Python Coding Cheat-Sheet

Made by Alex Rudyak
Code Blocks theme - tomorrow-night-blue

General

Make a copy of a list (unlinked)

```
listA = [0, 1, 2, 3]
listB = listA[:]
```

Reverse an Integer

4562 → 2654

```
n = 4562
rev = 0

while(n > 0):
    a = n % 10
    rev = rev * 10 + a
    n = n // 10

print(rev)
```

Reverse a String

```
text = "hello, world"
reversed_text = text[::-1]
```

Combine two lists into a dictionary

```
test_keys = []
test_values = []
dic = {}
for key in test_keys:
    for value in test_values:
        dic[key] = value
        test_values.remove(value)
        break
print (dic)
```

Check whether a char is int

```
string = "avs12ssd"
for char in string:
    if char.isnumeric():
      # char is an integer
```

Output list elements as string with spaces

[1, hi, 2, four] → “1 hi 2 four”

```
list = [1, hi, 2, four]
string = " ".join(list)
print (string) # "1 hi 2 four"
```

Convert int to hexadecimal including negatives

```
num = 10
hexaNum = format((num + (1<<32)) % (1<<32),'x') # 1<<32 is a bit wise operation to
move the bit "1" 32 steps to The left.
print (hexaNum) # A
```

Find longest common prefix in a list of strings

[“flower”,”flow”,”flight] → “fl”

```
strs = ["flower","flow","flight"]
if not strs: return ""
        prefix = strs[0]
        for i in range(len(strs)):
            while strs[i].find(prefix): # find the first index of prefix
                prefix = prefix[0:len(prefix)-1]
                if not prefix:
                    return ""
        return prefix
```

Iterate through index and value using enumerate

```
doc_list = "Adventures of Arch","Bamby","Princess and the pea"
# Iterate through the indices (i) and elements (doc) of documents
```

```
    for i, doc in enumerate(doc_list):
           #  i = 0, doc = "Adventures of Arch"
           #  i = 1, doc = "Bamby"
           #  i = 2, doc = "Princess and the pea"
```

Check whether input string is a valid IP

```
def is_valid_IP(strng):
    ip_cells = strng.split(".") # split the string to 4 cells like ipv4
    if len(ip_cells) == 4: # if the length is not 4 its not a valid ip
        for cell in ip_cells:
            if cell.isnumeric(): # if the cell is not numeric its not a valid ip
                if 0 <= int(cell) <= 255: # the values should be between 0 and 255
                    state = True
                else:
                    return False
            else:
                return False
            if len(cell) == 3: # if its a 3 digit number and starts with 0 its not
valid (although I think it is)
                if cell.startswith("0"):
                    return False
        return state
    else:
        return False
```

Find Square Root - Bi-section Search

```
x = 25
epsilon = 0.001
numGuess = 0
low = 1.0
high = x
guess = (high + low)/2.0

while abs(guess**2 - x) >= epsilon:
    print('low = ' + str(low) + ' high = ' + str(high) + ' | Our guess: ' +
str(guess))
    numGuess += 1
    if guess**2 < x:
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0

print('Number of Guesses = ' + str(numGuess))
print(str(guess) + ' is Close to square root of ' + str(x))
```

Guess the number game - Bi-section search

```
print("Please think of a number between 0 and 100!")
high = 100
low = 0
first_guess = (high - low)//2
user_input = ""
print("Is your secret number " + str(first_guess) + " ?")

while user_input != 'c':

    user_input = input("Enter 'h' to indicate the guess is too high. Enter 'l' to
indicate the guess is too low. Enter 'c' to indicate I guessed correctly. ")

    if user_input == 'h':
        high = first_guess
    elif user_input == 'l':
        low = first_guess
    elif user_input == 'c':
        print("Game over. Your secret number was: " + str(first_guess))
    else:
        print("Sorry, I did not understand your input.")

    first_guess = (high + low)//2

    if user_input != 'c':
        print("Is your secret number " + str(first_guess) + " ?")
```

Greatest Common Divisor - Loop

```
def gcdIter(a, b):
    '''
    a, b: positive integers

    returns: a positive integer, the greatest common divisor of a & b.
    '''
```

```python
    tmp = a if a<b else b
    while tmp > 0:
        if ((a%tmp == 0) and (b%tmp == 0)):
            return tmp
        tmp -= 1
```

Greatest Common Divisor - Recursion

```python
def gcdRecur(a, b):
    '''
    a, b: positive integers

    returns: a positive integer, the greatest common divisor of a & b.
    '''
    if b == 0:
        return a
    else:
        return gcdRecur(b, a%b)
```

Check whether a String is a Palindrome - Recursion

```python
def isPal(s):
    if len(s) < 1:
        return True
    else:
        return s[0] == s[-1] and isPal(s[1:-1])
```

Print every other element in a tuple to another tuple

```python
def oddTuples(aTup):
    '''
    aTup: a tuple

    returns: tuple, every other element of aTup.
    '''
    new_t = ()
    for num,t in enumerate(aTup):
        if num%2 == 0:
            new_t += (t,)
    return new_t
```

```
print(oddTuples(('I', 'am', 'a', 'test', 'tuple')))
```

Matplotlib

Pandas - Data manipulation

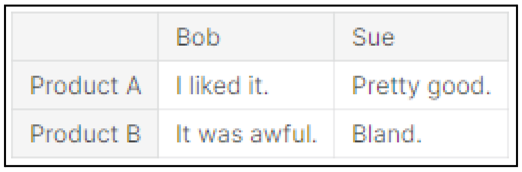
Importing the library

```
import pandas as pd
```

Create a DataFrame (Table)

```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], # column 1
              'Sue': ['Pretty good.', 'Bland.']}, # column 2
             index = ['Product A', 'Product B']) # index (row names)
```

We get:

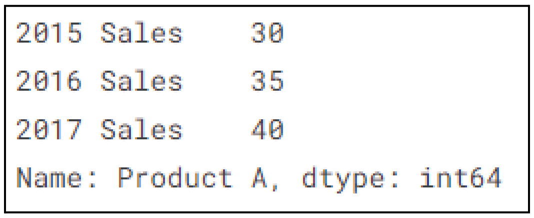


	Bob	Sue
Product A	I liked it.	Pretty good.
Product B	It was awful.	Bland.

Create a Series (Single column of a Table)

```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'],
name='Product A')
```

We get:



```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

Import data from a csv

Given a location with a csv file, import it into our workspace

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

Change the datatype of a column

Changes the data type of column "*points*" into a *float64* type.

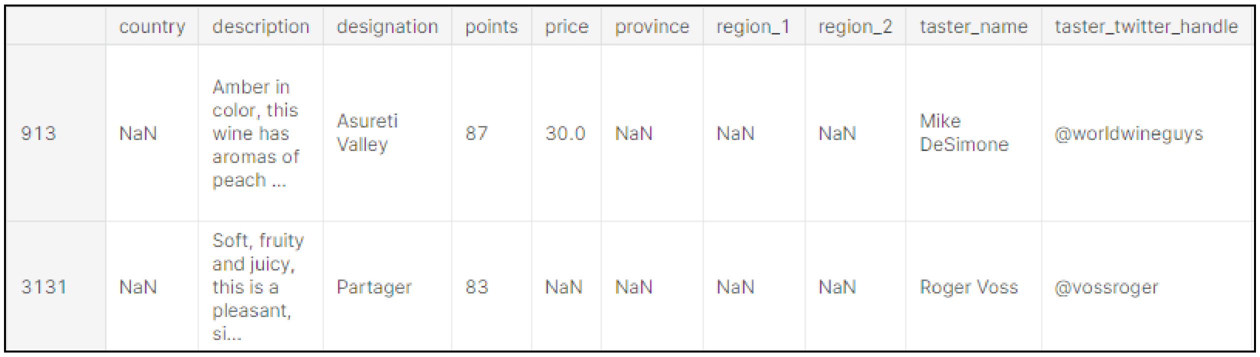
```
reviews.points.astype('float64')
```

Check whether we have missing data in our table

Given table "*Reviews*" get all the rows of the NaN values in "*Country*".

```
reviews[pd.isnull(reviews.country)]
```

We get:



	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle
913	NaN	Amber in color, this wine has aromas of peach ...	Asureti Valley	87	30.0	NaN	NaN	NaN	Mike DeSimone	@worldwineguys
3131	NaN	Soft, fruity and juicy, this is a pleasant, si...	Partager	83	NaN	NaN	NaN	NaN	Roger Voss	@vossroger

Change column name in a table

Given a table "*Reviews*" with a column named "*points*", change its name to "*score*".

```
reviews.rename(columns={'points': 'score'})
```

Change Index names (row and column)

Given table "*Reviews*" change its rows index name to "*wines*" and column index name to "*fields*".

```
reviews.rename_axis("wines", axis='rows').rename_axis("fields", axis='columns')
```

Concatenate two tables together

Given two tables "*canadian_youtube*" and "*british_youtube*" concatenate them side by side, as in "stick" them together basically adding more rows while keeping the same columns.

```
canadian_youtube = pd.read_csv("../input/youtube-new/CAvideos.csv")
british_youtube = pd.read_csv("../input/youtube-new/GBvideos.csv")
pd.concat([canadian_youtube, british_youtube])
```

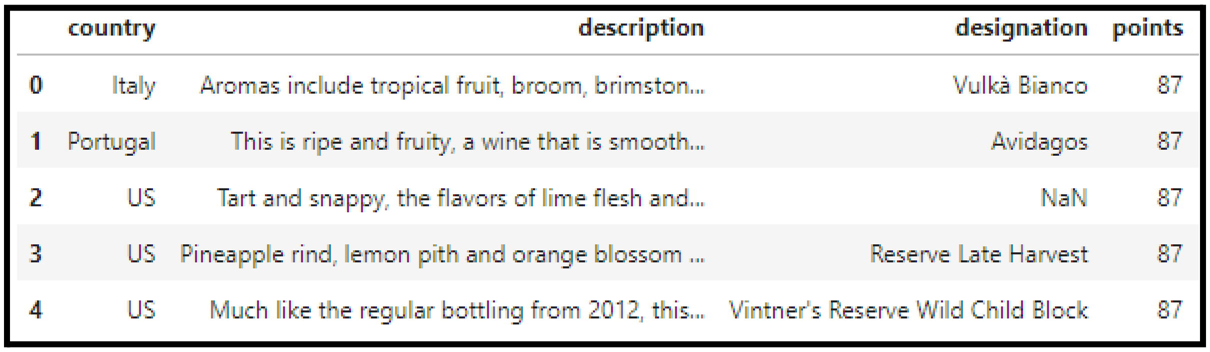
Merge two table together that has the same column

Given two tables "*Powerlifting_meets*" and "*Powerlifting_competitors*" that have the same column "*MeetID*" we can merge that together based on the MeetID index.
To make sure every column in the merged table has unique names we add suffixes to each column of the original tables, "*_meets*" for the first table and "*_competitors*" for the second.

```
left = powerlifting_meets.set_index(["MeetID"])
right = powerlifting_competitors.set_index(["MeetID"])
powerlifting_combined = left.join(right, lsuffix='_meets',
rsuffix='_competitors')
```

Count how many times have a word came up in a table column

Given a table "*Reviews*", find how many times have the words "*fruity*" and "*tropical*" appear in the description column.



	country	description	designation	points
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87

```
n_trop = reviews.description.map(lambda desc: "tropical" in desc).sum() # tropical
word frequency

n_fruity = reviews.description.map(lambda desc: "fruity" in desc).sum() # fruity
```

```
word frequency

descriptor_counts = pd.Series([n_trop, n_fruity], index=['tropical', 'fruity'])
```

Apply a function on every column in a table

Given a table “*Reviews*” with a variable “*country*” and “*points*” in it, change every column based on the criteria in the function.

```
def stars(row):
    if row.country == 'Canada':
        return 3
    elif row.points >= 95:
        return 3
    elif row.points >= 85:
        return 2
    else:
        return 1

star_ratings = reviews.apply(stars, axis='columns')
```

Find values from a table based on grouping in the table

Given table “*Reviews*”, we slice the table by “points” and take the minimum price.

```
reviews.groupby('points').price.min()
```

Sort table by particular parameter

Given table “*Countries_reviewed*”, sort the table by parameter “*len*”.

This will keep the original Indexes!

Index	Name
1	Bob
2	Alex

sort_values(by = ‘name’)
→

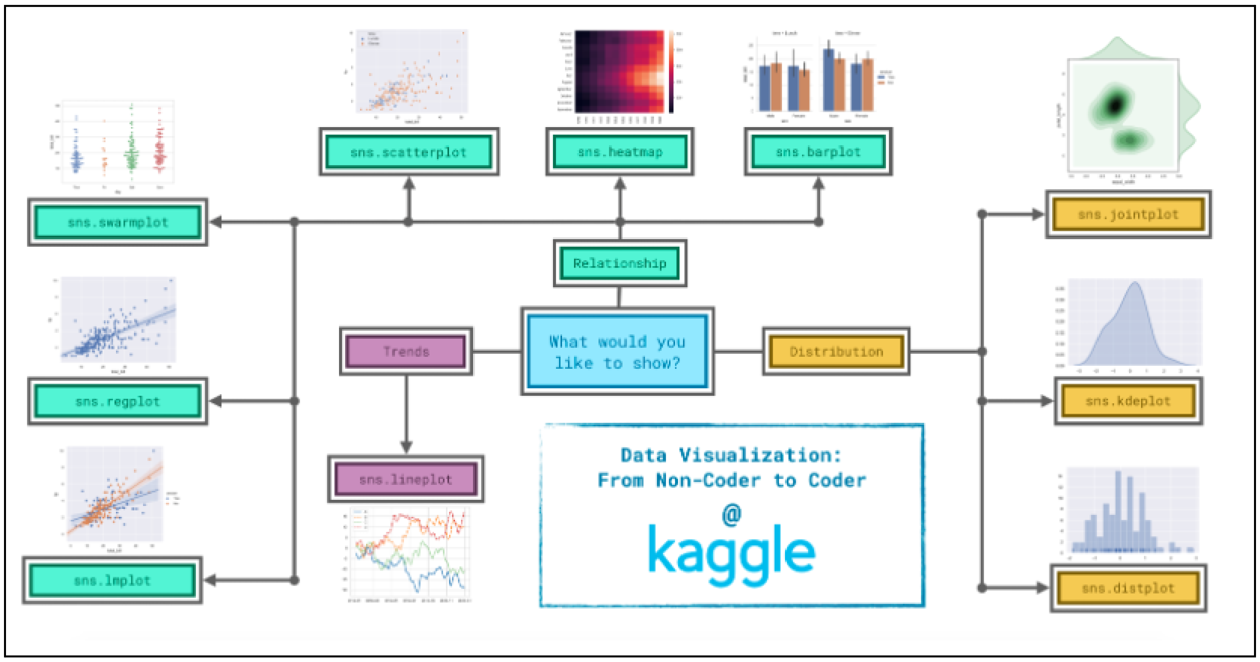
Index	Name
2	Alex
1	Bob

```
countries_reviewed.sort_values(by = 'len')
```

The default order is ascension, starting from the smallest number at [0] and largest number at [end]. To change that we can add the “ascending = False” key.

```
countries_reviewed.sort_values(by = 'len', ascending = False)
```

Seaborn - Data needs visualizing as well



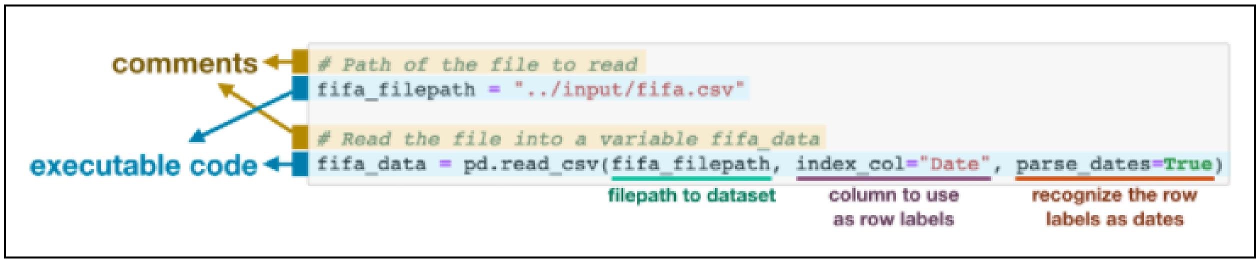
Importing the library

```
import seaborn as sns
```

To import the actual data we will use **Pandas**, the library discussed here as well.

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

Explained as:



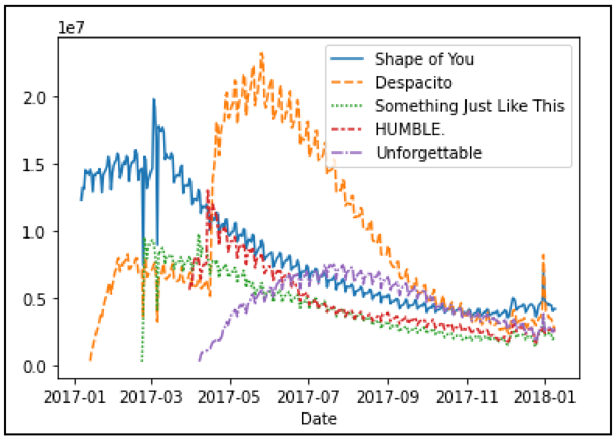
```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

comments
executable code
filepath to dataset
column to use as row labels
recognize the row labels as dates

Plot a line chart

Given CSV data "*spotify_data*" we can plot it using the following command, it will automatically use the table's left column as **X axis** and **Y axis** as the **values of the cell they are in.**
The table's upper index is used as a **Legend**.

```
# Line plot
sns.lineplot(data=spotify_data)
```

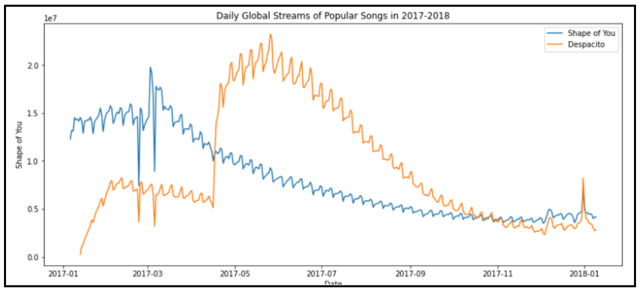


Changing the parameters of the plots

```
# Set the width and height of the figure
plt.figure(figsize=(14,6))
# Add title
plt.title("Daily Global Streams of Popular Songs in 2017-2018")
# Add label for horizontal axis
plt.xlabel("Date")
# Add label for vertical axis
plt.ylabel("Arrival delay (in minutes)")
# Add a Legend to the data
plt.legend()
# Change the style of the figure to the "dark" theme
sns.set_style("dark") # darkgrid || whitegrid || dark || white || ticks
```

Draw a line plot of only one column of data from a table

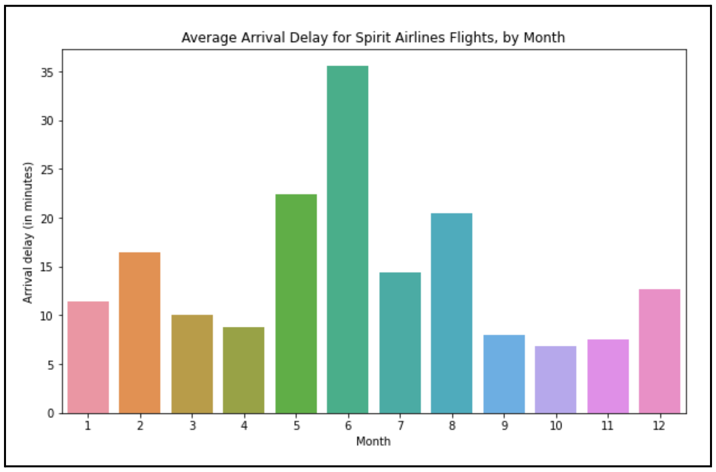
```
# Line chart showing only one column of the table
sns.lineplot(data=spotify_data['Shape of You'], label="Shape of You")
```



Plot a bar chart

Given table "*flight_data*" use the left index of the table as **X axis** and the **Y axis** plot values that **correspond to those indexes and their height.**

```
# Bar chart showing a single column in a table
sns.barplot(x=flight_data.index, y=flight_data['NK'])
```

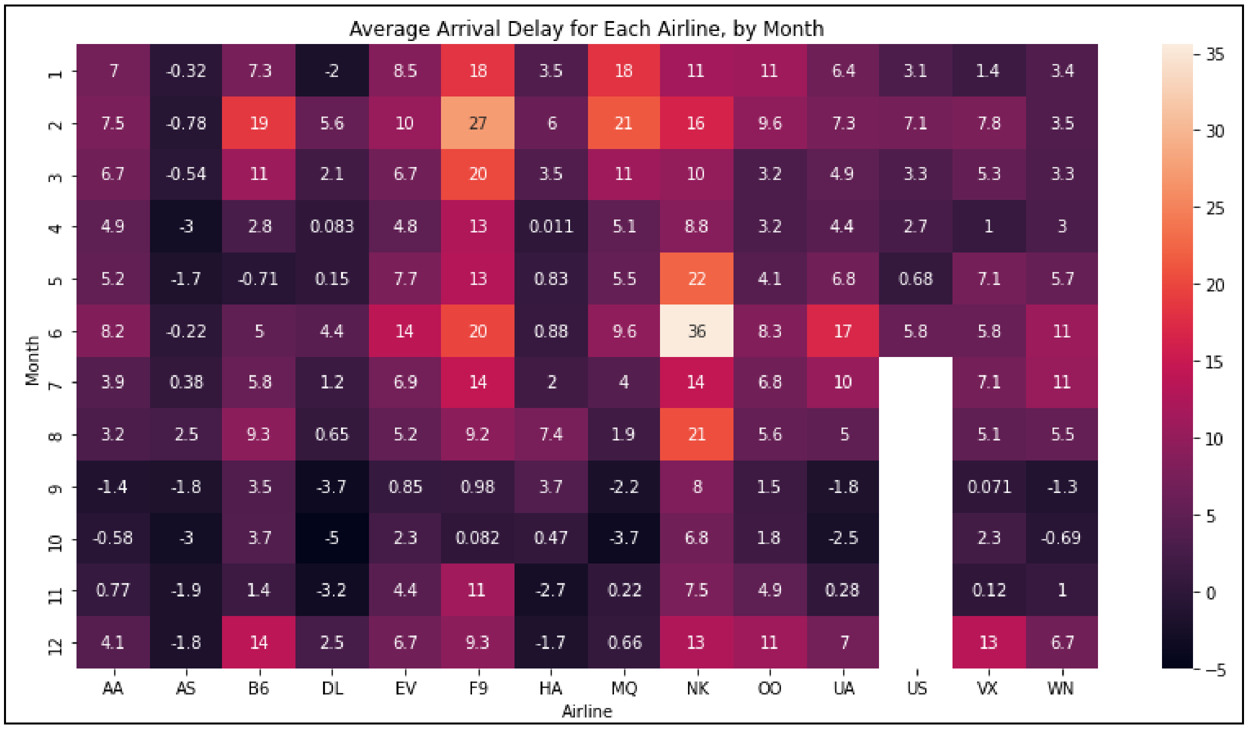


#This_Color_Pallete_is_beautiful.

Plot a heatmap (Spectrogram?!)

Given a table "*flight_data*" plot a heatmap of the data. This also resembles a **Spectrogram.** Basically turning the whole table into a color book.

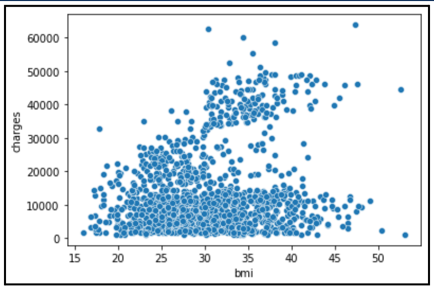
```
# Heatmap showing average arrival delay for each airline by month
sns.heatmap(data=flight_data, annot=True)
```



Plot a scatter plot

Given table "*insurance_data*", plot its "*bmi*" column as **X axis** and "*charges*" column as **Y axis.**

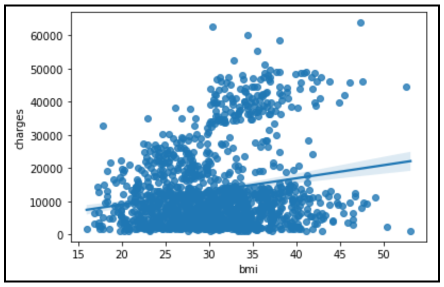
```
sns.scatterplot(x = insurance_data['bmi'], y = insurance_data['charges'])
```



Plot a scatter plot with a regression line

Changing "*scatterplot*" to "regplot" adds a regression line to the scatter plot.

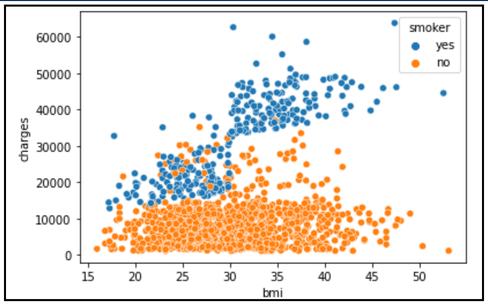
```
sns.regplot(x = insurance_data['bmi'], y = insurance_data['charges'])
```



Color the data set

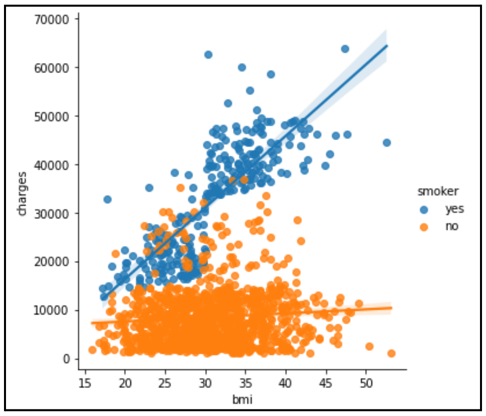
You can color the data set based on column values, to do so we add the "*hue*" keyword. Because the "*smoker*" column has only 2 values, we get only 2 colors.

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'],
hue=insurance_data['smoker'])
```



Adding regression lines based on sub-data

```
sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
```

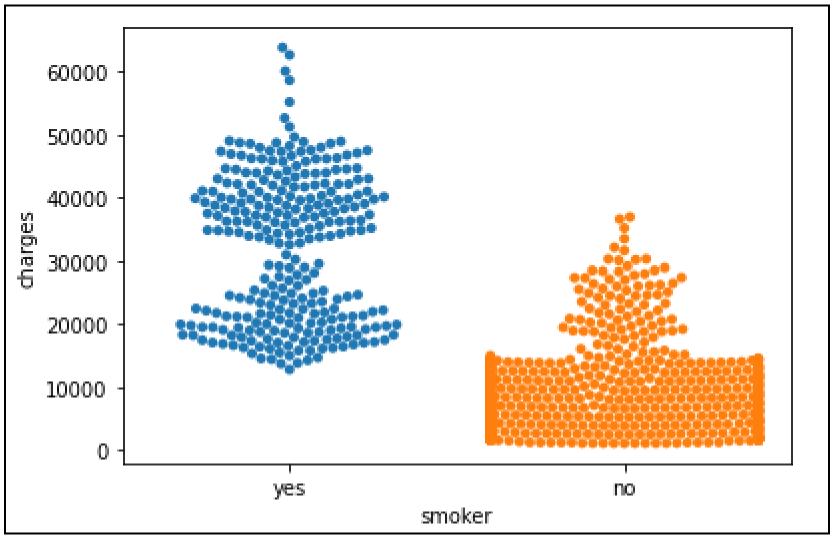


Plot a categorical scatter plot - swarmplot

Instead of **continuous** data as the **X axis** we use **discrete data** ('1' or '0') to plot.

The result is a little bit weird but we can get that the max charges are people who got “yes” (‘1’) and minimum charges are people that got “no” (‘0’).

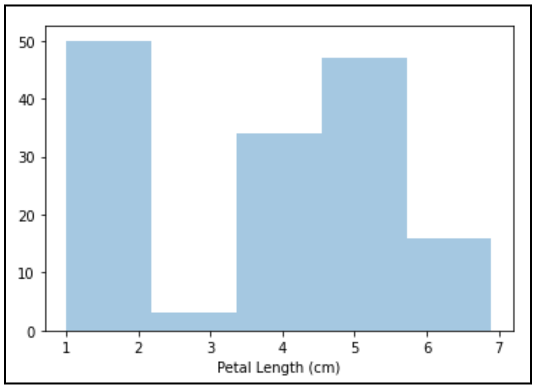
```
sns.swarmplot(x=insurance_data['smoker'], y=insurance_data['charges'])
```



Plot a Histogram

Given table “*iris_data*” plot a histogram with column “*Petal length (cm)*” as **X axis.**

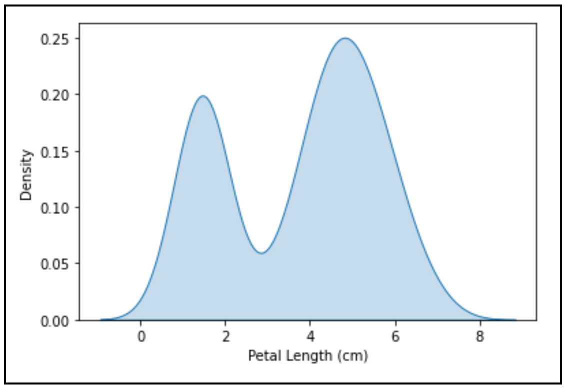
```
# Histogram
sns.distplot(a=iris_data['Petal Length (cm)'], kde=False)
```



Plot a Density plot (KDE)

Given table *“iris_data”*, plot column *“Petal Length (cm)”* as **X axis**.
To color the underneath graph we add *“shade”* keyword.

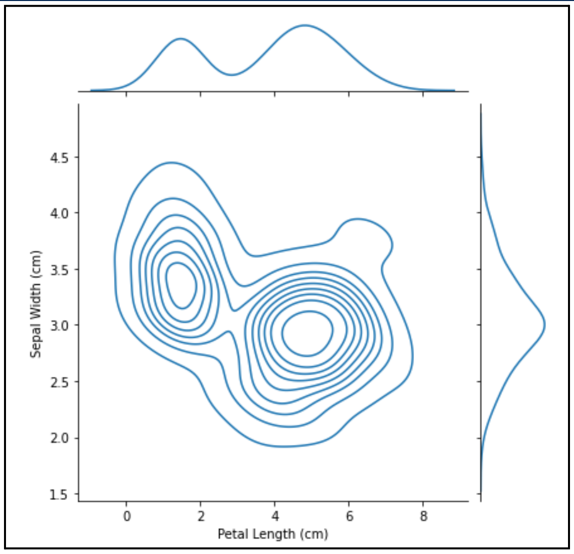
```
# KDE plot
sns.kdeplot(data=iris_data['Petal Length (cm)'], shade=True)
```



Plot a 2D KDE plot

Given table *“iris_data”,* plot column *“Petal Length (cm)”* as a **KDE plot** on the **X axis**, and column *“Sepal Width (cm)”* as a **KDE plot** on the **Y axis**.
Combine (join) them on a single plot.

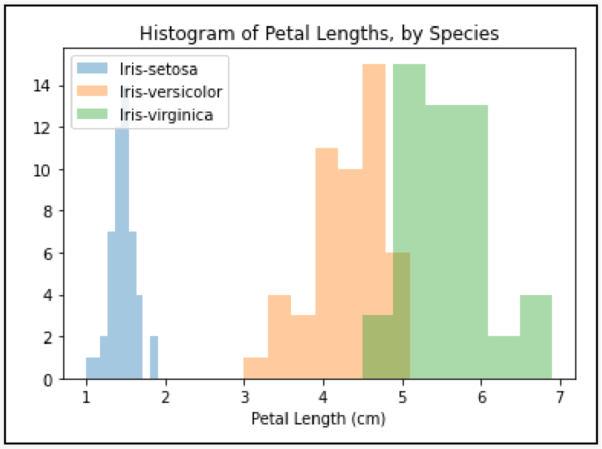
```
sns.jointplot(x=iris_data['Petal Length (cm)'], y=iris_data['Sepal Width (cm)'],
kind="kde")
```



Adding two or more histograms on one plot

Simply calling the histogram function again will add another plot to the existing one.

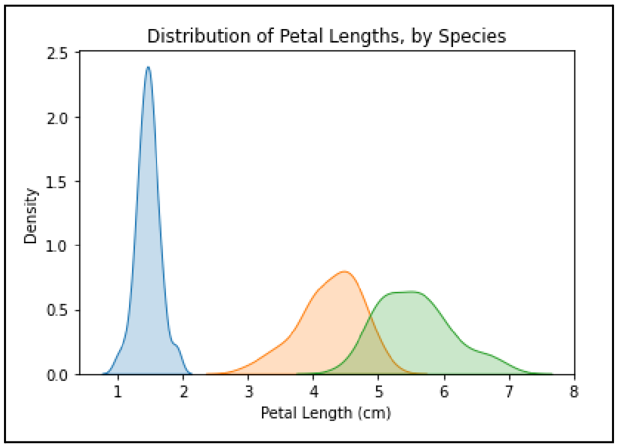
```
sns.distplot(a=iris_set_data['Petal Length (cm)'], label="Iris-setosa", kde=False)
sns.distplot(a=iris_ver_data['Petal Length(cm)'],label="Iris-versicolor",kde=False)
sns.distplot(a=iris_vir_data['Petal Length (cm)'],label="Iris-virginica",kde=False)
```



Adding two or more KDE on one plot

Simply calling the function again adds another plot to the figure.

```
sns.kdeplot(data=iris_set_data['PetalLength(cm)'],label="Iris-setosa",shade=True)
sns.kdeplot(data=iris_ver_data['PetalLength(cm)'],label="Iris-versicolor",shade=Tru
e)
sns.kdeplot(data=iris_vir_data['PetalLength(cm)'],label="Iris-virginica",shade=True
)
```



MITx 6.00.1x

Problem Set 1

Problem 1 - Vowel Count

Assume `s` is a string of lower case characters.

Write a program that counts up the number of vowels contained in the string `s`. Valid vowels are: '`a`', '`e`', '`i`', '`o`', and '`u`'. For example, if `s = 'azcbobobegghakl'`, your program should print:

```
Number of vowels: 5
```

```
vowel_cnt = 0
for char in s: # iterate over chars in sentence
    if char =='a' or char =='e' or char =='i' or char =='o' or char =='u':
        vowel_cnt += 1
print("Number of vowels: ", vowel_cnt)
```

Can make it shorter with:

```
vowl_string = "aeiou"
If char in vowl_string
```

Problem 2 - String in String (with dupe)

Assume `s` is a string of lower case characters.

Write a program that prints the number of times the string `'bob'` occurs in `s`. For example, if `s = 'azcbobobegghakl'`, then your program should print

```
Number of times bob occurs is: 2
```

```python
cnt = 0
for i in range(3, len(s)+1):
    temp = s[i-3:i]
    if temp == 'bob':
        cnt += 1
print("Number of times bob occurs is: ",cnt)
```

Problem 3 - Longest Substring Alphabetically

Assume `s` is a string of lower case characters.

Write a program that prints the longest substring of `s` in which the letters occur in alphabetical order. For example, if `s = 'azcbobobegghakl'`, then your program should print

```
Longest substring in alphabetical order is: beggh
```

In the case of ties, print the first substring. For example, if `s = 'abcbcd'`, then your program should print

```
Longest substring in alphabetical order is: abc
```

```
prefix = s[0]
alphabet = "abcdefghijklmnopqrstuvwxyz"
output = ""

for i in range(len(s)-1):
    if alphabet.find(s[i]) <= alphabet.find(s[i+1]): # find the first index
in a string
# can be replaced with | if s[i] <= s[i+1] because python automatically
measures strings by their ascii but was kept for readability
        prefix += s[i+1]
    elif len(output) < len(prefix):
        output = prefix
        prefix = s[i+1]
    else:
        prefix = s[i+1]

if len(output) < len(prefix):
    output = prefix

# go into "if" only if output is empty
if not output:
    output = prefix

print(output)
```

Problem set 2

Problem 1

Write a program to calculate the credit card balance after one year if a person only pays the minimum monthly payment required by the credit card company each month.The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal
3. `monthlyPaymentRate` - minimum monthly payment rate as a decimal

For each month, calculate statements on the monthly payment and remaining balance. At the end of 12 months, print out the remaining balance.

Be sure to print out no more than two decimal digits of accuracy - so print

```
Remaining balance: 813.41
```

instead of

```
Remaining balance: 813.4141998135
```

So your program only prints out one thing: the remaining balance at the end of the year in the format:

```
Remaining balance: 4784.0
```

A summary of the required math is found below:

Monthly interest rate= (Annual interest rate) / 12.0

Minimum monthly payment = (Minimum monthly payment rate) x (Previous balance)

Monthly unpaid balance = (Previous balance) - (Minimum monthly payment)

Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

```
def endOfYearBalance(balance,annualInterestRate,monthlyPaymentRate):
    updatedBalanceEachMonth = balance
    monthlyInterestRate = annualInterestRate/12.0
    for i in range (12):
        minimumMonthlyPayment = monthlyPaymentRate*updatedBalanceEachMonth
        monthlyUnpaidBalance = updatedBalanceEachMonth - minimumMonthlyPayment
        updatedBalanceEachMonth = monthlyUnpaidBalance +
(monthlyInterestRate*monthlyUnpaidBalance)
```

```
    print("Remaining balance: {}".format(round(updatedBalanceEachMonth,2)))

'''
balance = 42
annualInterestRate = 0.2
monthlyPaymentRate = 0.04
'''
endOfYearBalance(42,0.2,0.04)
'''
balance = 484
annualInterestRate = 0.2
monthlyPaymentRate = 0.04
'''
endOfYearBalance(484,0.2,0.04)
```

Problem 2

Now write a program that calculates the minimum **fixed** monthly payment needed in order to pay off a credit card balance within 12 months. By a fixed monthly payment, we mean a single number which does not change each month, but instead is a constant amount that will be paid each month.
In this problem, we will *not* be dealing with a minimum monthly payment rate.
The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal

The program should print out one line: the lowest monthly payment that will pay off all debt in under 1 year, for example:
`Lowest Payment: 180`

Assume that the interest is compounded monthly according to the balance at the end of the month (after the payment for that month is made). The monthly payment must be a multiple of $10 and is the same for all months. Notice that it is possible for the balance to become negative using this payment scheme, which is okay. A summary of the required math is found below:

Monthly interest rate = (Annual interest rate) / 12.0
Monthly unpaid balance = (Previous balance) - (Minimum fixed monthly payment)
Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

```
monthlyInterestRate = annualInterestRate/12
minimum_fixed = 0
```

```
while balance > 0:
   new_balance = balance
   for i in range(12):
       monthly_unpaid = new_balance - minimum_fixed
       new_balance = monthly_unpaid + monthlyInterestRate*monthly_unpaid
       if new_balance < 0:
           break
   minimum_fixed += 10
   if new_balance < 0:
       break

print(minimum_fixed - 10)
```

Debugging my code

Input method

If you put “*input*” at the end of an “*If*” statement or a “*For*” loop, it would **pause the iteration** until you press spacebar.
Useful if you have problems with your code and you don't get your desired output.

```
input()
```

Print method

Adding “*print*” at the end of an “*If*” statement or a “*For*” loop, will print the variable and help you debug the code.

```
print('output:', output)
```