In [1]:
```
ls
```

```
Untitled.ipynb  starbucks.csv
```

In [52]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

In [50]:
```
coffee = pd.read_csv('starbucks.csv')
ages = coffee['age']
distance = coffee['nearest_starbucks']


distance_mean = np.mean(distance)
```

In [51]:
```
distance_mean
```

Out[51]:
```
3.0245901639344264
```

In [53]:
```
coffee = pd.read_csv('starbucks.csv')
ages = coffee['age']
scaler = StandardScaler()
ages_reshaped =  np.array(ages).reshape(-1,1)
#print("========")
print(ages_reshaped)
#print("========")

## fit_transform
ages_scaled = scaler.fit_transform(ages_reshaped)
#print("========")
print(ages_scaled)
#print("========")
## print mean and standard deviation
print(np.mean(ages_scaled))
print(np.std(ages_scaled))
```

```
[[52]
 [35]
 [29]
 [28]
 [28]
 [28]
 [27]
 [26]
 [26]
 [26]
 [51]
 [26]
 [22]
 [21]
 [17]
 [15]
 [13]
 [65]
 [42]
```

In [9]:
```
pwd
```

Out[9]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

Data standardization is the process of rescaling the attributes so that they have mean as 0 and variance as 1. The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values. In sklearn.preprocessing.StandardScaler(), centering and scaling happens independently on each feature.

In [11]:
```
#Min & Max Normalization
```

In [13]:
```
import pandas as pd
import numpy as np

coffee = pd.read_csv('starbucks.csv')
spent = coffee['avg_spent']
max_spent = np.max(spent)
print(max_spent)
min_spent = np.min(spent)
print(min_spent)
spent_range = max_spent - min_spent

spent_normalized = (spent-min_spent)/(max_spent-min_spent)
print(spent_normalized)
```

```
28
0
0      0.464286
1      0.892857
2      0.357143
3      0.250000
4      0.357143
         ...
117    0.178571
118    0.571429
119    0.071429
120    0.678571
121    0.107143
Name: avg_spent, Length: 122, dtype: float64
```

In [14]:
```
#do the above using sklearn
```

In [17]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
mmscaler = MinMaxScaler()
coffee = pd.read_csv('starbucks.csv')
spent = coffee['avg_spent']
spent_reshaped = np.array(spent).reshape(-1,1)
reshaped_scaled = mmscaler.fit_transform(spent_reshaped)
print(np.min(reshaped_scaled))
print(np.max(reshaped_scaled))
print(reshaped_scaled)
```

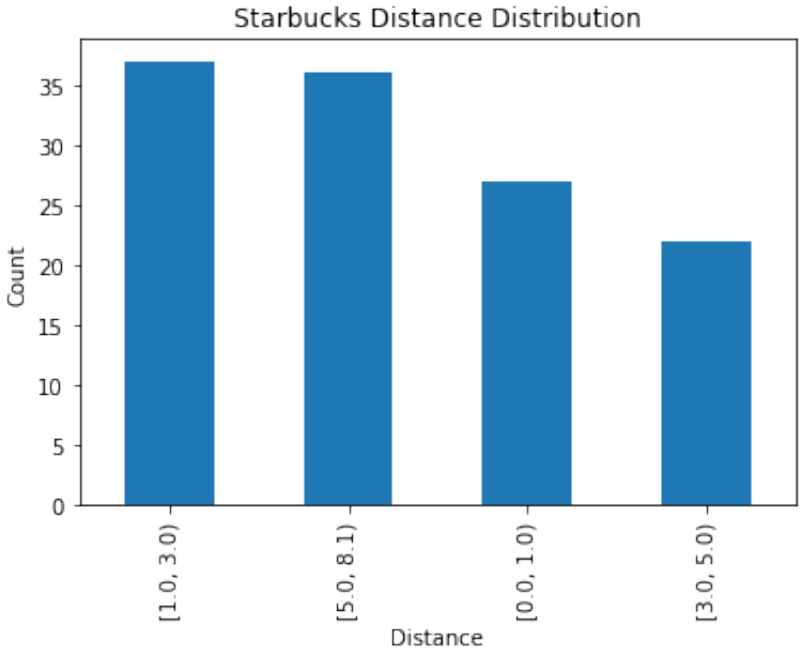
```
0.0
1.0
[[0.46428571]
 [0.89285714]
 [0.35714286]
 [0.25      ]
 [0.35714286]
 [0.39285714]
 [0.75      ]
 [0.53571429]
 [0.        ]
 [0.25      ]
 [0.07142857]
 [0.32142857]
 [0.53571429]
 [0.        ]
 [0.        ]
 [0.07142857]
 [0.46428571]
```

In [20]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [26]:
```
coffee = pd.read_csv('starbucks.csv')
bins = [0, 1, 3, 5, 8.1]
coffee['binned_distance'] = pd.cut(coffee['nearest_starbucks'], bin

print(coffee[['binned_distance', 'nearest_starbucks']].tail(30))
```

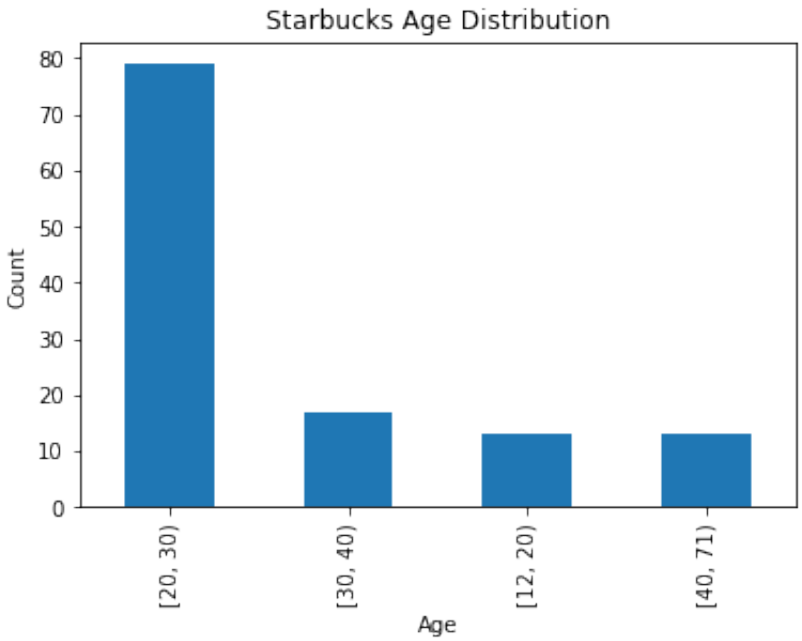
```
    binned_distance  nearest_starbucks
92       [1.0, 3.0)                  1
93       [1.0, 3.0)                  1
94       [1.0, 3.0)                  1
95       [0.0, 1.0)                  0
96       [0.0, 1.0)                  0
97       [0.0, 1.0)                  0
98       [0.0, 1.0)                  0
99       [0.0, 1.0)                  0
100      [0.0, 1.0)                  0
101      [0.0, 1.0)                  0
102      [0.0, 1.0)                  0
103      [0.0, 1.0)                  0
104      [0.0, 1.0)                  0
105      [0.0, 1.0)                  0
106      [0.0, 1.0)                  0
107      [0.0, 1.0)                  0
108      [0.0, 1.0)                  0
109      [0.0, 1.0)                  0
110      [0.0, 1.0)                  0
111      [0.0, 1.0)                  0
112      [0.0, 1.0)                  0
113      [0.0, 1.0)                  0
114      [0.0, 1.0)                  0
115      [0.0, 1.0)                  0
116      [0.0, 1.0)                  0
117      [0.0, 1.0)                  0
118      [0.0, 1.0)                  0
119      [0.0, 1.0)                  0
120      [0.0, 1.0)                  0
121      [0.0, 1.0)                  0
```

In [27]:
```
coffee['binned_distance'].value_counts().plot(kind='bar')

# Label the bar graph
plt.title('Starbucks Distance Distribution')
plt.xlabel('Distance')
plt.ylabel('Count')

# Show the bar graph
plt.show()
```

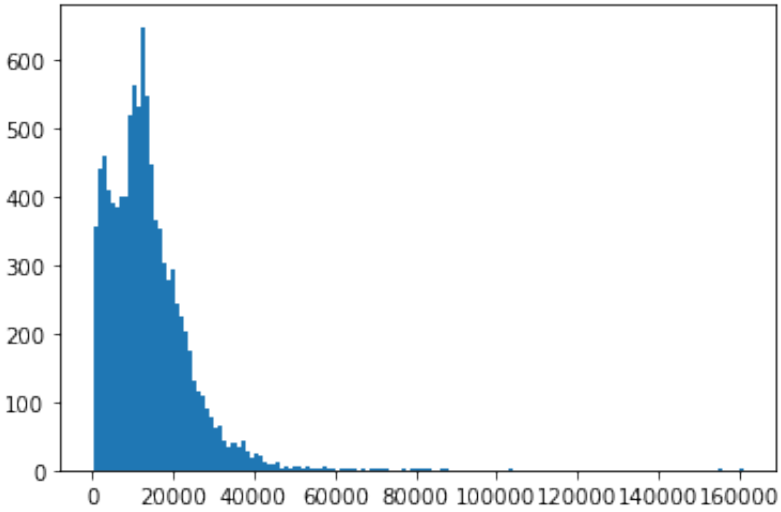


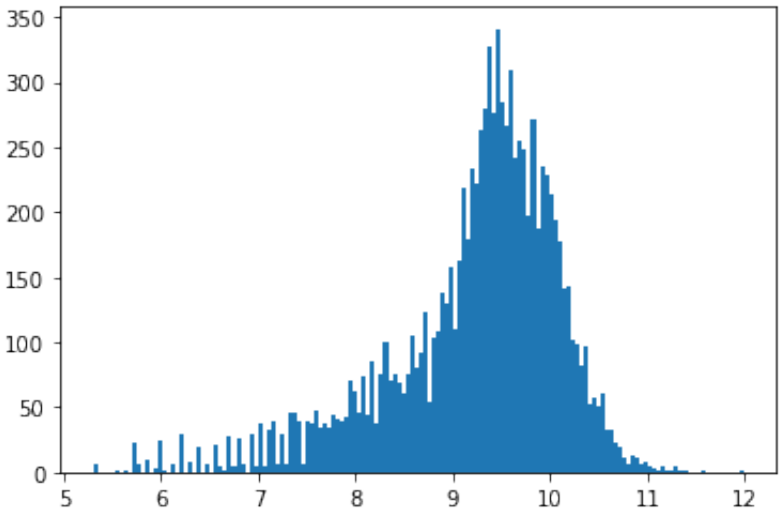
In [32]:
```
coffee = pd.read_csv('starbucks.csv')
ages = coffee['age']
print(np.min(ages))
print(np.max(ages))
age_bins = [12, 20, 30, 40, 71]
coffee['binned_ages'] = pd.cut(coffee['age'], age_bins, right = Fal
print(coffee['binned_ages'].head(10))
coffee['binned_ages'].value_counts().plot(kind='bar')
plt.title("Starbucks Age Distribution")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

```
13
70
0    [40, 71)
1    [30, 40)
2    [20, 30)
3    [20, 30)
4    [20, 30)
5    [20, 30)
6    [20, 30)
7    [20, 30)
8    [20, 30)
9    [20, 30)
Name: binned_ages, dtype: category
Categories (4, interval[int64, left]): [[12, 20) < [20, 30) < [30,
40) < [40, 71)]
```



In [35]:
```
import numpy as np
import matplotlib.pyplot as plt
cars = pd.read_csv('cars.csv')
prices = cars['sellingprice']
plt.hist(prices ,bins = 150)
plt.show()
log_prices = np.log(cars['sellingprice'])
plt.hist(log_prices ,bins = 150)
plt.show()
```





In [38]:
```
cp -p cars.csv cars1.csv
```

The Above is example if Natural Log Transformation Keep in mind, just because your data is skewed does not mean that a log transformation is the best answer. You would not want to log transform your feature if: You have values less than 0. The natural logarithm (which is what we've been talking about) of a negative number is undefined. You have left-skewed data. That data may call for a square or cube transformation. You have non-parametric data

In [39]:
```
ls
```

```
Untitled.ipynb  cars.csv        cars1.csv       starbucks.csv
```

In [40]:
```
head cars1.csv
```

```
  Input In [40]
    head cars1.csv
         ^
SyntaxError: invalid syntax
```

In [41]:
```
#Encoding Categorical Variables
```

In [42]:
```
#Ordinal encoding
print(cars['condition'].value_counts())
# #OUTPUT
# New          2881
# Like New     2860
# Good         2027
# Fair          753
# Excellent     186
```

```
1.9    732
3.5    507
4.3    475
4.4    468
4.1    460
3.7    446
3.6    422
4.2    404
2.0    376
4.0    370
2.9    357
3.9    351
2.8    338
2.7    303
3.4    303
3.8    297
2.6    225
4.9    224
4.6    220
4.8    220
4.5    219
2.5    203
3.3    199
3.0    197
4.7    191
5.0    186
3.2    179
2.4    175
2.1    162
3.1    156
2.3    154
1.0    127
2.2    110
1.8      8
1.7      5
1.3      3
1.4      2
1.6      1
1.5      1
1.1      1
Name: condition, dtype: int64
```

In [43]:
```
import pandas as pd

# import data
cars = pd.read_csv('cars.csv')

# check variable types
print(cars.dtypes)
```

```
Unnamed: 0        int64
year              int64
make             object
model            object
trim             object
body             object
transmission     object
vin              object
state            object
condition       float64
odometer        float64
color            object
interior         object
seller           object
mmr               int64
sellingprice      int64
saledate         object
Unnamed: 16     float64
dtype: object
```

In [44]:
```
print(cars['condition'].value_counts())
```

```
1.9    732
3.5    507
4.3    475
4.4    468
4.1    460
3.7    446
3.6    422
4.2    404
2.0    376
4.0    370
2.9    357
3.9    351
2.8    338
2.7    303
3.4    303
3.8    297
2.6    225
4.9    224
4.6    220
4.8    220
4.5    219
2.5    203
3.3    199
3.0    197
4.7    191
5.0    186
3.2    179
2.4    175
2.1    162
3.1    156
2.3    154
1.0    127
2.2    110
1.8      8
1.7      5
1.3      3
1.4      2
1.6      1
1.5      1
1.1      1
Name: condition, dtype: int64
```

In [45]:
```
#please look at notes to understand Encoding Categorical Variables
```

In [46]:
```
#Ordinal encoding
#Label encoding
#One-hot encoding
#Binary encoding
#Hashing
#Target encoding
#Date-time encoding
```

In [47]:
```
#Transforming Data into Features Project
```

In [48]:
```
ls
```

```
Untitled.ipynb  cars.csv        cars1.csv       starbucks.csv
```

In [50]:
```
cp -p /Users/kamallakannansekar/Downloads/Womens\ Clothing\ E-Comme
```

In [51]:
```
ls
```

```
Untitled.ipynb                          cars1.csv
Womens Clothing E-Commerce Reviews.csv  starbucks.csv
cars.csv
```

In [52]:
```
head Womens\ Clothing\ E-Commerce\ Reviews.csv
```

```
  Input In [52]
    head Womens\ Clothing\ E-Commerce\ Reviews.csv
         ^
SyntaxError: invalid syntax
```

In [53]:
```
mv Womens\ Clothing\ E-Commerce\ Reviews.csv reviews.csv
```

In [54]:
```
ls
```

```
Untitled.ipynb  cars.csv        cars1.csv       reviews.csv     st
arbucks.csv
```

In [73]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
```

In [74]:
```
reviews = pd.read_csv('reviews.csv')
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Unnamed: 0               23486 non-null  int64 
 1   Clothing ID              23486 non-null  int64 
 2   Age                      23486 non-null  int64 
 3   Title                    19676 non-null  object
 4   Review Text              22641 non-null  object
 5   Rating                   23486 non-null  int64 
 6   Recommended IND          23486 non-null  int64 
 7   Positive Feedback Count  23486 non-null  int64 
 8   Division Name            23472 non-null  object
 9   Department Name          23472 non-null  object
 10  Class Name               23472 non-null  object
dtypes: int64(6), object(5)
memory usage: 2.0+ MB
```

In [75]:
```
print(reviews['Recommended IND'].value_counts())
```

```
1    19314
0     4172
Name: Recommended IND, dtype: int64
```

In [61]:
```
#lets reverse it for FUN
```

In [76]:
```
binary_dict = {0:True, 1:False}
```

In [77]:
```
reviews['Recommended IND'] = reviews['Recommended IND'].map(binary_
```

In [78]:
```
print(reviews['Recommended IND'].value_counts())
```

```
False    19314
True      4172
Name: Recommended IND, dtype: int64
```

In [79]:
```
binary_dict1 = {True:0, False:1}
```

In [80]:
```
reviews['Recommended IND'] = reviews['Recommended IND'].map(binary_
```

In [81]:
```
print(reviews['Recommended IND'].value_counts())
```

```
1    19314
0     4172
Name: Recommended IND, dtype: int64
```

In [82]:
```
print(reviews['Recommended IND'].value_counts())
print(reviews['Rating'].value_counts())
```

```
1    19314
0     4172
Name: Recommended IND, dtype: int64
5    13131
4     5077
3     2871
2     1565
1      842
Name: Rating, dtype: int64
```

In [83]:
```
#rating_dict = {'Loved it':5, 'Liked it':4, 'Was okay':3, 'Not grea
#reviews['Rating'] = reviews['Rating'].map(rating_dict)
print(reviews['Rating'].value_counts())
```

```
5    13131
4     5077
3     2871
2     1565
1      842
Name: Rating, dtype: int64
```

In [84]:
```
print(reviews['Department Name'].value_counts())
```

```
Tops        10468
Dresses      6319
Bottoms      3799
Intimate     1735
Jackets      1032
Trend         119
Name: Department Name, dtype: int64
```

In [85]:
```
one_hot = pd.get_dummies(reviews['Department Name'])
reviews = reviews.join(one_hot)
print(reviews.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Unnamed: 0               23486 non-null  int64
 1   Clothing ID              23486 non-null  int64
 2   Age                      23486 non-null  int64
 3   Title                    19676 non-null  object
 4   Review Text              22641 non-null  object
 5   Rating                   23486 non-null  int64
 6   Recommended IND          23486 non-null  int64
 7   Positive Feedback Count  23486 non-null  int64
 8   Division Name            23472 non-null  object
 9   Department Name          23472 non-null  object
 10  Class Name               23472 non-null  object
 11  Bottoms                  23486 non-null  uint8
 12  Dresses                  23486 non-null  uint8
 13  Intimate                 23486 non-null  uint8
 14  Jackets                  23486 non-null  uint8
 15  Tops                     23486 non-null  uint8
 16  Trend                    23486 non-null  uint8
dtypes: int64(6), object(5), uint8(6)
memory usage: 2.1+ MB
None
```

In [86]:
```
print(reviews.head(5))
```

```
   Unnamed: 0  Clothing ID  Age                    Title  \
0           0          767   33                      NaN
1           1         1080   34                      NaN
2           2         1077   60  Some major design flaws
3           3         1049   50        My favorite buy!
4           4          847   47        Flattering shirt

                                         Review Text  Rating  Reco
mmended IND  \
0  Absolutely wonderful - silky and sexy and comf...       4
1
1  Love this dress!  it's sooo pretty.  i happene...       5
1
2  I had such high hopes for this dress and reall...       3
0
3  I love, love, love this jumpsuit. it's fun, fl...       5
1
4  This shirt is very flattering to all due to th...       5
1

   Positive Feedback Count   Division Name Department Name Class N
ame  \
0                        0       Initmates        Intimate  Intima
tes
1                        4         General         Dresses    Dres
ses
2                        0         General         Dresses    Dres
ses
3                        0  General Petite         Bottoms      Pa
nts
4                        6         General            Tops    Blou
ses

   Bottoms  Dresses  Intimate  Jackets  Tops  Trend
0        0        0         1        0     0      0
1        0        1         0        0     0      0
2        0        1         0        0     0      0
3        1        0         0        0     0      0
4        0        0         0        0     1      0
```

In [88]:
```
reviews = reviews[['Clothing ID', 'Age', 'Recommended IND' , 'Ratin
```

In [90]:
```
reviews = reviews.set_index('Clothing ID')
```

In [91]:
```
print(reviews.head(30))
print("++++++++++++++++++++++++++++++++++++")
```

```
             Age  Recommended IND  Rating  Bottoms  Dresses  Intim
ate  \
Clothing ID
```

767	33	1	4	0	0
1					
1080	34	1	5	0	1
0					
1077	60	0	3	0	1
0					
1049	50	1	5	1	0
0					
847	47	1	5	0	0
0					
1080	49	0	2	0	1
0					
858	39	1	5	0	0
0					
858	39	1	4	0	0
0					
1077	24	1	5	0	1
0					
1077	34	1	5	0	1
0					
1077	53	0	3	0	1
0					
1095	39	1	5	0	1
0					
1095	53	1	5	0	1
0					
767	44	1	5	0	0
1					
1077	50	1	3	0	1
0					
1065	47	1	4	1	0
0					
1065	34	1	3	1	0
0					
853	41	1	5	0	0
0					
1120	32	1	5	0	0
0					
1077	47	1	5	0	1
0					
847	33	1	4	0	0
0					
1080	55	1	4	0	1
0					
1077	31	0	2	0	1
0					
1077	34	1	3	0	1
0					
847	55	1	5	0	0
0					
697	31	0	3	0	0
1					
949	33	0	2	0	0
0					

```
1003          31                1       4        1        0
0

684           53                1       5        0        0
1
4             28                1       5        0        0
0

              Jackets  Tops  Trend
Clothing ID
767                 0     0      0
1080                0     0      0
1077                0     0      0
1049                0     0      0
847                 0     1      0
1080                0     0      0
858                 0     1      0
858                 0     1      0
1077                0     0      0
1077                0     0      0
1077                0     0      0
1095                0     0      0
1095                0     0      0
767                 0     0      0
1077                0     0      0
1065                0     0      0
1065                0     0      0
853                 0     1      0
1120                1     0      0
1077                0     0      0
847                 0     1      0
1080                0     0      0
1077                0     0      0
1077                0     0      0
847                 0     1      0
697                 0     0      0
949                 0     1      0
1003                0     0      0
684                 0     0      0
4                   0     1      0
++++++++++++++++++++++++++++++++++++
```

In [92]:
```
scaler = StandardScaler()
scaler.fit_transform(reviews)
print(reviews.head(30))
```

```
              Age  Recommended IND  Rating  Bottoms  Dresses  Intim
ate  \
Clothing ID
767            33                1       4        0        0
1
1080           34                1       5        0        1
0
1077           60                0       3        0        1
```

1077	50	0	3	0	1
0					
1049	50	1	5	1	0
0					
847	47	1	5	0	0
0					
1080	49	0	2	0	1
0					
858	39	1	5	0	0
0					
858	39	1	4	0	0
0					
1077	24	1	5	0	1
0					
1077	34	1	5	0	1
0					
1077	53	0	3	0	1
0					
1095	39	1	5	0	1
0					
1095	53	1	5	0	1
0					
767	44	1	5	0	0
1					
1077	50	1	3	0	1
0					
1065	47	1	4	1	0
0					
1065	34	1	3	1	0
0					
853	41	1	5	0	0
0					
1120	32	1	5	0	0
0					
1077	47	1	5	0	1
0					
847	33	1	4	0	0
0					
1080	55	1	4	0	1
0					
1077	31	0	2	0	1
0					
1077	34	1	3	0	1
0					
847	55	1	5	0	0
0					
697	31	0	3	0	0
1					
949	33	0	2	0	0
0					
1003	31	1	4	1	0
0					
684	53	1	5	0	0
1					

```
_
4              28                1       5        0        0
0
```

Clothing ID	Jackets	Tops	Trend
767	0	0	0
1080	0	0	0
1077	0	0	0
1049	0	0	0
847	0	1	0
1080	0	0	0
858	0	1	0
858	0	1	0
1077	0	0	0
1077	0	0	0
1077	0	0	0
1095	0	0	0
1095	0	0	0
767	0	0	0
1077	0	0	0
1065	0	0	0
1065	0	0	0
853	0	1	0
1120	1	0	0
1077	0	0	0
847	0	1	0
1080	0	0	0
1077	0	0	0
1077	0	0	0
847	0	1	0
697	0	0	0
949	0	1	0
1003	0	0	0
684	0	0	0
4	0	1	0

In [ ]:

In [93]:
```
# Simple y = mx + b
```

In [99]:
```
x = [1, 2, 3]
y = [5, 1, 3]


#y = x
m1 = 1
b1 = 0

y_predicted1 = [m1*x+b1 for x in x]

#y = 0.5x + 1
m2 = 0.5
b2 = 1
y_predicted2 = [m2*x+b2 for x in x]
total_loss1 = 0
for i in range(len(y)):
  total_loss1 = total_loss1 + (y[i]-y_predicted1[i])**2

total_loss2 = 0
for i in range(len(y)):
  total_loss2 = total_loss2 + (y[i]-y_predicted2[i])**2

print(total_loss1)
print(total_loss2)
better_fit = 2
```

```
17
13.5
```

In [95]:
```
#2nd line is better fit here as the error rate is less
```

In [100]:
```
#The goal of a linear regression model is to find the slope and int
```

Gradient Descent for Intercept As we try to minimize loss, we take each parameter we are changing, and move it as long as we are decreasing loss. It's like we are moving down a hill, and stop once we reach the bottom:

The process by which we do this is called gradient descent. We move in the direction that decreases our loss the most. Gradient refers to the slope of the curve at any point.

Simple Gradient Descent

def get_gradient_at_b(x,y,m,b): diff = 0 N = len(x) for i in range(0, len(x)): diff = diff + (y[i]-((m*x[i])+b)) *b_gradient* = -2/*N*(diff) return b_gradient

To obtain the gradient descent for the intercept, we calculate the partial derivative of the error function with respect to b, resulting in: GradDescForIntercept

To obtain the gradient descent for the slope, we calculate the partial derivative of the error function with respect to m instead, resulting in:

Now that we know how to calculate the gradient, we want to take a “step” in that direction. However, it’s important to think about whether that step is too big or too small. We don’t want to overshoot the minimum error!

We can scale the size of the step by multiplying the gradient by a learning rate.

To find a new b value, we would say:

new_b = current_b - (learning_rate * b_gradient) where current_b is our guess for what the b value is, b_gradient is the gradient of the loss curve at our current guess, and learning_rate is proportional to the size of the step we want to take.

In a few exercises, we’ll talk about the implications of a large or small learning rate, but for now, let’s use a fairly small value

In [101]:
```
def get_gradient_at_b(x, y, b, m):
  N = len(x)
  diff = 0
  for i in range(N):
    x_val = x[i]
    y_val = y[i]
    diff += (y_val - ((m * x_val) + b))
  b_gradient = -(2/N) * diff
  return b_gradient

def get_gradient_at_m(x, y, b, m):
  N = len(x)
  diff = 0
  for i in range(N):
      x_val = x[i]
      y_val = y[i]
      diff += x_val * (y_val - ((m * x_val) + b))
  m_gradient = -(2/N) * diff
  return m_gradient

# Define your step_gradient function here

def step_gradient(x, y, b_current, m_current):
  b_gradient = get_gradient_at_b(x, y, b_current, m_current)
  m_gradient = get_gradient_at_m(x, y, b_current, m_current)
  b = b_current - (0.01 * b_gradient)
  m = m_current - (0.01 * m_gradient)
  return b,m









months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
revenue = [52, 74, 79, 95, 115, 110, 129, 126, 147, 146, 156, 184]

# current intercept guess:
b = 0
# current slope guess:
m = 0
b, m = step_gradient(months, revenue, b, m)

# Call your function here to update b and m
print(b, m)
```

```
2.355 17.78333333333333
```

How do we know when we should stop changing the parameters m and b? How will we know when our program has learned enough?

To answer this, we have to define convergence. Convergence is when the loss stops changing (or changes very slowly) when parameters are changed.

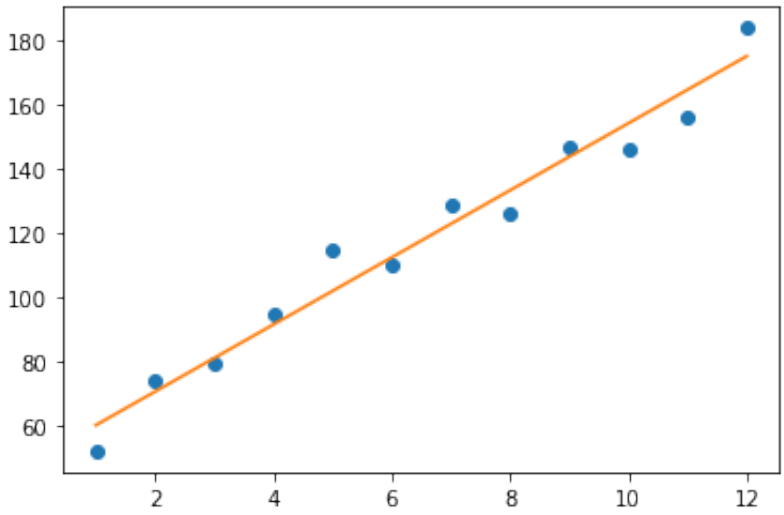
Hopefully, the algorithm will converge at the best values for the parameters m and b.

In [102]:
```
#Full Program Below
```

In [122]:
```
import matplotlib.pyplot as plt

def get_gradient_at_b(x, y, b, m):
  N = len(x)
  diff = 0
  for i in range(N):
    x_val = x[i]
    y_val = y[i]
    diff += (y_val - ((m * x_val) + b))
  b_gradient = -(2/N) * diff
  return b_gradient

def get_gradient_at_m(x, y, b, m):
  N = len(x)
  diff = 0
  for i in range(N):
      x_val = x[i]
      y_val = y[i]
      diff += x_val * (y_val - ((m * x_val) + b))
  m_gradient = -(2/N) * diff
  return m_gradient

#Your step_gradient function here

def gradient_descent(x, y, learning_rate, num_iterations):
  b = 0
  m = 0
  for i in range(num_iterations):
   b,m = step_gradient(b,m,x,y,learning_rate)
  return b,m

def step_gradient(b_current, m_current, x, y, learning_rate):
    b_gradient = get_gradient_at_b(x, y, b_current, m_current)
    m_gradient = get_gradient_at_m(x, y, b_current, m_current)
    b = b_current - (learning_rate * b_gradient)
    m = m_current - (learning_rate * m_gradient)
    return [b, m]

#Your gradient_descent function here:

months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
revenue = [52, 74, 79, 95, 115, 110, 129, 126, 147, 146, 156, 184]
```

```
revenue = [52, 74, 79, 95, 115, 110, 129, 126, 147, 146, 156, 164]

#Uncomment the line below to run your gradient_descent function
b, m = gradient_descent(months, revenue, .01, 1000)

