1

June 11, 2024

1 Activity: Variables in Python

1.1 Introduction

In this activity you will practice the following: - Assigning specific values / types to variables - Overwriting variables

Question 1 Create variable `name` and assign any value you want to it.

[1]:
```
name = 'Ashwin'
print (name)
```

```
Ashwin
```

[2]:
```
# Question 1 Grading Checks

print('Make sure you created the variable: name')
print(name)
```

```
Make sure you created the variable: name
Ashwin
```

Question 2 Assign the value of `2.7` to the variable `seconds`.

[4]:
```
seconds = 2.7
print(seconds)
```

```
2.7
```

[5]:
```
# Question 2 Grading Checks

assert isinstance(seconds, float), 'Have you assigned a float value to the␣
↪variable seconds?'
```

Question 3 Assign boolean value `True` to the variable `python_is_fun`.

[8]:
```
python_is_fun = True
print(python_is_fun)
```

```
True
```

[9]:
```
# Question 3 Grading Checks

assert isinstance(python_is_fun, bool), 'Have you assigned a boolean value to␣
 ↪the python_is_fun variable?'
```

2

June 11, 2024

1 Activity: Using Lists & Tuples

1.1 Introduction

In this activity you will practice using and creating lists and tuples. This activity contains the following topics: - Creating lists - Creating tuples - Indexing - Slicing - List methods - Unpacking

Question 1 Create a list called `countries` with the names of 5 different countries of your choice.

[64]:
```
countries = ['USA','India','France','Germany','Russia']
```

[65]:
```
# Question 1 Grading Checks

assert isinstance(countries, list), 'Your variable is not a list'
```

Question 2 Using the `countries` list, assign the first and last countries in the list to the variables `first_country` and `last_country` respectively.

[66]:
```
first_country = countries[0]
last_country = countries[4]
```

[67]:
```
# Question 2 Grading Checks

assert isinstance(first_country, str), 'Your variable is not a string'
assert isinstance(last_country, str), 'Your variable is not a string'
```

Question 3 Using the `countries` list, add a country to the end of the list and then assign the last 3 countries to a variable called `last_three_countries`.

[68]:
```
countries.append('Ireland')
last_three_countries=countries[-3:]
```

[69]:
```
# Question 3 Grading Checks

assert isinstance(last_three_countries, list), 'Your variable is not a list'
```

Question 4 You're given a tuple containing superheroes, unpack the tuple into variables `superman`, `batman`, and `black_panther`

[70]:
```
# Given tuple

superheroes = ('Superman', 'Batman', 'Black Panther')
```

[71]:
```
superman, batman, black_panther = superheroes
```

[72]:
```
# Question 6 Grading Checks

assert isinstance(superman, str), 'Your variable is not a string'
assert isinstance(batman, str), 'Your variable is not a string'
assert isinstance(black_panther, str), 'Your variable is not a string'
```

3

June 11, 2024

1 Activity: Using Dictionaries

1.1 Introduction

In this activity you will practice the following: - Creating dictionaries - Accessing values in a dictionary - Adding key-value pairs to a dictionary - Modifying values in a dictionary - Dictionary methods

Question 1 Create a dictionary called `student` which contains the following: - name: derrick - age: 17 - math_grade: 90

[14]:
```
student = {'name':'derrick','age':17,'math_grade':90}
```

[15]:
```
# Question 1 Grading Checks

assert isinstance(student, dict), 'Have you created a dictionary called student?
↪'
assert len(student) == 3, 'Have you added the correct number of key-value pairs␣
↪to the dictionary?'
```

Question 2 Access the value of the `math_grade` key in the `student` dictionary you created in Question 1 and assign it to a variable called `grade`.

[16]:
```
grade=student['math_grade']
```

[17]:
```
# Question 2 Grading Checks

assert isinstance(grade, int), 'Have you accessed the value of the math_grade␣
↪key in the student dictionary?'
```

Question 3 Add a new key-value pair to the `student` dictionary you created in Question 1. The new key should be `english_grade` and the value should be `85`.

[18]:
```
student['english_grade']=85
```

[19]:
```
# Question 3 Grading Checks

assert len(student) == 4, 'Have you added the correct number of key-value pairs␣
↪to the dictionary?'
```

Question 4 Using the given dictionary `top_student`, store all the values in the dictionary in a variable called `top_student_grades`.

[20]:
```
top_student = {
    "math_grade": 95,
    "english_grade": 98,
    "history_grade": 90,
    "science_grade": 93,
    "art_grade": 92,
    "music_grade": 96
}
```

[21]:
```
top_student_grades=top_student.values()
```

[22]:
```
# Question 5 Grading Checks
```

4

June 11, 2024

1 Activity: Using Booleans

1.1 Introduction

In this activ|ity, you will use booleans to complete the following questions. This activity includes: - Logical operators - Boolean expressions - Membership operators

Question 1 Check if the variable age is greater than or equal to 18 and less than or equal to 25. Assign the result to a variable called is_age_valid.

If the age is valid, negate the variable `can_drive` and assign the result to a new variable called `cannot_drive`.

[150]:
```
can_drive = False
age = 20
```

[151]:
```
cannot_drive = True
is_age_valid = 18 <= age <= 25

if is_age_valid:
    cannot_drive = not can_drive

print(can_drive)
print(cannot_drive)
```

```
False
True
```

[152]:
```
# Question 1 Grading Checks

assert isinstance(cannot_drive, bool), "Make sure you are assigning a boolean␣
 ↪value to cannot_drive"
assert isinstance(is_age_valid, bool), "Make sure you are assigning a boolean␣
 ↪value to is_age_valid"
```

Question 2 Check if the variable `name` is in the list `names`. Assign the result to a variable called `is_name_valid`.

[153]:
```
name = "John"
names = ["John", "Jane", "Jack"]
```

[154]:
```
is_name_valid = name in names
```

[155]:
```
# Question 2 Grading Checks

assert isinstance(is_name_valid, bool), "Make sure you are assigning a boolean␣
↪value to is_name_valid"
```

Question 3 Write a boolean expression that checks if the variable `whole_num` is not equal to `float_num`. Assign the result to a variable called `is_not_equal`.

[156]:
```
whole_num = 5
float_num = 5.0
```

[157]:
```
is_not_equal = (whole_num!=float_num)
```

[158]:
```
# Question 3 Grading Checks

assert isinstance(is_not_equal, bool), "Make sure you are assigning a boolean␣
↪value to is_not_equal"
```

5

June 11, 2024

1 Activity: Using Conditionals

1.1 Introduction

In this activity, you will use conditionals to solve the following questions. This activity contains: - `if` statements - `if-else` statements - `if-elif-else` statements - Nested `if` statements - `if` statements with logical operators

Question 1 Using the below modulo operator(%, which returns the remainder after division), write an `if-else` statement that assigns the value `True` to the variable `is_even` if `number` is even or `False` if `number` is odd.

[21]:
```
number = 24
is_even = False

#Modulo Operator
number % 2 == 0
```

[21]:
```
True
```

[22]:
```
if number % 2 == 0:
    is_even = True
else:
    is_even = False
```

[23]:
```
# Question 1 Grading Checks

assert type(is_even) == bool
```

Question 2 Write an `if-elif-else` statement that assigns the string `"positive"` to the variable `sign` if the `integer_number` is positive, `"negative"` if `integer_number` is negative, or `"zero"` if `integer_number` is 0.

[24]:
```
integer_number = 0
sign = ''
```

[25]:
```
if integer_number > 0:
    sign = "positive"
elif integer_number < 0:
    sign = "negative"
else:
    sign = "zero"
```

[26]:
```
# Question 2 Grading Checks

assert type(sign) == str
assert len(sign) > 0
```

Question 3 You are given a `dict` called `person` with a person's name and an age.

Write a nested `if` statement that first checks if the person is old enough to vote, and assigns the value `True` to the variable `can_vote` if they are older than 17.

Then, check if they are 21 or older and assigns the value `True` to the `can_rent_car` variable.

Finally, print the `can_vote` and `can_rent_car` variables.

[27]:
```
person = {
    "name": "James Dean",
    "age": 19,
}
can_vote = False
can_rent_car = False
```

[28]:
```
if person["age"] > 17:
    can_vote = True
    if person["age"] > 21:
        can_rent_car = True
        print(can_vote)
        print(can_rent_car)
```

[29]:
```
# Question 3 Grading Checks

assert type(can_vote) == bool
assert type(can_rent_car) == bool
```

6

June 11, 2024

1 Activity: Using Pandas

1.1 Introduction

In this activity you will practice using some of the basic functionality associated with Pandas. This activity will cover the following topics: - Creating a `DataFrame` - Displaying `DataFrame` information - Accessing column data - Getting ranges of column data - Creating Series from dictionaries - Using the `loc()` and `iloc()` methods - Getting data from multiple columns - Getting rows

Question 1 Create a DataFrame called `df` from the given CSV file `student_data.csv` then using the `df` DataFrame, assign the `Name` column to a `Series` called `names`.

[52]:
```
import pandas as pd
df = pd.read_csv("student_data.csv")
names = df['Name']
display(df)
display(names)
```

```
                Name  Age  Math Grade  English Grade
0    Jennifer Jackson   14          84             81
1     Michael Johnson   14          92             85
2          Robert Lee   18          87             80
3        Linda Harris   13          90             77
4       Michael Moore   18          88             99
..                ...  ...         ...            ...
70    William Johnson   17          99             75
71        Linda Smith   18          92             83
72       Jennifer Lee   17          75             88
73     Michael Miller   12          86             84
74      Emily Johnson   13          94            100

[75 rows x 4 columns]


0     Jennifer Jackson
1      Michael Johnson
2           Robert Lee
3         Linda Harris
```

```
4        Michael Moore
            ...
70     William Johnson
71         Linda Smith
72        Jennifer Lee
73      Michael Miller
74       Emily Johnson
Name: Name, Length: 75, dtype: object
```

[53]:
```
# Question 1 Grading Checks

assert isinstance(df, pd.DataFrame), "Did you create a DataFrame called df?"
assert isinstance(names, pd.Series), "Did you assign the Name column in a␣
 ↪variable called names?"
```

Question 2 Using the `df` DataFrame, assign the `Age` and `Math Grade` columns to a DataFrame called `age_math`.

[54]:
```
age_math= df[['Age','Math Grade']]
display(age_math)
```

```
    Age  Math Grade
0    14          84
1    14          92
2    18          87
3    13          90
4    18          88
..  ...         ...
70   17          99
71   18          92
72   17          75
73   12          86
74   13          94

[75 rows x 2 columns]
```

[55]:
```
# Question 2 Grading Checks

assert isinstance(age_math, pd.DataFrame), "Did you assign the Age and Math␣
 ↪Grade columns to a variable called age_math?"
```

Question 3 Using the `.loc()` method, assign the `Age` and `Math Grade` columns for the first 30 rows of `df` to a variable called `first_thirty_loc`.

[56]:
```
first_thirty_loc=df.loc[:29,['Age','Math Grade']]
display(first_thirty_loc)
```

	Age	Math Grade
0	14	84
1	14	92
2	18	87
3	13	90
4	18	88
5	12	96
6	18	95
7	16	85
8	18	91
9	18	84
10	17	91
11	17	79
12	17	75
13	16	93
14	15	89
15	13	100
16	14	91
17	13	99
18	12	85
19	14	95
20	18	92
21	15	89
22	12	84
23	16	96
24	12	86
25	17	94
26	15	88
27	12	80
28	13	82
29	17	85

[57]:
```
# Question 3 Grading Checks

assert isinstance(first_thirty_loc, pd.DataFrame), "Did you correctly assign␣
↪the first 30 rows to a variable called first_thirty_loc?"
```

Question 4 Get the even numbered rows from the `Name` and `English Grade` columns and assign the result to a variable called `even_rows_english`.

[58]:
```
even_rows_english = df.loc[0::2,['Name','English Grade']]
display(even_rows_english)
```

	Name	English Grade
0	Jennifer Jackson	81
2	Robert Lee	80
4	Michael Moore	99
6	Sarah Smith	90
8	Linda Jackson	79
10	Jane Thomas	77
12	Jane Brown	89
14	Michael Wilson	86
16	Mary Jones	76
18	Jennifer Miller	85
20	Susan Wilson	97
22	Emily Moore	81
24	John Moore	100
26	Jane Miller	98
28	John Moore	87
30	Michael Lee	77
32	Jennifer Johnson	86
34	Susan Miller	88
36	William Jackson	93
38	Michael Davis	84
40	Sarah Harris	99
42	Christopher Moore	83
44	Christopher Lee	91
46	John Lee	81
48	Michael Davis	93
50	Jane White	92
52	William Wilson	90
54	David Moore	86
56	Jennifer Davis	95
58	Linda Davis	78
60	Daniel Jones	83
62	Christopher Miller	76
64	Emily Johnson	82
66	Karen Moore	92
68	Christopher Jackson	99
70	William Johnson	75
72	Jennifer Lee	88
74	Emily Johnson	100

[59]:
```
# Question 4 Grading Checks

assert isinstance(even_rows_english, pd.DataFrame), "Did you correctly assign␣
↪even numbered rows to a variable called even_rows_english? Hint: the first␣
↪row is index 0."
```

7

June 11, 2024

1 Activity: Selective Subsets

1.1 Introduction

In this activity you will practice selecting subsets of data from a DataFrame using Pandas. This activity will cover the following topics: - Creating masks - Negating masks - Masks with slicing - Null value masks

Question 1 Create a `DataFrame` called `df` from the given CSV file `movie_data.csv`, and then create a mask called `before_millennium` to select all movies that were released before 2000.

[47]:
```
import pandas as pd
df=pd.read_csv("movie_data.csv")
display(df)
before_millennium=df['Year Released'] < 2000
display(before_millennium)
moviesbefore2000 = df.loc[before_millennium,'Title']
moviesbefore2000
```

```
                      Title  Year Released  Rating  Box Office ($M)
0   The Shawshank Redemption           1994     9.3             58.3
1               The Godfather          1972     9.2            246.1
2            The Dark Knight           2008     9.0           1005.0
3               Pulp Fiction           1994     8.9            213.9
4           Schindler's List           1993     8.9            321.3
..                       ...            ...     ...              ...
70                 Andhadhun           2018     8.3             48.0
71                 Gully Boy           2019     8.1             62.0
72            Dil Chahta Hai           2001     8.1             13.0
73          Dil To Pagal Hai           1997     7.1             11.0
74              Om Shanti Om           2007     6.7             23.0

[75 rows x 4 columns]


0      True
1      True
2     False
```

```
3      True
4      True
      ...
70    False
71    False
72    False
73     True
74    False
Name: Year Released, Length: 75, dtype: bool
```

[47]:
```
0        The Shawshank Redemption
1                   The Godfather
3                    Pulp Fiction
4                Schindler's List
5                     Fight Club
6                    Forrest Gump
8                      The Matrix
10       The Silence of the Lambs
12                The Green Mile
13         The Godfather: Part II
16             The Usual Suspects
17            Saving Private Ryan
18                          Se7en
21                        Titanic
31        The Godfather: Part III
32                The Big Lebowski
41                   Jurassic Park
45                   The Lion King
60    Dilwale Dulhania Le Jayenge
61             Kuch Kuch Hota Hai
64                        Baazigar
68                          Sholay
69                Mera Naam Joker
73                Dil To Pagal Hai
Name: Title, dtype: object
```

[48]:
```
# Question 1 Grading Checks

assert isinstance(df, pd.DataFrame), 'Did you create a DataFrame called df?'
```

Question 2 Using the `before_millennium` mask from Question 1, assign the titles of every movie that was released after 2000 to a `Series` called `newer_titles`.

[49]:
```
newer_titles= df[~before_millennium]['Title']
```

[50]:
```
# Question 2 Grading Checks

assert isinstance(newer_titles, pd.Series), 'Did you create a Series called␣
↪newer_titles?'
```

Question 3 Create a mask to select movies with a `Rating` of `8.9` and a `Box Office ($M)` value higher than `1000.0`. Assign the resulting `Series` to a variable called `popular_pg_movies`.

[51]:
```
popular_pg_movies = (df['Rating'] == 8.9) & (df['Box Office ($M)'] > 1000)
```

[52]:
```
# Question 3 Grading Checks

assert isinstance(popular_pg_movies, pd.Series), 'Did you create a Series␣
↪called popular_pg_movies?'
```

Question 4 Create a mask to select movies with a null value for `Box Office ($M)` or `Rating`. Assign the resulting `Series` to a variable called `missing_info`.

[53]:
```
missing_info = (df['Box Office ($M)'].isnull() ) | ( df['Rating'].isnull() )
```

[54]:
```
# Question 4 Grading Checks

assert isinstance(missing_info, pd.Series), 'Did you create a Series called␣
↪missing_info?'
```

8

June 11, 2024

1 Activity: Removing Data

1.1 Introduction

In this activity you will practice using Pandas functionality to check for and remove any unwanted data from a dataset. This activity will cover the following topics: - Removing columns from a DataFrame - Removing rows from a DataFrame - Removing rows based on a condition - Checking for duplicate data

Question 1 Create a `DataFrame` called `df` from the given CSV file `exotic_plants_data.csv`, then drop the column `Type` and assign the result to a new `DataFrame` called `df_no_type`.

[26]:
```
import pandas as pd
df=pd.read_csv("exotic_plants_data.csv")
display(df)
df_no_type= df.drop(columns='Type')
display(df_no_type)
```

	Plant Name	Type	Origin	Height (cm)
0	Orchid	Ornamental	Tropical	30
1	Fern	Ground Cover	Tropical	40
2	Bamboo	Grass	Asia	600
3	Cactus	Succulent	America	60
4	Bird of Paradise	Ornamental	Africa	150
..	...	...	...	...
71	Ficus	Tree	Asia	200
72	Columbine	Flower	North America	30
73	Jasmine	Shrub	Asia	90
74	Fuchsia	Flower	Central and South America	40
75	Amaranth	Flower	Various	80

[76 rows x 4 columns]

	Plant Name	Origin	Height (cm)
0	Orchid	Tropical	30
1	Fern	Tropical	40
2	Bamboo	Asia	600

```
3             Cactus                     America          60
4   Bird of Paradise                      Africa         150
..               ...                         ...         ...
71             Ficus                       Asia          200
72         Columbine              North America          30
73           Jasmine                       Asia          90
74           Fuchsia  Central and South America          40
75          Amaranth                     Various          80

[76 rows x 3 columns]
```

[27]:
```
# Question 1 Grading Checks

assert isinstance(df, pd.DataFrame), 'Have you created a DataFrame named df?'
assert isinstance(df_no_type, pd.DataFrame), 'Have you created a DataFrame␣
↪named df_no_type?'
```

Question 2 Check the `df DataFrame` for any duplicate rows and assign the result to a new `DataFrame` called `df_duplicates`.

[28]:
```
df_duplicates=df[df.duplicated()]
df.duplicated()
```

[28]:
```
0     False
1     False
2     False
3     False
4     False
      …
71    False
72    False
73    False
74    False
75    False
Length: 76, dtype: bool
```

[29]:
```
# Question 2 Grading Checks

assert isinstance(df_duplicates, pd.DataFrame), 'Have you created a DataFrame␣
↪named df_duplicates?'
```

Question 3 Check the `df DataFrame` for any duplicate rows based on the `Plant Name` and `Type` columns and assign the result to a new `DataFrame` called `df_plant_type_duplicates`.

[30]:
```
df_plant_type_duplicates = df[df.duplicated(subset=['Plant Name','Type'])]
display(df_plant_type_duplicates)
```

	Plant Name	Type	Origin	Height (cm)
6	Cactus	Succulent	America	60
22	Bamboo	Grass	Asia	500
30	Rafflesia	Flower	Southeast Asia	20
47	Kangaroo Paw	Flower	Australia	60
48	Bougainvillea	Shrub	South America	400
49	Bird of Paradise	Ornamental	Africa	150
50	Venus Flytrap	Carnivorous	North America	15
51	Rose	Flower	Asia	60
53	Tulip	Flower	Europe	30
55	Sunflower	Flower	North America	180
60	Cactus	Succulent	Americas	30
62	Bamboo	Grass	Asia	900
67	Aloe Vera	Succulent	Africa	30
73	Jasmine	Shrub	Asia	90

[31]:
```
# Question 3 Grading Checks

assert isinstance(df_plant_type_duplicates, pd.DataFrame), 'Have you created a␣
↪DataFrame named df_duplicates?'
```

Question 4 Create a mask called `clean_mask` that will clean up any duplicates in the `df` DataFrame that have the same `Plant Name` and `Origin` and only keep the most up-to-date duplicate entry.

[32]:
```
clean_mask = ~df.duplicated(subset=['Plant Name','Origin'],keep='last' )
display(clean_mask)
```

```
0      True
1      True
2     False
3     False
4     False
      ...
71     True
72     True
73     True
74     True
75     True
Length: 76, dtype: bool
```

[33]:
```
# Question 4 Grading Checks

assert isinstance(clean_mask, pd.Series), 'Have you created a Series named␣
↪clean_mask?'
```

9

June 11, 2024

1 Activity: Modifying & Replacing Values

1.1 Introduction

In this activity you will practice modifying and replacing values in a DataFrame using the various method that Pandas has to offer. This activity will cover the following, not necessarily in this order: - Checking for anomalous values - Using `.isnumeric()` - Using `min()` and `max()` methods - Using `.loc[]` to replace values - Using `isnull()` and `notnull()` methods

Question 1 Create a `DataFrame` called `df` from the given CSV file `employee_data.csv`, and then create a mask called `valid_names` that checks the `Name` column for any non-numeric values.

[111]:
```
import pandas as pd
df = pd.read_csv("employee_data.csv")
valid_names= df.Name.str.isnumeric()
display(df)
df.info()
```

```
                Name  Years of Employment  Weeks of Vacation  Position
0    Jennifer Jackson                     9                4.0  Engineer
1     Michael Johnson                     9                6.0   Analyst
2          Robert Lee                    13                3.0  Engineer
3         Linda Jones                     3                6.0   Manager
4        Karen Thomas                    14                2.0    Intern
..                ...                   ...                ...       ...
78                  1                     0               49.0   Unknown
79                  1                     0               47.0   Unknown
80                  1                    -5               46.0   Unknown
81                  1                    -4               52.0   Unknown
82                  1                     0               48.0   Unknown

[83 rows x 4 columns]


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83 entries, 0 to 82
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   Name                  83 non-null     object
 1   Years of Employment   83 non-null     int64
 2   Weeks of Vacation     80 non-null     float64
 3   Position              83 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 2.7+ KB
```

[112]:
```
# Question 1 Grading Checks

assert isinstance(df, pd.DataFrame), 'Have you created a DataFrame named df?'
assert isinstance(valid_names, pd.Series), 'Have you created a Series named␣
↪valid_names?'
```

Question 2 Using the original `DataFrame` `df`, create a mask called `unknown_position` that checks the `Position` column for any values that are equal to the string `Unknown`. Then, replace all such values with `Engineer`.

[113]:
```
unknown_position= df.Position == 'Unknown'
df.loc[unknown_position,'Position']= 'Engineer'
df
```

[113]:

	Name	Years of Employment	Weeks of Vacation	Position
0	Jennifer Jackson	9	4.0	Engineer
1	Michael Johnson	9	6.0	Analyst
2	Robert Lee	13	3.0	Engineer
3	Linda Jones	3	6.0	Manager
4	Karen Thomas	14	2.0	Intern
..	…	…	…	…
78	1	0	49.0	Engineer
79	1	0	47.0	Engineer
80	1	-5	46.0	Engineer
81	1	-4	52.0	Engineer
82	1	0	48.0	Engineer

[83 rows x 4 columns]

[114]:
```
# Question 2 Grading Checks

assert isinstance(unknown_position, pd.Series), 'Have you created a Series␣
↪named unknown_position?'
```

Question 3 Using the original `DataFrame` `df`, create a mask called `invalid_vacation` that checks the `Weeks of Vacation` column for any values that are null or missing. Then, use that mask to assign the value 0 to them.

[115]:
```
invalid_vacation = df['Weeks of Vacation'].isnull()
df.loc[invalid_vacation,'Weeks of Vacation'] = 0
```

[116]:
```
# Question 3 Grading Checks

assert isinstance(invalid_vacation, pd.Series), 'Have you created a Series␣
↪named invalid_vacation?'
```

Question 4 Using the original `DataFrame` `df`, find the maximum value in the `Weeks of Vacation` column and assign it to the variable `max_vac_before`. Then, replace all values in the `Weeks of Vacation` column that are greater than 6 with 6.

[117]:
```
max_vac_before = df ['Weeks of Vacation'].max()
df.loc[df['Weeks of Vacation'] > 6, 'Weeks of Vacation'] = 6
```

[118]:
```
# Question 4 Grading Checks

assert isinstance(df, pd.DataFrame), 'Have you created a DataFrame named df?'
```

10

June 11, 2024

1 Activity: Basic Exploration

1.1 Introduction

In this activity you will practice using exploration methods on a data set containing games of online chess*. This activity includes some or all of the following, not necessarily in this order: - Viewing the data - Finding the mean - Finding the median - Standard deviation - Aggregations - Grouping

*The data set is from Kaggle.

1.2 Note

This data set is larger than those used in previous activities. Please run the cell below which uses the `info()` method to get a sense of the data before you begin.

[1]:
```
import pandas as pd

df = pd.read_csv('chess_games.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20058 entries, 0 to 20057
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   game_id            20058 non-null  int64
 1   rated              20058 non-null  bool
 2   turns              20058 non-null  int64
 3   victory_status     20058 non-null  object
 4   winner             20058 non-null  object
 5   time_increment     20058 non-null  object
 6   white_id           20058 non-null  object
 7   white_rating       20058 non-null  int64
 8   black_id           20058 non-null  object
 9   black_rating       20058 non-null  int64
 10  moves              20058 non-null  object
 11  opening_code       20058 non-null  object
 12  opening_moves      20058 non-null  int64
```

```
 13  opening_fullname   20058 non-null  object
 14  opening_shortname  20058 non-null  object
 15  opening_response   1207 non-null   object
 16  opening_variation  14398 non-null  object
dtypes: bool(1), int64(5), object(11)
memory usage: 2.5+ MB
```

Question 1 Create two `DataFrame` objects called `first_three` and `last_three` and assign the first and last three rows of the data set to them, respectively.

[2]:
```
first_three = df.head(3)
last_three =df.tail(3)
first_three
```

[2]:

	game_id	rated	turns	victory_status	winner	time_increment	white_id	white_rating	black_id	black_rating	moves	opening_code	opening_moves	opening_fullname	opening_shortname	opening_response	opening_variation
0	1	False	13	Out of Time	White	15+2	bourgris	1500	a-00	1191	d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3 Ba5…	D10	5	Slav Defense: Exchange Variation	Slav Defense	NaN	Exchange Variation
1	2	True	16	Resign	Black	5+10	a-00	1322	skinnerua	1261	d4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe5 Nxe5 Qd4 Nc6…	B00	4	Nimzowitsch Defense: Kennedy Variation	Nimzowitsch Defense	NaN	Kennedy Variation
2	3	True	61	Mate	White	5+10	ischia	1496	a-00	1500	e4 e5 d3 d6 Be3 c6 Be2 b5 Nd2 a5 a4 c5 axb5 Nc…	C20	3	King's Pawn Game: Leonardis Variation	King's Pawn Game	NaN	Leonardis Variation

[3]:
```
# Question 1 Grading Checks

assert first_three.shape == (3, 17), 'Make sure that you chose only the first␣
↪three rows.'
assert last_three.shape == (3, 17), 'Make sure that you chose only the last␣
↪three rows.'
```

Question 2 Create two new `DataFrame` objects called `white_lower_rating` and `white_higher_rating` that are assigned the rows of data where the white player's rating is less than 1200 and greater than or equal to 1800, respectively.

[4]:
```
white_lower_rating = df[df['white_rating'] < 1200]
white_higher_rating = df[df['white_rating'] >= 1800]
```

[5]:
```
# Question 2 Grading Checks

assert isinstance(white_lower_rating, pd.DataFrame), 'Make sure that you are␣
↪creating a DataFrame object called white_lower_rating.'
assert isinstance(white_higher_rating, pd.DataFrame), 'Make sure that you are␣
↪creating a DataFrame object called white_higher_rating.'
```

Question 3 Using the `black_rating` column, create a `DataFrame` object called `top_10_percent_black` which is assigned the top 10% of black players by rating. That is, the only rows of where the `black_rating` is higher than 90% of all the `black_rating` values.

[6]:
```
top_10_percent_black = df[df['black_rating'] >= df['black_rating'].quantile(0.
↪9)]
```

[7]:
```
# Question 3 Grading Checks

assert top_10_percent_black.shape == (2011, 17), 'Make sure that you are␣
↪selecting the top 10% of black players by rating. Hint: Try using a␣
↪conditional statement to check which black_rating values are in the top 10%.'
```

11

June 11, 2024

1 Activity: Creating Visualizations

1.1 Introduction

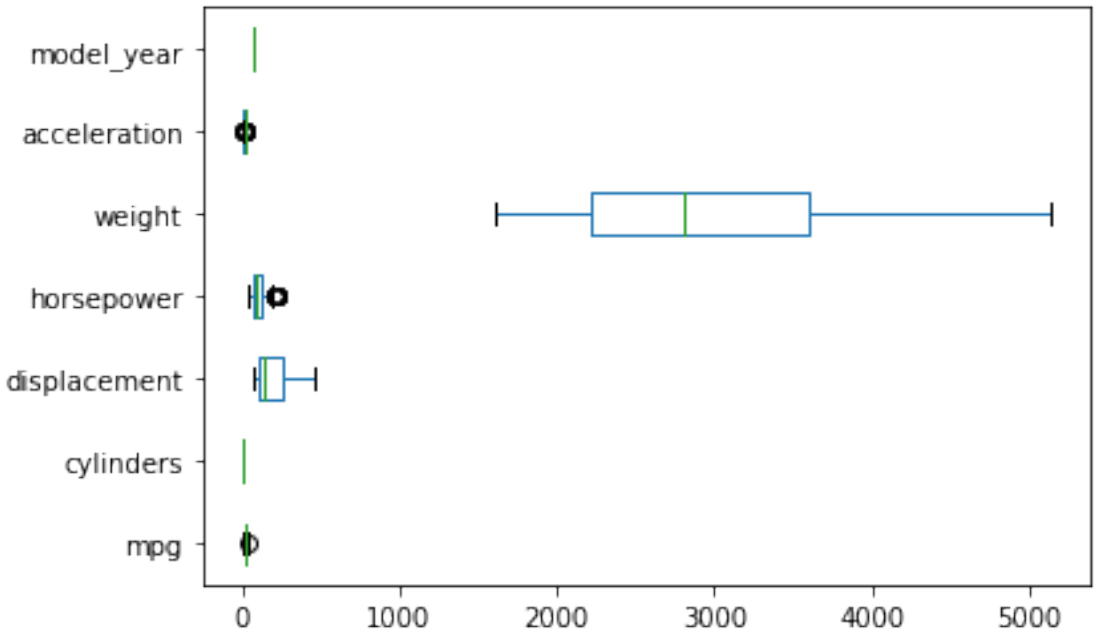
In this activity you will practice using Pandas functionality to create visualizations.

This activity will cover the following topics: - Creating individual and multiple box plots in one visualization. - Creating individual and multiple histograms in one visualization. - Creating scatter plots.

[28]:
```
import pandas as pd

# Data from https://github.com/mwaskom/seaborn-data/blob/
 ↪2b29313169bf8dfa77d8dc930f7bd3eba559a906/mpg.csv
df = pd.read_csv('mpg.csv')
```

Question 1 Create a box plot from `df` using Pandas. Assign the number of box plots that are created in the visualization to the variable `n_box_plots`.

[29]:
```
df.plot.box(vert = False)
n_box_plots = 7
```

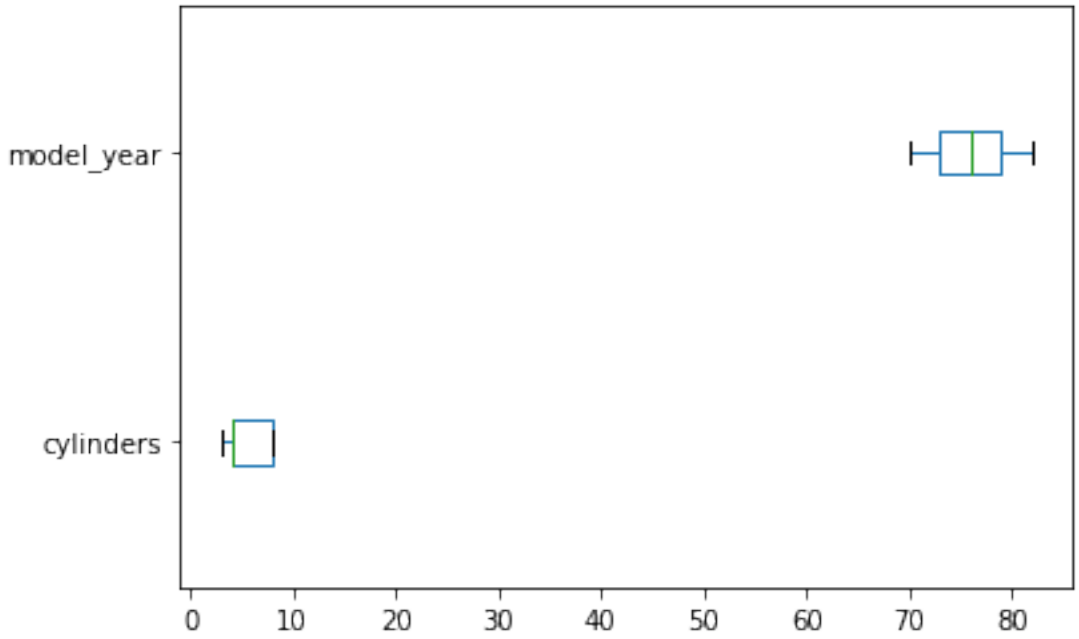


[30]:
```
# Question 1 Grading Checks

assert isinstance(n_box_plots, int), 'Did you assign a number (integer) to␣
↪n_box_plots representing the number of box plots that were created?'
```

Question 2 Using Pandas, create a visualization with box plots for the columns `cylinders` & `model_year`. What is the smallest and largest number label on the axis (representing values)?

Assign a best approximation of the smallest number label that is visible on the axis to the variable `smallest_label` and the largest number label to the variable `largest_label`.

[31]:
```
df[['cylinders','model_year']].plot.box(vert = False)
smallest_label = 0
largest_label = 80
```

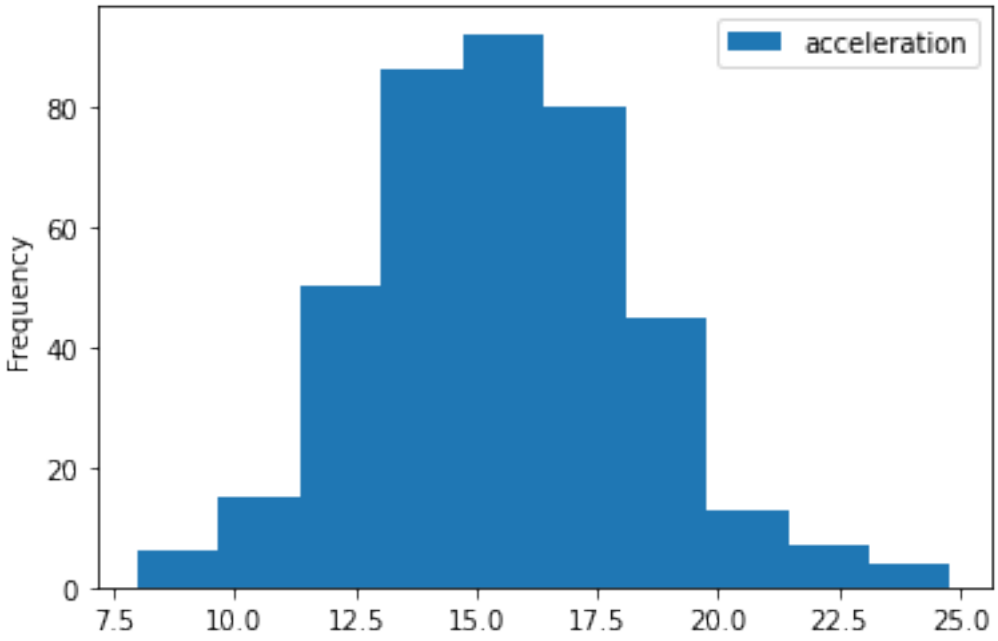


[32]:
```
# Question 2 Grading Checks

assert isinstance(smallest_label, (int, float)), 'Did you assign smallest_label␣
↪a number value?'
assert isinstance(largest_label, (int, float)), 'Did you assign largest_label a␣
↪number value?'
```

Question 3 Using Pandas, create a visualization of a histogram for the `acceleration` column. What is the approximate number for the peak of this histogram?

Assign the approximate number for the peak to the variable `approx_peak_height`.

[33]:
```
df[['acceleration']].plot.hist()
approx_peak_height = 90
```

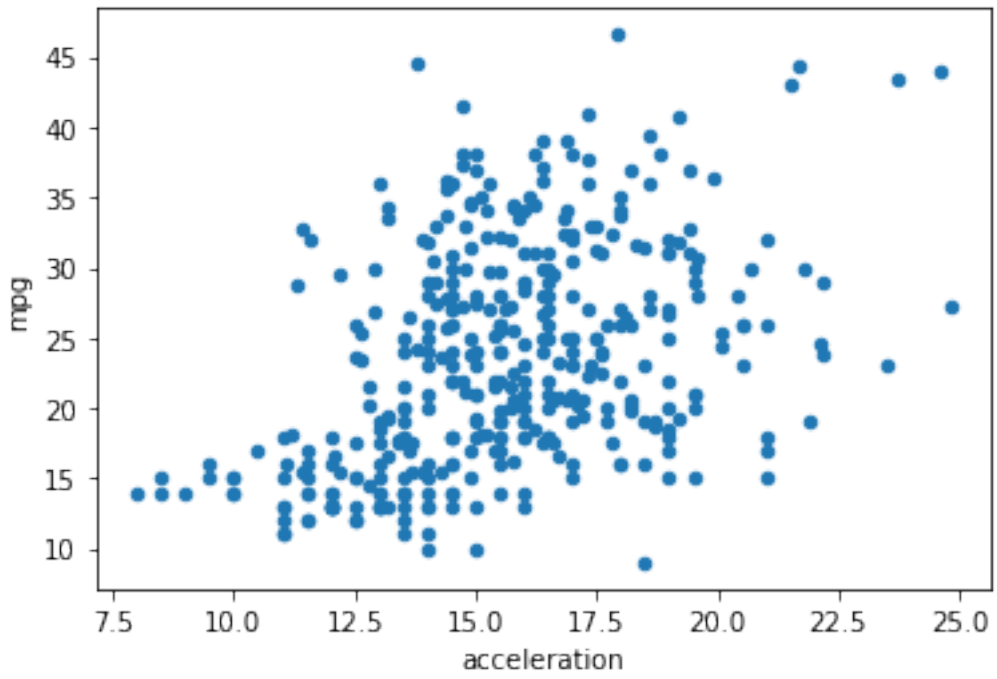


[34]:
```
# Question 3 Grading Checks


assert isinstance(approx_peak_height, (int, float)), 'Did you assign␣
↪approx_peak_height a number value?'
```

Question 4 Using Pandas, create a scatter plot of `acceleration` vs `mpg` (`acceleration` on the x-axis and `mpg` y-axis). What is the largest number label on the y-axis (representing `mpg`)?

Assign a best approximate for the largest number label on the `mpg` axis that is visible on the axis to the variable `largest_mpg_label`.

[35]:
```
df.plot.scatter(x='acceleration',y='mpg')
largest_mpg_label = 47
```



[36]:
```
# Question 4 Grading Checks

assert isinstance(largest_mpg_label, (int, float)), 'Did you assign␣
↪largest_mpg_label a number value?'
```

12

June 11, 2024

1 Activity: Exploring with Visualizations

1.1 Introduction

In this activity you will practice using Pandas functionality to create and explore visualizations.

This activity will cover the following topics: - Compare single values against one another. - Compare multiple values against one another. - Use different methods to change how you see the data.

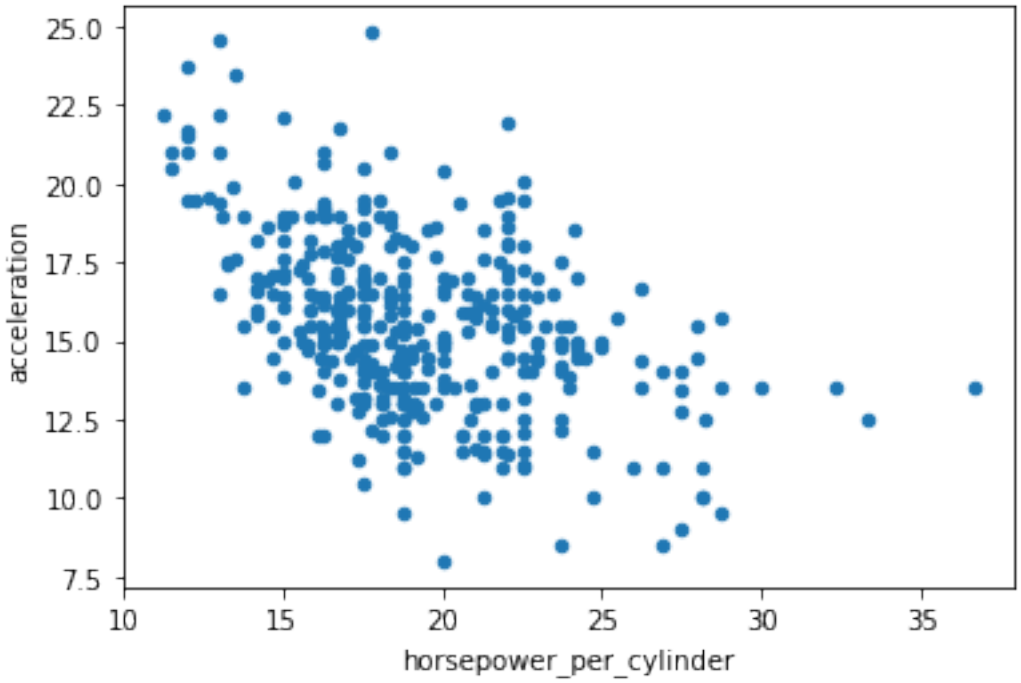
[38]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Data from https://github.com/mwaskom/seaborn-data/blob/
 ↪2b29313169bf8dfa77d8dc930f7bd3eba559a906/mpg.csv
df = pd.read_csv('mpg.csv')
```

Question 1 Create a new column in the DataFrame `df` called `horsepower_per_cylinder` that gives the value of `horsepower` per `cylinder`.

Then, create a scatter plot of `horsepower_per_cylinder` vs `acceleration` (`horsepower_per_cylinder` on the x-axis and `acceleration` on the y-axis). Does `acceleration` tend to *increase* or *decrease* as `horsepower_per_cylinder` *increases*?

Assign the boolean value `True` to the variable `acc_decreases` if `acceleration` decreases as `horsepower_per_cylinder` increases. Otherswise, assign the boolean value `False` to the variable `acc_decreases`.

[39]:
```
df['horsepower_per_cylinder'] = df['horsepower'] / df['cylinders']
df.plot.scatter(x='horsepower_per_cylinder',y='acceleration')
acc_decreases = True
```



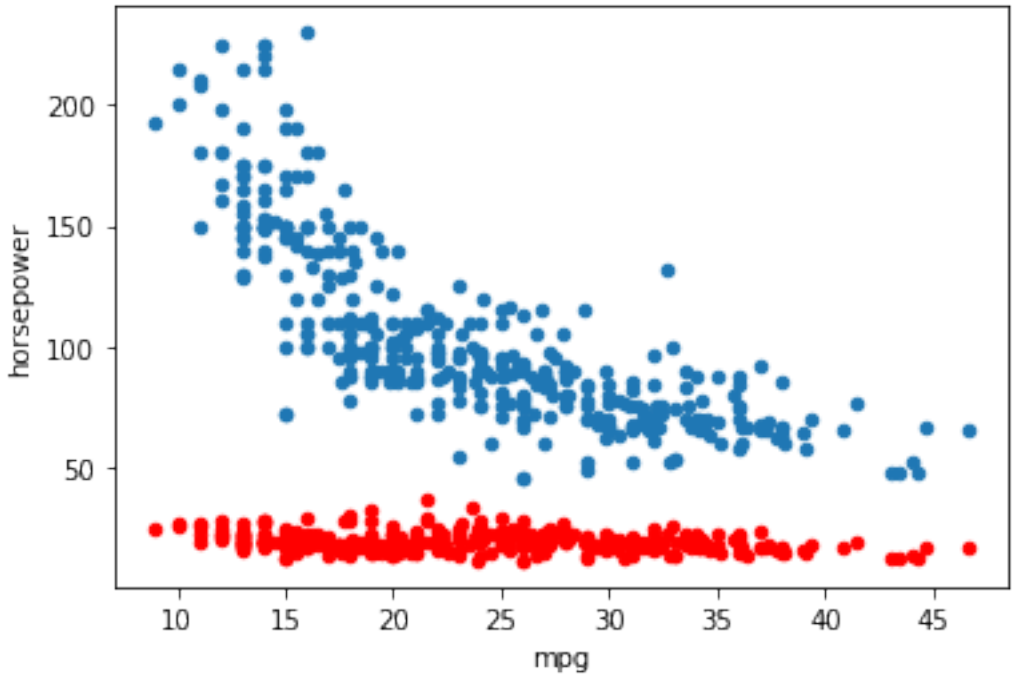
[40]:
```
# Question 1 Grading Checks

assert 'horsepower_per_cylinder' in df.columns, 'Did create a column called␣
↪`horsepower_per_cylinder` in the DataFrame?'
assert isinstance(acc_decreases, bool), 'Did you assign the either True or␣
↪False to acc_decreases?'
```

Question 2 Create a single visualization where `horsepower_per_cylinder` and `horsepower` are on the y-axis vs `mpg` on the x-axis in a scatter plot. Make each set of points a different color.

Set the result of the plot to the variable `ax`. Your code will look something like:

```
ax = # code to create a scatter plot
# ... other code
```

[41]:
```
ax = df.plot.scatter(x='mpg',y='horsepower_per_cylinder',color='red')
ax = df.plot.scatter(x='mpg',y='horsepower',ax=ax)
```



[42]:
```
# Question 2 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```

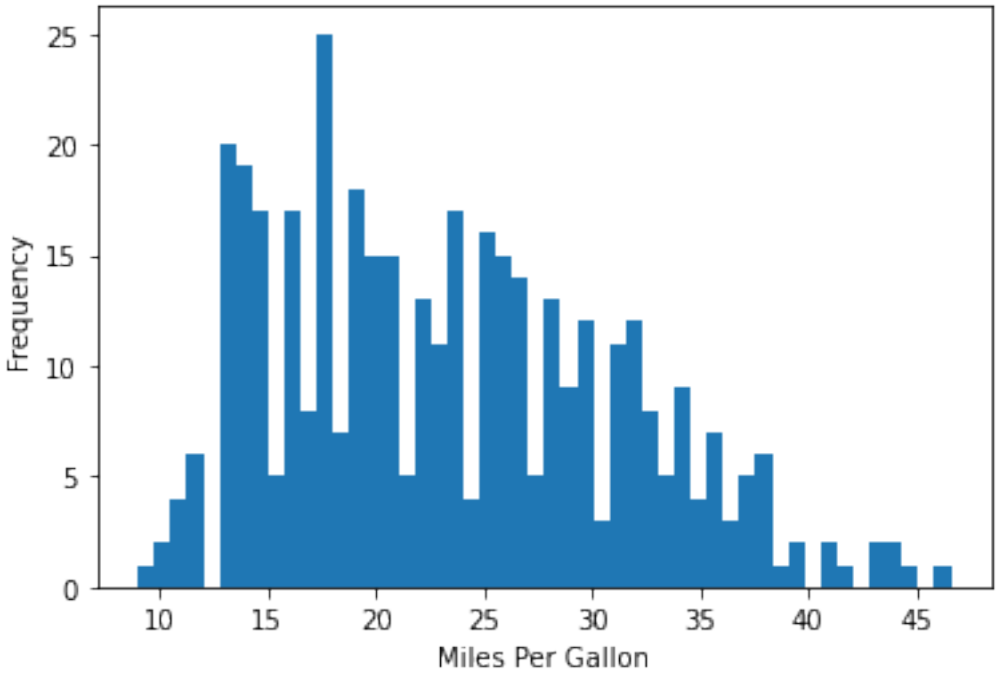
Question 3 Create a histogram of the `mpg` column with the number ***50*** bins. Change the x-axis of the visualization to ***'Miles Per Gallon'***

Set the result of the plot to the variable `ax`. Your code will look something like:

```
ax = # code to create a scatter plot
# ... other code
```

[43]:
```
ax = df['mpg'].plot.hist(bins=50)
ax.set_xlabel('Miles Per Gallon')
```

[43]:
```
Text(0.5, 0, 'Miles Per Gallon')
```



[44]:
```
# Question 3 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```

13

June 11, 2024

1 Activity: Aggregations

1.1 Introduction

In this activity you will practice using Pandas functionality to work with various aggregations.

This activity will cover the following topics: - Measure an aggregate statistic over a specific column. - Measure an aggregate statistic over a specific column of subsets using `groupby()` over one column. - Measure multiple aggregate statistics over a specific column of subsets using `groupby()` over one column. - Measure multiple aggregate statistics over a specific column of subsets using `groupby()` over multiple columns. - Measure multiple aggregate statistics over a specific column of subsets using `groupby()` over multiple columns. - Take the transpose of a DataFrame.

[21]:
```
import pandas as pd

# Data from https://github.com/mwaskom/seaborn-data/blob/
↪2b29313169bf8dfa77d8dc930f7bd3eba559a906/mpg.csv
df = pd.read_csv('mpg.csv')
```

Question 1 Assign the average of the `weight` column where the `origin` is `'usa'` to the variable `weight_usa`.

[22]:
```
weight_usa = df.groupby('origin')['weight'].mean()['usa']
```

[23]:
```
# Question 1 Grading Checks

assert isinstance(weight_usa, float), 'Did you assign a number to `weight_usa`?'
```

Question 2 Assign the maximum of the `weight` column where the `origin` is `'japan'` to the variable `weight_japan`.

[24]:
```
weight_japan = df.groupby('origin')['weight'].max()['japan']
```

[25]:
```
# Question 2 Grading Checks

print(weight_japan)
```

2930

Question 3 Using Pandas' `groupby()` method, group by the `cylinders` column to find the minimum and maximum `horsepower` and assign the result to the variable `weight_by_cylinder`.

[26]:
```
weight_by_cylinder = df.groupby('cylinders')['horsepower'].agg(['min','max'])
```

[27]:
```
# Question 3 Grading Checks

assert isinstance(weight_by_cylinder, pd.DataFrame), 'Did you create a␣
↪DataFrame with `groupby()`?'
```

14

June 11, 2024

1 Activity: Full OSEMN

1.1 Introduction

In this assignment, you will work on a data analysis project. This project will let you practice the skills you have learned in this course and write real code in Python.

You will perform the following steps of the OSEMN framework:
- Section 1.2 - Section 1.3 - Section 1.5

[128]:
```
# We'll import the libraries you'll likely use for this activity
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Data
df = pd.read_csv('transactions-pet_store.csv')
df_orig = df.copy()
```

1.2 Scrub

You will scrub the data. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

Question 1 Remove all rows that have are missing either the `Product_Name` or the `Product_Category`. Assign the cleaned DataFrame to the variable `df` (overwriting the original DataFrame.).

[129]:
```
df=df.dropna(subset=['Product_Name','Product_Category'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2758 entries, 0 to 2902
```

```
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              2758 non-null   object
 1   Order_Number      2758 non-null   object
 2   Customer_ID       2716 non-null   object
 3   Product_Name      2758 non-null   object
 4   SKU               2758 non-null   object
 5   Price             2758 non-null   float64
 6   Size              626 non-null    object
 7   Quantity          2758 non-null   int64
 8   Product_Category  2758 non-null   object
 9   Product_Line      2758 non-null   object
dtypes: float64(1), int64(1), object(8)
memory usage: 237.0+ KB
```

[130]:
```
# Question 1 Grading Checks

assert df.shape[0] <= 2874, 'Did you remove all the rows with missing values␣
↪for the columns Product_Name & Product_Category?'
assert df.shape[0] >= 2700, 'Did you remove too many the rows with missing␣
↪values?'
assert len(df.columns) == 10, 'Make sure you do not drop any columns.'
```

Question 2 Find any clearly "incorrect" values in the `Price` column and "clean" the DataFrame to address those values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[131]:
```
df.Price = df.Price[(df.Price<1000) & (df.Price>0)]
```

[132]:
```
# Question 2 Grading Checks

assert (df.Price < df.Price.quantile(0.0001)).sum() == 0, 'Check for very small␣
↪values'
assert (df.Price > df.Price.quantile(0.999)).sum() == 0, 'Check for very large␣
↪values'
```

Question 3 After you've done the cleaning above, remove any column that has more than `500` missing values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[133]:
```
df = df.drop(columns=df.columns [ (df.isna().sum() > 500) ] )
```

[134]:
```
# Question 3 Grading Checks

assert len(df.columns) < 10, 'You should have dropped 1 or more columns (with␣
↪more than 500 missing values)'
```

Question 4 Address the other missing values. You can replace the values or remvove them, but whatever method you decide to clean the DataFrame, you should no longer have any missing values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[135]:
```
df=df.dropna()
```

[136]:
```
# Question 4 Grading Checks

assert df.Customer_ID.isna().sum() == 0, 'Did you address all the missing␣
↪values?'
```

1.3 Explore

You will explore the data. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

You may use either exploratory statistics or exploratory visualizations to help answer these questions.

Note that the DataFrame loaded for this section (in the below cell) is different from the data you used in the Section 1.2 section.

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

[137]:
```
df = pd.read_csv('transactions-pet_store-clean.csv')
```

Question 5 Create a `Subtotal` column by multiplying the `Price` and `Quantity` values. This represents how much was spent for a given transaction (row).

[138]:
```
df['Subtotal'] = df.Price*df.Quantity
```

[139]:
```
# Question 5 Grading Checks

assert 'Subtotal' in df.columns, ''
```

Question 6 Determine most common category (`Product_Category`) purchases (number of total items) for both `Product_Line` categories. Assign the (string) name of these categories to their respective variables `common_category_cat` & `common_category_dog`.

[140]:
```
common_category_dog = (
    df[df.Product_Line=='dog']
    .groupby(['Product_Category'])
    .Quantity
    .agg('sum')
    .sort_values(ascending=False)
    .index[0]
)

common_category_cat = (
    df[df.Product_Line=='cat']
    .groupby(['Product_Category'])
    .Quantity
    .agg('sum')
    .sort_values(ascending=False)
    .index[0]
)
```

[141]:
```
# Question 6 Grading Checks

assert isinstance(common_category_dog, str), 'Ensure you assign the name of the␣
 ↪category (string) to the variable common_category_dog'
assert isinstance(common_category_cat, str), 'Ensure you assign the name of the␣
 ↪category (string) to the variable common_category_cat'
```

Question 7 Determine which categories (`Product_Category`), by `Product_Line` have the ***median*** highest `Price`. Assign the (string) name of these categories to their respective variables `priciest_category_cat` & `priciest_category_dog`.

[142]:
```
priciest_category_dog = (
    df[df.Product_Line=='dog']
    .groupby(['Product_Category'])
    .Price
    .agg('median')
    .sort_values(ascending=False)
    .index[0]
)

priciest_category_cat = (
    df[df.Product_Line=='cat']
    .groupby(['Product_Category'])
    .Price
    .agg('median')
```

```
    .sort_values(ascending=False)
    .index[0]
)
```

[143]:
```
# Question 7 Grading Checks

assert isinstance(priciest_category_dog, str), 'Ensure you assign the name of␣
 ↪the category (string) to the variable priciest_category_dog'
assert isinstance(priciest_category_cat, str), 'Ensure you assign the name of␣
 ↪the category (string) to the variable priciest_category_cat'
```

1.4 Modeling

This is the point of the framework where we'd work on modeling with our data. However, in this activity, we're going to move straight to interpretting.

1.5 Interpret

You will interpret the data based on what you found so far. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

Note that the DataFrame loaded for this section (in the below cell) is the same as the data you used in the Section 1.3 section.

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

Question 8 You want to emphasize to your stakeholders that the total number of product categories sold differ between the two `Product_Line` categories (`'cat'` & `'dog'`).

Create a ***horizontal*** **bar plot** that has `Product_Category` on the y-axis and the total number of that category sold (using the `Quantity`) by each `Product_Line` category. Also **change the axis labels** to something meaningful and add a title.

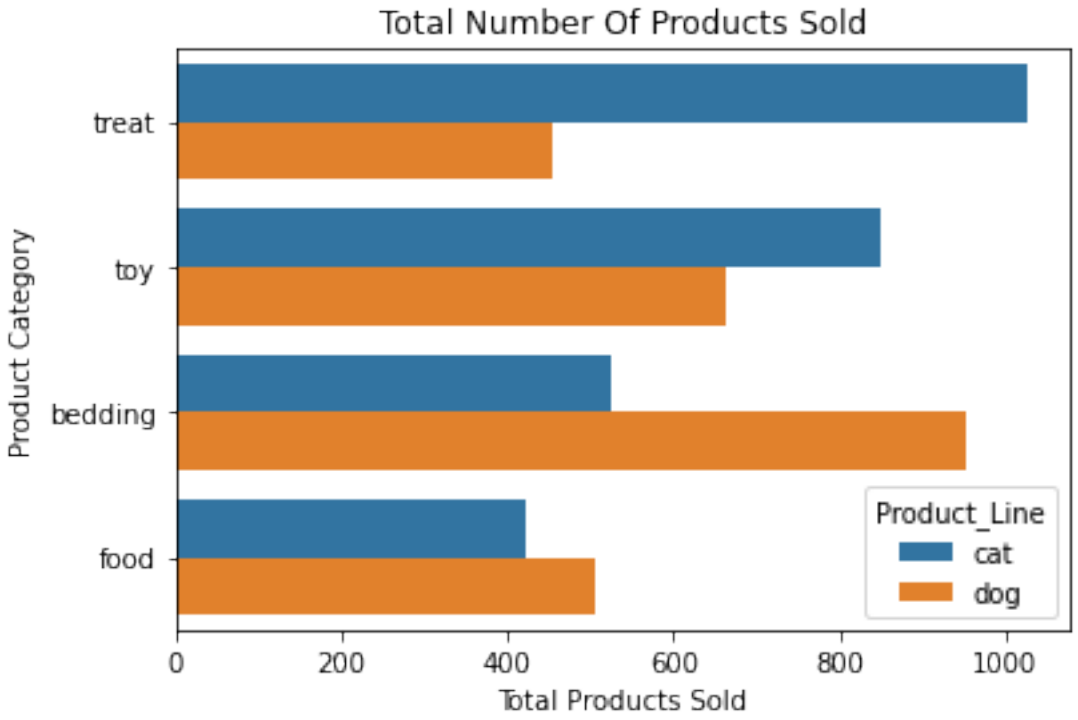
You will likely want to use Seaborn. Make sure you set the result to the variable `ax` like the following:

`ax = # code to create a bar plot`

[144]:
```
ax = sns.barplot(
    data=df,
    y='Product_Category',
    x='Quantity',
    estimator=sum,
    ci=None,
```

```
    hue='Product_Line',
)
ax.set_ylabel('Product Category')
ax.set_xlabel('Total Products Sold')
ax.set_title('Total Number Of Products Sold')
```

[144]:
```
Text(0.5, 1.0, 'Total Number Of Products Sold')
```



[145]:
```
# Question 8 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```

Question 9 Based on the plot from Section 1.5, what would you conclude for your stakeholders about what products they should sell? What would be the considerations and/or caveats you'd communicate to your stakeholders?

Write at least a couple sentences of your thoughts in a string assigned to the variable `answer_to_9`.

The cell below should look something like this:

```
answer_to_9 = '''
I think that based on the visualization that ****.
```

```
Therefore I would communicate with the stakeholders that ****
'''
```

[146]:
```
# Your code here
answer_to_9 = '''
Based on the following findings, the most sold product category are treats and␣
 ↪beddings
The number of products sold for cat treats is more than for dogs and in␣
 ↪beddings dogs are more in number
since these are our most sold products we need to concentrate on the gap␣
 ↪between the treats and beddings sold for both cats and dogs.
'''
```

[147]:
```
# Question 9 Grading Checks

assert isinstance(answer_to_9, str), 'Make sure you create a string for your␣
 ↪answer.'
```

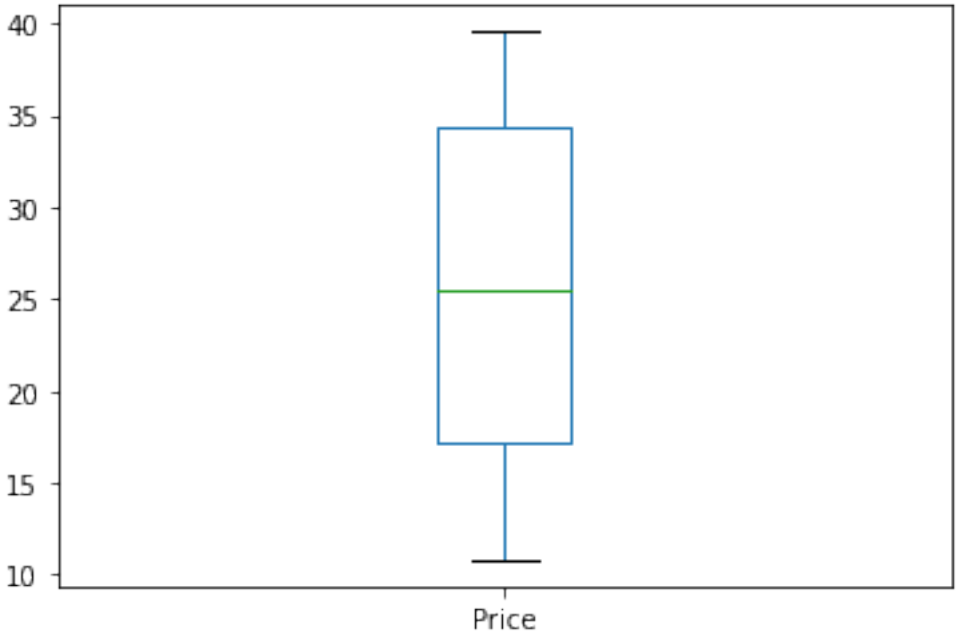
Question 10 The plot you created for Section 1.5 is good but could be modified to emphasize which products are important for the business.

Create an explanatory visualization that emphasizes the insight you about the product category. This would be a visualization you'd share with the business stakeholders.

Make sure you set the result to the variable `ax` like the following:

```
ax = # code to create explanatory visualization
```

[148]:
```
ax = df.Price.plot.box()
```



[149]:
```
# Question 10 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```