Module 6 Assignment

kNN with bike frame geometries

General reminders about kNN:

- A type of pattern recognition algorithm
- A type of classification algorithm
- Non-parametric (that is, there's no output values like a coefficient in linear regression)
- Simple but has been successful in several areas, e.g. handwritten digits

In this homework, we're going to be using data on bicycles to try and predict which class a new bicycle fits into. We'll be using data on the geometry, or the shape, of bicycles. The below graphic labels the different parts of a bike's geometry. We don't need to know very much about bicycles to solve this problem, this is presented more for informational purposes.

All we need to know is that there are several different classes of bike, and each serves a different purpose. And each class of bike has a different geometry, or shape, and that this geometry can identify the class of a given bike. We'll use this information to build a classification model, specifically using kNN.

Bike Geo

1. Imports

Run the imports cells to get started!

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

2. Read in the csv data

We have data on several measurements of bike geometries for 56cm sized bikes. We need to stick with the same size bike across all observations otherwise we'll conflate differences in scale.

Use `Pandas` and `read_csv` to read in the data. Save it as a DataFrame.

In [2]:
```
# TODO read in the bike geo csv
# TODO read in the csv and recast the index to datetime format

bike_geometries = pd.read_csv('C:/Users/iamontes/Python Projects/Finance ML Certificat

print(bike_geometries.sample(10))
```

```
print(bike_geometries.shape)
```

```
          Brand    Year           Model  Size Steer Cat  Head Angle  \
65      Bianchi  1983.0      Randonneur  58.0       NaN        72.0   
140        Trek  1988.0             400  56.5      race        73.0   
305        Fuji  2006.0        Team Pro  56.0      race        73.5   
253      Miyata  1980.0    Gran Touring  58.0       NaN        73.0   
182        Trek  1977.0           TX900  56.0      race        73.0   
131     Mercian  1980.0         Unknown  61.0      race        73.0   
257      Gitane  1976.0  Tour de France  60.0       NaN        73.0   
50      Univega  1986.0    Gran Turismo  54.0      tour        72.0   
314  Cannondale  1995.0            R800  58.0       NaN        73.5   
123        Trek  2006.0         Pilot    56.0       NaN        72.5   

     Fork Offset  Seat Angle  Chain Stay  Wheelbase Top Tube BB Drop Trail   \
65          5.20        72.0        43.0      103.5       57     6.6   55.8   
140         4.50        73.5        43.0      102.0     56.4     7.2  56.89   
305         4.50        73.0        41.0       98.6       56     6.9  53.78   
253         4.80        73.0        42.8      102.9     56.5     NaN  53.76   
182         4.50        73.0        42.0      100.3       56       6  56.89   
131         4.30        73.0        42.5      102.5       58     NaN  58.98   
257         6.35        73.0        41.9      101.6     57.8     NaN  37.55   
50          6.00        72.0        44.0        NaN      NaN     NaN  47.38   
314         4.12        74.0        40.6      101.0     57.5     NaN  57.74   
123         4.50        73.3        41.5      100.2     56.2     NaN  60.02   

      Flop  
65    16.4  
140  15.91  
305  14.65  
253  15.03  
182  15.91  
131  16.49  
257   10.5  
50   13.93  
314  15.72  
123  17.21  
(356, 14)
```

3. Exploratory Data Analysis

We've been being very prescriptive in the assignments so far, but now that you have more experience, it's time to start exploring a little bit on your own! In the next couple of cells, explore the dataset we just imported.

Some things you'll want to consider:

- What kinds of columns are in the dataset
- What's the datatype for each column
- HINT: Be sure to print out the column names and watch out for extra whitespace! Look into the `strip` function to remove whitespace. Look into the `rename` function to rename a column
- How many of each class of bicycle are in our dataset? **HINT: the class is in the `Steer Cat` column**

In [3]:
```
# TODO what columns are in the data

bike_geometries.columns
```

Out[3]:
```
Index(['Brand', 'Year', 'Model', 'Size', 'Steer Cat', 'Head Angle',
       'Fork Offset', 'Seat Angle', 'Chain Stay', 'Wheelbase', 'Top Tube',
       'BB Drop', 'Trail ', 'Flop'],
      dtype='object')
```

In [4]:
```
# TODO perhaps strip extra whitespace in the column names and rename the columns
bike_geometries.columns = bike_geometries.columns.str.strip()

print(bike_geometries.columns)
```

```
Index(['Brand', 'Year', 'Model', 'Size', 'Steer Cat', 'Head Angle',
       'Fork Offset', 'Seat Angle', 'Chain Stay', 'Wheelbase', 'Top Tube',
       'BB Drop', 'Trail', 'Flop'],
      dtype='object')
```

In [5]:
```
# Replace 'old_name' and 'new_name' with actual column names
bike_geometries.rename(columns={
    'Trail ': 'Trail',
}, inplace=True)
```

In [6]:
```
bike_geometries.columns
```

Out[6]:
```
Index(['Brand', 'Year', 'Model', 'Size', 'Steer Cat', 'Head Angle',
       'Fork Offset', 'Seat Angle', 'Chain Stay', 'Wheelbase', 'Top Tube',
       'BB Drop', 'Trail', 'Flop'],
      dtype='object')
```

In [7]:
```
# TODO use value_counts() to explore how many of each class of bike is in the dataset
bike_class_counts = bike_geometries['Steer Cat'].value_counts()

print(bike_class_counts)
```

```
Steer Cat
race     103
sport     49
cross     40
tour      18
crit       4
Name: count, dtype: int64
```

In [8]:
```python
# TODO other EDA, such as .info() or .describe() or any other plot you think is necess

print(bike_geometries.info())

print(bike_geometries.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 356 entries, 0 to 355
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Brand        356 non-null    object 
 1   Year         345 non-null    float64
 2   Model        354 non-null    object 
 3   Size         350 non-null    float64
 4   Steer Cat    214 non-null    object 
 5   Head Angle   356 non-null    float64
 6   Fork Offset  356 non-null    float64
 7   Seat Angle   356 non-null    float64
 8   Chain Stay   354 non-null    float64
 9   Wheelbase    346 non-null    float64
 10  Top Tube     338 non-null    object 
 11  BB Drop      186 non-null    object 
 12  Trail        349 non-null    object 
 13  Flop         349 non-null    object 
dtypes: float64(7), object(7)
memory usage: 39.1+ KB
None
              Year        Size  Head Angle  Fork Offset  Seat Angle  \
count   345.000000  350.000000  356.000000   356.000000  356.000000   
mean   1987.866667   57.040571   72.835000     4.863287   73.021348   
std      12.594511    1.576115    0.692122     0.668766    0.628564   
min    1939.000000   52.000000   70.000000     2.700000   70.000000   
25%    1978.000000   56.000000   72.500000     4.500000   73.000000   
50%    1985.000000   57.000000   73.000000     4.500000   73.000000   
75%    2004.000000   58.000000   73.000000     5.400000   73.300000   
max    2007.000000   64.000000   74.500000     8.000000   75.300000   

       Chain Stay   Wheelbase  
count  354.000000  346.000000  
mean    42.859887  104.309277  
std      1.540584   48.134609  
min     38.800000   96.800000  
25%     41.500000  100.000000  
50%     43.000000  101.300000  
75%     44.000000  103.500000  
max     47.000000  996.000000  
```

4. Set up modeling dataset

Now it's time to get started with the modeling! The **first step is to create a modeling dataset**.

A modeling dataset includes only the columns that we'll want to use for modeling. You will want to **drop** columns that aren't useful. For example, you may want to (**hint**) drop columns that have a lot of **missing values**, such as NaN or nulls. Remember, our machine learning models don't work with null/NaN (Not a Number) inputs. So if a row has a NaN in even just one column, that whole row has to be thrown away by the model. But, instead of throwing out a whole row because of one NaN in a column, we can get rid of columns that have a lot of NaNs, and thereby save more rows for our model.

Your next steps:

- Find out which columns have a lot of missing data (`NaN` in the language of `Pandas`)
- Drop columns that have a lot of `NaN` s (but don't drop ALL columns with `NaN` s yet! Which ones might you want to keep?)
- You might want to keep some columns with `NaN` s, but then fill in the `NaN` with a value, e.g. 0
- Think about which columns you want to drop, and which ones with `NaN` s might benefit from replacing the `NaN` with a 0; you don't have to be a bike expert here, just take a guess and try several things
- Remember, to drop columns you can look into `drop` or you can subset your DataFrame by using a list of column names you want to keep, e.g.:

```
new_df = old_df[ ['column_a', 'column_b', 'column_f'] ]
```

- Drop rows with `NaN` in the `Steer Cat` column; remember, `Steer Cat` is our class, so we need to have a value there! Look into `dropna()`
- You may also want to drop any rows with a class type that's very uncommon in the data

In [9]:

```
# TODO Create a new dataframe that includes only our modeling columns

nan_counts = bike_geometries.isna().sum()

print(nan_counts)

bike_geometries_new = bike_geometries.drop(columns=['BB Drop'])
```

```
Brand            0
Year            11
Model            2
Size             6
Steer Cat      142
Head Angle       0
Fork Offset      0
Seat Angle       0
Chain Stay       2
Wheelbase       10
Top Tube        18
BB Drop        170
Trail            7
Flop             7
dtype: int64
```

In [10]:
```
#TODO what's the shape of your modeling dataset?
bike_geometries_new.shape
```

Out[10]: (356, 13)

In [11]:
```
# TODO drop rows with NaN or NA for our output variable (Steer Cat); use dropna perhap

bike_geometries_cleaned = bike_geometries_new.dropna(subset=['Steer Cat'])

# There are few observations on Year and Model, but I can't use mediam or frequency to
bike_geometries_cleaned = bike_geometries_cleaned.dropna(subset=['Year'])
bike_geometries_cleaned = bike_geometries_cleaned.dropna(subset=['Model'])

nan_counts = bike_geometries_cleaned.isna().sum()

print(nan_counts)
```

```
Brand           0
Year            0
Model           0
Size            2
Steer Cat       0
Head Angle      0
Fork Offset     0
Seat Angle      0
Chain Stay      1
Wheelbase       9
Top Tube       11
Trail           3
Flop            3
dtype: int64
```

In [12]:
```
# TODO what's the shape of our modeling dataset now? Did we lose any observations?
print(bike_geometries.shape)

print(bike_geometries_cleaned.shape)
```

```
(356, 14)
(204, 13)
```

In [13]:
```
356-204
```

Out[13]: 152

- We lost 152 observations by dropping NaNs in Steer Cat, Year and Model. We interestingly didn't lose rows when droping BB Drop which had 170 NaNs. Meaning that the NaNs for Steer Cat were also NaNs for BB Drop.
- I decided to drop Year and Model because there is not a normal distribution for those so I can input median. I can use frequency to input but I am pretty sure that it would be wrong.

In [14]:
```
# TODO look at the head() of your dataframe; what do you notice?
bike_geometries_cleaned.head()
```

Out[14]:

	Brand	Year	Model	Size	Steer Cat	Head Angle	Fork Offset	Seat Angle	Chain Stay	Wheelbase	Top Tube	Tra
12	Bruce Gordon	2002.0	Rock'nRoad	59.0	cross	72.0	5.0	73.0	45.0	107.5	58	57.
13	Specialized	1983.0	Expedition	58.0	cross	72.0	5.1	74.0	45.0	106.1	56.5	56.8
14	Bianchi	2003.0	San Remo	58.0	cross	72.0	5.0	73.0	44.0	104.7	57	Nal
15	Miyata	1991.0	1000 LT	57.0	cross	72.0	5.0	72.0	45.0	104.7	56.5	57.
16	Nishiki	1974.0	International	58.0	cross	72.0	5.1	72.0	44.4	104.2	57.1	56.8

- There are still NAs in the dataset under Tail and Flop. We could replace with 0 but it's better to replace with the mean, median, or a regression that maps the missing value.
- I think median is best, not sensitive to outliers, and not complex like inputting a value via regression.

In [15]:

```
# TODO look into fillna to replace some missing values with, e.g., 0

# Specify columns to fill NaN values
columns_to_fill = ['Year', 'Wheelbase', 'Top Tube', 'Trail', 'Flop', 'Chain Stay', 'Si

# Fill NaN values with median for specified numeric columns
bike_geometries_cleaned[columns_to_fill] = bike_geometries_cleaned[columns_to_fill].fi

# Display cleaned DataFrame
print("\nDataFrame after filling NaN values with median:")
print(bike_geometries_cleaned)

nan_counts = bike_geometries_cleaned.isna().sum()

print(nan_counts)
```

```
DataFrame after filling NaN values with median:
            Brand    Year              Model  Size Steer Cat  Head Angle  \
12   Bruce Gordon  2002.0         Rock'nRoad  59.0     cross        72.0
13    Specialized  1983.0         Expedition  58.0     cross        72.0
14        Bianchi  2003.0           San Remo  58.0     cross        72.0
15         Miyata  1991.0            1000 LT  57.0     cross        72.0
16        Nishiki  1974.0      International  58.0     cross        72.0
..            ...     ...                ...   ...       ...         ...
310      Gaansari  2006.0         Van Cleve  56.0      race        73.5
311         Heron  2005.0        Rally/Road  56.0      race        73.5
323         Dawes  1978.0        Double Blue  58.0      crit        74.0
325    Holdsworth  1973.0      Racing Custom  61.0      crit        74.0
354     Waterford  2005.0  Track/Fixed-gear  56.0      crit        74.5

     Fork Offset  Seat Angle  Chain Stay  Wheelbase Top Tube  Trail   Flop
12          5.00        73.0        45.0      107.5       58   57.9  17.02
13          5.10        74.0        45.0      106.1     56.5  56.85  16.71
14          5.00        73.0        44.0      104.7       57  55.87  15.71
15          5.00        72.0        45.0      104.7     56.5   57.9  17.02
16          5.10        72.0        44.4      104.2     57.1  56.85  16.71
..           ...         ...         ...        ...      ...    ...    ...
310         4.25        72.5        42.0      100.3       57  56.39  15.36
311         4.25        72.5        42.5      100.3       57  56.39  15.36
323         3.80        73.0        43.2      101.3     59.7  57.96  15.36
325         3.18        75.0        41.3       99.1       56  64.41  17.07
354         3.00        75.0        39.5       97.0       56  63.16  16.26

[204 rows x 13 columns]
Brand          0
Year           0
Model          0
Size           0
Steer Cat      0
Head Angle     0
Fork Offset    0
Seat Angle     0
Chain Stay     0
Wheelbase      0
Top Tube       0
Trail          0
Flop           0
dtype: int64
```

In [16]:
```
# TODO look at your dataset again with head() to see if the missing values were fillec
bike_geometries_cleaned.head()
```

Out[16]:

	Brand	Year	Model	Size	Steer Cat	Head Angle	Fork Offset	Seat Angle	Chain Stay	Wheelbase	Top Tube	Tra
12	Bruce Gordon	2002.0	Rock'nRoad	59.0	cross	72.0	5.0	73.0	45.0	107.5	58	57.
13	Specialized	1983.0	Expedition	58.0	cross	72.0	5.1	74.0	45.0	106.1	56.5	56.8
14	Bianchi	2003.0	San Remo	58.0	cross	72.0	5.0	73.0	44.0	104.7	57	55.8
15	Miyata	1991.0	1000 LT	57.0	cross	72.0	5.0	72.0	45.0	104.7	56.5	57.
16	Nishiki	1974.0	International	58.0	cross	72.0	5.1	72.0	44.4	104.2	57.1	56.8

In [17]:
```
# TODO are there any classes (Steer Cat) that you want to drop from
# your modeling dataset because there are very few of them? If so, do so here!
bike_geometries_cleaned['Steer Cat'].value_counts().sum
```

Out[17]:
```
<bound method NDFrame._add_numeric_operations.<locals>.sum of Steer Cat
race     98
sport    48
cross    38
tour     17
crit      3
Name: count, dtype: int64>
```

In [18]:
```
# Drop rows where 'Steer Cat' is "crit"
bike_geometries_cleaned = bike_geometries_cleaned[bike_geometries_cleaned['Steer Cat']
```

In [19]:
```
# TODO what's the shape of your dataset now?
bike_geometries_cleaned.shape
```

Out[19]:
```
(201, 13)
```

In [20]:
```
# TODO what's the value_counts() of your classes now?

# Print Value counts after removing crit
bike_geometries_cleaned['Steer Cat'].value_counts().sum
```

Out[20]:
```
<bound method NDFrame._add_numeric_operations.<locals>.sum of Steer Cat
race     98
sport    48
cross    38
tour     17
Name: count, dtype: int64>
```

In [21]:
```
# Print Model
print(bike_geometries_cleaned['Model'].value_counts().sum)
# Print Brand
print(bike_geometries_cleaned['Brand'].value_counts().sum)
```

```
<bound method NDFrame._add_numeric_operations.<locals>.sum of Model
930               9
510               7
730               6
Tour de France    6
Super Corsa       6
                 ..
Accordo           1
Le Mans 12        1
Brava             1
Roubaix Pro       1
Rally/Road        1
Name: count, Length: 127, dtype: int64>
<bound method NDFrame._add_numeric_operations.<locals>.sum of Brand
Trek            70
Miyata          19
Gitane          13
Gaansari        10
Bianchi          8
Specialized      7
Bridgestone      7
Mercian          6
Ebisu            6
Jamis            5
Litespeed        4
Rivendell        4
Centurian        4
Fuji             4
LeMond           4
Schwinn          3
Nishiki          3
Lemond           2
Kogswell         2
Habanero         2
Velo Orange      1
Cinelli          1
Waterford        1
Holdsworth       1
Windsor          1
Merckx           1
Bruce Gordon     1
IF               1
Salsa            1
Soma             1
Terry            1
Univega          1
Victoria         1
Zabrakenko       1
Hetchins         1
Dawes            1
Ferrare          1
Heron            1
Name: count, dtype: int64>
```

In [22]:

```
bike_geometries_cleaned = bike_geometries_cleaned.drop(columns=['Model'])
```

In [23]:

```
# Count occurrences of each brand
brand_counts = bike_geometries_cleaned['Brand'].value_counts()
```

```python
# Identify brands with fewer than 7 occurrences
brands_to_replace = brand_counts[brand_counts < 4].index

# Replace those brands with "Other"
bike_geometries_cleaned['Brand'] = bike_geometries_cleaned['Brand'].replace(brands_to_

# Display the modified DataFrame
print(bike_geometries_cleaned)
```

```
           Brand    Year  Size Steer Cat  Head Angle  Fork Offset  Seat Angle  \
12         Other  2002.0  59.0     cross        72.0         5.00        73.0
13   Specialized  1983.0  58.0     cross        72.0         5.10        74.0
14       Bianchi  2003.0  58.0     cross        72.0         5.00        73.0
15        Miyata  1991.0  57.0     cross        72.0         5.00        72.0
16         Other  1974.0  58.0     cross        72.0         5.10        72.0
..           ...     ...   ...       ...         ...          ...         ...
307        Other  2006.0  56.0      race        73.5         4.30        73.5
308      Bianchi  2006.0  57.0      race        73.5         4.30        73.5
309     Gaansari  2006.0  58.0      race        73.5         4.25        72.5
310     Gaansari  2006.0  56.0      race        73.5         4.25        72.5
311        Other  2005.0  56.0      race        73.5         4.25        72.5

     Chain Stay  Wheelbase Top Tube  Trail   Flop
12         45.0      107.5       58   57.9  17.02
13         45.0      106.1     56.5  56.85  16.71
14         44.0      104.7       57  55.87  15.71
15         45.0      104.7     56.5   57.9  17.02
16         44.4      104.2     57.1  56.85  16.71
..          ...        ...      ...    ...    ...
307        40.4       98.1       56  55.87  15.21
308        40.6      100.3       56  55.87  15.21
309        42.0      100.3     58.5  56.39  15.36
310        42.0      100.3       57  56.39  15.36
311        42.5      100.3       57  56.39  15.36

[201 rows x 12 columns]
```

In [24]:
```python
bike_geometries_cleaned['Brand'].value_counts().sum
```

Out[24]:
```
<bound method NDFrame._add_numeric_operations.<locals>.sum of Brand
Trek           70
Other          30
Miyata         19
Gitane         13
Gaansari       10
Bianchi         8
Specialized     7
Bridgestone     7
Ebisu           6
Mercian         6
Jamis           5
Rivendell       4
Litespeed       4
Centurian       4
Fuji            4
LeMond          4
Name: count, dtype: int64>
```

In [25]:
```python
# Cast specified columns to float
bike_geometries_cleaned['Top Tube'] = bike_geometries_cleaned['Top Tube'].astype(float
```

```
bike_geometries_cleaned['Trail'] = bike_geometries_cleaned['Trail'].astype(float)
bike_geometries_cleaned['Flop'] = bike_geometries_cleaned['Flop'].astype(float)
```

- Looks like Model has too much cardinality, Brand is the only one we can rescue if we transform to other.

In [26]:
```
# Perform one-hot encoding on the 'Model' and 'Brand' columns
one_hot_encoded = pd.get_dummies(bike_geometries_cleaned[['Brand']], prefix=['Brand'])

# Convert True/False to 1/0
one_hot_encoded = one_hot_encoded.astype(int)

# Concatenate the one-hot encoded columns with the original DataFrame
bike_geometries_encoded = pd.concat([bike_geometries_cleaned, one_hot_encoded], axis=1

# Optionally drop the original 'Model' and 'Brand' columns if no longer needed
bike_geometries_encoded.drop('Brand', axis=1, inplace=True)

# Display the DataFrame after one-hot encoding
print("\nDataFrame after one-hot encoding:")
print(bike_geometries_encoded)
```

```
DataFrame after one-hot encoding:
       Year  Size Steer Cat  Head Angle  Fork Offset  Seat Angle  Chain Stay  \
12   2002.0  59.0     cross        72.0         5.00        73.0        45.0
13   1983.0  58.0     cross        72.0         5.10        74.0        45.0
14   2003.0  58.0     cross        72.0         5.00        73.0        44.0
15   1991.0  57.0     cross        72.0         5.00        72.0        45.0
16   1974.0  58.0     cross        72.0         5.10        72.0        44.4
..      ...   ...       ...         ...          ...         ...         ...
307  2006.0  56.0      race        73.5         4.30        73.5        40.4
308  2006.0  57.0      race        73.5         4.30        73.5        40.6
309  2006.0  58.0      race        73.5         4.25        72.5        42.0
310  2006.0  56.0      race        73.5         4.25        72.5        42.0
311  2005.0  56.0      race        73.5         4.25        72.5        42.5

     Wheelbase  Top Tube  Trail  ...  Brand_Gitane  Brand_Jamis  Brand_LeMond  \
12       107.5      58.0  57.90  ...             0            0             0
13       106.1      56.5  56.85  ...             0            0             0
14       104.7      57.0  55.87  ...             0            0             0
15       104.7      56.5  57.90  ...             0            0             0
16       104.2      57.1  56.85  ...             0            0             0
..         ...       ...    ...  ...           ...          ...           ...
307       98.1      56.0  55.87  ...             0            0             0
308      100.3      56.0  55.87  ...             0            0             0
309      100.3      58.5  56.39  ...             0            0             0
310      100.3      57.0  56.39  ...             0            0             0
311      100.3      57.0  56.39  ...             0            0             0

     Brand_Litespeed  Brand_Mercian  Brand_Miyata  Brand_Other  \
12                 0              0             0            1
13                 0              0             0            0
14                 0              0             0            0
15                 0              0             1            0
16                 0              0             0            1
..               ...            ...           ...          ...
307                0              0             0            1
308                0              0             0            0
309                0              0             0            0
310                0              0             0            0
311                0              0             0            1

     Brand_Rivendell  Brand_Specialized  Brand_Trek
12                 0                  0           0
13                 0                  1           0
14                 0                  0           0
15                 0                  0           0
16                 0                  0           0
..               ...                ...         ...
307                0                  0           0
308                0                  0           0
309                0                  0           0
310                0                  0           0
311                0                  0           0

[201 rows x 27 columns]
```

5. kNN classification

Now that our modeling dataset is ready, we can start building our kNN model! Run the import cell below to get started. Notice we'll also be using a train/test split!

In [27]:
```
# TODO run this cell
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

Before we can run our model, we have to create our X matrix and y vector. To do this, you can take your modeling dataset from above and drop the dependent variable (`Steer Cat`) to create the X input matrix. Then take only `Steer Cat` to create your y vector.

In [28]:
```
# TODO create X matrix and y vector from columns

X = bike_geometries_encoded.drop('Steer Cat', axis=1)
y = bike_geometries_encoded['Steer Cat']

print(X.shape , y.shape)
#''''''''
print(X.sample(5))
#''''''''''
print(y.sample(5))
```

```
(201, 26) (201,)
       Year  Size  Head Angle  Fork Offset  Seat Angle  Chain Stay  Wheelbase  \
221  1980.0  56.0        73.0          5.5        73.0        44.5      103.5   
190  2005.0  58.0        73.0          4.5        72.0        43.5      100.3   
224  1980.0  56.0        73.0          5.5        73.0        44.5      103.5   
217  1979.0  56.0        73.0          5.5        73.0        44.5      103.5   
176  1988.0  56.0        73.0          4.5        74.0        40.8       99.2   

     Top Tube  Trail   Flop  ...  Brand_Gitane  Brand_Jamis  Brand_LeMond  \
221      56.0  46.44  12.98  ...             0            0             0   
190      57.0  56.89  15.91  ...             0            0             0   
224      56.0  46.44  12.98  ...             0            0             0   
217      56.0  46.44  12.98  ...             0            0             0   
176      55.0  56.89  15.91  ...             0            0             0   

     Brand_Litespeed  Brand_Mercian  Brand_Miyata  Brand_Other  \
221                0              0             0            0   
190                0              0             0            1   
224                0              0             0            0   
217                0              0             0            0   
176                0              0             1            0   

     Brand_Rivendell  Brand_Specialized  Brand_Trek  
221                0                  0           1  
190                0                  0           0  
224                0                  0           1  
217                0                  0           1  
176                0                  0           0  

[5 rows x 26 columns]
173     race
138     race
311     race
42      tour
106    sport
Name: Steer Cat, dtype: object
```

In [29]:

```python
# TODO run info on X
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 201 entries, 12 to 311
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Year               201 non-null    float64
 1   Size               201 non-null    float64
 2   Head Angle         201 non-null    float64
 3   Fork Offset        201 non-null    float64
 4   Seat Angle         201 non-null    float64
 5   Chain Stay         201 non-null    float64
 6   Wheelbase          201 non-null    float64
 7   Top Tube           201 non-null    float64
 8   Trail              201 non-null    float64
 9   Flop               201 non-null    float64
 10  Brand_Bianchi      201 non-null    int32  
 11  Brand_Bridgestone  201 non-null    int32  
 12  Brand_Centurian    201 non-null    int32  
 13  Brand_Ebisu        201 non-null    int32  
 14  Brand_Fuji         201 non-null    int32  
 15  Brand_Gaansari     201 non-null    int32  
 16  Brand_Gitane       201 non-null    int32  
 17  Brand_Jamis        201 non-null    int32  
 18  Brand_LeMond       201 non-null    int32  
 19  Brand_Litespeed    201 non-null    int32  
 20  Brand_Mercian      201 non-null    int32  
 21  Brand_Miyata       201 non-null    int32  
 22  Brand_Other        201 non-null    int32  
 23  Brand_Rivendell    201 non-null    int32  
 24  Brand_Specialized  201 non-null    int32  
 25  Brand_Trek         201 non-null    int32  
dtypes: float64(10), int32(16)
memory usage: 29.8 KB
```

In [30]:

```
# TODO in info above, did you see any columns that should be floats but aren't?
# If so, cast them as floats now! e.g. X = X.astype(float)
# TODO run info on X
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 201 entries, 12 to 311
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Year               201 non-null    float64
 1   Size               201 non-null    float64
 2   Head Angle         201 non-null    float64
 3   Fork Offset        201 non-null    float64
 4   Seat Angle         201 non-null    float64
 5   Chain Stay         201 non-null    float64
 6   Wheelbase          201 non-null    float64
 7   Top Tube           201 non-null    float64
 8   Trail              201 non-null    float64
 9   Flop               201 non-null    float64
 10  Brand_Bianchi      201 non-null    int32  
 11  Brand_Bridgestone  201 non-null    int32  
 12  Brand_Centurian    201 non-null    int32  
 13  Brand_Ebisu        201 non-null    int32  
 14  Brand_Fuji         201 non-null    int32  
 15  Brand_Gaansari     201 non-null    int32  
 16  Brand_Gitane       201 non-null    int32  
 17  Brand_Jamis        201 non-null    int32  
 18  Brand_LeMond       201 non-null    int32  
 19  Brand_Litespeed    201 non-null    int32  
 20  Brand_Mercian      201 non-null    int32  
 21  Brand_Miyata       201 non-null    int32  
 22  Brand_Other        201 non-null    int32  
 23  Brand_Rivendell    201 non-null    int32  
 24  Brand_Specialized  201 non-null    int32  
 25  Brand_Trek         201 non-null    int32  
dtypes: float64(10), int32(16)
memory usage: 29.8 KB
```

In [31]:

```
# TODO check info again to make sure any casts worked
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 201 entries, 12 to 311
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Year               201 non-null    float64
 1   Size               201 non-null    float64
 2   Head Angle         201 non-null    float64
 3   Fork Offset        201 non-null    float64
 4   Seat Angle         201 non-null    float64
 5   Chain Stay         201 non-null    float64
 6   Wheelbase          201 non-null    float64
 7   Top Tube           201 non-null    float64
 8   Trail              201 non-null    float64
 9   Flop               201 non-null    float64
 10  Brand_Bianchi      201 non-null    int32  
 11  Brand_Bridgestone  201 non-null    int32  
 12  Brand_Centurian    201 non-null    int32  
 13  Brand_Ebisu        201 non-null    int32  
 14  Brand_Fuji         201 non-null    int32  
 15  Brand_Gaansari     201 non-null    int32  
 16  Brand_Gitane       201 non-null    int32  
 17  Brand_Jamis        201 non-null    int32  
 18  Brand_LeMond       201 non-null    int32  
 19  Brand_Litespeed    201 non-null    int32  
 20  Brand_Mercian      201 non-null    int32  
 21  Brand_Miyata       201 non-null    int32  
 22  Brand_Other        201 non-null    int32  
 23  Brand_Rivendell    201 non-null    int32  
 24  Brand_Specialized  201 non-null    int32  
 25  Brand_Trek         201 non-null    int32  
dtypes: float64(10), int32(16)
memory usage: 29.8 KB
```

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

Out[32]:

	Year	Size	Head Angle	Fork Offset	Seat Angle	Chain Stay	Wheelbase	Top Tube	Trail	Flop	...	Brand_Gitane	Brar
12	2002.0	59.0	72.0	5.0	73.0	45.0	107.5	58.0	57.90	17.02	...	0	
13	1983.0	58.0	72.0	5.1	74.0	45.0	106.1	56.5	56.85	16.71	...	0	
14	2003.0	58.0	72.0	5.0	73.0	44.0	104.7	57.0	55.87	15.71	...	0	
15	1991.0	57.0	72.0	5.0	72.0	45.0	104.7	56.5	57.90	17.02	...	0	
16	1974.0	58.0	72.0	5.1	72.0	44.4	104.2	57.1	56.85	16.71	...	0	

5 rows × 26 columns

5a) kNN with train/test split

Before we build any model, ever, we have to do a train/test split. Go ahead and do a 70/30 train/test split now. Use the built-in sklearn method `train_test_split`.

Then fill in the blank (...) in the code below to get your first kNN model working!

In [33]:
```
import math
print(X.shape)
result = math.sqrt(201)

print(result)
```

```
(201, 26)
14.177446878757825
```

In [34]:
```
X
```

Out[34]:

	Year	Size	Head Angle	Fork Offset	Seat Angle	Chain Stay	Wheelbase	Top Tube	Trail	Flop	...	Brand_Gitane	Bra
12	2002.0	59.0	72.0	5.00	73.0	45.0	107.5	58.0	57.90	17.02	...	0	
13	1983.0	58.0	72.0	5.10	74.0	45.0	106.1	56.5	56.85	16.71	...	0	
14	2003.0	58.0	72.0	5.00	73.0	44.0	104.7	57.0	55.87	15.71	...	0	
15	1991.0	57.0	72.0	5.00	72.0	45.0	104.7	56.5	57.90	17.02	...	0	
16	1974.0	58.0	72.0	5.10	72.0	44.4	104.2	57.1	56.85	16.71	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
307	2006.0	56.0	73.5	4.30	73.5	40.4	98.1	56.0	55.87	15.21	...	0	
308	2006.0	57.0	73.5	4.30	73.5	40.6	100.3	56.0	55.87	15.21	...	0	
309	2006.0	58.0	73.5	4.25	72.5	42.0	100.3	58.5	56.39	15.36	...	0	
310	2006.0	56.0	73.5	4.25	72.5	42.0	100.3	57.0	56.39	15.36	...	0	
311	2005.0	56.0	73.5	4.25	72.5	42.5	100.3	57.0	56.39	15.36	...	0	

201 rows × 26 columns

In [35]:
```
y
```

Out[35]:
```
12      cross
13      cross
14      cross
15      cross
16      cross
        ...
307      race
308      race
309      race
310      race
311      race
Name: Steer Cat, Length: 201, dtype: object
```

In [36]:
```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()
```

```
# Fit and transform the labels
y_encoded = label_encoder.fit_transform(y)
```

In [37]:
```
y
```

Out[37]:
```
12     cross
13     cross
14     cross
15     cross
16     cross
       ...
307     race
308     race
309     race
310     race
311     race
Name: Steer Cat, Length: 201, dtype: object
```

In [38]:
```
y_encoded
```

Out[38]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1])
```

- 14 neighbors, seems like a lot, but lets test different amounts

In [39]:
```
# TODO A 70/30 train/test split using sklearn's train_test_split

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

In [40]:
```
# fit on train
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
```

Out[40]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [41]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("\nData types:")
print("X_train types:\n", X_train.dtypes)
print("y_train type:", y_train.dtype)
```

```
X_train shape: (140, 26)
y_train shape: (140,)

Data types:
X_train types:
 Year                  float64
Size                  float64
Head Angle            float64
Fork Offset           float64
Seat Angle            float64
Chain Stay            float64
Wheelbase             float64
Top Tube              float64
Trail                 float64
Flop                  float64
Brand_Bianchi           int32
Brand_Bridgestone       int32
Brand_Centurian         int32
Brand_Ebisu             int32
Brand_Fuji              int32
Brand_Gaansari          int32
Brand_Gitane            int32
Brand_Jamis             int32
Brand_LeMond            int32
Brand_Litespeed         int32
Brand_Mercian           int32
Brand_Miyata            int32
Brand_Other             int32
Brand_Rivendell         int32
Brand_Specialized       int32
Brand_Trek              int32
dtype: object
y_train type: object
```

In [42]:
```
print("\nMissing values in X_train:\n", X_train.isnull().sum())
print("Missing values in y_train:\n", y_train.isnull().sum())
```

```
Missing values in X_train:
 Year                  0
Size                  0
Head Angle            0
Fork Offset           0
Seat Angle            0
Chain Stay            0
Wheelbase             0
Top Tube              0
Trail                 0
Flop                  0
Brand_Bianchi         0
Brand_Bridgestone     0
Brand_Centurian       0
Brand_Ebisu           0
Brand_Fuji            0
Brand_Gaansari        0
Brand_Gitane          0
Brand_Jamis           0
Brand_LeMond          0
Brand_Litespeed       0
Brand_Mercian         0
Brand_Miyata          0
Brand_Other           0
Brand_Rivendell       0
Brand_Specialized     0
Brand_Trek            0
dtype: int64
Missing values in y_train:
 0
```

In [43]:

```
print("\nMissing values in X_train:\n", X_test.isnull().sum())
print("Missing values in y_train:\n", y_test.isnull().sum())
```

```
Missing values in X_train:
 Year                 0
Size                 0
Head Angle           0
Fork Offset          0
Seat Angle           0
Chain Stay           0
Wheelbase            0
Top Tube             0
Trail                0
Flop                 0
Brand_Bianchi        0
Brand_Bridgestone    0
Brand_Centurian      0
Brand_Ebisu          0
Brand_Fuji           0
Brand_Gaansari       0
Brand_Gitane         0
Brand_Jamis          0
Brand_LeMond         0
Brand_Litespeed      0
Brand_Mercian        0
Brand_Miyata         0
Brand_Other          0
Brand_Rivendell      0
Brand_Specialized    0
Brand_Trek           0
dtype: int64
Missing values in y_train:
 0
```

In [44]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
import numpy as np
```

In [45]:

```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("\nX_train data types:\n", X_train.dtypes)
print("\nX_test data types:\n", X_test.dtypes)
```

```
X_train shape: (140, 26)
X_test shape: (61, 26)

X_train data types:
 Year                 float64
Size                float64
Head Angle          float64
Fork Offset         float64
Seat Angle          float64
Chain Stay          float64
Wheelbase           float64
Top Tube            float64
Trail               float64
Flop                float64
Brand_Bianchi         int32
Brand_Bridgestone     int32
Brand_Centurian       int32
Brand_Ebisu           int32
Brand_Fuji            int32
Brand_Gaansari        int32
Brand_Gitane          int32
Brand_Jamis           int32
Brand_LeMond          int32
Brand_Litespeed       int32
Brand_Mercian         int32
Brand_Miyata          int32
Brand_Other           int32
Brand_Rivendell       int32
Brand_Specialized     int32
Brand_Trek            int32
dtype: object

X_test data types:
 Year                 float64
Size                float64
Head Angle          float64
Fork Offset         float64
Seat Angle          float64
Chain Stay          float64
Wheelbase           float64
Top Tube            float64
Trail               float64
Flop                float64
Brand_Bianchi         int32
Brand_Bridgestone     int32
Brand_Centurian       int32
Brand_Ebisu           int32
Brand_Fuji            int32
Brand_Gaansari        int32
Brand_Gitane          int32
Brand_Jamis           int32
Brand_LeMond          int32
Brand_Litespeed       int32
Brand_Mercian         int32
Brand_Miyata          int32
Brand_Other           int32
Brand_Rivendell       int32
Brand_Specialized     int32
Brand_Trek            int32
dtype: object
```

In [46]:
```
# Convert your training and test sets to C-contiguous arrays
X_train = np.ascontiguousarray(X_train)
X_test = np.ascontiguousarray(X_test)

# Convert your training and test sets to C-contiguous arrays
y_train = np.ascontiguousarray(y_train)
y_test = np.ascontiguousarray(y_test)
```

In [47]:
```
X_train
```

Out[47]:
```
array([[1.978e+03, 5.600e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.984e+03, 5.800e+01, 7.200e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.984e+03, 5.700e+01, 7.200e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       ...,
       [1.986e+03, 5.700e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [2.004e+03, 6.300e+01, 7.350e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.979e+03, 5.600e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00]])
```

In [48]:
```
X_test
```

Out[48]:
```
array([[1.980e+03, 5.600e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.988e+03, 5.600e+01, 7.200e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.984e+03, 5.800e+01, 7.200e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       ...,
       [2.006e+03, 5.800e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.978e+03, 5.600e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.986e+03, 5.900e+01, 7.300e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00]])
```

In [49]:
```
# Convert all columns to float if they are not already
#X_train = X_train.astype(float)
#X_test = X_test.astype(float)

# Convert all columns to float if they are not already
#y_train = y_train.astype(float)
#y_test = y_test.astype(float)
```

In [50]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

# Fit the KNN classifier on the training data
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)

# Predict on test set
y_pred = knn.predict(X_test.values)

# Print accuracy metric
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy of the KNN classifier:", accuracy)
```

```
C:\Users\iamontes\AppData\Local\anaconda3\Lib\site-packages\sklearn\base.py:464: User
Warning: X does not have valid feature names, but KNeighborsClassifier was fitted wit
h feature names
  warnings.warn(
Accuracy of the KNN classifier: 0.8524590163934426
```

What's another metric we can use on a classification model, other than accuracy? A **confusiong matrix**, perhaps? Use the built-in `sklearn` `confusion_matrix` from `sklearn.metrics`. Use the import below, then fill in the (...) to create a confusion matrix that compares the actual `y` against the predicted `y`.

In [51]:
```
# TODO run the import
from sklearn.metrics import confusion_matrix
```

In [52]:
```
# TODO fill in the blanks (...) and calculate the confusion matrix between the actual

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print("Confusion Matrix:\n", cm)

# Normalize the confusion matrix to get percentages
cm_percentage = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100

# Print the confusion matrix as percentages with two decimal places
print("Confusion Matrix (Percentages):")
print(np.round(cm_percentage, 2))  # Round to two decimal places
```

```
Confusion Matrix:
 [[ 4  4  0  0]
 [ 2 31  0  0]
 [ 0  1 15  1]
 [ 1  0  0  2]]
Confusion Matrix (Percentages):
[[50.   50.    0.    0.  ]
 [ 6.06 93.94  0.    0.  ]
 [ 0.    5.88 88.24  5.88]
 [33.33  0.    0.   66.67]]
```

5b) kNN with cross-validation: tuning for value of k

What value of k gives us the best kNN model? Let's use cross-validation to **tune the hyperparameter, k**. If you need a refresher on hypermarameter tuning with cross-validation, revisit the second instructor webinar. We'll walk you through it using kNN here.

In the webinar, we talked about how there are two uses for tuning a hyperparameter with cross-validation:

1. To choose the best value for a parameter (here, the k in kNN)

2. To choose the best model type between several alogirthms (the next step below, choosing between kNN and logisitic regression)

What are the inputs to `cross_val_score`? Use the documentation to fill in the (...) in the code skeleton below!

Remember, as we saw above, kNN is called `KNeighborsClassifier` in `sklearn`. If you need to, be sure to look at the documentation.

In [53]:
```
# TODO run the import for cross-val-score

from sklearn.model_selection import cross_val_score
```

In [54]:
```
print("NaN values in X:", np.isnan(X).sum())
print("Infinite values in X:", np.isinf(X).sum())
```

```
NaN values in X: Year                0
Size                 0
Head Angle           0
Fork Offset          0
Seat Angle           0
Chain Stay           0
Wheelbase            0
Top Tube             0
Trail                0
Flop                 0
Brand_Bianchi        0
Brand_Bridgestone    0
Brand_Centurian      0
Brand_Ebisu          0
Brand_Fuji           0
Brand_Gaansari       0
Brand_Gitane         0
Brand_Jamis          0
Brand_LeMond         0
Brand_Litespeed      0
Brand_Mercian        0
Brand_Miyata         0
Brand_Other          0
Brand_Rivendell      0
Brand_Specialized    0
Brand_Trek           0
dtype: int64
Infinite values in X: Year                0
Size                 0
Head Angle           0
Fork Offset          0
Seat Angle           0
Chain Stay           0
Wheelbase            0
Top Tube             0
Trail                0
Flop                 0
Brand_Bianchi        0
Brand_Bridgestone    0
Brand_Centurian      0
Brand_Ebisu          0
Brand_Fuji           0
Brand_Gaansari       0
Brand_Gitane         0
Brand_Jamis          0
Brand_LeMond         0
Brand_Litespeed      0
Brand_Mercian        0
Brand_Miyata         0
Brand_Other          0
Brand_Rivendell      0
Brand_Specialized    0
Brand_Trek           0
dtype: int64
```

In [55]:
```
X = np.ascontiguousarray(X.values.astype(float))
```

In [56]:
```
# TODO (1) Run a 10-fold cross-validation with K=7 for kNN (the n_neighbors parameter)
#
```

```python
# Create the KNN classifier with n_neighbors=7
knn = KNeighborsClassifier(n_neighbors=7)

# Run a 10-fold cross-validation
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')

# Print the accuracy scores from cross-validation
print("Cross-validation accuracy scores:", scores)
```

```
Cross-validation accuracy scores: [0.70731707 0.775      0.9        0.775      0.8
]
```

In [57]:

```python
# TODO (2) Get average accuracy as an estimate of out-of-sample accuracy

# Calculate average accuracy as an estimate of out-of-sample accuracy
average_accuracy = scores.mean()
print("Average accuracy (out-of-sample estimate):", average_accuracy)
```

```
Average accuracy (out-of-sample estimate): 0.7914634146341463
```

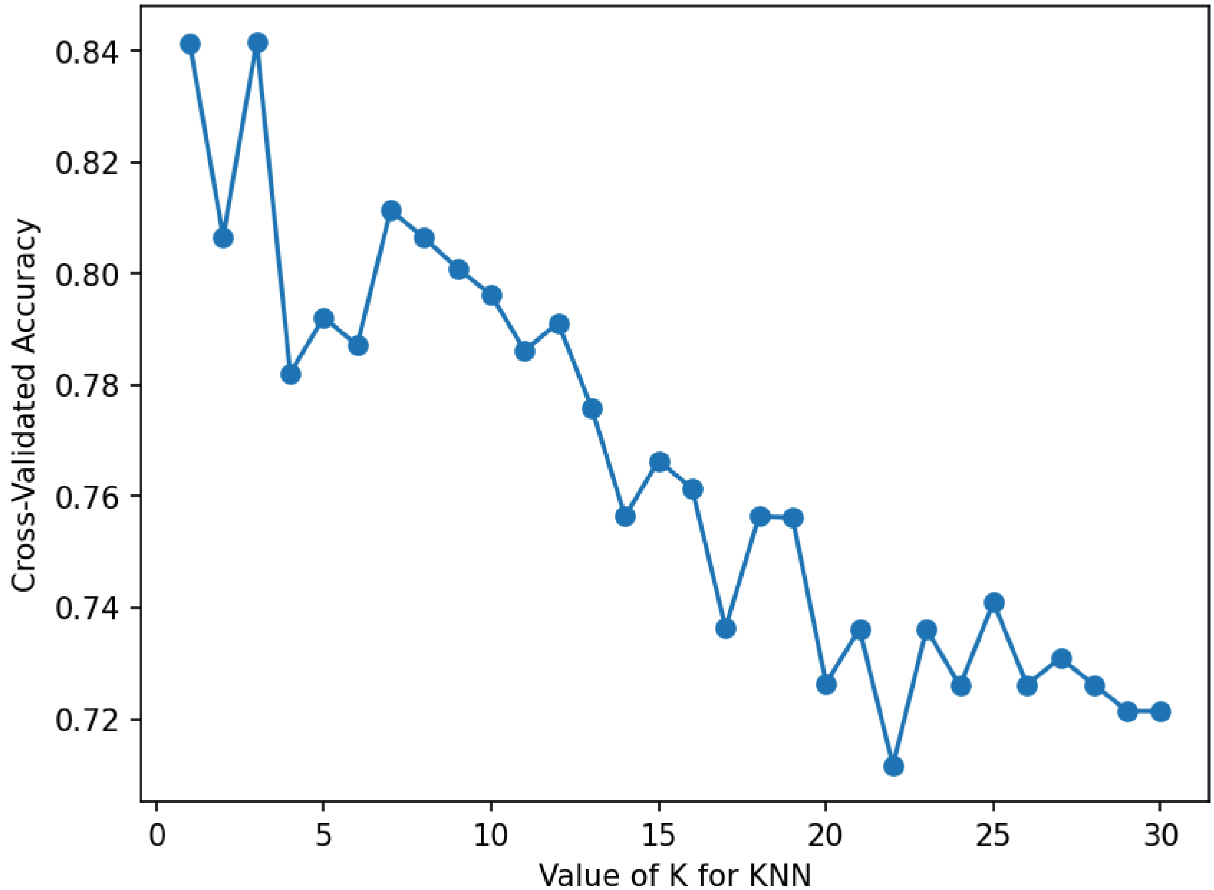
In [58]:

```python
# TODO (3) Search for an optimal value of k from 1-30 (write a loop)

# Initialize variables for k range and scores
k_range = list(range(1, 31))
k_scores = []

# Loop through k values from 1 to 30
for k in k_range:
    # Create a new KNeighborsClassifier with the current value of k
    knn = KNeighborsClassifier(n_neighbors=k)

    # Perform cross-validation and get accuracy scores
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')

    # Append the average score to k_scores
    k_scores.append(scores.mean())

# Print the average accuracy scores for each value of k with two decimal places
print("Average accuracy scores for each value of k:")
for k, score in zip(k_range, k_scores):
    print(f"k={k}: {score:.2f}")
```

```
Average accuracy scores for each value of k:
k=1: 0.84
k=2: 0.81
k=3: 0.84
k=4: 0.78
k=5: 0.79
k=6: 0.79
k=7: 0.81
k=8: 0.81
k=9: 0.80
k=10: 0.80
k=11: 0.79
k=12: 0.79
k=13: 0.78
k=14: 0.76
k=15: 0.77
k=16: 0.76
k=17: 0.74
k=18: 0.76
k=19: 0.76
k=20: 0.73
k=21: 0.74
k=22: 0.71
k=23: 0.74
k=24: 0.73
k=25: 0.74
k=26: 0.73
k=27: 0.73
k=28: 0.73
k=29: 0.72
k=30: 0.72
```

In [59]:
```
# TODO plot the value of K for kNN (x-axis) versus the cross-validated accuracy (y-axi
plt.figure(dpi=150)
plt.plot(k_range, k_scores, marker='o') # where are the scores for each value of k in
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

Out[59]:
```
Text(0, 0.5, 'Cross-Validated Accuracy')
```

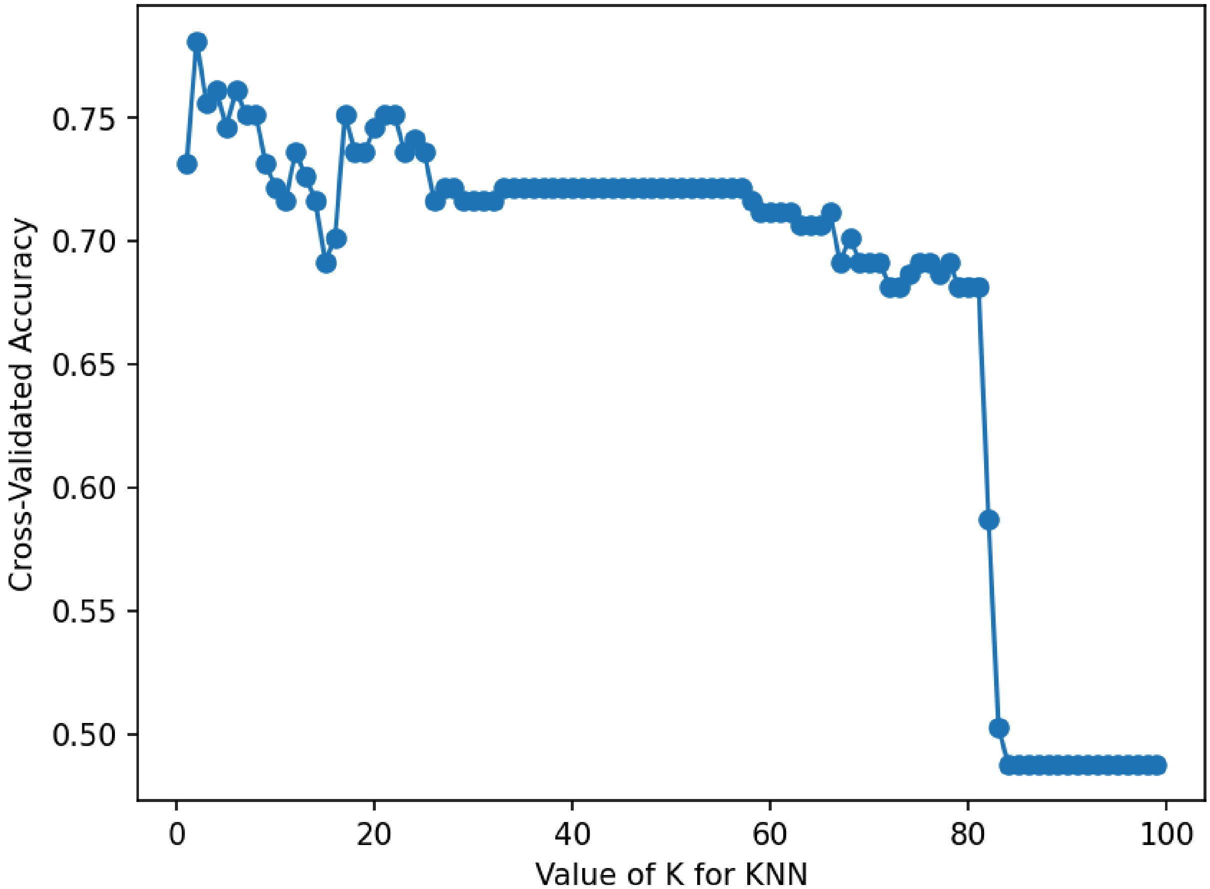


In [60]:

```
# (4) TODO now try with k-fold = 3 (cv parameter)
# fill in the loop and replace the ...'s!

k_range = list(range(1, 100))
k_scores = []

for k in k_range:
    ## same code as above, just change cv=3
    # Create a new KNeighborsClassifier with the current value of k
    knn = KNeighborsClassifier(n_neighbors=k)

    # Perform cross-validation and get accuracy scores
    scores = cross_val_score(knn, X, y, cv=3, scoring='accuracy')
    k_scores.append(scores.mean())

plt.figure(dpi=150)
plt.plot(k_range, k_scores, marker='o')
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

Out[60]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



5c) kNN with cross-validation: model selection of kNN vs. logistic

We know our optimal value of k, but let's use cross-validation to see how kNN compares against a logistic regression. This is called using cross-validation for **model selection**.

In [61]:
```
# TODO Use 10-fold cross-validation with the best KNN model
# that is, set k equal to the k that gave you the best model above!

knn = KNeighborsClassifier(n_neighbors=3)
print(cross_val_score(knn, X, y, cv=10, scoring='accuracy').mean())
```

```
0.8416666666666666
```

We know the best value of `k` for a kNN, that's what we found in step 5b. We repeated that in the cell above so we can now compare it to a logistic regression. Use 10-fold cross-validation to see how well a logistic regression performs.

In [62]:
```
# TODO run the import statement for a logistic regression
from sklearn.linear_model import LogisticRegression
```

In [63]:
```
# TODO run 10-fold cross-validation with logistic regression

logreg = LogisticRegression(solver='liblinear',multi_class='auto') # use these values
print(cross_val_score(logreg, X, y, cv=10, scoring='accuracy').mean())
```

```
0.9452380952380952
```

6) Automating parameter tuning using `GridSearchCV`

We already used grid search above to find the best value of `k`. But there we used a manual loop that checked the values k=1 to k=30. Is there an easier way? Of course there is! `sklearn` has a built-in `GridSearchCV` that combines a grid search and cross-validation. Import it and let's get to using it.

You may need to refer to the documentation for GridSearchCV.

In [64]:
```python
# TODO run the import
from sklearn.model_selection import GridSearchCV
```

In [65]:
```python
# TODO define the parameter values that should be searched
# That is, what's the range of k's you want to try?

k_range = list(range(1, 31))
```

Now we just need to create the hyperparameter grid search input for `sklearn`. We need to pass it in as a dictionary, so let's create a dictionary now. We'll call our variable `param_grid` and it will hold the dictionary in this form:

```
{ name_of_hyperparameter : [ values, of, hyperparameter ] }
```

In [66]:
```python
# TODO create a parameter grid: map the parameter names to the values that should be s
# (just run this cell!)
param_grid = dict(n_neighbors=k_range)
print(param_grid)
```

```
{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]}
```

In [67]:
```python
# TODO instantiate the grid
# Use the param_grid from above and pass it into GridSearchCV
# Refer to the documentation if needed (link above)

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10, scoring='accuracy')
```

In [68]:
```python
# TODO fit the grid with data
# Remember, we're fitting with our data (X, y)

grid.fit(X , y);
```

In [69]:
```python
# TODO examine the best model
# Just run this cell; you should have created grid in the cell above!

print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
0.8416666666666666
{'n_neighbors': 3}
KNeighborsClassifier(n_neighbors=3)
```

7) Reducing computational expense using `RandomizedSearchCV`

Instead of searching the entire search space (every possible combination of our hyperparameters), we can use a randomized search. A randomized grid search might not get us the **globally best model** but it will get us close enough! Refer to the second instructor webinar for more information on the difference between Grid Search and Randomized Grid Search.

In [70]:
```
# TODO run this cell for importing randomized grid search
from sklearn.model_selection import RandomizedSearchCV
```

For the grid search, we used a parameter *grid* because we wanted to search every possible value of `k`. But here, instead of a `param_grid`, we'll be using a `param_dist` (parameter *distribution*) to sample from.

What's happening here is that we're not going to try every single value from our range of `k`. So let's say our range of `k` is 1-30. In grid search, we try 1, 2, 3, 4, ..., 30. In randomized grid search, we'll use a distribution to *sample* values of `k` from 1-30.

Here we'll use a **uniform distribution**. A uniform distribution will make every value of k, from 1 to 30, equally likely to be pulled. To tell this to `sklearn`, we'll be using the input `weights=` in `RandomizedSearchCV`. We'll first create a helper variable, though, called `param_dist` that will create a dictionary that we'll pass in to randomized search.

We don't *have to* create this helper variable, but it's just easier to create a dictionary with all of our hyperparameters and pass just one variable into `RandomizedSearchCV`. In this case, our two hyperparameters are `n_neighbors` and `weights`.

In [71]:
```
# TODO specify "parameter distributions" rather than a "parameter grid"

k_range = list(range(1, 30))

param_dist = {
    'n_neighbors': k_range,
    'weights':['uniform','distance']
}
```

In [72]:
```
# n_iter controls the number of searches -- that is, how much of the potential grid dc
# n_iter default = 10
# TODO try running with different values of n_iter!

rand = RandomizedSearchCV(knn, param_dist, cv=10, scoring='accuracy', \
                          n_iter=20, random_state=42)
rand.fit(X, y)
```

Out[72]:

```
▸ RandomizedSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

In [73]:

```python
# TODO examine the best model from our random grid search

print(rand.best_score_)
print(rand.best_params_)
```

```
0.8766666666666666
{'weights': 'distance', 'n_neighbors': 3}
```

How well does a randomized search actually do? We can try running the randomized search multiple times and see how the output differs. If the output is similar in each run of the random search, we can be fairly confident that the random search will get us close enough to a full grid search.

In [75]:

```python
# TODO run RandomizedSearchCV 20 times (with n_iter=10) and record the best score

best_scores = []
knn = KNeighborsClassifier()

for _ in range(20):
    ##### your code here!
    # create a randomized grid search
    rand = RandomizedSearchCV(knn, param_distributions=param_dist, cv=10, scoring='acc
    # fit the rand search
    rand.fit(X, y)
    # append the best score from fit to best_scores
    rand = RandomizedSearchCV(knn, param_dist, cv=10, scoring='accuracy', n_iter=10)
    rand.fit(X, y)
    best_scores.append(round(rand.best_score_, 3))
print(best_scores)
```

```
[0.841, 0.847, 0.877, 0.846, 0.877, 0.856, 0.847, 0.856, 0.856, 0.872, 0.872, 0.877,
0.877, 0.877, 0.856, 0.856, 0.856, 0.856, 0.846, 0.877]
```

- I am confident that the random search will get me close enough to a full grid search because I am getting values between 85% and 88%.