#Uncomment the lines below to see the line you've settled upon!
y = [m*x + b for x in months]
print(m)
print(b)

plt.plot(months, revenue, "o")
plt.plot(months, y)

plt.show()
```

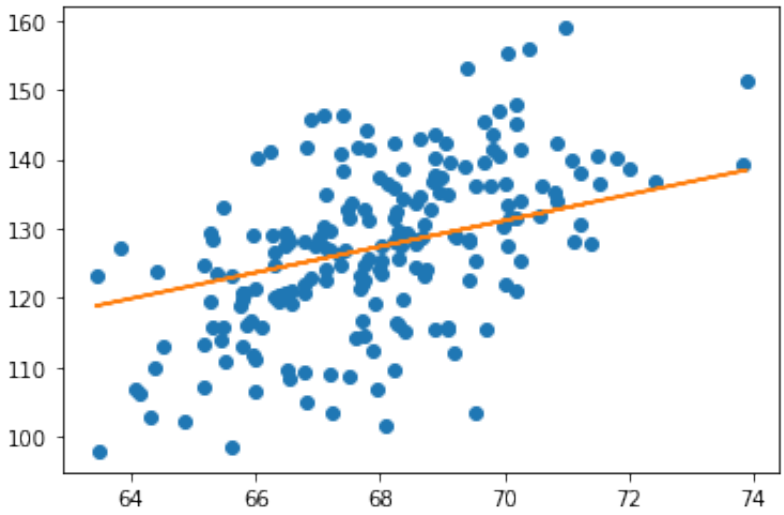
```
10.463427732364998
49.60215351339813
```



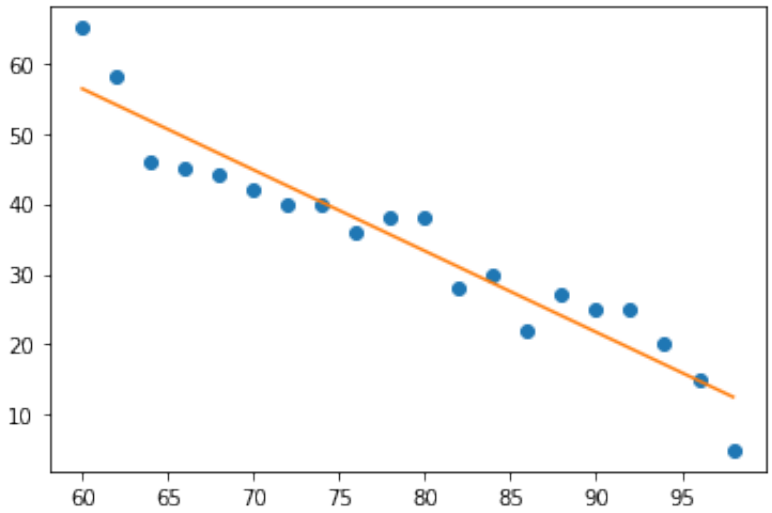
In [123]:
```
ls
```

```
Untitled.ipynb                gradient_descent_funcs.py
cars.csv                      reviews.csv
cars1.csv                     starbucks.csv
```

In [126]:
```
#import codecademylib3_seaborn
from gradient_descent_funcs import gradient_descent
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("heights.csv")

X = df["height"]
y = df["weight"]
b,m = gradient_descent(X, y, .0001, 1000)
y_predictions = [ m*x + b for x in X]

plt.plot(X, y, 'o')
plt.plot(X,y_predictions)
#plot your line here:

plt.show()
```



In [127]:
```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np

temperature = np.array(range(60, 100, 2))
temperature = temperature.reshape(-1, 1)
sales = [65, 58, 46, 45, 44, 42, 40, 40, 36, 38, 38, 28, 30, 22, 27

plt.plot(temperature, sales, 'o')

line_fitter = LinearRegression()
line_fitter.fit(temperature, sales)
sales_predict = line_fitter.predict(temperature)

plt.plot(temperature, sales_predict)
plt.show()
```



In [128]:
```
#The above does the same with out manaully calculating the m & b ,
```

In [130]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston

# Boston housing dataset
boston = load_boston()

df = pd.DataFrame(boston.data, columns = boston.feature_names)

# Set the x-values to the nitrogen oxide concentration:
X = df[['NOX']]
# Y-values are the prices:
y = boston.target

# Can we do linear regression on this?
line_fitter = LinearRegression()
line_fitter.fit(X, y)
price_predict = line_fitter.predict(X)

plt.plot(X, price_predict)



plt.scatter(X, y, alpha=0.4)
# Plot line here:

plt.title("Boston Housing Dataset")
plt.xlabel("Nitric Oxides Concentration")
plt.ylabel("House Price ($)")
plt.show()
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/utils/deprecation.py:87: FutureWarning: Function load_bo
ston is deprecated; `load_boston` is deprecated in 1.0 and will be
removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You
can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the
use of this
    dataset unless the purpose of the code is to study and educate
about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the origi
nal
    source::

        import pandas as pd
```

In [131]:
```
# play with https://scikit-learn.org/stable/datasets/toy_dataset.ht
```

In [132]:
```
#Purpose of Gradient Descent - to move the parameters to minimize t
```

In [133]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
```

In [137]:
```
from sklearn.datasets import load_diabetes
```

In [138]:
```
diabetes = load_diabetes()
```

In [139]:
```
df = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
```

In [140]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
```

In [141]:
```
df.head()
```

Out[141]:

	age	sex	bmi	bp	s1	s2	s3	s4	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	(
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-(
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	(
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	(
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-(

In [142]:
```
df.tail()
```

Out[142]:

	age	sex	bmi	bp	s1	s2	s3	s4
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309
439	0.041708	0.050680	-0.015906	0.017282	-0.037344	-0.013840	-0.024993	-0.011080
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560
441	-0.045472	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173816	-0.039493

In [143]:
```
#useless above
```

In [144]:
```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
...                           'Parrot', 'Parrot'],
...                   'Max Speed': [380., 370., 24., 26.]})
```

In [145]:
```
df.groupby(['Animal']).mean()
```

Out[145]:

Animal	Max Speed
Falcon	375.0
Parrot	25.0

In [146]:
```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
...                           'Parrot', 'Parrot'],
...                   'Max Speed': [380., 370., 24., 26.]})
```

In [147]:
```
df.groupby(['Max Speed']).mean()
```

```
/var/folders/v_/dt4zl16x3zbd4h2zk9d9rbs00000gn/T/ipykernel_23101/1
938835801.py:1: FutureWarning: Dropping invalid columns in DataFra
meGroupBy.mean is deprecated. In a future version, a TypeError wil
l be raised. Before calling .mean, select only columns which shoul
d be valid for the function.
  df.groupby(['Max Speed']).mean()
```

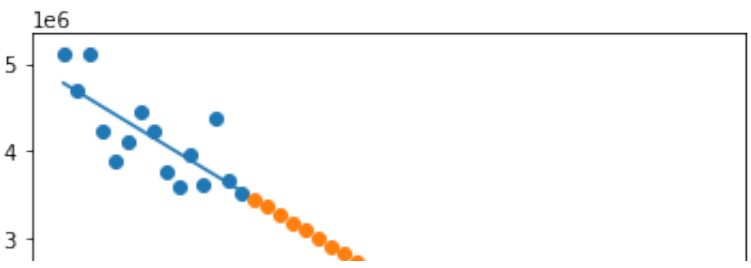
Out[147]:

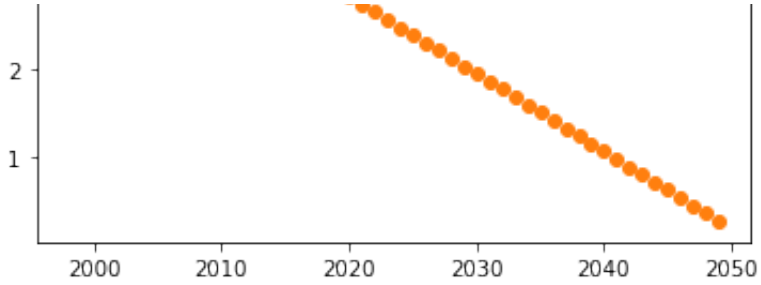
Max Speed
24.0
26.0
370.0
380.0

In [155]:
```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model

df = pd.read_csv("honeyproduction.csv")
#print(df.head())
prod_per_year = df.groupby(['year']).totalprod.mean().reset_index()
print(prod_per_year)
X = prod_per_year['year']
X = X.values.reshape(-1,1)
#print(X)
y = prod_per_year['totalprod']
plt.scatter(X,y)
#plt.show()
regr = linear_model.LinearRegression()
regr.fit(X,y)
print(regr.coef_)
print(regr.intercept_)
y_predict = regr.predict(X)
plt.plot(X,y_predict)
#plt.show()

X_future = np.array(range(2013, 2050))
X_future = X_future.reshape(-1, 1)
future_predict = regr.predict(X_future)
plt.scatter(X_future,future_predict)
plt.show()
```

```
    year     totalprod
0   1998  5.105093e+06
1   1999  4.706674e+06
2   2000  5.106000e+06
3   2001  4.221545e+06
4   2002  3.892386e+06
5   2003  4.122091e+06
6   2004  4.456805e+06
7   2005  4.243146e+06
8   2006  3.761902e+06
9   2007  3.600512e+06
10  2008  3.974927e+06
11  2009  3.626700e+06
12  2010  4.382350e+06
13  2011  3.680025e+06
14  2012  3.522675e+06
[-88303.18915238]
181208083.1073298
```



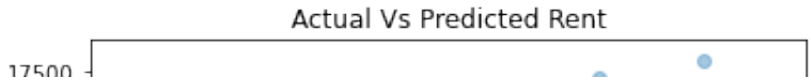


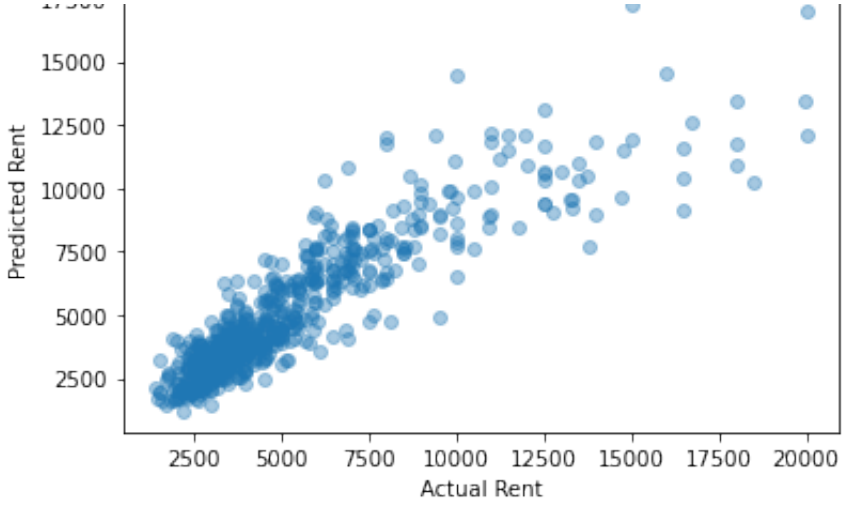
In [ ]:

In [1]:
```
#Muti Linear Regression
```

In [4]:
```
#import codecademylib3_seaborn
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

streeteasy = pd.read_csv("https://raw.githubusercontent.com/sonnyno

df = pd.DataFrame(streeteasy)

x = df[['bedrooms', 'bathrooms', 'size_sqft', 'min_to_subway', 'flo

y = df[['rent']]

x_train, x_test, y_train, y_test = train_test_split(x, y, train_siz

mlr = LinearRegression()
mlr.fit(x_train, y_train)
y_predict = mlr.predict(x_test)

sonny_apartment = [[1, 1, 620, 16, 1, 98, 1, 0, 1, 0, 0,1, 1, 0]]
predict = mlr.predict(sonny_apartment)
print("Predicted rent: $%.2f" % predict)

plt.scatter(y_test, y_predict , alpha=0.4)
plt.xlabel("Actual Rent")
plt.ylabel("Predicted Rent")
plt.title("Actual Vs Predicted Rent")
plt.show()
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/base.py:450: UserWarning: X does not have valid feature
names, but LinearRegression was fitted with feature names
  warnings.warn(

Predicted rent: $2393.58
```



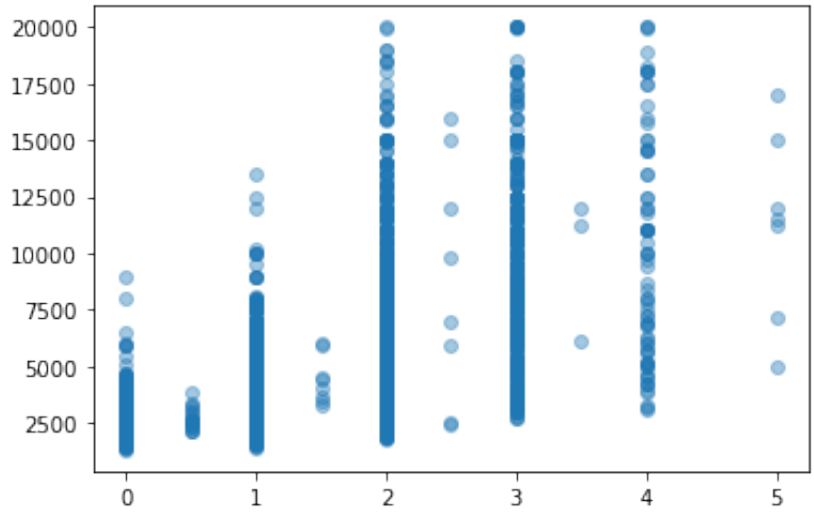


In [5]:
```
print(mlr.coef_)
```

```
[[-302.73009383 1199.3859951     4.79976742  -24.28993151   24.198
24177
    -7.58272473 -140.90664773   48.85017415  191.4257324  -151.114
53388
    89.408889    -57.89714551  -19.31948556  -38.92369828]]
```

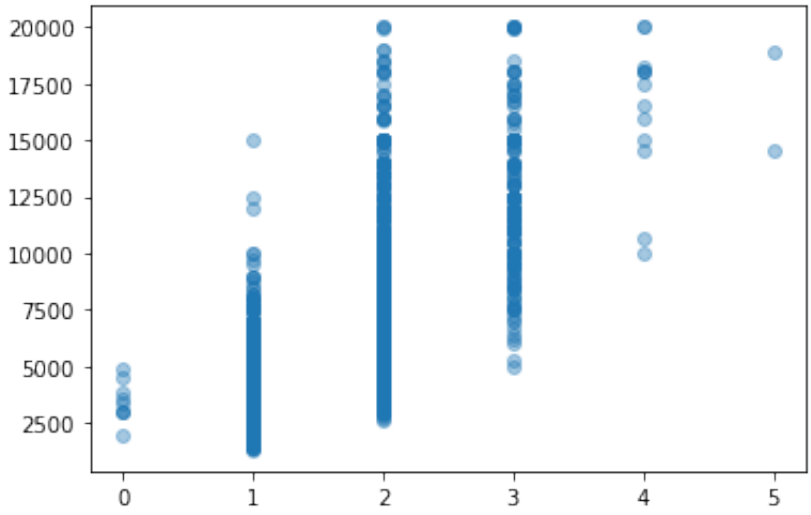
In [6]:
```
plt.scatter(df[['bedrooms']], df[['rent']], alpha=0.4)
```

Out[6]: <matplotlib.collections.PathCollection at 0x7fa108383f10>



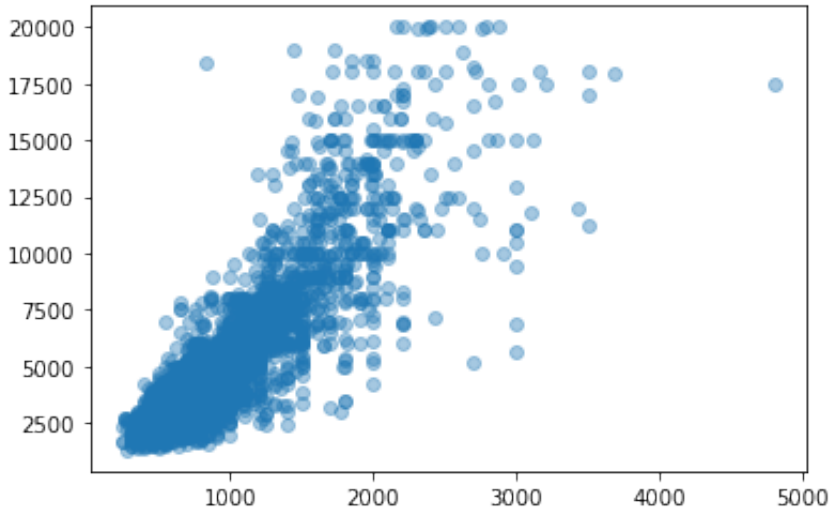
In [7]:
```
plt.scatter(df[['bathrooms']], df[['rent']], alpha=0.4)
```

Out[7]: <matplotlib.collections.PathCollection at 0x7fa108502fd0>



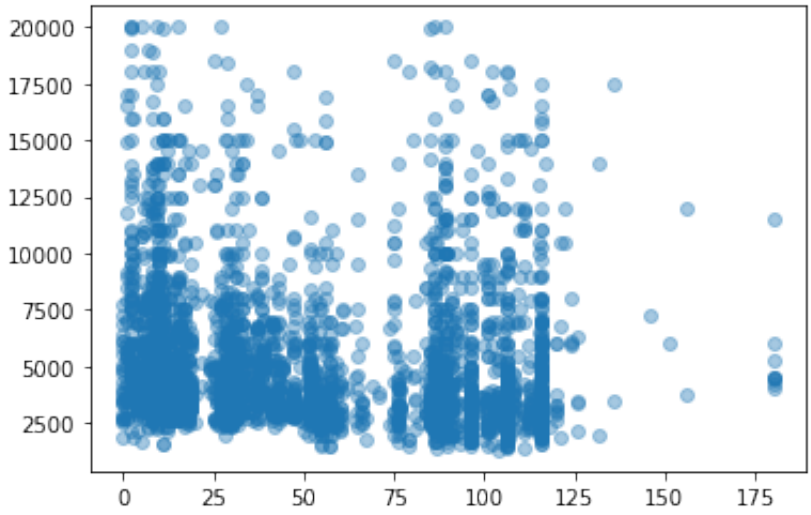
In [8]:
```
plt.scatter(df[['size_sqft']], df[['rent']], alpha=0.4)
```

Out[8]: <matplotlib.collections.PathCollection at 0x7fa11a44bdf0>



In [9]:
```
plt.scatter(df[['building_age_yrs']], df[['rent']], alpha=0.4)
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x7fa128beb550>
```



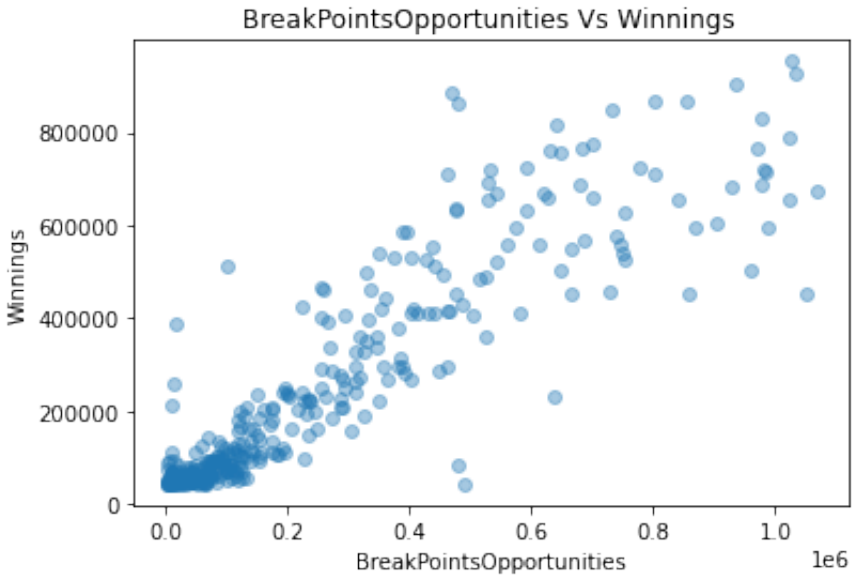
In [10]:
```
print("Train score:")
print(mlr.score(x_train, y_train))

print("Test score:")
print(mlr.score(x_test, y_test))
```

```
Train score:
0.7725460559817883
Test score:
0.8050371975357653
```

In [11]:
```
#Next Programs
```

In [15]:
```
#import codecademylib3_seaborn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('tennis_stats.csv')
#print(df.head())

#plt.scatter(df[['FirstServe']], df[['FirstServePointsWon']], alpha
#plt.show()
#plt.scatter(df[['Winnings']], df[['Ranking']], alpha=0.4)
#plt.show()

#plt.scatter(df[['Ranking']], df[['Winnings']], alpha=0.4)
#plt.show()
x1 = df[['BreakPointsOpportunities']]
y1 = df[['Winnings']]
x_train, x_test, y_train, y_test = train_test_split(x1, y1, train_s
```

```
mlr = LinearRegression()
mlr.fit(x_train, y_train)
y_predict = mlr.predict(x_test)


# load and investigate the data here:

plt.scatter(y_test, y_predict , alpha=0.4)
plt.xlabel("BreakPointsOpportunities")
plt.ylabel("Winnings")
plt.title("BreakPointsOpportunities Vs Winnings ")
plt.show()
TT=[[0.4]]
predict = mlr.predict(TT)
k1 = mlr.score(x_train, y_train)
print("The efficiency is :", k1)
print(predict)
```




```
The efficiency is : 0.8111412774695739
[[44104.96955472]]

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1721 entries, 0 to 1720
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Player                      1721 non-null   object 
 1   Year                        1721 non-null   int64  
 2   FirstServe                  1721 non-null   float64
 3   FirstServePointsWon         1721 non-null   float64
 4   FirstServeReturnPointsWon   1721 non-null   float64
 5   SecondServePointsWon        1721 non-null   float64
 6   SecondServeReturnPointsWon  1721 non-null   float64
 7   Aces                        1721 non-null   int64  
 8   BreakPointsConverted        1721 non-null   float64
 9   BreakPointsFaced            1721 non-null   int64  
 10  BreakPointsOpportunities    1721 non-null   int64  
 11  BreakPointsSaved            1721 non-null   float64
 12  DoubleFaults                1721 non-null   int64  
 13  ReturnGamesPlayed           1721 non-null   int64  
 14  ReturnGamesWon              1721 non-null   float64
 15  ReturnPointsWon             1721 non-null   float64
 16  ServiceGamesPlayed          1721 non-null   int64  
 17  ServiceGamesWon             1721 non-null   float64
 18  TotalPointsWon              1721 non-null   float64
 19  TotalServicePointsWon       1721 non-null   float64
 20  Wins                        1721 non-null   int64  
 21  Losses                      1721 non-null   int64  
 22  Winnings                    1721 non-null   int64  
 23  Ranking                     1721 non-null   int64  
dtypes: float64(12), int64(11), object(1)
memory usage: 322.8+ KB
```

In [18]:
```
df.columns.values
```

Out[18]:
```
array(['Player', 'Year', 'FirstServe', 'FirstServePointsWon',
       'FirstServeReturnPointsWon', 'SecondServePointsWon',
       'SecondServeReturnPointsWon', 'Aces', 'BreakPointsConverted
',
       'BreakPointsFaced', 'BreakPointsOpportunities', 'BreakPoint
sSaved',
       'DoubleFaults', 'ReturnGamesPlayed', 'ReturnGamesWon',
       'ReturnPointsWon', 'ServiceGamesPlayed', 'ServiceGamesWon',
       'TotalPointsWon', 'TotalServicePointsWon', 'Wins', 'Losses'
,
       'Winnings', 'Ranking'], dtype=object)
```

In [19]:
```
list(df.columns.values)
```

Out[19]:
```
['Player',
 'Year',
 'FirstServe',
 'FirstServePointsWon',
 'FirstServeReturnPointsWon',
 'SecondServePointsWon',
 'SecondServeReturnPointsWon',
 'Aces',
 'BreakPointsConverted',
 'BreakPointsFaced',
 'BreakPointsOpportunities',
 'BreakPointsSaved',
 'DoubleFaults',
 'ReturnGamesPlayed',
 'ReturnGamesWon',
 'ReturnPointsWon',
 'ServiceGamesPlayed',
 'ServiceGamesWon',
 'TotalPointsWon',
 'TotalServicePointsWon',
 'Wins',
 'Losses',
 'Winnings',
 'Ranking']
```

In [20]:
```
cl=list(df.columns.values)
```

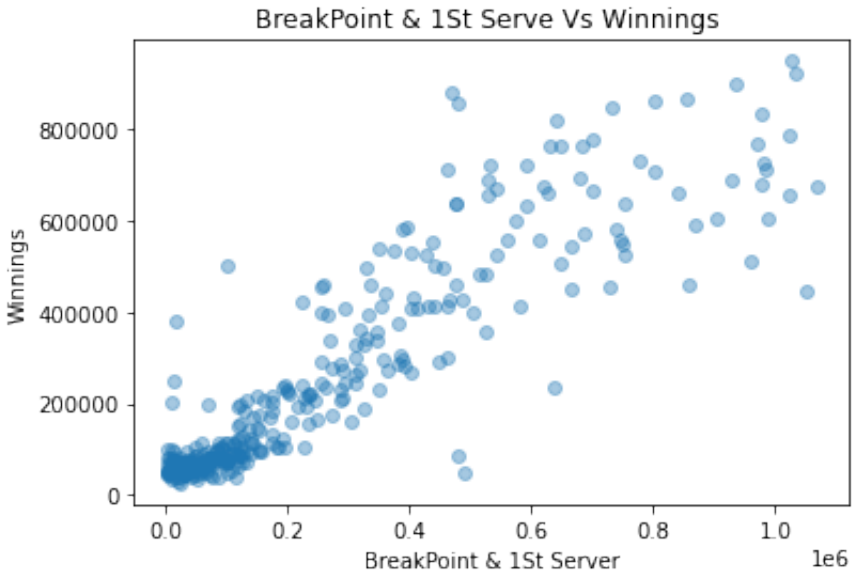
In [21]:
```
cl[0]
```

Out[21]:
```
'Player'
```

In [43]:
```
for i in cl:
    if i == 'Player':
        continue
    else:
        x1 = df[[i]]
        y1 = df[['Winnings']]
        x_train, x_test, y_train, y_test = train_test_split(x1, y1,

        mlr = LinearRegression()
        mlr.fit(x_train, y_train)
        y_predict = mlr.predict(x_test)
        plt.scatter(y_test, y_predict, alpha = 0.3 )
        plt.xlabel("Actual Winnings")
        plt.ylabel("Predicted Winnings")
        plt.title(f"{i} is X ")
        plt.show()
        k1 = mlr.score(x_train, y_train)
        print(f"The efficiency for {i} is :", k1)
```

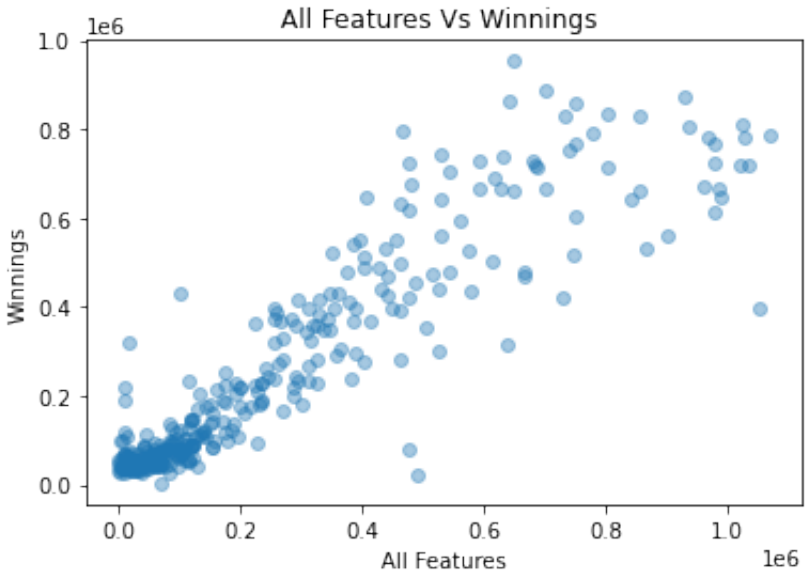
The efficiency for Year is : 0.0023552044926662408

In [35]:
```
features = df[['BreakPointsOpportunities',
'FirstServeReturnPointsWon']]
outcome = df[['Winnings']]
x_train, x_test, y_train, y_test = train_test_split(features, outco

mlr = LinearRegression()
mlr.fit(x_train, y_train)
y_predict = mlr.predict(x_test)
plt.scatter(y_test, y_predict , alpha=0.4)
plt.xlabel("BreakPoint & 1St Server")
plt.ylabel("Winnings")
plt.title("BreakPoint & 1St Serve Vs Winnings ")
plt.show()
k2 = mlr.score(x_train, y_train)
print("The efficiency for This is :", k2)
```




```
The efficiency for This is : 0.8130656716515741
```

In [37]:
```
features = df[['FirstServe','FirstServePointsWon','FirstServeReturn
'SecondServePointsWon','SecondServeReturnPointsWon','Aces',
'BreakPointsConverted','BreakPointsFaced','BreakPointsOpportunities
'BreakPointsSaved','DoubleFaults','ReturnGamesPlayed','ReturnGamesW
'ReturnPointsWon','ServiceGamesPlayed','ServiceGamesWon','TotalPoin
'TotalServicePointsWon']]
outcome = df[['Winnings']]
x_train, x_test, y_train, y_test = train_test_split(features, outco

mlr = LinearRegression()
mlr.fit(x_train, y_train)
y_predict = mlr.predict(x_test)
plt.scatter(y_test, y_predict , alpha=0.4)
plt.xlabel("All Features")
plt.ylabel("Winnings")
plt.title("All Features Vs Winnings ")
plt.show()
k2 = mlr.score(x_train, y_train)
print("The efficiency for This is :", k2)
```




```
The efficiency for This is : 0.8446529121626147
```

In [38]:
```
#  Simple Understanding
```

```
1) Load tennis_stats csv as dataframe ,24 Columns and 1000+ Rows .
2) Winnings is selected as Y (Prediction) , X can be anything - single column or multiple columns
3) use train_test_split to split the data in train and test data(both for x & y) .
4) Use LinearRegression fit method to train the model
5) use plt.scatter to plot the y_test vs y_predicted
6) for loop was written to create model using all the columns one at the time & Efficiency is printed
7) All features were used and trained the model and efficenty is found .
```

In [46]:
```
import numpy as np

sample_x = np.linspace(-16.65, 33.35, 300).reshape(-1,1)
```

In [47]:
```
sample_x
```

Out[47]:
```
array([[-16.65      ],
       [-16.48277592],
       [-16.31555184],
       [-16.14832776],
       [-15.98110368],
       [-15.8138796 ],
       [-15.64665552],
       [-15.47943144],
       [-15.31220736],
       [-15.14498328],
       [-14.9777592 ],
       [-14.81053512],
       [-14.64331104],
       [-14.47608696],
       [-14.30886288],
       [-14.1416388 ],
       [-13.97441472],
       [-13.80719064],
       [-13.63996656],
       [-13.47274247]
```

In [55]:
```
ls
```

```
Untitled.ipynb              heights.csv
__pycache__/                honeyproduction.csv
cars.csv                    reviews.csv
cars1.csv                   starbucks.csv
grades.csv                  tennis_stats.csv
gradient_descent_funcs.py
```

In [12]:
```
import pandas as pd
codecademyU = pd.read_csv('grades.csv')

# Separate out X and y
X = codecademyU[['hours_studied', 'practice_test']]
y = codecademyU['passed_exam']
#y = codecademyU.passed_exam

# Transform X
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
#print(X)
#The above scaler was done to rescale the attributes so that they h

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create and fit the logistic regression model here:
from sklearn.linear_model import LogisticRegression
cc_lr = LogisticRegression()
cc_lr.fit(X_train, y_train)
print(cc_lr.coef_)
print(cc_lr.intercept_)
```

```
[[1.5100409  0.12002228]]
[-0.13173123]
```

```
Both coefficients are positive, which makes sense: we expect
students who study more and earn higher grades on the practice
test to be more likely to pass the final exam. The coefficient on
hours_studied is larger than the coefficient on practice_test,
suggesting that hours_studied is more strongly associated with
students' probability of passing.
```

In [1]:
```
pwd
```

Out[1]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

In [2]:
```
ls
```

```
Untitled.ipynb              heights.csv
__pycache__/                honeyproduction.csv
cars.csv                    reviews.csv
cars1.csv                   starbucks.csv
grades.csv                  tennis_stats.csv
gradient_descent_funcs.py
```

In [ ]:

In [4]:
```
cat grades.csv
```

```
hours_studied,practice_test,passed_exam
0,55,0
1,75,0
2,32,0
3,80,0
4,75,0
5,95,0
6,83,0
7,87,0
8,78,0
9,85,1
10,77,1
11,89,0
12,96,0
13,83,1
14,98,1
15,87,1
16,90,1
17,92,1
18,92,1
19,100,1
```

In [14]:
```
# Import pandas and the data
import pandas as pd
codecademyU = pd.read_csv('grades.csv')

# Separate out X and y
X = codecademyU[['hours_studied', 'practice_test']]
y = codecademyU.passed_exam

# Transform X
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create and fit the logistic regression model here:
from sklearn.linear_model import LogisticRegression
cc_lr = LogisticRegression()
cc_lr.fit(X_train,y_train)
print(cc_lr.predict(X_test))
print(cc_lr.predict_proba(X_test))
print(y_test)
```

```
[0 1 0 1 1]
[[0.67934073 0.32065927]
 [0.2068119  0.7931881 ]
 [0.94452517 0.05547483]
 [0.42252072 0.57747928]
 [0.12929566 0.87070434]]
7     0
15    1
0     0
11    0
17    1
Name: passed_exam, dtype: int64
```

In [15]:
```
# Pick an alternative threshold here:
alternative_threshold = 0.6
# note: any value between 0.577 and 0.793 will work

# Import pandas and the data
import pandas as pd
codecademyU = pd.read_csv('grades.csv')

# Separate out X and y
X = codecademyU[['hours_studied', 'practice_test']]
y = codecademyU.passed_exam

# Transform X
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create and fit the logistic regression model here:
from sklearn.linear_model import LogisticRegression
cc_lr = LogisticRegression()
cc_lr.fit(X_train,y_train)

# Print out the predicted outcomes for the test data
print(cc_lr.predict(X_test))

# Print out the predicted probabilities for the test data
print(cc_lr.predict_proba(X_test)[:,1])

# Print out the true outcomes for the test data
print(y_test)
```

```
[0 1 0 1 1]
[0.32065927 0.7931881  0.05547483 0.57747928 0.87070434]
7     0
15    1
0     0
11    0
17    1
Name: passed_exam, dtype: int64
```

In [16]:
```
#Confusion matrix
```

In [17]:
```
y_true = [0, 0, 1, 1, 1, 0, 0, 1, 0, 1]
y_pred = [0, 1, 1, 0, 1, 0, 1, 1, 0, 1]

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_true, y_pred))
```

```
[[3 2]
 [1 4]]
```

This output tells us that there are 3 true negatives, 1 false negative, 4 true positives, and 2 false positives. Ideally, we want the numbers on the main diagonal (in this case, 3 and 4, which are the true negatives and true positives, respectively) to be as large as possible.

```
# Import pandas and the data
import pandas as pd
codecademyU = pd.read_csv('grades.csv')

# Separate out X and y
X = codecademyU[['hours_studied', 'practice_test']]
y = codecademyU.passed_exam

# Transform X
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state = 27)

# Create and fit the logistic regression model here:
from sklearn.linear_model import LogisticRegression
cc_lr = LogisticRegression()
cc_lr.fit(X_train,y_train)

# Save and print the predicted outcomes
y_pred = cc_lr.predict(X_test)
print('predicted classes: ', y_pred)

# Print out the true outcomes for the test data
print('true classes: ', y_test)

# Print out the confusion matrix here
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

In [21]:
```
#0 1 0 1 1
#0 1 0 0 1
#True Negative False Positive
#False Negative True Positive
```

```
TN FP FN TP
Let us consider a task to classify whether a person is pregnant or
not pregnant

Evaluating correctly(True) a pregnant person(+ve) – True
Positive(TP)
Evaluating correctly (True) a non Pregnant person(–ve) – True
Negative (TN)
Evaluating wrongly (False) a pregnant person as not pregnant(–ve)
– False Negative (FN)
Evaluating wrongly (False) a non pregnant person as pregnant(+ve)
– False Positive (FP)
```

Accuracy = (TN+TP)/(TN+TP+FN+FP)
So in the previous example
Accuracy = 4/5 = 80%

Why Accuracy is not the only thing one should consider

There are 90 people who are healthy (negative) and 10 people who have some disease (positive). Now let's say our machine learning model perfectly classified the 90 people as healthy but it also classified the unhealthy people as healthy. What will happen in this scenario? Let us see the confusion matrix and find out the accuracy?

Accuracy = (90+0)/(90+0+10+0) = 90% -- even though score is high , it predicted all wrong for Some disease . accuracy is not a good metric when the data set is unbalanced.

Next Metric we can caluculate is Precision

precision = TP/(TP+FP)
0/0 - not good

Now we will introduce another important metric called recall. Recall is also known as sensitivity or true positive rate and is defined as follows:

recall = TP/(TP+FN)

Recall should ideally be 1 (high) for a good classifier. Recall becomes 1 only when the numerator and denominator are equal i.e TP = TP +FN, this also means FN is zero

So ideally in a good classifier, we want both precision and recall to be one which also means FP and FN are zero. Therefore we need a metric that takes into account both precision and recall. F1-score is a metric which takes into account both precision and recall and is defined as follows:

F1 Score = 2*(precision*recall/(precsision+recall))

In [24]:
```
# Import pandas and the data
import pandas as pd
codecademyU = pd.read_csv('grades.csv')

# Separate out X and y
X = codecademyU[['hours_studied', 'practice_test']]
y = codecademyU.passed_exam
```

```
y = students.passed_exam

# Transform X
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create and fit the logistic regression model here:
from sklearn.linear_model import LogisticRegression
cc_lr = LogisticRegression()
cc_lr.fit(X_train,y_train)

# Save and print the predicted outcomes
y_pred = cc_lr.predict(X_test)
print('predicted classes: ', y_pred)

# Print out the true outcomes for the test data
print('true classes: ', y_test)

# Print out the confusion matrix
from sklearn.metrics import confusion_matrix
print('confusion matrix: ')
print(confusion_matrix(y_test, y_pred))

# Print accuracy here:
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

# Print F1 score here:
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred))
```

```
predicted classes:  [0 1 1 1 0]
true classes:  7     0
11    0
15    1
17    1
4     0
Name: passed_exam, dtype: int64
confusion matrix:
[[2 1]
 [0 2]]
0.8
0.8
```

In [25]:
```
# in the above by changing the random state , diff training data wi
```

In [27]:
```
import seaborn
```

```
import pandas as pd
import numpy as np
#import codecademylib3
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
transactions = pd.read_csv('transactions.csv')
print(transactions.head(5))
transactions.info()
print(transactions['amount'].describe())
transactions['isPayment'] = 0
transactions['isPayment'][transactions['type'].isin(['PAYMENT','DEB
transactions['isMovement'] = 0
transactions['isMovement'][transactions['type'].isin(['CASH_OUT','T

transactions['accountDiff'] = abs(transactions['oldbalanceDest'] -
print(transactions.head(5))

features = transactions[['amount','isPayment','isMovement','account
label = transactions['isFraud']

X_train, X_test, y_train, y_test = train_test_split(features, label

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)



model = LogisticRegression()
model.fit(X_train, y_train)
print(model.score(X_train, y_train))
print(model.score(X_test, y_test))
print(model.coef_)
#plt.show()
# Load the data


# Summary statistics on amount column


# Create isPayment field


# Create isMovement field


# Create accountDiff field


# Create features and label variables
```

```
# Split dataset


# Normalize the features variables


# Fit the model to the training data


# Score the model on the training data


# Score the model on the test data


# Print the model coefficients


# New transaction data
transaction1 = np.array([123456.78, 0.0, 1.0, 54670.1])
transaction2 = np.array([98765.43, 1.0, 0.0, 8524.75])
transaction3 = np.array([543678.31, 1.0, 0.0, 510025.5])
your_transaction = np.array([692541.63, 0.0, 1.0, 23670.1])

# Create a new transaction


# Combine new transactions into a single array

# Combine new transactions into a single array
sample_transactions = np.stack((transaction1,transaction2,transacti
# Normalize the new transactions
sample_transactions = scaler.transform(sample_transactions)

# Predict fraud on the new transactions
print("Good")
# Predict fraud on the new transactions
print(model.predict(sample_transactions))

# Show probabilities on the new transactions
print(model.predict_proba(sample_transactions))
# Show probabilities on the new transactions
```

```
   step      type     amount     nameOrig  oldbalanceOrg  newbalan
ceOrig  \
0     8  CASH_OUT  158007.12   C424875646           0.00
0.00
1   236  CASH_OUT  457948.30  C1342616552           0.00
0.00
2    37   CASH_IN  153602.99   C900876541    11160428.67     11314
031.67
3   331  CASH_OUT   49555.14   C177696810       10865.00
0.00
4   250  CASH OUT   29648.02   C788941490           0.00
```

```
0.00

      nameDest  oldbalanceDest  newbalanceDest  isFraud
0  C1298177219       474016.32      1618631.97        0
1  C1323169990      2720411.37      3178359.67        0
2   C608741097      3274930.56      3121327.56        0
3   C462716348            0.00        49555.14        0
4  C1971700992        56933.09        86581.10        0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199999 entries, 0 to 199998
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            199999 non-null  int64
 1   type            199999 non-null  object
 2   amount          199999 non-null  float64
 3   nameOrig        199999 non-null  object
 4   oldbalanceOrg   199999 non-null  float64
 5   newbalanceOrig  199999 non-null  float64
 6   nameDest        199999 non-null  object
 7   oldbalanceDest  199999 non-null  float64
 8   newbalanceDest  199999 non-null  float64
 9   isFraud         199999 non-null  int64
dtypes: float64(5), int64(2), object(3)
memory usage: 15.3+ MB
count    1.999990e+05
mean     1.802425e+05
std      6.255482e+05
min      0.000000e+00
25%      1.338746e+04
50%      7.426695e+04
75%      2.086376e+05
max      5.204280e+07
Name: amount, dtype: float64

/var/folders/v_/dt4zl16x3zbd4h2zk9d9rbs00000gn/T/ipykernel_30762/1
344531732.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  transactions['isPayment'][transactions['type'].isin(['PAYMENT','
DEBIT'])] = 1
/var/folders/v_/dt4zl16x3zbd4h2zk9d9rbs00000gn/T/ipykernel_30762/1
344531732.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  transactions['isMovement'][transactions['type'].isin(['CASH_OUT'
```

```
,'TRANSFER'])] = 1
```

```
   step      type     amount     nameOrig  oldbalanceOrg  newbalan
ceOrig  \
0     8  CASH_OUT  158007.12   C424875646           0.00
0.00
1   236  CASH_OUT  457948.30  C1342616552           0.00
0.00
2    37   CASH_IN  153602.99   C900876541    11160428.67     11314
031.67
3   331  CASH_OUT   49555.14   C177696810       10865.00
0.00
4   250  CASH_OUT   29648.02   C788941490           0.00
0.00

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isPayment
\
0  C1298177219       474016.32      1618631.97        0          0
1  C1323169990      2720411.37      3178359.67        0          0
2   C608741097      3274930.56      3121327.56        0          0
3   C462716348            0.00        49555.14        0          0
4  C1971700992        56933.09        86581.10        0          0

   isMovement  accountDiff
0           1    474016.32
1           1   2720411.37
2           0   7885498.11
3           1     10865.00
4           1     56933.09
0.998592847091765
0.9985166666666667
[[ 0.21600648 -0.73323334  2.2612512  -0.60009074]]
Good
[0 0 0 0]
[[9.96641491e-01 3.35850903e-03]
 [9.99992473e-01 7.52659180e-06]
 [9.99991839e-01 8.16133989e-06]
 [9.95874314e-01 4.12568587e-03]]

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/base.py:450: UserWarning: X does not have valid feature
names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [1]:

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
Input In [1], in <cell line: 1>()
----> 1 L

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

In [6]:

```
movie_dataset = {'Avatar': [0.01940156245995175, 0.4812286689419795
```

The above one is already normalized data of budget , length, year of release

In [7]:

```
movie_dataset['My Date with Drew']
```

Out[7]:

```
[7.22033494937031e-08, 0.18088737201365188, 0.8651685393258427]
```

In [12]:

```
def distance(movie1, movie2):
  squared_difference = 0
  for i in range(len(movie1)):
    squared_difference += (movie1[i] - movie2[i]) ** 2
  final_distance = squared_difference ** 0.5
  return final_distance

def classify(unknown, dataset, k):
  distances = []
  for i in dataset:
    distance_to_point = distance(dataset[i],unknown)
    distances.append([distance_to_point,i])

  distances.sort()
  neighbors = distances[0:k]

  return neighbors

print(classify([.4, .2, .9], movie_dataset, 5))
```

```
[[0.08273614694606074, 'Lady Vengeance'], [0.22989623153818367, 'S
teamboy'], [0.23641372358159884, 'Fateless'], [0.26735445689589943
, 'Princess Mononoke'], [0.3311022951533416, 'Godzilla 2000']]
```

[[0.08273614694606074, 'Lady Vengeance'], [0.22989623153818367, 'Steamboy'], [0.23641372358159884, 'Fateless'], [0.26735445689589943, 'Princess Mononoke'], [0.3311022951533416, 'Godzilla 2000']]

In [21]:
```
labels = {'Avatar': 1, "Pirates of the Caribbean: At World's End":
```

In [29]:
```
def normalize_point(lst):
  minimum = min(lst)
  maximum = max(lst)
  normalized = []
  for i in lst:
    val = (i-minimum)/(maximum-minimum)
    normalized.append(val)

  return normalized

# This one is min & max normalization , not a great one , other way
```

In [27]:
```
#from movies import movie_dataset, movie_labels
#print(movie_labels)

def distance(movie1, movie2):
  squared_difference = 0
  for i in range(len(movie1)):
    squared_difference += (movie1[i] - movie2[i]) ** 2
  final_distance = squared_difference ** 0.5
  return final_distance

def classify(unknown, dataset, labels, k):
  distances = []
  num_good = 0
  num_bad = 0
  #Looping through all points in the dataset
  for title in dataset:
    movie = dataset[title]
    distance_to_point = distance(movie, unknown)
    #Adding the distance and point associated with that distance
    distances.append([distance_to_point, title])
  distances.sort()
  #Taking only the k closest points
  neighbors = distances[0:k]
  print(neighbors)

  for i in neighbors:
    title = i[1]
    if labels[title] == 0:
      num_bad += 1
    else:
      num_good += 1
    if num_good > num_bad:
      return 1
    else:
      return 0
```

In [24]:
```
'Avatar' in movie_dataset
```

Out[24]:
```
True
```

In [28]:
```
print("Call Me By Your Name" in movie_dataset)
my_movie = [3500000, 132, 2017]
normalized_my_movie = normalize_point(my_movie)
print(normalized_my_movie)
print(classify(normalized_my_movie, movie_dataset, labels, 5))
```

```
False
[1.0, 0.0, 0.0005385917411742386]
[[0.9214248817953455, 'The Host'], [1.0151033469326625, 'Top Hat']
, [1.0177906839705453, '42nd Street'], [1.0182457782046241, 'Snow
White and the Seven Dwarfs'], [1.019309914817951, 'Modern Times']]
1
```

```
The Three steps are

Normalize the data
Find the k nearest neighbors
Classify the new point based on those neighbors
```

```
K-Nearest Neighbors Classifier

What it is ?
K-Nearest Neighbors (KNN) is a classification algorithm. The
central idea is that data points with similar attributes tend to
fall into similar categories.

Distance :-

One Can find the distance between 2 points using the below formula
((A1-B1)^2+(A2-B2)^2+....+(An-Bn)^2)^0.5
where A1,A2... are features of 1st point
B1,B2... are features of 2nd point

Normalize the data :-

Not all features are of same scale . for example length of 2
movies can be 180 & 160 (a diff of 20) ... budget of 2 movies can
be 1000000 & 1500000 (a diff of 500k) , distance formula treats
both in same scale which is absurd ...so we need to normalize the
data ..there are multiple way to normailize like (min-max ,Z-Score
Normalization etc)

Find the k nearest neighbors:-

Once normalized , find the 5 nearest neighbour (k=5 in this case)
.. find the distance of all points and sort it to find the nearest
5 points

Classify the new point based on those neighbors :-

Ex:- now compare the 5 points (if 3 of them are red , then our
data is red)
Ex:- if >3 points are good movie , our movie also will be good
```

In [30]:
```
from sklearn.datasets import load_breast_cancer

breast_cancer_data = load_breast_cancer()
```

In [32]:
```
#print(breast_cancer_data)
```

In [35]:
```
breast_cancer_data.data[0]
```

Out[35]:
```
array([1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.77
6e-01,
       3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.05
3e-01,
       8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.58
7e-02,
       3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.01
9e+03,
       1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.18
9e-01])
```

In [36]:
```
breast_cancer_data.feature_names
```

Out[36]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area
',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimen
sion',
       'radius error', 'texture error', 'perimeter error', 'area e
rror',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave poin
ts',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [37]:
```
a=3
```

In [39]:
```
type(a)
```

Out[39]:
```
int
```

In [40]:
```
type(breast_cancer_data)
```

Out[40]:
```
sklearn.utils.Bunch
```

In [41]:
```
breast_cancer_data
```

Out[41]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01,
4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-0
1,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-0
1,
```

```
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-0
1,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-0
1,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-0
1,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0
, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1
, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0
, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1
, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0
, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1
, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1
, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1
, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1
, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0
, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0
, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1
, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1
, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1
, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1
, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0
, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1
, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1
, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1
, 0, 0,
```

```
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1
, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0
, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1
, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (
diagnostic) dataset\n--------------------------------------------\
n\n**Data Set Characteristics:**\n\n    :Number of Instances: 569\
n\n    :Number of Attributes: 30 numeric, predictive attributes an
d the class\n\n    :Attribute Information:\n        - radius (mean
of distances from center to points on the perimeter)\n        - te
xture (standard deviation of gray-scale values)\n        - perimet
er\n        - area\n        - smoothness (local variation in radiu
s lengths)\n        - compactness (perimeter^2 / area - 1.0)\n
- concavity (severity of concave portions of the contour)\n
- concave points (number of concave portions of the contour)\n
- symmetry\n        - fractal dimension ("coastline approximation"
- 1)\n\n        The mean, standard error, and "worst" or largest (
mean of the three\n        worst/largest values) of these features
were computed for each image,\n        resulting in 30 features.
For instance, field 0 is Mean Radius, field\n        10 is Radius
SE, field 20 is Worst Radius.\n\n        - class:\n
- WDBC-Malignant\n                - WDBC-Benign\n\n    :Summary St
atistics:\n\n    ===================================== ====== ====
==\n                                        Min    Max\n    ===
================================== ====== ======\n    radius (mean
):                           6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                       43.79  188
.5\n    area (mean):                             143.5  2501.0\n    s
moothness (mean):                     0.053  0.163\n    compactness
(mean):                    0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                 0.0    0.2
01\n    symmetry (mean):                         0.106  0.304\n    fr
actal dimension (mean):              0.05   0.097\n    radius (stan
dard error):               0.112  2.873\n    texture (standard erro
r):              0.36   4.885\n    perimeter (standard error):
0.757  21.98\n    area (standard error):                  6.802  542
.2\n    smoothness (standard error):           0.002  0.031\n    co
mpactness (standard error):          0.002  0.135\n    concavity (s
tandard error):            0.0    0.396\n    concave points (standa
rd error):      0.0    0.053\n    symmetry (standard error):
0.008  0.079\n    fractal dimension (standard error):   0.001  0.0
3\n    radius (worst):                          7.93   36.04\n    tex
ture (worst):                       12.02  49.54\n    perimeter (wo
rst):                     50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                    0.071  0.
223\n    compactness (worst):                   0.027  1.058\n    c
```

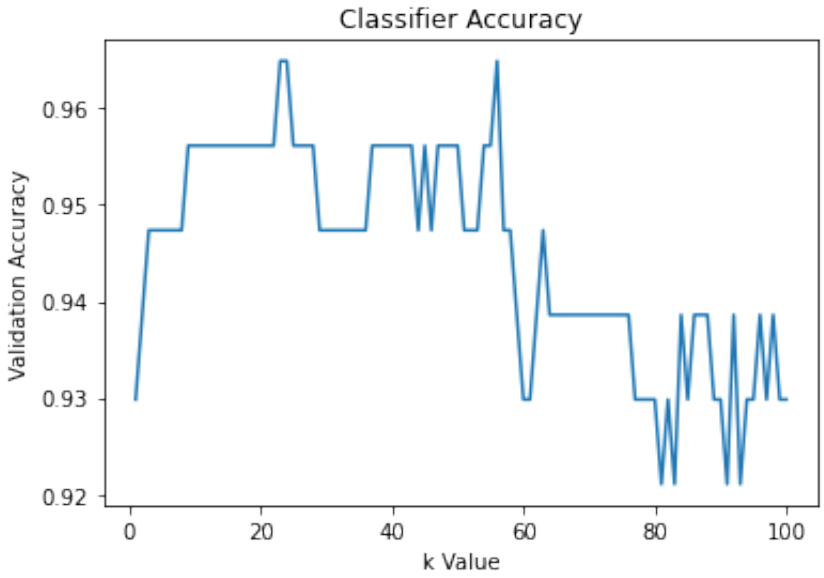
oncavity (worst):   0.0   1.252\n   concave poi
nts (worst):   0.0   0.291\n   symmetry (worst):
0.156 0.664\n   fractal dimension (worst):   0.055 0.2
08\n   ================================== ====== ======\n\n
:Missing Attribute Values: None\n\n   :Class Distribution: 212 -
Malignant, 357 - Benign\n\n   :Creator: Dr. William H. Wolberg,
W. Nick Street, Olvi L. Mangasarian\n\n   :Donor: Nick Street\n\n
:Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wi
sconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures
are computed from a digitized image of a fine needle\naspirate (FN
A) of a breast mass. They describe\ncharacteristics of the cell n
uclei present in the image.\n\nSeparating plane described above wa
s obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett,
"Decision Tree\nConstruction Via Linear Programming." Proceedings
of the 4th\nMidwest Artificial Intelligence and Cognitive Science
Society,\npp. 97-101, 1992], a classification method which uses li
near\nprogramming to construct a decision tree. Relevant features
\nwere selected using an exhaustive search in the space of 1-4\nfe
atures and 1-3 separating planes.\n\nThe actual linear program use
d to obtain the separating plane\nin the 3-dimensional space is th
at described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Li
near\nProgramming Discrimination of Two Linearly Inseparable Sets"
,\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis data
base is also available through the UW CS ftp server:\n\nftp ftp.cs
.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topi
c:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasar
ian. Nuclear feature extraction \n   for breast tumor diagnosis.
IS&T/SPIE 1993 International Symposium on \n   Electronic Imagin
g: Science and Technology, volume 1905, pages 861-870,\n   San J
ose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolber
g. Breast cancer diagnosis and \n   prognosis via linear program
ming. Operations Research, 43(4), pages 570-577, \n   July-Augus
t 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Mac
hine learning techniques\n   to diagnose breast cancer from fine
-needle aspirates. Cancer Letters 77 (1994) \n   163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean peri
meter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dime
nsion',
       'radius error', 'texture error', 'perimeter error', 'area
error',
       'smoothness error', 'compactness error', 'concavity error'
,
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture'
,
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave poi
nts',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'
),
 'filename': 'breast_cancer.csv',

```
 'data_module': 'sklearn.datasets.data'}
```

In [42]:
```
#import codecademylib3_seaborn
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt


breast_cancer_data = load_breast_cancer()
#print(type(breast_cancer_data))
breast_cancer_data.data[0]
print(breast_cancer_data.feature_names)
print(breast_cancer_data.target)
print(breast_cancer_data.target_names)

training_data, validation_data, training_labels, validation_labels
print(len(training_data))
print(len(training_labels))
class_score = []
for k in range(1,101):
  classifier = KNeighborsClassifier(n_neighbors = k)
  classifier.fit(training_data, training_labels)
  class_score.append(classifier.score(validation_data, validation_l
x = range(1,101)
plt.plot(x, class_score)
plt.title('Classifier Accuracy')
plt.xlabel('k Value')
plt.ylabel('Validation Accuracy')
plt.show()

#print(class_score)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension'
]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0
 1 0
```

```
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1
0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0
0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1
1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0
0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1
1 1 1 1
 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1
1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 0 1
 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1
0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1
 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
['malignant' 'benign']
455
455
```



In [43]:
```
#K-Nearest Neighbors
```

In [44]:
```
pwd
```

Out[44]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

In [45]:
```
from movies import movie_dataset, movie_ratings

def distance(movie1, movie2):
  squared_difference = 0
  for i in range(len(movie1)):
    squared_difference += (movie1[i] - movie2[i]) ** 2
  final_distance = squared_difference ** 0.5
  return final_distance

def predict(unknown, dataset, movie_ratings, k):
  distances = []
  #Looping through all points in the dataset
  for title in dataset:
    movie = dataset[title]
    distance_to_point = distance(movie, unknown)
    #Adding the distance and point associated with that distance
    distances.append([distance_to_point, title])
  distances.sort()
  #Taking only the k closest points
  neighbors = distances[0:k]
  total = 0

  for neighbor in neighbors:
    title = neighbor[1]
    total += movie_ratings[title]
  return (total/len(neighbors))

print(movie_dataset["Life of Pi"])
print(movie_ratings["Life of Pi"])

print(predict([0.016, 0.300, 1.022], movie_dataset, movie_ratings,
```

```
---------------------------------------------------------------------
---------
UnpicklingError                           Traceback (most recent c
all last)
Input In [45], in <cell line: 1>()
----> 1 from movies import movie_dataset, movie_ratings
      3 def distance(movie1, movie2):
      4   squared_difference = 0

File ~/Documents/codecdemy/ML/movies.py:3, in <module>
      1 import pickle
----> 3 movie_dataset = pickle.load( open( "movie_regression_datas
et.p", "rb" ) )
      4 movie_ratings = pickle.load( open( "movie_regression_label
s.p", "rb" ) )

UnpicklingError: invalid load key, '{'.
```

In [46]:
```
#Above is serializing pickle error
```

In [48]:
```
a=[0.016, 0.300, 1.022]
```

In [53]:
```
k=np.array(a)
```

In [56]:
```
ls
```

```
Untitled.ipynb              movie_regression_dataset.p
__pycache__/                movie_regression_labels.p
cars.csv                    movie_value_list.csv
cars1.csv                   movies.py
grades.csv                  reviews.csv
gradient_descent_funcs.py   starbucks.csv
heights.csv                 tennis_stats.csv
honeyproduction.csv         transactions.csv
movie_list.csv
```

In [61]:
```
from sklearn.neighbors import KNeighborsRegressor
import numpy as np
import pandas as pd
```

In [74]:
```
regressor = KNeighborsRegressor(n_neighbors = 5, weights = "distanc
movie_dataset = pd.read_csv('movie_list.csv')

movie_ratings = pd.read_csv('movie_value_list.csv')
#movie_dataset.info()
movie_dataset = movie_dataset.drop('0', axis=1)
movie_dataset.info()
movie_ratings = movie_ratings.drop('0', axis=1)
movie_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3654 entries, 0 to 3653
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   0.01940156245995175  3654 non-null   float64
 1   0.4812286689419795   3654 non-null   float64
 2   0.9213483146067416   3654 non-null   float64
dtypes: float64(3)
memory usage: 85.8 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3654 entries, 0 to 3653
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   7.9     3654 non-null   float64
dtypes: float64(1)
memory usage: 28.7 KB
```

In [75]:
```
regressor.fit(movie_dataset, movie_ratings)
```

Out[75]:
```
KNeighborsRegressor(weights='distance')
```

In [ ]:
```

```

In [76]:
```
print(regressor.predict(np.array([0.016, 0.300, 1.022]).reshape(1,-
```

```
[[6.84913968]]

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/base.py:450: UserWarning: X does not have valid feature
names, but KNeighborsRegressor was fitted with feature names
  warnings.warn(
```

In [79]:
```
#Decision Tree

```

In [80]:
```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-d
```

In [81]:
```
print(df.head())
```

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

In [82]:
```
df.iloc[:,0:6]
```

Out[82]:

	buying	maint	doors	persons	lug_boot	safety
0	vhigh	vhigh	2	2	small	low
1	vhigh	vhigh	2	2	small	med
2	vhigh	vhigh	2	2	small	high
3	vhigh	vhigh	2	2	med	low
4	vhigh	vhigh	2	2	med	med
...	...	...	...	...	...	...
1723	low	low	5more	more	med	med
1724	low	low	5more	more	med	high
1725	low	low	5more	more	big	low
1726	low	low	5more	more	big	med
1727	low	low	5more	more	big	high

1728 rows × 6 columns

In [87]:
```
(df['accep']=='unacc')
```

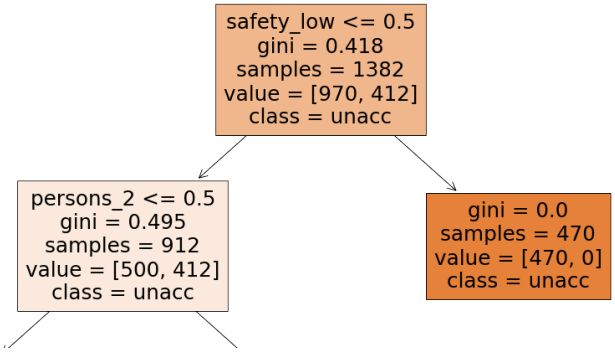
Out[87]:
```
0        True
1        True
2        True
3        True
4        True
        ...
1723    False
1724    False
1725     True
1726    False
1727    False
Name: accep, Length: 1728, dtype: bool
```

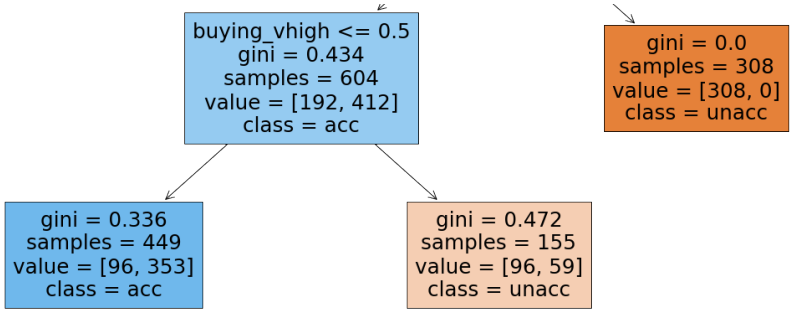
In [89]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


#Import models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

#Loading the dataset
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-d

## 1a. Take a look at the dataset
```

```
## 1a. Take a look at the dataset
print(df.head())

## 1b. Setting the target and predictor variables
df['accep'] = ~(df['accep']=='unacc') #1 is acceptable, 0 if not ac
X = pd.get_dummies(df.iloc[:,0:6])
print(X.head())
y = df['accep']
print(y.head())

## 1c. Examine the new features
print(X.columns)
print(len(X.columns))

## 2a. Performing the train-test split
x_train, x_test, y_train, y_test = train_test_split(X,y, random_sta

## 2b.Fitting the decision tree classifier
dt = DecisionTreeClassifier(max_depth=3, ccp_alpha=0.01,criterion='
dt.fit(x_train, y_train)

## 3.Plotting the Tree
plt.figure(figsize=(20,12))
tree.plot_tree(dt, feature_names = x_train.columns, max_depth=5, cl
plt.tight_layout()
plt.show()
```

```
  buying  maint doors persons lug_boot safety  accep
0  vhigh  vhigh     2       2    small    low  unacc
1  vhigh  vhigh     2       2    small    med  unacc
2  vhigh  vhigh     2       2    small   high  unacc
3  vhigh  vhigh     2       2      med    low  unacc
4  vhigh  vhigh     2       2      med    med  unacc
   buying_high  buying_low  buying_med  buying_vhigh  maint_high
maint_low  \
0            0           0           0             1           0
0
1            0           0           0             1           0
0
2            0           0           0             1           0
0
3            0           0           0             1           0
0
4            0           0           0             1           0
0

   maint_med  maint_vhigh  doors_2  doors_3  ...  doors_5more  per
sons_2  \
0          0            1        1        0  ...            0
1
1          0            1        1        0  ...            0
1
2          0            1        1        0  ...            0
1
3          0            1        1        0  ...            0
```

```
1
4          0            1        1        0  ...            0
1

   persons_4  persons_more  lug_boot_big  lug_boot_med  lug_boot_s
mall  \
0          0             0             0             0
1
1          0             0             0             0
1
2          0             0             0             0
1
3          0             0             0             1
0
4          0             0             0             1
0

   safety_high  safety_low  safety_med
0            0           1           0
1            0           0           1
2            1           0           0
3            0           1           0
4            0           0           1

[5 rows x 21 columns]
0    False
1    False
2    False
3    False
4    False
Name: accep, dtype: bool
Index(['buying_high', 'buying_low', 'buying_med', 'buying_vhigh',
'maint_high',
       'maint_low', 'maint_med', 'maint_vhigh', 'doors_2', 'doors_
3',
       'doors_4', 'doors_5more', 'persons_2', 'persons_4', 'person
s_more',
       'lug_boot_big', 'lug_boot_med', 'lug_boot_small', 'safety_h
igh',

       'safety_low', 'safety_med'],
      dtype='object')
21
```





