Data Science for Python Folks Without (or With!) a Ph.D.

Douglas Starnes

Granite State Code Camp 2023

SHOWER
FAN
CONE
CENTER

Spoiler Alert

You need to remember only three ideas today

1) Data is worthless

2) Analytics are worth pennies

3) Decisions are worth dollars

Obligatory Narcissism Slide (who is this guy?)

- Hi! I’m Douglas!
- Memphis, TN area
- Entrepreneur and technical author
- 4x Microsoft MVP (DevTech/Python)
- Memphis Python & Memphis Azure
- Scenic City Summit
- TDevConf
- Trained composer, board game and LEGO collector

Commercial Break

- Memphis Azure User Group - virtual
  - Monday, Dec. 4 at 6:00 PM Central
    Matt Eland – Automating My Dog with Azure AI Services
    RSVP: https://bit.ly/memazug-dec-23
- Memphis Python User Group – virtual
  - Monday, Dec. 18 at 6:00 PM Central
    New Year's Resolutions – Launch That Startup
    RSVP: https://bit.ly/mempy-dec-23
- Interested in speaking?
  - douglasastarnes@outlook.com
  - https://linktr.ee/douglasstarnes (LinkedIn is best)
- DataTune 2024
  - March 8-9, 2024, Nashville, TN
    https://datatuneconf.com

Agenda

Python

numpy

pandas

matplotlib

Jupyter Notebook

Python

Why Python?

- According to GitHub, it is the 2nd most used language
- Simple, clean syntax
- Easy to learn, read, and remember
- Open source
- Cross platform
- Numerous applications
- Large number of 3rd party packages

Slicing strings

```
String sample = “Granite State Code Camp”

// “Granite” (first 7)
System.out.println(sample.Substring(0, 7));

// “State”
System.out.println(sample.Substring(8, 13));

// “Camp” (last 4)
System.out.println(sample.Substring(sample.length() – 4));
```

Java

```
sample = “Granite State Code Camp”

# “Granite” (first 7)
print(sample[:7]

# “State”
print(sample[8:13])

# “Camp” (last 4)
print(sample[-4:]
```

Python

Reversing a string

```
String sample = “Granite State Code Camp”;
char[] charArray = sample.toCharArray();
int left = 0;
int right = charArray.length - 1;

while (left < right) {
  char temp = charArray[left];
  charArray[left] = charArray[right];
  charArray[right] = temp;

  left++;
  right--;
}
```

Java

```
sample = “Granite State Code Camp”

reversed = “”

for c in sample:
  reversed = c + reversed

print(reversed)
```

Python

Reversing a string

```
String sample = “Granite State Code Camp”;
char[] charArray = sample.toCharArray();
int left = 0;
int right = charArray.length - 1;

while (left < right) {
  char temp = charArray[left];
  charArray[left] = charArray[right];
  charArray[right] = temp;

  left++;
  right--;
}
```

Java

```
sample = “Granite State Code Camp”

reversed = list(sample)

reversed.reverse()

print(“”.join(reversed))
```

Python

Reversing a string

```
String sample = “Granite State Code Camp”;
char[] charArray = sample.toCharArray();
int left = 0;
int right = charArray.length - 1;

while (left < right) {
  char temp = charArray[left];
  charArray[left] = charArray[right];
  charArray[right] = temp;

  left++;
  right--;
}
```

Java

```
sample = “Granite State Code Camp”

print(sample[::-1])
```

Python

Methods / Functions

```java
public static int[] divide(int a, int b) {
  int[] result = new int[2];
  result[0] = a / b;
  result[1] = a % b;
  return result;
}

public static void Main() {
  int[] result = divide(5, 3);
  int quotient = result[0];
  int remainder = result[1];
  // display
}
```

Java

```python
def divide(a, b):
  return (a // b, a % b)

result = divide(5, 3)

quotient = result[0]
remainder = result[1]

print(f“q: {quotient}, r: {remainder}”)
```

Python

Methods / Functions

```java
public static int[] divide(int a, int b) {
  int[] result = new int[2];
  result[0] = a / b;
  result[1] = a % b;
  return result;
}

public static void Main() {
  int[] result = divide(5, 3);
  int quotient = result[0];
  int remainder = result[1];
  // display
}
```

Java

```python
def divide(a, b):
  return (a // b, a % b)

quotient, remainder = divide(5, 3)

print(f“q: {quotient}, r: {remainder}”)
```

Python

Methods / Functions

```java
public static int[] divide(int a, int b) {
  int[] result = new int[2];
  result[0] = a / b;
  result[1] = a % b;
  return result;
}

public static void Main() {
  int[] result = divide(5, 3);
  int quotient = result[0];
  int remainder = result[1];
  // display
}
```

Java

```python
from collections import namedtuple

Result = namedtuple("Result", ["quotient", "remainder"])

def divide(a, b):
  return Result(a // b, a % b)

result = divide(5, 3)

print(f"q: {result.quotient}, r: {result.remainder}")
```

Python

Methods / Functions

```
public static int[] divide(int a, int b) {
  int[] result = new int[2];
  result[0] = a / b;
  result[1] = a % b;
  return result;
}

public static void Main() {
  int[] result = divide(5, 3);
  int quotient = result[0];
  int remainder = result[1];
  // display
}
```

Java

```
from collections import namedtuple

Result = namedtuple("Result", ["quotient", "remainder"])

def divide(a, b):
  return Result(a // b, a % b)

try:
  result = divide(5, 0)
  print(f"q: {result.quotient}, r: {result.remainder}")
except ZeroDivisionError as e:
  print(e)
```

Python

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

numpy

1) Data is worthless

```
nums = [2, 4, 8, 7, 3,
nums_inc = nums + 1
```

`TypeError`

Python doesn’t know if you want to append 1 to the list or add 1 to each element in the list

```
nums = [2, 4, 8, 7, 3, 1]
nums_inc = [num + 1 for num in nums]
nums_inc = nums + [1]
import numpy as np
num_arr = np.array(nums)
num_arr_inc = num_arr + 1
```

```
array([3, 5, 9, 8, 4, 2])
```

The fundamental data structure in `numpy` is the `ndarray`.

A numpy `ndarray` is like a Python `list` that has superpowers.

A numpy `ndarray` can have multiple dimensions.

The fundamental data structure in numpy is the `ndarray`.

A numpy ndarray is like a Python list that has superpowers.

A numpy ndarray can have multiple dimensions.

```
import random

def random_matrix(rows=5, col
    A = []
    for _ in range(rows):
        A.append([random.ran
    return A

M = random_matrix()
```

```
[[4, 4, 4, 3], [3, 1, 0, 4], [8, 9, 4, 8],
 [5, 7, 3, 4], [4, 2, 1, 7]]
```

The fundamental data structure in numpy is the `ndarray`.

A numpy ndarray is like a Python list that has superpo

A numpy ndarray can have multiple dimensions.

```
import numpy as np

M = np.random.randint(0, 11, (5, 4))
```

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

The fundamental data structure in numpy is the `ndarray`.

A numpy ndarray is like a Python list that has superpo

A numpy ndarray can have multiple dimensions.

```
import numpy as np

M = np.random.randint(0, 11, (5, 4))
```

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

You can access indi   elements and slice an ndarray using list syntax:

```
M[1][3]
```

```
1
```

The fundamental data structure in numpy is the `ndarray`.

A numpy ndarray is like a Python list that has superpo

A numpy ndarray can have multiple dimensions.

```
import numpy as np

M = np.random.randint(0, 11, (5, 4))
```

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

You can access individual elements and slice an ndarray using list syntax:

```
M[1][3]
```

An ndarray has a sp nd cleaner syntax for elements in multiple dimensions

```
M[1, 3]
```

```
1
```

The fundamental data structure in numpy is the `ndarray`.

A numpy ndarray is like a Python list that has superpo

A numpy ndarray can have multiple dimensions.

```
import numpy as np

M = np.random.randint(0, 11, (5, 4))
```

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

You can access individual elements and slice an ndarray using list syntax

```
M[1][3]
```

An ndarray has a special and cleaner syntax for elements in multiple dimensions

```
M[1, 3]
```

You can also use sli of an ndarray

```
M[:, 3]
```

```
array([ 9, 1, 10, 8, 3])
```

An ndarray is aware o

```
M.shape
```

(5, 4)

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of the new and old dimensions must be equal)

```
M.reshape(10, 2)
```

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

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
```

```
3
```

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
M[:, 0, :]
```

```
array([[0, 3, 7, 4],
       [5, 9, 5, 3]])
```

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t

equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
M[:, 0, :]
M.reshape(4, 2, 3)
```

```
array([[[ 0, 3, 7],
        [ 4, 0, 1]],

       [[ 4, 10, 10],
        [ 2, 9, 6]],

       [[ 5, 9, 5],
        [ 3, 5, 2]],

       [[ 6, 0, 6],
        [ 5, 1, 2]]])
```

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
M[:, 0, :]
M.reshape(4, 2, 3)
M.reshape(2, 2, 2, 3)
```

```
array([[[[ 0, 3, 7],
         [ 4, 0, 1]],

        [[ 4, 10, 10],
         [ 2, 9, 6]]],


       [[[ 5, 9, 5],
         [ 3, 5, 2]],

        [[ 6, 0, 6],
         [ 5, 1, 2]]]])
```

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
M[:, 0, :]
M.reshape(4, 2, 3)
M.reshape(2, 2, 2, 3)
M.reshape(8, 3)
```

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

An ndarray is aware of it’s shape

```
M.shape
```

The shape of an ndarray can be modified (the product of t equal)

```
M.reshape(10, 2)
```

The features of an ndarray scale to infinite dimensions

```
M = np.random.randint(0, 11, (2, 3, 4))
```

```
array([[[ 0, 3, 7, 4],
        [ 0, 1, 4, 10],
        [10, 2, 9, 6]],

       [[ 5, 9, 5, 3],
        [ 5, 2, 6, 0],
        [ 6, 5, 1, 2]]])
```

```
M[1, 0, 3]
M[:, 0, :]
M.reshape(4, 2, 3)
M.reshape(2, 2, 2, 3)
M.reshape(8, 3)
M.flatten()
```

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

pandas

2) Analytics are worth pennies

The fundamental data structure in pandas is the DataFrame.

You can create a DataFrame from many sources.

```
import pandas as pd

games_df = pd.read_csv(“games.csv”)
```

```
year,week,home,away,win,loss,pts_win,pts_loss
2000,1,Vikings,Bears,Vikings,Bears,30,27
2000,1,Chiefs,Colts,Colts,Chiefs,27,14
2000,1,Redskins,Panthers,Redskins,Panthers,20,17
2000,1,Falcons,49ers,Falcons,49ers,36,28
2000,1,Steelers,Ravens,Ravens,Steelers,16,0
2000,1,Browns,Jaguars,Jaguars,Browns,27,7
2000,1,Patriots,Buccaneers,Buccaneers,Patriots,21,16
```

- read_csv()
- read_excel()
- read_html()
- read_json()
- read_pickle()
- read_sql()

The fundamental data structure in pandas is the DataFrame.

You can create a DataFrame from many sources.

```
import pandas as pd

games_df = pd.read_

games_df.head()
```

```
   year  week      home      away       win      loss  pts_win  pts_loss
0  2000     1   Vikings     Bears   Vikings     Bears       30        27
1  2000     1    Chiefs     Colts     Colts    Chiefs       27        14
2  2000     1  Redskins  Panthers  Redskins  Panthers       20        17
3  2000     1   Falcons     49ers   Falcons     49ers       36        28
4  2000     1  Steelers    Ravens    Ravens  Steelers       16         0
```

The fundamental data structure in

You can create a DataFrame from

```
import pandas as pd

games_df = pd.read_csv(

games_df.head()

games_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5324 entries, 0 to 5323
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      5324 non-null   int64
 1   week      5324 non-null   int64
 2   home      5324 non-null   object
 3   away      5324 non-null   object
 4   win       5324 non-null   object
 5   loss      5324 non-null   object
 6   pts_win   5324 non-null   int64
 7   pts_loss  5324 non-null   int64
dtypes: int64(4), object(4)
memory usage: 332.9+ KB
```

```
games_df.describe()
```

```
              year         week      pts_win     pts_loss
count  5324.000000  5324.000000  5324.000000  5324.000000
mean   2009.527047     9.511833    27.781555    16.088843
std       5.754236     5.271909     8.830090     8.137451
min    2000.000000     1.000000     3.000000     0.000000
25%    2005.000000     5.000000    21.000000    10.000000
50%    2010.000000    10.000000    27.000000    16.000000
75%    2015.000000    14.000000    34.000000    21.000000
max    2019.000000    21.000000    62.000000    51.000000
```

The fundamental data structure in pandas is the DataFrame.

You can create a DataFrame from many sources.

