DSO 545: Statistical Computing and Data Visualization

Abbass Al Sharif

Fall 2019

Lab 7: Data Wrangling

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1. Data Science Tools: map(), apply(), and lambda functions

1. Load the titanic.csv dataset into a pandas DataFrame.

```
data = pd.read_csv("titanic.csv")
```

2. Use the map() function to create a new variable call "Sex_Numeric" (0 for males, and 1 for females).

```
data["Sex_Numeric"] = data['Sex'].map({"female":1, "male":0})
```

3. What is the percentage of females in the Titanic dataset?

```
data["Sex_Numeric"].mean()

### OR
```

```
## 0.35241301907968575
```

```
data.Sex.value_counts()/data.shape[0]
```

```
## male      0.647587
## female    0.352413
## Name: Sex, dtype: float64
```

4. Create a new variable called "Fare_ceil" to show the fare value of the trip rounded up.

```
data['Fare_ceil'] = data['Fare'].apply(np.ceil)
```

5. Load the drinks.csv dataset into a dataset called drinks.

```
drinks = pd.read_csv("drinks.csv")
```

6. Find the max beer, spirit, and wine servings among all countries.

```
drinks.loc[:,'beer_servings':'wine_servings'].apply(max, axis = 0)
```

```
## beer_servings      376
## spirit_servings    438
## wine_servings      370
## dtype: int64
```

7. Find the max serving among beer, spirit, and wine for each country.

```
drinks.loc[:,'beer_servings':'wine_servings'].apply(max, axis = 1).head()
```

```
## 0      0
## 1    132
## 2     25
## 3    312
## 4    217
## dtype: int64
```

8. Find the max category of serving among beer, spirit, and wine for each country.

```
drinks.loc[:,'beer_servings':'wine_servings'].apply(np.argmax, axis = 1).head()
```

```
## 0      beer_servings
## 1    spirit_servings
## 2      beer_servings
## 3      wine_servings
## 4      beer_servings
## dtype: object
##
## /anaconda3/envs/r-reticulate/lib/python3.7/site-packages/numpy/core/fromnumeric.py:61: FutureWarning
## The current behaviour of 'Series.argmax' is deprecated, use 'idxmax'
## instead.
## The behavior of 'argmax' will be corrected to return the positional
## maximum in the future. For now, use 'series.values.argmax' or
## 'np.argmax(np.array(values))' to get the position of the maximum
## row.
##   return bound(*args, **kwds)
```

9. Creata a function that take a input value x and returns it four-fold.

```
def four_fold(x):
  return 4*x

four_fold(3)
```

```
## 12
```

10. Create a new column in the drinks dataset `beer_4fold` which multiplies the value of the beer servings in each country by 4.

```
drinks['beer_4fold'] = drinks['beer_servings'].apply(four_fold).head()
```

11. Use a lambda function to answer the previous question, i.e. don't use the function `four_fold()`.

```
drinks['beer_4fold'] = drinks['beer_servings'].apply(lambda x:4*x).head()

## OR

drinks['beer_4fold'] = list(map(lambda x:4*x, drinks['beer_servings']))
```

2. Data Wrangling

Filtering Data

12. Find all passangers who are above 30 years old.

```
s1 = data[data.Age > 30]
```

13. Find all female passangers who are above 30 years old.

```
s2 = data[(data.Age > 30) & (data.Sex == "female")]
```

Selecting Variables

14. Create a dataframe which has only two columns: `PassengerId`, `Survived`, and `Cabin`.

```
s3 = data[['PassengerId', 'Survived', 'Cabin']]
```

15. Create a dataframe that has all variables in the dataset except the "Cabin" variable.

```
s4 = s3.drop('Cabin', axis = 1)
```

Arranging Data

16. Find all female passangers who are above 30 years old. The resulting dataframe should have three columns: PassengerId, Survived, Age. The dataframe should be arranged by age in descending order.

```
data.loc[(data.Age > 30) & (data.Sex == "female"),
         ["PassengerId", "Survived", "Age"]].sort_values('Age', ascending = False)
```

```
##      PassengerId  Survived   Age
## 275          276         1  63.0
## 483          484         1  63.0
## 829          830         1  62.0
## 366          367         1  60.0
## 195          196         1  58.0
## ..           ...       ...   ...
## 215          216         1  31.0
## 328          329         1  31.0
## 18            19         0  31.0
## 318          319         1  31.0
## 767          768         0  30.5
##
## [103 rows x 3 columns]
```

Grouping and Summarizing Data

17. Find the average age of both male and female passangers.

```
data.groupby('Sex')['Age'].mean()
```

```
## Sex
## female    27.915709
## male      30.726645
## Name: Age, dtype: float64
```

18. Find the median fare for passengers according to their class.

```
data.groupby('Pclass').Fare.median()
```

```
## Pclass
## 1    60.2875
## 2    14.2500
## 3     8.0500
## Name: Fare, dtype: float64
```

19. Find the average and median, and the difference between mean and median age of both male and female passangers.

```
data.groupby('Sex')['Age'].agg({"Average": 'mean',
                                "Median": 'median',
                                "Diff": lambda x: x.mean()- x.median()})
```

```
##           Average  Median      Diff
## Sex
## female  27.915709    27.0  0.915709
## male    30.726645    29.0  1.726645
##
## /anaconda3/envs/r-reticulate/bin/python:3: FutureWarning: using a dict on a Series for aggregation
## is deprecated and will be removed in a future version. Use                 named aggregation instead
##
##     >>> grouper.agg(name_1=func_1, name_2=func_2)
```

20. Find the average age for males and females who survived the Titanic disaster.

```
data.groupby(['Sex', 'Survived'])['Age'].mean()
```

```
## Sex     Survived
## female  0           25.046875
##         1           28.847716
## male    0           31.618056
##         1           27.276022
## Name: Age, dtype: float64
```

```
# we can reset the index to make the data a dataframe
data.groupby(['Sex', 'Survived'])['Age'].mean().reset_index()
```

```
##       Sex  Survived        Age
## 0  female         0  25.046875
## 1  female         1  28.847716
## 2    male         0  31.618056
## 3    male         1  27.276022
```

21. Find the median fare for passengers embarked from different ports and among different classes.

```
data.groupby(['Pclass','Embarked']).Fare.median()
```

```
## Pclass  Embarked
## 1       C           78.2667
##         Q           90.0000
##         S           52.0000
## 2       C           24.0000
##         Q           12.3500
##         S           13.5000
## 3       C            7.8958
##         Q            7.7500
##         S            8.0500
## Name: Fare, dtype: float64
```

Mutating Data

22. Create a new column (age_cat) in the dataset to based on the age variable ("young" if age <=50 otherwise "older").

```
data['age_cat'] = data['Age'].apply(lambda x: "Young" if x<=50 else "Older")
data.age_cat.unique()
```

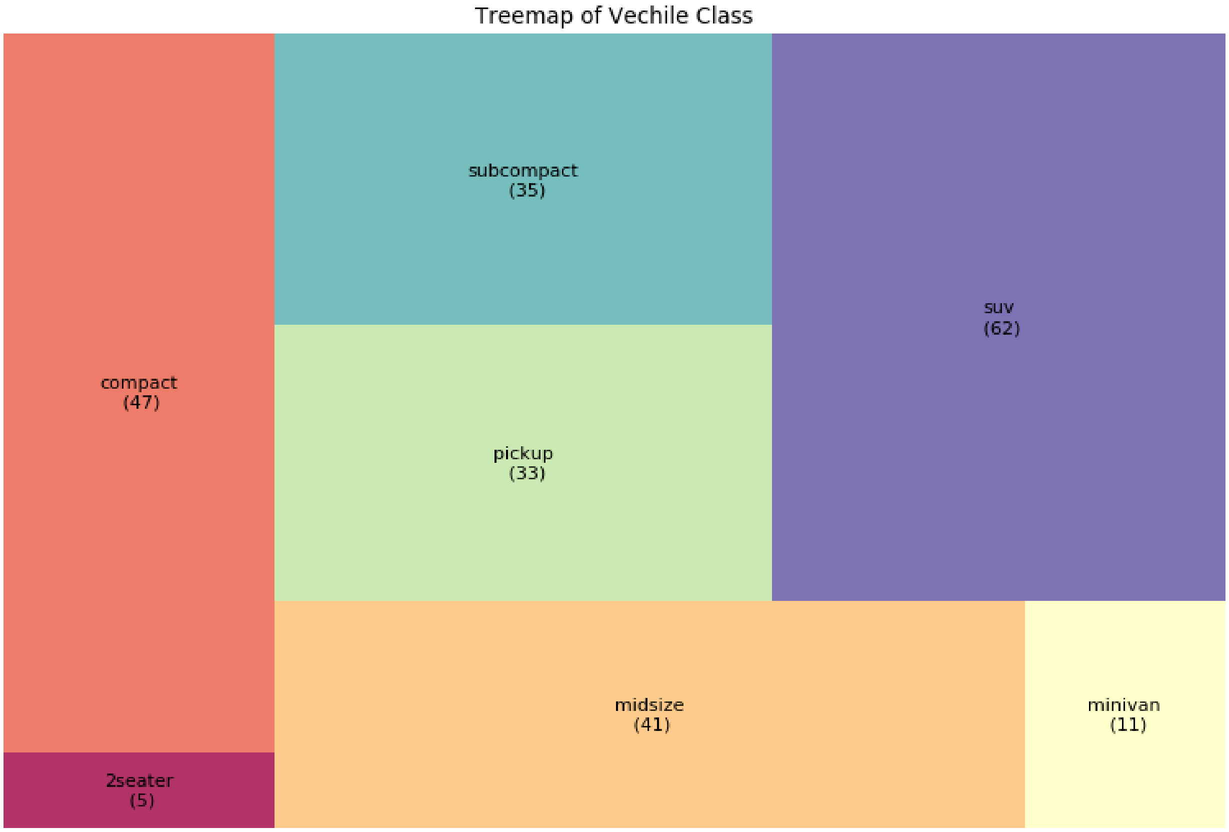
```
## array(['Young', 'Older'], dtype=object)
```

23. Create a new column (age_cat1) in the dataset to based on the age variable ("young" if age <=20, "mature" if 20<age<=50 otherwise "older").

```
data['age_cat1'] = data['Age'].apply(lambda x: "Young" if x <= 20 else ("Mature" if x <= 50 else "Older
data.age_cat1.unique()
```

```
## array(['Mature', 'Older', 'Young'], dtype=object)
```

24. Create the following Tree map using the `mpg.csv` dataset and `squarify` Python package.


Treemap of Vechile Class
subcompact
(35)
suv
(62)
compact
(47)
pickup
(33)
midsize
(41)
minivan
(11)
2seater
(5)