In [ ]:

In [90]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-d
df['accep'] = ~(df['accep']=='unacc') #1 is acceptable, 0 if not ac
X = pd.get_dummies(df.iloc[:,0:6])
y = df['accep']

x_train, x_test, y_train, y_test = train_test_split(X,y, random_sta

## Functions to calculate gini impurity and information gain

def gini(data):
    """calculate the Gini Impurity
    """
    data = pd.Series(data)
    return 1 - sum(data.value_counts(normalize=True)**2)

def info_gain(left, right, current_impurity):
    """Information Gain associated with creating a node/split data.
    Input: left, right are data in left branch, right banch, respec
    current_impurity is the data impurity before splitting into lef
    """
    # weight for gini score of the left branch
    w = float(len(left)) / (len(left) + len(right))
    return current_impurity - w * gini(left) - (1 - w) * gini(right

## 1. Calculate gini and info gain for a root node split at safety_
y_train_sub = y_train[x_train['safety_low']==0]
x_train_sub = x_train[x_train['safety_low']==0]

gi = gini(y_train_sub)
print(f'Gini impurity at root: {gi}')

## 2. Information gain when using feature `persons_2`
left = y_train[x_train['persons_2']==0]
right = y_train[x_train['persons_2']==1]

print(f'Information gain for persons_2: {info_gain(left, right, gi)

## 3. Which feature split maximizes information gain?
```

```
## 3. Which feature split maximizes information gain?

info_gain_list = []
for i in x_train.columns:
     left = y_train_sub[x_train_sub[i]==0]
     right = y_train_sub[x_train_sub[i]==1]
     info_gain_list.append([i, info_gain(left, right, gi)])

info_gain_table = pd.DataFrame(info_gain_list).sort_values(1,ascend
print(f'Greatest impurity gain at:{info_gain_table.iloc[0,:]}')
print(info_gain_table)
```

```
Gini impurity at root: 0.49534472145275465
Information gain for persons_2: 0.16699155320608106
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
                  0         1
12       persons_2  0.208137
14    persons_more  0.056305
13       persons_4  0.048902
7       maint_vhigh  0.027772
3     buying_vhigh  0.025267
17  lug_boot_small  0.015210
1        buying_low  0.014392
5         maint_low  0.009816
15    lug_boot_big  0.009160
6         maint_med  0.008975
2        buying_med  0.008964
20       safety_med  0.006929
18     safety_high  0.006929
8           doors_2  0.004097
0      buying_high  0.002943
4        maint_high  0.001084
16    lug_boot_med  0.000759
11     doors_5more  0.000656
10          doors_4  0.000386
9           doors_3  0.000327
19       safety_low  0.000000
```

In [ ]:
```
# not understood 100% ... need to spend sometime more
```

In [ ]:
```

```

In [92]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

def gini(data):
```

```
    """calculate the Gini impurity
    """
    data = pd.Series(data)
    return 1 - sum(data.value_counts(normalize=True)**2)

def info_gain(left, right, current_impurity):
    """Information Gain associated with creating a node/split data.
    Input: left, right are data in left branch, right banch, respec
    current_impurity is the data impurity before splitting into lef
    """
    # weight for gini score of the left branch
    w = float(len(left)) / (len(left) + len(right))
    return current_impurity - w * gini(left) - (1 - w) * gini(right

df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-d
df['accep'] = ~(df['accep']=='unacc') #1 is acceptable, 0 if not ac
X = pd.get_dummies(df.iloc[:,0:6])
y = df['accep']

x_train, x_test, y_train, y_test = train_test_split(X,y, random_sta

y_train_sub = y_train[x_train['safety_low']==0]
x_train_sub = x_train[x_train['safety_low']==0]

gi = gini(y_train_sub)
print(f'Gini impurity at root: {gi}')

info_gain_list = []
for i in x_train.columns:
       left = y_train_sub[x_train_sub[i]==0]
       right = y_train_sub[x_train_sub[i]==1]
       info_gain_list.append([i, info_gain(left, right, gi)])

       info_gain_table = pd.DataFrame(info_gain_list).sort_values(1
       print(f'Greatest impurity gain at:{info_gain_table.iloc[0,:]
```

```
Gini impurity at root: 0.49534472145275465
Greatest impurity gain at:0    buying_high
1       0.002943
Name: 0, dtype: object
Greatest impurity gain at:0    buying_low
1      0.014392
Name: 1, dtype: object
Greatest impurity gain at:0    buying_low
1      0.014392
Name: 1, dtype: object
Greatest impurity gain at:0    buying_vhigh
1       0.025267
Name: 3, dtype: object
Greatest impurity gain at:0    buying_vhigh
1       0.025267
Name: 3, dtype: object
Greatest impurity gain at:0    buying_vhigh
```

```
1        0.025267
Name: 3, dtype: object
Greatest impurity gain at:0    buying_vhigh
1        0.025267
Name: 3, dtype: object
Greatest impurity gain at:0    maint_vhigh
1       0.027772
Name: 7, dtype: object
Greatest impurity gain at:0    maint_vhigh
1       0.027772
Name: 7, dtype: object
Greatest impurity gain at:0    maint_vhigh
1       0.027772
Name: 7, dtype: object
Greatest impurity gain at:0    maint_vhigh
1       0.027772
Name: 7, dtype: object
Greatest impurity gain at:0    maint_vhigh
1       0.027772
Name: 7, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2
1     0.208137
Name: 12, dtype: object
Greatest impurity gain at:0    persons_2

1     0.208137
```

In [ ]:

Gini Impurity
https://www.youtube.com/watch?v=u4IxOk2ijSs
Gini Impurity measures diversity of data , a higher means the diversity is high

Gini Index - Probability of picking 2 distinct elements

Gini Purity index (P of both diff) = 1(P of Anything) -P1^2-p2^2-...-Pn^n(P of both equal)

Ex of gini index :-

#######*** (0.42)
####***@@& (0.7)
########## (0) = 1-1^2
!@#$%^)(*& (0.9) = 1 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2 - 0.1^2

Example of Information Gain

Lets say at Node 1 (top node) -- the gini is 0.418 for sample of 1382(970,412)(acc,unacc), the split creates 2 nodes , left is node 2 & right is leaf
Node2 there is another splith with gini value of .495 on diff criteria of person_2<=0.5 .. here the sample is 912(500,412)with gini valud of 0.495
leaf node gini value is 0 ..

so now lets calculate the inforation gain

new weighter impurity = 912/1382*(.495)+470/1382*(0)=0.3267

Then the information gain (or reduction in impurity after the split) is

0.4185-0.3267=0.0918

The higher the information gain the better - if information gain is 0, then splitting the data on that feature was useless!

In [113]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

## Loading the data and setting target and predictor variables
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-d
print(df.head())
df.info()

df['accep'] = ~(df['accep']=='unacc') #1 is acceptable, 0 if not ac
X = pd.get_dummies(df.iloc[:,0:6])
print("Hellow")
print(X.head())
y = df['accep']
print(df.head())
df.info()

## Train-test split and fitting the tree
x_train, x_test, y_train, y_test = train_test_split(X,y, random_sta
dtree = DecisionTreeClassifier(max_depth=3)
dtree.fit(x_train, y_train)

## Visualizing the tree
plt.figure(figsize=(27,12))
tree.plot_tree(dtree)
plt.tight_layout()
plt.show()

## Text-based visualization of the tree (View this in the Output te
print(tree.export_text(dtree))
```

```
  buying  maint doors persons lug_boot safety  accep
0  vhigh  vhigh     2       2    small    low  unacc
1  vhigh  vhigh     2       2    small    med  unacc
2  vhigh  vhigh     2       2    small   high  unacc
3  vhigh  vhigh     2       2      med    low  unacc
4  vhigh  vhigh     2       2      med    med  unacc
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   accep     1728 non-null   object
dtypes: object(7)
```

In [103]:
```
for col in X.columns:
    print(col)
```

```
buying_high
buying_low
buying_med
buying_vhigh
maint_high
maint_low
maint_med
maint_vhigh
doors_2
doors_3
doors_4
doors_5more
persons_2
persons_4
persons_more
lug_boot_big
lug_boot_med
lug_boot_small
safety_high
safety_low
safety_med
```

In [108]:
```
X['safety_low']
```

Out[108]:
```
0       1
1       0
2       0
3       1
4       0
       ..
1723    0
1724    0
1725    1
1726    0
1727    0
Name: safety_low, Length: 1728, dtype: uint8
```

In [122]:
```
X['safety_low'].value_counts()
```

Out[122]:
```
0    1152
1     576
Name: safety_low, dtype: int64
```

In [116]:
```
X[X.columns[::20]]
```

Out[116]:

	buying_high	safety_med
0	0	0
1	0	1
2	0	0
3	0	0
4	0	1
...	...	...
1723	0	1
1724	0	0
1725	0	0
1726	0	1
1727	0	0

1728 rows × 2 columns

In [117]:
```
print(X.head(3))
```

```
   buying_high  buying_low  buying_med  buying_vhigh  maint_high
maint_low  \
0            0           0           0             1           0
0
1            0           0           0             1           0
0
2            0           0           0             1           0
0

   maint_med  maint_vhigh  doors_2  doors_3  ...  doors_5more  per
sons_2  \
0          0            1        1        0  ...            0
1
1          0            1        1        0  ...            0
1
2          0            1        1        0  ...            0
1

   persons_4  persons_more  lug_boot_big  lug_boot_med  lug_boot_s
mall  \
0          0             0             0             0
1
1          0             0             0             0
1
2          0             0             0             0
1

   safety_high  safety_low  safety_med
0            0           1           0
1            0           0           1
2            1           0           0

[3 rows x 21 columns]
```

In [121]:
```
X.iloc[:,19:20]
```

Out[121]:

	safety_low
0	1
1	0
2	0
3	1
4	0
...	...
1723	0
1724	0
1725	1
1726	0
1727	0

1728 rows × 1 columns

In [1]:
```
pwd
```

Out[1]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

In [4]:
```
import pandas as pd
df = pd.read_csv("Admission_Predict.csv")
#df.columns = df.columns.str.strip().str.replace(' ','_').str.lower
```

In [5]:
```
df.head()
```

Out[5]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

In [6]:
```
df.columns = df.columns.str.strip().str.replace(' ','_').str.lower(
```

In [7]:
```
df.head()
```

Out[7]:

	serial_no.	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	chance_of
0	1	337	118	4	4.5	4.5	9.65	1	
1	2	324	107	4	4.0	4.5	8.87	1	
2	3	316	104	3	3.0	3.5	8.00	1	
3	4	322	110	3	3.5	2.5	8.67	1	
4	5	314	103	2	2.0	3.0	8.21	0	

In [8]:
```
X = df.loc[:,'gre_score':'research']
```

In [9]:
```
X.head()
```

Out[9]:

	gre_score	toefl_score	university_rating	sop	lor	cgpa	research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

In [10]:
```
y = df['chance_of_admit']>=.8
```

In [15]:
```
y.head()
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(X,y,random_stat
dt = DecisionTreeClassifier(max_depth=2, ccp_alpha=0.01,criterion='
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
print(dt.score(x_test,y_test))
print(accuracy_score(y_test,y_pred))
```

```
0.925
0.925
```

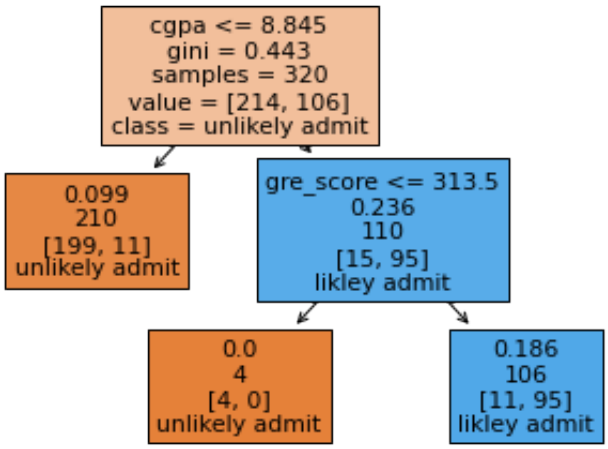
In [17]:
```
from sklearn import tree
```

In [20]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gre_score          400 non-null    int64  
 1   toefl_score        400 non-null    int64  
 2   university_rating  400 non-null    int64  
 3   sop                400 non-null    float64
 4   lor                400 non-null    float64
 5   cgpa               400 non-null    float64
 6   research           400 non-null    int64  
dtypes: float64(3), int64(4)
memory usage: 22.0 KB
```

In [19]:
```
tree.plot_tree(dt, feature_names = x_train.columns,
               max_depth=3, class_names = ['unlikely admit', 'likle
               label='root', filled=True)
print(tree.export_text(dt, feature_names = X.columns.tolist()))
```

```
|--- cgpa <= 8.85
|   |--- class: False
|--- cgpa >  8.85
|   |--- gre_score <= 313.50
|   |   |--- class: False
|   |--- gre_score >  313.50
|   |   |--- class: True
```



In [21]:
```
def gini(data):
    """Calculate the Gini Impurity Score
    """
    data = pd.Series(data)
    return 1 - sum(data.value_counts(normalize=True)**2)

gi = gini(y_train)
print(f'Gini impurity at root: {round(gi,3)}')
```

```
Gini impurity at root: 0.443
```

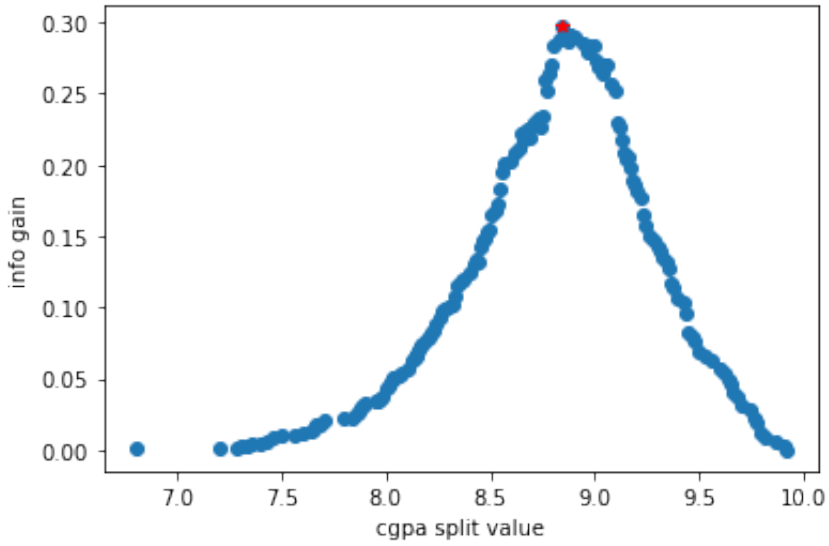
In [23]:
```
def info_gain(left, right, current_impurity):
    """Information Gain associated with creating a node/split data.
    Input: left, right are data in left branch, right banch, respec
    current_impurity is the data impurity before splitting into lef
    """
    # weight for gini score of the left branch
    w = float(len(left)) / (len(left) + len(right))
    return current_impurity - w * gini(left) - (1 - w) * gini(right

info_gain_list = []
for i in x_train.cgpa.unique():
    left = y_train[x_train.cgpa<=i]
    right = y_train[x_train.cgpa>i]
    info_gain_list.append([i, info_gain(left, right, gi)])

ig_table = pd.DataFrame(info_gain_list, columns=['split_value', 'in
ig_table.head(10)
```

Out[23]:

	split_value	info_gain
10	8.84	0.296932
124	8.85	0.291464
139	8.88	0.290704
18	8.90	0.290054
98	8.83	0.287810
110	8.87	0.286050
152	8.94	0.284714
57	8.96	0.284210
96	8.80	0.283371
21	9.00	0.283364

In [27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.plot(ig_table['split_value'], ig_table['info_gain'],'o')
plt.plot(ig_table['split_value'].iloc[0], ig_table['info_gain'].ilo
plt.xlabel('cgpa split value')
plt.ylabel('info gain')
```

Out[27]:
```
Text(0, 0.5, 'info gain')
```



In [28]:
```
# now lets do regression for the above
```

In [29]:
```
X = df.loc[:,'gre_score':'research']
y = df['chance_of_admit']
```

In [31]:
```
X.head()
```

Out[31]:

	gre_score	toefl_score	university_rating	sop	lor	cgpa	research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

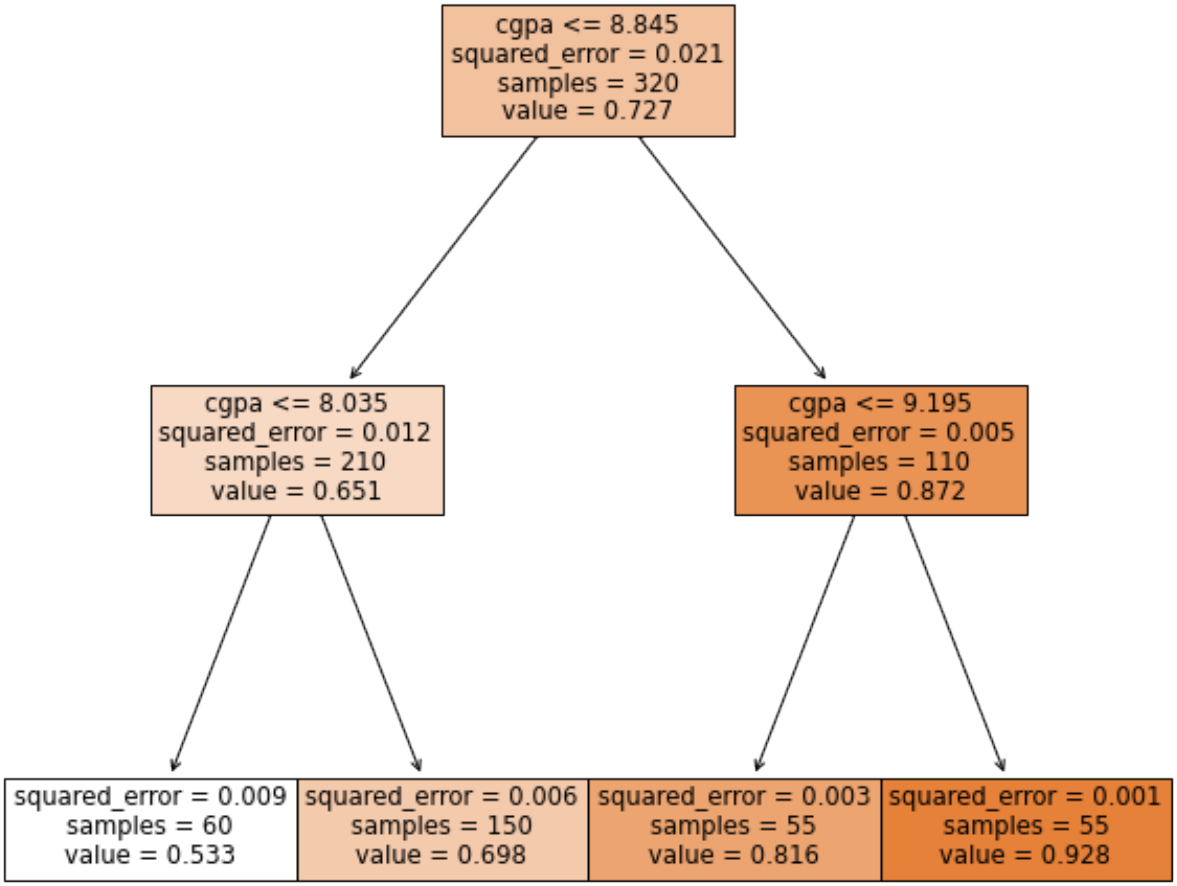
In [32]:
```
y.head()
```

Out[32]:
```
0    0.92
1    0.76
2    0.72
3    0.80
4    0.65
Name: chance_of_admit, dtype: float64
```

In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
x_train, x_test, y_train, y_test = train_test_split(X,y, random_sta
dt = DecisionTreeRegressor(max_depth=3, ccp_alpha=0.001)
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
print(dt.score(x_test, y_test))
```

```
0.5230242793515553
```

In [35]:
```
plt.figure(figsize=(10,10))
tree.plot_tree(dt, feature_names = x_train.columns,
               max_depth=2, filled=True);
```



Split Criteria

Unlike the classification problem, there are no longer classes to split the tree by. Instead, at each level, the value is the average of all samples that fit the logical criteria. In terms of evaluating the split, the default method is MSE. For example, the root node, the average target value is 0.727 (verify y_train.mean()). Then the MSE (mean-squared error) if we were to use 0.727 as the value for all samples, would be:

np.mean((y_train - y_train.mean())**2) = 0.02029

Now to determine the split, for each value of cpga, the information gain, or decrease in MSE after the split, is calculated and then values are sorted. Like before, we can modify our functions for the regression version, and see the best split is again cpga<=8.84.

The below code challenges walks you through the details — in the regression version, instead of Gini impurity, MSE is used, and the information gain function is modified to mse_gain.

In [36]:
```
def mse(data):
    """Calculate the MSE of a data set
    """
    return np.mean((data - data.mean())**2)

def mse_gain(left, right, current_mse):
    """Information Gain (MSE) associated with creating a node/split
    Input: left, right are data in left branch, right banch, respec
    current_impurity is the data impurity before splitting into lef
    """
    # weight for gini score of the left branch
    w = float(len(left)) / (len(left) + len(right))
    return current_mse - w * mse(left) - (1 - w) * mse(right)

m = None
print(f'MSE at root: {round(m,3)}')

mse_gain_list = []
for i in x_train.cgpa.unique():
    left = y_train[x_train.cgpa<=i]
    right = y_train[x_train.cgpa>i]
    mse_gain_list.append([i, mse_gain(left, right, m)])

mse_table = pd.DataFrame(mse_gain_list,columns=['split_value', 'inf
print(mse_table.head(10))

print(f'Split with highest information gain is: {None}')

plt.plot(mse_table['split_value'], mse_table['info_gain'],'o')
plt.plot(mse_table['split_value'].iloc[0], mse_table['info_gain'].i
plt.xlabel('cgpa split value')
plt.ylabel('info gain')
```

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
Input In [36], in <cell line: 16>()
     13     return current_mse - w * mse(left) - (1 - w) *
mse(right)
     15 m = None
---> 16 print(f'MSE at root: {round(m,3)}')
     18 mse_gain_list = []
     19 for i in x_train.cgpa.unique():

TypeError: type NoneType doesn't define __round__ method
```

In [38]:
```
# not correct
```

In [39]:
```
#### Find the flag challenge
```

In [49]:
```
#import codecademylib3
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

#https://archive.ics.uci.edu/ml/machine-learning-databases/flags/fl
cols = ['name','landmass','zone', 'area', 'population', 'language',
'red','green','blue','gold','white','black','orange','mainhue','cir
'crosses','saltires','quarters','sunstars','crescent','triangle','i

df= pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-da
print("====================Zero==================")
print(df.head())

#variable names to use as predictors
var = [ 'red', 'green', 'blue','gold', 'white', 'black', 'orange',
print("======one========")
#Print number of countries by landmass, or continent
print(df.landmass.value_counts())

#Create a new dataframe with only flags from Europe and Oceania
df_36 = df[df["landmass"].isin([3,6])]
print("======two========")
#Print the average vales of the predictors for Europe and Oceania
print(df_36.groupby('landmass')[var].mean().T)
print(df["landmass"].head())

#Create labels for only Europe and Oceania
df_36 = df[df["landmass"].isin([3,6])]
labels = df_36["landmass"]
print("======Three========")
print(labels.head())

#Print the variable types for the predictors
print(df[var].dtypes)

#Create dummy variables for categorical predictors
data = pd.get_dummies(df_36[var])
print("=========four=========")
print(data.head())
#Split data into a train and test set
train_data, test_data, train_labels, test_labels = train_test_split

#Fit a decision tree for max_depth values 1-20; save the accuracy s
depths = range(1, 21)
acc_depth = []
```

```
for i in depths:
    dt = DecisionTreeClassifier(random_state = 10, max_depth = i)
    dt.fit(train_data, train_labels)
    acc_depth.append(dt.score(test_data, test_labels))

#Plot the accuracy vs depth
plt.plot(depths, acc_depth)
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.show()

#Find the largest accuracy and the depth this occurs
max_acc = np.max(acc_depth)
best_depth = depths[np.argmax(acc_depth)]
print(f'Highest accuracy {round(max_acc,3)*100}% at depth {best_dep

#Refit decision tree model with the highest accuracy and plot the d
plt.figure(figsize=(14,8))
dt = DecisionTreeClassifier(random_state = 1, max_depth = best_dept
dt.fit(train_data, train_labels)
tree.plot_tree(dt, feature_names = train_data.columns,
               class_names = ['Europe', 'Oceania'],
                filled=True)
plt.show()

#Create a new list for the accuracy values of a pruned decision tre
#the values of ccp and append the scores to the list
acc_pruned = []
ccp = np.logspace(-3, 0, num=20)
for i in ccp:
    dt_prune = DecisionTreeClassifier(random_state = 1, max_depth =
    dt_prune.fit(train_data, train_labels)
    acc_pruned.append(dt_prune.score(test_data, test_labels))

plt.plot(ccp, acc_pruned)
plt.xscale('log')
plt.xlabel('ccp_alpha')
plt.ylabel('accuracy')
plt.show()

#Find the largest accuracy and the ccp value this occurs
max_acc_pruned = np.max(acc_pruned)
best_ccp = ccp[np.argmax(acc_pruned)]

print(f'Highest accuracy {round(max_acc_pruned,3)*100}% at ccp_alph

#Fit a decision tree model with the values for max_depth and ccp_al
dt_final = DecisionTreeClassifier(random_state = 1, max_depth = bes
dt_final.fit(train_data, train_labels)

#Plot the final decision tree
plt.figure(figsize=(14,8))
tree.plot_tree(dt_final, feature_names = train_data.columns,
               class_names = ['Europe', 'Oceania'],
```

```
                class_names = ['Europe', 'Oceania'],
                filled=True)
plt.show()
```

```
====================Zero==================
             name  landmass  zone  area  population  language  rel
igion  bars  \
0     Afghanistan         5     1   648          16        10
2     0
1         Albania         3     1    29           3         6
6     0
2         Algeria         4     1  2388          20         8
2     2
3  American-Samoa         6     3     0           0         1
1     0
4         Andorra         3     1     0           0         6
0     3

   stripes  colours  ...  saltires  quarters  sunstars  crescent
triangle  \
0        3        5  ...         0         0         1         0
0
1        0        3  ...         0         0         1         0
0
```

Normalization
This article describes why normalization is necessary. It also demonstrates the pros and cons of min-max normalization and z-score normalization.
Why Normalize?

Many machine learning algorithms attempt to find trends in the data by comparing features of data points. However, there is an issue when the features are on drastically different scales.

For example, consider a dataset of houses. Two potential features might be the number of rooms in the house, and the total age of the house in years. A machine learning algorithm could try to predict which house would be best for you. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other. Take a look at the image below:

Data points on the y-axis range from 0 to 20. Data points on the x-axis range from 0 to 100

When the data looks squished like that, we know we have a problem. The machine learning algorithm should realize that there is a huge difference between a house with 2 rooms and a house with 20 rooms. But right now, because two houses can be 100 years apart, the difference in the number of rooms contributes less to the overall difference.

As a more extreme example, imagine what the graph would look like if the x-axis was the cost of the house. The data would look even

more squished; the difference in the number of rooms would be even less relevant because the cost of two houses could have a difference of thousands of dollars.

The goal of normalization is to make every datapoint have the same scale so each feature is equally important. The image below shows the same house data normalized using min-max normalization.

Data points on the y-axis range from 0 to 1. Data points on the x-axis range from 0 to 1

Min-Max Normalization

Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

For example, if the minimum value of a feature was 20, and the maximum value was 40, then 30 would be transformed to about 0.5 since it is halfway between 20 and 40. The formula is as follows:

v
a
l
u
e
−
m
i
n
m
a
x
−
m
i
n
max−min
value−min
•
Min-max normalization has one fairly significant downside: it does not handle outliers very well. For example, if you have 99 values between 0 and 40, and one value is 100, then the 99 values will all be transformed to a value between 0 and 0.4. That data is just as squished as before! Take a look at the image below to see an example of this.

Almost all normalized data points have an x value between 0 and 0.4

Normalizing fixed the squishing problem on the y-axis, but the x-axis is still problematic. Now if we were to compare these points,

the y-axis would dominate; the y-axis can differ by 1, but the x-axis can only differ by 0.4.

Z-Score Normalization

Z-score normalization is a strategy of normalizing data that avoids this outlier issue. The formula for Z-score normalization is below:

v
a
l
u
e
−
μ
σ
σ
value−μ
•
Here, μ is the mean value of the feature and σ is the standard deviation of the feature. If a value is exactly equal to the mean of all the values of the feature, it will be normalized to 0. If it is below the mean, it will be a negative number, and if it is above the mean it will be a positive number. The size of those negative and positive numbers is determined by the standard deviation of the original feature. If the unnormalized data had a large standard deviation, the normalized values will be closer to 0.

Take a look at the graph below. This is the same data as before, but this time we're using z-score normalization.

All points have a similar range in both the x and y dimensions

While the data still looks squished, notice that the points are now on roughly the same scale for both features — almost all points are between -2 and 2 on both the x-axis and y-axis. The only potential downside is that the features aren't on the exact same scale.

With min-max normalization, we were guaranteed to reshape both of our features to be between 0 and 1. Using z-score normalization, the x-axis now has a range from about -1.5 to 1.5 while the y-axis has a range from about -2 to 2. This is certainly better than before; the x-axis, which previously had a range of 0 to 40, is no longer dominating the y-axis.

Review

Normalizing your data is an essential part of machine learning. You might have an amazing dataset with many great features, but if you forget to normalize, one of those features might completely dominate the others. It's like you're throwing away almost all of

your information! Normalizing solves this problem. In this article, you learned the following techniques to normalize:

Min-max normalization: Guarantees all features will have the exact same scale but does not handle outliers well.
Z-score normalization: Handles outliers, but does not produce normalized data with the exact same scale.

In [50]:
```
from sklearn.metrics import confusion_matrix

actual = [1, 0, 0, 1, 1, 1, 0, 1, 1, 1]
predicted = [0, 1, 1, 1, 1, 0, 1, 0, 1, 0]

true_positives = 0
true_negatives = 0
false_positives = 0
false_negatives = 0

for i in range(len(predicted)):
  if actual[i] == 1 and predicted[i] == 1 :
    true_positives += 1
  if actual[i] == 0 and predicted[i] == 0 :
    true_negatives += 1
  if actual[i] == 1 and predicted[i] == 0 :
     false_negatives += 1
  if actual[i] == 0 and predicted[i] == 1 :
    false_positives += 1


print(true_positives, true_negatives, false_negatives, false_positi

conf_matrix = confusion_matrix(actual, predicted)
print(conf_matrix)
```

```
3 0 4 3
[[0 3]
 [4 3]]
```

In [51]:
```
accuracy = (true_positives + true_negatives)/(true_positives + true

print(accuracy)
```

```
0.3
```

In [52]:
```
recall = true_positives/(true_positives+false_negatives)
print(recall)
```

```
0.42857142857142855
```

In [53]:
```
precision = true_positives/(true_positives+false_positives)

f_1 = (2*precision*recall)/(precision+recall)
print(precision)
print(f_1)
```

```
0.5
0.4615384615384615
```

In [54]:
```
from sklearn.metrics import accuracy_score, recall_score, precision

actual = [1, 0, 0, 1, 1, 1, 0, 1, 1, 1]
predicted = [0, 1, 1, 1, 1, 0, 1, 0, 1, 0]

print(accuracy_score(actual, predicted))

print(recall_score(actual, predicted))

print(precision_score(actual, predicted))

print(f1_score(actual,predicted))
```

```
0.3
0.42857142857142855
0.5
0.4615384615384615
```

Introduction to Feature Selection Methods
Learn about the pros and cons of different feature selection methods.
Introduction and motivation

Imagine you want to build a machine learning model for predicting the presence of a disease, and you have a dataset of patient records containing hundreds of different metrics, from demographic characteristics to medical test results to past treatment history. While having access to so many data points is always a good starting point, you most likely would not want to use all of the available features to construct your model, as many of them can be redundant or irrelevant, and would only serve to contribute unnecessary noise. Instead, a more effective approach is to select only a subset of relevant, predictive features to use in your model – this process is known as feature selection.

Feature selection is an important step in the machine learning pipeline, and when done right, can optimize the performance and predictive power of your model. The goal is to improve the model's accuracy and efficiency by eliminating extraneous variables that do not contribute any useful information. In addition, simplifying the model through feature selection not only makes the results more easily interpretable, but also reduces the time and resources required to run the model.

There are three broad categories of feature selection that we will discuss in this article: filter methods, wrapper methods, and embedded methods.

Feature Selection Methods

Filter methods

Filter methods are the simplest type of feature selection method. They work by filtering features prior to model building based on some criteria.

Advantages

They are computationally inexpensive, since they do not involve testing the subsetted features using a model.
They can work for any type of machine learning model.
Disadvantages

It is more difficult to take multivariate relationships into account because we are not evaluating model performance. For example, a variable might not have much predictive power on its own, but can be informative when combined with other variables.
They are not tailored toward specific types of models.
Examples

Variance thresholds
Correlation
Mutual information
Wrapper methods

Wrapper methods involve fitting a model and evaluating its performance for a particular subset of features. They work by using a search algorithm to find which combination of features can optimize the performance of a given model.

Advantages

They can determine the optimal set of features that produce the best results for a specific machine learning problem.
They can better account for multivariate relationships because model performance is evaluated.
Disadvantages

They are computationally expensive because the model needs to be re-fitted for each feature set being tested.
Examples

Forward/backward/bidirectional sequential feature selection
Recursive feature elimination
Embedded methods

Embedded methods also involve building and evaluating models for different feature subsets, but their feature selection process

different feature subsets, but their feature selection process happens at the same time as their model fitting step.

Advantages

Like wrapper methods, they can optimize the feature set for a particular model and account for multivariate relationships. They are also generally less computationally expensive because feature selection happens during model training.
Examples

Regularization (e.g., lasso/ridge regression)
Tree-based feature importance

In [55]:
```
#lets do a Feature selection using filter methond
```

In [56]:
```
import pandas as pd

df = pd.DataFrame(data={
    'edu_goal': ['bachelors', 'bachelors', 'bachelors', 'masters',
    'hours_study': [1, 2, 3, 3, 3, 4, 3, 4, 5, 5],
    'hours_TV': [4, 3, 4, 3, 2, 3, 2, 2, 1, 1],
    'hours_sleep': [10, 10, 8, 8, 6, 6, 8, 8, 10, 10],
    'height_cm': [155, 151, 160, 160, 156, 150, 164, 151, 158, 152]
    'grade_level': [8, 8, 8, 8, 8, 8, 8, 8, 8, 8],
    'exam_score': [71, 72, 78, 79, 85, 86, 92, 93, 99, 100]
})

print(df)
```

```
    edu_goal  hours_study  hours_TV  hours_sleep  height_cm  grade
_level  \
0  bachelors            1         4           10        155
8
1  bachelors            2         3           10        151
8
2  bachelors            3         4            8        160
8
3    masters            3         3            8        160
8
4    masters            3         2            6        156
8
5    masters            4         3            6        150
8
6    masters            3         2            8        164
8
7        phd            4         2            8        151
8
8        phd            5         1           10        158
8
9        phd            5         1           10        152
8

   exam_score
0          71
1          72
2          78
3          79
4          85
5          86
6          92
7          93
8          99
9         100
```

Our goal is to use the data to predict how well each student will perform on the exam. Thus, our target variable is exam_score and the remaining 6 variables are our features. We'll prepare the data by separating the features matrix (X) and the target vector (y):

In [57]:
```
X = df.drop(columns=['exam_score'])

print(X)
```

```
    edu_goal  hours_study  hours_TV  hours_sleep  height_cm  grade
_level
0  bachelors            1         4           10        155
8
1  bachelors            2         3           10        151
8
2  bachelors            3         4            8        160
8
3    masters            3         3            8        160
8
4    masters            3         2            6        156
8
5    masters            4         3            6        150
8
6    masters            3         2            8        164
8
7        phd            4         2            8        151
8
8        phd            5         1           10        158
8
9        phd            5         1           10        152
8
```

In [58]:
```
y = df['exam_score']

print(y)
```

```
0     71
1     72
2     78
3     79
4     85
5     86
6     92
7     93
8     99
9    100
Name: exam_score, dtype: int64
```

One of the most basic filter methods is to use a variance threshold to remove any features that have little to no variation in their values. This is because features with low variance do not contribute much information to a model. Since variance can only be calculated on numeric values, this method only works on quantitative features. That said, we may also want to remove categorical features for which all or a majority of the values are the same. To do that, we would need to dummy code the categorical variables first, but we won't demonstrate that here.

In our example dataset, edu_goal is the only feature that is not numeric. We can use the .drop() method to remove it from our features DataFrame and store the remaining numeric features in X_num:

In [59]:
```
X_num = X.drop(columns=['edu_goal'])

print(X_num)
```

	hours_study	hours_TV	hours_sleep	height_cm	grade_level
0	1	4	10	155	8
1	2	3	10	151	8
2	3	4	8	160	8
3	3	3	8	160	8
4	3	2	6	156	8
5	4	3	6	150	8
6	3	2	8	164	8
7	4	2	8	151	8
8	5	1	10	158	8
9	5	1	10	152	8

Now, we'll be able to use the VarianceThreshold class from scikit-learn to help remove the low-variance features from X_num. By default, it drops all features with zero variance, but we can adjust the threshold during class instantiation using the threshold parameter if we want to allow some variation. The .fit_transform() method returns the filtered features as a numpy array:

In [60]:
```
from sklearn.feature_selection import VarianceThreshold

selector = VarianceThreshold(threshold=0)  # 0 is default

print(selector.fit_transform(X_num))
```

```
[[  1   4  10 155]
 [  2   3  10 151]
 [  3   4   8 160]
 [  3   3   8 160]
 [  3   2   6 156]
 [  4   3   6 150]
 [  3   2   8 164]
 [  4   2   8 151]
 [  5   1  10 158]
 [  5   1  10 152]]
```

In [61]:
```
num_cols = list(X_num.columns[selector.get_support(indices=True)])

print(num_cols)
```

```
['hours_study', 'hours_TV', 'hours_sleep', 'height_cm']
```

In [62]:
```
X_num = X_num[num_cols]

print(X_num)
```

```
   hours_study  hours_TV  hours_sleep  height_cm
0            1         4           10        155
1            2         3           10        151
2            3         4            8        160
3            3         3            8        160
4            3         2            6        156
5            4         3            6        150
6            3         2            8        164
7            4         2            8        151
8            5         1           10        158
9            5         1           10        152
```

In [63]:
```
#Finally, to obtain our entire features DataFrame, including the ca

X = X[['edu_goal'] + num_cols]

print(X)
```

```
    edu_goal  hours_study  hours_TV  hours_sleep  height_cm
0  bachelors            1         4           10        155
1  bachelors            2         3           10        151
2  bachelors            3         4            8        160
3    masters            3         3            8        160
4    masters            3         2            6        156
5    masters            4         3            6        150
6    masters            3         2            8        164
7        phd            4         2            8        151
8        phd            5         1           10        158
9        phd            5         1           10        152
```

Another type of filter method involves finding the correlation between variables. In particular, the Pearson's correlation coefficient is useful for measuring the linear relationship between two numeric, continuous variables – a coefficient close to 1 represents a positive correlation, -1 represents a negative correlation, and 0 represents no correlation. Like variance, Pearson's correlation coefficient cannot be calculated for categorical variables. Although, there is a related point biserial correlation coefficient that can be computed when one variable is dichotomous, but we won't focus on that here.

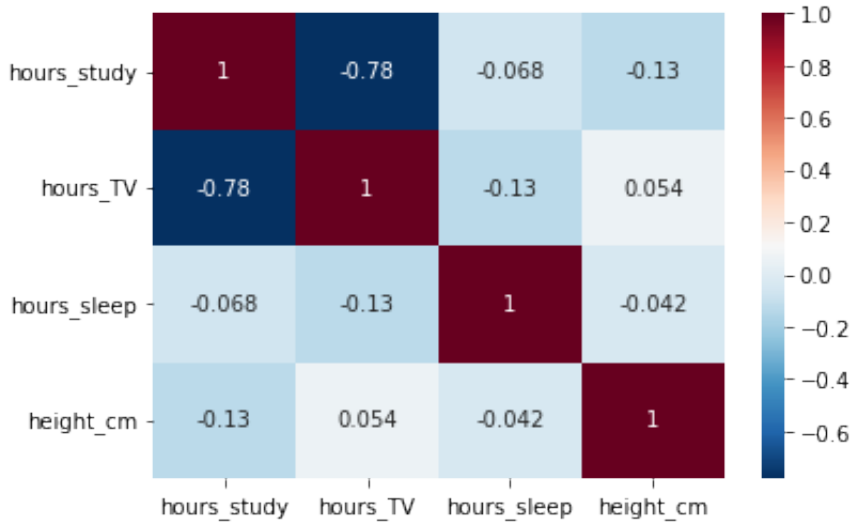
There are 2 main ways of using correlation for feature selection – to detect correlation between features and to detect correlation between a feature and the target variable.

Correlation between features

When two features are highly correlated with one another, then keeping just one to be used in the model will be enough because otherwise they provide duplicate information. The second variable would only be redundant and serve to contribute unnecessary noise.

To determine which variables are correlated with one another, we can use the .corr() method from pandas to find the correlation coefficient between each pair of numeric features in a DataFrame. By default, .corr() computes the Pearson's correlation coefficient, but alternative methods can be specified using the method parameter. We can visualize the resulting correlation matrix using a heatmap:

In [64]:
```
import matplotlib.pyplot as plt
import seaborn as sns

corr_matrix = X_num.corr(method='pearson')  # 'pearson' is default

sns.heatmap(corr_matrix, annot=True, cmap='RdBu_r')
plt.show()
```



In [65]:
```
#Let's define high correlation as having a coefficient of greater t

# Loop over bottom diagonal of correlation matrix
for i in range(len(corr_matrix.columns)):
    for j in range(i):

        # Print variables with high correlation
        if abs(corr_matrix.iloc[i, j]) > 0.7:
            print(corr_matrix.columns[i], corr_matrix.columns[j], c
```

```
hours_TV hours_study -0.780763315142435
```

As seen, hours_TV appears to be highly negatively correlated with hours_study – a student who watches a lot of TV tends to spend fewer hours studying, and vice versa. Because they provide redundant information, we can choose to remove one of those variables. To decide which one, we can look at their correlation with the target variable, then remove the one that is less associated with the target. This is explored in the next section.

Correlation between feature and target

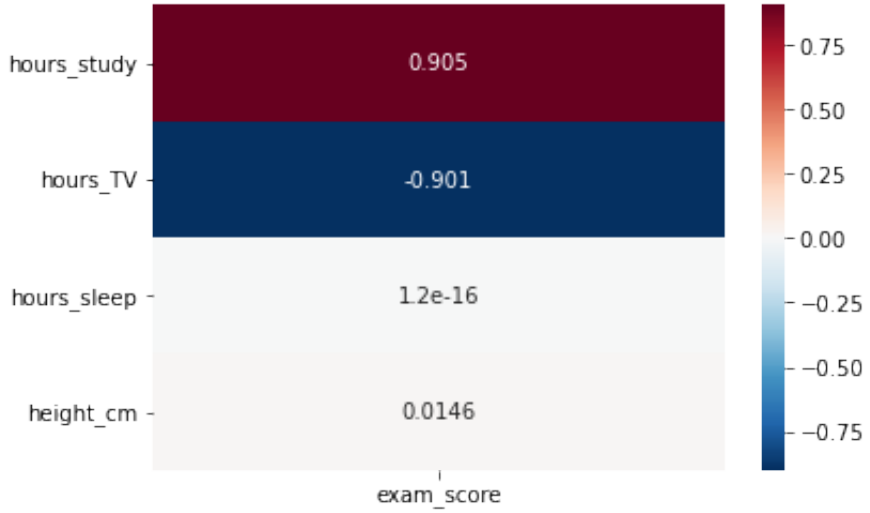
As mentioned, the second way correlation can be used is to determine if there is a relationship between a feature and the target variable. In the case of Pearson's correlation, this is especially useful if we intend to fit a linear model, which assumes a linear relationship between the target and predictor variables. If a feature is not very correlated with the target variable, such as having a coefficient of between −0.3 and 0.3, then it may not be very predictive and can potentially be filtered out.

We can use the same .corr() method seen previously to obtain the correlation between the target variable and the rest of the features. First, we'll need to create a new DataFrame containing the numeric features with the exam_score column:

In [66]:
```
X_y = X_num.copy()
X_y['exam_score'] = y

print(X_y)
```

```
   hours_study  hours_TV  hours_sleep  height_cm  exam_score
0            1         4           10        155          71
1            2         3           10        151          72
2            3         4            8        160          78
3            3         3            8        160          79
4            3         2            6        156          85
5            4         3            6        150          86
6            3         2            8        164          92
7            4         2            8        151          93
8            5         1           10        158          99
9            5         1           10        152         100
```

In [67]:
```
#Then, we can generate the correlation matrix and isolate the colum

corr_matrix = X_y.corr()

# Isolate the column corresponding to `exam_score`
corr_target = corr_matrix[['exam_score']].drop(labels=['exam_score'

sns.heatmap(corr_target, annot=True, fmt='.3', cmap='RdBu_r')
plt.show()
```



In [68]:
```
#As seen, hours_study is positively correlated with exam_score and

X = X.drop(columns=['hours_TV'])

print(X)
```

	edu_goal	hours_study	hours_sleep	height_cm
0	bachelors	1	10	155
1	bachelors	2	10	151
2	bachelors	3	8	160
3	masters	3	8	160
4	masters	3	6	156
5	masters	4	6	150
6	masters	3	8	164
7	phd	4	8	151
8	phd	5	10	158
9	phd	5	10	152

The other two features, hours_sleep and height_cm, both do not seem to be correlated with exam_score, suggesting they would not be very good predictors. We could potentially remove either or both of them as being uninformative. But before we do, it is a good idea to use other methods to double check that the features truly are not predictive. We will do that in the next section by using mutual information to see if there are any non-linear associations between the features and target variable.

To conclude this section, we'll briefly note an alternative approach for assessing the correlation between variables. Instead of generating the full correlation matrix, we could use the f_regression() function from scikit-learn to find the F-statistic for a model with each predictor on its own. The F-statistic will be larger (and p-value will be smaller) for predictors that are more highly correlated with the target variable, thus it will perform the same filtering:

In [69]:
```
from sklearn.feature_selection import f_regression

print(f_regression(X_num, y))
```

```
(array([3.61362007e+01, 3.44537037e+01, 0.00000000e+00, 1.70259066
e-03]), array([3.19334945e-04, 3.74322763e-04, 1.00000000e+00, 9.6
8097878e-01]))
```

The function returns the F-statistic in the first array and the p-value in the second. As seen, the result is consistent with what we had observed in the correlation matrix — the stronger the correlation (either positive or negative) between the feature and target, the higher the corresponding F-statistic and lower the p-value. For example, amongst all the features, hours_study has the largest correlation coefficient (0.905), highest F-statistic (3.61e+01), and lowest p-value (3.19e-04).

Mutual information

The final filter method we'll look at is using mutual information to rank and select the top features. Mutual information is a measure of dependence between two variables and can be used to gauge how much a feature contributes to the prediction of the target variable. It is similar to Pearson's correlation, but is not limited to detecting linear associations. This makes mutual information useful for more flexible models where a linear functional form is not assumed. Another advantage of mutual information is that it also works on discrete features or target, unlike correlation. Although, categorical variables need to be numerically encoded first.

In our example, we can encode the edu_goal column using the LabelEncoder class from scikit-learn's preprocessing module:

In [70]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Create copy of `X` for encoded version
X_enc = X.copy()
X_enc['edu_goal'] = le.fit_transform(X['edu_goal'])

print(X_enc)
```

```
   edu_goal  hours_study  hours_sleep  height_cm
0         0            1           10        155
1         0            2           10        151
2         0            3            8        160
3         1            3            8        160
4         1            3            6        156
5         1            4            6        150
6         1            3            8        164
7         2            4            8        151
8         2            5           10        158
9         2            5           10        152
```

Now, we can compute the mutual information between each feature and exam_score using mutual_info_regression(). This function is used because our target variable is continuous, but if we had a discrete target variable, we would use mutual_info_classif(). We specify the random_state in the function in order obtain reproducible results:

In [71]:
```
from sklearn.feature_selection import mutual_info_regression

print(mutual_info_regression(X_enc, y, random_state=68))
```

```
[0.50396825 0.40896825 0.06896825 0.        ]
```

The estimated mutual information between each feature and the target is returned in a numpy array, where each value is a non-negative number – the higher the value, the more predictive power is assumed.

However, we are missing one more important piece here. Earlier, even though we encoded edu_goal to be numeric, that does not mean it should be treated as a continuous variable. In other words, the values of edu_goal are still discrete and should be interpreted as such. If we plot edu_goal against exam_score on a graph, we can clearly see the steps between the values of edu_goal:

In order to properly calculate the mutual information, we need to tell mutual_info_regression() which features are discrete by providing their index positions using the discrete_features parameter:

In [72]:
```
print(mutual_info_regression(X_enc, y, discrete_features=[0], rando
```

```
[0.75563492 0.38896825 0.18563492 0.        ]
```

What is interesting to note is that the mutual information between hours_sleep and exam_score is a positive value, even though their Pearson's correlation coefficient is 0. The answer becomes more clear when we plot the relationship between hours_sleep and exam_score:

As seen, there do seem to be some association between the variables, only it is not a linear one, which is why it was detected using mutual information but not Pearson's correlation coefficient.

Finally, let's look at using the SelectKBest class from scikit-learn to help pick out the top k features with the highest ranked scores. In our case, we are looking to select features that share the most mutual information with the target variable. When we instantiate SelectKBest, we'll specify which scoring function to use and how many top features to select. Here, our scoring function is mutual_info_regression(), but because we want to specify additional arguments besides the X and y inputs, we'll need the help of the partial() function from Python's built-in functools module. Then, the .fit_transform() method will return the filtered features as a numpy array:

In [73]:
```
from sklearn.feature_selection import SelectKBest
from functools import partial

score_func = partial(mutual_info_regression, discrete_features=[0],

# Select top 3 features with the most mutual information
selection = SelectKBest(score_func=score_func, k=3)

print(selection.fit_transform(X_enc, y))
```

```
[[ 0  1 10]
 [ 0  2 10]
 [ 0  3  8]
 [ 1  3  8]
 [ 1  3  6]
 [ 1  4  6]
 [ 1  3  8]
 [ 2  4  8]
 [ 2  5 10]
 [ 2  5 10]]
```

As seen above, we selected the top 3 features based on mutual information, thus dropping height_cm. Like VarianceThreshold, SelectKBest also offers the .get_support() method that returns the indices of the selected features, so we could subset our original features DataFrame:

In [74]:
```
X = X[X.columns[selection.get_support(indices=True)]]

print(X)
```

```
    edu_goal  hours_study  hours_sleep
0  bachelors            1           10
1  bachelors            2           10
2  bachelors            3            8
3    masters            3            8
4    masters            3            6
5    masters            4            6
6    masters            3            8
7        phd            4            8
8        phd            5           10
9        phd            5           10
```

Conclusion

In our example dataset, we started out with 6 features for predicting the exam_score of students. Using various filter methods, we narrowed down that set to just the top most relevant and informative ones. First, we eliminated grade_level because it has zero variance and would contribute nothing to the model. Then, we dropped hours_TV since it is highly correlated with hours_study and is therefore redundant. Lastly, we filtered out height_cm based on mutual information, which suggested that it does not have any meaningful association with the target variable, linear or otherwise, and would not have been very predictive.

Phew! That was a lot we were able to accomplish using filter methods. Being the most simple type of feature selection method, they sure do not lack power nor potential. It is certainly worth considering how you might want to incorporate filter methods into your next machine learning project.

Introduction to Wrapper Methods
Machine learning problems often involve datasets with many features. Some of those features might be very important for a specific machine learning model. Other features might be irrelevant. Given a feature set and a model, we would like to be able to distinguish between important and unimportant features (or even important combinations of features). Wrapper methods do exactly that.

A wrapper method for feature selection is an algorithm that selects features by evaluating the performance of a machine learning model on different subsets of features. These algorithms add or remove features one at a time based on how useful those features are to the model.

Wrapper methods have some advantages over filter methods. The main advantage is that wrapper methods evaluate features based on their performance with a specific model. Filter methods, on the other hand, can't tell how important a feature is to a model.

Another upside of wrapper methods is that they can take into account relationships between features. Sometimes certain features aren't very useful on their own but instead perform well only when combined with other features. Since wrapper methods test subsets of features, they can account for those situations.

This lesson will explain five different wrapper methods:

Sequential forward selection
Sequential backward selection
Sequential forward floating selection
Sequential backward floating selection
Recursive feature elimination

In [75]:
```
pwd
```

Out[75]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

In [76]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Load the data
health = pd.read_csv("dataR2.csv")
```

In [78]:
```
print(health.head(10))
```

```
   Age        BMI  Glucose  Insulin      HOMA   Leptin  Adiponecti
n  Resistin  \
0   48  23.500000       70    2.707  0.467409   8.8071     9.70240
0   7.99585
1   83  20.690495       92    3.115  0.706897   8.8438     5.42928
5   4.06405
2   82  23.124670       91    4.498  1.009651  17.9393    22.43204
0   9.27715
3   68  21.367521       77    3.226  0.612725   9.8827     7.16956
0  12.76600
4   86  21.111111       92    3.549  0.805386   6.6994     4.81924
0  10.57635
5   49  22.854458       92    3.226  0.732087   6.8317    13.67975
0  10.31760
6   89  22.700000       77    4.690  0.890787   6.9640     5.58986
5  12.93610
7   76  23.800000      118    6.470  1.883201   4.3110    13.25132
0   5.10420
8   73  22.000000       97    3.350  0.801543   4.4700    10.35872
5   6.28445
9   75  23.000000       83    4.952  1.013839  17.1270    11.57899
0   7.09130

      MCP.1  Classification
0   417.114               1
1   468.786               1
2   554.697               1
3   928.220               1
4   773.920               1
5   530.410               1
6  1256.083               1
7   280.694               1
8   136.855               1
9   318.302               1
```

In [79]:
```
health.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             116 non-null    int64  
 1   BMI             116 non-null    float64
 2   Glucose         116 non-null    int64  
 3   Insulin         116 non-null    float64
 4   HOMA            116 non-null    float64
 5   Leptin          116 non-null    float64
 6   Adiponectin     116 non-null    float64
 7   Resistin        116 non-null    float64
 8   MCP.1           116 non-null    float64
 9   Classification  116 non-null    int64  
dtypes: float64(7), int64(3)
memory usage: 9.2 KB
```

In [83]:
```
X = health.iloc[:,:-1]
y = health.iloc[:,-1]
# in X we specify every row (:)  & every colum except last colum (:
# in y we specify every row(:) & only last column(-1)
```

In [84]:
```
X
```

Out[84]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920
...	...	...	...	...	...	...	...	...	...
111	45	26.850000	92	3.330	0.755688	54.6800	12.100000	10.96000	268.230
112	62	26.840000	100	4.530	1.117400	12.4500	21.420000	7.32000	330.160
113	65	32.050000	97	5.730	1.370998	61.4800	22.540000	10.33000	314.050
114	72	25.590000	82	2.820	0.570392	24.9600	33.750000	3.27000	392.460
115	86	27.180000	138	19.910	6.777364	90.2800	14.110000	4.35000	90.090

116 rows × 9 columns

In [85]:
```
y
```

Out[85]:
```
0      1
1      1
2      1
3      1
4      1
      ..
111    2
112    2
113    2
114    2
115    2
Name: Classification, Length: 116, dtype: int64
```

In [87]:
```
from sklearn.linear_model import LogisticRegression

# Create and fit the logistic regression model
lr = LogisticRegression(max_iter=1000)
lr.fit(X, y)
```

Out[87]:
```
LogisticRegression(max_iter=1000)
```

In [88]:
```
print(lr.score(X,y))
```

```
0.8017241379310345
```

lets do this

Sequential Forward Selection
Now that we have a specific machine learning model, we can use a wrapper method to choose a smaller feature subset.

Sequential forward selection is a wrapper method that builds a feature set by starting with no features and then adding one feature at a time until a desired number of features is reached. In the first step, the algorithm will train and test a model using only one feature at a time. The algorithm keeps the feature that performs best.

In each subsequent step, the algorithm will test the model on each possible new feature addition. Whichever feature improves model performance the most is then added to the feature subset. This process stops once we have the desired number of features.

Let’s say we want to use three features, and we have five features to choose from: age, height, weight, blood_pressure, and resting_heart_rate. Sequential forward selection will train your machine learning model on five different feature subsets: one for each feature.

If the model performs best on the subset {age}, the algorithm will then train and test the model on the following four subsets:

{age, height}
{age, weight}
{age, blood_pressure}
{age, resting_heart_rate}
If the model performs best on {age, resting_heart_rate}, the algorithm will test the model on the following three subsets:

{age, height, resting_heart_rate}
{age, weight, resting_heart_rate}
{age, blood_pressure, resting_heart_rate}
If the model performs best on {age, weight, resting_heart_rate}, it will stop the algorithm and use that feature set.

Sequential forward selection is a greedy algorithm: instead of checking every possible feature set by brute force, it adds whichever feature gives the best immediate performance gain.

In [89]:
```
set1 = {"age", "height", "weight", "resting_heart_rate"}
set2 = {"age", "weight", "blood_pressure", "resting_heart_rate"}
```

In [93]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as
# Sequential forward selection
sfs = SFS(lr,
          k_features=3, # number of features to select
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=0)


# Fit the equential forward selection model
sfs.fit(X, y)
```

Out[93]:
```
SequentialFeatureSelector(cv=0, estimator=LogisticRegression(max_i
ter=1000),
                          k_features=(3, 3), scoring='accuracy')
```

In [99]:
```
print(sfs.subsets_[3]['feature_names'])
```

```
('Age', 'Glucose', 'Insulin')
```

In [100]:
```
print(sfs.subsets_[3]['avg_score'])
```

```
0.7672413793103449
```

In [101]:
```
print(sfs.subsets_[2]['avg_score'])
```

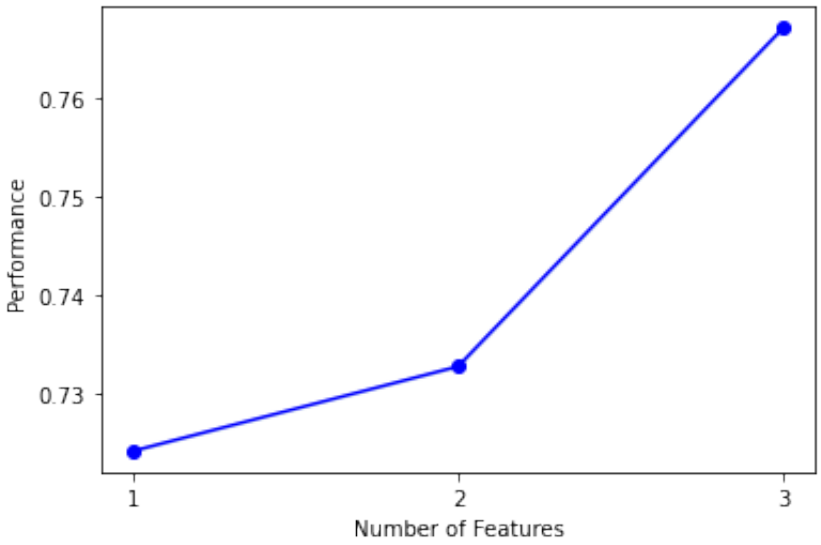
```
0.7327586206896551
```

In [102]:
```
print(sfs.subsets_[1]['avg_score'])
```

```
0.7241379310344828
```

In [103]:
```
from mlxtend.plotting import plot_sequential_feature_selection as p
import matplotlib.pyplot as plt

# Plot the accuracy of the model as a function of the number of fea
plot_sfs(sfs.get_metric_dict())
plt.show()
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:262: RuntimeWarning: Degrees of freedom <
= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:254: RuntimeWarning: invalid value encoun
tered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



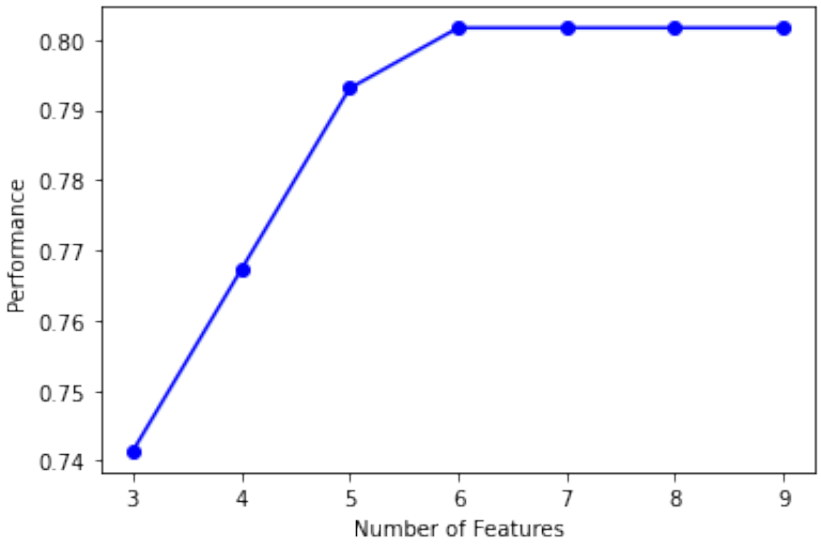
Sequential backward selection is another wrapper method for feature selection. It is very similar to sequential forward selection, but there is one key difference. Instead of starting with no features and adding one feature at a time, sequential backward selection starts with all of the available features and removes one feature at a time.

In [104]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as

# Load the data
health = pd.read_csv("dataR2.csv")
X = health.iloc[:,:-1]
y = health.iloc[:,-1]

# Logistic regression model
lr = LogisticRegression(max_iter=1000)

# Sequential backward selection
sbs = SFS(lr,
          k_features=3,
          forward=False,
          floating=False,
          scoring='accuracy',
          cv=0)

#forward = False means in its Sequential Backward Selection

# Fit sbs to X and y
sbs.fit(X,y)
```

Out[104]:
```
SequentialFeatureSelector(cv=0, estimator=LogisticRegression(max_i
ter=1000),
                          forward=False, k_features=(3, 3), scorin
g='accuracy')
```

In [105]:
```
# Print the chosen feature names
print(sbs.subsets_[3]['feature_names'])
# Print the accuracy of the model after sequential backward selecti
print(sbs.subsets_[3]['avg_score'])

plot_sfs(sbs.get_metric_dict())
plt.show()

# Plot the model accuracy
```

```
('Age', 'Glucose', 'Resistin')
0.7413793103448276

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:262: RuntimeWarning: Degrees of freedom <
= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:254: RuntimeWarning: invalid value encoun
tered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



In [113]:
```
for i in 9, 8, 7, 6, 5, 4, 3:
    print(i)
    print(sbs.subsets_[i]['feature_names'])
```

```
9
('Age', 'BMI', 'Glucose', 'Insulin', 'HOMA', 'Leptin', 'Adiponecti
n', 'Resistin', 'MCP.1')
8
('Age', 'BMI', 'Glucose', 'Insulin', 'HOMA', 'Leptin', 'Adiponecti
n', 'Resistin')
7
('Age', 'BMI', 'Glucose', 'Insulin', 'HOMA', 'Leptin', 'Resistin')
6
('Age', 'BMI', 'Glucose', 'Insulin', 'Leptin', 'Resistin')
5
('Age', 'BMI', 'Glucose', 'Insulin', 'Resistin')
4
('Age', 'BMI', 'Glucose', 'Resistin')
3
('Age', 'Glucose', 'Resistin')
```

Sequential Forward and Backward Floating Selection
Sequential forward floating selection is a variation of sequential forward selection. It starts with zero features and adds one feature at a time, just like sequential forward selection, but after each addition, it checks to see if we can improve performance by removing a feature.

If performance can’t be improved, the floating algorithm will continue to the next step and add another feature.
If performance can be improved, the algorithm will make the removal that improves performance the most (unless removal of that feature would lead to an infinite loop of adding and removing the same feature over and over again).
For example, let’s say that the algorithm has just added weight to the feature set {age, resting_heart_rate}, resulting in the set {age, weight, resting_heart_rate}. The floating algorithm will test whether it can improve performance by removing age or resting_heart_rate. If the removal of age improves performance, then the algorithm will proceed with the set {weight, resting_heart_rate}.

Sequential backward floating selection works similarly. Starting with all available features, it removes one feature at a time. After each feature removal, it will check to see if any feature additions will improve performance (but it won’t add any features that would result in an infinite loop).

Floating selection algorithms are sometimes preferable to their non-floating counterparts because they test the model on more possible feature subsets. They can detect useful relationships between variables that plain sequential forward and backward selection might miss.

In [116]:
```
#sffs & sbfs

import pandas as pd
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as

# Load the data
health = pd.read_csv("dataR2.csv")
X = health.iloc[:,:-1]
y = health.iloc[:,-1]

# Logistic regression model
lr = LogisticRegression(max_iter=1000)

# Sequential forward floating selection
sffs = SFS(lr,
           k_features=3,
           forward=True,
           floating=True,
           scoring='accuracy',
           cv=0)
sffs.fit(X, y)

# Print a tuple with the names of the features chosen by sequential
print(sffs.subsets_[3]['feature_names'])
# Sequential backward floating selection
sbfs = SFS(lr,
           k_features=3,
           forward=False,
           floating=True,
           scoring='accuracy',
           cv=0)
sbfs.fit(X, y)

# Print a tuple with the names of the features chosen by sequential
print(sbfs.subsets_[3]['feature_names'])
```

```
('Age', 'Glucose', 'Insulin')
('Age', 'Glucose', 'Resistin')
```

Recursive Feature Elimination with scikit-learn
We can use scikit-learn to implement recursive feature elimination. Since we're using a logistic regression model, it's important to standardize data before we proceed.

We can standardize features using scikit-learn's StandardScaler().

```
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(X)
```

Once the data is standardized, you can train the model and do recursive feature elimination using RFE() from scikit-learn. As before with the sequential feature selection methods, you have to specify a scikit-learn model for the estimator parameter (in this case, lr for our logistic regression model). n_features_to_select is self-explanatory: set it to the number of features you want to select.

In [117]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler

# Load the data
health = pd.read_csv("dataR2.csv")
X = np.array(health.iloc[:,:-1])
y = np.array(health.iloc[:,-1])

# Standardize the data
X = StandardScaler().fit_transform(X)

# Logistic regression model
lr = LogisticRegression(max_iter=1000)

# Recursive feature elimination
rfe = RFE(lr, n_features_to_select=3)
rfe.fit(X, y)
```

Out[117]:
```
RFE(estimator=LogisticRegression(max_iter=1000), n_features_to_sel
ect=3)
```

Evaluating the Result of Recursive Feature Elimination
You can inspect the results of recursive feature elimination by looking at rfe.ranking_ and rfe.support_.

rfe.ranking_ is an array that contains the rank of each feature. Here are the features from the fire dataset:

['Temperature', 'RH', 'Ws', 'Rain', 'DMC', 'FWI']
Here are the feature rankings after recursive feature elimination is done on the fire dataset.

```
print(rfe.ranking_)
```
[2 5 4 1 3 1]
A 1 at a certain index indicates that recursive feature elimination kept the feature at the same index. In this example, the model kept the features at indices 3 and 5: Rain and FWI. The other numbers indicate at which step a feature was removed. The 5 (the highest rank in the array) at index 1 means that RH (the feature at index 1) was removed first. The 4 at index 2 means that Ws (the feature at index 2) was removed in the next step, and so on.

rfe.support_ is an array with True and False values that indicate which features were chosen. Here's an example of what this looks like, again using the fire dataset.

```
print(rfe.support_)
```
[False False False True False True]
This array indicates that the features at indices 3 and 5 were chosen. The features at indices 0, 1, 2, and 4 were eliminated.

If you have a list of feature names, you can use a list comprehension and rfe.support_ to get a list of chosen feature names.

```
# features is a list of feature names
# Get a list of features chosen by rfe
rfe_features = [f for (f, support) in zip(features, rfe.support_)
if support]

print(rfe_features)
```
['Rain ', 'FWI']
You can use rfe.score(X, y) to check the accuracy of the model.

In [118]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler

# Load the data
health = pd.read_csv("dataR2.csv")
X = health.iloc[:,:-1]
y = health.iloc[:,-1]

# Create a list of feature names
feature_list = list(X.columns)

# Standardize the data
X = StandardScaler().fit_transform(X)

# Logistic regression
lr = LogisticRegression(max_iter=1000)

# Recursive feature elimination
rfe = RFE(estimator=lr, n_features_to_select=3)
rfe.fit(X, y)

# List of features chosen by recursive feature elimination
rfe_features = [f for (f, support) in zip(feature_list, rfe.support

# Print the accuracy of the model with features chosen by recursive
print(rfe.score(X, y))
```

```
0.7327586206896551
```

WRAPPER METHODS
Review
Congratulations! You've learned about a few different wrapper methods. You've also learned how to implement them in Python and evaluate the results. Let's recap what we covered.

Wrapper methods for feature selection are algorithms that select features by evaluating the performance of a machine learning model on different subsets of features. Here are some advantages of wrapper methods.

They can detect relationships between features that might be relevant to the machine learning model.
Unlike filter methods, they are designed to choose features that are relevant to whatever machine learning model you are using.
We covered four different greedy wrapper methods and implemented them using mlextend in Python.

Sequential forward selection adds one feature at a time.
Sequential backward selection removes one feature at a time.
Sequential forward floating selection adds (and sometimes removes) one feature at a time.
Sequential backward floating selection removes (and sometimes adds) one feature at a time.
We also covered recursive feature elimination, which ranks features by importance and removes the least important feature at every step. We used the scikit-learn library to implement that algorithm and investigate the results.

The forest fire data for this lesson were taken from the UCI Machine Learning Repository Faroudja ABID et al., Predicting Forest Fire in Algeria using Data Mining Techniques: Case Study of the Decision Tree Algorithms, International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD 2019) , 08 - 11 July , 2019, Marrakech, Morocco.

The breast cancer data for this lesson were taken from the UCI Machine Learning Repository. [Patricio, 2018] Patrício, M., Pereira, J., Crisóstomo, J., Matafome, P., Gomes, M., Seiça, R., & Caramelo, F. (2018). Using Resistin, glucose, age and BMI to predict the presence of breast cancer. BMC Cancer, 18(1).

In [1]:
```
pwd
```

Out[1]:
```
'/Users/kamallakannansekar/Documents/codecdemy/ML'
```

In [2]:
```
#Project
```

Wrapper Methods

In this project, you'll analyze data from a survey conducted by Fabio Mendoza Palechor and Alexis de la Hoz Manotas that asked people about their eating habits and weight. The data was obtained from the [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition+). Categorical variables were changed to numerical ones in order to facilitate analysis.

First, you'll fit a logistic regression model to try to predict whether survey respondents are obese based on their answers to questions in the survey. After that, you'll use three different wrapper methods to choose a smaller feature subset.

You'll use sequential forward selection, sequential backward floating selection, and recursive feature elimination. After implementing each wrapper method, you'll evaluate the model accuracy on the resulting smaller feature subsets and compare that with the model accuracy using all available features.

In [3]:
```
# Import libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as
from mlxtend.plotting import plot_sequential_feature_selection as p
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
```

Evaluating a Logistic Regression Model

The data set `obesity` contains 18 predictor variables. Here's a brief description of them.

* `Gender` is `1` if a respondent is male and `0` if a respondent is female.
* `Age` is a respondent's age in years.
* `family_history_with_overweight` is `1` if a respondent has family member who is or was overweight, `0` if not.
* `FAVC` is `1` if a respondent eats high caloric food frequently, `0` if not.
* `FCVC` is `1` if a respondent usually eats vegetables in their meals, `0` if not.
* `NCP` represents how many main meals a respondent has daily (`0` for 1–2 meals, `1` for 3 meals, and `2` for more than 3 meals).
* `CAEC` represents how much food a respondent eats between meals on a scale of `0` to `3`.
* `SMOKE` is `1` if a respondent smokes, `0` if not.
* `CH2O` represents how much water a respondent drinks on a scale of `0` to `2`.
* `SCC` is `1` if a respondent monitors their caloric intake, `0` if not.
* `FAF` represents how much physical activity a respondent does on a scale of `0` to `3`.
* `TUE` represents how much time a respondent spends looking at devices with screens on a scale of `0` to `2`.
* `CALC` represents how often a respondent drinks alcohol on a scale of `0` to `3`.
* `Automobile`, `Bike`, `Motorbike`, `Public_Transportation`, and `Walking` indicate a respondent's primary mode of transportation. Their primary mode of transportation is indicated by a `1` and the other columns will contain a `0`.

The outcome variable, `NObeyesdad`, is a `1` if a patient is obese and a `0` if not.

Use the `.head()` method and inspect the data.

In [6]:
```
# https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+lev

# Load the data
#obesity = pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv")
obesity = pd.read_csv("obesity.csv")

# Inspect the data
obesity.head()
```

Out[6]:

	Gender	Age	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O
0	0	21.0	1	0	2.0	3.0	1	0	2.0
1	0	21.0	1	0	3.0	3.0	1	1	3.0
2	1	23.0	1	0	2.0	3.0	1	0	2.0
3	1	27.0	0	0	3.0	3.0	1	0	2.0
4	1	22.0	0	0	2.0	1.0	1	0	2.0

Split the data into `X` and `y`

In order to use a linear regression model, you'll need to split the data into two parts: the predictor variables and an outcome variable. Do this by splitting the data into a DataFrame of predictor variables called `X` and a Series of outcome variables `y`.

In [7]:
```
# Split the data into predictor variables and an outcome variable
X = obesity.drop(["NObeyesdad"], axis=1)
y = obesity.NObeyesdad
```

Logistic regression model

Create a logistic regression model called `lr`. Include the parameter `max_iter=1000` to make sure that the model will converge when you try to fit it.

In [8]:
```
# Create a logistic regression model
lr = LogisticRegression(max_iter=1000)
```

In [9]:
```
# Fit the logistic regression model
lr.fit(X, y)
```

Out[9]:
```
LogisticRegression(max_iter=1000)
```

In [10]:
```
# Print the accuracy of the model
print(lr.score(X,y))
```

```
0.7659876835622927
```

Sequential Forward Selection

Now that you've created a logistic regression model and evaluated its performance, you're ready to do some feature selection.

Create a sequential forward selection model called `sfs`.

- Be sure to set the `estimator` parameter to `lr` and set the `forward` and `floating` parameters to the appropriate values.
- Also use the parameters `k_features=9`, `scoring='accuracy'`, and `cv=0`.

In [11]:
```
# Create a sequential forward selection model
sfs = SFS(lr,
          k_features=6,
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=0)
```

In [12]:
```
# Fit the sequential forward selection model to X and y
sfs.fit(X, y)
```

Out[12]:
```
SequentialFeatureSelector(cv=0, estimator=LogisticRegression(max_i
ter=1000),
                          k_features=(6, 6), scoring='accuracy')
```

In [15]:
```
# Inspect the results of sequential forward selection
print(sfs.subsets_[6])
```

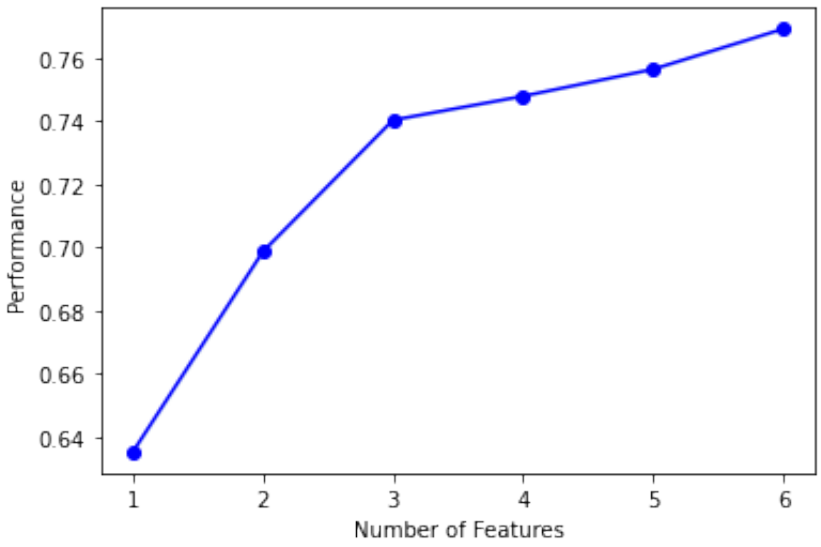
```
{'feature_idx': (1, 2, 3, 6, 9, 10), 'cv_scores': array([0.7693036
5]), 'avg_score': 0.7693036475603979, 'feature_names': ('Age', 'fa
mily_history_with_overweight', 'FAVC', 'CAEC', 'SCC', 'FAF')}
```

In [16]:
```
# See which features sequential forward selection chose
print(sfs.subsets_[6]['feature_names'])

# Print the model accuracy after doing sequential forward selection
print(sfs.subsets_[6]['avg_score'])
```

```
('Age', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SCC', '
FAF')
0.7693036475603979
```

In [17]:
```
# Plot the model accuracy as a function of the number of features u
plot_sfs(sfs.get_metric_dict())
plt.show()
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:262: RuntimeWarning: Degrees of freedom <
= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:254: RuntimeWarning: invalid value encoun
tered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



Sequential Backward Selection

Sequential forward selection was able to find a feature subset that performed marginally better than the full feature set. Let's use a different sequential method and see how it compares.

Create a sequential backward selection model called `sbs`.

- Be sure to set the `estimator` parameter to `lr` and set the `forward` and `floating` parameters to the appropriate values.
- Also use the parameters `k_features=7`, `scoring='accuracy'`, and `cv=0`.

In [18]:
```
# Create a sequential backward selection model
sbs = SFS(lr,
          k_features=7,
          forward=False,
          floating=False,
          scoring='accuracy',
          cv=0)
```

In [19]:
```
# Fit the sequential backward selection model to X and y
sbs.fit(X, y)
```

Out[19]:
```
SequentialFeatureSelector(cv=0, estimator=LogisticRegression(max_i
ter=1000),
                          forward=False, k_features=(7, 7), scorin
g='accuracy')
```

In [20]:
```
# Inspect the results of sequential backward selection
print(sbs.subsets_[7])
```

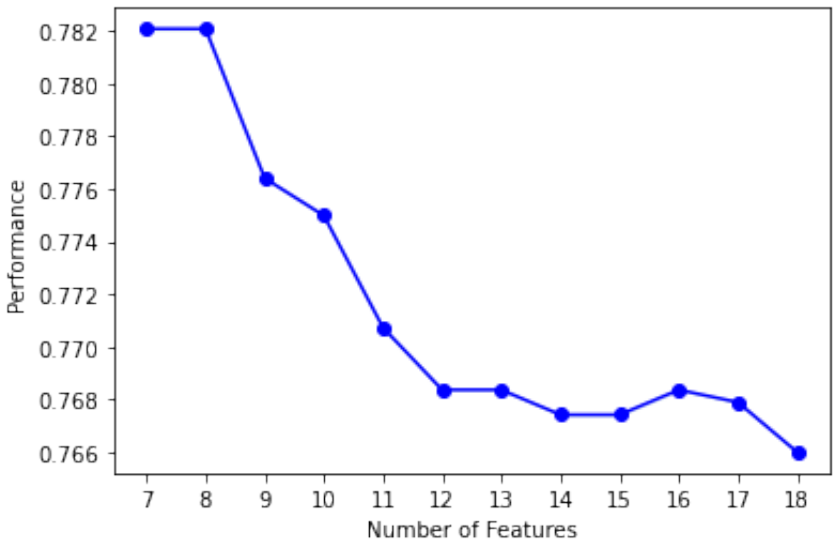
```
{'feature_idx': (0, 1, 2, 3, 6, 9, 10), 'cv_scores': array([0.7820
9379]), 'avg_score': 0.7820937944102321, 'feature_names': ('Gender
', 'Age', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SCC',
'FAF')}
```

In [21]:
```
# See which features sequential backward selection chose
print(sbs.subsets_[7]['feature_names'])

# Print the model accuracy after doing sequential backward selectio
print(sbs.subsets_[7]['avg_score'])
```

```
('Gender', 'Age', 'family_history_with_overweight', 'FAVC', 'CAEC'
, 'SCC', 'FAF')
0.7820937944102321
```

In [22]:
```
# Plot the model accuracy as a function of the number of features u
plot_sfs(sbs.get_metric_dict())
plt.show()
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:262: RuntimeWarning: Degrees of freedom <
= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/numpy/core/_methods.py:254: RuntimeWarning: invalid value encoun
tered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



Recursive Feature Elimination

So far you've tried two different sequential feature selection methods. Let's try one more: recursive feature elimination. First you'll standardize the data, then you'll fit the RFE model and inspect the results.

At a later step of this project, you'll need to be able to access feature names. Enter the code `features = X.columns` for use later.

In [25]:
```
# Get feature names
features = X.columns
```

Standardize the data

Before doing applying recursive feature elimination it is necessary to standardize the data. Standardize `X` and save it as a DataFrame by creating a `StandardScaler()` object and using the `.fit_transform()` method.

In [27]:
```
# Standardize the data
X = pd.DataFrame(StandardScaler().fit_transform(X))
```

Recursive feature elimination model

Create an `RFE()` object that selects `8` features. Be sure to set the `estimator` parameter to `lr`.

In [29]:
```
# Create a recursive feature elimination model
rfe = RFE(estimator=lr, n_features_to_select=6)
```

In [30]:
```
# Fit the recursive feature elimination model to X and y
rfe.fit(X, y)
```

Out[30]:
```
RFE(estimator=LogisticRegression(max_iter=1000), n_features_to_sel
ect=6)
```

Inspect chosen features

Now that you've fit the RFE model you can evaluate the results. Create a list of chosen feature names and call it `rfe_features`. You can use a list comprehension and filter the features in `zip(features, rfe.support_)` based on whether their support is `True` (meaning the model kept them) or `False` (meaning the model eliminated them).

Hint: `[f for (f, support) in zip(features, rfe.support_) if support]` will produce the desired list of feature names.

In [31]:
```
# See which features recursive feature elimination chose
rfe_features = [f for (f, support) in zip(features, rfe.support_) i
print(rfe_features)
```

```
['Age', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SCC', '
Automobile']
```

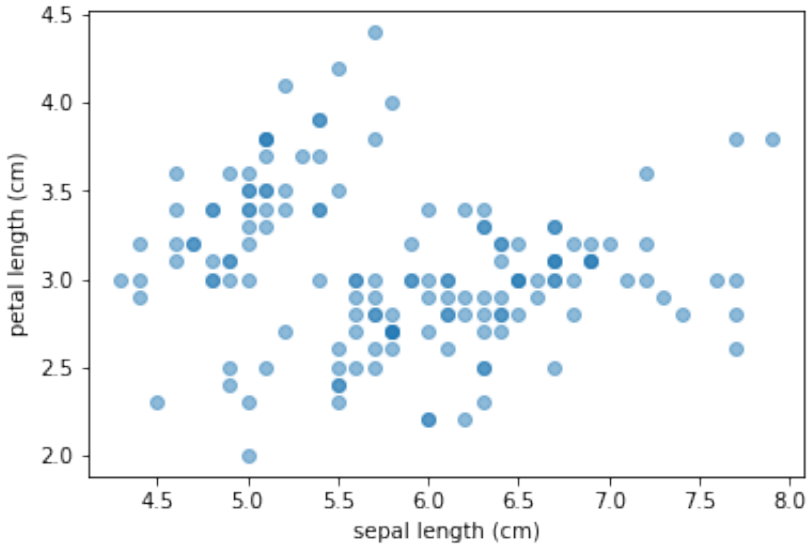
In [32]:
```
# Print the model accuracy after doing recursive feature eliminatio
print(rfe.score(X, y))
```

```
0.757934628138323
```

Unsupervised Learning

```
K-MEANS CLUSTERING - K is number of cluster , means is distance
from the centre of the cluster .
```

In [42]:
```
#import codecademylib3_seaborn
import matplotlib.pyplot as plt

from sklearn import datasets

iris = datasets.load_iris()

# Store iris.data
samples = iris.data

# Create x and y
x = samples[:,0]
y = samples[:,1]

# Plot x and y
plt.scatter(x,y, alpha=0.5)

plt.xlabel('sepal length (cm)')
plt.ylabel('petal length (cm)')

# Show the plot
plt.show()
#Adding alpha=0.5 makes some points look darker than others. The da
```



The K-Means algorithm:

1.Place k random centroids for the initial clusters.
2.Assign data samples to the nearest centroid.
3.Update centroids based on the above-assigned data samples.
4.Repeat Steps 2 and 3 until convergence.

After looking at the scatter plot and having a better understanding of the Iris data, let's start implementing the k-means algorithm.

In this exercise, we will implement Step 1.

Because we expect there to be three clusters (for the three species of flowers), let's implement k-means where the k is 3. In real-life situations you won't always know how many clusters to look for. We'll learn more about how to choose k later.

Using the NumPy library, we will create three random initial centroids and plot them along with our samples.

In [9]:
```
#import codecademylib3_seaborn
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()
print(iris)
print("kdkdkdkdk")

samples = iris.data

#The features of the dataset are:

#Column 0: Sepal length
#Column 1: Sepal width
#Column 2: Petal length
#Column 3: Petal width

#With Matplotlib, we can create a 2D scatter plot of the Iris datas
#below is sepal length vs sepal width.
x = samples[:,0]
y = samples[:,1]
#print(x)
print("======")
#print(y)
print(iris.target)
# iris.target will o/p list of 0,1,2 == which represents on of the

# step 1 - Place k random centroids for the initial clusters.
# Number of clusters
k=3
```

```
# Create x coordinates of k random centroids, 3(k) random points ar

centroids_x = np.random.uniform(min(x), max(x), size=k)
# Create y coordinates of k random centroids
centroids_y = np.random.uniform(min(y), max(y), size=k)

centroids = np.array(list(zip(centroids_x, centroids_y)))
#print(centroids_x)
#print(centroids_y)
print(centroids)
plt.scatter(x, y, alpha=0.5)
plt.scatter(centroids_x, centroids_y)

plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')

plt.show()

# Create centroids array

# Make a scatter plot of x, y

# Make a scatter plot of the centroids

# Display plot
```

```
{'data': 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],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

In [10]:

```
#2.Assign data samples to the nearest centroid.

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()
```

```
iris = datasets.load_iris()

samples = iris.data
#print(samples)

x = samples[:,0]
y = samples[:,1]

sepal_length_width = np.array(list(zip(x, y)))

# Step 1: Place K random centroids

k = 3

centroids_x = np.random.uniform(min(x), max(x), size=k)
centroids_y = np.random.uniform(min(y), max(y), size=k)

centroids = np.array(list(zip(centroids_x, centroids_y)))

# Step 2: Assign samples to nearest centroid

# Distance formula

def distance(a, b):
  one = (a[0] - b[0]) ** 2
  two = (a[1] - b[1]) ** 2
  distance = (one+two) ** 0.5
  return distance



# Cluster labels for each point (either 0, 1, or 2)
labels = np.zeros(len(samples))
# A function that assigns the nearest centroid to a sample
def assign_to_centroid(sample, centroids):
  k = len(centroids)
  distances = np.zeros(k)
  for i in range(k):
    distances[i] = distance(sample, centroids[i])
  closest_centroid = np.argmin(distances)
  return closest_centroid
# Assign the nearest centroid to each sample

for i in range(len(samples)):
  labels[i] = assign_to_centroid(samples[i], centroids)

print(labels)


# Print labels
```

```
[2. 2. 2. 2. 2. 0. 2. 2. 2. 2. 0. 2. 2. 2. 0. 0. 0. 2. 0. 2. 2. 2.
2. 2.
```

```
 2. 2. 2. 2. 2. 2. 2. 2. 0. 0. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2.
 2. 2. 1. 0. 1. 2. 2. 2. 0. 2. 1. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2.
2. 2.
 2. 2. 2. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 0. 1. 2. 2. 2. 2. 2. 2. 2.
2. 2.
 2. 2. 2. 2. 0. 2. 1. 2. 1. 1. 2. 1. 2. 1. 0. 2. 1. 2. 2. 0. 1. 1.
1. 2.
 1. 2. 1. 2. 1. 1. 2. 2. 2. 1. 1. 1. 2. 2. 2. 1. 0. 0. 2. 1. 1. 1.
2. 1.
 1. 1. 2. 1. 0. 2.]
```

In [11]:

```
#step 3 (not fully clear)

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from copy import deepcopy

iris = datasets.load_iris()

samples = iris.data
samples = iris.data

x = samples[:,0]
y = samples[:,1]

sepal_length_width = np.array(list(zip(x, y)))
#print(sepal_length_width[2])

# Step 1: Place K random centroids

k = 3

centroids_x = np.random.uniform(min(x), max(x), size=k)
centroids_y = np.random.uniform(min(y), max(y), size=k)

centroids = np.array(list(zip(centroids_x, centroids_y)))
print("+++++++")
#print(centroids)

# Step 2: Assign samples to nearest centroid

def distance(a, b):
  one = (a[0] - b[0]) **2
  two = (a[1] - b[1]) **2
  distance = (one+two) ** 0.5
  return distance

# Cluster labels for each point (either 0, 1, or 2)
labels = np.zeros(len(samples))

# Distances to each centroid
distances = np.zeros(k)
```

```
distances = np.zeros(k)

for i in range(len(samples)):
  distances[0] = distance(sepal_length_width[i], centroids[0])
  distances[1] = distance(sepal_length_width[i], centroids[1])
  distances[2] = distance(sepal_length_width[i], centroids[2])
  cluster = np.argmin(distances)
  #print(cluster)
  labels[i] = cluster

#print(labels)

# Step 3: Update centroids
centroids_old = deepcopy(centroids)
for i in range(k):
  points = [sepal_length_width[j] for j in range(len(sepal_length_w
  centroids[i] = np.mean(points, axis=0)
#print(points)

print(centroids_old)
print("- - - - - - - - - - - - - -")
print(centroids)
```

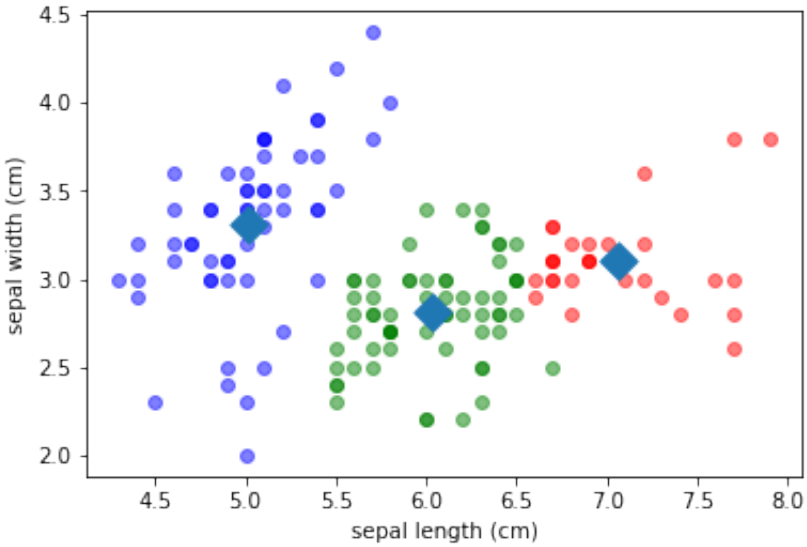
```
+++++++
[[6.82999077 3.03829138]
 [5.0765316  3.811242  ]
 [6.41067319 2.57679171]]
- - - - - - - - - - - - - -
[[6.85609756 3.13414634]
 [5.01454545 3.35454545]
 [5.91851852 2.6962963 ]]
```

In [12]:

```
#step4
#import codecademylib3_seaborn
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from copy import deepcopy

iris = datasets.load_iris()

samples = iris.data

x = samples[:,0]
y = samples[:,1]

sepal_length_width = np.array(list(zip(x, y)))

# Step 1: Place K random centroids

k = 3

centroids_x = np.random.uniform(min(x), max(x), size=k)
centroids_y = np.random.uniform(min(y), max(y), size=k)
```

```
centroids = np.array(list(zip(centroids_x, centroids_y)))

def distance(a, b):
  one = (a[0] - b[0]) ** 2
  two = (a[1] - b[1]) ** 2
  distance = (one + two) ** 0.5
  return distance

# A function that assigns the nearest centroid to a sample
def assign_to_centroid(sample, centroids):
  k = len(centroids)
  distances = np.zeros(k)
  for i in range(k):
    distances[i] = distance(sample, centroids[i])
  closest_centroid = np.argmin(distances)
  return closest_centroid

# To store the value of centroids when it updates
centroids_old = np.zeros(centroids.shape)

# Cluster labeles (either 0, 1, or 2)
labels = np.zeros(len(samples))

distances = np.zeros(3)

# Initialize error:
error = np.zeros(3)

for i in range(k):
  error[i] = distance(centroids[i], centroids_old[i])

# Repeat Steps 2 and 3 until convergence:

while error.all() != 0:

  # Step 2: Assign samples to nearest centroid

  for i in range(len(samples)):
    labels[i] = assign_to_centroid(samples[i], centroids)

  # Step 3: Update centroids

  centroids_old = deepcopy(centroids)

  for i in range(k):
    points = [sepal_length_width[j] for j in range(len(sepal_length
    centroids[i] = np.mean(points, axis=0)
    error[i] = distance(centroids[i], centroids_old[i])

colors = ['r', 'g', 'b']

for i in range(k):
  points = np.array([sepal_length_width[j] for j in range(len(sampl
```

```
  plt.scatter(points[:, 0], points[:, 1], c=colors[i], alpha=0.5)

plt.scatter(centroids[:, 0], centroids[:, 1], marker='D', s=150)

plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')

plt.show()
```



In [13]:
```
# using sklearn
```

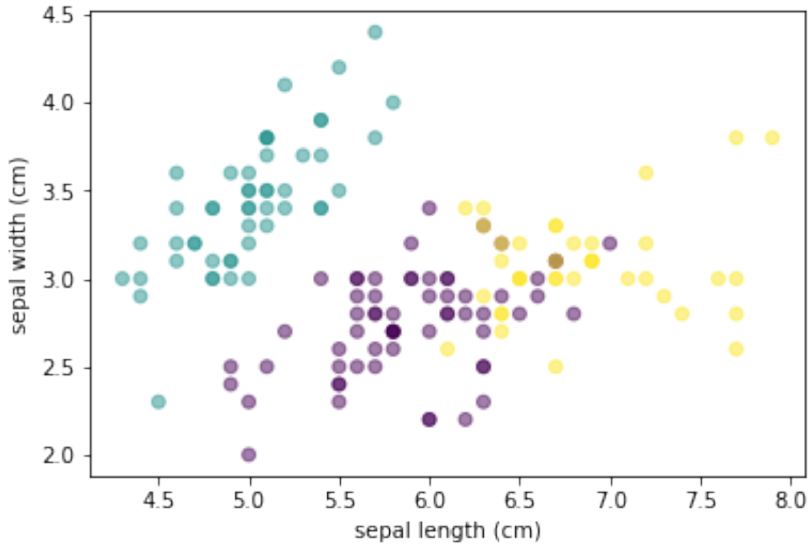
In [22]:
```
#import codecademylib3_seaborn
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans

# From sklearn.cluster, import Kmeans class

iris = datasets.load_iris()

samples = iris.data
model = KMeans(n_clusters = 3)
model.fit(samples)
labels = model.predict(samples)

print(labels)

# Use KMeans() to create a model that finds 3 clusters

# Use .fit() to fit the model to samples

# Use .predict() to determine the labels of samples

# Print the labels
new_samples = np.array([[5.7, 4.4, 1.5, 0.4],
   [6.5, 3. , 5.5, 0.4],
   [5.8, 2.7, 5.1, 1.9]])



# Predict labels for the new_samples

new_labels = model.predict(new_samples)

print(new_labels)

new_names = [iris.target_names[label] for label in new_labels]
print(new_names)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0
 2 0]
[1 0 0]
['versicolor', 'setosa', 'setosa']
```

In [21]:
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
```

```
iris = datasets.load_iris()

samples = iris.data
#print(samples)
print("================================")

model = KMeans(n_clusters=3)

model.fit(samples)

labels = model.predict(samples)

print(labels)

x = samples[:,0]
y = samples[:,1]

plt.scatter(x, y, c=labels, alpha=0.5)

plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')

plt.show()
```

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



In [25]:
```
import matplotlib.pyplot as plt
```

```
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd

iris = datasets.load_iris()

samples = iris.data

target = iris.target
#print(target)

model = KMeans(n_clusters=3)

model.fit(samples)
print(iris.target_names)
print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$")

labels = [iris.target_names[s] for s in model.predict(samples)]
print(labels)
print("==================================")

# Code starts here:
#actuals i think
species = [iris.target_names[t] for t in list(target)]
print(species)

df = pd.DataFrame({'labels': labels, 'species': species})

print(df)

ct = pd.crosstab(df['labels'], df['species'])
print(ct)
```

```
['setosa' 'versicolor' 'virginica']
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'seto
sa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', '
setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa
', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'se
tosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setos
a', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 's
etosa', 'setosa', 'setosa', 'setosa', 'versicolor', 'versicolor',
'virginica', 'versicolor', 'versicolor', 'versicolor', 'versicolor
', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versic
olor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 've
rsicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolo
r', 'versicolor', 'virginica', 'versicolor', 'versicolor', 'versic
olor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 've
rsicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolo
r', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versi
```

```
color', 'virginica', 'versicolor', 'virginica', 'virginica', 'virg
inica', 'virginica', 'versicolor', 'virginica', 'virginica', 'virg
inica', 'virginica', 'virginica', 'virginica', 'versicolor', 'vers
icolor', 'virginica', 'virginica', 'virginica', 'virginica', 'vers
icolor', 'virginica', 'versicolor', 'virginica', 'versicolor', 'vi
rginica', 'virginica', 'versicolor', 'versicolor', 'virginica', 'v
irginica', 'virginica', 'virginica', 'virginica', 'versicolor', 'v
irginica', 'virginica', 'virginica', 'virginica', 'versicolor', 'v
irginica', 'virginica', 'virginica', 'versicolor', 'virginica', 'v
irginica', 'virginica', 'versicolor', 'virginica', 'virginica', 'v
ersicolor']
==================================
['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'seto
sa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', '
setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa
', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'se
tosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setos
a', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 's
etosa', 'setosa', 'setosa', 'setosa', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolo
r', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versi
color', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'v
ersicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor'
, 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versico
lor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'ver
sicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor',
'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolo
r', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versi
color', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'v
ersicolor', 'virginica', 'virginica', 'virginica', 'virginica', 'v
irginica', 'virginica', 'virginica', 'virginica', 'virginica', 'vi
rginica', 'virginica', 'virginica', 'virginica', 'virginica', 'vir
ginica', 'virginica', 'virginica', 'virginica', 'virginica', 'virg
inica', 'virginica', 'virginica', 'virginica', 'virginica', 'virgi
nica', 'virginica', 'virginica', 'virginica', 'virginica', 'virgin
ica', 'virginica', 'virginica', 'virginica', 'virginica', 'virgini
ca', 'virginica', 'virginica', 'virginica', 'virginica', 'virginic
a', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica
', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica'
]
         labels    species
0        setosa     setosa
1        setosa     setosa
2        setosa     setosa
3        setosa     setosa
4        setosa     setosa
..          ...        ...
145   virginica  virginica
146  versicolor  virginica
147   virginica  virginica
148   virginica  virginica
149  versicolor  virginica
```

```
[150 rows x 2 columns]
species     setosa  versicolor  virginica
labels
setosa          50           0          0
versicolor       0          48         14
virginica        0           2         36
```

In [26]:
```
# from the above we can see predicgtion is bad for Virginica
```

The Number of Clusters
At this point, we have grouped the Iris plants into 3 clusters. But suppose we didn’t know there are three species of Iris in the dataset, what is the best number of clusters? And how do we determine that?

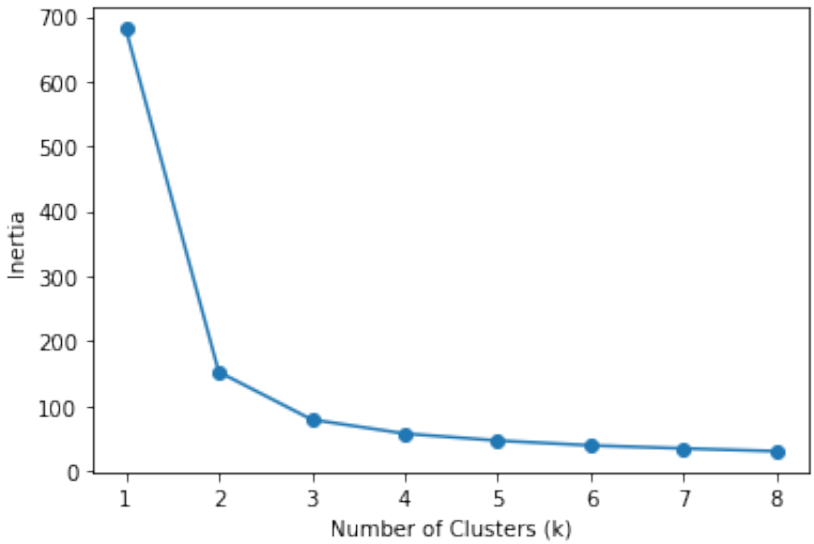
Before we answer that, we need to define what is a good cluster?

Good clustering results in tight clusters, meaning that the samples in each cluster are bunched together. How spread out the clusters are is measured by inertia. Inertia is the distance from each sample to the centroid of its cluster. The lower the inertia is, the better our model has done.

You can check the inertia of a model by:

```
print(model.inertia_)
```

In [27]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans

iris = datasets.load_iris()

samples = iris.data

num_clusters = list(range(1, 9))
inertias = []

for k in num_clusters:
  model = KMeans(n_clusters=k)
  model.fit(samples)
  inertias.append(model.inertia_)


plt.plot(num_clusters, inertias, '-o')

plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')

plt.show()
```



In [1]:
```
# there is another exercise , lets do it later .. spend sometime an
```

In [3]:
```
#import codecademylib3_seaborn
import numpy as np
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
model = KMeans(n_clusters=10, random_state=42)

digits = datasets.load_digits()
```

```
print(digits.DESCR)
print("============================")
print(digits.data)
print("============================")
print(digits.target)

plt.gray()

plt.matshow(digits.images[100])

plt.show()

print(digits.target[100])

# Figure size (width, height)

fig = plt.figure(figsize=(6, 6))

# Adjust the subplots

fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05,

# For each of the 64 images

for i in range(64):

    # Initialize the subplots: add a subplot in the grid of 8 by 8,

    ax = fig.add_subplot(8, 8, i+1, xticks=[], yticks=[])

    # Display an image at the i-th position

    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='

    # Label the image with the target value

    ax.text(0, 7, str(digits.target[i]))


plt.show()
model.fit(digits.data)
fig = plt.figure(figsize=(8, 3))

fig.suptitle('Cluser Center Images', fontsize=14, fontweight='bold'

for i in range(10):

  # Initialize subplots in a grid of 2X5, at i+1th position
  ax = fig.add_subplot(2, 5, 1 + i)

  # Display images
  ax.imshow(model.cluster_centers_[i].reshape((8, 8)),cmap=plt.cm.b
```

```
plt.show()
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the ran
ge 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits d
atasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Han
dwritten+Digits
(https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Ha
```

PCA serves an important role in many different parts of data science and analytics in general, as this process allows us to maximize the amount of information we can extract from data while reducing computational time down the line. We just saw a common use case for PCA with our pizza dataset. We took a higher dimensional dataset (5 dimensions in our case), and reduced it down to 2 dimensions. This two-dimensional dataset can now be an input to a variety of Machine Learning models. For example, we could use this new dataset as part of a forecasting model, or perform linear regression. These techniques would have been much more difficult prior to performing PCA.

PCA is also inherently an unsupervised learning algorithm and can be used to identify clusters in data on its own. Very similar to the popular k-means algorithms, PCA will look at overall similarities between the different features in a dataset. When we set the number of principal components to keep, we are defining the number of similar “rotations” of our dataset, which will act very much like a cluster of their own. Typically, many practitioners will implement PCA as a precursor to other clustering algorithms to augment the accuracy, but it is an interesting application to do clustering with PCA alone!

In [1]:
```
#PCA Problem
#Task 1 Observing the Dataset
```

In [3]:
```
import numpy as np
import pandas as pd
#import codecademylib3
import matplotlib.pyplot as plt
import seaborn as sns
```

```
  .

# ---------------
# Task 1: Drop nan values.
# ---------------

# Read the csv data as a DataFrame
df = pd.read_csv('./telescope_data.csv', index_col=0)

# Remove null and na values
df.dropna()

# Print the DataFrame head
print('Task 1:')
print(df.head())


# ---------------
# Task 2: Extract class column.
# ---------------
# Extract the class classes
classes = df['class']
data_matrix = df.drop(columns='class')

print('Task 2:')
print(data_matrix)

# ---------------
# Task 3: Create a correlation matrix.
# ---------------
# Use the `.corr()` method on `data_matrix` to get the correlation
correlation_matrix = data_matrix.corr()

ax = plt.axes()
sns.heatmap(correlation_matrix, cmap='Greens', ax=ax)
ax.set_title('Task 3:')
plt.show()


# ---------------
# Task 4: Perform eigendecomposition.
# ---------------
print('Task 4:')

# Perform eigendecomposition using `np.linalg.eig`
eigenvalues, eigenvectors = np.linalg.eig(correlation_matrix)
print(f'Eigenvalues length: {eigenvalues.size}, Original Number of

# Order the eigenvalues by ordering the indices of the eigenvalues
indices = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[indices]
eigenvectors = eigenvectors[:, indices]

print(eigenvalues.shape, eigenvectors.shape)
```

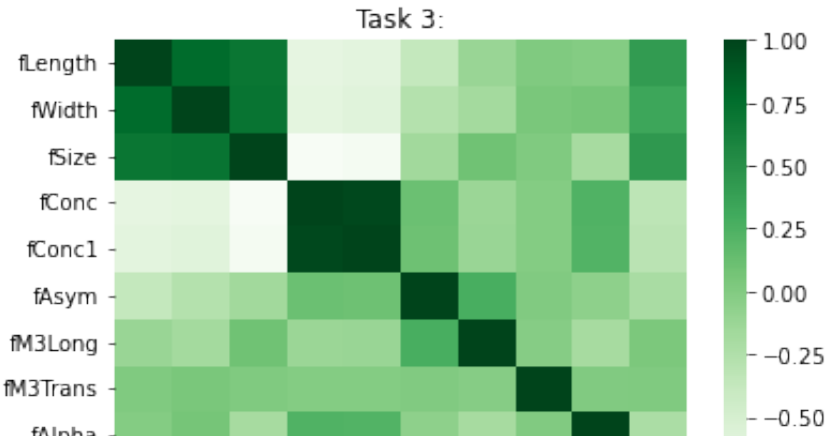
```
# ---------------
# Task 5: Find the variance/information percentages for each eigenv
# ---------------
# Find the percentages of information for each eigenvector, which i
information_proportions = eigenvalues / eigenvalues.sum()
information_percents = information_proportions * 100

# Plot the principal axes vs the information proportions for each p
plt.figure()
plt.plot(information_percents, 'ro-', linewidth=2)
plt.title('Task 5: Scree Plot')
plt.xlabel('Principal Axes')
plt.ylabel('Percent of Information Explained')
plt.show()


# ---------------
# Task 6: Find the cumulative variance/information percentages for
# ---------------
# Find the cumulative sum of the percentages
cumulative_information_percents = np.cumsum(information_percents)

# Plot the cumulative percentages array
plt.figure()
plt.plot(cumulative_information_percents, 'ro-', linewidth=2)

# Also plot a horizontal line indicating the 95% mark, and a vertic
plt.hlines(y=95, xmin=0, xmax=15)
plt.vlines(x=3, ymin=0, ymax=100)
plt.title('Task 6: Cumulative Information percentages')
plt.xlabel('Principal Axes')
plt.ylabel('Cumulative Proportion of Variance Explained')
plt.show()
```

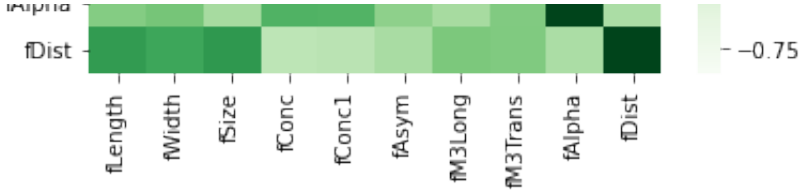
```
Task 1:
    fLength    fWidth   fSize   fConc  fConc1     fAsym  fM3Long
fM3Trans  \
0   28.7967   16.0021  2.6449  0.3918  0.1982   27.7004  22.0110
-8.2027
1   31.6036   11.7235  2.5185  0.5303  0.3773   26.2722  23.8238
-9.9574
2  162.0520  136.0310  4.0612  0.0374  0.0187  116.7410 -64.8580
-45.2160
3   23.8172    9.5728  2.3385  0.6147  0.3922   27.2107  -6.4633
-7.1513
4   75.1362   30.9205  3.1611  0.3168  0.1832   -5.5277  28.5525
21.8393

    fAlpha     fDist class
0  40.0920   81.8828     g
1   6.3609  205.2610     g
2  76.9600  256.7880     g
3  10.4490  116.7370     g
4   4.6480  356.4620     g
```

Task 2:

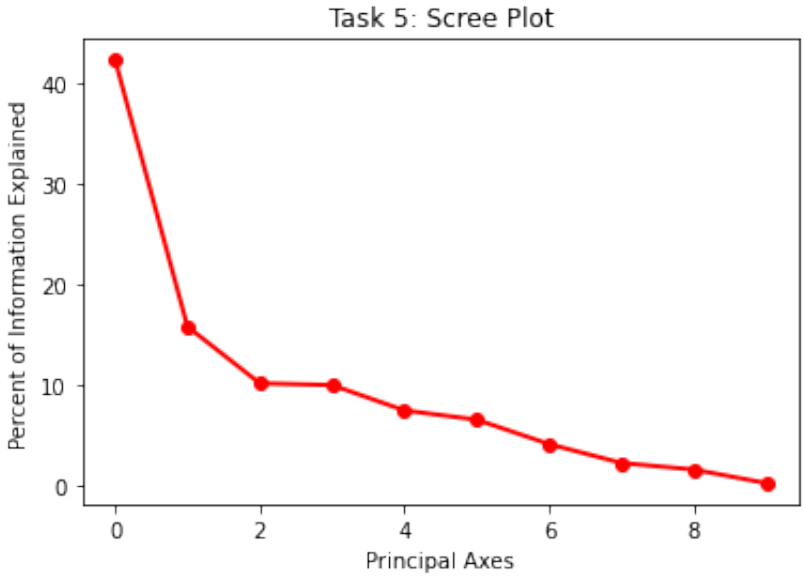
	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620
...	...	...	...	...	...	...	...	...	...	...
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174

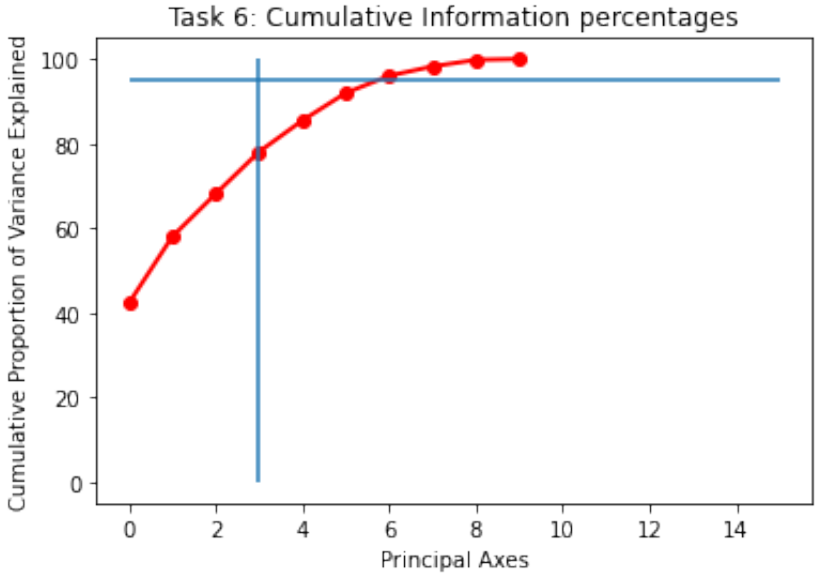
[19020 rows x 10 columns]






```
Task 4:
Eigenvalues length: 10, Original Number of Features: 10
(10,) (10, 10)
```





In [4]:
```
#Task 2 Performing PCA
```

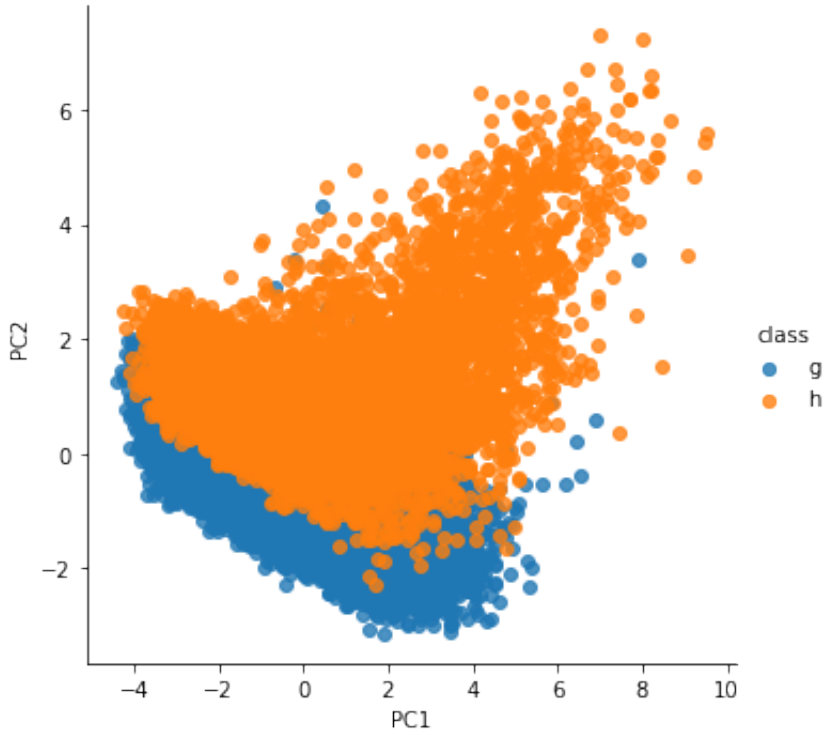
In [7]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
#import codecademylib3
import matplotlib.pyplot as plt
```

```
import seaborn as sns


data_matrix = pd.read_csv('./data_matrix.csv')
classes = pd.read_csv('./classes.csv', index_col=0)['class']

# ---------------
# Task 1: Calculate the standardized data matrix.
# ---------------
# Standardize the data matrix
mean = data_matrix.mean(axis=0)
sttd = data_matrix.std(axis=0)
data_matrix_standardized = (data_matrix - mean) / sttd


# ---------------
# Task 2: Perform PCA by fitting and transforming the data matrix.
# ---------------
# Find the principal components
pca = PCA()

# Fit the standardized data and calculate the principal components
principal_components = pca.fit_transform(data_matrix_standardized)
print(f'Number of features in the data matrix: {principal_component
print(f'Number of features in the principal components: {data_matri


# ---------------
# Task 3: Calculate the eigenvalues from the singular values and ex
# ---------------
# Find the eigenvalues from the singular values
singular_values = pca.singular_values_
eigenvalues = singular_values ** 2

# Eigenvectors are in the property `.components_` as row vectors. T
eigenvectors = pca.components_.T


# ---------------
# Task 4: Extract the variance ratios, which are equivalent to the
# ---------------
# Get the variance ratios from the `explained_variance_ratio_`
principal_axes_variance_ratios = pca.explained_variance_ratio_
principal_axes_variance_percents = principal_axes_variance_ratios *


# ---------------
# Task 5: Perform PCA once again but with 2 components
# ---------------
# Calculating principal components with 2 components
# Initialize a PCA object with 2 components
pca = PCA(n_components=2)

# Fit the standardized data and calculate the principal components
```

```
principal_components = pca.fit_transform(data_matrix_standardized)

# Print the DataFrame
print(f'Number of Principal Components Features: {principal_compone
print(f'Number of Original Data Features: {data_matrix_standardized


# ---------------
# Task 6: Plot the principal components and have its class as its h
# ---------------
# Plot the principal components as a scatterplot
principal_components_data = pd.DataFrame({
    'PC1': principal_components[:, 0],
    'PC2': principal_components[:, 1],
    'class': classes,
})

sns.lmplot(x='PC1', y='PC2', data=principal_components_data, hue='c
plt.show()


# We will use the one-hot-encoded classes as the y
y = classes.astype('category').cat.codes

# ---------------
# Task 7: Fit the transformed features onto the classifier and gene
# ---------------
# Get principal components with 2 features
# Perform PCA using 2 components
pca_1 = PCA(n_components=2)

# Use the principal components as X and split the data into 33% tes
X = pca_1.fit_transform(data_matrix_standardized)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create a Linear Support Vector Classifier
svc_1 = LinearSVC(random_state=0, tol=1e-5)
svc_1.fit(X_train, y_train)

# Generate a score for the testing data
score_1 = svc_1.score(X_test, y_test)
print(f'Score for model with 2 PCA features: {score_1}')

# ---------------
# Task 8: Now, fit the classifier with the first two feture of the
# ---------------
# Using the original features
# Select two features from the original data
first_two_original_features = [0, 1]
X_original = data_matrix_standardized.iloc[:, first_two_original_fe

# Split the data intro 33% testing and the rest training
X_train, X_test, y_train, y_test = train_test_split(X_original, y,
```

```
# Create a Linear Support Vector Classifier
svc_2 = LinearSVC(random_state=0)
svc_2.fit(X_train, y_train)

# Generate a score for the testing data
score_2 = svc_2.score(X_test, y_test)
print(f'Score for model with 2 randomly selected features: {score_2
```

```
Number of features in the data matrix: 11
Number of features in the principal components: 11
Number of Principal Components Features: 2
Number of Original Data Features: 11
```




```
Score for model with 2 PCA features: 0.8649036163772503
Score for model with 2 randomly selected features: 0.9993627529074
398

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed
to converge, increase the number of iterations.
  warnings.warn(
```

In [8]:
```
#PCA using Python
```

In [24]:
```
import pandas as pd
#import codecademylib3

# Read the csv data as a DataFrame
df = pd.read_csv('./Dry_Bean.csv')
print(df.head())

# Remove null and na values
```

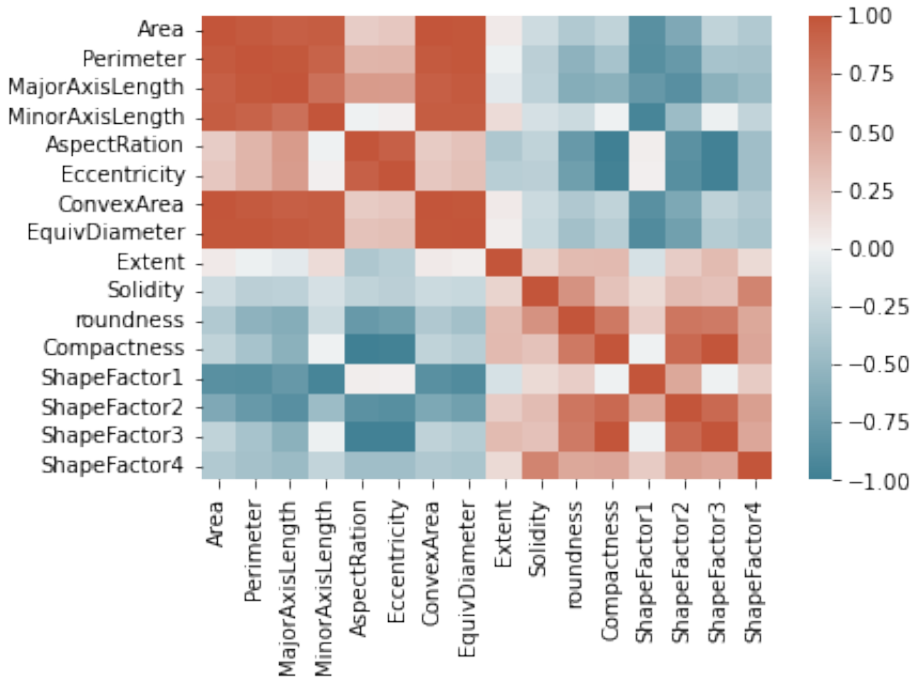
```
df.dropna()

# 1. Print the DataFrame head


# 2. Extract the numerical columns
data_matrix = df.drop(columns='Class')

# Extract the classes
classes = df['Class']
data_matrix.to_csv('data_matrix.csv', index=False)
```

```
    Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRatio
n  \
0  28395    610.291       208.178117       173.888747      1.19719
1
1  28734    638.018       200.524796       182.734419      1.09735
6
2  29380    624.110       212.826130       175.931143      1.20971
3
3  30008    645.884       210.557999       182.516516      1.15363
8
4  30140    620.134       201.847882       190.279279      1.06079
8

   Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  ro
undness  \
0      0.549812       28715     190.141097  0.763923  0.988856   0
.958027
1      0.411785       29172     191.272751  0.783968  0.984986   0
.887034
2      0.562727       29690     193.410904  0.778113  0.989559   0
.947849
3      0.498616       30724     195.467062  0.782681  0.976696   0
.903936
4      0.333680       30417     195.896503  0.773098  0.990893   0
.984877

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFac
tor4  Class
0     0.913358      0.007332      0.003147      0.834222      0.99
8724  SEKER
1     0.953861      0.006979      0.003564      0.909851      0.99
8430  SEKER
2     0.908774      0.007244      0.003048      0.825871      0.99
9066  SEKER
3     0.928329      0.007017      0.003215      0.861794      0.99
4199  SEKER
4     0.970516      0.006697      0.003665      0.941900      0.99
9166  SEKER
```

In [25]:

```
data_matrix = pd.read_csv('./data_matrix.csv')
```

```
# 1. Use the `.corr()` method on `data_matrix` to get the correlati
correlation_matrix = data_matrix.corr()

## Heatmap code:
red_blue = sns.diverging_palette(220, 20, as_cmap=True)
sns.heatmap(correlation_matrix, vmin = -1, vmax = 1, cmap=red_blue)
plt.show()

# 2. Perform eigendecomposition using `np.linalg.eig`
eigenvalues, eigenvectors = np.linalg.eig(correlation_matrix)


# 3. Print out the eigenvectors and eigenvalues
print('eigenvectors: ')
print(eigenvectors)

print('eigenvalues: ')
print(eigenvalues)

#dummy = pd.DataFrame(eigenvalues)

# save the dataframe as a csv file
#dummy.to_csv("eigenvalues.csv", index=False)
```

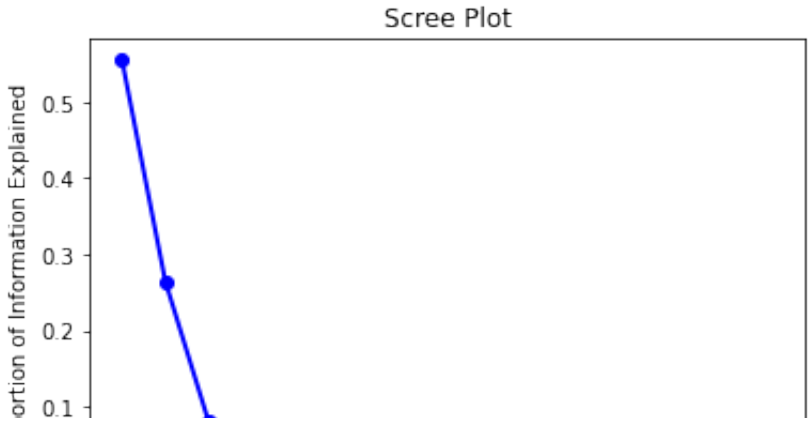


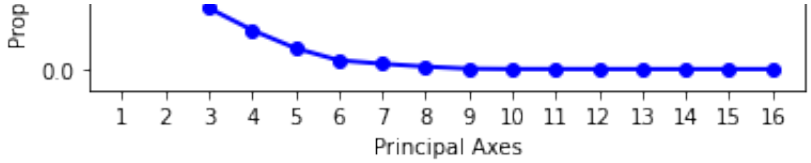

```
eigenvectors:
[[ 2.82457959e-01  2.45882017e-01 -6.14466787e-02 -3.15461931e-02
  -9.13256234e-02 -3.66390029e-01  1.25044861e-01  7.17479179e-02
   3.50665669e-02 -3.90419516e-01 -1.77686475e-01  5.44842282e-02
   4.62948861e-02  6.55727948e-01  2.31435926e-01  1.33190281e-01]
 [ 3.10891123e-01  1.79302922e-01 -1.88525952e-02 -4.24678975e-02
   8.18198663e-02 -1.02508210e-02  8.15296990e-02  3.17295058e-02
  -1.57501171e-01  3.44383066e-01  1.99453621e-01 -7.50549982e-01
   3.17920275e-01  8.13901113e-02  1.46143834e-02  1.26584691e-02]
 [ 3.25823976e-01  1.00756516e-01 -8.46919067e-02 -6.79308126e-03
```

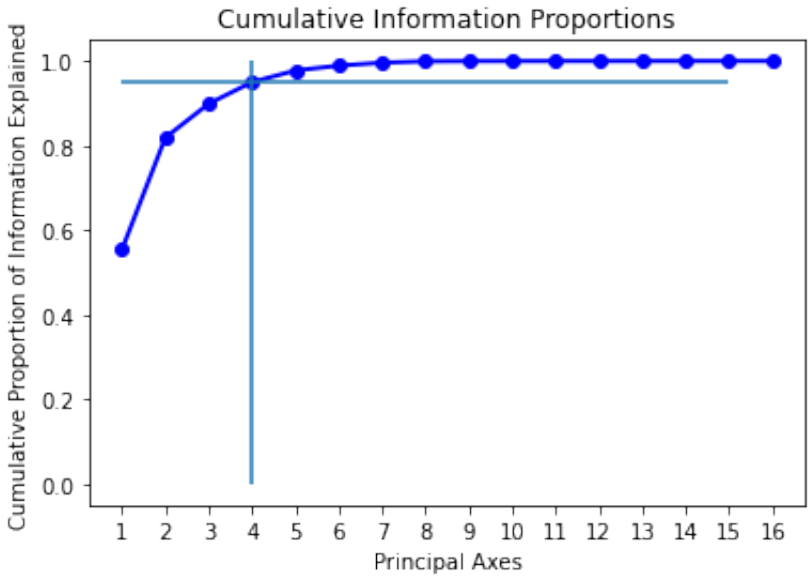
```
 -4.42163116e-02 -1.49091929e-02  1.18162546e-01 -2.00947006e-01
 -3.52366452e-01  1.01996482e-01  1.73639683e-01  2.73549959e-02
 -6.85301970e-01 -1.86251185e-01  3.46019418e-01  1.74431583e-01]
[ 2.36199383e-01  3.43460651e-01  7.50039030e-03 -6.12997105e-02
 -4.29258549e-03 -2.78820146e-02 -6.23528140e-02  9.47252766e-02
  4.14230636e-01  4.81150315e-01  4.73720993e-03  4.13935449e-01
  2.58014714e-01 -1.83095642e-01  3.31749325e-01  1.55445406e-01]
[ 2.29298328e-01 -3.30844185e-01 -1.69058011e-01  5.36461191e-02
 -2.47566532e-02  7.59699103e-02  3.67891855e-01 -5.29805906e-01
  1.21518443e-01  2.08046178e-01 -5.40202985e-01 -3.34425476e-03
  8.74672429e-02  2.66661427e-02 -1.23575706e-01  1.02810024e-01]
[ 2.31526055e-01 -3.19433875e-01 -1.63042022e-01  1.18388551e-01
 -6.03039593e-02 -1.90427992e-01 -5.11031662e-01  4.09120604e-01
 -3.22591880e-01  2.41533155e-01 -4.11402535e-01  3.41133220e-02
 -2.23275476e-02  4.59487115e-05  1.47926800e-02 -4.23063139e-02]
[ 2.83199889e-01  2.44630067e-01 -5.36490752e-02 -3.09595575e-02
 -8.91133649e-02 -3.69215707e-01  1.21178732e-01  6.45192756e-02
  2.58278807e-02 -3.94275552e-01 -1.79002314e-01 -7.73945617e-02
  1.02365543e-01 -6.48621711e-01 -2.24752670e-01 -1.30973875e-01]
[ 2.97483844e-01  2.22802185e-01 -4.99135477e-02 -3.24273855e-02
 -2.19534105e-02 -3.35147364e-02 -6.30912872e-04 -3.40422281e-02
 -1.66940989e-02  2.53700143e-01  1.96001889e-01  2.74139585e-01
 -1.98496927e-01  2.66409059e-01 -6.75588993e-01 -3.32487860e-01]
[-5.98079606e-02  2.20619259e-01 -8.52582080e-02  9.48254269e-01
  1.97598918e-01  5.10553897e-04  4.45929047e-02 -1.47000878e-02
  6.00988144e-03 -8.95631344e-04 -2.77069112e-03 -4.38701370e-05
  4.62379038e-04 -5.65700266e-05 -4.74695176e-06 -1.17646776e-06]
[-1.43016314e-01  1.03322337e-01 -7.38670228e-01 -4.95457556e-02
 -2.82194373e-01  3.25692613e-01  3.09528792e-01  3.72834092e-01
 -1.24670321e-02 -6.14713024e-03  8.92367231e-04  3.39336586e-04
  5.39707165e-04 -7.67128823e-03 -2.11537089e-03 -1.38727449e-03]
[-2.48164811e-01  2.14805282e-01 -1.63325487e-01  6.74824148e-02
 -6.48700706e-01 -1.73439085e-01 -4.16624414e-01 -4.61145752e-01
  9.44150591e-03  7.30163366e-02  3.45701555e-02 -1.20891182e-01
  4.96397316e-02  1.46685284e-02  1.89361431e-03  2.37532217e-03]
[-2.38378001e-01  3.28914360e-01  1.49700768e-01 -8.71555716e-02
  5.85957324e-02  1.23232305e-02 -3.24244642e-02  1.67809467e-01
  3.93833779e-02  1.17265401e-01 -3.22502975e-01 -1.53253508e-01
 -2.54641355e-01  2.02577299e-03 -3.72516051e-01  6.52515601e-01]
[-2.21318903e-01 -3.32548514e-01 -3.26229309e-02  7.23303405e-02
 -1.12907779e-01 -6.33211910e-01  2.93567734e-01  1.91922106e-01
  3.36948538e-01  2.71583838e-01  2.00505414e-01 -1.28472846e-01
 -2.30735279e-01  4.01059407e-03 -1.12850106e-02 -5.46798048e-03]
[-3.14624593e-01  1.29419241e-01  1.20076675e-01 -4.65438196e-02
 -2.64141427e-02 -2.59245737e-01  3.54310851e-01 -3.19852488e-02
 -6.59508821e-01  2.12589947e-01 -3.73372481e-02  2.97904717e-01

  3.19900059e-01 -1.85035834e-02  1.46683912e-02  5.54470208e-03]
[-2.38983301e-01  3.27521662e-01  1.49570241e-01 -9.56788529e-02
  6.22269463e-02  4.74498436e-02  8.31848115e-02  4.52594153e-02
  9.00540674e-02  1.65952214e-01 -4.68319821e-01 -1.89335830e-01
 -2.82232572e-01  4.61837470e-02  2.34064701e-01 -6.01334519e-01]
[-1.98009429e-01  1.00061082e-01 -5.36903055e-01 -2.10119897e-01
  6.40371689e-01 -2.80088867e-01 -2.40046279e-01 -2.66748910e-01
```

```
   2.97986049e-03 -3.46314997e-03  2.84785031e-02  9.79933869e-03
   2.73100957e-03 -4.29724599e-03  1.03771345e-02  6.16002750e-04]
]
eigenvalues:
[8.87463018e+00 4.22895571e+00 1.28105028e+00 8.18252847e-01
 4.38286865e-01 1.83961749e-01 1.11624116e-01 5.20132000e-02
 8.26026072e-03 1.45388993e-03 1.05418870e-03 2.93982938e-04
 1.48794566e-04 1.00102669e-05 1.78479175e-06 2.14611337e-06]
```

In [26]:
```
eigenvalues = pd.read_csv('eigenvalues.csv')['eigenvalues'].values

# 1. Find the proportion of information for each eigenvector, which
info_prop = eigenvalues / eigenvalues.sum()
#print("==========")
#print(info_prop)

## Plot the principal axes vs the information proportions for each

plt.plot(np.arange(1,len(info_prop)+1),info_prop, 'bo-', linewidth=
plt.title('Scree Plot')
plt.xlabel('Principal Axes')
plt.xticks(np.arange(1,len(info_prop)+1))
plt.ylabel('Proportion of Information Explained')
plt.show()
plt.clf()

# 2. Find the cumulative sum of the proportions
cum_info_prop = np.cumsum(info_prop)
#print("===============")
#print(cum_info_prop)

## Plot the cumulative proportions array

plt.plot(np.arange(1,len(info_prop)+1), cum_info_prop, 'bo-', linew
plt.hlines(y=.95, xmin=1, xmax=15)
plt.vlines(x=4, ymin=0, ymax=1)
plt.title('Cumulative Information Proportions')
plt.xlabel('Principal Axes')
plt.xticks(np.arange(1,len(info_prop)+1))
plt.ylabel('Cumulative Proportion of Information Explained')
plt.show()
```







In [29]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
#import codecademylib3

data_matrix = pd.read_csv('./data_matrix.csv')

# 1. Standardize the data matrix
mean = data_matrix.mean(axis=0)
sttd = data_matrix.std(axis=0)
data_matrix_standardized = (data_matrix - mean) / sttd
print(data_matrix_standardized.head())

# 2. Find the principal components
pca = PCA()
components = pca.fit(data_matrix_standardized).components_
components = pd.DataFrame(components).transpose()
components.index =  data_matrix.columns
print(components)

# 3. Calculate the variance/info ratios
var_ratio = pca.explained_variance_ratio_
var_ratio= pd.DataFrame(var_ratio).transpose()
print(var_ratio)

data_matrix_standardized.to_csv("data_matrix_standardized.csv", ind
```

```
       Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRa
tion  \
0 -0.840718  -1.143277        -1.306550        -0.631130     -1.56
4995
```

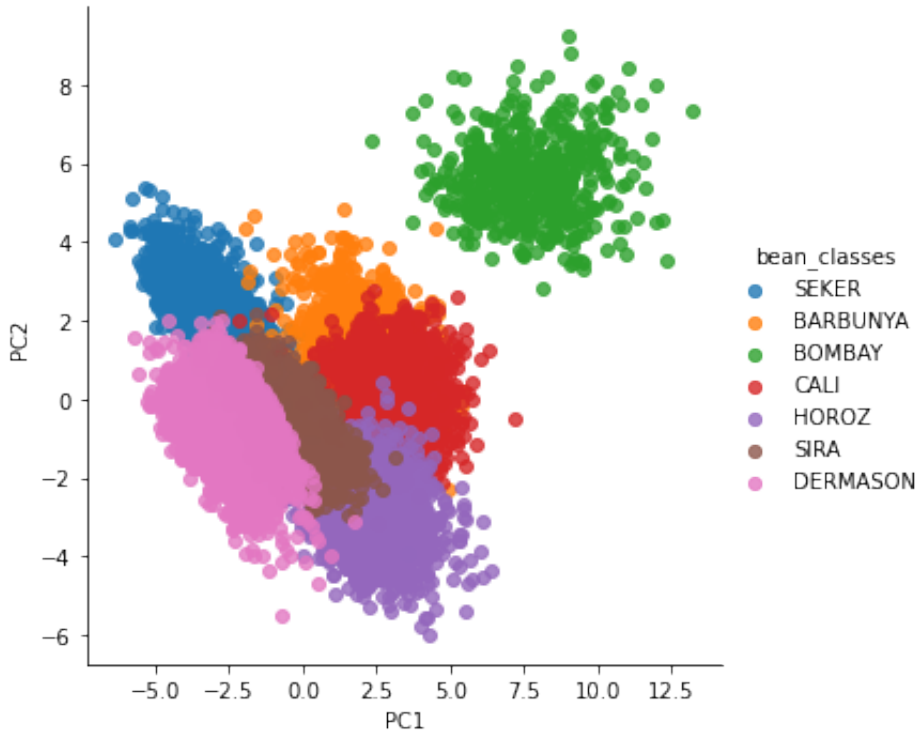
```
1 -0.829157  -1.013887        -1.395860        -0.434429     -1.96
9712
2 -0.807128  -1.078789        -1.252311        -0.585713     -1.51
4236
3 -0.785712  -0.977179        -1.278778        -0.439274     -1.74
1554
4 -0.781210  -1.097344        -1.380420        -0.266654     -2.11
7915

   Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  ro
undness  \
0     -2.185640   -0.841420      -1.063302  0.289077  0.367600   1
.423815
1     -3.685904   -0.826071      -1.044178  0.697451 -0.462889   0
.231046
2     -2.045261   -0.808674      -1.008047  0.578174  0.518398   1
.252819
3     -2.742110   -0.773947      -0.973301  0.671235 -2.241685   0
.515030
4     -4.534862   -0.784257      -0.966044  0.476003  0.804743   1
.874924

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFac
tor4
0     1.839049      0.680761      2.402084      1.925653      0.83
8340
1     2.495358      0.367953      3.100780      2.689603      0.77
1110
2     1.764778      0.603107      2.235009      1.841288      0.91
6721
3     2.081639      0.401703      2.514982      2.204169     -0.19
7978
4     2.765229      0.118264      3.270862      3.013352      0.93
9605
                        0         1         2         3         4
5   \
Area             0.282458  0.245882 -0.061447 -0.031546 -0.091326
0.366390
Perimeter        0.310891  0.179303 -0.018853 -0.042468  0.081820
0.010251
MajorAxisLength  0.325824  0.100757 -0.084692 -0.006793 -0.044216
0.014909
MinorAxisLength  0.236199  0.343461  0.007500 -0.061300 -0.004293
0.027882
AspectRation     0.229298 -0.330844 -0.169058  0.053646 -0.024757
-0.075970
Eccentricity     0.231526 -0.319434 -0.163042  0.118389 -0.060304
0.190428
ConvexArea       0.283200  0.244630 -0.053649 -0.030960 -0.089113
0.369216
EquivDiameter    0.297484  0.222802 -0.049914 -0.032427 -0.021953
0.033515
Extent          -0.059808  0.220619 -0.085258  0.948254  0.197599
```

```
-0.000511
Solidity        -0.143016  0.103322 -0.738670 -0.049546 -0.282194
-0.325693
roundness       -0.248165  0.214805 -0.163325  0.067482 -0.648701
0.173439
Compactness     -0.238378  0.328914  0.149701 -0.087156  0.058596
-0.012323
ShapeFactor1    -0.221319 -0.332549 -0.032623  0.072330 -0.112908
0.633212
ShapeFactor2    -0.314625  0.129419  0.120077 -0.046544 -0.026414
0.259246
ShapeFactor3    -0.238983  0.327522  0.149570 -0.095679  0.062227
-0.047450
ShapeFactor4    -0.198009  0.100061 -0.536903 -0.210120  0.640372
0.280089

                       6         7         8         9         10
11  \
Area            0.125045  0.071748  0.035067  0.390420 -0.177686
0.054484
Perimeter       0.081530  0.031730 -0.157501 -0.344383  0.199454
-0.750550
MajorAxisLength  0.118163 -0.200947 -0.352366 -0.101996  0.173640
0.027355
MinorAxisLength -0.062353  0.094725  0.414231 -0.481150  0.004737
0.413935
AspectRation     0.367892 -0.529806  0.121518 -0.208046 -0.540203
-0.003344
Eccentricity    -0.511032  0.409121 -0.322592 -0.241533 -0.411403
0.034113
ConvexArea       0.121179  0.064519  0.025828  0.394276 -0.179002
-0.077395
EquivDiameter   -0.000631 -0.034042 -0.016694 -0.253700  0.196002
0.274140
Extent           0.044593 -0.014700  0.006010  0.000896 -0.002771
-0.000044
Solidity         0.309529  0.372834 -0.012467  0.006147  0.000892
0.000339
roundness       -0.416624 -0.461146  0.009442 -0.073016  0.034570
-0.120891
Compactness     -0.032424  0.167809  0.039383 -0.117265 -0.322503
-0.153254
ShapeFactor1     0.293568  0.191922  0.336949 -0.271584  0.200505
-0.128473
ShapeFactor2     0.354311 -0.031985 -0.659509 -0.212590 -0.037337
0.297905
ShapeFactor3     0.083185  0.045259  0.090054 -0.165952 -0.468320
-0.189336
ShapeFactor4    -0.240046 -0.266749  0.002980  0.003463  0.028479
0.009799

                       12        13        14        15
Area            -0.046295 -0.655728  0.133190  0.231436
```

```
Perimeter       -0.317920 -0.081390  0.012658  0.014614
MajorAxisLength  0.685302  0.186251  0.174432  0.346019
MinorAxisLength -0.258015  0.183096  0.155445  0.331749
AspectRation    -0.087467 -0.026666  0.102810 -0.123576
Eccentricity     0.022328 -0.000046 -0.042306  0.014793
ConvexArea      -0.102366  0.648622 -0.130974 -0.224753
EquivDiameter    0.198497 -0.266409 -0.332488 -0.675589
Extent          -0.000462  0.000057 -0.000001 -0.000005
Solidity        -0.000540  0.007671 -0.001387 -0.002115
roundness       -0.049640 -0.014669  0.002375  0.001894
Compactness      0.254641 -0.002026  0.652516 -0.372516
ShapeFactor1     0.230735 -0.004011 -0.005468 -0.011285
ShapeFactor2    -0.319900  0.018504  0.005545  0.014668
ShapeFactor3     0.282233 -0.046184 -0.601335  0.234065
ShapeFactor4    -0.002731  0.004297  0.000616  0.010377
         0        1         2         3         4         5
6   \
0  0.554664  0.26431  0.080066  0.051141  0.027393  0.011498  0.00
6977

         7         8         9         10        11        12
13  \
0  0.003251  0.000516  0.000091  0.000066  0.000018  0.000009  6.2
56417e-07

              14            15
0  1.341321e-07  1.115495e-07
```

In [31]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
#import codecademylib3
import matplotlib.pyplot as plt
import seaborn as sns

data_matrix_standardized = pd.read_csv('./data_matrix_standardized.
classes = pd.read_csv('./classes.csv')['Class']

# 1. Transform the data into 4 new features using the first PCs
pca = PCA(n_components = 4)
data_pcomp = pca.fit_transform(data_matrix_standardized)
data_pcomp = pd.DataFrame(data_pcomp)
data_pcomp.columns = ['PC1', 'PC2', 'PC3', 'PC4']
print(data_pcomp.head())

## 2. Plot the first two principal components colored by the bean c

data_pcomp['bean_classes'] = classes
sns.lmplot(x='PC1', y='PC2', data=data_pcomp, hue='bean_classes', f
plt.show()
```

```
        PC1       PC2       PC3       PC4
0 -4.981378  1.824630  0.748993 -0.390797
1 -5.436593  2.932257  2.182294 -0.431944
2 -4.757913  1.826817  0.514019 -0.125849
3 -4.300383  2.003587  3.554316  0.082961
4 -6.349107  4.088055  1.179156 -0.830327
```



In [32]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split


data_matrix_standardized = pd.read_csv('./data_matrix_standardized.
classes = pd.read_csv('./classes.csv')

# We will use the classes as y
y = classes.Class.astype('category').cat.codes

# Get principal components with 4 features and save as X
pca_1 = PCA(n_components=4)
X = pca_1.fit_transform(data_matrix_standardized)

# Split the data into 33% testing and the rest training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Create a Linear Support Vector Classifier
svc_1 = LinearSVC(random_state=0, tol=1e-5)
svc_1.fit(X_train, y_train)

# Generate a score for the testing data
score_1 = svc_1.score(X_test, y_test)
print(f'Score for model with 4 PCA features: {score_1}')

# Split the original data intro 33% testing and the rest training
X_train, X_test, y_train, y_test = train_test_split(data_matrix_sta

# Create a Linear Support Vector Classifier
svc_2 = LinearSVC(random_state=0)
svc_2.fit(X_train, y_train)

# Generate a score for the testing data
score_2 = svc_2.score(X_test, y_test)
print(f'Score for model with original features: {score_2}')
```

```
/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed
to converge, increase the number of iterations.
  warnings.warn(

Score for model with 4 PCA features: 0.847506678539626
Score for model with original features: 0.9169634906500446

/Users/kamallakannansekar/opt/anaconda3/lib/python3.9/site-package
s/sklearn/svm/_base.py:1206: ConvergenceWarning: Liblinear failed
to converge, increase the number of iterations.
  warnings.warn(
```

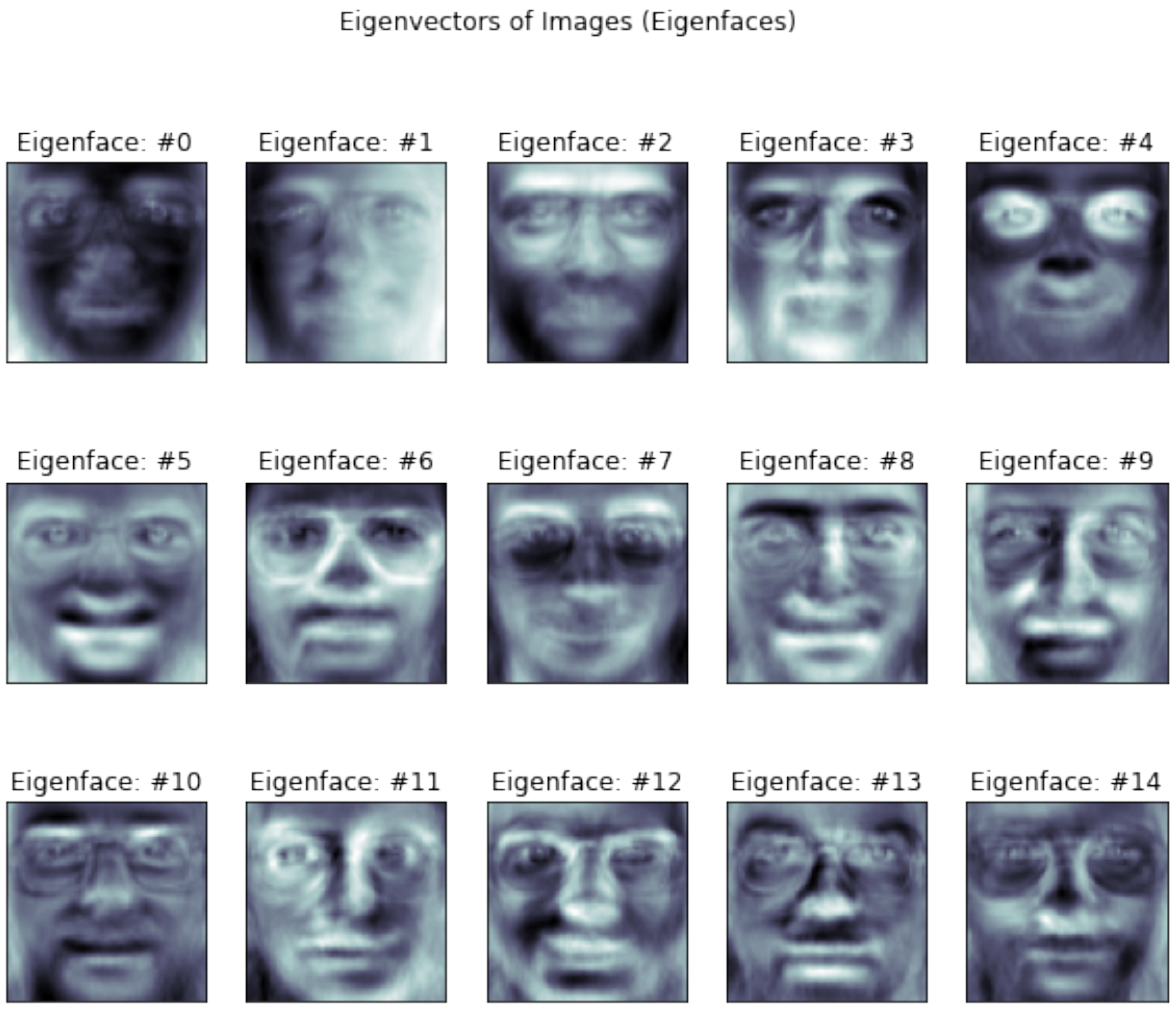
In [ ]:

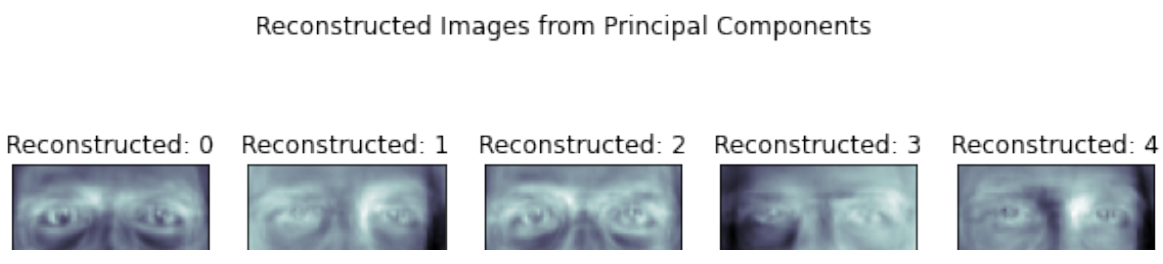
In [36]:
```
import numpy as np
from sklearn import datasets
#import codecademylib3
import matplotlib.pyplot as plt


# Download the data from sklearn's datasets
faces = datasets.fetch_olivetti_faces()['data']

# 1. Standardize the images using the mean and standard deviation
faces_mean = faces.mean(axis=0)
faces_std = faces.std(axis=0)
faces_standardized = (faces - faces_mean) / faces_std
dummy = pd.DataFrame(faces_standardized)

# save the dataframe as a csv file
#dummy.to_csv("eigenvalues.csv", index=False)
dummy.to_csv("faces_standardized.csv", index=False)
# 2. Find the number of features per image
n_images, n_features = faces_standardized.shape
side_length = int(np.sqrt(n_features))
print(f'Number of features(pixels) per image: {n_features}')
print(f'Square image side length: {side_length}')


# 3. Plot the images
# Create an empty 10x8 plot
fig = plt.figure(figsize=(10, 8))

# Observe the first 15 images.
for i in range(15):

    # Create subplot, remove x and y ticks, and add title
    ax = fig.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    ax.set_title(f'Image of Face: #{i}')

    # Get an image from a row based on the current value of i
    face_image = faces_standardized[i]

    # Reshape this image into side_length x side_length
    face_image_reshaped = face_image.reshape(side_length, side_leng

    # Show the image
    ax.imshow(face_image_reshaped, cmap=plt.cm.bone)
plt.show()
```

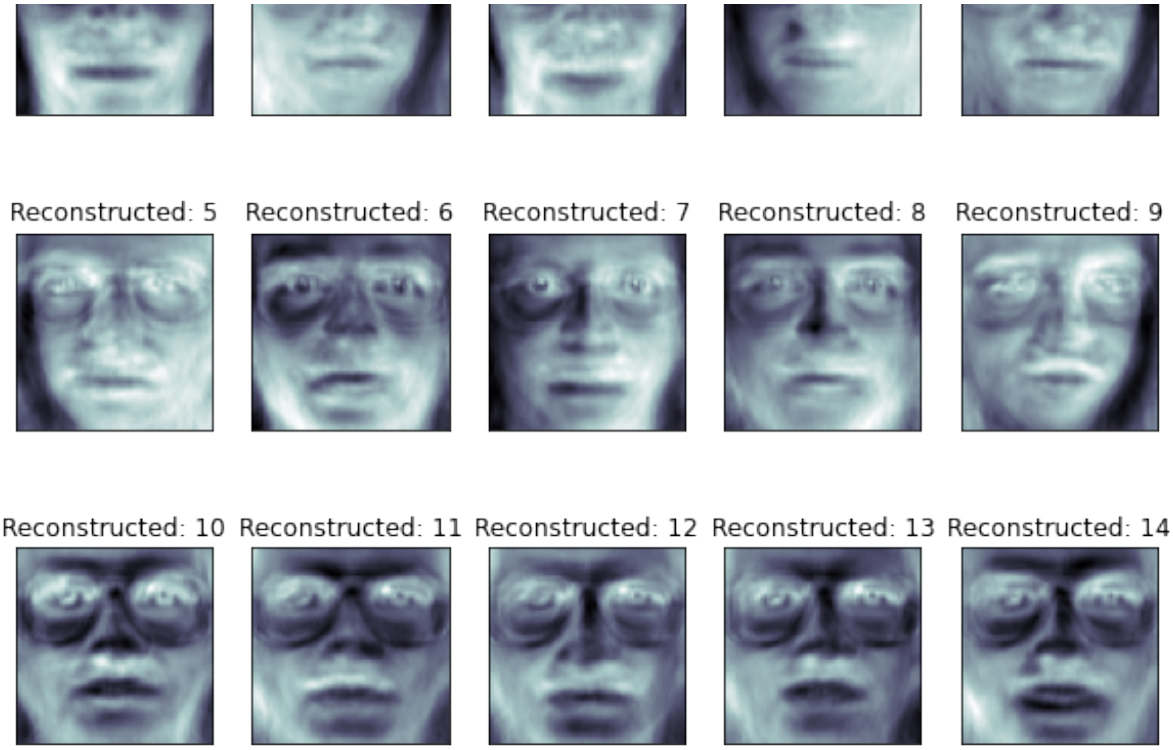
```
Number of features(pixels) per image: 4096
Square image side length: 64
```

In [37]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
#import codecademylib3
import matplotlib.pyplot as plt


faces_standardized = pd.read_csv('./faces_standardized.csv').values

# 1. Instantiate a PCA object and fit the standardized faces datase
pca = PCA(n_components=40)
pca.fit(faces_standardized)

# 2. Retrieve and plot eigenvectors (eigenfaces)
eigenfaces = pca.components_

fig = plt.figure(figsize=(10, 8))
fig.suptitle('Eigenvectors of Images (Eigenfaces)')
for i in range(15):
    # Create subplot, remove x and y ticks, and add title
    ax = fig.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    ax.set_title(f'Eigenface: #{i}')

    # Get an eigenvector from the current value of i
    eigenface = eigenfaces[i]

    # Reshape this image into 64x64 since the flattened shape was 4
    eigenface_reshaped = eigenface.reshape(64, 64)

    # Show the image
    ax.imshow(eigenface_reshaped, cmap=plt.cm.bone)
plt.show()

# 3. Reconstruct images from the compressed principal components
```

```
# 5. Reconstruct images from the compressed principal components
# The principal components are usually calculated using `faces_stan
principal_components = pca.transform(faces_standardized)

# The `inverse_transform` method allows for reconstruction of image
faces_reconstructed = pca.inverse_transform(principal_components)

# Plot the reconstructed images
fig = plt.figure(figsize=(10, 8))
fig.suptitle('Reconstructed Images from Principal Components')
for i in range(15):
    ax = fig.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    ax.set_title(f'Reconstructed: {i}')

    reconstructed_face = faces_reconstructed[i]
    reconstructed_face_reshaped = reconstructed_face.reshape(64, 64
    ax.imshow(reconstructed_face_reshaped, cmap=plt.cm.bone)
plt.show()
```






Reconstructed: 5
Reconstructed: 6
Reconstructed: 7
Reconstructed: 8
Reconstructed: 9
Reconstructed: 10
Reconstructed: 11
Reconstructed: 12
Reconstructed: 13
Reconstructed: 14

In [ ]: