numpy array of ones having 4 rows and 6 columns
np.ones((4, 6))
```

Out[27]:
```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

In [28]:
```
# create a numpy array of 9 evenly spaced numbers from 1 to 2, including 1 and 2
np.linspace(1, 2, 9)
```

Out[28]:
```
array([1.   , 1.125, 1.25 , 1.375, 1.5  , 1.625, 1.75 , 1.875, 2.   ])
```

In [29]:
```
# create a numpy array of 10 random integers from 20 to 50
np.random.randint(20, 50, 10)
```

Out[29]:
```
array([42, 44, 48, 37, 38, 43, 43, 33, 42, 38])
```

Min_max

In [30]:
```
# import relevant libraries
import numpy as np
```

In [31]:
```
# create a numpy array of 20 random integers from 1 to 50 & save in a variable
array_random = np.random.randint(1, 50, 20)
array_random
```

Out[31]:
```
array([11, 20, 48, 46, 41, 32, 21, 38, 10,  7, 49,  6,  9, 18, 22, 24,  4,
        4, 14, 10])
```

In [32]:
```
# find minimum value in array_random
array_random.min()
```

Out[32]:
```
4
```

In [33]:
```
# find maximum value in array_random
array_random.max()
```

Out[33]:
```
49
```

Indices of min_max

In [34]:
```
# import relevant libraries
```

```
import numpy as np
```

In [35]:
```
# create a numpy array of 20 random integers from 1 to 50 & save in a variable
array_random = np.random.randint(1, 50, 20)
array_random
```

Out[35]:
```
array([45, 49,  5, 24, 39,  9, 45, 22, 36, 15, 46,  8, 25, 15, 36, 48, 12,
       20, 30,  9])
```

In [36]:
```
# find index of minimum value in array_random
array_random.argmin()
```

Out[36]:
```
2
```

In [37]:
```
# find index of maximum value in array_random
array_random.argmax()
```

Out[37]:
```
1
```

Shapes

In [38]:
```
# import relevant libraries
import numpy as np
```

In [39]:
```
array0 = np.array([])
```

In [40]:
```
# find the shape of array0
array0.shape
```

Out[40]:
```
(0,)
```

In [41]:
```
# find the shape of array0
array0.shape
```

Out[41]:
```
(0,)
```

In [42]:
```
array1 = np.array([1, 2, 3, 4, 5])
```

In [43]:
```
# find the shape of array1
array1.shape
```

Out[43]:
```
(5,)
```

In [44]:
```
array2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

In [45]:
```
# find the shape of array2
array2.shape
```

Out[45]:
```
(3, 3)
```

In [46]:
```
array_zeros = np.zeros((4, 5))
```

In [47]:
```
# find the shape of array_zeros
array_zeros.shape
```

Out[47]:
```
(4, 5)
```

In [48]:
```
array_ones = np.ones((3, 6))
```

In [49]:
```
# find the shape of array_ones
array_ones.shape
```

Out[49]:
```
(3, 6)
```

In [50]:
```
array_r0 = np.random.randint(1, 50, 25)
array_r0
```

Out[50]:
```
array([10, 44, 14, 24, 10, 38, 49,  1, 10, 14, 10, 33, 30, 48, 20, 19, 32,
       30,  3, 19, 26, 21,  2, 40,  8])
```

In [51]:
```
# find the shape of array_r0
array_r0.shape
```

Out[51]:
```
(25,)
```

In [52]:
```
# reshape array_r0
# to create a new multi-dimensional numpy array
# containing array_r0's items with 5 rows & 5 columns
array_r0.reshape((5, 5))
```

Out[52]:
```
array([[10, 44, 14, 24, 10],
       [38, 49,  1, 10, 14],
       [10, 33, 30, 48, 20],
       [19, 32, 30,  3, 19],
       [26, 21,  2, 40,  8]])
```

In [53]:
```
# note that array_r0 itself was not modified
array_r0
```

Out[53]:
```
array([10, 44, 14, 24, 10, 38, 49,  1, 10, 14, 10, 33, 30, 48, 20, 19, 32,
       30,  3, 19, 26, 21,  2, 40,  8])
```

In [54]:
```
array_r1 = np.random.randint(1, 50, 20)
array_r1
```

Out[54]:
```
array([17, 35, 39, 27,  9, 15, 19, 20, 43, 14, 46, 18, 17, 36, 41, 33,  5,
        2, 25, 11])
```

In [55]:
```
# find the shape of array_r1
array_r1.shape
```

Out[55]:
```
(20,)
```

In [56]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 10 rows & 2 columns
array_r1.reshape((10, 2))
```

Out[56]:
```
array([[17, 35],
       [39, 27],
       [ 9, 15],
       [19, 20],
       [43, 14],
       [46, 18],
       [17, 36],
       [41, 33],
       [ 5,  2],
       [25, 11]])
```

In [57]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 2 rows & 10 columns
array_r1.reshape((2, 10))
```

Out[57]:
```
array([[17, 35, 39, 27,  9, 15, 19, 20, 43, 14],
       [46, 18, 17, 36, 41, 33,  5,  2, 25, 11]])
```

In [58]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 4 rows & 5 columns
array_r1.reshape((4, 5))
```

Out[58]:
```
array([[17, 35, 39, 27,  9],
       [15, 19, 20, 43, 14],
       [46, 18, 17, 36, 41],
       [33,  5,  2, 25, 11]])
```

In [59]:
```
# reshape array_r1
# to create a new multi-dimensional numpy array containing array_r1's items
# with 5 rows & 4 columns
array_r1.reshape((5, 4))
```

Out[59]:
```
array([[17, 35, 39, 27],
       [ 9, 15, 19, 20],
       [43, 14, 46, 18],
       [17, 36, 41, 33],
       [ 5,  2, 25, 11]])
```

In [60]:
```
# note that array_r1 itself was not modified
array_r1
```

Out[60]:
```
array([17, 35, 39, 27,  9, 15, 19, 20, 43, 14, 46, 18, 17, 36, 41, 33,  5,
        2, 25, 11])
```

In [61]:
```
array_r2 = np.random.randint(1, 50, (4, 4))
array_r2
```

Out[61]:
```
array([[10, 23, 34, 27],
       [14, 15, 33,  8],
       [31, 46, 36, 46],
       [ 3,  8,  2, 23]])
```

In [62]:
```
# find the shape of array_r2
array_r2.shape
```

Out[62]:
```
(4, 4)
```

In [63]:
```
# reshape array_r2
# to create a new one-dimensional numpy array containing array_r2's items
# with length 16
array_r2.reshape(16)
```

Out[63]:
```
array([10, 23, 34, 27, 14, 15, 33,  8, 31, 46, 36, 46,  3,  8,  2, 23])
```

In [64]:
```
# note that array_r2 itself was not modified
array_r2
```

Out[64]:
```
array([[10, 23, 34, 27],
       [14, 15, 33,  8],
       [31, 46, 36, 46],
       [ 3,  8,  2, 23]])
```

Groups numpy

In [65]:
```
# initialize array0
# as a one-dimensional numpy array containing integers 0 to 4 including 0 and 4
array0 = np.arange(5)
array0
```

Out[65]:
```
array([0, 1, 2, 3, 4])
```

In [66]:
```
# select the item at index 1 from array0
array0[1]
```

Out[66]:
```
1
```

In [67]:
```
# select the items at indices 1 to 3 inclusive from array0
array0[1:4]
```

Out[67]:
```
array([1, 2, 3])
```

In [68]:
```
# select the items at indices 0 to 3 inclusive from array0
array0[:4]
```

Out[68]:
```
array([0, 1, 2, 3])
```

In [69]:
```
# select the items starting at index 3 until the end of array0
array0[3:]
```

Out[69]:
```
array([3, 4])
```

In [70]:
```
# initialize array1 as a three-dimensional numpy array
array1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
array1
```

Out[70]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [71]:
```
# select the item in row 0 column 0 from array1 using double bracket notation
array1[0][0]
```

Out[71]:
```
1
```

In [72]:
```
# select the item in row 0 column 0 from array1 using single bracket notation
array1[0, 0]
```

Out[72]:
```
1
```

In [73]:
```
# select the item in row 1 column 2 from array1 using double bracket notation
array1[1][2]
```

Out[73]:
```
6
```

In [74]:
```
# select the item in row 1 column 2 from array1 using single bracket notation
array1[1, 2]
```

Out[74]:
```
6
```

In [75]:
```
# select first two items from row 0 of array1
array1[:1, :2]
```

Out[75]:
```
array([[1, 2]])
```

In [76]:
```
# select first two items from every row until row 1 inclusive from array1
array1[:2, :2]
```

Out[76]:
```
array([[1, 2],
       [4, 5]])
```

In [77]:
```
# select first two items from each row of array1
array1[:, :2]
```

Out[77]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

In [78]:
```
# select row 0 from array1
array1[0]
```

Out[78]:
```
array([1, 2, 3])
```

In [79]:
```
# select everything before row 2 from array1
array1[:2]
```

Out[79]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [80]:
```
# initialize array2
# as a one-dimensional numpy array containing all even integers between 0 and 20 inclus
array2 = np.arange(0, 21, 2)
array2
```

Out[80]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [81]:
```
# select the items from array2 that are greater than or equal to 12
array2[array2 >= 12]
```

Out[81]:
```
array([12, 14, 16, 18, 20])
```

In [82]:
```
# select the items from array2 that are greater than 7 and less than 13
array2[(array2 > 7) & (array2 < 13)]
```

Out[82]:
```
array([ 8, 10, 12])
```

Arithmetic

In [83]:
```
# initialize arrayA as a one-dimensional numpy array
# containing the odd integers between 1 and 20 inclusive
arrayA = np.arange(1, 21, 2)
arrayA
```

Out[83]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

In [84]:
```
# initialize arrayB as a one-dimensional numpy array
# containing the integers 1 to 10 inclusive
arrayB = np.arange(1, 11)
arrayB
```

Out[84]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [85]:
```
# add each element of arrayA to each corresponding element of arrayB,
# creating a new numpy array
arrayA + arrayB
```

Out[85]:
```
array([ 2,  5,  8, 11, 14, 17, 20, 23, 26, 29])
```

In [86]:
```
# subtract each element of arrayB from each corresponding element from arrayA,
# creating a new numpy array
arrayA - arrayB
```

Out[86]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [87]:
```
# multiply each element of arrayA by each corresponding element of arrayB,
# creating a new numpy array
arrayA * arrayB
```

Out[87]:
```
array([  1,   6,  15,  28,  45,  66,  91, 120, 153, 190])
```

In [88]:
```
# divide each element of arrayA by each corresponding element of arrayB,
# creating a new numpy array
arrayA / arrayB
```

Out[88]:
```
array([1.        , 1.5       , 1.66666667, 1.75      , 1.8       ,
       1.83333333, 1.85714286, 1.875     , 1.88888889, 1.9       ])
```

In [89]:
```
# initialize arrayC as a multi-dimensional numpy array with 2 rows and 3 columns
arrayC = np.array([[1, 2, 3], [4, 5, 6]])
arrayC
```

Out[89]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [90]:
```
# initialize arrayD as another multi-dimensional numpy array with 2 rows and 3 columns
arrayD = np.array([[7, 8, 9], [10, 11, 12]])
arrayD
```

Out[90]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [91]:
```
# add each element of arrayC to each corresponding element of arrayD,
# creating a new numpy array
arrayC + arrayD
```

Out[91]:
```
array([[ 8, 10, 12],
       [14, 16, 18]])
```

In [92]:
```
# subtract each element of arrayD from each corresponding element from arrayC,
# creating a new numpy array
arrayC - arrayD
```

Out[92]:
```
array([[-6, -6, -6],
```

```
       [-6, -6, -6]])
```

In [93]:
```
# multiply each element of arrayC by each corresponding element of arrayD,
# creating a new numpy array
arrayC * arrayD
```

Out[93]:
```
array([[ 7, 16, 27],
       [40, 55, 72]])
```

In [94]:
```
# divide each element of arrayC by each corresponding element of arrayD,
# creating a new numpy array
arrayC / arrayD
```

Out[94]:
```
array([[0.14285714, 0.25      , 0.33333333],
       [0.4       , 0.45454545, 0.5       ]])
```

Scalar Numpy

In [96]:
```
# initialize arrayA as a one-dimensional numpy array
# containing the even integers between 2 and 20 inclusive
arrayA = np.arange(2, 21, 2)
arrayA
```

Out[96]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [97]:
```
# add 3 to each element of arrayA, creating a new numpy array
arrayA + 3
```

Out[97]:
```
array([ 5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

In [98]:
```
# subtract 4 from each element of arrayA, creating a new numpy array
arrayA - 4
```

Out[98]:
```
array([-2,  0,  2,  4,  6,  8, 10, 12, 14, 16])
```

In [99]:
```
# multiply each element of arrayA by 5, creating a new numpy array
arrayA * 5
```

Out[99]:
```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

In [100…
```
# divide each element of arrayA by 2, creating a new numpy array
arrayA / 2
```

Out[100…
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

In [101…
```
# initialize arrayB as a multi-dimensional numpy array with 2 rows and 3 columns
arrayB = np.array([[1, 2, 3], [4, 5, 6]])
arrayB
```

```
array([[1, 2, 3],
```

Out[101…
```
       [4, 5, 6]])
```

In [102…
```
# add 3 to each element of arrayB, creating a new numpy array
arrayB + 3
```

Out[102…
```
array([[4, 5, 6],
       [7, 8, 9]])
```

In [103…
```
# multiply each element of arrayB by 5, creating a new numpy array
arrayB * 5
```

Out[103…
```
array([[ 5, 10, 15],
       [20, 25, 30]])
```

In [104…
```
# divide each element of arrayB by 2, creating a new numpy array
arrayB / 2
```

Out[104…
```
array([[0.5, 1. , 1.5],
       [2. , 2.5, 3. ]])
```

statistical

In [105…
```
# initialize arrayA as a numpy array
# representing the scores of participants in a competition
scores = np.random.randint(50, 101, 200)
```

In [106…
```
# compute the median of scores
np.median(scores)
```

Out[106…
```
73.0
```

In [107…
```
# compute the mean of scores
np.mean(scores)
```

Out[107…
```
74.105
```

In [108…
```
# compute the variance of scores
np.var(scores)
```

Out[108…
```
217.233975
```

In [109…
```
# compute the standard deviation of scores
np.std(scores)
```

Out[109…
```
14.738859352066562
```

Other functions

In [110…
```
# initialize arrayA as a one-dimensional numpy array
# containing the integers 0 to 5 inclusive
arrayA = np.arange(6)
arrayA
```

Out[110…
```
array([0, 1, 2, 3, 4, 5])
```

In [111…
```
# compute the square of each item in arrayA,
# creating a new numpy array
np.square(arrayA)
```

Out[111…
```
array([ 0,  1,  4,  9, 16, 25], dtype=int32)
```

In [112…
```
# initialize arrayB as the following one-dimensional numpy array
arrayB = np.array([36, 49, 64, 81, 100, 121, 144, 169, 196, 225])
```

In [113…
```
# compute the square root of each item in arrayB,
# creating a new numpy array
np.sqrt(arrayB)
```

Out[113…
```
array([ 6.,  7.,  8.,  9., 10., 11., 12., 13., 14., 15.])
```

In [114…
```
# compute the exponential of each item in arrayA,
# creating a new numpy array
np.exp(arrayA)
```

Out[114…
```
array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,
        54.59815003, 148.4131591 ])
```

In [115…
```
# initialize arrayC as a one-dimensional numpy array
# containing the integers 1 to 11 inclusive
arrayC = np.arange(1, 11)
arrayC
```

Out[115…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [116…
```
# compute the natural logarithm of each item in arrayC,
# creating a new numpy array
np.log(arrayC)
```

Out[116…
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

In [117…
```
# initialize arrayD as a one-dimensional numpy array
# representing some angles in radians
arrayD = np.array([0, np.pi/6, np.pi/4, np.pi/3, np.pi/2, 2*np.pi/3, 3*np.pi/4, 5*np.pi
```

In [118…
```
# compute the sine of each item in arrayD,
# creating a new numpy array
np.sin(arrayD)
```

Out[118…
```
array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ,
       0.8660254 , 0.70710678, 0.5       , 1.        ])
```

In [119…
```
# initialize arrayE as a one-dimensional numpy array containing the integers -10 to 10
arrayE = np.arange(-10, 11)
arrayE
```

Out[119…
```
array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,
         3,   4,   5,   6,   7,   8,   9,  10])
```

In [120…
```
# compute the absolute value of each item in arrayE,
# creating a new numpy array
np.abs(arrayE)
```

Out[120…
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  2,  3,  4,  5,  6,
        7,  8,  9, 10])
```

In [121…
```
# initialize arrayF as the following one-dimensional numpy array
arrayF = np.arange(21)
arrayF
```

Out[121…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20])
```

In [122…
```
# compute the sum of all the items in arrayF
np.sum(arrayF)
```

Out[122…
```
210
```

In [123…
```
# initialize arrayG as the following multi-dimensional numpy array
arrayG = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arrayG
```

Out[123…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [124…
```
# compute the sum of all the items in arrayG
np.sum(arrayG)
```

Out[124…
```
45
```

In [125…
```
# compute the sum of the items in each column of arrayG,
# creating a new numpy array containing the sum for each column
np.sum(arrayG, axis=0)
```

Out[125…
```
array([12, 15, 18])
```

In [126…
```
# compute the sum of the items in each row of arrayG,
# creating a new numpy array containing the sum for each row
np.sum(arrayG, axis=1)
```

Out[126… array([6, 15, 24])

Chapter 4

Linear algebra

In [127…
```
# import relevant libraries and modules
import numpy as np
from scipy import linalg
```

In [128…
```
# create a numpy array representing a 2 by 2 matrix & save in variable
matrix2by2 = np.array([[1, 2],
                       [3, 4]])
matrix2by2
```

Out[128…
```
array([[1, 2],
       [3, 4]])
```

In [129…
```
# compute determinant of matrix2by2
linalg.det(matrix2by2)
```

Out[129… -2.0

In [130…
```
# compute inverse of matrix2by2 & save in variable
inv_matrix2by2 = linalg.inv(matrix2by2)
inv_matrix2by2
```

Out[130…
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

statistical

In [131…
```
# import relevant libraries and modules
import numpy as np
from scipy import stats
```

In [132…
```
# declare B to be a binomial discrete random variable with parameters 10 and 0.4
B = stats.binom(10, 0.4)
```

In [133…
```
# compute value of its probability mass function at 2
B.pmf(2)
```

Out[133… 0.12093235199999991

In [134…
```
# compute value of its cumulative density function at 3
B.cdf(3)
```

Out[134… 0.3822806015999999

In [135…
```
# declare P to be a poission discrete random variable with parameter 2
P = stats.poisson(2)
```

In [136…
```
# compute value of its probability mass function at 2
P.pmf(2)
```

Out[136… 0.2706705664732254

In [137…
```
# compute value of its cumulative density function at 4
P.cdf(4)
```

Out[137… 0.9473469826562889

In [138…
```
# declare G to be a geometric discrete random variable with parameter 0.25
G = stats.geom(0.25)
```

In [139…
```
# compute value of its probability mass function at 3
G.pmf(3)
```

Out[139… 0.140625

In [140…
```
# compute value of its cumulative density function at 4
G.cdf(4)
```

Out[140… 0.68359375

In [141…
```
# declare N to be a normal continuous random variable with parameters 0 and 1
N = stats.norm(0, 1)
```

In [142…
```
# compute its probability density at 0.1
N.pdf(0.1)
```

Out[142… 0.3969525474770118

In [143…
```
# compute its cumulative density at -0.2
N.cdf(-0.2)
```

Out[143… 0.42074029056089696

In [144…
```
# declare E to be an exponential continuous random variable with parameter 4
E = stats.expon(4)
```

In [145…
```
# declare X to be a beta continuous random variable with parameters 1 and 3
```

```
X = stats.beta(1, 3)
```

In [146…
```
# compute its probability density at 0.6
X.pdf(0.6)
```

Out[146…
```
0.4799999999999999
```

In [147…
```
# compute its cumulative density at 0.5
X.cdf(0.5)
```

Out[147…
```
0.875
```

In [148…
```
# create a variable named scores containing a numpy array
# that represents the scores of participants in a competition
scores = np.random.randint(50, 101, 300)
```

In [149…
```
# compute the 50th percentile of scores
stats.scoreatpercentile(scores, 50)
```

Out[149…
```
73.0
```

In [150…
```
# compute the 90th percentile of scores
stats.scoreatpercentile(scores, 90)
```

Out[150…
```
95.10000000000002
```

In [151…
```
# create a variable named round1_scores containing a numpy array
# that represents the scores of participants from school A in competition
schoolA_scores = np.random.normal(70, 15, size=100)
```

In [152…
```
# create a variable named round2_scores containing a numpy array
# that represents the scores of participants from school B in competition
schoolB_scores = np.random.normal(80, 15, size=10)
```

In [153…
```
# conduct the T-test for the means of schoolA_scores and schoolB_scores
stats.ttest_ind(schoolA_scores, schoolB_scores)
```

Out[153…
```
Ttest_indResult(statistic=-2.6210423934797293, pvalue=0.010029730310732704)
```

Chapter 5

In [154…
```
# create a pandas series containing 8 random integers from -10 to 10 inclusive
pd.Series(np.random.randint(-10, 11, size=8))
```

Out[154…
```
0    -3
1     4
```

```
2    10
3     8
4    -1
5    -9
6    -8
7     2
dtype: int32
```

In [155…
```
# create a pandas series containing 8 random integers from -10 to 10 inclusive
# with index being a through h
pd.Series(np.random.randint(-10, 11, size=8),
          index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
```

Out[155…
```
a    -1
b    -5
c     0
d     9
e     3
f     3
g     7
h   -10
dtype: int32
```

In [156…
```
# create a pandas series from a Python dictionary
pd.Series({'Mon': True, 'Tue': False, 'Wed': True, 'Thu': False,
           'Fri': True, 'Sat': False, 'Sun': True})
```

Out[156…
```
Mon     True
Tue    False
Wed     True
Thu    False
Fri     True
Sat    False
Sun     True
dtype: bool
```

In [157…
```
# create a pandas series from a scalar value
# to represent the maximum number of points students can earn
# in each of 10 exams for a particular course
pd.Series(150, index=np.arange(1, 11))
```

Out[157…
```
1     150
2     150
3     150
4     150
5     150
6     150
7     150
8     150
9     150
10    150
dtype: int64
```

In [158…
```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [159…

```
# display grades
grades
```

Out[159…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	NaN
31	4	2	80.0

	exam	student_id	grade
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [160…

```
# display first few rows of grades
grades.head()
```

Out[160…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [161…

```
# create a Python dictionary of Series & save in a variable named points
points = {'player1': pd.Series([15, 10, 20, 25],
                               index=['game1', 'game2', 'game3', 'game4']),
          'player2': pd.Series([10, 15, 23, 27],
                               index=['game1', 'game2', 'game3', 'game4'])}

# create a pandas dataframe from points
pd.DataFrame(points)
```

Out[161...

	player1	player2
game1	15	10
game2	10	15
game3	20	23
game4	25	27

In [162...

```
# create a Python dictionary of lists & save in a variable named sales
sales = {'foodTruck1': [216,275,203,210,315,402,380],
         'foodTruck2': [374,90,95,115,130,150,140]}

# create a pandas dataframe from sales with index being day1 through day7
pd.DataFrame(sales, index=['day1', 'day2', 'day3', 'day4', 'day5', 'day6', 'day7'])
```

Out[162...

	foodTruck1	foodTruck2
day1	216	374
day2	275	90
day3	203	95
day4	210	115
day5	315	130
day6	402	150
day7	380	140

In [163...

```
# create a multi-dimensional numpy array & save in a variable named passengers
passengers = np.array([[20, 40, 60, 80], [15, 30, 45, 60], [10, 20, 30, 40]])

# create a pandas DataFrame from passengers
# with index being plane1 through plane3
# and columns being infants, children, adults, seniors
pd.DataFrame(passengers,
             index=['plane1', 'plane2', 'plane3'],
             columns=['infants', 'children', 'adults', 'seniors'])
```

Out[163...

	infants	children	adults	seniors
plane1	20	40	60	80
plane2	15	30	45	60
plane3	10	20	30	40

Select DataFrame

In [164...

```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [165…
```
# display first few rows of grades
grades.head()
```

Out[165…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [166…
```
# select exam column and grade column from grades dataframe
# result will be a pandas dataframe containing just those two columns
grades.loc[:, ['exam', 'grade']]
```

Out[166…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	NaN
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	NaN
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0

	exam	grade
22	3	87.0
23	3	NaN
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0
30	4	NaN
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	NaN
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [167…

```
# select grade column from grades dataframe
# result will be a pandas dataframe containing just that column
grades.loc[:, ['grade']]
```

Out[167…

	grade
0	86.0

	grade
1	65.0
2	70.0
3	98.0
4	89.0
5	NaN
6	75.0
7	56.0
8	90.0
9	81.0
10	79.0
11	60.0
12	78.0
13	75.0
14	NaN
15	80.0
16	87.0
17	82.0
18	95.0
19	96.0
20	78.0
21	80.0
22	87.0
23	NaN
24	89.0
25	90.0
26	100.0
27	72.0
28	73.0
29	75.0
30	NaN
31	80.0
32	81.0
33	82.0
34	83.0

	grade
35	84.0
36	85.0
37	86.0
38	87.0
39	88.0
40	90.0
41	NaN
42	91.0
43	92.0
44	93.0
45	94.0
46	95.0
47	96.0
48	97.0
49	98.0

In [168…
```
# select row 0 from grades dataframe such that result is a pandas series
grades.iloc[0]
```

Out[168…
```
exam          1.0
student_id    1.0
grade        86.0
Name: 0, dtype: float64
```

In [169…
```
# select row 0 from grades dataframe such that result is a pandas dataframe
grades.iloc[[0], :]
```

Out[169…

	exam	student_id	grade
0	1	1	86.0

In [170…
```
# select row 0 and row 4 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[0, 10], :]
```

Out[170…

	exam	student_id	grade
0	1	1	86.0
10	2	1	79.0

In [171…
```
# select item at row 4 column 2 from grades dataframe
```

```
grades.iloc[4, 2]
```

Out[171…

```
89.0
```

In [172…

```
# select column 0 and column 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[:, [0, 2]]
```

Out[172…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	NaN
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	NaN
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	NaN
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0

	exam	grade
28	3	73.0
29	3	75.0
30	4	NaN
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	NaN
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [173…

```
# select row 0 and row 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[0, 2], :]
```

Out[173…

	exam	student_id	grade
0	1	1	86.0
2	1	3	70.0

In [174…

```
# select rows 35 and 45 and columns 0 and 2 from grades dataframe
# result will be a pandas dataframe
grades.iloc[[35, 45], [0, 2]]
```

Out[174…

	exam	grade

	exam	grade
35	4	84.0
45	5	94.0

In [175…

```
# select every row from grades dataframe for which entry in grade column is atleast 70.
grades[grades['grade'] >= 70.0]
```

Out[175…

	exam	student_id	grade
0	1	1	86.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
6	1	7	75.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
12	2	3	78.0
13	2	4	75.0
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0

	exam	student_id	grade
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [176…

```
# select all data representing students' grades on exam no. 5 that were atleast 70.0%
# in other words, select every row from grades dataframe for which
# entry in exam column is 5 and entry in grade column is atleast 70.0
grades[(grades['exam'] == 5) & (grades['grade'] >= 70.0)]
```

Out[176…

	exam	student_id	grade
40	5	1	90.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

Modifying pandas objects

In [177…

```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [178…

```
# display grades
grades
```

Out[178…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	NaN

	exam	student_id	grade
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [179…

```
# fill missing values in grade column with zeros
grades['grade'] = grades['grade'].fillna(0)
```

In [180…

```
# display grades
grades
```

Out[180…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	0.0
6	1	7	75.0
7	1	8	56.0

	exam	student_id	grade
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	0.0
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	0.0
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0
29	3	10	75.0
30	4	1	0.0
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0

	exam	student_id	grade
41	5	2	0.0
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [181…

```
# drop student_id column from grades
grades = grades.drop(columns=['student_id'])
```

In [182…

```
# display grades
grades
```

Out[182…

	exam	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	0.0
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	0.0
15	2	80.0
16	2	87.0
17	2	82.0

	exam	grade
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	0.0
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0
30	4	0.0
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	0.0
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

In [183…

```
# rename exam column --- change that column's label from 'exam' to 'exam #'
```

```
grades = grades.rename(columns={'exam': 'exam #'})
```

In [184…

```
# display grades
grades
```

Out[184…

	exam #	grade
0	1	86.0
1	1	65.0
2	1	70.0
3	1	98.0
4	1	89.0
5	1	0.0
6	1	75.0
7	1	56.0
8	1	90.0
9	1	81.0
10	2	79.0
11	2	60.0
12	2	78.0
13	2	75.0
14	2	0.0
15	2	80.0
16	2	87.0
17	2	82.0
18	2	95.0
19	2	96.0
20	3	78.0
21	3	80.0
22	3	87.0
23	3	0.0
24	3	89.0
25	3	90.0
26	3	100.0
27	3	72.0
28	3	73.0
29	3	75.0

	exam #	grade
30	4	0.0
31	4	80.0
32	4	81.0
33	4	82.0
34	4	83.0
35	4	84.0
36	4	85.0
37	4	86.0
38	4	87.0
39	4	88.0
40	5	90.0
41	5	0.0
42	5	91.0
43	5	92.0
44	5	93.0
45	5	94.0
46	5	95.0
47	5	96.0
48	5	97.0
49	5	98.0

Combining Data

In [185…

```
# create a pandas dataframe containing grades on exam #1
# for students with student id numbers 1 through 5
exam1_grades = pd.DataFrame({'SID': [1, 2, 3, 4, 5],
                             'exam1': [86.0, 65.0, 70.0, 98.0, 89.0]})

# create a pandas dataframe containing grades on exam #2
# for students with student id numbers 1 through 5
exam2_grades = pd.DataFrame({'SID': [1, 2, 3, 4, 5],
                             'exam2': [80.0, 87.0, 82.0, 95.0, 96.0]})
```

In [186…

```
# create a pandas dataframe containing grades on exam #1 and exam #2
# for students with id numbers 1 through 7
sid_1_to_7 = pd.DataFrame({'SID': [1, 2, 3, 4, 5, 6, 7],
              'exam1': [86.0, 65.0, 70.0, 98.0, 89.0, 75.0, 56.0],
              'exam2': [80.0, 87.0, 82.0, 95.0, 96.0, 78.0, 80.0]})

# create a pandas dataframe containing grades on exam #1 and exam #2
```

```
# for students with id numbers 8 through 10
sid_8_to_10 = pd.DataFrame({'SID': [8, 9, 10],
                            'exam1': [90.0, 81.0, 0.0],
                            'exam2': [87.0, 82.0, 95.0]})
```

In [187…

```
# create a pandas dataframe containing grades on exam #1 and exam #2
# for students 1 through 5
exams1and2 = pd.DataFrame({'exam1': [86.0, 65.0, 70.0, 98.0, 89.0],
              'exam2': [80.0, 87.0, 82.0, 95.0, 96.0]},
              index=['student1', 'student2', 'student3', 'student4', 'student5'])

# create a pandas dataframe containing grades on exam #3 for students 1 through 5
exam3 = pd.DataFrame({'exam3': [78.0, 80.0, 87.0, 89.0, 89.0]},
              index=['student1', 'student2', 'student3', 'student4', 'student5'])
```

In [188…

```
# display exam1_grades
exam1_grades
```

Out[188…

	SID	exam1
0	1	86.0
1	2	65.0
2	3	70.0
3	4	98.0
4	5	89.0

In [189…

```
# display exam2_grades
exam2_grades
```

Out[189…

	SID	exam2
0	1	80.0
1	2	87.0
2	3	82.0
3	4	95.0
4	5	96.0

In [190…

```
# merge exam1_grades and exam2_grades on 'SID'
# result will be a new pandas dataframe
pd.merge(exam1_grades, exam2_grades, on='SID')
```

Out[190…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0

	SID	exam1	exam2
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0

In [191…
```
# display sid_1_to_7
sid_1_to_7
```

Out[191…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0
5	6	75.0	78.0
6	7	56.0	80.0

In [192…
```
# display sid_8_to_10
sid_8_to_10
```

Out[192…

	SID	exam1	exam2
0	8	90.0	87.0
1	9	81.0	82.0
2	10	0.0	95.0

In [193…
```
# concatenate sid_1_to_7 and sid_8_to_10 along the rows (along axis 0)
# result will be a new pandas dataframe
pd.concat([sid_1_to_7, sid_8_to_10], axis=0)
```

Out[193…

	SID	exam1	exam2
0	1	86.0	80.0
1	2	65.0	87.0
2	3	70.0	82.0
3	4	98.0	95.0
4	5	89.0	96.0
5	6	75.0	78.0
6	7	56.0	80.0

	SID	exam1	exam2
0	8	90.0	87.0
1	9	81.0	82.0
2	10	0.0	95.0

In [194…
```
# display exams1and2
exams1and2
```

Out[194…

	exam1	exam2
student1	86.0	80.0
student2	65.0	87.0
student3	70.0	82.0
student4	98.0	95.0
student5	89.0	96.0

In [195…
```
# display exam3
exam3
```

Out[195…

	exam3
student1	78.0
student2	80.0
student3	87.0
student4	89.0
student5	89.0

In [196…
```
# concatenate exams1and2 and exam3 along the columns (along axis 1)
# result will be a new pandas dataframe
pd.concat([exams1and2, exam3], axis=1)
```

Out[196…

	exam1	exam2	exam3
student1	86.0	80.0	78.0
student2	65.0	87.0	80.0
student3	70.0	82.0	87.0
student4	98.0	95.0	89.0
student5	89.0	96.0	89.0

Grouping Data Pandas

In [197…
```
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [198…
```
# display grades
grades
```

Out[198…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0
5	1	6	NaN
6	1	7	75.0
7	1	8	56.0
8	1	9	90.0
9	1	10	81.0
10	2	1	79.0
11	2	2	60.0
12	2	3	78.0
13	2	4	75.0
14	2	5	NaN
15	2	6	80.0
16	2	7	87.0
17	2	8	82.0
18	2	9	95.0
19	2	10	96.0
20	3	1	78.0
21	3	2	80.0
22	3	3	87.0
23	3	4	NaN
24	3	5	89.0
25	3	6	90.0
26	3	7	100.0
27	3	8	72.0
28	3	9	73.0

	exam	student_id	grade
29	3	10	75.0
30	4	1	NaN
31	4	2	80.0
32	4	3	81.0
33	4	4	82.0
34	4	5	83.0
35	4	6	84.0
36	4	7	85.0
37	4	8	86.0
38	4	9	87.0
39	4	10	88.0
40	5	1	90.0
41	5	2	NaN
42	5	3	91.0
43	5	4	92.0
44	5	5	93.0
45	5	6	94.0
46	5	7	95.0
47	5	8	96.0
48	5	9	97.0
49	5	10	98.0

In [199…

```
# drop exam column from grades, group by student_id,
# and compute mean grade for each student_id
# result will be a new pandas dataframe
grades.drop(columns='exam').groupby('student_id').mean()
```

Out[199…

	grade
student_id	
1	83.25
2	71.25
3	81.40
4	86.75
5	88.50
6	87.00

student_id	grade
7	88.40
8	78.40
9	88.40
10	87.60

In [200…
```
# drop student_id column from grades, group by exam,
# and compute mean grade for each exam
# result will be a new pandas dataframe
grades.drop(columns='student_id').groupby('exam').mean()
```

Out[200…

exam	grade
1	78.888889
2	81.333333
3	82.666667
4	84.000000
5	94.000000

Chapter 6

Line Plots

In [205…
```
# import relevant libraries and modules
import pandas as pd
import matplotlib.pyplot as plt
```

In [206…
```
# create a list of days
days = [1,2,3,4,5,6,7]

# create a list of temperatures (in fahrenheit)
temps = [72,73,77,81,79,72,68]

# create a pandas dataframe
# containing the temperatues (in fahrenheit) for 7 consecutive days in Santa Barbara, C
temp_sb = pd.DataFrame(data={'day':days,'temperature':temps})
```

In [207…
```
# create a list of days
days = [1,2,3,4,5,6,7]

# create a list of temperatures (in fahrenheit)
temps = [93,91,91,91,91,88,90]
```

```
# create a pandas dataframe
# containing the temperatues (in fahrenheit) for 7 consecutive days in Memphis, TN
temp_mem = pd.DataFrame(data={'day':days,'temperature':temps})
```

In [208…

```
# display temp_sb
temp_sb
```

Out[208…

	day	temperature
0	1	72
1	2	73
2	3	77
3	4	81
4	5	79
5	6	72
6	7	68

In [209…

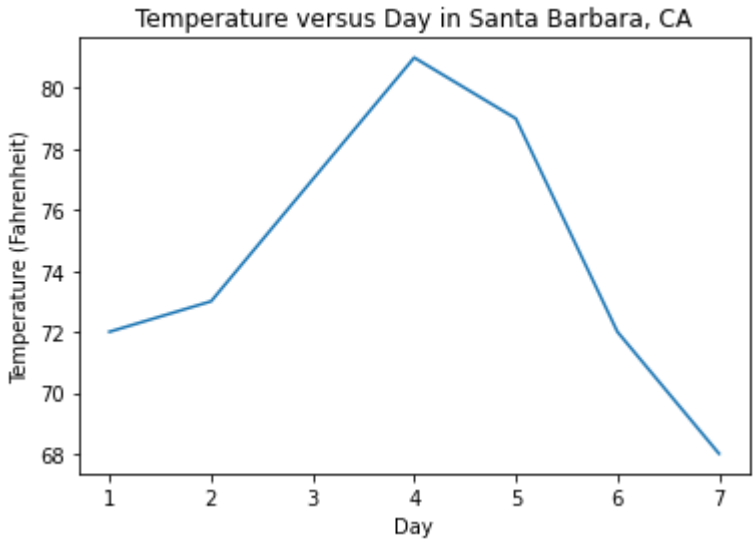
```
# display temp_mem
temp_mem
```

Out[209…

	day	temperature
0	1	93
1	2	91
2	3	91
3	4	91
4	5	91
5	6	88
6	7	90

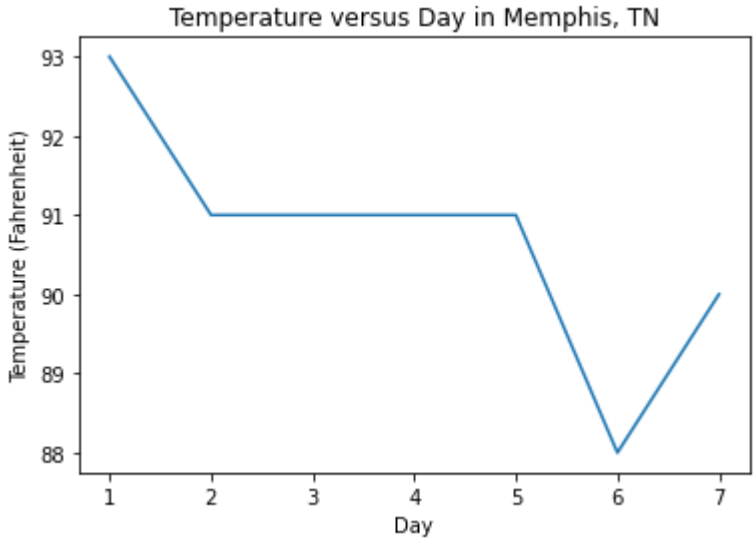
In [210…

```
# create a line plot of Santa Barbara data:
# plot temperature versus day in Santa Barbara
plt.plot(temp_sb['day'], temp_sb['temperature'])
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.title('Temperature versus Day in Santa Barbara, CA')
plt.show()
```



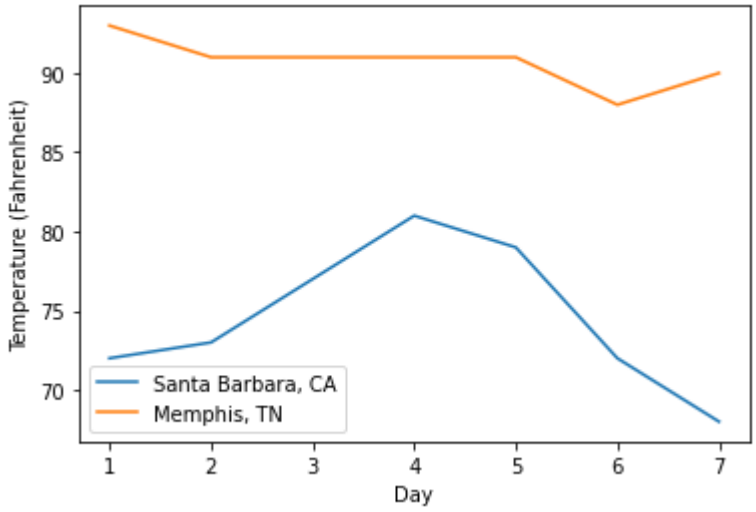
In [211…

```
# create a line plot of Memphis data:
# plot temperature versus day in Memphis
plt.plot(temp_mem['day'], temp_mem['temperature'])
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.title('Temperature versus Day in Memphis, TN')
plt.show()
```



In [212…

```
# plot santa barbara data and memphis data on the same line plot with a legend
plt.plot(temp_sb['day'], temp_sb['temperature'], label='Santa Barbara, CA')
plt.plot(temp_mem['day'], temp_mem['temperature'], label='Memphis, TN')
plt.xlabel('Day')
plt.ylabel('Temperature (Fahrenheit)')
plt.legend()
plt.show()
```



Scatter Plot

In [213…
```
# import relevant libraries and modules
import pandas as pd
import matplotlib.pyplot as plt
```

In [214…
```
# create a list of temperatures (in fahrenheit)
temp = [78, 89, 73, 75, 90, 99, 101,100,50,68,81, 85, 86, 70]

# create a list of sales (in $)
sale = [216,275,203,210,315,402,380,374,90,95,115,130,150,140]

# create a pandas dataframe containing the the sales
# (in dollars) of an ice cream truck for 14 days
# and the (temperature) in fahrenheit for each day
sales = pd.DataFrame(data={'temp':temp,'ice cream truck sale':sale})
```

In [215…
```
# display sales
sales
```

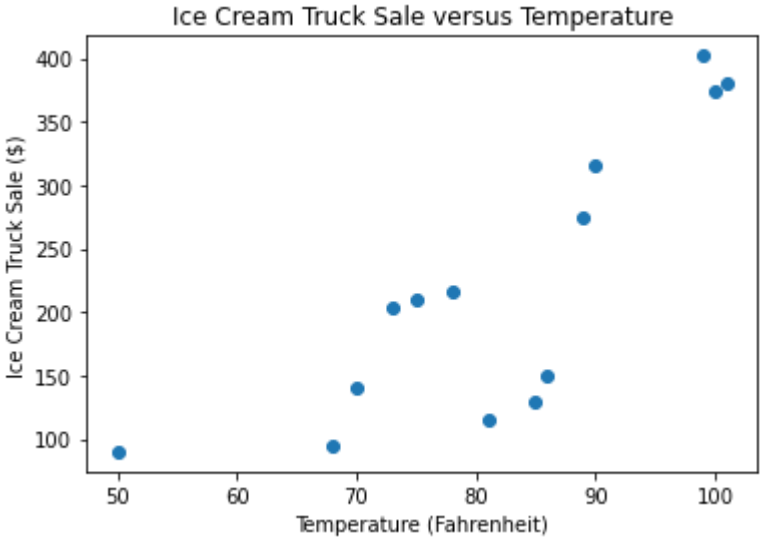
Out[215…

	temp	ice cream truck sale
0	78	216
1	89	275
2	73	203
3	75	210
4	90	315
5	99	402
6	101	380
7	100	374
8	50	90

	temp	ice cream truck sale
9	68	95
10	81	115
11	85	130
12	86	150
13	70	140

In [204…

```
# create a scatter plot --- plot ice cream truck sale versus temp
plt.scatter(sales['temp'], sales['ice cream truck sale'])
plt.xlabel('Temperature (Fahrenheit)')
plt.ylabel('Ice Cream Truck Sale ($)')
plt.title('Ice Cream Truck Sale versus Temperature')
plt.show()
```



Bar Plots

In [216…

```
# create a pandas dataframe containing the means for five exams taken by a set of stude
# save in a variable named exam_means
exam_means = pd.DataFrame(data={'exam':[1,2,3,4,5],
                                'mean':[73.2,71.5,79.0,62.0,84.6]})
```

In [217…

```
# display exam_means
exam_means
```

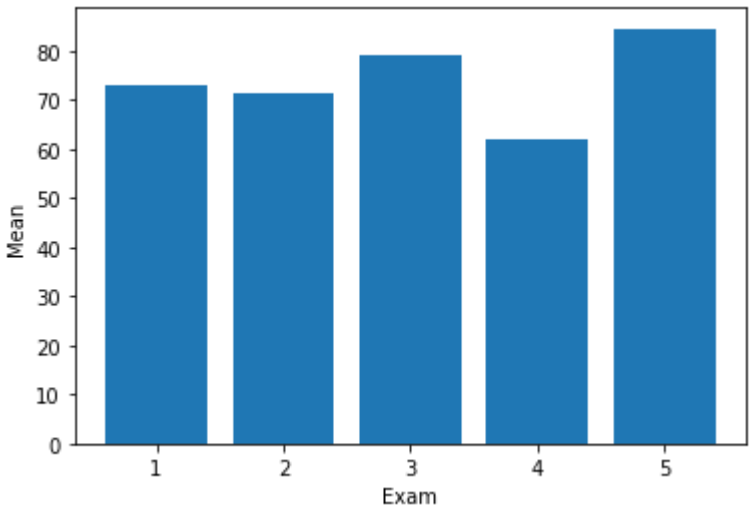
Out[217…

	exam	mean
0	1	73.2
1	2	71.5
2	3	79.0
3	4	62.0

	exam	mean
4	5	84.6

In [218…

```
# create a bar plot --- plot mean versus exam
plt.bar(exam_means['exam'], exam_means['mean'])
plt.xlabel('Exam')
plt.ylabel('Mean')
plt.show()
```



Pie Chart

In [219…

```
# create a list of categories
categ = ['Homework', 'Labs', 'Quizzes', 'Midterm', 'Final']

# create a list of weights (in %)
weights = [15, 15, 15, 22, 33]

# create a pandas dataframe containing the weighted components
# used to give students their grades in a particular course
breakdown = pd.DataFrame(data={'category':categ,'weight':weights})
```

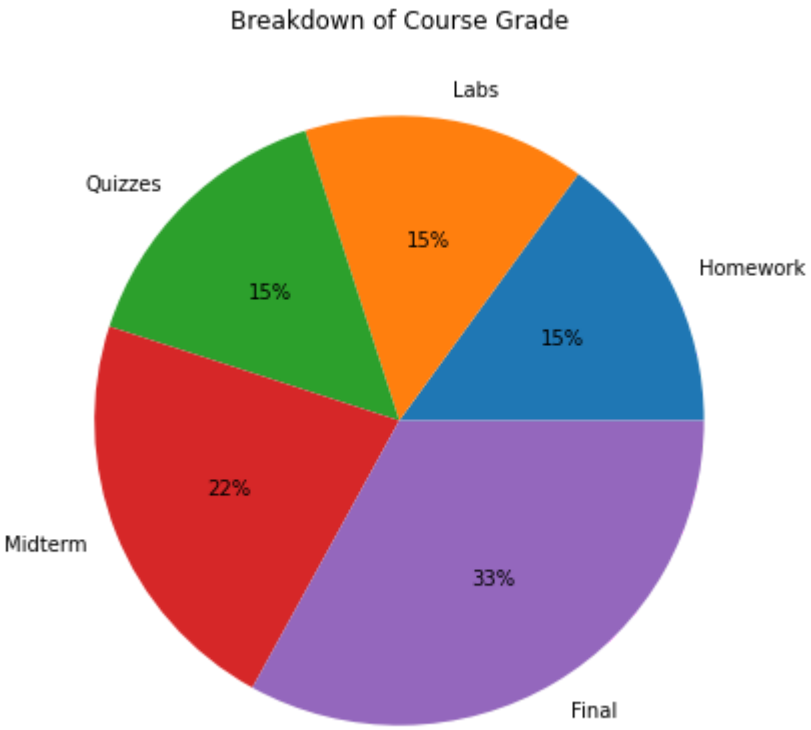
In [220…

```
# display breakdown
breakdown
```

Out[220…

	category	weight
0	Homework	15
1	Labs	15
2	Quizzes	15
3	Midterm	22
4	Final	33

In [221…
```
# create a pie chart --- illustrate the breakdown of students' course
# grades in a particular course
plt.figure(figsize=(7,7))
plt.pie(breakdown['weight'], labels=breakdown['category'],
        autopct='%1.0f%%')
plt.title('Breakdown of Course Grade')
plt.show()
```



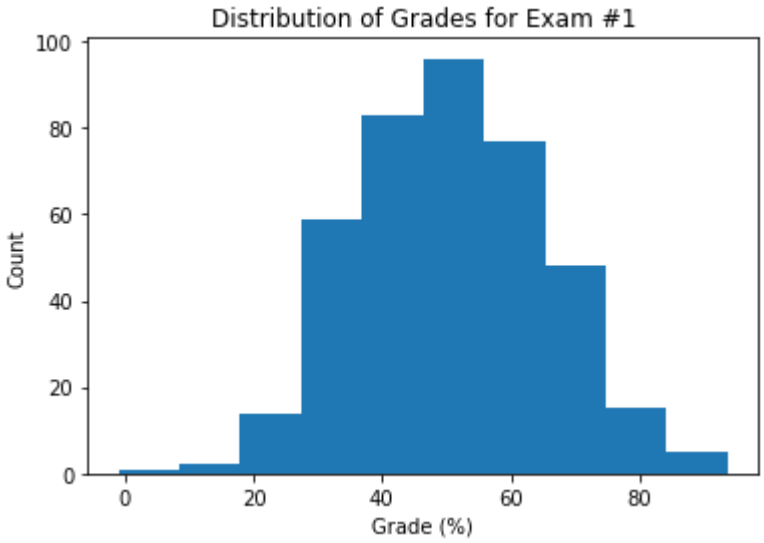
Histogram

In [222…
```
# create a pandas series containing students' grades on Exam #1 in a particular course
exam1_grades = pd.Series(np.random.normal(50, 15, 400))
```

In [223…
```
# display first few values in exam1_grades
exam1_grades.head()
```

Out[223…
```
0    56.979267
1    48.140717
2    46.521488
3    30.051874
4    53.095763
dtype: float64
```

In [224…
```
# create a histogram --- visualize exam1_grades
plt.hist(exam1_grades)
plt.xlabel('Grade (%)')
plt.ylabel('Count')
plt.title('Distribution of Grades for Exam #1')
plt.show()
```



Subplots

In [225…
```
# create a pandas series containing students' grades on Exam #1 in a particular course
exam1_grades = pd.Series(np.random.normal(50, 15, 400))
```

In [226…
```
# create a pandas series containing students' grades on Exam #2 in the same course
exam2_grades = pd.Series(np.random.normal(60, 10, 400))
```

In [227…
```
# display first few values in exam1_grades
exam1_grades.head()
```

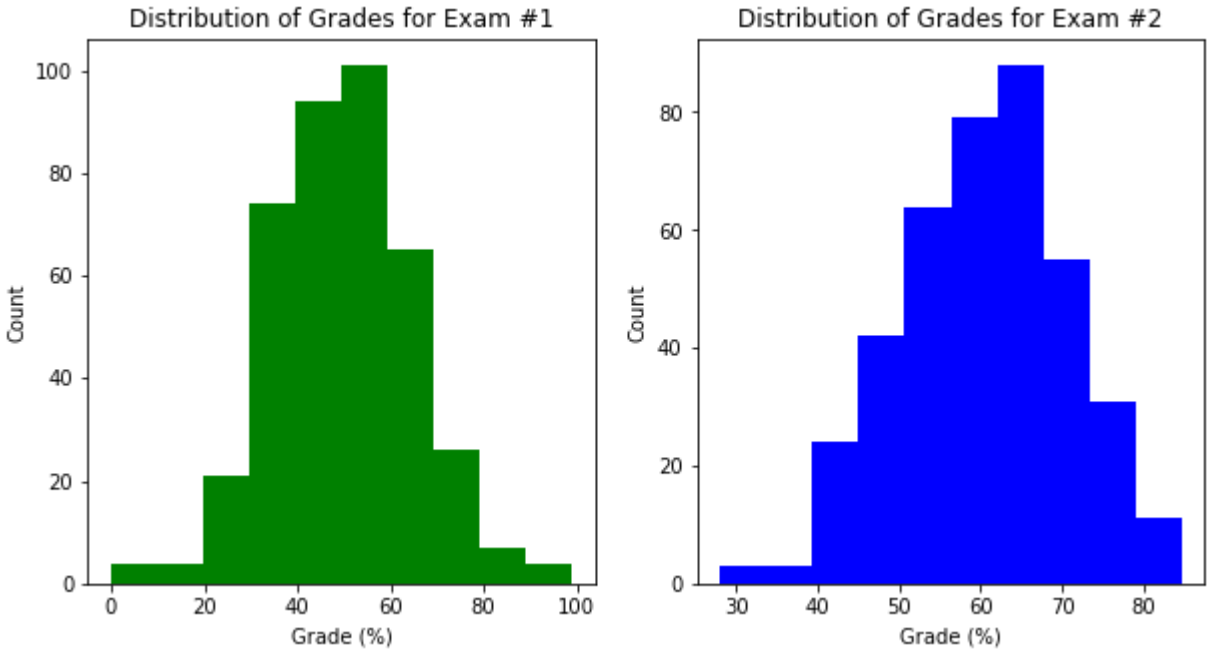
Out[227…
```
0    42.144654
1    58.725325
2    49.573887
3    53.166365
4    53.719261
dtype: float64
```

In [228…
```
# display first few values in exam2_grades
exam2_grades.head()
```

Out[228…
```
0    55.975289
1    51.774943
2    81.480904
3    62.079183
4    46.547136
dtype: float64
```

In [229…
```
# create 2 horizontally stacked subplots:
# display histogram for exam1_grades and histogram for exam2_grades side by side

figure, axes = plt.subplots(1, 2, figsize=(10,5))

axes[0].hist(exam1_grades, color='green')
axes[0].set_title('Distribution of Grades for Exam #1')
axes[1].hist(exam2_grades, color='blue')
```

```
axes[1].set_title('Distribution of Grades for Exam #2')

for a in axes.flat:
    a.set(xlabel='Grade (%)', ylabel='Count')

plt.show()
```



Chapter 7

BoxPlots

In [233…

```
# import relevant libraries
import pandas as pd
import seaborn as sns
# read grades.csv into a pandas dataframe & save the dataframe in a variable
grades = pd.read_csv('grades.csv')
```

In [234…

```
# display first few rows of grades
grades.head()
```

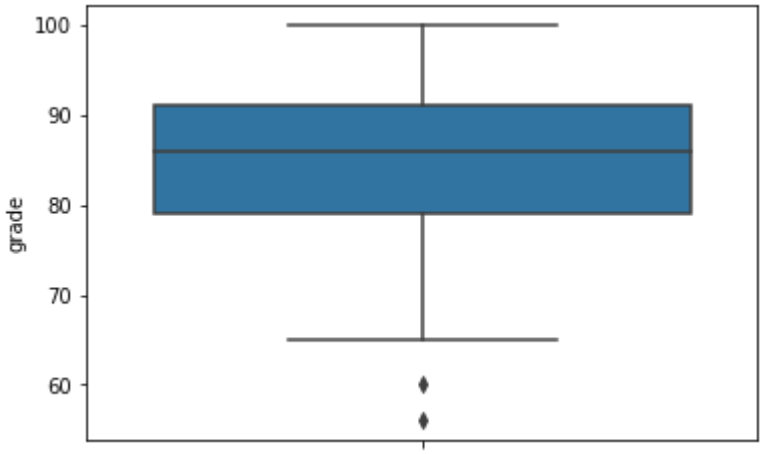
Out[234…

	exam	student_id	grade
0	1	1	86.0
1	1	2	65.0
2	1	3	70.0
3	1	4	98.0
4	1	5	89.0

In [235…

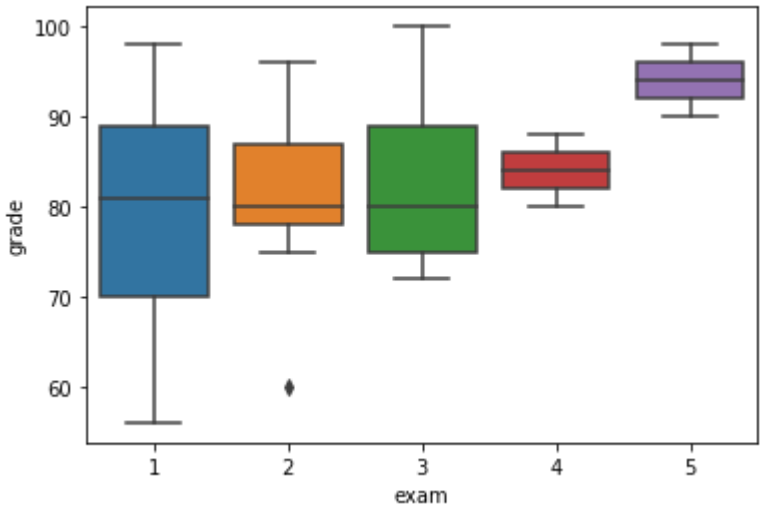
```
# create a vertical box plot of the data in the grade column of the grades dataset
```

```
sns.boxplot(y='grade', data=grades);
```



In [236…

```
# create a vertical boxplot of the data in the grade column of the grades dataset
# grouped by exam
sns.boxplot(x='exam', y='grade', data=grades);
```



Kernel Density

In [237…

```
# create a pandas series representing the scores of 400 students
# on a particular exam
scores = pd.Series(np.random.normal(70, 15, size=400))
```

In [238…

```
# initialize mean as a list containing two means,
# corresponding to the bivariate distribution
mean = [60, 70]

# initialize covariance_matrix as the covariance matrix
# of the bivariate distribution
covariance_matrix = [(1, .5), (.5, 1)]

# initialize data as a two-dimensional numpy array containing
# random samples drawn from
# the specified bivariate normal distribution
```

```
data = np.random.multivariate_normal(mean,
                                     covariance_matrix, size=400)

# initialize midterms as a pandas dataframe
# containing data as column values
# and midterm1, midterm2 as column labels
midterms = pd.DataFrame(data, columns=["midterm1", "midterm2"])
```

In [239…

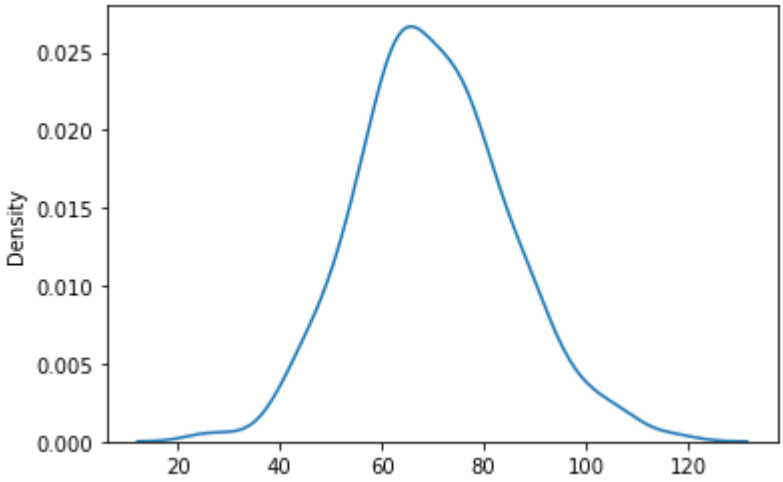
```
# display first few values in scores
scores.head()
```

Out[239…

```
0    66.287644
1    66.828713
2    50.950913
3    62.745157
4    64.495271
dtype: float64
```

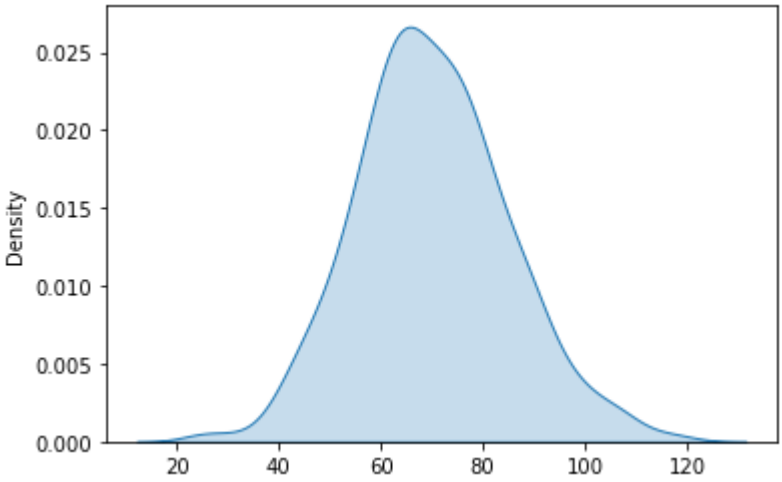
In [240…

```
# create a kernel density estimate plot
# to visualize the univariate distribution of scores
sns.kdeplot(scores);
```



In [241…

```
# create a shaded kernel density estimate plot
# to visualize the univariate distribution of scores
sns.kdeplot(scores, shade=True);
```



In [242…

```
# display first few rows of midterms
midterms.head()
```

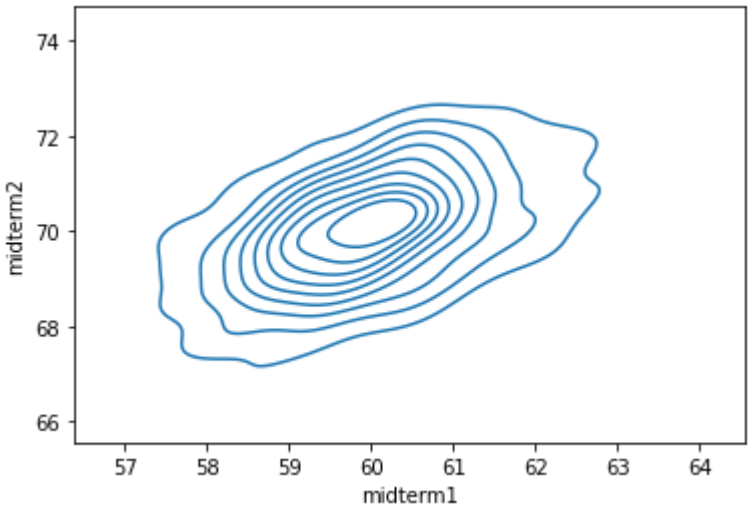
Out[242…

	midterm1	midterm2
0	59.850150	69.704398
1	60.497660	71.662953
2	59.089453	69.945242
3	58.844031	68.423547
4	59.241816	70.689407

In [243…

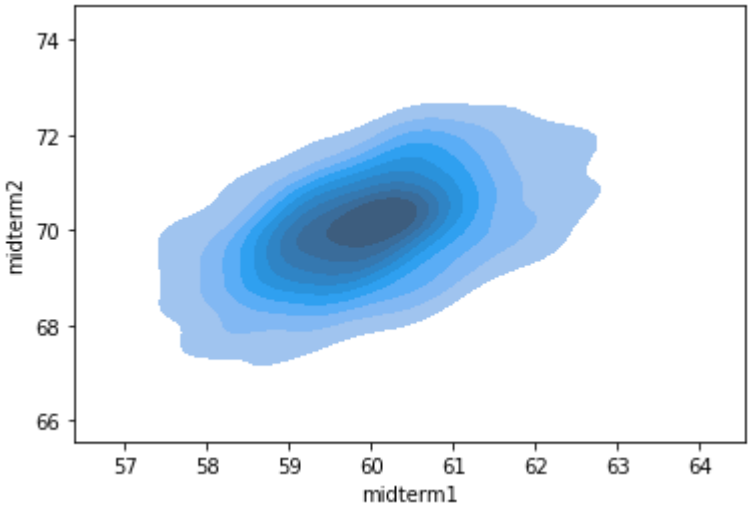
```
# create a two-dimensional kernel density estimate plot
# to visualize the bivariate distribution of midterm1 and midterm2
sns.kdeplot(midterms['midterm1'], midterms['midterm2']);
```

```
S:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the followi
ng variable as a keyword arg: y. From version 0.12, the only valid positional argument w
ill be `data`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```



In [244…
```
# create a shaded two-dimensional kernel density estimate plot
# to visualize the bivariate distribution of midterm1 and midterm2
sns.kdeplot(midterms['midterm1'], midterms['midterm2'], shade=True);
```

```
S:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the followi
ng variable as a keyword arg: y. From version 0.12, the only valid positional argument w
ill be `data`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```



Violin Plots

In [245…
```
# create a numpy array representing the scores of 400 students
# on midterm1
m1 = np.random.normal(70, 10, size=400)

# create a numpy array representing the students' scores
# on midterm2
m2 = np.random.normal(80, 15, size=400)

# create a numpy array representing the students' scores
# on the final exam
final = np.random.normal(75, 20, size=400)

# create a list of the exam types
exams = ['midterm1']*400 + ['midterm2']*400 + ['final']*400

# create a numpy array containing all the scores
scores = np.append(m1, np.array([m2, final]))

# create a pandas dataframe representing midterm1, midterm2, and final exam scores
# for the class of 400 students
exam_scores = pd.DataFrame({'exam': exams, 'score': scores})
```

In [246…
```
# display first few rows of exam_scores
exam_scores.head()
```

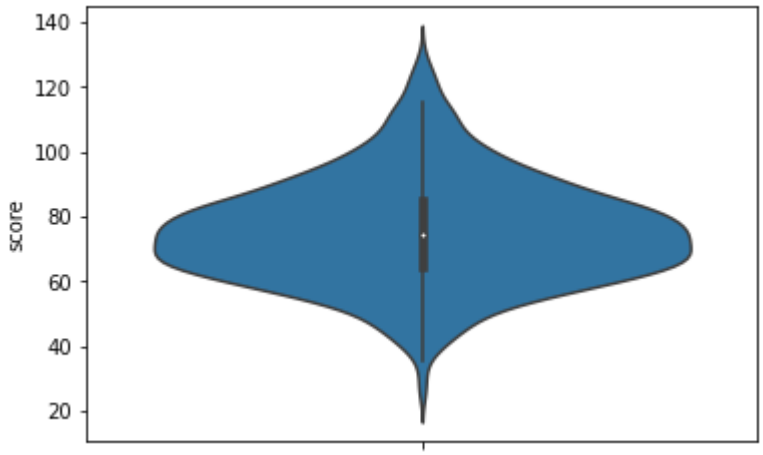
Out[246…

	exam	score

	exam	score
0	midterm1	63.484465
1	midterm1	65.329124
2	midterm1	71.971705
3	midterm1	62.939158
4	midterm1	77.619794

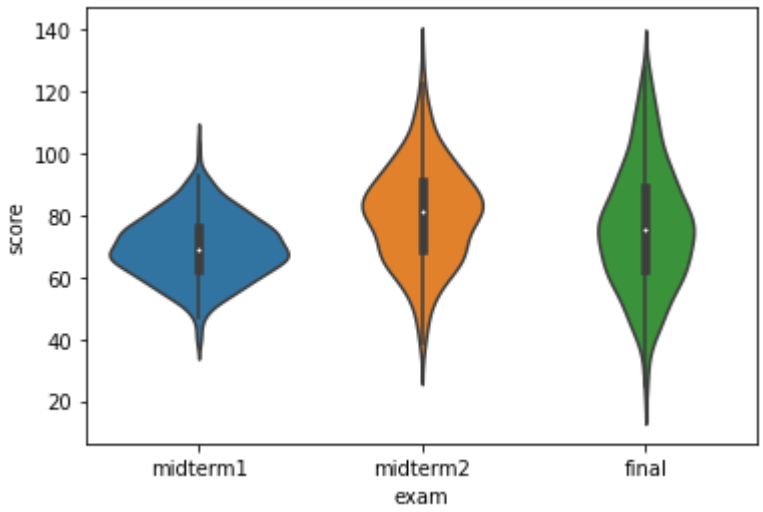
In [247…

```
# draw a vertical violin plot of the scores
sns.violinplot(y='score', data=exam_scores);
```



In [248…

```
# draw a vertical violin plot of the scores grouped by exam type
sns.violinplot(x='exam', y='score', data=exam_scores);
```



Heatmaps

In [249…

```
# create a variable that contains a numpy array representing the sales of a small busin
data = np.random.randint(100, 301, size=(12,12))

# create a variable that contains the list of months
```

```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
          'August', 'September', 'October', 'November', 'December']

# create a variable that contains a list of the years from 2008 to 2019 inclusive
years = [2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]

# create a variable named sales that contains a dataframe
# the dataframe consists of the sales of a small business across 12 months
# there is an entry for each month of each year between 2008 and 2019 inclusive
sales = pd.DataFrame(data, index=months, columns=years)
```

In [250…

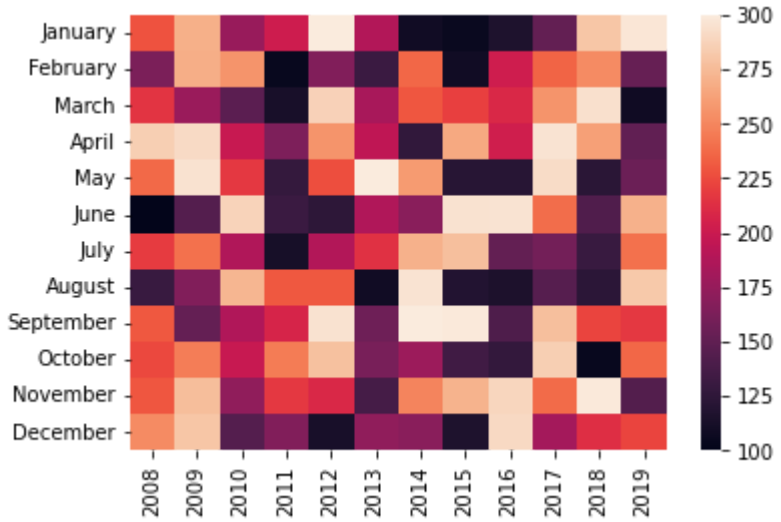
```
# display first few rows of sales
sales.head()
```

Out[250…

	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
January	228	269	176	201	300	188	109	105	116	151	280	297
February	163	268	256	104	165	131	236	109	202	235	252	153
March	215	177	147	113	286	183	229	220	209	256	294	108
April	285	291	198	164	256	194	126	266	203	296	261	150
May	237	295	217	128	226	300	259	121	121	292	122	155

In [251…

```
# create a heatmap of sales
sns.heatmap(sales);
```



In [ ]: