

**SUBJECT : Artificial Intelligence**

**CODE: MCA317**

**VIVEKANANDA SCHOOL OF INFORMATION TECHNOLOGY**



**MASTERS OF COMPUTER APPLICATIONS**

**SUBMITTED TO :**

**Dr. Pawan Whig**

**SUBMITTED BY:**

**Vinita Verma**


**06917704418**



**VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES**

**BATCH- (2018-2021)**

# INDEX

S NO.	 PRACTICALS	DATE
1.	Data analysis using via cognitive labs using numpy/pandas	19/9/20
2.	Linear regression	22/9/20
3.	SVM	11/11/20
4.	Naive Bayes	15/11/20
5.	Decision Tree	16/11/20
6.	Covid-19 data analysis	1/12/20
7.	Multiple Regression	6/12/20
8.	Logistic Regression	10/12/20

# 1. Data Analysis via cognitive labs using pandas and numpy

## Software - IBM Cloud Watson Studio

NumPy stands for 'Numerical Python' or 'Numeric Python'. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation.

Similar to NumPy, Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Dataframe. It is like a spreadsheet with column names and row labels.

## Import Dataset

In [2]:

```
import types
import pandas as pd
from boto3.client import Config
import boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your
# credentials.
# You might want to remove those credentials before you share the notebook.
client_38c83f8f236a4358a4f89d762c5e9aa4 = boto3.client(service_name='s3',
    ibm_api_key_id='RtNCGnw1LwvYR3MHbf5KmmIc8bFhdT5Bp8fKqi3ZP38z',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.eu-geo.objectstorage.service.networklayer.com')

body = client_38c83f8f236a4358a4f89d762c5e9aa4.get_object(Bucket='artificialintelligenc
epracticalfi-donotdelete-pr-guqjnqp0qaqgzi',Key='Ecommerce Customers.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data= pd.read_csv(body)
df_data.head()
```

Out[2]:

	Email	Address	Avatar	Avg. Session Length	Time on App
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189

**Pandas** - pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

In [14]:

```
df_data.columns
```

Out[14]:

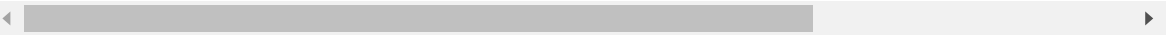
```
Index(['Email', 'Address', 'Avatar', 'Avg. Session Length', 'Time on App',  
      'Time on Website', 'Length of Membership', 'Yearly Amount Spent'],  
      dtype='object')
```

In [3]:

```
df_data.head(5)
```

Out[3]:

	Email	Address	Avatar	Avg. Session Length	Time on App
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189



In [4]:

```
df_data.tail(5)
```

Out[4]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Ti W
495	lewisjessica@craig-evans.com	4483 Jones Motorway Suite 872\nLake Jamiefurt,...	Tan	33.237660	13.566160	36.4
496	katrina56@gmail.com	172 Owen Divide Suite 497\nWest Richard, CA 19320	PaleVioletRed	34.702529	11.695736	37.1
497	dale88@hotmail.com	0787 Andrews Ranch Apt. 633\nSouth Chadburgh, ...	Cornsilk	32.646777	11.499409	38.3
498	cwilson@hotmail.com	680 Jennifer Lodge Apt. 808\nBrendacheater, TX...	Teal	33.322501	12.391423	36.8
499	hannahwilson@davidson.com	49791 Rachel Heights Apt. 898\nEast Drewboroug...	DarkMagenta	33.715981	12.418808	35.7

In [5]:

```
df_data.shape
```

Out[5]:

(500, 8)

In [6]:

```
df_data.describe()
```

Out[6]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

In [7]:

```
df_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Email                 500 non-null   object
1   Address               500 non-null   object
2   Avatar               500 non-null   object
3   Avg. Session Length  500 non-null   float64
4   Time on App          500 non-null   float64
5   Time on Website      500 non-null   float64
6   Length of Membership  500 non-null   float64
7   Yearly Amount Spent  500 non-null   float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

In [8]:

```
df_data.dtypes
```

Out[8]:

```
Email                object
Address              object
Avatar               object
Avg. Session Length  float64
Time on App          float64
Time on Website      float64
Length of Membership float64
Yearly Amount Spent  float64
dtype: object
```

In [9]:

```
df_data.sort_values("Avg. Session Length")
```

Out[9]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Time W
12	knelson@gmail.com	6705 Miller Orchard Suite 186\nLake Shanestad,...	RoyalBlue	29.532429	10.961298	37.4
312	douglasdunlap@boone- rose.com	093 Larson Ports\nWest Kathryn, OK 91243	Purple	30.393185	11.802986	36.1
299	morganorozco@hotmail.com	0001 Mack Mill\nNorth Jennifer, NE 42021-5936	LightPink	30.492537	11.562936	35.9
330	dbenson@simpson.net	732 Heather Place\nNorth Michael, VT 92527	DodgerBlue	30.574364	11.351049	37.0
15	jstark@anderson.com	49558 Ramirez Road Suite 399\nPhillipstad, OH ...	Peru	30.737720	12.636606	36.1
...	...	...	...	...	...	...
257	maureenlopez@gmail.com	82537 Alice Centers\nGregland, OR 71749	SeaShell	35.530904	11.379257	36.6
488	zscott@wright.com	9909 Hoffman Ranch Suite 195\nScotthaven, SC 5...	PeachPuff	35.630854	12.125402	38.1
396	waltonkaren@gmail.com	355 Villegas Isle Apt. 070\nWest Jenniferview,...	Green	35.742670	10.889828	35.1
390	michaelcampbell@yahoo.com	96480 White Lane Suite 521\nPattersonhaven, OR...	Gray	35.860237	11.730661	36.1
154	nathan86@hotmail.com	748 Michael Plaza\nWest Billyside, UT 20799	MidnightBlue	36.139662	12.050267	36.9

500 rows × 8 columns





In [10]:

```
df_data.sort_values("Avatar")
```

Out[10]:

	Email	Address	Avatar	Avg. Session Length	Time on App	T V
209	wagnerbrian@hotmail.com	50593 Wells Roads Apt. 110\nSouth Amy, MI 0696...	AliceBlue	32.559493	11.797796	37.
448	flevine@gmail.com	5292 Melanie Crescent Apt. 064\nFischerborough...	AliceBlue	32.204655	12.480702	37.
376	sfarley@jones.com	0554 Powers Curve\nNathanchester, FL 06878-6336	AntiqueWhite	32.397422	12.055340	37.
302	lmalone@gmail.com	USS Beasley\nFPO AP 50556-7615	AntiqueWhite	32.975193	13.394452	37.
208	freemantina@cannon.org	870 Dennis Throughway\nWilsonport, PW 12658	AntiqueWhite	32.903454	10.542645	35.
...	...	...	...	...	...	...
145	linda90@yoder.org	979 Alison Motorway Apt. 676\nNorth Frank, HI ...	WhiteSmoke	33.477190	12.488067	36.
207	lewiskendra@yahoo.com	988 Matthew Plaza\nLake Jacobshire, AL 37889	Yellow	33.324241	11.084584	36.
238	archeremily@baldwin.com	USCGC Hernandez\nFPO AE 53064	YellowGreen	31.260647	13.266760	36.
351	tluna@hotmail.com	01512 Hendricks Rue\nEast Pamela, PR 46481	YellowGreen	32.189845	11.386776	38.
469	kstafford@estes- nguyen.com	PSC 4856, Box 1297\nAPO AA 17032- 7944	YellowGreen	31.169507	13.970181	36.

500 rows × 8 columns



In [12]:

```
df_data["Avatar"].value_counts()
```

Out[12]:

```
CadetBlue          7
Teal               7
Cyan              7
SlateBlue         7
GreenYellow       7
..
PowderBlue        1
MediumPurple      1
Coral             1
LightGoldenRodYellow 1
PaleGreen         1
Name: Avatar, Length: 138, dtype: int64
```

In [13]:

```
df_data.nunique()
```

Out[13]:

```
Email            500
Address          500
Avatar           138
Avg. Session Length 500
Time on App      500
Time on Website  500
Length of Membership 500
Yearly Amount Spent 500
dtype: int64
```

In [15]:

```
df_data['Time on App'].mean()
```

Out[15]:

```
12.052487937166132
```

In [17]:

```
df_data[0:-1]
```

Out[17]:

	Email	Address	Avatar	Avg. Session Length	Time A
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.6556
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.1094
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.3302
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.7175
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.7951
...	...	...	...	...	...
494	kellydeborah@chan.biz	354 Sanchez Wall Suite 884\nJuliabury, VI 39735	DarkTurquoise	33.431097	13.3506
495	lewisjessica@craig-evans.com	4483 Jones Motorway Suite 872\nLake Jamiefurt,...	Tan	33.237660	13.5661
496	katrina56@gmail.com	172 Owen Divide Suite 497\nWest Richard, CA 19320	PaleVioletRed	34.702529	11.6957
497	dale88@hotmail.com	0787 Andrews Ranch Apt. 633\nSouth Chadburgh, ...	Cornsilk	32.646777	11.4994
498	cwilson@hotmail.com	680 Jennifer Lodge Apt. 808\nBrendachester, TX...	Teal	33.322501	12.3914

499 rows × 8 columns



Using Numpy

NumPy is a commonly used Python data analysis package. By using NumPy, you can speed up your workflow, and interface with other packages in the Python ecosystem, like scikit-learn, that use NumPy under the hood. NumPy was originally developed in the mid 2000s, and arose from an even older package called Numeric

In [25]:

```
import numpy as np
```

In [26]:

```
df_data = np.array(df_data)
```

In [27]:

```
df_data
```

Out[27]:

```
array([[ 'mstephenson@fernandez.com',  
        '835 Frank Tunnel\nWrightmouth, MI 82180-9605', 'Violet', ...,  
        39.57766801952616, 4.0826206329529615, 587.9510539684005],  
 [ 'hduke@hotmail.com',  
        '4547 Archer Common\nDiazchester, CA 06566-8576', 'DarkGreen',  
        ..., 37.268958868297744, 2.66403418213262, 392.2049334443264],  
 [ 'pallen@yahoo.com',  
        '24645 Valerie Unions Suite 582\nCobbborough, DC 99414-7564',  
        'Bisque', ..., 37.110597442120856, 4.104543202376424,  
        487.54750486747207],  
 ...,  
 [ 'dale88@hotmail.com',  
        '0787 Andrews Ranch Apt. 633\nSouth Chadburgh, TN 56128',  
        'Cornsilk', ..., 38.33257633196044, 4.958264472618699,  
        551.6201454762477],  
 [ 'cwilson@hotmail.com',  
        '680 Jennifer Lodge Apt. 808\nBrendacheater, TX 05000-5873',  
        'Teal', ..., 36.840085729767004, 2.336484668112853,  
        456.46951006629797],  
 [ 'hannahwilson@davidson.com',  
        '49791 Rachel Heights Apt. 898\nEast Drewborough, OR 55919-9528',  
        'DarkMagenta', ..., 35.771016191612965, 2.7351595670822753,  
        497.7786422156802]], dtype=object)
```

In [30]:

```
#indexing numpy arrays  
df_data[2,3]
```

Out[30]:

```
33.000914755642675
```

In [31]:

```
#slicing numpy arrays
df_data[0:3,3]
```

Out[31]:

```
array([34.49726772511229, 31.92627202636016, 33.000914755642675],
      dtype=object)
```

In [33]:

```
df_data[:3]
```

Out[33]:

```
array([[ 'mstephenson@fernandez.com',
        '835 Frank Tunnel\nWrightmouth, MI 82180-9605', 'Violet',
        34.49726772511229, 12.65565114916675, 39.57766801952616,
        4.0826206329529615, 587.9510539684005],
 [ 'hduke@hotmail.com',
        '4547 Archer Common\nDiazchester, CA 06566-8576', 'DarkGreen',
        31.92627202636016, 11.109460728682564, 37.268958868297744,
        2.66403418213262, 392.2049334443264],
 [ 'pallen@yahoo.com',
        '24645 Valerie Unions Suite 582\nCobbborough, DC 99414-7564',
        'Bisque', 33.000914755642675, 11.330278057777512,
        37.110597442120856, 4.104543202376424, 487.54750486747207]],
      dtype=object)
```

In [34]:

```
df_data[:,:]
```

Out[34]:

```
array([[ 'mstephenson@fernandez.com',
        '835 Frank Tunnel\nWrightmouth, MI 82180-9605', 'Violet', ...,
        39.57766801952616, 4.0826206329529615, 587.9510539684005],
 [ 'hduke@hotmail.com',
        '4547 Archer Common\nDiazchester, CA 06566-8576', 'DarkGreen',
        ..., 37.268958868297744, 2.66403418213262, 392.2049334443264],
 [ 'pallen@yahoo.com',
        '24645 Valerie Unions Suite 582\nCobbborough, DC 99414-7564',
        'Bisque', ..., 37.110597442120856, 4.104543202376424,
        487.54750486747207],
 ...,
 [ 'dale88@hotmail.com',
        '0787 Andrews Ranch Apt. 633\nSouth Chadburgh, TN 56128',
        'Cornsilk', ..., 38.33257633196044, 4.958264472618699,
        551.6201454762477],
 [ 'cwilson@hotmail.com',
        '680 Jennifer Lodge Apt. 808\nBrendachester, TX 05000-5873',
        'Teal', ..., 36.840085729767004, 2.336484668112853,
        456.46951006629797],
 [ 'hannahwilson@davidson.com',
        '49791 Rachel Heights Apt. 898\nEast Drewborough, OR 55919-9528',
        'DarkMagenta', ..., 35.771016191612965, 2.7351595670822753,
        497.7786422156802]], dtype=object)
```

In [35]:

```
#to generate a random vector
```

```
np.random.rand(3)
```

Out[35]:

```
array([0.56241195, 0.81095058, 0.5500436 ])
```

In [ ]:

## 2. Linear Regression

- Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope.
- Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable
- Regression is a method of modelling a target value based on independent predictors.
- It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog)

## Imports Library

In [ ]:

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

**Read in the Ecommerce Customers csv file as a DataFrame called customers.**

**Import the dataset**

In [ ]:

```
customers = pd.read_csv('Ecommerce Customers.csv')
```

**Exploratory Data Analysis**

In [ ]:

```
customers.head(10)
```

Out[ ]:

	Email	Address	Avatar	Avg. Session Length	Time on App
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189
5	alvareznancy@lucas.biz	645 Martha Park Apt. 611\nJeffreychester, MN 6...	FloralWhite	33.871038	12.026925
6	katherine20@yahoo.com	68388 Reyes Lights Suite 692\nJosephbury, WV 9...	DarkSlateBlue	32.021596	11.366348
7	awatkins@yahoo.com	Unit 6538 Box 8980\nDPO AP 09026- 4941	Aqua	32.739143	12.351959
8	vchurch@walter-martinez.com	860 Lee Key\nWest Debra, SD 97450-0495	Salmon	33.987773	13.386235
9	bonnie69@lin.biz	PSC 2734, Box 5255\nAPO AA 98456- 7482	Brown	31.936549	11.814128





In [ ]:

```
customers.tail(10)
```

Out[ ]:

	Email	Address	Avatar	Avg. Session Length	Time (s)
490	brian28@sanchez.org	7446 Mary Ferry\nLake Sherryfurt, GA 49066-0207	GhostWhite	34.695591	11.6089
491	leonardhancock@hotmail.com	64147 Alexander Station Apt. 474\nEast Jasonvi...	SeaShell	34.343922	11.6930
492	davidsonkathleen@gmail.com	70128 Zimmerman Overpass\nRobertsshire, VA 59860	DarkBlue	33.680937	11.2015
493	nathan84@lowery.net	01242 Stephanie Ways Suite 003\nChurchville, M...	MediumSeaGreen	32.060914	12.6254
494	kellydeborah@chan.biz	354 Sanchez Wall Suite 884\nJuliabury, VI 39735	DarkTurquoise	33.431097	13.3506
495	lewisjessica@craig-evans.com	4483 Jones Motorway Suite 872\nLake Jamiefurt,...	Tan	33.237660	13.5661
496	katrina56@gmail.com	172 Owen Divide Suite 497\nWest Richard, CA 19320	PaleVioletRed	34.702529	11.6957
497	dale88@hotmail.com	0787 Andrews Ranch Apt. 633\nSouth Chadburgh, ...	Cornsilk	32.646777	11.4994
498	cwilson@hotmail.com	680 Jennifer Lodge Apt. 808\nBrendachester, TX...	Teal	33.322501	12.3914
499	hannahwilson@davidson.com	49791 Rachel Heights Apt. 898\nEast Drewboroug...	DarkMagenta	33.715981	12.4188



In [ ]:

```
customers.describe()
```

Out[ ]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

In [ ]:

```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Email                  500 non-null    object
1   Address                 500 non-null    object
2   Avatar                 500 non-null    object
3   Avg. Session Length    500 non-null    float64
4   Time on App             500 non-null    float64
5   Time on Website         500 non-null    float64
6   Length of Membership    500 non-null    float64
7   Yearly Amount Spent     500 non-null    float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

## Data Visualization

In [ ]:

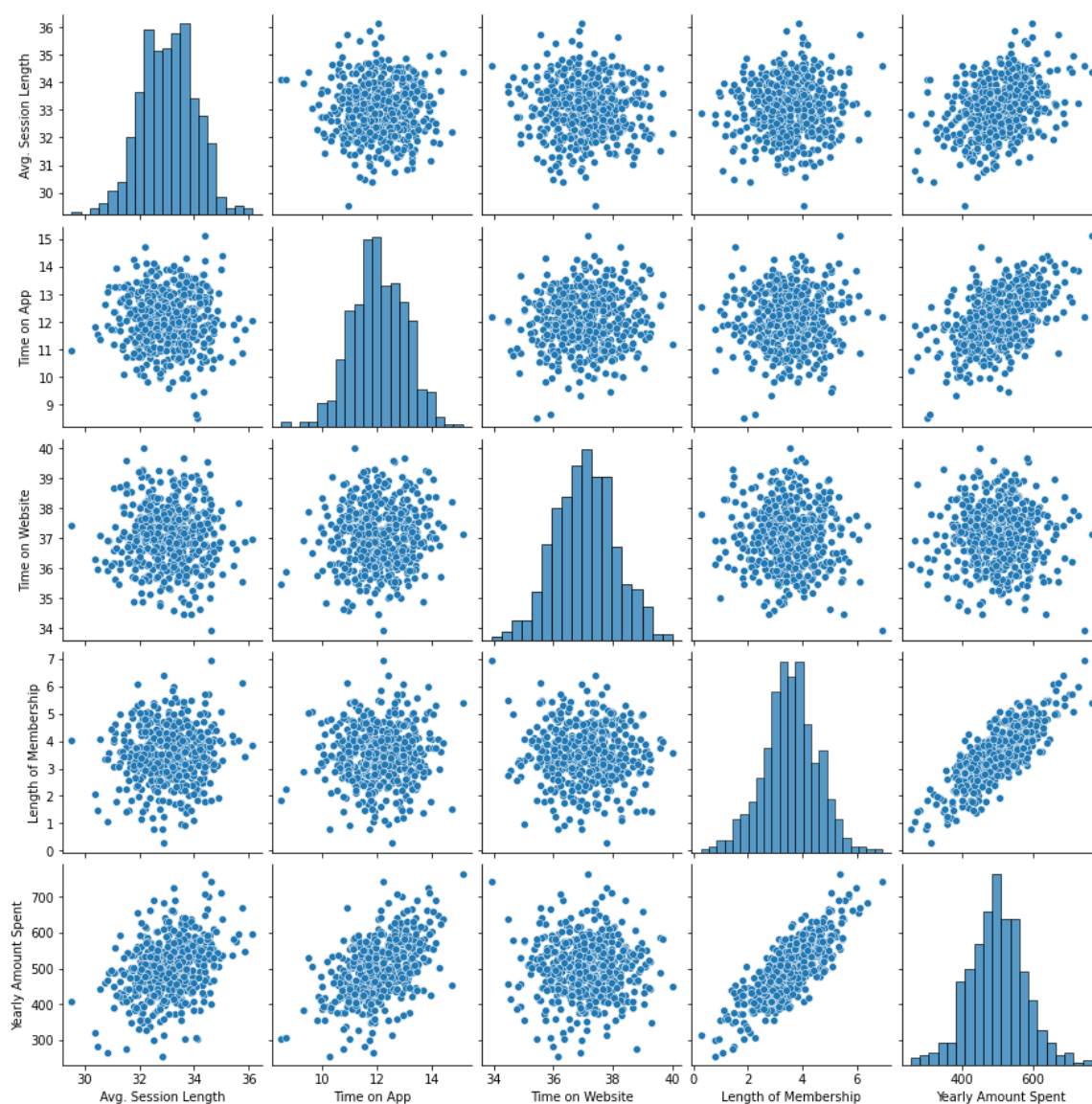
```
import seaborn as sns
```

In [ ]:

```
sns.pairplot(customers)
```

Out[ ]:

<seaborn.axisgrid.PairGrid at 0x7f6e9eacada0>



**Based off this plot what looks to be the most correlated feature with Yearly Amount Spent?**

In [ ]:

```
#Length of Membership
```

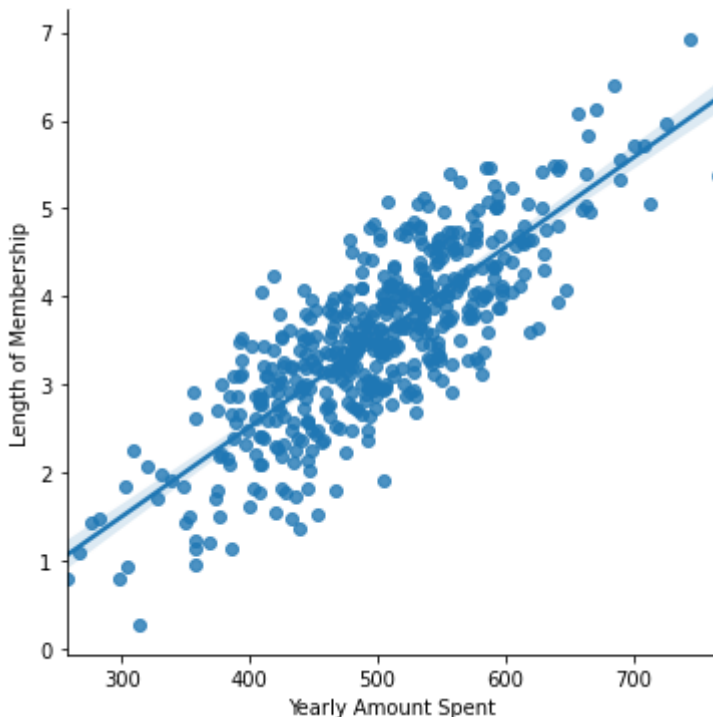
**Create a linear model plot (using seaborn's Implot) of Yearly Amount Spent vs. Length of Membership.**

In [ ]:

```
sns.lmplot(x='Yearly Amount Spent', y='Length of Membership', data=customers)
```

Out[ ]:

<seaborn.axisgrid.FacetGrid at 0x7f6e8b63f8d0>



## Training and Testing Data

Now that we've explored the data a bit, let's go ahead and split the data into training and testing sets. **Set a variable X equal to the numerical features of the customers and a variable y equal to the "Yearly Amount Spent" column.**

