01 GETTING & KNOWING YOUR DATA

GETTING AND KNOWING YOUR DATA

CHIPOTLE

Step 1. Import the necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

```
url =
'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'

chipo = pd.read_csv(url, sep = '\t')
```

Step 4. See the first 10 entries

In [3]:

```
chipo.head(10)
```

Step 5. What is the number of observations in the dataset?

```
# Solution 1

chipo.shape[0]  # entries <= 4622 observations


# Solution 2

chipo.info() # entries <= 4622 observations
```

Step 6. What is the number of columns in the dataset?

In [6]:

```
chipo.shape[1]
```

Step 7. Print the name of all the columns.

```
chipo.columns
```

Step 8. How is the dataset indexed?

In [8]:
```
chipo.index
```

Step 9. Which was the most-ordered item?

In [9]:
```
c = chipo.groupby('item_name')
c = c.sum()
c = c.sort_values(['quantity'], ascending=False)
c.head(1)
```

Step 10. For the most-ordered item, how many items were ordered?

In [10]:
```
c = chipo.groupby('item_name')
c = c.sum()
c = c.sort_values(['quantity'], ascending=False)
c.head(1)
```

Step 11. What was the most ordered item in the choice_description column?

In [11]:
```
c = chipo.groupby('choice_description').sum()
c = c.sort_values(['quantity'], ascending=False)
c.head(1)
# Diet Coke 159
```

Step 12. How many items were orderd in total?

In [12]:
```
total_items_orders = chipo.quantity.sum()
total_items_orders
```

Step 13. Turn the item price into a float

Step 13.a. Check the item price type

```
chipo.item_price.dtype
```

Step 13.b. Create a lambda function and change the type of item price

In [14]:

```
dollarizer = lambda x: float(x[1:-1])
chipo.item_price = chipo.item_price.apply(dollarizer)
```

Step 13.c. Check the item price type

In [15]:

```
chipo.item_price.dtype
```

Step 14. How much was the revenue for the period in the dataset?

```
revenue = (chipo['quantity']* chipo['item_price']).sum()

print('Revenue was: $' + str(np.round(revenue,2)))
```

Step 15. How many orders were made in the period?

In [23]:

```
orders = chipo.order_id.value_counts().count()
orders
```

Step 16. What is the average revenue amount per order?

In [31]:

```
# Solution 1

chipo['revenue'] = chipo['quantity'] * chipo['item_price']
order_grouped = chipo.groupby(by=['order_id']).sum()
order_grouped.mean()['revenue']


# Solution 2

chipo.groupby(by=['order_id']).sum().mean()['revenue']
```

Step 17. How many different items are sold?

```
chipo.item_name.value_counts().count()
```

GETTING AND KNOWING YOUR DATA

OCCUPATION

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called users and use the 'user_id' as index

```
users =
pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u
.user',
                      sep='|', index_col='user_id')
```

Step 4. See the first 25 entries

```
users.head(25)
```

Step 5. See the last 10 entries

```
users.tail(10)
```

Step 6. What is the number of observations in the dataset?

```
users.shape[0]
```

Step 7. What is the number of columns in the dataset?

```
users.shape[1]
```

Step 8. Print the name of all the columns.

```
users.columns
```

Step 9. How is the dataset indexed?

```
# "the index" (aka "the labels")
users.index
```

Step 10. What is the data type of each column?

```
users.dtypes
```

Step 11. Print only the occupation column

```
users.occupation

#or

users['occupation']
```

Step 12. How many different occupations are in this dataset?

```
users.occupation.nunique()
#or by using value_counts() which returns the count of unique elements
#users.occupation.value_counts().count()
```

Step 13. What is the most frequent occupation?

```
#Because "most" is asked
users.occupation.value_counts().head(1).index[0]

#or
#to have the top 5

# users.occupation.value_counts().head()
```

Step 14. Summarize the DataFrame.

```
users.describe() #Notice: by default, only the numeric columns are returned.
```

Step 15. Summarize all the columns

```
users.describe(include = "all") #Notice: By default, only the numeric columns
are returned.
```

Step 16. Summarize only the occupation column

```
users.occupation.describe()
```

Step 17. What is the mean age of users?

```
round(users.age.mean())
```

Step 18. What is the age with least occurrence?

```
users.age.value_counts().tail() #7, 10, 11, 66 and 73 years -> only 1
occurrence
```

01 GETTING & KNOWING YOUR DATA

GETTING AND KNOWING YOUR DATA

WORLD FOOD FACTS

Step 1. Go to https://www.kaggle.com/openfoodfacts/world-food-facts/data

Step 2. Download the dataset to your computer and unzip it.

```
import pandas as pd
import numpy as np
```

Step 3. Use the tsv file and assign it to a dataframe called food

```
food = pd.read_csv('~/Desktop/en.openfoodfacts.org.products.tsv', sep='\t')
```

Step 4. See the first 5 entries

```
food.head()
```

Step 5. What is the number of observations in the dataset?

```
food.shape #will give you both (observations/rows, columns)
food.shape[0] #will give you only the observations/rows number
```

Step 6. What is the number of columns in the dataset?

```
print(food.shape) #will give you both (observations/rows, columns)
print(food.shape[1]) #will give you only the columns number

#OR

food.info() #Columns: 163 entries
```

Step 7. Print the name of all the columns.

```
food.columns
```

Step 8. What is the name of 105th column?

```
food.columns[104]
```

Step 9. What is the type of the observations of the 105th column?

```
food.dtypes['-glucose_100g']
```

Step 10. How is the dataset indexed?

```
food.index
```

Step 11. What is the product name of the 19th observation?

```
food.values[18][7]
```

02 FILTERING AND SORTING DATA

CHIPOTLE

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

```
url =
'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'

chipo = pd.read_csv(url, sep = '\t')
```

Step 4. How many products cost more than $10.00?

```
# clean the item_price column and transform it in a float
prices = [float(value[1 : -1]) for value in chipo.item_price]

# reassign the column with the cleaned prices
chipo.item_price = prices

# delete the duplicates in item_name and quantity
chipo_filtered =
chipo.drop_duplicates(['item_name','quantity','choice_description'])

# chipo_filtered

# select only the products with quantity equals to 1
chipo_one_prod = chipo_filtered[chipo_filtered.quantity == 1]
chipo_one_prod

# chipo_one_prod[chipo_one_prod['item_price']>10].item_name.nunique()
# chipo_one_prod[chipo_one_prod['item_price']>10]



chipo.query('price_per_item > 10').item_name.nunique()
```

Step 5. What is the price of each item?

print a data frame with only two columns item_name and item_price

In [25]:

```
# delete the duplicates in item_name and quantity
# chipo_filtered = chipo.drop_duplicates(['item_name','quantity'])
```

```
chipo[(chipo['item_name'] == 'Chicken Bowl') & (chipo['quantity'] == 1)]

# select only the products with quantity equals to 1
# chipo_one_prod = chipo_filtered[chipo_filtered.quantity == 1]

# select only the item_name and item_price columns
# price_per_item = chipo_one_prod[['item_name', 'item_price']]

# sort the values from the most to less expensive
# price_per_item.sort_values(by = "item_price", ascending = False).head(20)
```

Step 6. Sort by the name of the item

In [25]:
```
chipo.item_name.sort_values()

# OR

chipo.sort_values(by = "item_name")
```

Step 7. What was the quantity of the most expensive item ordered?

In [26]:
```
chipo.sort_values(by = "item_price", ascending = False).head(1)
```

Step 8. How many times was a Veggie Salad Bowl ordered?

In [18]:
```
chipo_salad = chipo[chipo.item_name == "Veggie Salad Bowl"]

len(chipo_salad)
```

Step 9. How many times did someone order more than one Canned Soda?

In [28]:
```
chipo_drink_steak_bowl = chipo[(chipo.item_name == "Canned Soda") &
(chipo.quantity > 1)]
len(chipo_drink_steak_bowl)
```

02 FILTERING AND SORTING DATA

EURO12

Step 1. Import the necessary libraries

In [2]:

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called euro12.

In [3]:

```
euro12 =
pd.read_csv('https://raw.githubusercontent.com/guipsamora/pandas_exercises/ma
ster/02_Filtering_%26_Sorting/Euro12/Euro_2012_stats_TEAM.csv', sep=',')
euro12
```

Step 4. Select only the Goal column.

In [37]:

```
euro12.Goals
```

Step 5. How many team participated in the Euro2012?

In [43]:

```
euro12.shape[0]
```

Step 6. What is the number of columns in the dataset?

In [44]:

```
euro12.info()
```

Step 7. View only the columns Team, Yellow Cards and Red Cards and assign them to a dataframe called discipline

In [82]:

```
# filter only giving the column names

discipline = euro12[['Team', 'Yellow Cards', 'Red Cards']]
discipline
```

Step 8. Sort the teams by Red Cards, then to Yellow Cards

In [56]:

```
discipline.sort_values(['Red Cards', 'Yellow Cards'], ascending = False)
```

Step 9. Calculate the mean Yellow Cards given per Team

In [55]:
```
round(discipline['Yellow Cards'].mean())
```

Step 10. Filter teams that scored more than 6 goals

In [57]:
```
euro12[euro12.Goals > 6]
```

Step 11. Select the teams that start with G

In [66]:
```
euro12[euro12.Team.str.startswith('G')]
```

Step 12. Select the first 7 columns

In [84]:
```
# use .iloc to slices via the position of the passed integers
# : means all, 0:7 means from 0 to 7

euro12.iloc[: , 0:7]
```

Step 13. Select all columns except the last 3.

In [86]:
```
# use negative to exclude the last 3 columns

euro12.iloc[: , :-3]
```

Step 14. Present only the Shooting Accuracy from England, Italy and Russia

In [89]:
```
# .loc is another way to slice, using the labels of the columns and indexes

euro12.loc[euro12.Team.isin(['England', 'Italy', 'Russia']),
['Team','Shooting Accuracy']]
```

02 FILTERING AND SORTING DATA

FICTIONAL ARMY

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. This is the data given as a dictionary

```
# Create an example dataframe about a fictional army
raw_data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks',
'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts',
'Scouts', 'Scouts', 'Scouts'],
            'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd',
'2nd','1st', '1st', '2nd', '2nd'],
            'deaths': [523, 52, 25, 616, 43, 234, 523, 62, 62, 73, 37, 35],
            'battles': [5, 42, 2, 2, 4, 7, 8, 3, 4, 7, 8, 9],
            'size': [1045, 957, 1099, 1400, 1592, 1006, 987, 849, 973, 1005,
1099, 1523],
            'veterans': [1, 5, 62, 26, 73, 37, 949, 48, 48, 435, 63, 345],
            'readiness': [1, 2, 3, 3, 2, 1, 2, 3, 2, 1, 2, 3],
            'armored': [1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1],
            'deserters': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
            'origin': ['Arizona', 'California', 'Texas', 'Florida', 'Maine',
'Iowa', 'Alaska', 'Washington', 'Oregon', 'Wyoming', 'Louisana', 'Georgia']}
```

Step 3. Create a dataframe and assign it to a variable called army.

Don't forget to include the columns names in the order presented in the dictionary ('regiment', 'company', 'deaths'...) so that the column index order is consistent with the solutions. If omitted, pandas will order the columns alphabetically.

```
army = pd.DataFrame(data=raw_data)
army
```

Step 4. Set the 'origin' colum as the index of the dataframe

```
army.set_index('origin', inplace=True)
```

Step 5. Print only the column veterans

```
army.veterans
```

Step 6. Print the columns 'veterans' and 'deaths'

```
army[["veterans", "deaths"]]
```

Step 7. Print the name of all the columns.

```
army.columns
```

Step 8. Select the 'deaths', 'size' and 'deserters' columns from Maine and Alaska

```
army.loc[["Maine", "Alaska"], ["deaths", "size", "deserters"]]
```

Step 9. Select the rows 3 to 7 and the columns 3 to 6

```
army.iloc[2:7, 2:6]
```

Step 10. Select every row after the fourth row and all columns

```
army.iloc[4:, :]
```

Step 11. Select every row up to the 4th row and all columns

```
army.iloc[:4, :]
```

Step 12. Select the 3rd column up to the 7th column

```
army.iloc[:, 2:7]
```

Step 13. Select rows where df.deaths is greater than 50

```
army[army["deaths"] > 50]
```

Step 14. Select rows where df.deaths is greater than 500 or less than 50

```
army[(army["deaths"] > 500) | (army["deaths"] < 50)]
```

Step 15. Select all the regiments not named "Dragoons"

```
army[army["regiment"] != "Dragoons"]
```

Step 16. Select the rows called Texas and Arizona

```
army.loc[["Texas", "Arizona"], :]
```

Step 17. Select the third cell in the row named Arizona

```
army.loc[["Arizona"]].iloc[:, 2]
```

Step 18. Select the third cell down in the column named deaths

```
army.loc[:, ["deaths"]].iloc[2]
```

03 GROUPING

ALCOHOL CONSUMPTION

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called drinks.

```
drinks =
pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/d
rinks.csv')
drinks.head()
```

Step 4. Which continent drinks more beer on average?

```
drinks.groupby('continent').beer_servings.mean()
```

Step 5. For each continent print the statistics for wine consumption.

```
drinks.groupby('continent').wine_servings.describe()
```

Step 6. Print the mean alcohol consumption per continent for every column

```
drinks.groupby('continent').mean()
```

Step 7. Print the median alcohol consumption per continent for every column

```
drinks.groupby('continent').median()
```

Step 8. Print the mean, min and max values for spirit consumption.

This time output a DataFrame

```
drinks.groupby('continent').spirit_servings.agg(['mean', 'min', 'max'])
```

03 GROUPING

OCCUPATION

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called users.

```
users = 
pd.read_table('https://raw.githubusercontent.com/justmarkham/DAT8/master/data
/u.user', 
                      sep='|', index_col='user_id')
users.head()
```

Step 4. Discover what is the mean age per occupation

```
users.groupby('occupation').age.mean()
```

Step 5. Discover the Male ratio per occupation and sort it from the most to the least

```
# create a function
def gender_to_numeric(x):
    if x == 'M':
        return 1
    if x == 'F':
        return 0

# apply the function to the gender column and create a new column
users['gender_n'] = users['gender'].apply(gender_to_numeric)


a = users.groupby('occupation').gender_n.sum() / 
users.occupation.value_counts() * 100

# sort to the most male 
a.sort_values(ascending = False)
```

Step 6. For each occupation, calculate the minimum and maximum ages

```
users.groupby('occupation').age.agg(['min', 'max'])
```

Step 7. For each combination of occupation and gender, calculate the mean age

In [152]:
```
users.groupby(['occupation', 'gender']).age.mean()
```

Step 8. For each occupation present the percentage of women and men

In [154]:
```
# create a data frame and apply count to gender
gender_ocup = users.groupby(['occupation', 'gender']).agg({'gender':
'count'})

# create a DataFrame and apply count for each occupation
occup_count = users.groupby(['occupation']).agg('count')

# divide the gender_ocup per the occup_count and multiply per 100
occup_gender = gender_ocup.div(occup_count, level = "occupation") * 100

# present all rows from the 'gender column'
occup_gender.loc[: , 'gender']
```

03 GROUPING

REGIMENT

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Create the DataFrame with the following values:

```
raw_data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks',
'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts',
'Scouts', 'Scouts', 'Scouts'],
        'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd',
'2nd','1st', '1st', '2nd', '2nd'],
        'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon',
'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'],
        'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
        'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
```

Step 3. Assign it to a variable called regiment.

Don't forget to name each column

```
regiment = pd.DataFrame(raw_data, columns = raw_data.keys())
regiment
```

Step 4. What is the mean preTestScore from the regiment Nighthawks?

```
regiment[regiment['regiment'] == 'Nighthawks'].groupby('regiment').mean()
```

Step 5. Present general statistics by company

```
regiment.groupby('company').describe()
```

Step 6. What is the mean of each company's preTestScore?

```
regiment.groupby('company').preTestScore.mean()
```

Step 7. Present the mean preTestScores grouped by regiment and company

```
regiment.groupby(['regiment', 'company']).preTestScore.mean()
```

Step 8. Present the mean preTestScores grouped by regiment and company without heirarchical indexing

```
regiment.groupby(['regiment', 'company']).preTestScore.mean().unstack()
```

Step 9. Group the entire dataframe by regiment and company

```
regiment.groupby(['regiment', 'company']).mean()
```

Step 10. What is the number of observations in each regiment and company

```
regiment.groupby(['company', 'regiment']).size()
```

Step 11. Iterate over a group and print the name and the whole data from the regiment

```
# Group the dataframe by regiment, and for each regiment,
for name, group in regiment.groupby('regiment'):
    # print the name of the regiment
    print(name)
    # print the data of that regiment
    print(group)
```

04 APPLY

STUDENTS ALCOHOL CONSUMPTION

Introduction:

This time you will download a dataset from the UCI.

Step 1. Import the necessary libraries

```
import pandas as pd
import numpy
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called df.

```
csv_url =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/04_Appl
y/Students_Alcohol_Consumption/student-mat.csv'
df = pd.read_csv(csv_url)
df.head()
```

How many Columns

```
df.shape[1]
```

How many Rows

```
df.shape[0]
```

Print Column

```
df.columns
```

Step 4. For the purpose of this exercise slice the dataframe from 'school' until the 'guardian' column

```
stud_alcoh = df.loc[: , "school":"guardian"]
stud_alcoh.head()
```

Step 5. Create a lambda function that will capitalize strings.

```
capitalizer = lambda x: x.capitalize()
```

Step 6. Capitalize both Mjob and Fjob

```
stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'].apply(capitalizer)
```

Step 8. Did you notice the original dataframe is still lowercase? Why is that? Fix it and capitalize Mjob and Fjob.

```
stud_alcoh['Mjob'] = stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'] = stud_alcoh['Fjob'].apply(capitalizer)
stud_alcoh.tail()
```

Step 9. Create a function called majority that returns a boolean value to a new column called legal_drinker (Consider majority as older than 17 years old)

```
def majority(legal):
  if legal > 17:
    return 'Legal'
  else:
     return 'Not Legal'
stud_alcho['Legal'] = stud_alcho['age'].apply(majority)
stud_alcho.head()
```

Step 10. Multiply every number of the dataset by 10.

I know this makes no sense, don't forget it is just an exercise

```
def times10(x):
  if type(x) is int:
    return 10 * x
  return x
```

```
#stud_alcho.applymap(times10).head() (this multiply the whole table)
stud_alcho['Medu'] = stud_alcho['Medu'].map(times10) (only single table)
stud_alcho.head()
```

04 APPLY

UNITED STATES - CRIME RATES - 1960 - 2014

Step 1. Import the necessary libraries

In [95]:

```
import numpy as np
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called crime.

In [265]:

```
url =
"https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/04_Appl
y/US_Crime_Rates/US_Crime_Rates_1960_2014.csv"
crime = pd.read_csv(url)
crime.head()
```

Step 4. What is the type of the columns?

In [266]:

```
crime.info()
```

Have you noticed that the type of Year is int64. But pandas has a different type to work with Time Series. Let's see it now.

Step 5. Convert the type of the column Year to datetime64

In [267]:

```
# pd.to_datetime(crime)
crime.Year = pd.to_datetime(crime.Year, format='%Y')
crime.info()
```

Step 6. Set the Year column as the index of the dataframe

In [268]:

```
crime = crime.set_index('Year', drop = True)
crime.head()
```

Step 7. Delete the Total column

In [269]:

```
del crime['Total']
crime.head()
```

Step 8. Group the year by decades and sum the values

Pay attention to the Population column number, summing this column is a mistake

In [270]:

```
# To learn more about .resample (https://pandas.pydata.org/pandas-
docs/stable/generated/pandas.DataFrame.resample.html)
# To learn more about Offset Aliases (http://pandas.pydata.org/pandas-
docs/stable/timeseries.html#offset-aliases)

# Uses resample to sum each decade
crimes = crime.resample('10AS').sum()

# Uses resample to get the max value only for the "Population" column
population = crime['Population'].resample('10AS').max()

# Updating the "Population" column
crimes['Population'] = population

crimes
```

Step 9. What is the most dangerous decade to live in the US?

In [276]:

```
# apparently the 90s was a pretty dangerous time in the US
crime.idxmax(0)
```

05 MERGE

AUTO MPG

Step 1. Import the necessary libraries

```
import pandas as pd
import numpy as np
```

Step 2. Import the first dataset cars1 and cars2.

Step 3. Assign each to a to a variable called cars1 and cars2

```
cars1 =
pd.read_csv("https://raw.githubusercontent.com/guipsamora/pandas_exercises/ma
ster/05_Merge/Auto_MPG/cars1.csv")
cars2 =
pd.read_csv("https://raw.githubusercontent.com/guipsamora/pandas_exercises/ma
ster/05_Merge/Auto_MPG/cars2.csv")

print(cars1.head())
print(cars2.head())
```

Step 4. Oops, it seems our first dataset has some unnamed blank columns, fix cars1

```
cars1 = cars1.loc[:, "mpg":"car"]
cars1.head()
```

Step 5. What is the number of observations in each dataset?

```
print(cars1.shape)
print(cars2.shape)
```

```
(198, 9)
(200, 9)
```

Step 6. Join cars1 and cars2 into a single DataFrame called cars

```
#cars = pd.concat([car1, car2], ignore_index=True)
cars = pd.concat([car1, car2])
cars.tail()
```

Step 7. Oops, there is a column missing, called owners. Create a random number Series from 15,000 to 73,000.

```
car_owners = np.random.randint(15000, high=73001, size=398, dtype='int')
car_owners
```

Step 8. Add the column owners to cars

```
cars['owners'] = car_owners
cars.tail()
```

05 MERGE

FICTITOUS NAMES

Step 1. Import the necessary libraries

In [1]:

```
import pandas as pd
```

Step 2. Create the 3 DataFrames based on the following raw data

In [2]:

```
raw_data_1 = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}

raw_data_2 = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}

raw_data_3 = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
```

Step 3. Assign each to a variable called data1, data2, data3

In [12]:

```
data1 = pd.DataFrame(raw_data_1, columns = ['subject_id', 'first_name',
'last_name'])
data2 = pd.DataFrame(raw_data_2, columns = ['subject_id', 'first_name',
'last_name'])
data3 = pd.DataFrame(raw_data_3, columns = ['subject_id','test_id'])

data3
```

Step 4. Join the two dataframes along rows and assign all_data

```
all_data = pd.concat([data1, data2])
all_data
#mew_data = pd.concat([data1, data2], ignore_index=True)
#mew_data
```

Step 5. Join the two dataframes along columns and assing to all_data_col

In [10]:

```
all_data_col = pd.concat([data1, data2], axis = 1)
all_data_col
```

Step 6. Print data3

In [13]:

```
data3
```

Step 7. Merge all_data and data3 along the subject_id value

In [15]:

```
pd.merge(all_data, data3, on='subject_id')
```

Step 8. Merge only the data that has the same 'subject_id' on both data1 and data2

In [16]:

```
pd.merge(data1, data2, on='subject_id', how='inner')
```

Step 9. Merge all values in data1 and data2, with matching records from both sides where available.

In [17]:

```
pd.merge(data1, data2, on='subject_id', how='outer')
```

05 MERGE

HOUSING MARKET

Step 1. Import the necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
```

Step 2. Create 3 differents Series, each of length 100, as follows:

1. The first a random number from 1 to 4
2. The second a random number from 1 to 3
3. The third a random number from 10,000 to 30,000

```
s1 = pd.Series(np.random.randint(1, high=5, size=100, dtype='int'))
s2 = pd.Series(np.random.randint(1, high=4, size=100, dtype='int'))
s3 = pd.Series(np.random.randint(1000, high=30001, size=100, dtype='int'))
print(s1, s2, s3)
```

Step 3. Let's create a DataFrame by joinning the Series by column

In [29]:

```
housemkt = pd.concat([s1, s2, s3], axis=1)
housemkt.head()
```

Step 4. Change the name of the columns to bedrs, bathrs, price_sqr_meter

In [36]:

```
housemkt.rename(columns = {0: 'bedrs', 1: 'bathrs', 2: 'price_sqr_meter'},
inplace=True)
housemkt.head()
```

Step 5. Create a one column DataFrame with the values of the 3 Series and assign it to 'bigcolumn'

In [59]:

```
# join concat the values
bigcolumn = pd.concat([s1, s2, s3], axis=0)

# it is still a Series, so we need to transform it to a DataFrame
```

```
bigcolumn = bigcolumn.to_frame()
print(type(bigcolumn))

bigcolumn
```

Step 6. Oops, it seems it is going only until index 99. Is it true?

In [45]:

```
# no the index are kept but the length of the DataFrame is 300
len(bigcolumn)
```

Step 7. Reindex the DataFrame so it goes from 0 to 299

In [69]:

```
bigcolumn.reset_index(drop=True, inplace=True)
bigcolumn
```

06 STATS

US BABY NAMES

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called baby_names.

```
baby_names =
pd.read_csv('https://raw.githubusercontent.com/guipsamora/pandas_exercises/ma
ster/06_Stats/US_Baby_Names/US_Baby_Names_right.csv')
baby_names.info()
```

Step 4. See the first 10 entries

```
baby_names.head(10)
```

Step 5. Delete the column 'Unnamed: 0' and 'Id'

```
# deletes Unnamed: 0
del baby_names['Unnamed: 0']

# deletes Unnamed: 0
del baby_names['Id']

baby_names.head()
```

Step 6. Are there more male or female names in the dataset?

```
baby_names['Gender'].value_counts()
```

Step 7. Group the dataset by name and assign to names

```
# you don't want to sum the Year column, so you delete it
```

```
#del baby_names['Year']
#del baby_names['Gender']
#del baby_names['State']
```

```
# group the data
```

```
names = baby_names.groupby("Name").sum()

# print the first 5 observations
names.head()

# print the size of the dataset
print(names.shape)

# sort it from the biggest value to the smallest one
names.sort_values("Count", ascending = 0).head()
```

Step 8. How many different names exist in the dataset?

```
# as we have already grouped by the name, all the names are unique already.
# get the length of names
len(names)
```

Step 9. What is the name with most occurrences?

```
names.Count.idxmax()

# OR

# names[names.Count == names.Count.max()]
```

Step 10. How many different names have the least occurrences?

```
len(names[names.Count == names.Count.min()])
```

Step 11. What is the median name occurrence?

```
names[names.Count == names.Count.median()]
```

Step 12. What is the standard deviation of names?

```
names.Count.std()
```

Step 13. Get a summary with the mean, min, max, std and quartiles.

```
names.describe()
```

06 STATS

WIND STATISTICS

Step 1. Import the necessary libraries

In [1]:

```
import pandas as pd
import datetime
```

Step 2. Import the dataset from this [address]()

Step 3. Assign it to a variable called data and replace the first 3 columns by a proper datetime index.

In [3]:

```
# parse_dates gets 0, 1, 2 columns and parses them as the index
data_url =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/06_Stat
s/Wind_Stats/wind.data'
data = pd.read_csv(data_url, sep = "\s+", parse_dates = [[0,1,2]])
data.head()
```

Step 4. Year 2061? Do we really have data from this year? Create a function to fix it and apply it.

In [4]:

```
# The problem is that the dates are 2061 and so on...

# function that uses datetime
def fix_century(x):
  year = x.year - 100 if x.year > 1989 else x.year
  return datetime.date(year, x.month, x.day)

# apply the function fix_century on the column and replace the values to the
right ones
data['Yr_Mo_Dy'] = data['Yr_Mo_Dy'].apply(fix_century)

# data.info()
data.head()
```

Step 5. Set the right dates as the index. Pay attention at the data type, it should be datetime64[ns].

In [5]:

```
# transform Yr_Mo_Dy it to date type datetime64
data["Yr_Mo_Dy"] = pd.to_datetime(data["Yr_Mo_Dy"])

# set 'Yr_Mo_Dy' as the index
data = data.set_index('Yr_Mo_Dy')

data.head()
# data.info()
```

Step 6. Compute how many values are missing for each location over the entire record.

They should be ignored in all calculations below.

In [6]:

```
# "Number of non-missing values for each location: "
data.isnull().sum()
```

Step 7. Compute how many non-missing values there are in total.

In [10]:

```
#number of columns minus the number of missing values for each location
data.shape[0] - data.isnull().sum()

#or

data.notnull().sum()
```

Step 8. Calculate the mean windspeeds of the windspeeds over all the locations and all the times.

A single number for the entire dataset.

In [11]:

```
data.sum().sum() / data.notna().sum().sum()
```

Step 9. Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days

A different set of numbers for each location.

In [12]:

```
data.describe(percentiles=[])
```

Step 10. Create a DataFrame called day_stats and calculate the min, max and mean windspeed and standard deviations of the windspeeds across all the locations at each day.

A different set of numbers for each day.

In [13]:

```
# create the dataframe
day_stats = pd.DataFrame()

# this time we determine axis equals to one so it gets each row.
day_stats['min'] = data.min(axis = 1) # min
day_stats['max'] = data.max(axis = 1) # max
day_stats['mean'] = data.mean(axis = 1) # mean
day_stats['std'] = data.std(axis = 1) # standard deviations

day_stats.head()
```

Step 11. Find the average windspeed in January for each location.

Treat January 1961 and January 1962 both as January.

In [14]:

```
data.loc[data.index.month == 1].mean()
```

Step 12. Downsample the record to a yearly frequency for each location.

```
data.groupby(data.index.to_period('A')).mean()
```

Step 13. Downsample the record to a monthly frequency for each location.

```
data.groupby(data.index.to_period('M')).mean()
```

Step 14. Downsample the record to a weekly frequency for each location.

```
data.groupby(data.index.to_period('W')).mean()
```

Step 15. Calculate the min, max and mean windspeeds and standard deviations of the windspeeds across all locations for each week (assume that the first week starts on January 2 1961) for the first 52 weeks.

```
# resample data to 'W' week and use the functions
weekly = data.resample('W').agg(['min','max','mean','std'])
```

```
# slice it for the first 52 weeks and locations
weekly.loc[weekly.index[1:53], "RPT":"MAL"] .head(10)
```

https://www.perplexity.ai/

Ages: 22, 25, 29, 30, 31, 32, 35, 36, 40, 42

Simplified Interpretation of Mean and Standard Deviation

Mean Age

- The **mean age** of the group is **34.2 years**. This means that if you add up all the ages and divide by the number of people, you get 34.2 years. It represents the average age of everyone in this group.

Standard Deviation

- The **standard deviation** is **6.59 years**. This number tells us how much the ages differ from the average age (34.2 years).

What This Means

- Since the standard deviation is about **6.59 years**, it suggests that most people's ages are close to the average age of 34.2 years. Specifically, we can say that most individuals are likely between **27 years old and 41 years old** (34.2 - 6.59 to 34.2 + 6.59).

Understanding Variability

- The standard deviation helps us understand how spread out the ages are in this group:
- If the standard deviation were small (like 1 or 2), it would mean that most ages are very close to the average.
- A larger standard deviation (like 6.59) indicates that there is a wider range of ages in the group.

Importance of This Information

- Knowing the average age and how much variation there is can help in many fields:
- **Marketing**: Companies can target their products better by knowing the typical age of their customers.
- **Healthcare**: Understanding age distribution can help in planning health services for different age groups.
- **Social Sciences**: Researchers can analyze trends and behaviors based on age demographics.

In summary, the mean gives us a central point (the average age), while the standard deviation tells us how much people’s ages vary around that average. Together, they provide valuable insights into the group's characteristics!

07 VISUALIZATION

CHIPOTLE

Step 1. Import the necessary libraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter

# set this so the
%matplotlib inline
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

In [2]:

```
url =
'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'

chipo = pd.read_csv(url, sep = '\t')
```

Step 4. See the first 10 entries

In [134]:

```
chipo.head(10)
```

Step 5. Create a histogram of the top 5 items bought

In [4]:

```
# get the Series of the names
x = chipo.item_name

# use the Counter class from collections to create a dictionary with
keys(text) and frequency
letter_counts = Counter(x)

# convert the dictionary to a DataFrame
df = pd.DataFrame.from_dict(letter_counts, orient='index')

# sort the values from the top to the least value and slice the first 5 items
df = df[0].sort_values(ascending = True)[45:50]
```

```
# create the plot
df.plot(kind='bar')

# Set the title and labels
plt.xlabel('Items')
plt.ylabel('Number of Times Ordered')
plt.title('Most ordered Chipotle\'s Items')

# show the plot
plt.show()
```

Step 6. Create a scatterplot with the number of items orderered per order price

Hint: Price should be in the X-axis and Items ordered in the Y-axis

In [5]:

```
# create a list of prices
chipo.item_price = [float(value[1:-1]) for value in chipo.item_price] # strip
the dollar sign and trailing space

# then groupby the orders and sum
orders = chipo.groupby('order_id').sum()

# creates the scatterplot
# plt.scatter(orders.quantity, orders.item_price, s = 50, c = 'green')
plt.scatter(x = orders.item_price, y = orders.quantity, s = 50, c = 'green')

# Set the title and labels
plt.xlabel('Order Price')
plt.ylabel('Items ordered')
plt.title('Number of items ordered per order price')
plt.ylim(0)
```

07 VISUALIZATION

ONLINE RETAIL

Step 1. Import the necessary libraries

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

# set the graphs to show in the jupyter notebook
%matplotlib inline

# set seaborn graphs to a better style
sns.set(style="ticks")
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called online_rt

Note: if you receive a utf-8 decode error, set encoding = 'latin1' in pd.read_csv().

```
path =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/07_Visu
alization/Online_Retail/Online_Retail.csv'

online_rt = pd.read_csv(path, encoding = 'latin1')

online_rt.head()
```

Step 4. Create a histogram with the 10 countries that have the most 'Quantity' ordered except UK

```
# group by the Country
countries = online_rt.groupby('Country').sum()

# sort the value and get the first 10 after UK
countries = countries.sort_values(by = 'Quantity',ascending = False)[1:11]

# create the plot
countries['Quantity'].plot(kind='bar')

# Set the title and labels
plt.xlabel('Countries')
plt.ylabel('Quantity')
plt.title('10 Countries with most orders')

# show the plot
```

```
plt.show()
```

Step 5. Exclude negative Quantity entries

```
online_rt = online_rt[online_rt.Quantity > 0]
online_rt.head()
```

Step 6. Create a scatterplot with the Quantity per UnitPrice by CustomerID for the top 3 Countries (except UK)

```
# groupby CustomerID
customers = online_rt.groupby(['CustomerID','Country']).sum()

# there is an outlier with negative price
customers = customers[customers.UnitPrice > 0]

# get the value of the index and put in the column Country
customers['Country'] = customers.index.get_level_values(1)

# top three countries
top_countries =  ['Netherlands', 'EIRE', 'Germany']

# filter the dataframe to just select ones in the top_countries
customers = customers[customers['Country'].isin(top_countries)]

#################
# Graph Section #
#################

# creates the FaceGrid
g = sns.FacetGrid(customers, col="Country")

# map over a make a scatterplot
g.map(plt.scatter, "Quantity", "UnitPrice", alpha=1)

# adds legend
g.add_legend()
```

Step 7. Investigate why the previous results look so uninformative.

This section might seem a bit tedious to go through. But I've thought of it as some kind of a simulation of problems one might encounter when dealing with data and other people. Besides there is a prize at the end (i.e. Section 8).

(But feel free to jump right ahead into Section 8 if you want; it doesn't require that you finish this section.)

Step 7.1 Look at the first line of code in Step 6. And try to figure out if it leads to any kind of problem.

Step 7.1.1 Display the first few rows of that DataFrame.

```
#This takes our initial dataframe groups it primarily by 'CustomerID' and
secondarily by 'Country'.
#It sums all the (non-indexical) columns that have numerical values under
each group.
customers = online_rt.groupby(['CustomerID','Country']).sum().head()

#Here's what it looks like:
customers
```

Step 7.1.2 Think about what that piece of code does and display the dtype of UnitPrice

```
customers.UnitPrice.dtype
#So it's 'float64'
#But why did we sum 'UnitPrice', to begin with?
#If 'UnitPrice' wasn't something that we were interested in then it would be
OK
#since we wouldn't care whether UnitPrice was being summed or not.
#But we want our graphs to reflect 'UnitPrice'!
#Note that summing up 'UnitPrice' can be highly misleading.
#It doesn't tell us much as to what the customer is doing.
#Suppose, a customer places one order of 1000 items that are worth $1 each.
#Another customer places a thousand orders of 1 item worth $1.
#There isn't much of a difference between what the former and the latter
customers did.
#After all, they've spent the same amount of money.
#so we should be careful when we're summing columns. Sometimes we intend to
sum just one column
#('Quantity' in this case) and another column like UnitPrice gets ito the
mix.
```

Step 7.1.3 Pull data from online_rtfor CustomerIDs 12346.0 and 12347.0.

```
display(online_rt[online_rt.CustomerID == 12347.0].
        sort_values(by='UnitPrice', ascending = False).head())
display(online_rt[online_rt.CustomerID == 12346.0].
        sort_values(by='UnitPrice', ascending = False).head())
#The result is exactly what we'd suspected. Customer 12346.0 placed
#one giant order, whereas 12347.0 placed a lot of smaller orders.
#So we've identified one potential reason why our plots looked so weird at
section 6.
#At this stage we need to go back to the initial problem we've specified at
section 6.
```

Step 7.2 Reinterpreting the initial problem.

To reiterate the question that we were dealing with:
"Create a scatterplot with the Quantity per UnitPrice by CustomerID for the top 3 Countries"

The question is open to a set of different interpretations. We need to disambiguate.

We could do a single plot by looking at all the data from the top 3 countries. Or we could do one plot per country. To keep things consistent with the rest of the exercise, let's stick to the latter oprion. So that's settled.

But "top 3 countries" with respect to what? Two answers suggest themselves: Total sales volume (i.e. total quantity sold) or total sales (i.e. revenue). This exercise goes for sales volume, so let's stick to that.

Step 7.2.1 Find out the top 3 countries in terms of sales volume.

```
sales_volume =
online_rt.groupby('Country').Quantity.sum().sort_values(ascending=False)

top3 = sales_volume.index[1:4] #We are excluding UK
top3
```

Step 7.3 Modify, select and plot data

Step 7.3.1 Add a column to online_rt called Revenue calculate the revenue (Quantity * UnitPrice) from each sale.

We will use this later to figure out an average price per customer.

```
online_rt['Revenue'] = online_rt.Quantity * online_rt.UnitPrice
online_rt.head()



grouped =
online_rt[online_rt.Country.isin(top3)].groupby(['CustomerID','Country'])

plottable = grouped['Quantity','Revenue'].agg('sum')
#plottable = grouped[['Quantity','Revenue']].agg('sum')#This worked
instead

plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

# get the value of the index and put in the column Country
plottable['Country'] = plottable.index.get_level_values(1)
plottable.head()
```

Step 7.4 What to do now?

We aren't much better-off than what we started with. The data are still extremely scattered around and don't seem quite informative.

But we shouldn't despair! There are two things to realize:

1. The data seem to be skewed towaards the axes (e.g. we don't have any values where Quantity = 50000 and AvgPrice = 5). So that might suggest a trend.
2. We have more data! We've only been looking at the data from 3 different countries and they are plotted on different graphs.

So: we should plot the data regardless of Country and hopefully see a less scattered graph.

Step 7.4.1 Plot the data for each CustomerID on a single graph

```
grouped = online_rt.groupby(['CustomerID'])
```

```
plottable = grouped['Quantity','Revenue'].agg('sum')
#plottable = grouped[['Quantity','Revenue']].agg('sum') #This works
instead

plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

# map over a make a scatterplot
plt.scatter(plottable.Quantity, plottable.AvgPrice)
plt.plot()


#Turns out the graph is still extremely skewed towards the axes like an
exponential decay function.
```

Step 7.4.2 Zoom in so we can see that curve more clearly

```
grouped = online_rt.groupby(['CustomerID','Country'])
plottable = grouped.agg({'Quantity': 'sum',
                         'Revenue': 'sum'})
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

# map over a make a scatterplot
plt.scatter(plottable.Quantity, plottable.AvgPrice)

#Zooming in. (I'm starting the axes from a negative value so that
#the dots can be plotted in the graph completely.)
plt.xlim(-40,2000)
plt.ylim(-1,80)

plt.plot()


#And there is still that pattern, this time in close-up!
```

8. Plot a line chart showing revenue (y) per UnitPrice (x).

Did Step 7 give us any insights about the data? Sure! As average price increases, the quantity ordered decreses. But that's hardly surprising. It would be surprising if that wasn't the case!

Nevertheless the rate of drop in quantity is so drastic, it makes me wonder how our revenue changes with respect to item price. It would not be that surprising if it didn't change that much. But it would be interesting to know whether most of our revenue comes from expensive or inexpensive items, and how that relation looks like.

That is what we are going to do now.

8.1 Group UnitPrice by intervals of 1 for prices [0,50), and sum Quantity and Revenue.

```
#These are the values for the graph.
#They are used both in selecting data from
#the DataFrame and plotting the data so I've assigned
#them to variables to increase consistency and make things easier
```

```
#when playing with the variables.
price_start = 0
price_end = 50
price_interval = 1

#Creating the buckets to collect the data accordingly
buckets = np.arange(price_start,price_end,price_interval)

#Select the data and sum
revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice,
buckets)).Revenue.sum()
revenue_per_price.head()
```

8.3 Plot.

```
revenue_per_price.plot()
plt.xlabel('Unit Price (in intervals of '+str(price_interval)+')')
plt.ylabel('Revenue')
plt.show()
```

8.4 Make it look nicer.

x-axis needs values.
y-axis isn't that easy to read; show in terms of millions.

```
revenue_per_price.plot()

#Place labels
plt.xlabel('Unit Price (in buckets of '+str(price_interval)+')')
plt.ylabel('Revenue')

#Even though the data is bucketed in intervals of 1,
#I'll plot ticks a little bit further apart from each other to avoid
cluttering.
plt.xticks(np.arange(price_start,price_end,3),
           np.arange(price_start,price_end,3))
plt.yticks([0, 500000, 1000000, 1500000, 2000000, 2500000],
           ['0', '$0.5M', '$1M', '$1.5M', '$2M', '$2.5M'])
plt.show()

#Looks like a major chunk of our revenue comes from items worth $0-$3!
```

07 VISUALIZATION

SCORES

Step 1. Import the necessary libraries

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

%matplotlib inline
```

Step 2. Create the DataFrame it should look like below.

```
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
            'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
            'female': [0, 1, 1, 0, 1],
            'age': [42, 52, 36, 24, 73],
            'preTestScore': [4, 24, 31, 2, 3],
            'postTestScore': [25, 94, 57, 62, 70]}

df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age',
'female', 'preTestScore', 'postTestScore'])

df
```

Step 3. Create a Scatterplot of preTestScore and postTestScore, with the size of each point determined by age

Hint: Don't forget to place the labels

```
plt.scatter(df.preTestScore, df.postTestScore, s=df.age)

#set labels and titles
plt.title("preTestScore x postTestScore")
plt.xlabel('preTestScore')
plt.ylabel('preTestScore')
```

Step 4. Create a Scatterplot of preTestScore and postTestScore.

This time the size should be 4.5 times the postTestScore and the color determined by sex

```
plt.scatter(df.preTestScore, df.postTestScore, s= df.postTestScore * 4.5, c =
df.female)

#set labels and titles
plt.title("preTestScore x postTestScore")
plt.xlabel('preTestScore')
plt.ylabel('preTestScore')
```

07 VISUALIZATION

TIPS

Step 1. Import the necessary libraries:

```
import pandas as pd

# visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns


# print the graphs in the notebook
% matplotlib inline

# set seaborn style to white
sns.set_style("white")
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called tips

```
url =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/07_Visu
alization/Tips/tips.csv'
tips = pd.read_csv(url)

tips.head()
```

Step 4. Delete the Unnamed 0 column

```
del tips['Unnamed: 0']

tips.head()
```

Step 5. Plot the total_bill column histogram

```
# create histogram
ttbill = sns.distplot(tips.total_bill);

# set lables and titles
ttbill.set(xlabel = 'Value', ylabel = 'Frequency', title = "Total Bill")

# take out the right and upper borders
sns.despine()
```

Step 6. Create a scatter plot presenting the relationship between total_bill and tip

```
sns.jointplot(x ="total_bill", y ="tip", data = tips)
```

Step 7. Create one image with the relationship of total_bill, tip and size.

Hint: It is just one function.

```
sns.pairplot(tips)
```

Step 8. Present the relationship between days and total_bill value

```
sns.stripplot(x = "day", y = "total_bill", data = tips, jitter = True);
```

Step 9. Create a scatter plot with the day as the y-axis and tip as the x-axis, differ the dots by sex

```
sns.stripplot(x = "tip", y = "day", hue = "sex", data = tips, jitter = True);
```

Step 10. Create a box plot presenting the total_bill per day differetiation the time (Dinner or Lunch)

```
sns.boxplot(x = "day", y = "total_bill", hue = "time", data = tips);
```

Step 11. Create two histograms of the tip value based for Dinner and Lunch. They must be side by side.

```
# better seaborn style
sns.set(style = "ticks")

# creates FacetGrid
g = sns.FacetGrid(tips, col = "time")
g.map(plt.hist, "tip");
```

Step 12. Create two scatterplots graphs, one for Male and another for Female, presenting the total_bill value and tip relationship, differing by smoker or no smoker

They must be side by side.

```
g = sns.FacetGrid(tips, col = "sex", hue = "smoker")
g.map(plt.scatter, "total_bill", "tip", alpha =.7)

g.add_legend();
```

07 VISUALIZATION

VISUALIZING THE TITANIC DISASTER

Step 1. Import the necessary libraries

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

%matplotlib inline
```

Step 2. Import the dataset from this address

Step 3. Assign it to a variable titanic

```
url =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/Visuali
zation/Titanic_Desaster/train.csv'

titanic = pd.read_csv(url)

titanic.head()
```

Step 4. Set PassengerId as the index

```
titanic.set_index('PassengerId').head()
```

Step 5. Create a pie chart presenting the male/female proportion

```
# sum the instances of males and females
males = (titanic['Sex'] == 'male').sum()
females = (titanic['Sex'] == 'female').sum()

# put them into a list called proportions
proportions = [males, females]

# Create a pie chart
plt.pie(
    # using proportions
    proportions,

    # with the labels being officer names
    labels = ['Males', 'Females'],
    # with no shadows
    shadow = False,

    # with colors
    colors = ['blue','red'],

    # with one slide exploded out
    explode = (0.15 , 0),

    # with the start angle at 90%
    startangle = 90,

    # with the percent listed as a fraction
    autopct = '%1.1f%%'
    )

# View the plot drop above
plt.axis('equal')

# Set labels
plt.title("Sex Proportion")

# View the plot
plt.tight_layout()
plt.show()
```

Step 6. Create a scatterplot with the Fare payed and the Age, differ the plot color by gender

```
# creates the plot using
lm = sns.lmplot(x = 'Age', y = 'Fare', data = titanic, hue = 'Sex',
fit_reg=False)

# set title
lm.set(title = 'Fare x Age')

# get the axes object and tweak it
axes = lm.axes
axes[0,0].set_ylim(-5,)
axes[0,0].set_xlim(-5,85)
```

Step 7. How many people survived?

```
titanic.Survived.sum()
```

Step 8. Create a histogram with the Fare payed

```
# sort the values from the top to the least value and slice the first 5 items
df = titanic.Fare.sort_values(ascending = False)
df

# create bins interval using numpy
binsVal = np.arange(0,600,10)
binsVal

# create the plot
plt.hist(df, bins = binsVal)

# Set the title and labels
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.title('Fare Payed Histrogram')

# show the plot
plt.show()
```

08 CREATING SERIES AND DATAFRAMES

POKEMON

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Create a data dictionary

```
raw_data = {"name": ['Bulbasaur', 'Charmander','Squirtle','Caterpie'],
            "evolution": ['Ivysaur','Charmeleon','Wartortle','Metapod'],
            "type": ['grass', 'fire', 'water', 'bug'],
            "hp": [45, 39, 44, 45],
            "pokedex": ['yes', 'no','yes','no']
            }
```

Step 3. Assign it to a variable called pokemon

```
pokemon = pd.DataFrame(raw_data)
pokemon.head()
```

Step 4. Ops...it seems the DataFrame columns are in alphabetical order. Place the order of the columns as name, type, hp, evolution, pokedex

```
pokemon = pokemon[['name', 'type', 'hp', 'evolution','pokedex']]
pokemon
```

Step 5. Add another column called place, and insert what you have in mind.

```
pokemon['place'] = ['park','street','lake','forest']
pokemon
```

Step 6. Present the type of each column

```
pokemon.dtypes
```

09 TIME SERIES

APPLE STOCK

Introduction:

We are going to use Apple's stock price.

Step 1. Import the necessary libraries

In [4]:

```
import pandas as pd
import numpy as np

# visualization
import matplotlib.pyplot as plt

%matplotlib inline
```

Step 2. Import the dataset from this address

Step 3. Assign it to a variable apple

In [32]:

```
url =
'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/09_Time
_Series/Apple_Stock/appl_1980_2014.csv'
apple = pd.read_csv(url)

apple.head()
```

Step 4. Check out the type of the columns

In [33]:

```
apple.dtypes
```

Step 5. Transform the Date column as a datetime type

In [34]:

```
apple.Date = pd.to_datetime(apple.Date)

apple['Date'].head()
```

Step 6. Set the date as the index

In [35]:

```
apple = apple.set_index('Date')

apple.head()
```

Step 7. Is there any duplicate dates?

In [36]:

```
# NO! All are unique
apple.index.is_unique
```

Step 8. Ops...it seems the index is from the most recent date. Make the first entry the oldest date.

In [39]:

```
apple.sort_index(ascending = True).head()
```

Step 9. Get the last business day of each month

In [48]:

```
apple_month = apple.resample('BME').mean()

apple_month.head()
```

Step 10. What is the difference in days between the first day and the oldest

In [65]:

```
(apple.index.max() - apple.index.min()).days
```

Step 11. How many months in the data we have?

In [66]:

```
apple_months = apple.resample('BM').mean()
```

```
len(apple_months.index)
```

Step 12. Plot the 'Adj Close' value. Set the size of the figure to 13.5 x 9 inches

In [81]:

```
# makes the plot and assign it to a variable
appl_open = apple['Adj Close'].plot(title = "Apple Stock")

# changes the size of the graph
fig = appl_open.get_figure()
fig.set_size_inches(13.5, 9)
```