--Guessing game with Python

```
name = input('Enter your full name: ')
print('Welcome, ' + name + ','+' lets play a guessing game')
print('You will get 3 chances to guess a number between 0 to 10, can you guess it right?')
secret_number = 9
guess_count=0
guess_limit=3
while guess_count<guess_limit:
    guess = int(input('Guess the number: '))
    guess_count+=1
    if guess==secret_number:
        print('Congratulations, you guessed it right, YOU WON!!')
        break
    else:
        print("Wrong guess, try again")
else:
    print('Sorry, you could not guess the number. Goodbye')
```

--Messing with car

```
command=""
print("Let's mess with you car abit, type help for full options")
started = False
while True:
    command = input("> ").lower()
    if command == "start":
        if started:
```

```
                print("Car is already started")
            else:
                started = True
                print("Car started, ready to go ")
        elif command=="stop":
            if not started:
                print("Car is already stopped, cannot stop a stopped car")
            else:
                started = False
                print("Car stopped")
        elif command=="help":
            print("""
start - to start the car
stop - to stop the car
quit - to quit
                """)
        elif command=="quit":
            print("Bye, have a nice day")
            break
        else:
            print("Sorry, I don't understand that")
```

--Finding the highest number among the numbers entered

```
print('I will identify the highest number among the numbers you enter')
number = input('Please enter numbers sepated by comma: ').split(',')
number1 = [eval(i) for i in number]
max = number1[0]
for number in number1:
```

```
    if number>max:
        max=number
print("The highest number is: "+str(max))
```

--Phone number in words

```
print("Ever wondered how your phone number will look like in words :)")
phone= input("Enter your phone: ")
digit_mapping={"1":"One", "2": "Two", "3": "Three","4":"Four","5":"Five",
"6":"Six","7":"Seven", "8":"Eight", "9":"Nine","0":"Zero"}
output = ""
for number in phone:
    output+=digit_mapping.get(number)+" "
print("In words: " + output)
```

--Fun with comprehensions

```
str_digit=['1','3','abv','edf','5']
str_digits2=[c for c in str_digit if c.isdigit()]
print(str_digits2)
```

--Using set operation

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
dict2 = {'c': 10, 'd': 20, 'e': 30, 'f': 40, 'g': 50}

shared_values=set(dict1.values()).intersection(dict2.values())
```

```
print(shared_values)


shared_keys = set(dict1).intersection(dict2)
print(shared_keys)
```

--Writing custom functions

```
def compound(balance,rate,num_periods):
    """
    calculates compounded balance at the end of the specified period for the specified intitial
    balances @ of specified annual rate
    """
    for i in range(1,num_periods+1):
        balance=round(balance*(1+rate),2)
    return balance



def compound_by_period(balance,rate,num_periods):
    """
    calculates compounded balance at the end of the each year for the specified period, specified
    annual rate and specified intitial balance
    """
    bal= [ ]
    for i in range(0,num_periods+1):
        bal.append(balance)
        balance=round(balance*(1+rate),2)
    return bal
```

```
def change_per_period(alist):
    """
    calculates the changes in intitial balance at the end of each year for the specified period,
     specified annual rate and specified intial balance
    """
    changes=[ ]
    for i in range(0,len(alist)-1):
          difference=round(alist[i+1]-alist[i],2)
         changes.append(difference)
    return changes
```

--Writing custom class, function to encrypt and decode user input message with user defined shift

```
class ShiftCipher:
    """
    ShiftCipher objects that can encrypt and decode text messages based on a specific shift
    length.
    """

    def __init__(self, shift):
      """
       Constructs a ShiftCipher for the specified degree of shift (positive or negative),
       by building a cipher (dictionary mapping from letters to other letters), and
       a decoder (the inverse of the cipher)
       """
       self.shift = shift
       self.letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
        self.cipher = {self.letters[i]: self.letters[(i+self.shift) %len(self.letters)] for i in
        range(len(self.letters))}
```

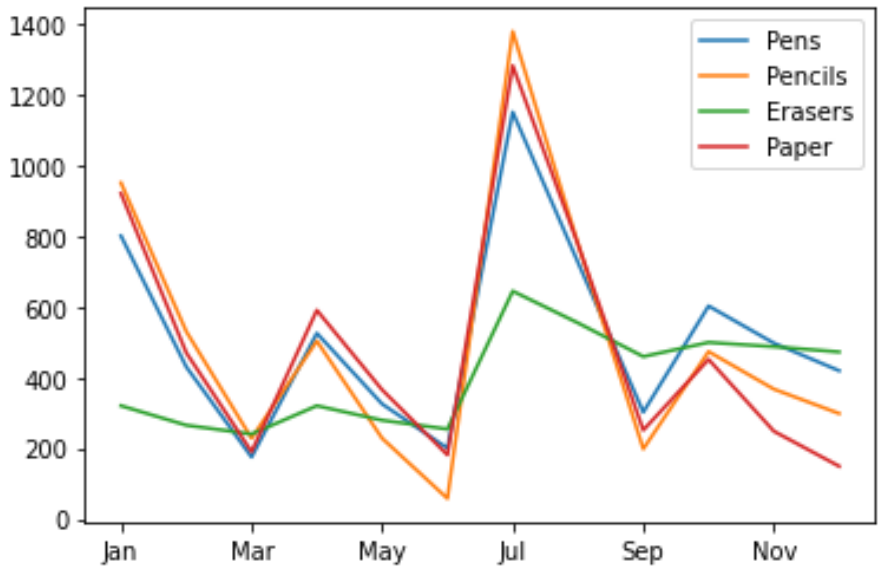
```
        self.decoder = {self.letters[i]: self.letters[(i-self.shift)%len(self.letters)] for i in
        range(len(self.letters))}


    def transform_message(self, message, cipher):
        """
        Transforms a message using the specified cipher.  Is not called by users directly,
        and can be called with either the cipher (to encrypt) or the decoder (to decode).
      """
        tmsg = ''
        for c in message:
            tmsg = tmsg + cipher.get(c, c)
        return tmsg


    def encrypt(self, message):
        """
        Transforms a message using the cipher, by calling self.transform_message
        """
        return self.transform_message(message,self.cipher)


    def decode(self, message):
        """
        Transforms a message using the decoder, by calling self.transform_message
        """
        return self.transform_message(message,self.decoder)
```

--Analysis with Python

##The sales2018 dataframe records the sales in each product category during each month.

##The orders2018 dataframe records the purchasing of new inventory at the start of each quarter during 2018

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

sales2018 = pd.read_csv('salesdata.csv', index_col='Month')
orders2018 = pd.read_csv('orders.csv', index_col='Month')

## Outer joining sales and orders

sales_and_orders = pd.concat((sales2018,orders2018),axis=1, sort=False,
                keys=('sales','orders'))

##Filling missing data

sales_and_orders.fillna(0.0,inplace='True')

## Calculating inventory and plotting the data

inventory = sales_and_orders["orders"].cumsum() - sales_and_orders["sales"].cumsum()
inventory.plot();
```



##Reading files

##Available in directory

```
import glob

def read_multiyear_sales_data(directory):
    sales = {}
   salesfiles = glob.glob('sales_directory/sales*.csv')
    for filename in salesfiles:
        stop = filename.find('.csv')
        start = stop-4
        year = filename[start:stop]
        df = pd.read_csv(filename, index_col='Month')
        sales[year] = df
    return sales

sales_by_year = read_multiyear_sales_data('sales_directory')
```

Using URL

```
import pandas as pd
poll_data = pd.read_csv("https://projects.fivethirtyeight.com/polls-page/president_primary_polls.csv")
```

Reading excel file

```
dfraw = pd.read_excel(r'C:\Users\Sandeep\Desktop\Python\WorldHappinessReport\WHR2018Chapter2OnlineData.xls',sheet_name='Table2.1')
```

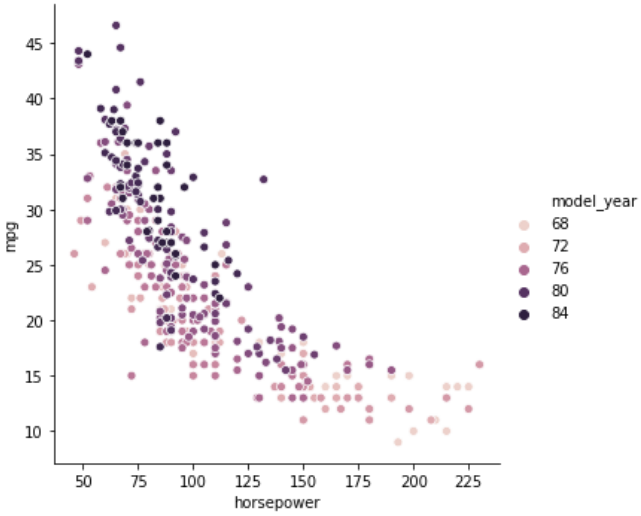
##Creating a data frame

```
def make_dataframe_from_sales_data(sales):
    df = pd.concat(sales, axis=0, keys=sorted(sales.keys()), names=['Year', 'Month'])
     lookup = {'Jan': '01', 'Feb': '02', 'Mar': '03',
                'Apr': '04', 'May': '05', 'Jun': '06',
                'Jul': '07', 'Aug': '08', 'Sep': '09',
                'Oct': '10', 'Nov': '11', 'Dec': '12'}
    df = df.rename(index=lookup)
    df.index = ["-".join(x) for x in df.index.ravel()]
    df.index = pd.to_datetime(df.index)
    return df

sales_df = make_dataframe_from_sales_data(sales_by_year)
```

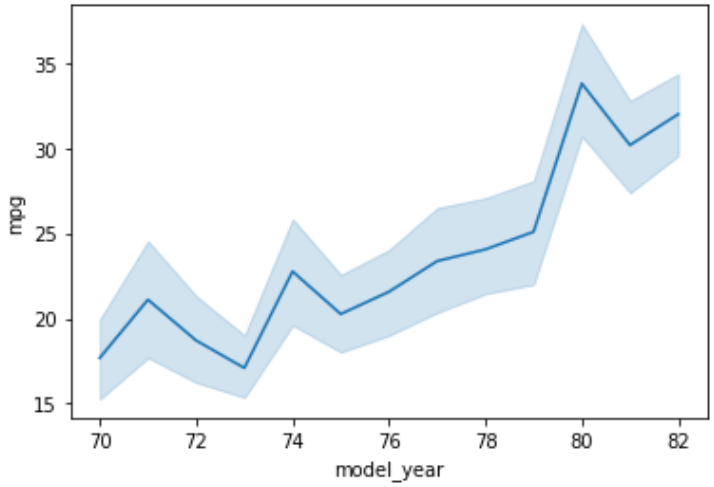
##Plotting with Seaborn and Matplotlib

```
Import seaborn as sns
Import matplotlib.pyplot as plt
```

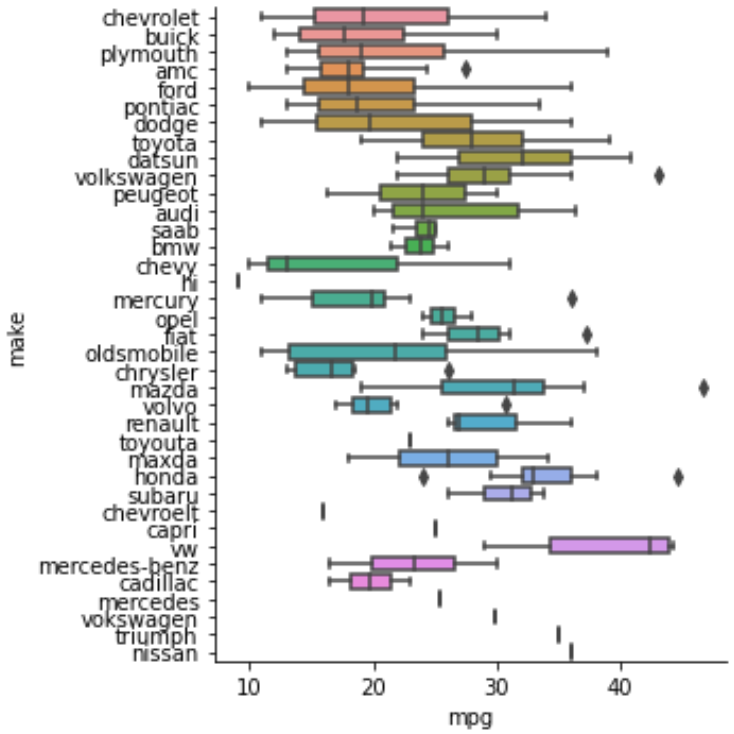
```
sns.relplot(data=mpg,x="horsepower",y="mpg",hue="model_year");
```



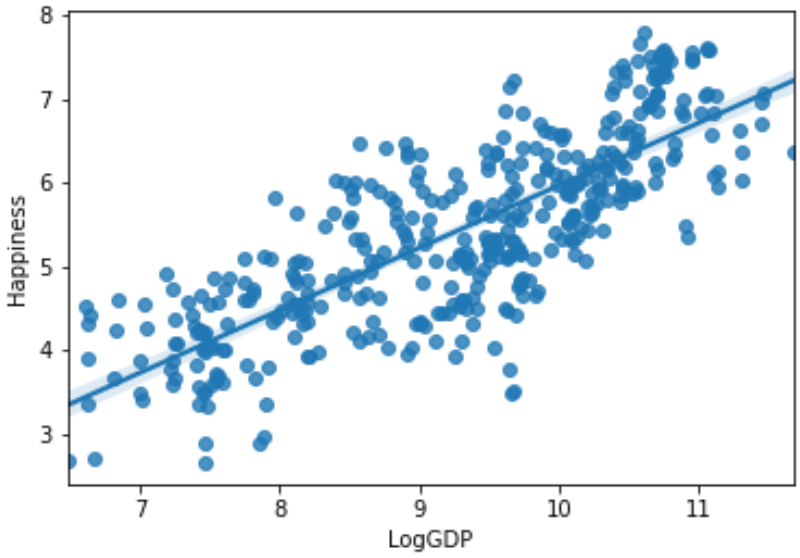

```
sns.lineplot(data=mpg,x="model_year",y="mpg",ci=99)
```



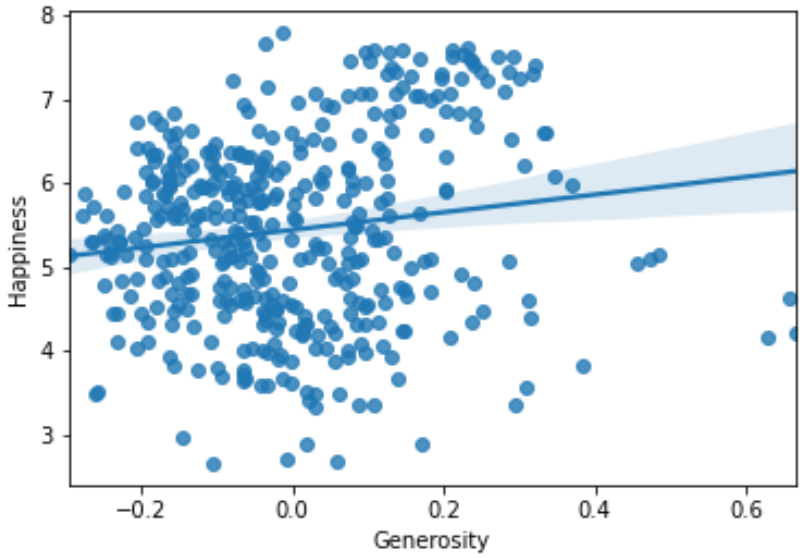
```
sns.catplot(x="mpg",y="make",data=mpg,kind='box');
```




```
sns.regplot(x='LogGDP', y='Happiness', data=df1517);
```

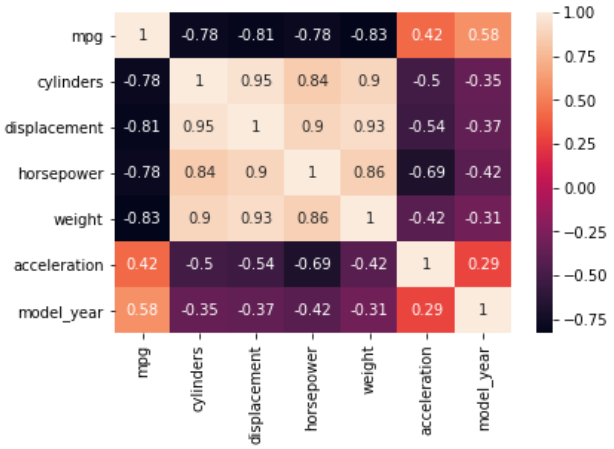


```
sns.regplot(x='Generosity', y='Happiness', data=df1517);
```



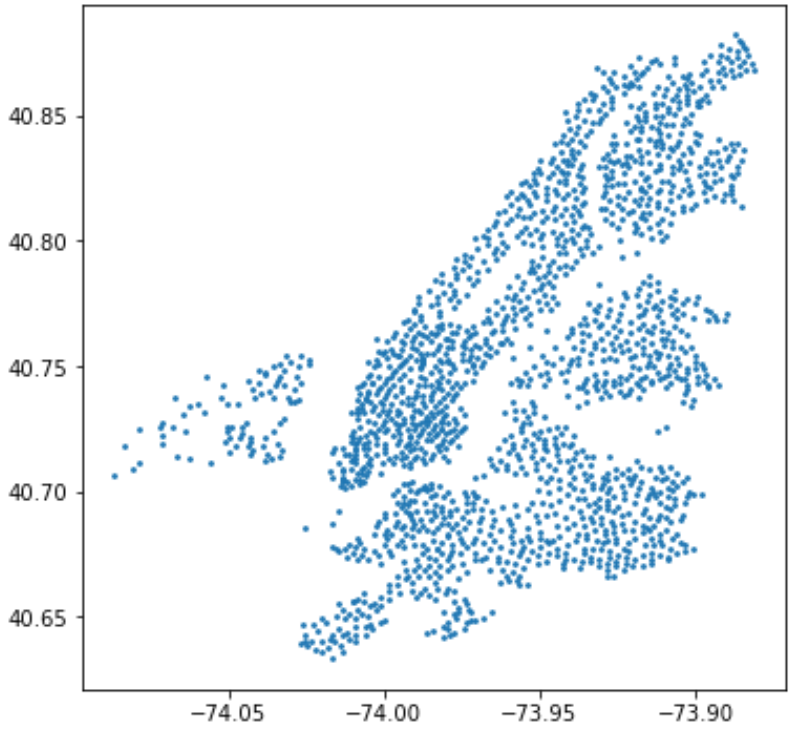
##Examining the correlation between the data and plotting the relation

```
mpg_corr=mpg.corr()
sns.heatmap(data=mpg_corr,annot=True)
```



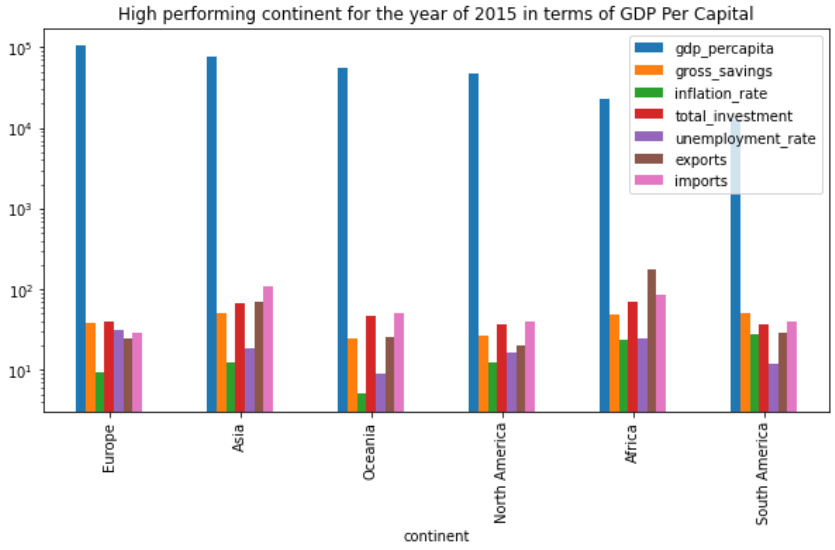
Extracting the coordinates of the stations and plotting them

```
import numpy as np
pairs=[]
for k in datadict['data']['stations']:
    if k['lat']!=0 and k['lat']!=0:
        pairs.append([(k.get('lon')),(k.get('lat'))])

coordinates = np.array(pairs)

import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(6,6))
plt.scatter(coordinates[:,0],coordinates[:,1], s=3 );
```



Analyzing economic indicators of continent

```
max_conti_2015 = data_2015.drop(columns='year',
axis=1).groupby('continent').max().sort_values(by = 'gdp_percapita', ascending=False)

max_conti_2010.plot(kind='bar',title='High performing continent for the year of 2015 in terms of GDP Per Capital',logy=True,figsize=(10,5));
```



Visualizing economic indicators

```
max_conti_2015['gdp_percapita'].plot(kind='bar',legend=True)

max_conti_2015['inflation_rate'].plot(kind='bar',color='r',logy=True, legend=True)

max_conti_2015['total_investment'].plot(kind='bar',color='g',logy=True, legend=True);
```