```
import pandas as pd

games_df = pd.read_csv(“games.csv”)

games_df.head()

games_df.info()

games_df.describe()
```

Individual columns of a DataFra

```
games_df[“home”]
```

```
0         Vikings
1          Chiefs
2        Redskins
3         Falcons
4        Steelers
           ...
5319       Chiefs
5320      Packers
5321       Chiefs
5322        49ers
5323       Chiefs
Name: home, Length: 5324, dtype: object
```

The fundamental data structure in pandas is the DataFrame.

You can create a DataFrame from many sources.

```
        import pandas as pd

        games_df = pd.read_csv

        games_df.head()

        games_df.info()

        games_df.describe()
```

Individual columns of a DataFra

```
        games_df[“home”]
```

```
0        Vikings
1         Chiefs
2       Redskins
3        Falcons
4       Steelers
             ...
5319      Chiefs
5320     Packers
5321      Chiefs
5322       49ers
5323      Chiefs
Name: home, Length: 5324, dtype: object
```

The data structure for a column is a Series, which is like a Python list with an index

A DataFrame is a collection of Series that share an index

You can access individual rows by index:

```
games_df.loc[1]
```

Or by zero based position

```
games_df.iloc[1]
```

In this case they are the same.

A row is a `Series`

```
year          2000
week             1
home        Chiefs
away         Colts
win          Colts
loss        Chiefs
pts_win         27
pts_loss        14
Name: 1, dtype: object
```

You can access individual rows by index:

```
games_df.loc[1]
```

Or by zero based position

```
games_df.iloc[1]
```

In this case they are the same.

A row is a `Series`

You can also slice the rows, like a Python `list` (returns a `DataFrame`)

```
games_df.iloc[1:5]
```

```
   year  week      home     away      win     loss  pts_win  pts_loss
1  2000     1    Chiefs    Colts    Colts   Chiefs       27        14
2  2000     1  Redskins Panthers Redskins Panthers       20        17
3  2000     1   Falcons    49ers  Falcons    49ers       36        28
4  2000     1  Steelers   Ravens   Ravens Steelers       16         0
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
```

```
(False, True)
```

```
    year  week     home    away      win    loss  pts_win  pts_loss
0   2000     1  Vikings   Bears  Vikings   Bears       30        27
22  2000     2   Titans  Chiefs   Titans  Chiefs       17        14
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
games_df[titans_home_games]
```

```
      year  week    home        away     win        loss  pts_win  pts_loss
22    2000     2  Titans      Chiefs  Titans      Chiefs       17        14
62    2000     5  Titans      Giants  Titans      Giants       28        14
99    2000     7  Titans     Jaguars  Titans     Jaguars       27        13
134   2000    10  Titans    Steelers  Titans    Steelers        9         7
150   2000    11  Titans      Ravens  Ravens      Titans       24        23
...    ...   ...     ...         ...     ...         ...      ...       ...
5169  2019     8  Titans  Buccaneers  Titans  Buccaneers       27        23
5197  2019    10  Titans      Chiefs  Titans      Chiefs       35        32
5229  2019    12  Titans     Jaguars  Titans     Jaguars       42        20
5272  2019    15  Titans      Texans  Texans      Titans       24        21
5287  2019    16  Titans      Saints  Saints      Titans       38        28

[163 rows x 8 columns]
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
games_df[titans_home_games]
```

Use the and (&) and or (|) operators to combine filters:

```
titans_wins = games_df[“win”] == “Titans”
titans_home_wins = games_df[titans_home_games & titans_wins]
```

```
      year  week    home        away     win        loss  pts_win  pts_loss
22    2000     2  Titans      Chiefs  Titans      Chiefs       17        14
62    2000     5  Titans      Giants  Titans      Giants       28        14
99    2000     7  Titans     Jaguars  Titans     Jaguars       27        13
134   2000    10  Titans    Steelers  Titans    Steelers        9         7
165   2000    12  Titans      Browns  Titans      Browns       24        10
...    ...   ...     ...         ...     ...         ...      ...       ...
5014  2018    16  Titans    Redskins  Titans    Redskins       25        16
5158  2019     7  Titans    Chargers  Titans    Chargers       23        20
5169  2019     8  Titans  Buccaneers  Titans  Buccaneers       27        23
5197  2019    10  Titans      Chiefs  Titans      Chiefs       35        32
5229  2019    12  Titans     Jaguars  Titans     Jaguars       42        20

[88 rows x 8 columns]
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
games_df[titans_home_games]
```

Use the and (&) and or (|) operators to combine filters:

```
titans_wins = games_df[“win”] == “Titans”
titans_home_wins = games_df[titans_home_games & titans_wins]
```

Compute statistics on a column:

```
titans_home_wins[“pts_win”].mean()
```

```
26.806818181818183
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
games_df[titans_home_games]
```

Use the and (&) and or (|) operators to combine filters:

```
titans_wins = games_df[“win”] == “Titans”
titans_home_wins = games_df[titans_home_games & titans_wins]
```

Compute statistics on a column:

```
titans_home_wins[“pts_win”].mean()
```

Add a new column:

```
titans_home_wins[“spread”] = titans_home_wins[“pts_win”] – titans_home_wins[“pts_loss”]
titans_home_wins.head()
```

```
     year  week    home     away     win     loss  pts_win  pts_loss  spread
22   2000     2  Titans   Chiefs  Titans   Chiefs       17        14       3
62   2000     5  Titans   Giants  Titans   Giants       28        14      14
99   2000     7  Titans  Jaguars  Titans  Jaguars       27        13      14
134  2000    10  Titans Steelers  Titans Steelers        9         7       2
165  2000    12  Titans   Browns  Titans   Browns       24        10      14
```

You can filter a column with a Boolean expression:

```
titans_home_games = games_df[“home”] == “Titans”
titans_home_games[0], titans_home_games[22]
games_df[titans_home_games]
```

Use the and (&) and or (|) operators to combine filters:

```
titans_wins = games_df[“win”] == “Titans”
titans_home_wins = games_df[titans_home_games & titans_wins]
```

Compute statistics on a column:

```
titans_home_wins[“pts_win”].mean()
```

Add a new column:

```
titans_home_wins[“s

titans_home_wins.he
```

Sort by spread descending:

```
titans_home_wins.sort_values(“spread”, ascending=False).head()
```

```
      year  week    home     away     win     loss  pts_win  pts_loss  spread
2590  2009    14  Titans     Rams  Titans     Rams       47         7      40
210   2000    15  Titans  Bengals  Titans  Bengals       35         3      32
247   2000    17  Titans  Cowboys  Titans  Cowboys       31         0      31
2661  2010     1  Titans  Raiders  Titans  Raiders       38        13      25
3512  2013     4  Titans     Jets  Titans     Jets       38        13      25
```

You can group a DataFrame by one or more columns

```
pts_grps = games_df.groupby([“year”, “win”])
```

And compute summary statistics

```
pts_sum = pts_grps.sum()[[“pts_win”, “pts_loss”]]
```

The index has multiple values

```
pts_sum.loc[2019, “Titans”]
```

```
pts_win     350
pts_loss    195
Name: (2019, Titans),
dtype: int64
```

```
               pts_win  pts_loss
year win
2000 49ers         167        74
     Bears         114        95
     Bengals        84        51
     Bills         192       131
     Broncos       380       226
...                ...       ...
2019 Seahawks      333       263
     Steelers      180       110
     Texans        294       214
     Titans        350       195
     Vikings       329       179

[636 rows x 2 columns]
```

You can group a DataFrame by one or more columns

```
pts_grps = games_df.groupby([“year”, “win”])
```

And compute summary statistics

```
pts_sum = pts_grps.sum()[[“pts_win”, “pts_loss”]]
```

The index has multiple values

```
pts_sum.loc[2019, “Titans”]
```

The DataFrame can be sorted with the grouped columns

```
pts_sum.sort_values([“year”, “win”], ascending=[True, False])
```

```
               pts_win  pts_loss
year win
2000 Raiders       414       191
     Ravens        407       136
     Rams          400       272
     Broncos       380       226
     Vikings       331       216
...                ...       ...
2019 Dolphins      144       120
     Giants        133        89
     Lions          98        87
     Redskins       65        53
     Bengals        55        29

[636 rows x 2 columns]
```

matplotlib

3) Decisions are worth dollars

Let’s look at the weekly attendance of Titans games in 2000

```
import pandas as pd

attend_df = pd.read_csv(“attendance.csv”)

titans_attendance = attend_df[attend_df[“team_name”] == “Titans”]
```

```
            team team_name  year    total    home    away  week  weekly_attendance
493    Tennessee    Titans  2000  1091274  547524  543750     1            72492.0
494    Tennessee    Titans  2000  1091274  547524  543750     2            68203.0
495    Tennessee    Titans  2000  1091274  547524  543750     3                NaN
496    Tennessee    Titans  2000  1091274  547524  543750     4            51769.0
497    Tennessee    Titans  2000  1091274  547524  543750     5            68341.0
...          ...       ...   ...      ...     ...     ...   ...                ...
10824  Tennessee    Titans  2019  1047496  516074  531422    13            60361.0
10825  Tennessee    Titans  2019  1047496  516074  531422    14            52760.0
10826  Tennessee    Titans  2019  1047496  516074  531422    15            65265.0
10827  Tennessee    Titans  2019  1047496  516074  531422    16            66756.0
10828  Tennessee    Titans  2019  1047496  516074  531422    17            71794.0

[340 rows x 8 columns]
```

Let’s look at the weekly attendance of Titans games in 2000

```
import pandas as pd

attend_df = pd.read_csv(“attendance.csv”)

titans_attendance = attend_df[attend_df[“team_name”] == “Titans”]


bye_weeks = titans_attendance[“weekly_attendance”].isna()
titans_attendance.drop(titans_attendance[bye_weeks].index, inplace=True)
```

```
            team team_name  year    total    home    away  week  weekly_attendance
493    Tennessee    Titans  2000  1091274  547524  543750     1            72492.0
494    Tennessee    Titans  2000  1091274  547524  543750     2            68203.0
496    Tennessee    Titans  2000  1091274  547524  543750     4            51769.0
497    Tennessee    Titans  2000  1091274  547524  543750     5            68341.0
498    Tennessee    Titans  2000  1091274  547524  543750     6            63406.0
...          ...       ...   ...      ...     ...     ...   ...                ...
10824  Tennessee    Titans  2019  1047496  516074  531422    13            60361.0
10825  Tennessee    Titans  2019  1047496  516074  531422    14            52760.0
10826  Tennessee    Titans  2019  1047496  516074  531422    15            65265.0
10827  Tennessee    Titans  2019  1047496  516074  531422    16            66756.0
10828  Tennessee    Titans  2019  1047496  516074  531422    17            71794.0

[320 rows x 8 columns]
```

Let’s look at the weekly attendance of Titans games in 2000

```
import pandas as pd

att

ti


by
ti


ti
```

TIMES UP!

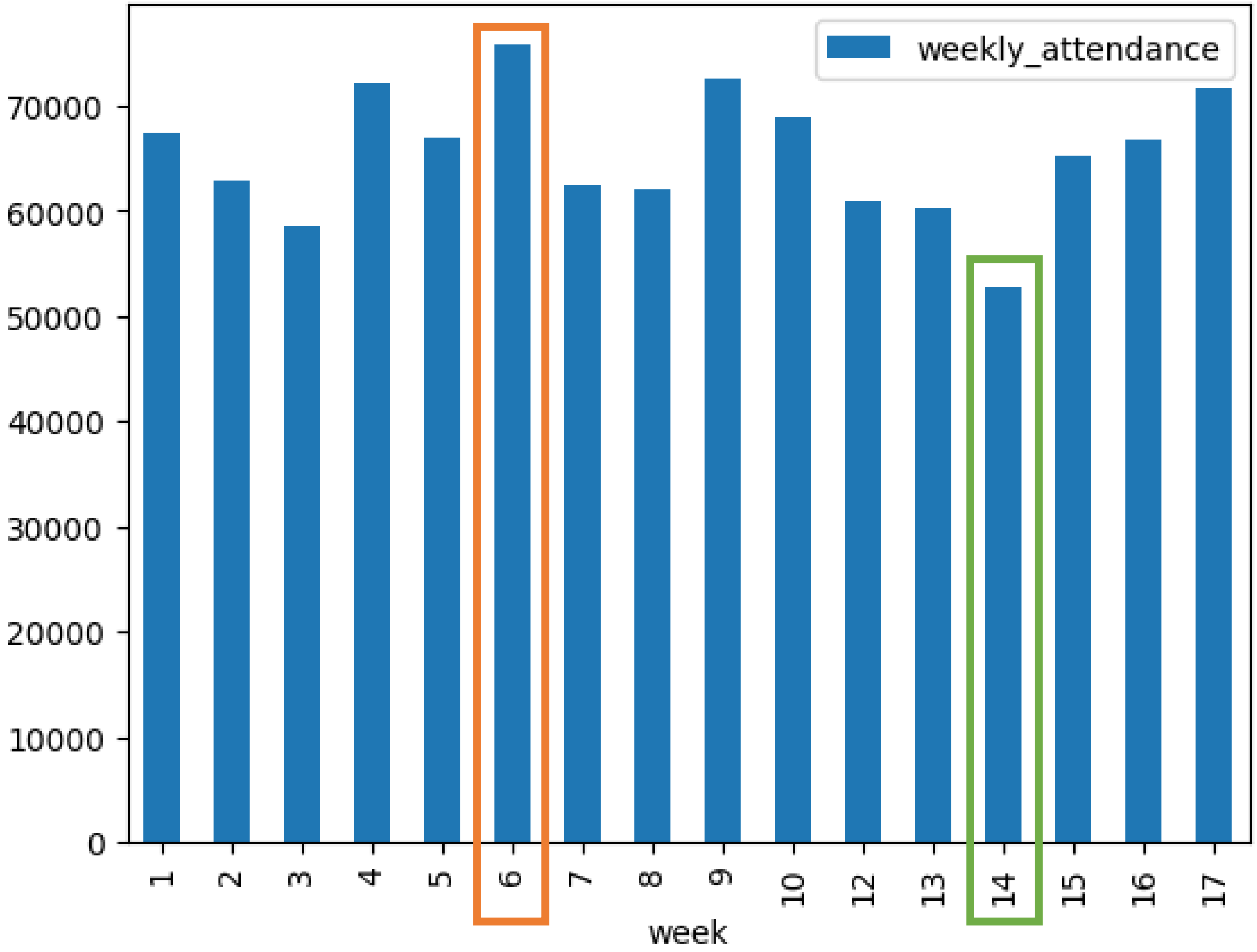
Which week in the 2000 season did the Titans have the highest attendance?

Let’s look at the weekly attendance of Titans games in 2000

```
import pandas as pd

att

ti


by
ti


ti
```

```
          team team_name  year    total    home    away  week  weekly_attendance
493  Tennessee    Titans  2000  1091274  547524  543750     1            72492.0
494  Tennessee    Titans  2000  1091274  547524  543750     2            68203.0
496  Tennessee    Titans  2000  1091274  547524  543750     4            51769.0
497  Tennessee    Titans  2000  1091274  547524  543750     5            68341.0
498  Tennessee    Titans  2000  1091274  547524  543750     6            63406.0
499  Tennessee    Titans  2000  1091274  547524  543750     7            68498.0
500  Tennessee    Titans  2000  1091274  547524  543750     8            69200.0
501  Tennessee    Titans  2000  1091274  547524  543750     9            83472.0
502  Tennessee    Titans  2000  1091274  547524  543750    10            68498.0
503  Tennessee    Titans  2000  1091274  547524  543750    11            68490.0
504  Tennessee    Titans  2000  1091274  547524  543750    12            68498.0
505  Tennessee    Titans  2000  1091274  547524  543750    13            65454.0
506  Tennessee    Titans  2000  1091274  547524  543750    14            65639.0
507  Tennessee    Titans  2000  1091274  547524  543750    15            68498.0
508  Tennessee    Titans  2000  1091274  547524  543750    16            72318.0
509  Tennessee    Titans  2000  1091274  547524  543750    17            68498.0
```

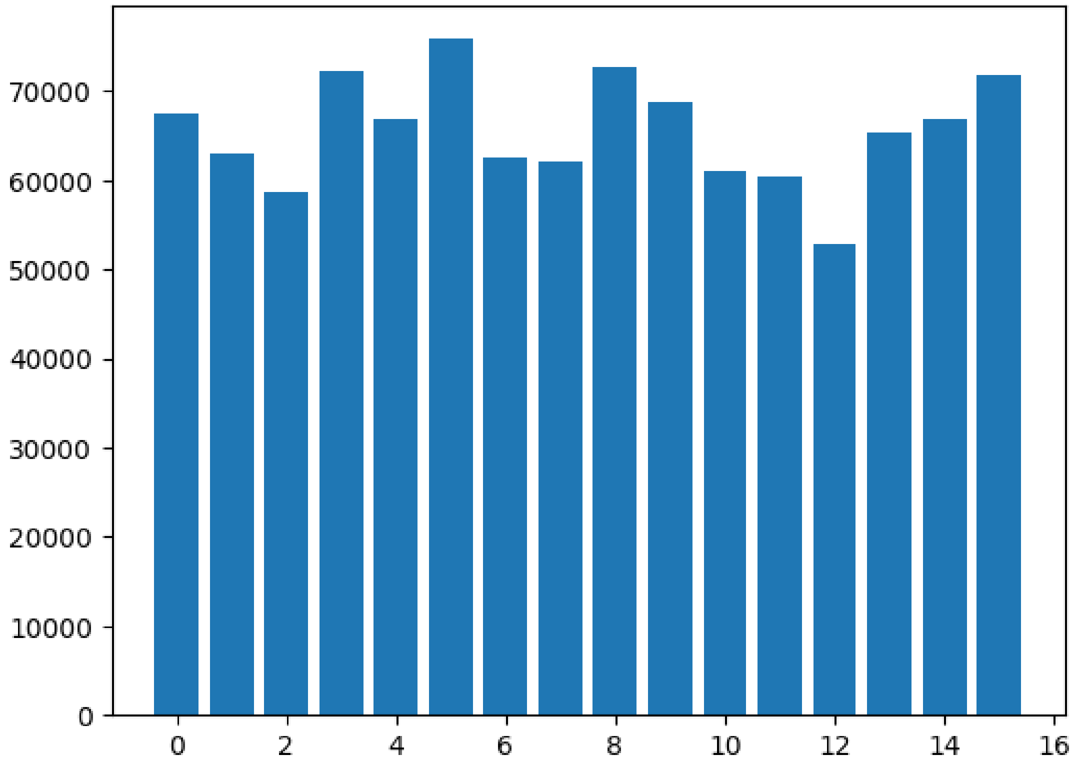
Which week in the 2000 season did the Titans have the highest attendance?



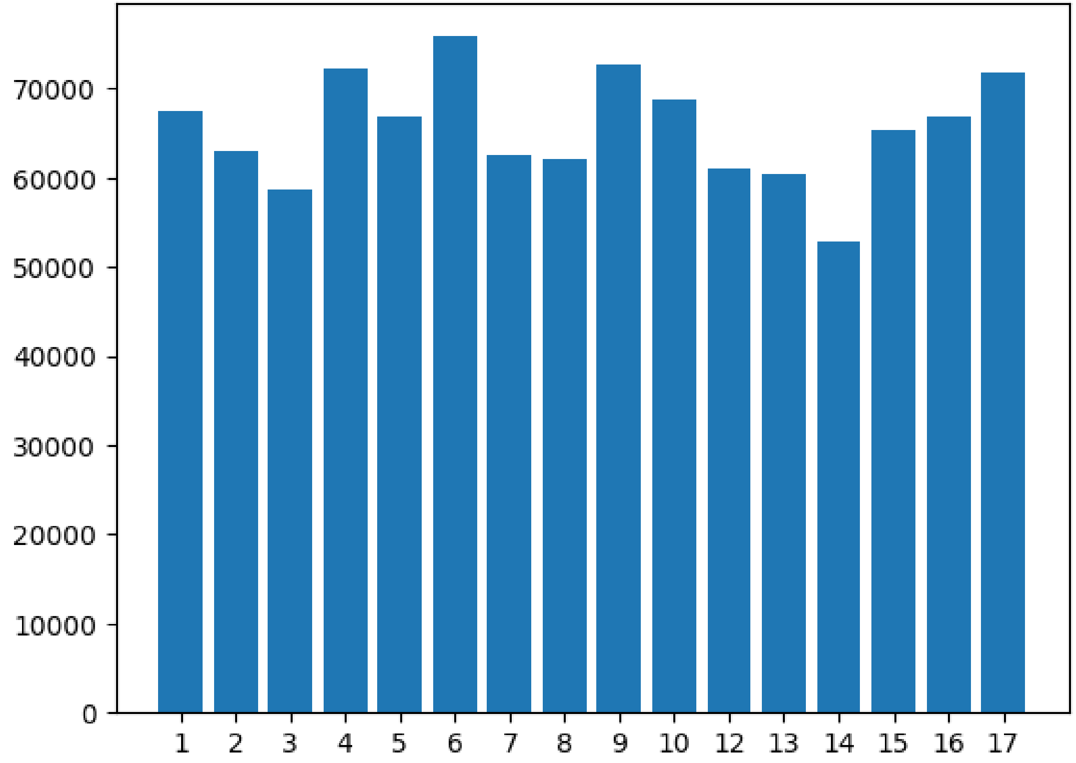
Which week in the 2019 season did the Titans have the highest attendance?

Which week in the 2019 season did the Titans have the lowest attendance?

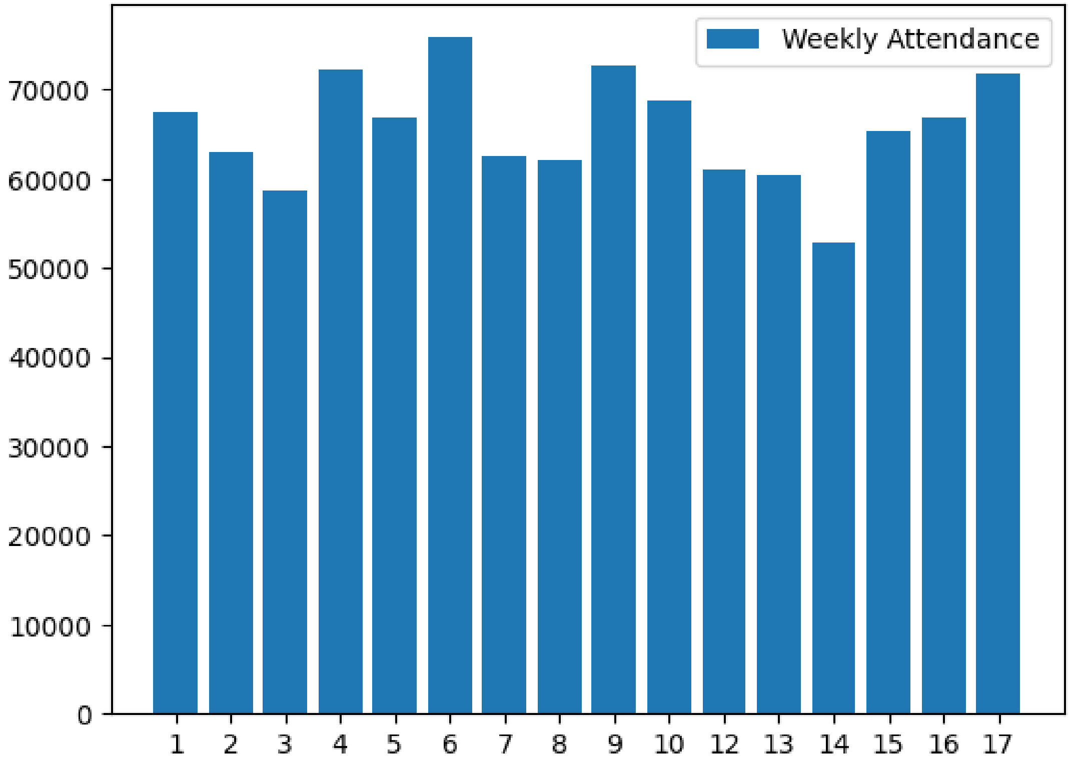
```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2019)),
    titans_attendance_2019[“weekly_attendance”])
```



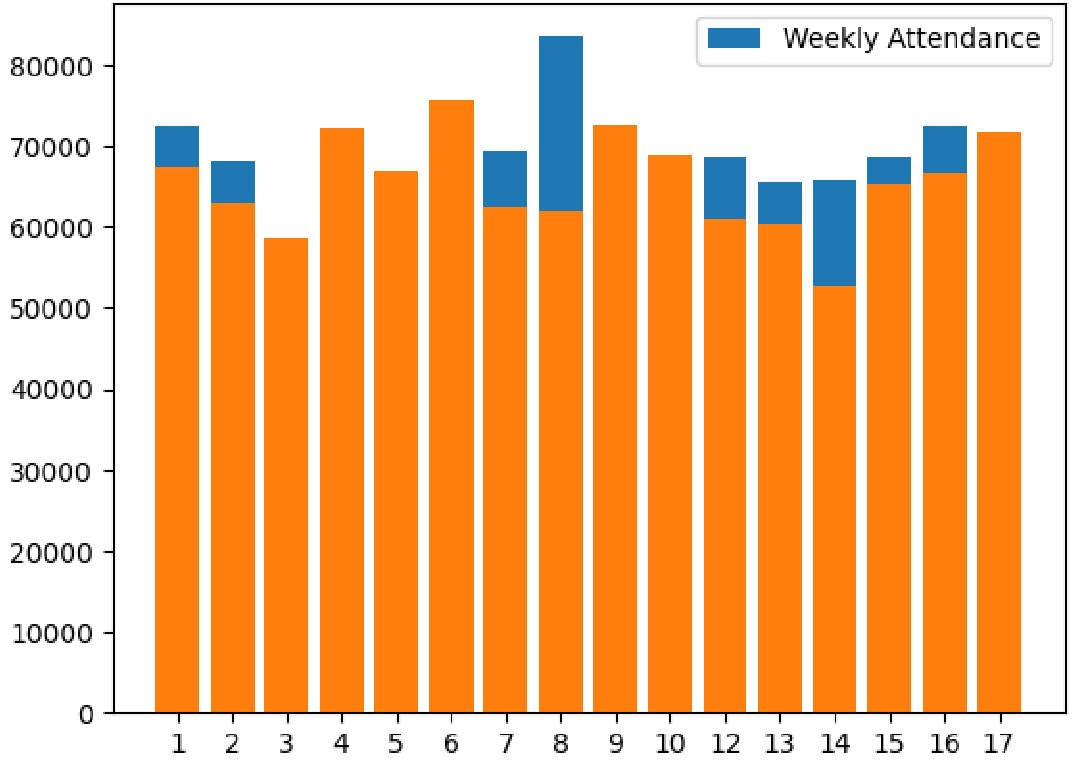
```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2019)),
    titans_attendance_2019[“weekly_attendance”])

plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])
```



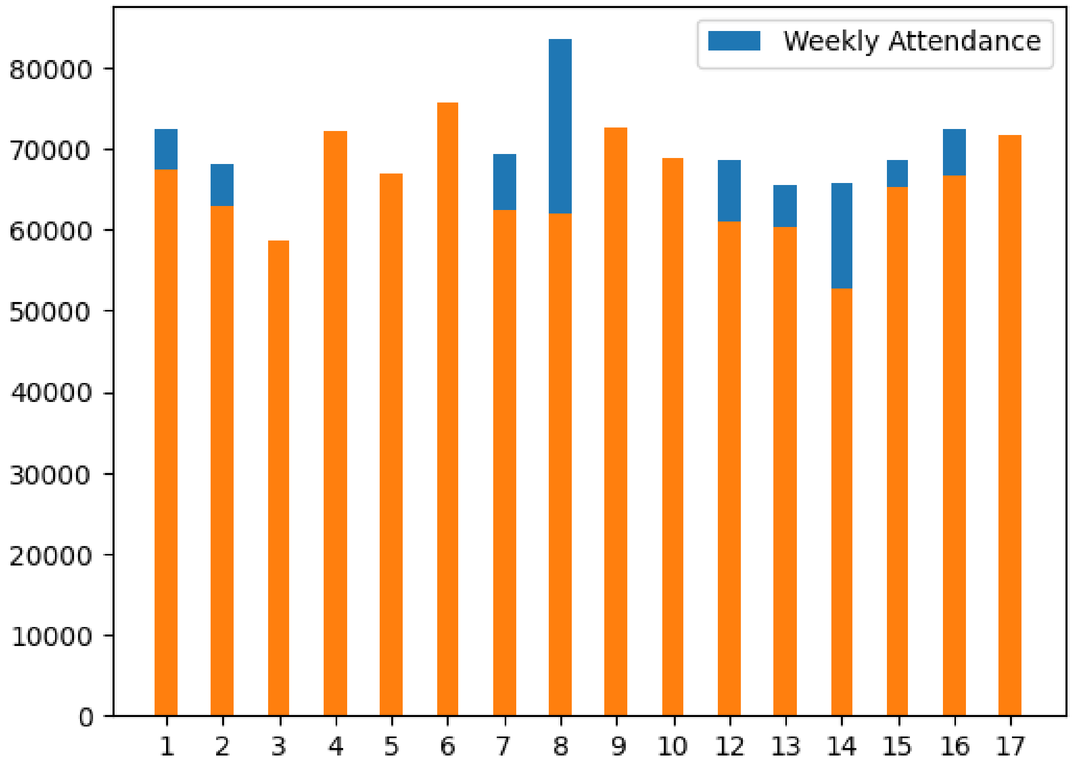
```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2019)),
    titans_attendance_2019[“weekly_attendance”])

plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])

plt.legend([“Weekly Attendance”])
```



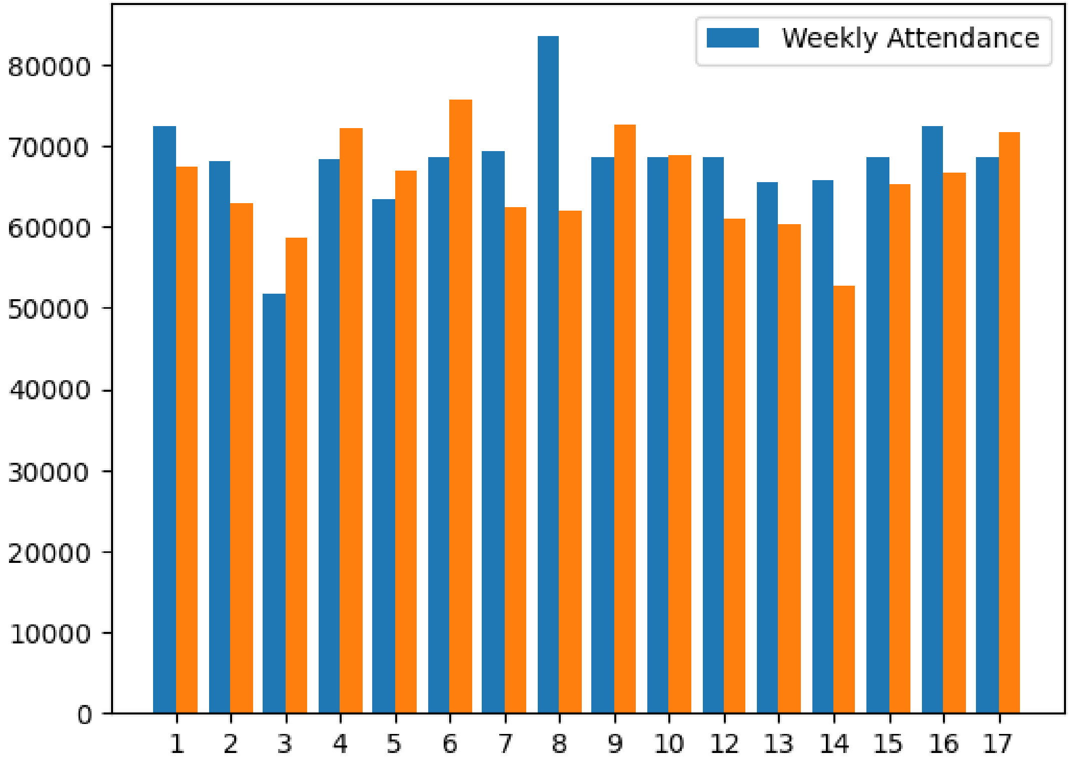
```
import matplotlib.pyplot as plt
plt.bar(
    np.arange(len(titans_attendance_2000)),
    titans_attendance_2000[“weekly_attendance”])
plt.bar(
    np.arange(len(titans_attendance_2019)),
    titans_attendance_2019[“weekly_attendance”])
plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])
plt.legend([“Weekly Attendance”])
```



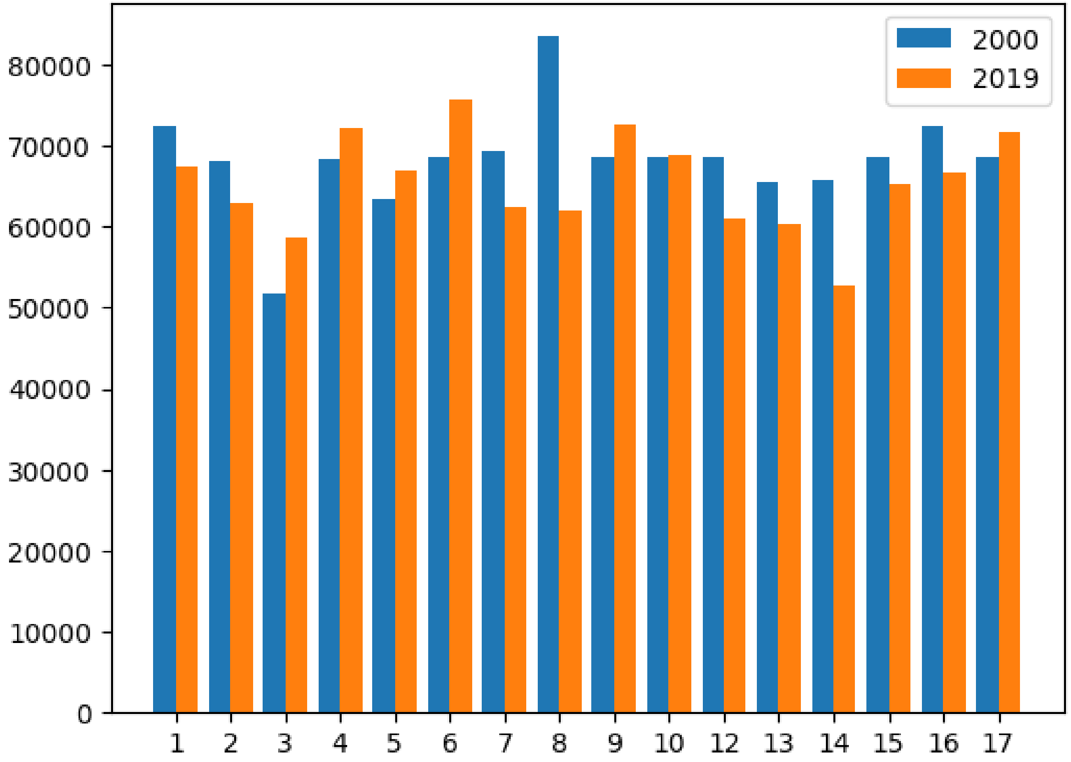
```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2000)),
    titans_attendance_2000[“weekly_attendance”], width=0.4)
plt.bar(
    np.arange(len(titans_attendance_2019)),
    titans_attendance_2019[“weekly_attendance”], width=0.4)

plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])

plt.legend([“Weekly Attendance”])
```



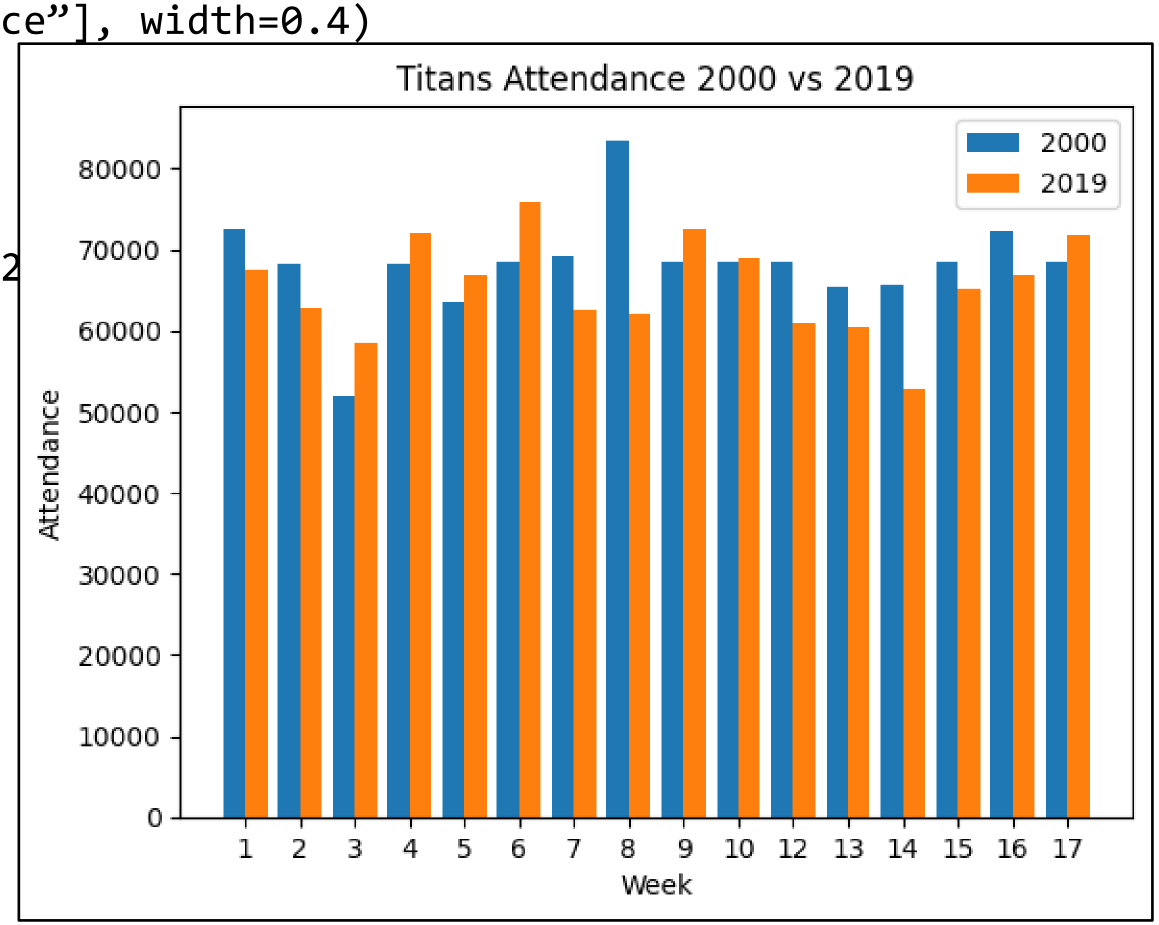
```
import matplotlib.pyplot as plt
plt.bar(
    np.arange(len(titans_attendance_2000)) - 0.2,
    titans_attendance_2000[“weekly_attendance”], width=0.4)
plt.bar(
    np.arange(len(titans_attendance_2019)) + 0.2,
    titans_attendance_2019[“weekly_attendance”], width=0.4)
plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])
plt.legend([“Weekly Attendance”])
```



```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2000)) – 0.2,
    titans_attendance_2000[“weekly_attendance”], width=0.4)
plt.bar(
    np.arange(len(titans_attendance_2019)) + 0.2,
    titans_attendance_2019[“weekly_attendance”], width=0.4)

plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])

plt.legend([“2000”, “2019”])
```



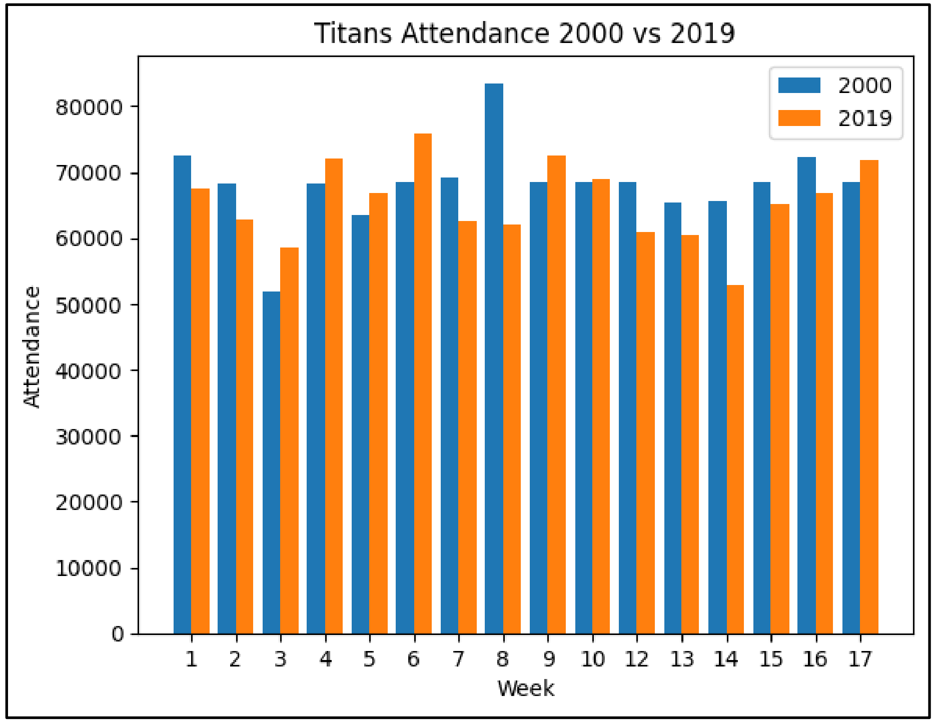
```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2000)) – 0.2,
    titans_attendance_2000[“weekly_attendance”], width=0.4)
plt.bar(
    np.arange(len(titans_attendance_2019)) + 0.2,
    titans_attendance_2019[“weekly_attendance”], width=0.4)

plt.title("Titans Attendance 2000 vs 2019")
plt.xlabel("Week")
plt.ylabel("Attendance")

plt.xticks(np.arange(len(titans_attendance_2

plt.legend([“2000”, “2019”])
```



```
import matplotlib.pyplot as plt

plt.bar(
    np.arange(len(titans_attendance_2000)) – 0.2,
    titans_attendance_2000[“weekly_attendance”], width=0.4)
plt.bar(
    np.arange(len(titans_attendance_2019)) + 0.2,
    titans_attendance_2019[“weekly_attendance”], width=0.4)

plt.title("Titans Attendance 2000 vs 2019")
plt.xlabel("Week")
plt.ylabel("Attendance")

plt.xticks(np.arange(len(titans_attendance_2019)), titans_attendance_2019[“week”])

plt.legend([“2000”, “2019”])
```

```
titans_no_game = attend_df[
    (attend_df[“team_name”] == “Titans”) & (attend_df[“weekly_attendance”].isna())]
titans_no_game[titans_no_game[“year”].isin([2000, 2019])]
```

```
            team team_name  year    total    home    away  week  weekly_attendance
495    Tennessee    Titans  2000  1091274  547524  543750     3                NaN
10822  Tennessee    Titans  2019  1047496  516074  531422    11                NaN
```



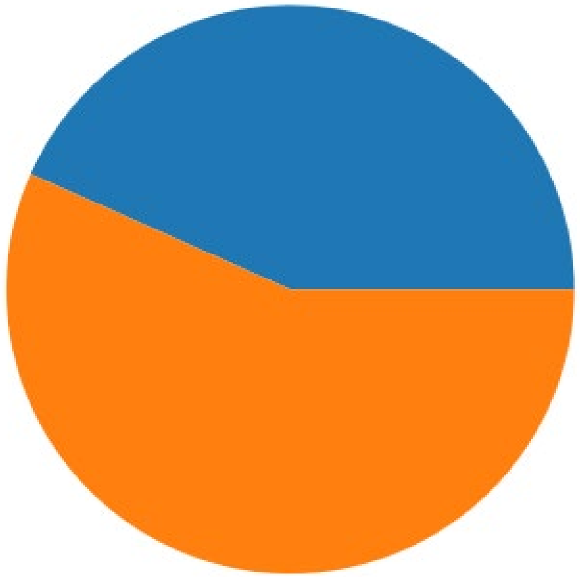
Let’s get the home and away attendance for each team in 2000

```
home_away_2000 = attend_df[(attend_df["year"] == 2000) & (attend_df["week"] == 1)]
```

```
          team  team_name  year    total    home    away  week  weekly_attendance
0      Arizona  Cardinals  2000   893926  387475  506451     1            77434.0
17     Atlanta    Falcons  2000   964579  422814  541765     1            54626.0
34   Baltimore     Ravens  2000  1062373  551695  510678     1            55049.0
51     Buffalo      Bills  2000  1098587  560695  537892     1            72492.0
68    Carolina   Panthers  2000  1095192  583489  511703     1            80257.0
```

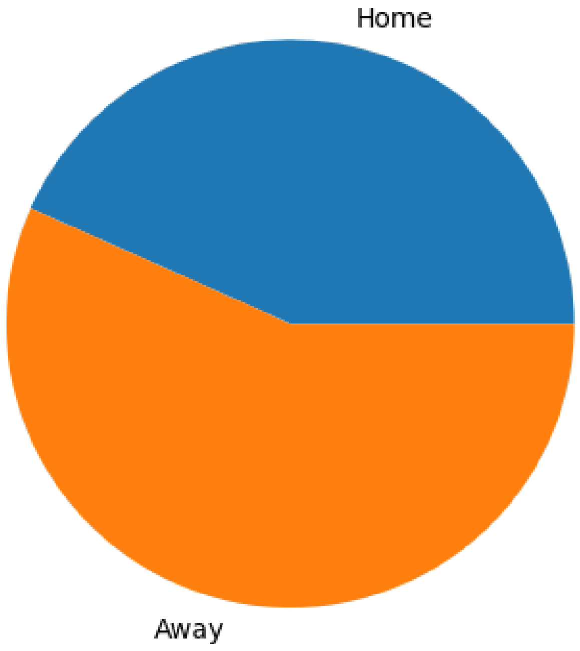
Let’s get the home and away attendance for each team in 2000

```
home_away_2000 = attend_df[(attend_df["year"] == 2000) & (attend_df["week"] == 1)]
cardinals_2000 = home_away_2000[home_away_2000["team_name"] == "Cardinals"].iloc[0]
plt.pie([cardinals_2000["home"], cardinals_2000["away"]])
```



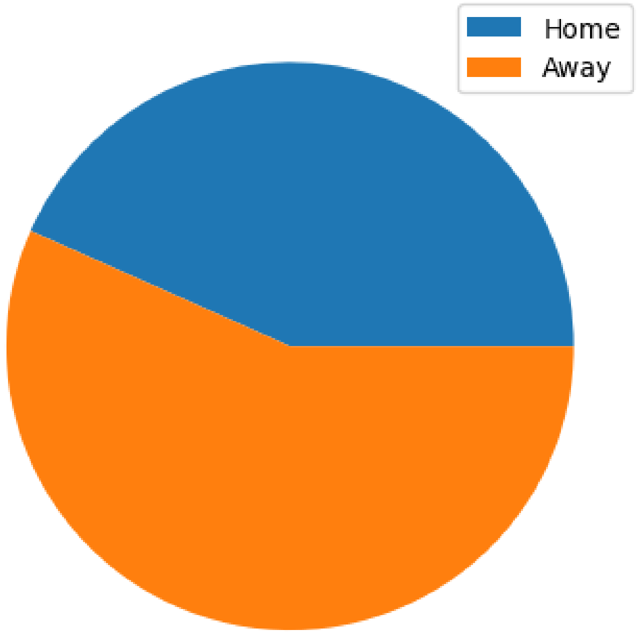
Let’s get the home and away attendance for each team in 2000

```
home_away_2000 = attend_df[(attend_df["year"] == 2000) & (attend_df["week"] == 1)]
cardinals_2000 = home_away_2000[home_away_2000["team_name"] == "Cardinals"].iloc[0]
plt.pie([cardinals_2000["home"], cardinals_2000["away"]], labels = [“Home”, “Away”])
```

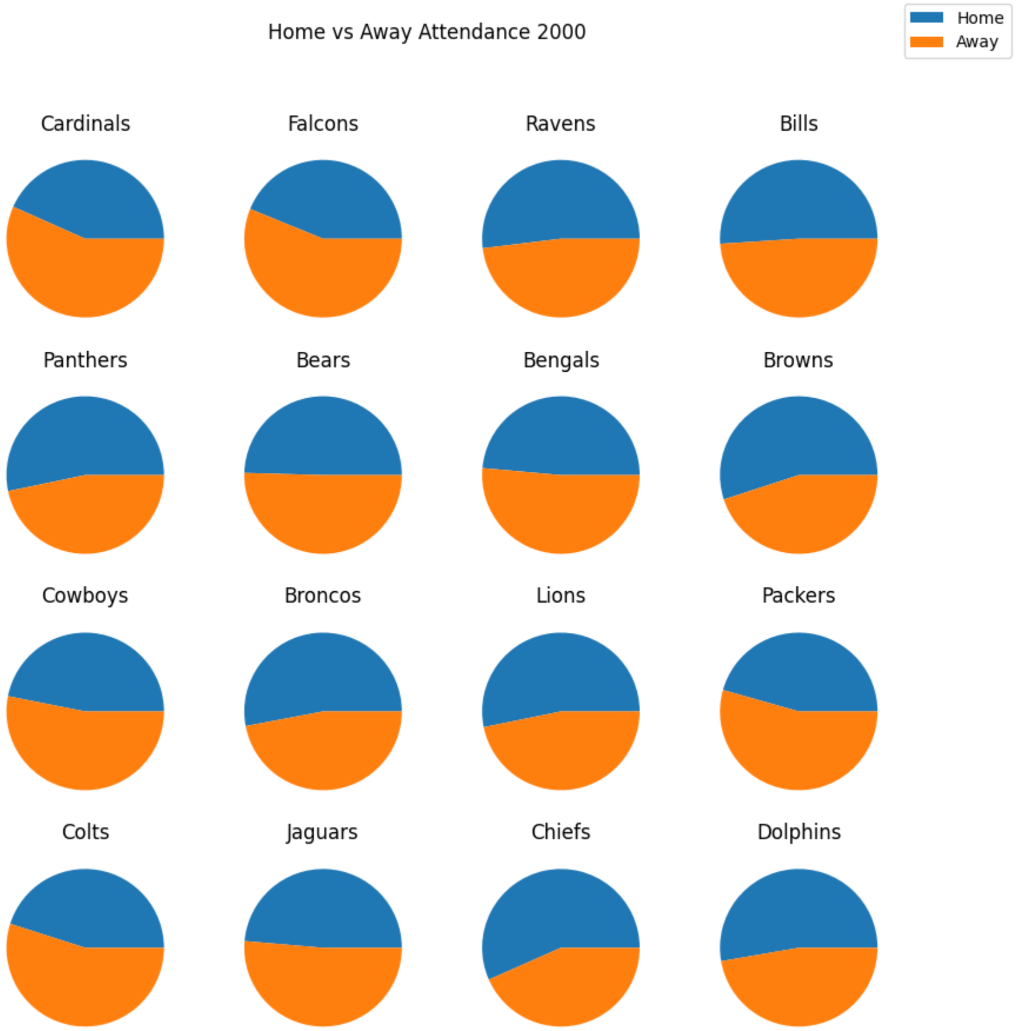


Let’s get the home and away attendance for each team in 2000

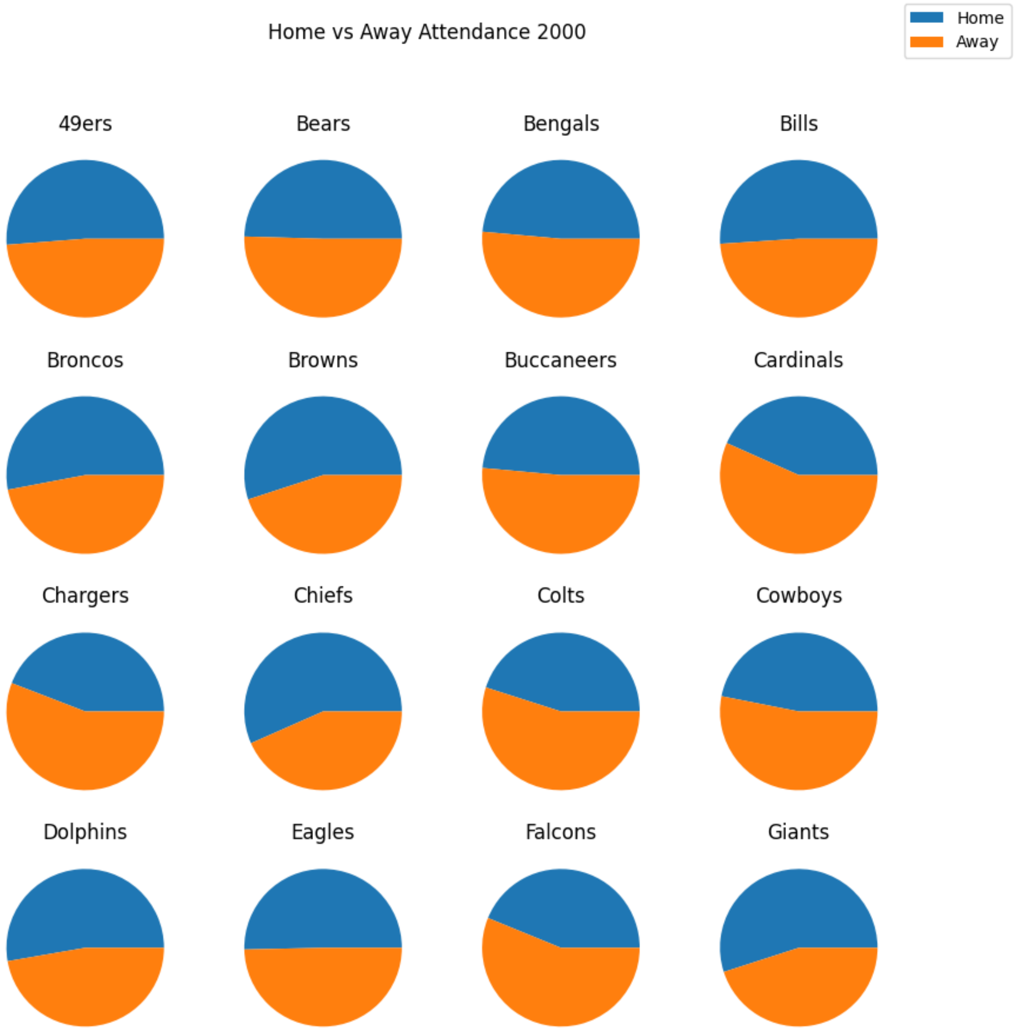
```
home_away_2000 = attend_df[(attend_df["year"] == 2000) & (attend_df["week"] == 1)]
cardinals_2000 = home_away_2000[home_away_2000["team_name"] == "Cardinals"].iloc[0]
plt.pie([cardinals_2000["home"], cardinals_2000["away"]])
plt.legend([“Home”, “Away”])
```



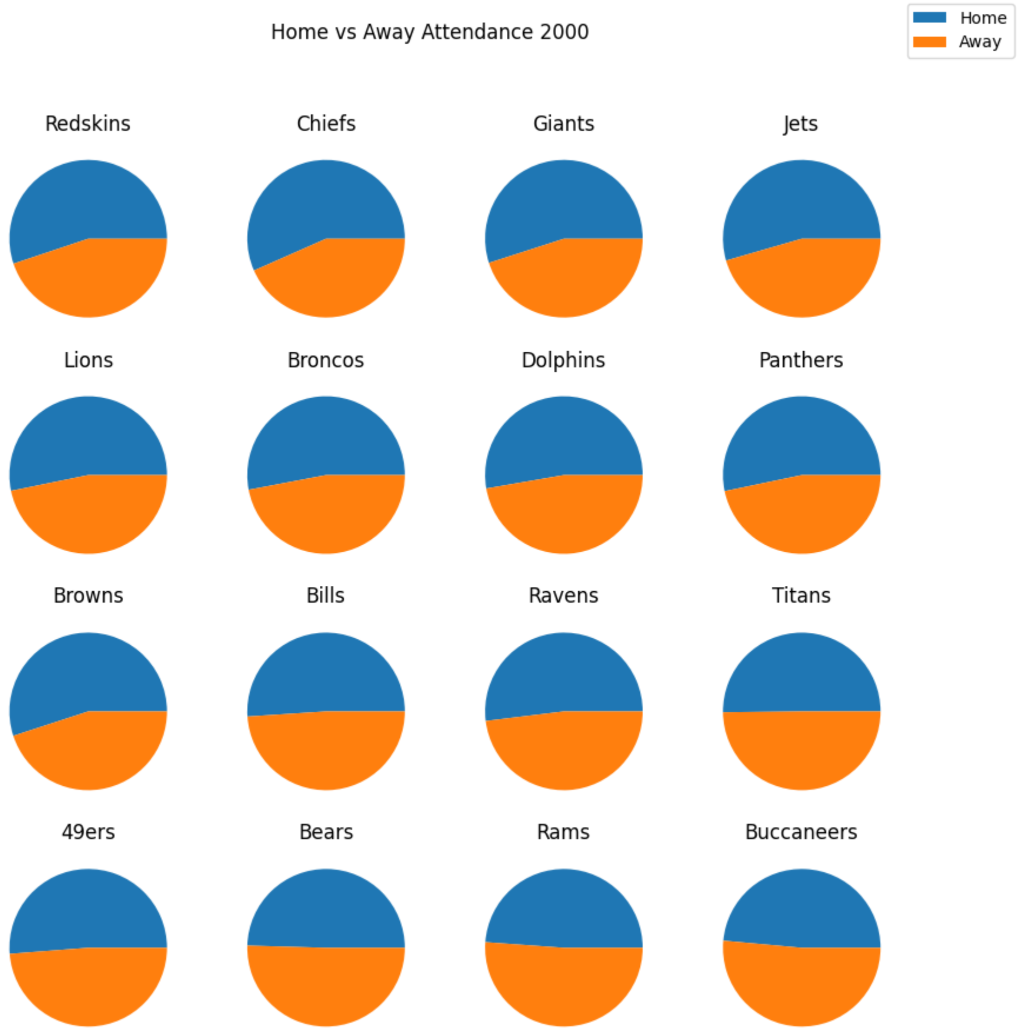
```
fig, ax = plt.subplots(4, 4, figsize=(10, 10))
for idx, row in enumerate(home_away_2000.iloc[:16].index):
    team_name = home_away_2000.loc[row, "team_name"]
    attendance = home_away_2000.loc[row, ["home", "away"]]
    ax[idx//4, idx%4].pie(attendance)
    ax[idx//4, idx%4].set_title(team_name)
fig.legend(["Home", "Away"])
fig.suptitle("Home vs Away Attendance 2000")
```



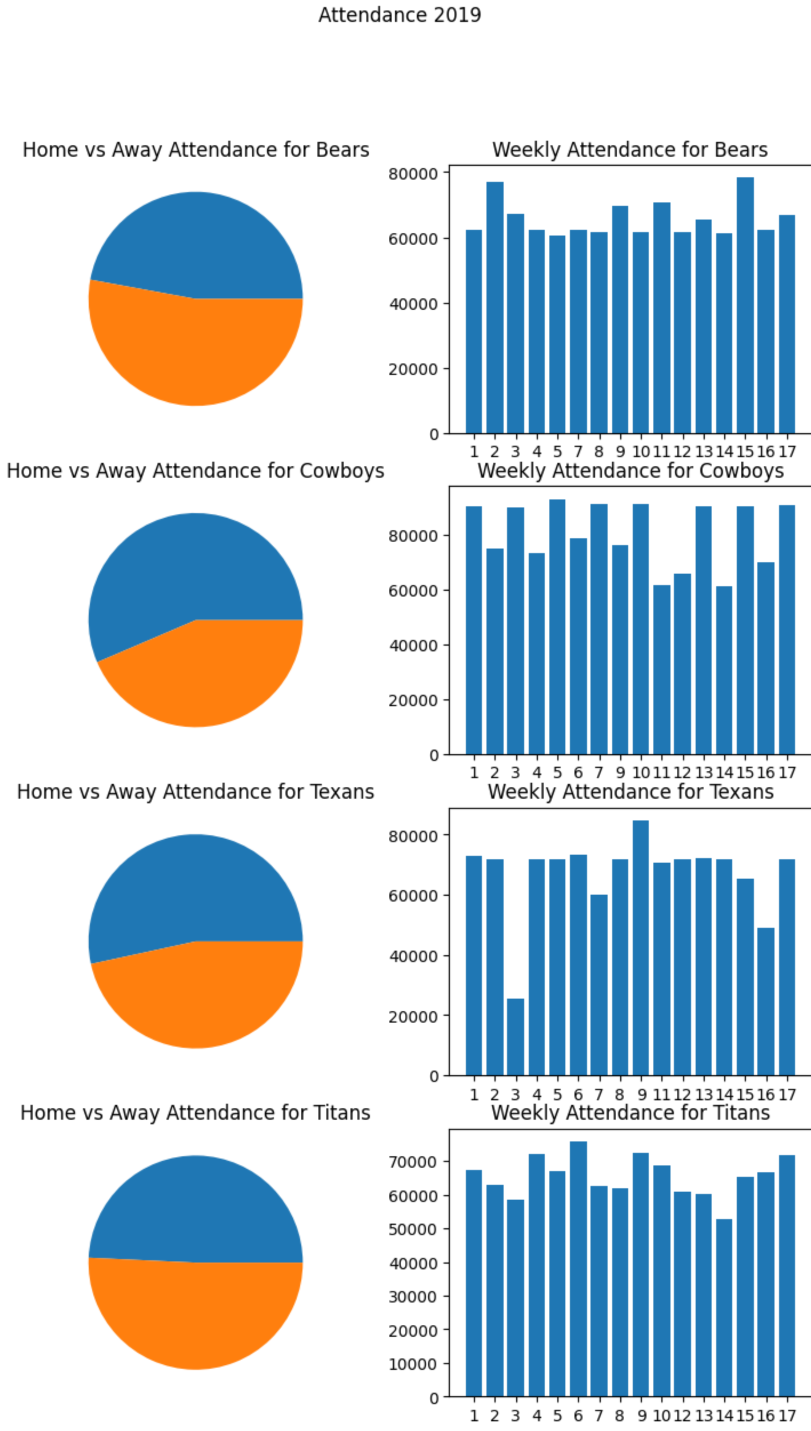
```
fig, ax = plt.subplots(4, 4, figsize=(10, 10))
for idx, row in enumerate(home_away_2000.sort_values(“team_name”).iloc[:16].index):
    team_name = home_away_2000.loc[row, "team_name"]
    attendance = home_away_2000.loc[row, ["home", "away"]]
    ax[idx//4, idx%4].pie(attendance)
    ax[idx//4, idx%4].set_title(team_name)
fig.legend(["Home", "Away"])
fig.suptitle("Home vs Away Attendance 2000")
```



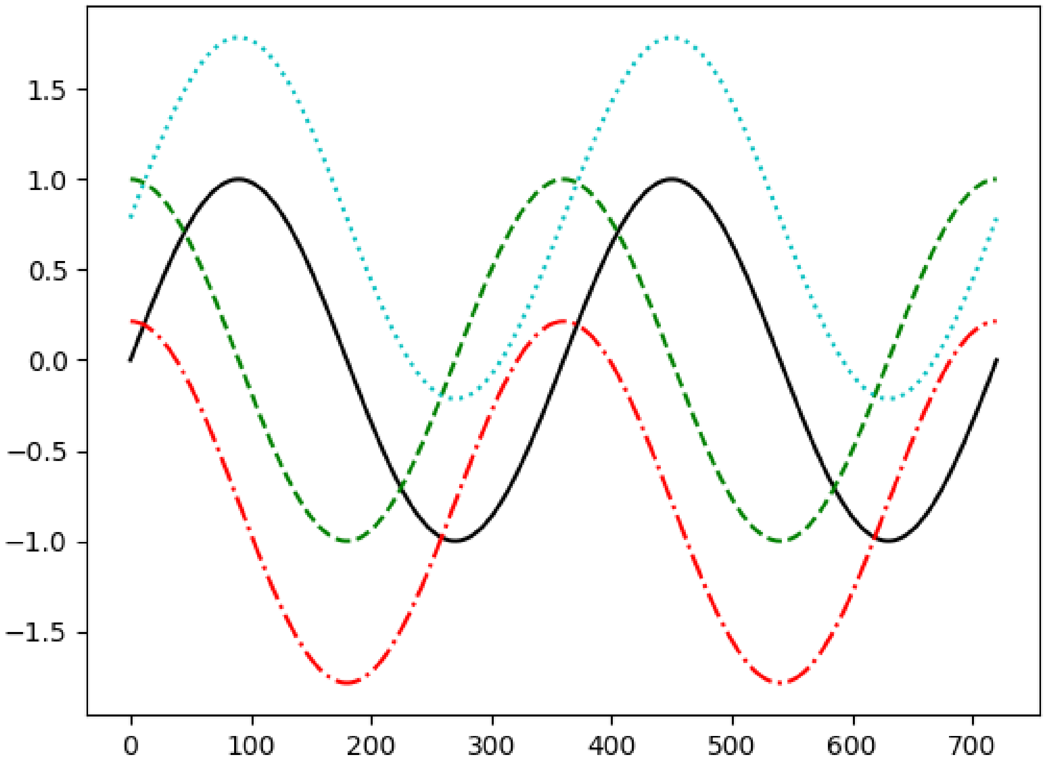
```
fig, ax = plt.subplots(4, 4, figsize=(10, 10))
for idx, row in enumerate(home_away_2000.sort_values(“home”, ascending=False).iloc[:16].index):
    team_name = home_away_2000.loc[row, "team_name"]
    attendance = home_away_2000.loc[row, ["home", "away"]]
    ax[idx//4, idx%4].pie(attendance)
    ax[idx//4, idx%4].set_title(team_name)
fig.legend(["Home", "Away"])
fig.suptitle("Home vs Away Attendance 2000")
```



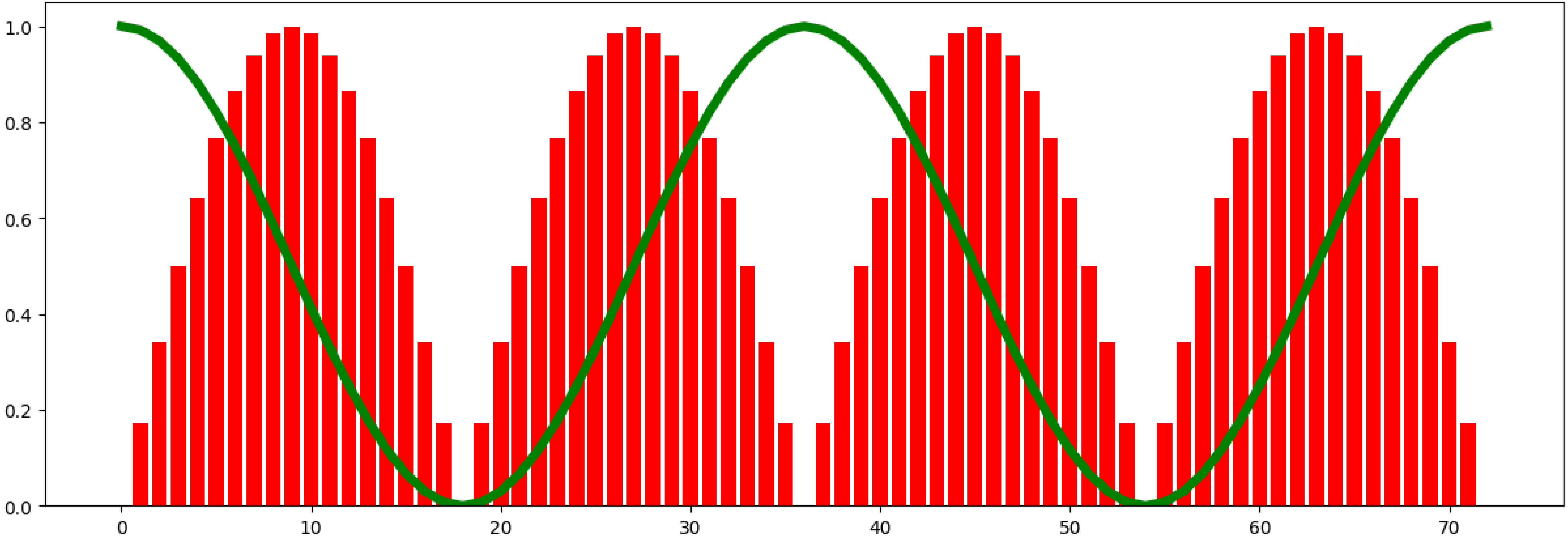
```
fig, ax = plt.subplots(4, 2, figsize=(9, 14))
for idx, team_name in enumerate(["Bears", "Cowboys", "Texans", "Titans"]):
    year = 2019
    data = attendance_df[(attendance_df["year"] == year)
        & (attendance_df["team_name"] == team_name)]
    home = data["home"].iloc[0]
    away = data["away"].iloc[0]
    ax[idx, 0].pie([home, away])
    ax[idx, 0].set_title(f"Home vs Away Attendance for {team_name}")
    ax[idx, 1].bar(np.arange(len(data)), data["weekly_attendance"])
    ax[idx, 1].set_xticks(np.arange(len(data)), data["week"])
    ax[idx, 1].set_title(f"Weekly Attendance for {team_name}")
fig.suptitle("Attendance 2019")
```



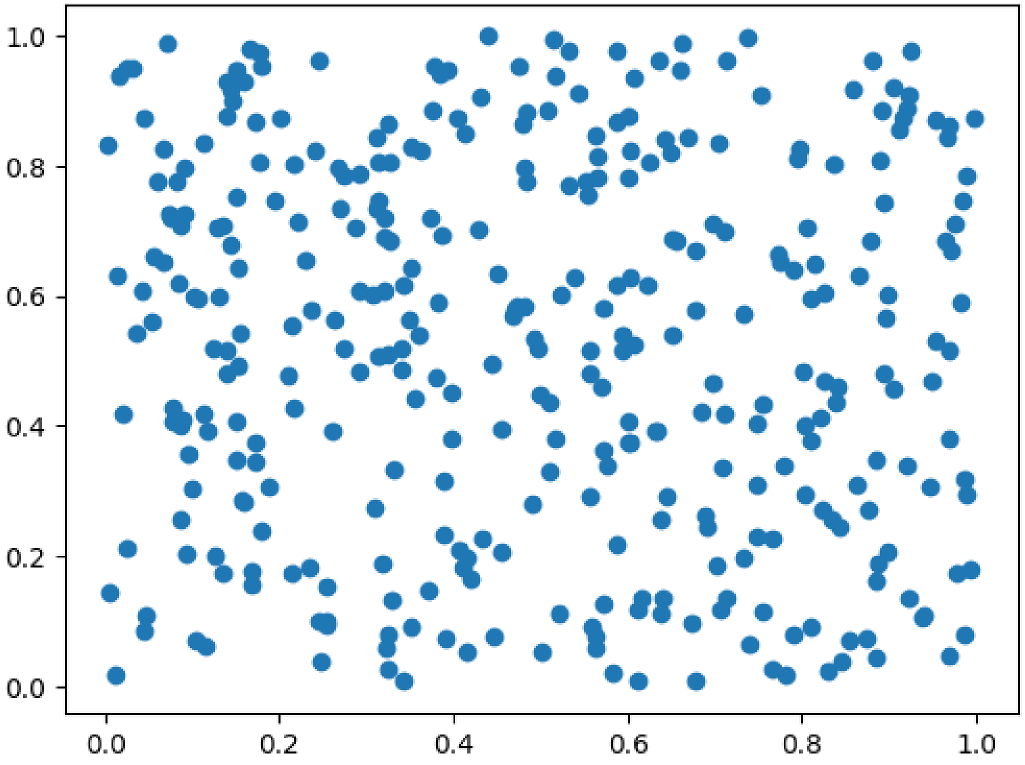
```
x = np.linspace(0, np.pi*4, 721)
plt.plot(np.sin(x), color="k")
plt.plot(np.cos(x), linestyle="--", color="g")
plt.plot(np.sin(x) + (np.pi / 4), linestyle=":", color="c")
plt.plot(np.cos(x) - (np.pi / 4), linestyle="-.", color="r")
```



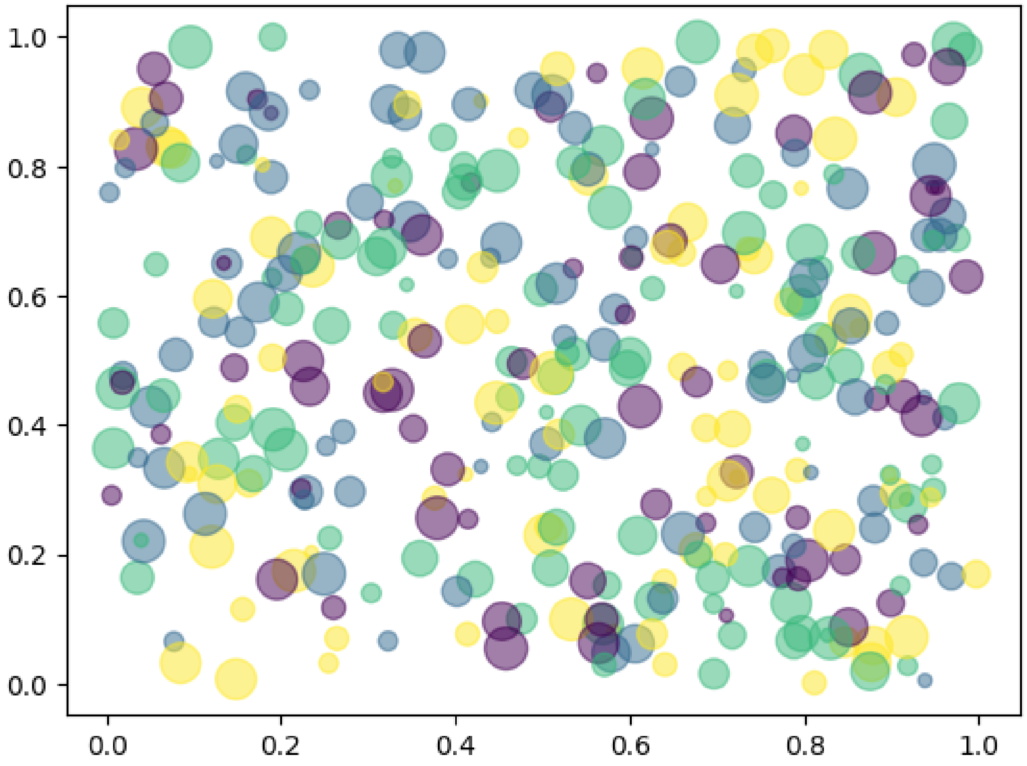
```
x = np.linspace(0, np.pi*4, 73)
plt.figure(figsize=(15,5))
plt.bar(np.arange(len(x)), np.abs(np.sin(x)), color="r")
plt.plot(np.arange(len(x)), np.cos(x) / 2 + 0.5, color="g", linewidth=5)
```



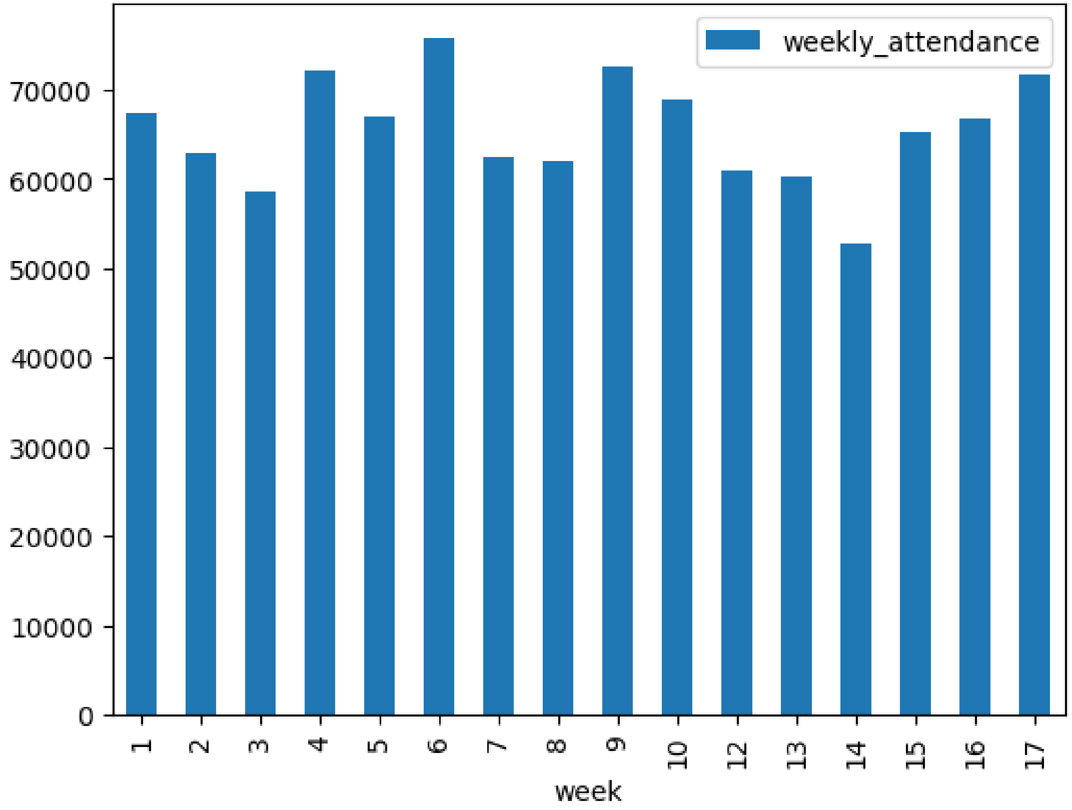
```
x = np.random.uniform(size=350)
y = np.random.uniform(size=350)
```



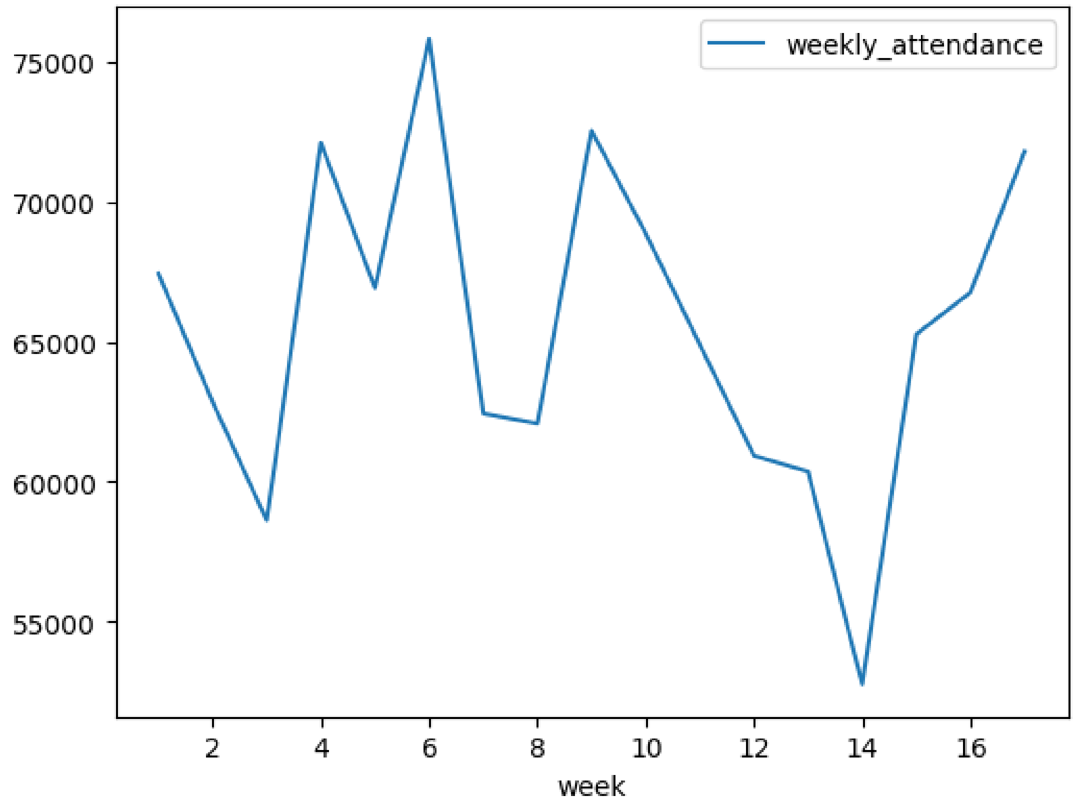
```
x = np.random.uniform(size=350)
y = np.random.uniform(size=350)
c = np.random.choice(np.arange(4), size=350)
s = np.random.choice(np.arange(1, 11) * 25, size=350)
plt.scatter(x, y, c=c, s=s, alpha=0.5)
```



```
titans_attendance_2019[titans_attendance_2019["weekly_attendance"].notna()]
    .plot(x="week", y="weekly_attendance", kind="bar")
```



```
titans_attendance_2019[titans_attendance_2019["weekly_attendance"].notna()]
    .plot(x="week", y="weekly_attendance", kind="line")
```



Jupyter Notebook

DEMO

Three Ideas to Remember

Idea	Why	Package
Data is worthless	Big glob of numbers	numpy
Analytics are worth pennies	Discover the story of the data	pandas
Decisions are worth dollars	Tell the story of the data	matplotlib

THANK YOU!

https://linktr.ee/douglasstarnes