In [ ]:

```
y = customers['Yearly Amount Spent']
```

In [ ]:

```
X = customers[['Avg. Session Length', 'Time on App', 'Time on Website', 'Length of Membership']]
```

## Split the dataset

Use `model_selection.train_test_split` from `sklearn` to split the data into training and testing sets. Set `test_size=0.3` and `random_state=101`

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

## Training the Model

In [ ]:

```
X_train
```

Out[ ]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership
161	33.503705	12.399436	35.012806	0.968622
72	32.386252	10.674653	38.006583	3.401522
246	31.909627	11.347264	36.323652	5.314354
230	32.351478	13.105159	35.574842	3.641497
391	33.481931	11.918670	37.317705	3.336339
...	...	...	...	...
63	32.789773	11.670066	37.408748	3.414688
326	33.217188	10.999684	38.442767	4.243813
337	31.827979	12.461147	37.428997	2.974737
11	33.879361	11.584783	37.087926	3.713209
351	32.189845	11.386776	38.197483	4.808320

400 rows × 4 columns

In [ ]:

```
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
lm = LinearRegression()
```

In [ ]:

```
lm.fit(X_train,y_train)
```

Out[ ]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [ ]:

```
print('Coefficients: \n', lm.coef_)
```

```
Coefficients:  
[26.02948861 38.70983485  0.35618404 61.47280903]
```

## Predicting Test Data

In [ ]:

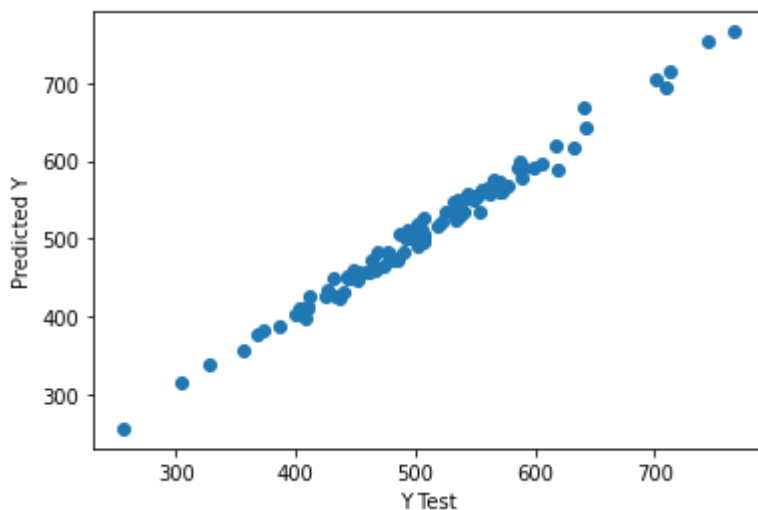
```
predictions = lm.predict(X_test)
```

In [ ]:

```
plt.scatter(y_test, predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[ ]:

Text(0, 0.5, 'Predicted Y')



## Evaluating the Model

Let's evaluate our model performance by calculating the residual sum of squares and the explained variance score ( $R^2$ ).

**Calculate the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error.**

In [ ]:

```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

MAE: 7.758907540457862  
MSE: 91.82335857016922  
RMSE: 9.582450551407465

# Conclusion

We still want to figure out the answer to the original question, do we focus our effort on mobile app or website development? Or maybe that doesn't even really matter, and Membership Time is what is really important. Let's see if we can interpret the coefficients at all to get an idea.

Recreate the dataframe below.

In [ ]:

```
coefficients = pd.DataFrame(lm.coef_,X.columns)
coefficients.columns = ['Coefficient']
coefficients
```

Out[ ]:

	Coefficient
Avg. Session Length	26.029489
Time on App	38.709835
Time on Website	0.356184
Length of Membership	61.472809

## Importing the Necessary Libraries

First we import the necessary libraries of the python for demonstration of the Decision Tree Classifier

In [1]:

```
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

Read the data of the weather from the csv file using read\_csv function of pandas dataframe

In [3]:

```
data = pd.read_csv('daily_weather.csv')
```

## Daily Weather Data Description

The file **daily\_weather.csv** is a comma-separated file that contains weather data. This data comes from a weather station located in San Diego, California. The weather station is equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured.

Let's now check all the columns in the data.

Know about various columns in the dataset.

In [4]:

```
data.columns
```

Out[4]:

```
Index(['number', 'air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9am',
      'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9am',
      'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9am',
      'relative_humidity_3pm'],
      dtype='object')
```



In [5]:

```
data.head()
```

Out[5]:

	number	air_pressure_9am	air_temp_9am	avg_wind_direction_9am	avg_wind_speed_9am
0	0	918.060000	74.822000	271.100000	2.080354
1	1	917.347688	71.403843	101.935179	2.443009
2	2	923.040000	60.638000	51.000000	17.067852
3	3	920.502751	70.138895	198.832133	4.337363
4	4	921.160000	44.294000	277.800000	1.856660

Checking is there exists null values in the dataset or not

In [6]:

```
data[data.isnull().any(axis=1)].head()
```

Out[6]:

	number	air_pressure_9am	air_temp_9am	avg_wind_direction_9am	avg_wind_speed_9am
16	16	917.890000	NaN	169.200000	2.192200
111	111	915.290000	58.820000	182.600000	15.613840
177	177	915.900000	NaN	183.300000	4.719940
262	262	923.596607	58.380598	47.737753	10.636270
277	277	920.480000	62.600000	194.400000	2.751430

## **\*\*Exploratory Data Analysis\*\***

We will not need to number for each row so we can clean it.

Data Cleaning process --> As number column contains unique values which can not help us making any decision

In [7]:

```
del data['number']
```

Calculating the amount of data or say number of rows in the dataset before removing the rows containing null values

In [8]:

```
before_rows = data.shape[0]  
print(before_rows)
```

1095

Removing the rows which contains the null values

In [9]:

```
data = data.dropna()
```

Calculating the amount of data or say number of rows in the dataset after removing the rows containing null values

In [10]:

```
after_rows = data.shape[0]  
print(after_rows)
```

1064

Calculate how many rows are deleted which contains the Null Values

In [ ]:

In [11]:

```
before_rows - after_rows
```

Out[11]:

31

Filter the values which contains more than 24.99 relative humidity at 3pm.

In [12]:

```
clean_data = data.copy()  
clean_data['high_humidity_label'] = (clean_data['relative_humidity_3pm'] > 24.99) * 1  
clean_data['high_humidity_label'].head()
```

Out[12]:

```
0    1  
1    0  
2    0  
3    0  
4    1
```

Name: high\_humidity\_label, dtype: int64

In [13]:

```
y = clean_data[['high_humidity_label']].copy()
y.head()
```

Out[13]:

	high_humidity_label
0	1
1	0
2	0
3	0
4	1

In [14]:

```
clean_data['relative_humidity_3pm'].head()
```

Out[14]:

```
0    36.160000
1    19.426597
2    14.460000
3    12.742547
4    76.740000
Name: relative_humidity_3pm, dtype: float64
```

In [15]:

```
y.head()
```

Out[15]:

	high_humidity_label
0	1
1	0
2	0
3	0
4	1

## Use 9am Sensor Signals as Features to Predict Humidity at 3pm

Storing all the Morning features other than Humidity at 3 pm in the morning feature

In [16]:

```
morning_features = ['air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9am',  
                    'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9am',  
                    'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9am']
```

Copying the values from the clean\_data dataset to new dataset x which only consist of the Morning Feature Data

In [17]:

```
x=clean_data[morning_features].copy()  
x.columns
```

Out[17]:

```
Index(['air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9am',  
      'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9a  
m',  
      'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9a  
m'],  
      dtype='object')
```

In [18]:

```
y.columns
```

Out[18]:

```
Index(['high_humidity_label'], dtype='object')
```

## Splitting the Dataset

By using train\_test\_split we have split the data into training dataset and testing datasets.

In [20]:

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=324)
```

## 3. Support Vector Machine (SVM)

- “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.
- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

## Fit on Train Set

In [21]:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
```

Out[21]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

### Making Predictions

In [22]:

```
y_pred = svclassifier.predict(X_test)
```

Evaluating the algorithm - Confusion matrix, precision, recall, and F1 measures are the most commonly used metrics for classification tasks. Scikit-Learn's metrics library contains the `classification_report` and `confusion_matrix` methods, which can be readily used to find out the values for these important metrics.

In [23]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[162  13]
 [ 22 155]]
```

	precision	recall	f1-score	support
0	0.88	0.93	0.90	175
1	0.92	0.88	0.90	177
accuracy			0.90	352
macro avg	0.90	0.90	0.90	352
weighted avg	0.90	0.90	0.90	352

## Measure Accuracy of the Classifier

In [24]:

```
print(accuracy_score(y_test,y_pred)*100)
```

90.05681818181817

## 4. Naive Bayes Classifier

- It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.
- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

In [25]:

```
from sklearn.naive_bayes import GaussianNB
```

```
clf = GaussianNB()  
clf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/naive_bayes.py:206: DataCon  
versionWarning: A column-vector y was passed when a 1d array was expected.  
Please change the shape of y to (n_samples, ), for example using ravel().  
  y = column_or_1d(y, warn=True)
```

Out[25]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [26]:

```
y_pred1 = clf.predict(X_test)
```

Evaluating the algorithm-

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred1))
print(classification_report(y_test,y_pred1))
```

[[171 4] [ 73 104]]	precision	recall	f1-score	support
0	0.70	0.98	0.82	175
1	0.96	0.59	0.73	177
accuracy			0.78	352
macro avg	0.83	0.78	0.77	352
weighted avg	0.83	0.78	0.77	352

## Measure Accuracy of the Classifier

```
print(accuracy_score(y_test,y_pred1)*100)
```

78.125

## 5. Decision Tree

- Decision tree algorithm falls under the category of supervised learning.
- They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.

We have made a classifier for making the Decision Tree and to train the data using this classifier

```
humidity_classifier = DecisionTreeClassifier(max_leaf_nodes=10,random_state=0)
humidity_classifier.fit(X_train,y_train)
```

Out[29]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                      max_depth=None, max_features=None, max_leaf_nodes=1
0,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```

In [30]:

```
type(humidity_classifier)
```

Out[30]:

```
sklearn.tree._classes.DecisionTreeClassifier
```

## Predict on Test Set

Using humidity\_classifier we have predicted the value for the X\_test and stored it to y\_predicted

In [31]:

```
y_predicted = humidity_classifier.predict(X_test)
```

In [32]:

```
y_predicted[:10]
```

Out[32]:

```
array([0, 0, 1, 1, 1, 1, 1, 0, 1, 1])
```

In [33]:

```
y_test['high_humidity_label'][:10]
```

Out[33]:

```
456      0
845      0
693      1
259      1
723      1
224      1
300      1
442      0
585      1
1057     1
Name: high_humidity_label, dtype: int64
```

## Measure Accuracy of the Classifier

Checking our accuracy of the model using accuracy\_score function from sklearn metrics which in this case is with around 90% accuracy

In [34]:

```
accuracy_score(y_test,y_predicted)*100
```

Out[34]:

```
90.05681818181817
```



## 6. Covid-19 Analysis

- IBM CLOUD WATSON STUDIO

In [0]:

```
pip install plotly
```

Python interpreter will be restarted.

Collecting plotly

Downloading plotly-4.12.0-py2.py3-none-any.whl (13.1 MB)

Requirement already satisfied: six in /databricks/python3/lib/python3.7/site-packages (from plotly) (1.14.0)

Collecting retrying>=1.3.3

Downloading retrying-1.3.3.tar.gz (10 kB)

Building wheels for collected packages: retrying

Building wheel for retrying (setup.py): started

Building wheel for retrying (setup.py): finished with status 'done'

Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size=1143

0 sha256=528640eb6b7c1d585a2b8396b17ffb379afd31f68708f1cea0cd98d3125c9cfc

Stored in directory: /root/.cache/pip/wheels/f9/8d/8d/f6af3f7f9eea3553bc2fe6d53e4b287dad18b06a861ac56ddf

Successfully built retrying

Installing collected packages: retrying, plotly

Successfully installed plotly-4.12.0 retrying-1.3.3

Python interpreter will be restarted.

In [0]:

```
pip install jinja2
```

Python interpreter will be restarted.

Collecting jinja2

Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)

Collecting MarkupSafe>=0.23

Downloading MarkupSafe-1.1.1-cp37-cp37m-manylinux1\_x86\_64.whl (27 kB)

Installing collected packages: MarkupSafe, jinja2

Successfully installed MarkupSafe-1.1.1 jinja2-2.11.2

Python interpreter will be restarted.

In [0]:

```
import seaborn as sns
import plotly.express as px
%matplotlib inline
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

df = pd.read_csv("https://pandemicdatalake.blob.core.windows.net/public/raw/covid-19/ec
dc_cases/latest/ECDCCases.csv")
df.head(1000)
```

	dateRep	day	month	year	cases	deaths	countriesAndTerritories	geold	countryter
0	08/11/2020	8	11	2020	126	6	Afghanistan	AF	
1	07/11/2020	7	11	2020	58	2	Afghanistan	AF	
2	06/11/2020	6	11	2020	40	0	Afghanistan	AF	
3	05/11/2020	5	11	2020	121	6	Afghanistan	AF	
4	04/11/2020	4	11	2020	86	4	Afghanistan	AF	
...	...	...	...	...	...	...	...	...	...
995	24/06/2020	24	6	2020	0	0	Andorra	AD	
996	23/06/2020	23	6	2020	0	0	Andorra	AD	
997	22/06/2020	22	6	2020	0	0	Andorra	AD	
998	21/06/2020	21	6	2020	0	0	Andorra	AD	
999	20/06/2020	20	6	2020	0	0	Andorra	AD	

1000 rows × 12 columns

In [0]:

```
df.dtypes
```

```
Out[2]: dateRep          object
day                int64
month             int64
year              int64
cases             int64
deaths            int64
countriesAndTerritories object
geoId             object
countryterritoryCode object
popData2019       float64
continentExp       object
Cumulative_number_for_14_days_of_COVID-19_cases_per_100000 float64
dtype: object
```

In [0]:

```
df_1 = pd.read_csv("https://pandemicdatalake.blob.core.windows.net/public/raw/covid-19/ecdccases/latest/ECDCCases.csv")
df_1 = spark.createDataFrame(df_1)
```

In [0]:

```
df_1.write.mode("overwrite").saveAsTable("covidtable")
```

In [0]:

```
df.set_index('dateRep', inplace=True)
```

In [0]:

```
cv19_countries_day = df.groupby(by=['dateRep', 'countriesAndTerritories']).sum()[['cases', 'deaths']]

Total_confirmed = cv19_countries_day.groupby('dateRep').sum()[['cases', 'deaths']].sum()[['cases']]
Total_deaths = cv19_countries_day.groupby('dateRep').sum()[['cases', 'deaths']].sum()[['deaths']]

dicc = {'TotalConfirmed' : Total_confirmed, 'TotalDeaths' : Total_deaths, 'DeathRate' : round((Total_deaths/Total_confirmed)*100,2)}
total = pd.DataFrame(dicc, index=['Counter'])[['TotalConfirmed', 'TotalDeaths', 'DeathRate']]

total.style.set_properties(**{
    'background-color': 'white',
    'font-size': '20pt',
    'color' : 'red'
})
```

	TotalConfirmed	TotalDeaths	DeathRate
Counter	49945364	1250275	2.500000

In [0]:

```
total= spark.createDataFrame(total)
```

In [0]:

```
total.write.mode("overwrite").saveAsTable("Total_Covid_numbers")
```

*Visualization for total deaths and confirmed cases with timeline*

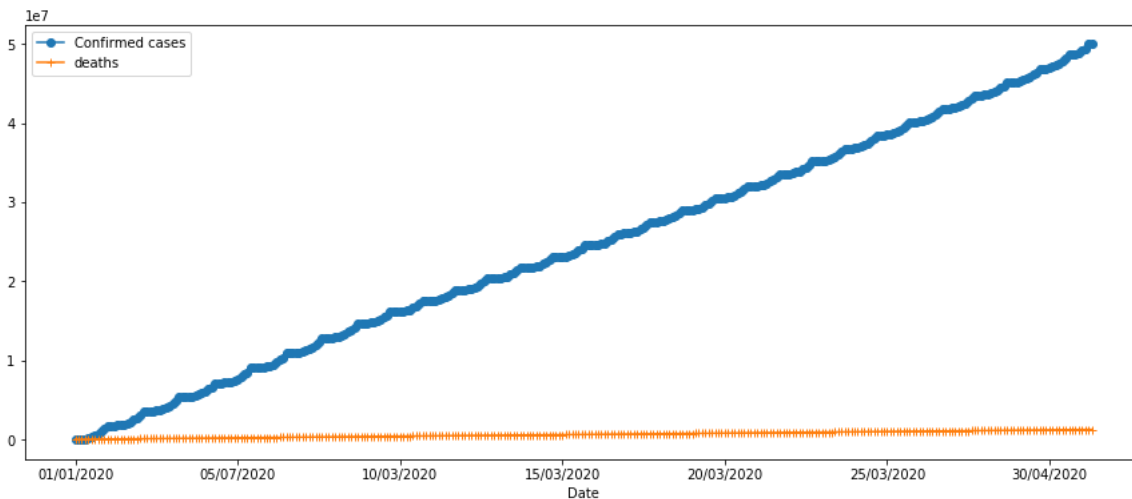
In [0]:

```
covid19_total = df[['countriesAndTerritories', 'cases']].groupby(by='countriesAndTerritories').sum().sort_values(by='cases', ascending=False)
covid19_total.columns=['cases']
covid19_total_d = df[['countriesAndTerritories', 'deaths']].groupby(by='countriesAndTerritories').sum().sort_values(by='deaths', ascending=False)
covid19_total_d.columns=['deaths']
```

In [0]:

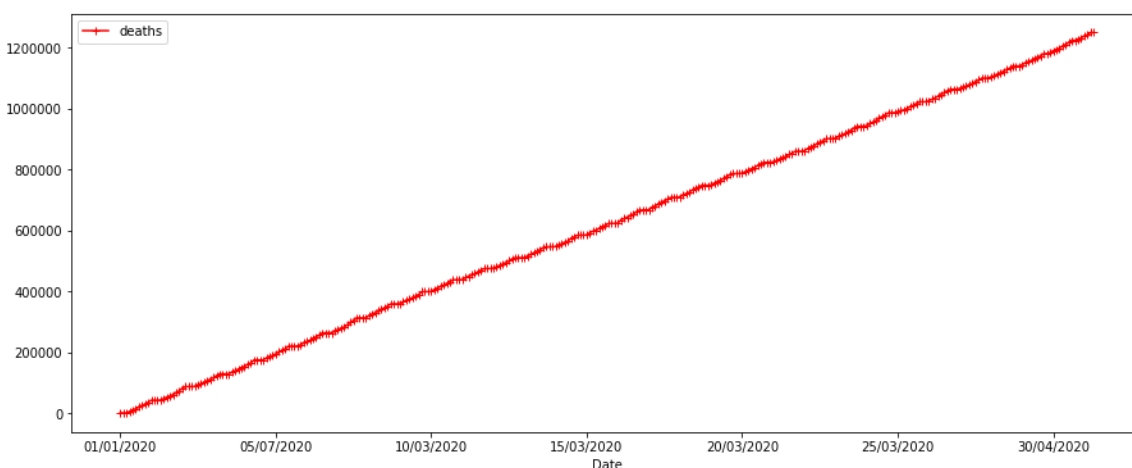
```
cv19_countries_day = df.groupby(by=['dateRep']).sum()[['cases', 'deaths']]
cv19_countries_day['cases'].cumsum().plot(figsize=(15,6),label='Confirmed cases',marker='o')
cv19_countries_day['deaths'].cumsum().plot(label="deaths",marker='+')
plt.legend()
plt.xlabel('Date')
plt.show()

#cv19_countries_day.sort_values(by='DateRep',ascending = False)
```



In [0]:

```
cv19_countries_day['deaths'].cumsum().plot(figsize=(15,6),color='red',label="deaths",marker='+')
plt.legend()
plt.xlabel('Date')
plt.show()
```



Now below we will create a table with with only the most latetst data of numbers for cases and deaths in respect to countries.

In [0]:

```
Todaynumbers=df.groupby('countriesAndTerritories').first().filter(['cases','deaths'])
```

In [0]:

```
Todaynumbers = spark.createDataFrame(Todaynumbers)
```

In [0]:

```
Todaynumbers.write.mode("overwrite").saveAsTable("Today_Covid_numbers")
```

In [0]:

```
#Top 10 cases
t10 = pd.concat([covid19_total.head(10),covid19_total_d.head(10)],axis=1,sort=False).head(10)
```

*Fatality Rate*

In [0]:

```
def highlight_max_yellow(s):
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]

def highlight_max(data, color='yellow'):
    attr = 'background-color: {}'.format(color)
    if data.ndim == 1: # Series from .apply(axis=0) or axis=1
        is_max = data == data.max()
        return [attr if v else '' for v in is_max]
    else: # from .apply(axis=None)
        is_max = data == data.max().max()
        return pd.DataFrame(np.where(is_max, attr, ''),
                             index=data.index, columns=data.columns)

def highlight_max_all(s):
    is_max = s == s.max()
    return ['background-color: #f59d71' if v else '' for v in is_max]

def highlight_min(data):
    color_min= '#b5f5d4' #green
    attr = 'background-color: {}'.format(color_min)

    if data.ndim == 1: # Series from .apply(axis=0) or axis=1
        is_min = data == data.min()
        return [attr if v else '' for v in is_min]
    else:
        is_min = data.groupby(level=0).transform('min') == data
        return pd.DataFrame(np.where(is_min, attr, ''),
                             index=data.index, columns=data.columns)
```

In [0]:

```
t10['DeathRatio'] = round((t10['deaths'] / t10['cases']) *100,2)
t10.sort_values(by='DeathRatio',ascending = False)
t10f = t10[['DeathRatio']].sort_values(by='DeathRatio',ascending=False).dropna()
t10f.style.apply(highlight_max, color='red', axis=None)
```

	DeathRatio
Mexico	9.860000
United_Kingdom	4.170000
Spain	2.920000
Brazil	2.880000
United_States_of_America	2.400000
France	2.300000
India	1.480000

In [0]:

```
t10f = spark.createDataFrame(t10f)
```

In [0]:

```
t10f.write.mode("overwrite").saveAsTable("fatility_covid")
```

## CHANGE COVID RATE

In [0]:

```
covid19_change_global = cv19_countries_day.cumsum()
covid19_change_global[['Cases Day', 'Deaths Day']] = cv19_countries_day[['cases', 'deaths']]
covid19_change_global = covid19_change_global.pct_change(1)
covid19_change_global = covid19_change_global.sort_values(by='dateRep', ascending=False)
covid19_change_global = covid19_change_global.replace([np.inf, -np.inf], np.nan)
covid19_change_global = covid19_change_global.fillna(0)
covid19_change_global = round(covid19_change_global*100,2)
covid19_change_global = covid19_change_global.reset_index()

covid19_change_global_d = cv19_countries_day.cumsum()
covid19_change_global_d[['Cases Day', 'Deaths Day']] = cv19_countries_day[['cases', 'deaths']]
covid19_change_global_d = covid19_change_global_d.pct_change(1)
covid19_change_global_d = covid19_change_global_d.sort_values(by='dateRep', ascending=False)
covid19_change_global_d = covid19_change_global_d.replace([np.inf, -np.inf], np.nan)
covid19_change_global_d = covid19_change_global_d.fillna(0)
covid19_change_global_d = round(covid19_change_global_d*100,2)
covid19_change_global_d = covid19_change_global_d.reset_index()
```

In [0]:

```
px.bar(data_frame=covid19_change_global,x=covid19_change_global['dateRep'],y=covid19_change_global['cases'], \
       color='cases', \
       labels={'Cases':'Date','Deaths':'% change'}, \
       title='Cases: Global change percentage per day')
```

In [0]:

```
px.bar(data_frame=covid19_change_global_d,x=covid19_change_global_d['dateRep'],y=covid19_change_global_d['deaths'], \
       color='deaths', \
       labels={'DateRep':'Date','Deaths':'% change'}, \
       title='Deaths: Global change percentage per day')
```

In [0]:

```
#Calculate change by country for the 15 first

Impacted_countries = df[['countriesAndTerritories','cases']].sort_values(by=['dateRep',
'cases'],ascending=False).head(15)['countriesAndTerritories']
Impacted_countries

top_impact = pd.DataFrame()

for country in Impacted_countries:
    top_impact[country] = df[df['countriesAndTerritories']==country]['cases']

top_impact = top_impact.reset_index().sort_values(by='dateRep',ascending=True) #true

#Normalize

#top_impact_norm = top_impact/top_impact.iloc[0] * 100
```

(Cases) Growth by time period (%)  $C(t)=C(t-1)*\Delta$

In [0]:

```
#Australia 0 cases correction (mean between days 25.03 and 27.03)
#top_impact['Australia'][1] = 671.5

growth_impact_day = top_impact.set_index('dateRep').pct_change(1).reset_index().sort_
values(by='dateRep',ascending=False)
growth_impact_day = round(growth_impact_day.set_index('dateRep')*100,2).head(10)
growth_impact_day.style.apply(highlight_max_all).apply(highlight_min)
```

	China	Afghanistan	Algeria	Armenia	Australia	Austria
dateRep						
31/12/2019	-18.180000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000
31/10/2020	3.120000	726.320000	-12.360000	1820.160000	-89.430000	1765.840000
31/08/2020	-88.410000	-73.240000	-39.530000	-64.970000	-82.940000	72.390000
31/07/2020	5420.000000	-91.800000	352.630000	-0.280000	5908.330000	270.450000
31/05/2020	-94.050000	3107.410000	82.190000	610.000000	-97.410000	-94.530000
31/03/2020	-95.760000	inf	inf	inf	46300.000000	inf
31/01/2020	7820.000000	-100.000000	-100.000000	-100.000000	-93.330000	-100.000000
30/10/2020	8.700000	720.000000	97.420000	633.330000	-21.050000	496.260000
30/09/2020	-14.810000	400.000000	-59.100000	86.860000	-80.810000	314.940000
30/08/2020	-87.890000	inf	-38.270000	-32.430000	-64.390000	6.100000

In [0]:

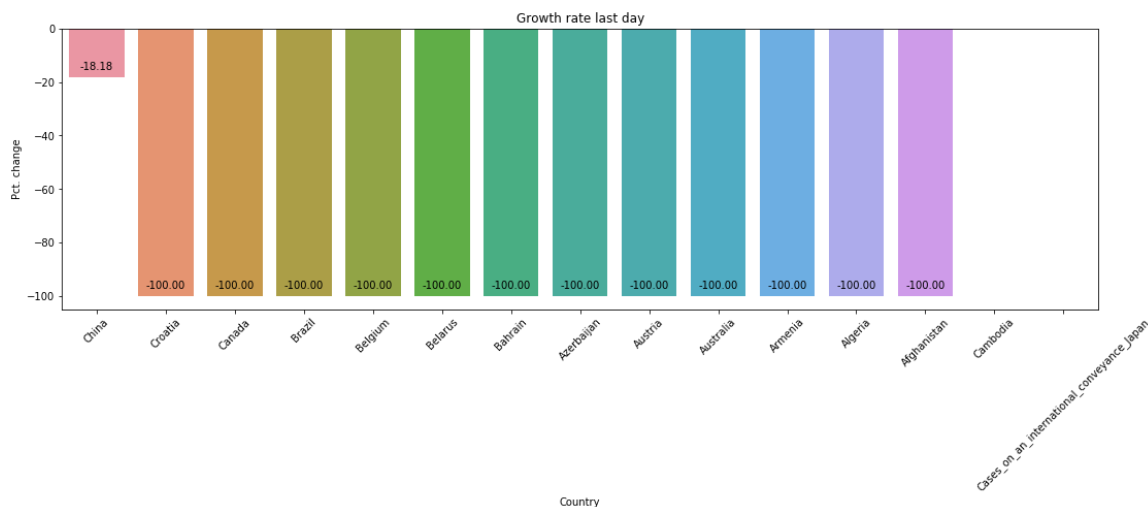
```
import seaborn as sns
```



In [0]:

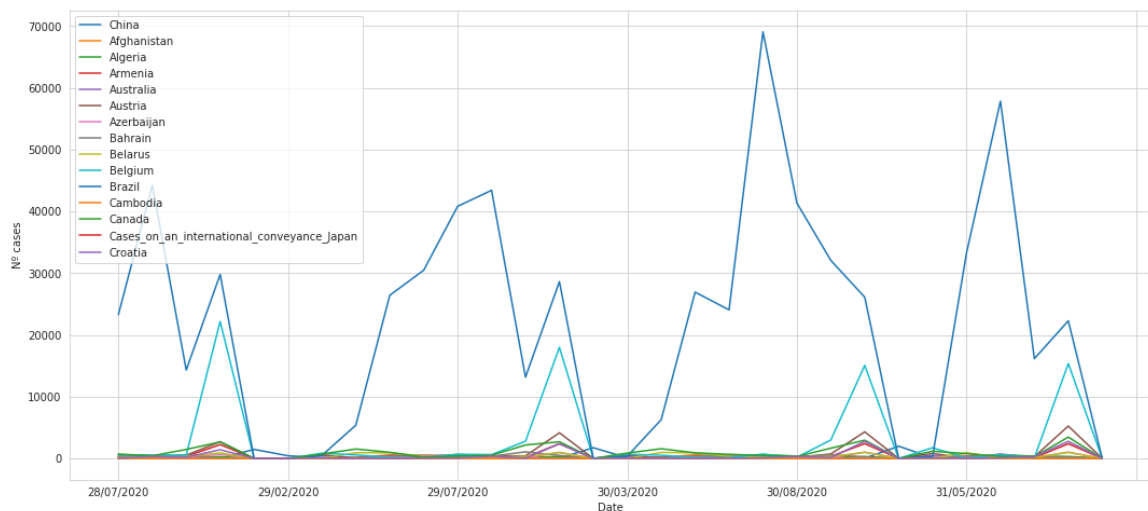
```
last_gi = growth_impact_day.iloc[0].sort_values(ascending = False)

last_gi = pd.DataFrame(data=[last_gi],index=[0],columns=last_gi.index)
plt.figure(figsize=(18,5))
splot = sns.barplot(x='index',y=0,data=last_gi.T.reset_index())
plt.title('Growth rate last day')
plt.ylabel('Pct. change')
plt.xlabel('Country')
plt.xticks(rotation=45)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
plt.show()
```



In [0]:

```
sns.set_style('whitegrid')
top_impact.set_index('dateRep').tail(30).plot(figsize=(18,8))
plt.ylabel('N° cases')
plt.xlabel('Date')
plt.show()
```



(Deaths) Growth per day time period (%)  $D(t)=D(t-1)*\Delta$

In [0]:

```
Impacted_countries_d = df[['countriesAndTerritories','deaths']].sort_values(by=['deaths'],ascending=True).head(15)['countriesAndTerritories']

top_impact_d = pd.DataFrame()

for country in Impacted_countries_d:
    top_impact_d[country] = df[df['countriesAndTerritories']==country]['deaths']

top_impact_d = top_impact_d.reset_index().sort_values(by='dateRep',ascending=False)

#Normalize

#top_impact_norm = top_impact/top_impact.iloc[0] * 100
```

In [0]:

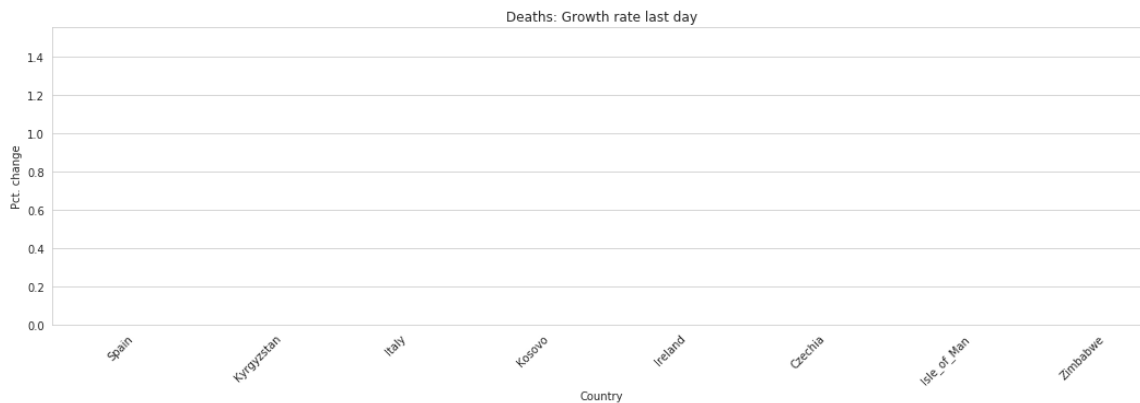
```
growth_impact_d_day = top_impact_d.set_index('dateRep').pct_change(1).reset_index().sort_values(by='dateRep',ascending=True)
growth_impact_d_day = round(growth_impact_d_day.set_index('dateRep')*100,2).head(10)
growth_impact_d_day.style.apply(highlight_max_all).apply(highlight_min)
```

	Spain	Kyrgyzstan	Italy	Kosovo	Ireland	Czechia	Isle
dateRep							
01/01/2020	nan	nan	nan	nan	nan	nan	
01/02/2020	nan	nan	-100.000000	nan	nan	nan	
01/03/2020	-100.000000	nan	-99.050000	nan	-100.000000	-100.000000	
01/04/2020	202.140000	nan	194.390000	-100.000000	-59.520000	-22.220000	
01/05/2020	inf	nan	280.000000	inf	inf	800.000000	
01/06/2020	-100.000000	-100.000000	226.090000	-100.000000	-100.000000	0.000000	
01/07/2020	inf	-73.680000	155.560000	-86.670000	inf	-66.670000	
01/08/2020	-100.000000	1800.000000	50.000000	66.670000	nan	200.000000	
01/09/2020	-68.130000	0.000000	-68.420000	inf	-100.000000	-94.740000	
01/10/2020	inf	-80.000000	-93.600000	-100.000000	-80.000000	-89.020000	

In [0]:

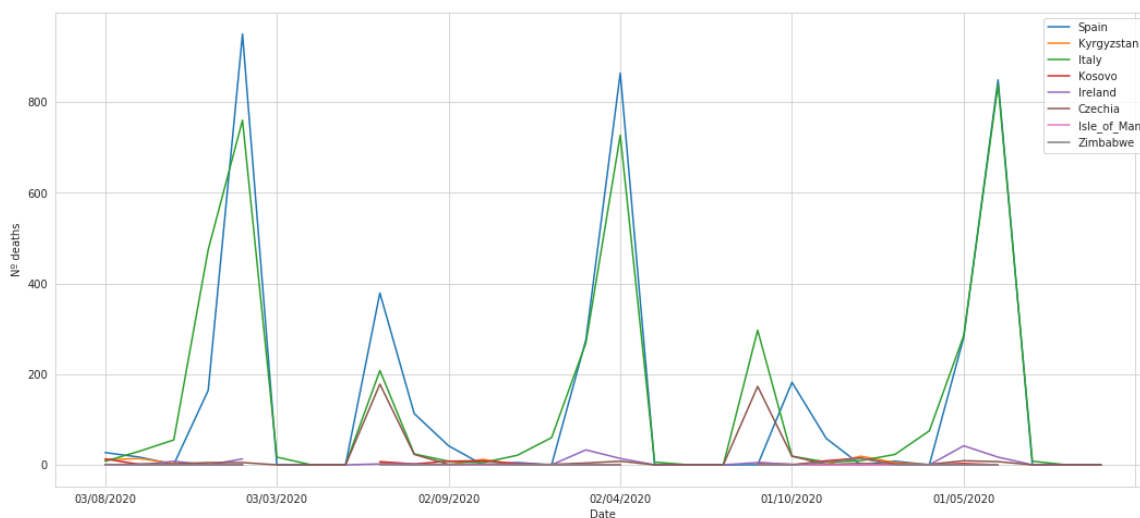
```
last_gi = growth_impact_d_day.iloc[0].sort_values(ascending = False)

last_gi = pd.DataFrame(data=[last_gi],index=[0],columns=last_gi.index)
plt.figure(figsize=(18,5))
splot = sns.barplot(x='index',y=0,data=last_gi.T.reset_index())
plt.title('Deaths: Growth rate last day')
plt.ylabel('Pct. change')
plt.xlabel('Country')
plt.xticks(rotation=45)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
plt.show()
```



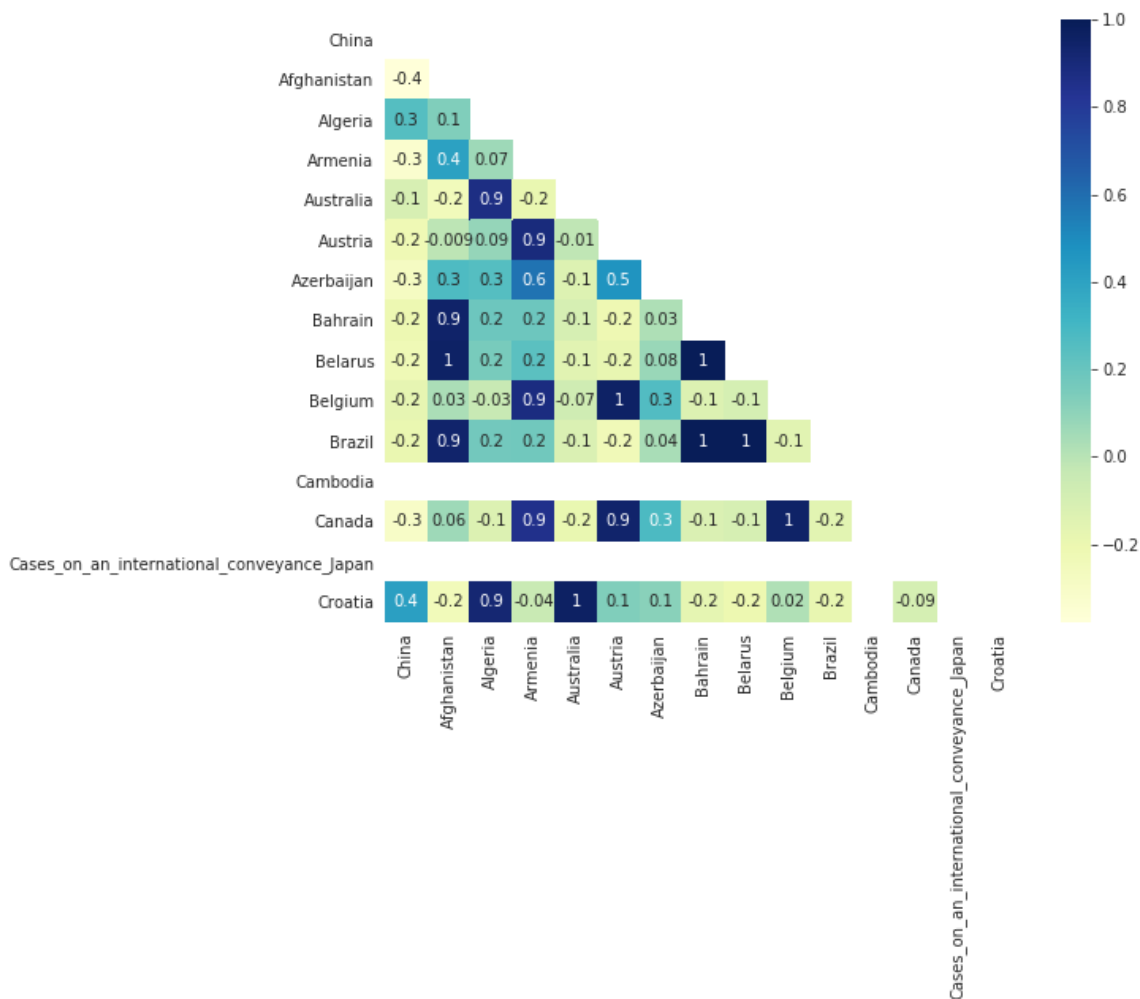
In [0]:

```
sns.set_style('whitegrid')
top_impact_d.set_index('dateRep').tail(30).plot(figsize=(18,8))
plt.ylabel('Nº deaths')
plt.xlabel('Date')
plt.show()
```



In [0]:

```
mask = np.zeros_like(growth_impact_day.corr())
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(9,7))
sns.heatmap(growth_impact_day.corr(),mask=mask,cmap='YlGnBu', annot = True, fmt='.1g')
plt.show()
```



In [0]:

```
India_df = df[df['countriesAndTerritories']=='India']
Brazil_df = df[df['countriesAndTerritories']=='Brazil']
USA_df = df[df['countriesAndTerritories']=='United_States_of_America']

India_df = India_df.sort_values(by='dateRep',ascending=True)
India_df['Cases-5-days-SMA']=India_df['cases'].rolling(window=5).mean()
India_df['Deaths-5-days_SMA']=India_df['deaths'].rolling(window=5).mean()
India_df = India_df.sort_values(by='dateRep',ascending=False)

Brazil_df = Brazil_df.sort_values(by='dateRep',ascending=True)
Brazil_df['Cases-5-days-SMA']=Brazil_df['cases'].rolling(window=5).mean()
Brazil_df['Deaths-5-days_SMA']=Brazil_df['deaths'].rolling(window=5).mean()
Brazil_df = Brazil_df.sort_values(by='dateRep',ascending=False)

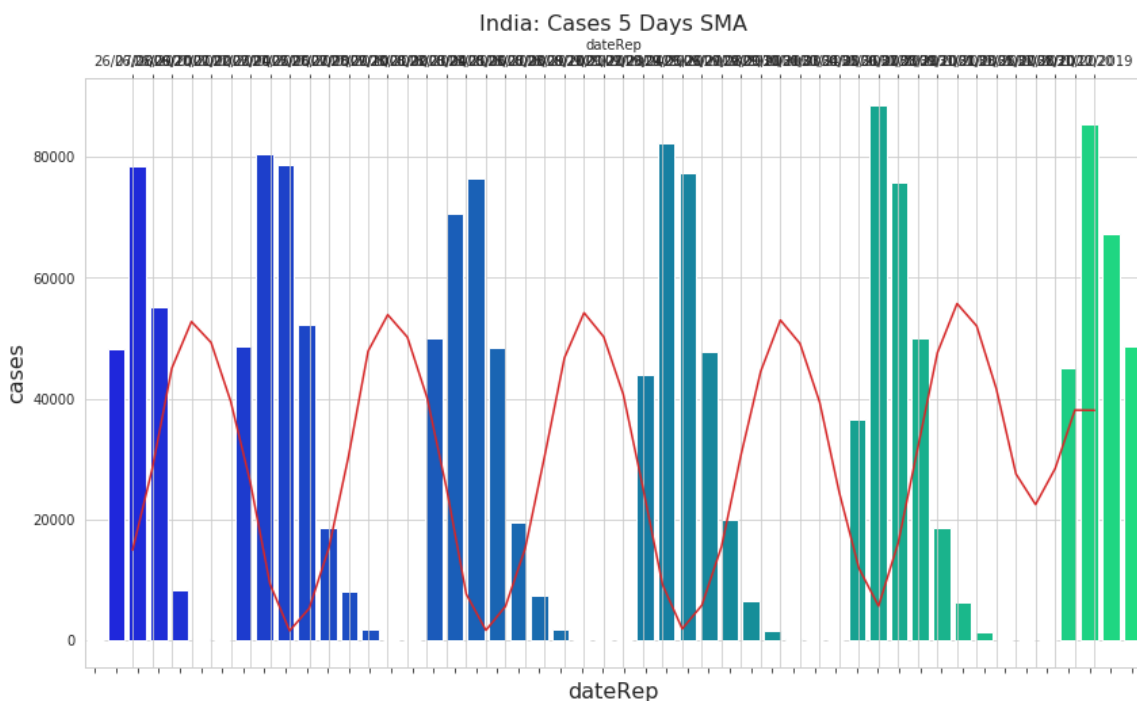
USA_df = USA_df.sort_values(by='dateRep',ascending=True)
USA_df['Cases-5-days-SMA']=USA_df['cases'].rolling(window=5).mean()
USA_df['Deaths-5-days_SMA']=USA_df['deaths'].rolling(window=5).mean()
USA_df =USA_df.sort_values(by='dateRep',ascending=False)
```

In [0]:

```
#Create combo chart
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('India: Cases 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('Cases', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='cases', data = India_df.reset_index()[:50], palette=
'winter')

ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')

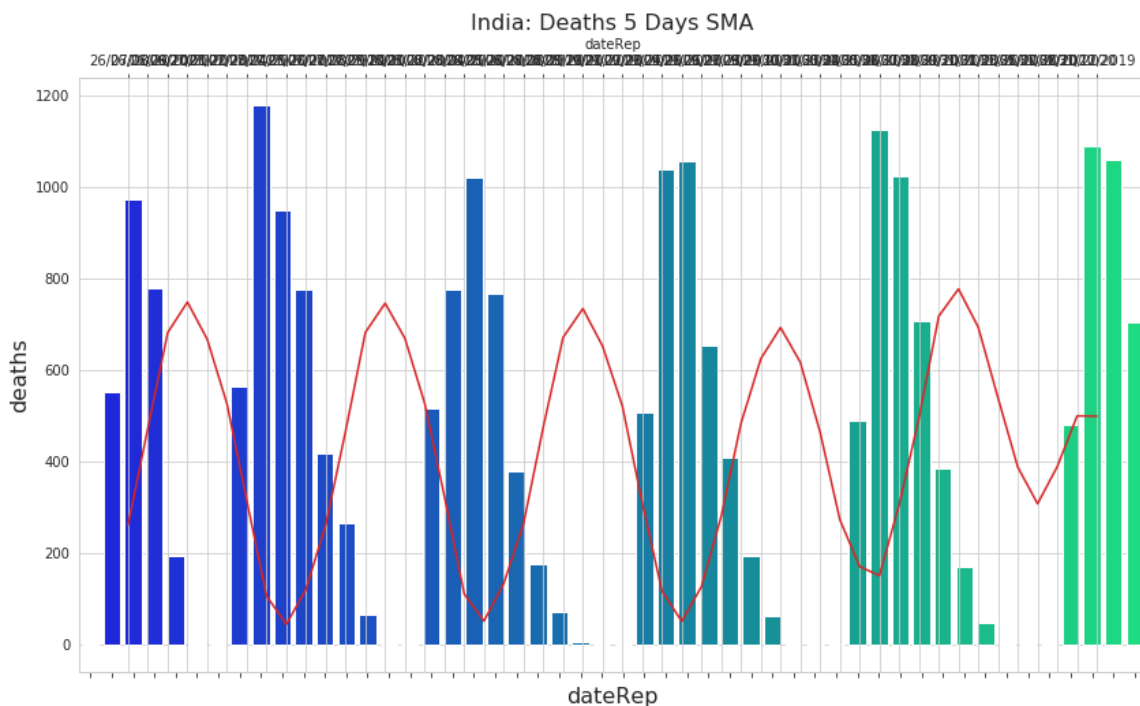
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#Line plot creation
#ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Cases-5-days-SMA', data=India_df.reset_index()[:50],
color=color)
ax2.tick_params(axis='y', color=color)
plt.show()
```



In [0]:

```
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('India: Deaths 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('deaths', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='deaths', data = India_df.reset_index()[:50], palette=
'winter')

ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#line plot creation
ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Deaths-5-days_SMA', data = India_df.reset_index()[:50], color=color)
ax2.tick_params(axis='y', color=color)
#show plot
plt.show()
```

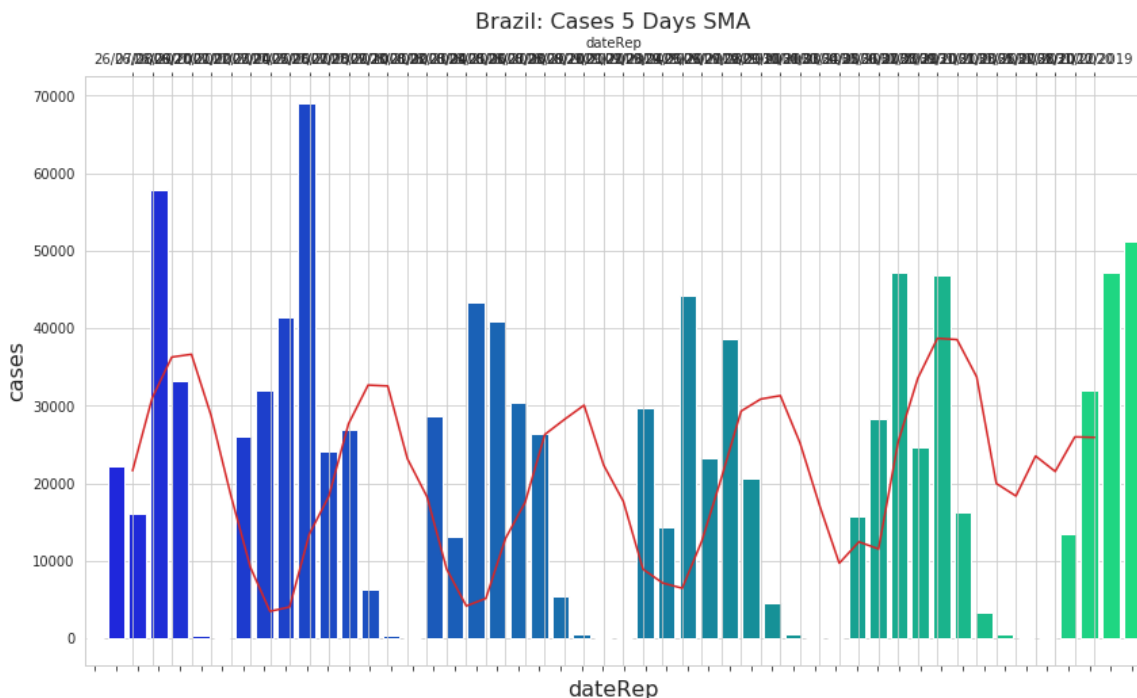


In [0]:

```
#Create combo chart
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('Brazil: Cases 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('Cases', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='cases', data = Brazil_df.reset_index()[:50], palette=
'winter')

ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')

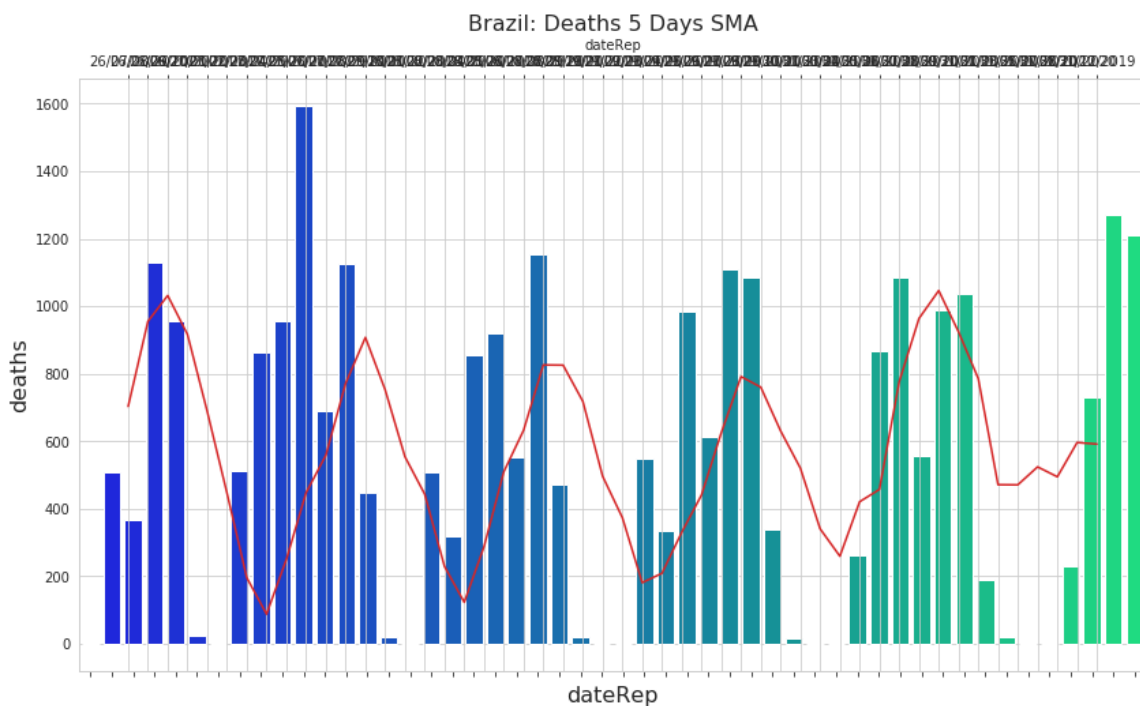
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#line plot creation
#ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Cases-5-days-SMA', data=Brazil_df.reset_index()[:50
], color=color)
ax2.tick_params(axis='y', color=color)
plt.show()
```



In [0]:

```
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('Brazil: Deaths 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('deaths', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='deaths', data = Brazil_df.reset_index()[:50], palette
='winter')

ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#line plot creation
ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Deaths-5-days_SMA', data = Brazil_df.reset_index()[:
50], color=color)
ax2.tick_params(axis='y', color=color)
#show plot
plt.show()
```

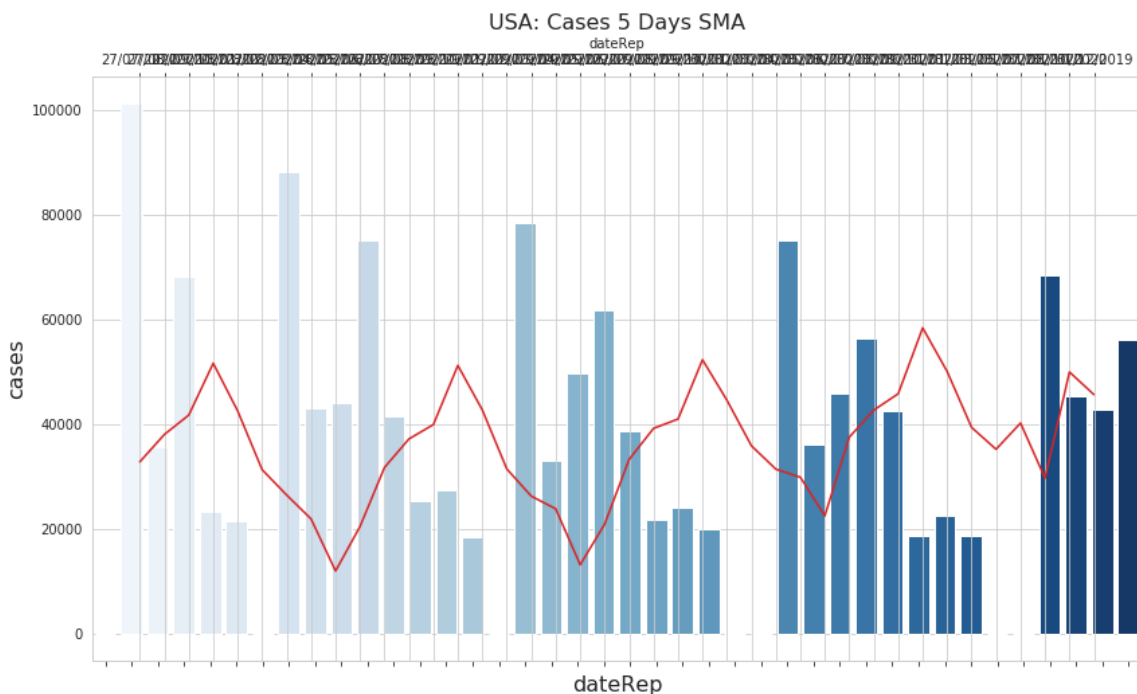




In [0]:

```
#Create combo chart
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('USA: Cases 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('Cases', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='cases', data = USA_df.reset_index()[:40], palette='Blues')

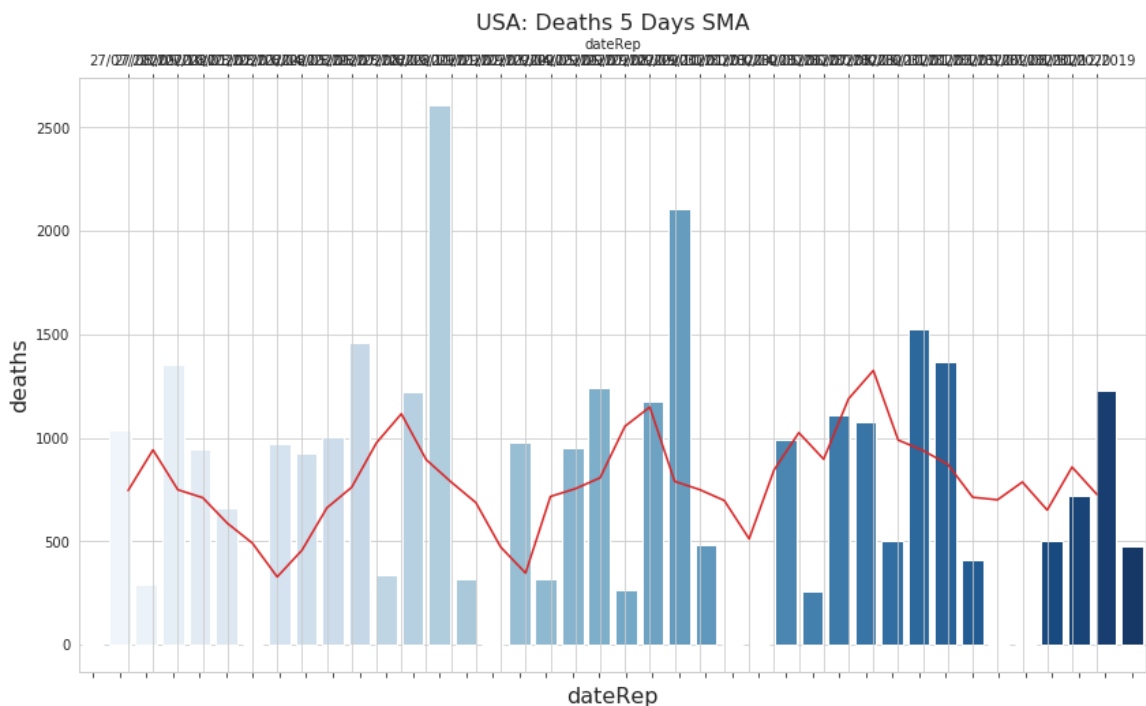
ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#line plot creation
ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Cases-5-days-SMA', data = USA_df.reset_index()[:40],
color=color)
ax2.tick_params(axis='y', color=color)
#show plot
plt.show()
```



In [0]:

```
#Create combo chart
fig, ax1 = plt.subplots(figsize=(14,8))
color = 'tab:green'
#bar plot creation
ax1.set_title('USA: Deaths 5 Days SMA', fontsize=16)
ax1.set_xlabel('Date', fontsize=16)
ax1.set_ylabel('Deaths', fontsize=16)
ax1 = sns.barplot(x='dateRep', y='deaths', data = USA_df.reset_index()[:40], palette='Blues')

ax1.set_xticklabels(
    ax1.get_xticklabels(minor=True),
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-large'
)
ax1.tick_params(axis='y')
#specify we want to share the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'
#line plot creation
ax2.set_ylabel('5 days SMA', fontsize=16)
ax2 = sns.lineplot(x='dateRep', y='Deaths-5-days_SMA', data = USA_df.reset_index()[:40], color=color)
ax2.tick_params(axis='y', color=color)
#show plot
plt.show()
```



In [0]:

```
India_df =spark.createDataFrame(India_df)
USA_df =spark.createDataFrame(USA_df)
Brazil_df =spark.createDataFrame(Brazil_df)
```

In [0]:

```
India_df.write.mode("overwrite").saveAsTable("India_Covid")
USA_df.write.mode("overwrite").saveAsTable("USA_Covid")
Brazil_df.write.mode("overwrite").saveAsTable("Brazil_Covid")
```

Top Countries

In [0]:

```
import plotly.graph_objects as go
import plotly.express as px
import matplotlib.pyplot as plt

df.loc[:, ['countriesAndTerritories', 'cases', 'deaths']].groupby(['countriesAndTerritories']).max().sort_values(by='cases', ascending=False).reset_index()[0:15].style.background_gradient(cmap='rainbow')
```

	countriesAndTerritories	cases	deaths
0	United_States_of_America	130623	4928
1	India	97894	2003
2	France	86852	2004
3	Brazil	69074	1595
4	Spain	55019	1623
5	Italy	39809	971
6	Chile	36179	1057
7	Poland	27875	445
8	United_Kingdom	26687	1224
9	Germany	23399	315
10	Belgium	22176	321
11	Switzerland	21842	93
12	Russia	20582	389
13	Kazakhstan	19246	324
14	Argentina	18326	3351

In [0]:

```
top_countries = df.loc[:, ['countriesAndTerritories', 'cases', 'deaths']].groupby(['countriesAndTerritories']).max().sort_values(by='cases', ascending=False).reset_index()
```

In [0]:

```
display(top_countries)
```

countriesAndTerritories	cases	deaths
United_States_of_America	130623	4928
India	97894	2003
France	86852	2004
Brazil	69074	1595
Spain	55019	1623
Italy	39809	971
Chile	36179	1057
Poland	27875	445
United_Kingdom	26687	1224
Germany	23399	315
Belgium	22176	321

In [0]:

```
top_countries =spark.createDataFrame(top_countries)
```

In [0]:

```
top_countries.write.mode("overwrite").saveAsTable("top_countries")
```

*Saving main dataframe into a table.*

## AFRICA

In [0]:

```
africa_df = df[df['continentExp']=='Africa']
```

In [0]:

```
display(africa_df)
```

day	month	year	cases	deaths	countriesAndTerritories	geold	countryterritoryCode	pc
8	11	2020	581	12	Algeria	DZ	DZA	4
7	11	2020	1273	25	Algeria	DZ	DZA	4
6	11	2020	0	0	Algeria	DZ	DZA	4
5	11	2020	548	10	Algeria	DZ	DZA	4
4	11	2020	405	9	Algeria	DZ	DZA	4
3	11	2020	302	7	Algeria	DZ	DZA	4
2	11	2020	330	9	Algeria	DZ	DZA	4
1	11	2020	291	8	Algeria	DZ	DZA	4
31	10	2020	319	7	Algeria	DZ	DZA	4

In [0]:

```
africa_df = spark.createDataFrame(africa_df)
```

In [0]:

```
africa_df.write.mode("overwrite").saveAsTable("Africa_covidtable")
```

## ASIA

In [0]:

```
asia_df = df[df['continentExp']=='Asia']  
display(asia_df)
```

day	month	year	cases	deaths	countriesAndTerritories	geold	countryterritoryCode	p
8	11	2020	126	6	Afghanistan	AF	AFG	3
7	11	2020	58	2	Afghanistan	AF	AFG	3
6	11	2020	40	0	Afghanistan	AF	AFG	3
5	11	2020	121	6	Afghanistan	AF	AFG	3
4	11	2020	86	4	Afghanistan	AF	AFG	3
3	11	2020	95	3	Afghanistan	AF	AFG	3
2	11	2020	132	5	Afghanistan	AF	AFG	3
1	11	2020	76	0	Afghanistan	AF	AFG	3
31	10	2020	157	4	Afghanistan	AF	AFG	3

In [0]:

```
asia_df = spark.createDataFrame(asia_df)
```

In [0]:

```
asia_df.write.mode("overwrite").saveAsTable("Asia_covidtable")
```

## Europe

In [0]:

```
europe_df = df[df['continentExp']=='Europe']  
display(europe_df)
```

day	month	year	cases	deaths	countriesAndTerritories	geold	countryterritoryCode	pc
8	11	2020	495	8	Albania	AL	ALB	
7	11	2020	489	6	Albania	AL	ALB	
6	11	2020	421	7	Albania	AL	ALB	
5	11	2020	396	4	Albania	AL	ALB	
4	11	2020	381	5	Albania	AL	ALB	
3	11	2020	321	9	Albania	AL	ALB	
2	11	2020	327	9	Albania	AL	ALB	
1	11	2020	241	7	Albania	AL	ALB	
31	10	2020	319	3	Albania	AL	ALB	

In [0]:

```
europe_df = spark.createDataFrame(europe_df)
```

In [0]:

```
europe_df.write.mode("overwrite").saveAsTable("Europe_covidtable")
```

## Oceania

In [0]:

```
Oceania_df = df[df['continentExp']=='Oceania']  
display(Oceania_df)
```

day	month	year	cases	deaths	countriesAndTerritories	geold	countryterritoryCode	pc
8	11	2020	7	0	Australia	AU	AUS	
7	11	2020	12	0	Australia	AU	AUS	
6	11	2020	11	0	Australia	AU	AUS	
5	11	2020	12	0	Australia	AU	AUS	
4	11	2020	8	0	Australia	AU	AUS	
3	11	2020	7	0	Australia	AU	AUS	
2	11	2020	5	0	Australia	AU	AUS	
1	11	2020	8	0	Australia	AU	AUS	
31	10	2020	13	0	Australia	AU	AUS	

In [0]:

```
Oceania_df = spark.createDataFrame(Oceania_df)
```

In [0]:

```
Oceania_df.write.mode("overwrite").saveAsTable("Oceania_covidtable")
```

## America

In [0]:

```
America_df = df[df['continentExp']=='America']  
display(America_df)
```

day	month	year	cases	deaths	countriesAndTerritories	geold	countryterritoryCode	pc
8	11	2020	0	0	Anguilla	AI	AIA	
7	11	2020	0	0	Anguilla	AI	AIA	
6	11	2020	0	0	Anguilla	AI	AIA	
5	11	2020	0	0	Anguilla	AI	AIA	
4	11	2020	0	0	Anguilla	AI	AIA	
3	11	2020	0	0	Anguilla	AI	AIA	
2	11	2020	0	0	Anguilla	AI	AIA	
1	11	2020	0	0	Anguilla	AI	AIA	
31	10	2020	0	0	Anguilla	AI	AIA	

In [0]:

```
America_df = spark.createDataFrame(America_df)
```

In [0]:

```
America_df.write.mode("overwrite").saveAsTable("America_covidtable")
```

In [4]:

```
print("VINITA VERMA - 06917704418" )  
print("MCA -5B")
```

```
VINITA VERMA - 06917704418  
MCA -5B
```

In [ ]:

In [ ]:



## 7. Multiple Regression

- Multiple linear regression is the most common form of linear regression analysis.
- As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables.
- The independent variables can be continuous or categorical (dummy coded as appropriate).

**Problem Statement:** We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

### Import the libraries

In [ ]:

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

In [ ]:

```
df = pandas.read_csv("cars.csv")
```

Then make a list of the independent values and call this variable X.

Put the dependent values in a variable called y.

In [ ]:

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

In [ ]:

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Out[ ]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

In [ ]:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])
```

In [ ]:

```
print(predictedCO2)
```

[107.2087328]

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO<sub>2</sub> for every kilometer it drives.

In [ ]:

```
print(regr.coef_)
```

[0.00755095 0.00780526]

### Conclusion :

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095 Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO<sub>2</sub> emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm<sup>3</sup>, the CO<sub>2</sub> emission increases by 0.00780526 g.

We have already predicted that if a car with a 1300cm<sup>3</sup> engine weighs 2300kg, the CO<sub>2</sub> emission will be approximately 107g.

In [ ]:

## 8. Logistic Regression

- Logistic Regression is an example of a classification algorithm which is used to find a relationship between features and probability of a particular outcome.
- Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y).
- A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.
- logistic regression equation:

$$y = e^{(b_0 + b_1x)} / (1 + e^{(b_0 + b_1x)})$$

**Problem Statement** - To predict whether a person will buy a car (1) or (0)

### Dataset description

We have a Data set having 5 columns namely: User ID, Gender, Age, EstimatedSalary and Purchased. Now we have to build a model that can predict whether on the given parameter a person will buy a car or not.

## Importing the libraries

In [ ]:

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

## Importing the Data set

In [ ]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

In [ ]:

```
dataset.shape
```

Out[ ]:

```
(400, 5)
```

In [ ]:

```
dataset.head(5)
```

Out[ ]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

## Train- Test split

Splitting our Data set in Dependent and Independent variables.

In our Data set we'll consider Age and EstimatedSalary as Independent variable and Purchased as Dependent Variable.

In [ ]:

```
X = dataset.iloc[:, [2,3]].values  
y = dataset.iloc[:, 4].values
```

Here X is Independent variable and y is Dependent variable. Logistic model and Test data will be used to validate our model. We'll use Sklearn to split our data. We'll import train\_test\_split from sklearn.model\_selection

In [ ]:

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

## Feature Scaling

we'll do feature scaling to scale our data between 0 and 1 to get better accuracy. Here Scaling is important because there is a huge difference between Age and EstimatedSalary.

- Import StandardScaler from sklearn.preprocessing
- Then make an instance sc\_X of the object StandardScaler
- Then fit and transform X\_train and transform X\_test

In [ ]:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

## Fitting Logistic Regression to the Training Set

In [ ]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

Out[ ]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

## Predicting the Test set results

In [ ]:

```
y_pred = classifier.predict(X_test)
```

In [ ]:

```
y_pred
```

Out[ ]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
        0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
        1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1])
```

## Making the confusion matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [ ]:

```
cm
```

Out[ ]:

```
array([[65,  3],
       [ 8, 24]])
```

In [ ]:

```
import sklearn  
sklearn.metrics.accuracy_score(y_test, y_pred)*100
```

Out[ ]:

89.0

In [1]:

```
print("VINITA VERMA ")  
print("MCA-5B")  
print("06917704418")
```

VINITA VERMA  
MCA-5B  
06917704418

In [ ]: