Homework 1

Through this homework, you'll practice the basics of data cleaning, data partition, data normalization, and data visualization.

Please enter the code along with your comments in the **TODO** sections.

Please refer to the **Hint** section if you do not know where to start.

Alternative solutions are totally welcomed.

Part 1: Data cleaning and pre-processing

Problem 1 (25 points)

Glass Identification Data

Source: https://archive.ics.uci.edu/ml/datasets/glass+identification

Creator: B. German

```
Central Research Establishment
```

```
Home Office Forensic Science Service
```

```
Aldermaston, Reading, Berkshire RG7 4PN
```

Donor: Vina Spiehler, Ph.D., DABFT

```
Diagnostic Products Corporation
```

```
!pip install --upgrade openpyxl
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: openpyxl in /usr/local/lib/python3.8/dist-packages (3.1.0)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.8/dist-packages (from openpyxl) (1.1.0)
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

```
from google.colab import files
file = files.upload()  #upload file into google colab session
df = pd.read_excel("Glass_Identification_Data.xlsx")
df.head()
```

Choose Files Glass_Iden…n_Data.xlsx

- **Glass_Identification_Data.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 24268 bytes, last modified: 1/24/2023 - 100% done

```
Saving Glass_Identification_Data.xlsx to Glass_Identification_Data (1).xlsx
```

	ID	RI	Na	Mg	Al	Si	K	CA	Ba	Fe	Class
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	NaN	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	NaN	0.0	1
2	3	NaN	13.53	3.55	1.54	72.99	0.39	7.78	NaN	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	NaN	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	NaN	0.0	1

TODO1:

- Count the the **percentage** of null/missing values for each variable

- Drop the variables which have more than 75% missing values *(Avoid manual intervention. Code should work even if the attribute/data changes)*

```
#finding the missing/null values percentage for every column
percent_missing = df.isnull().sum()/len(df)*100
print(percent_missing)      #printing the missing percent in each column
```

```
ID        0.000000
RI        0.934579
Na        0.934579
Mg        7.943925
Al        0.000000
Si        0.467290
K         2.336449
CA        0.000000
Ba       78.037383
Fe        0.000000
Class     0.000000
dtype: float64
```

```
#dropping the columns which have total null values greater than 75
df = df.dropna(thresh=len(df)*0.75, axis = 1)
```

```
df
```

	ID	RI	Na	Mg	Al	Si	K	CA	Fe	Class
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	1
2	3	NaN	13.53	3.55	1.54	72.99	0.39	7.78	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	0.0	7
213	214	1.51711	14.23	0.00	2.08	NaN	0.00	8.62	0.0	7

214 rows × 10 columns

Hint:

Handle missing data in Python

dropna() thresh option

Note: You can try other methods as well apart from the ones mentioned in the hint

TODO2:

- If a variable contains more than 10 missing records, impute the records by using the mean value of records from the respective class instead of using the mean value of the entire column. *(Avoid manual intervention. Code should work even if the attribute/data changes)*
- If a variable contains less than 10 missing records, impute the records with the previous non-NAN value from a row with the same 'Class' *(Avoid manual intervention. Code should work even if the attribute/data changes)*
- What is imputation in Data Mining?

```
#finding the columns where null values are present and forming a dataframe
cols_with_nulls = df.columns[df.isnull().sum() > 0]
df_with_nulls = df[cols_with_nulls]
df_with_nulls                        #displaying the new dataframe where each column has null values
```

	RI	Na	Mg	Si	K
0	1.52101	13.64	4.49	71.78	0.06
1	1.51761	13.89	3.60	72.73	0.48
2	NaN	13.53	3.55	72.99	0.39
3	1.51766	13.21	3.69	72.61	0.57
4	1.51742	13.27	3.62	73.08	0.55
...	...	...	...	...	...
209	1.51623	14.14	0.00	72.61	0.08
210	1.51685	14.92	0.00	73.06	0.00
211	1.52065	14.36	0.00	73.42	0.00
212	1.51651	14.38	0.00	73.61	0.00

```
#finding the null values total for each column correspondingly
df.isnull().sum()
```

```
ID        0
RI        2
Na        2
Mg       17
Al        0
Si        1
K         5
CA        0
Fe        0
Class     0
dtype: int64
```

```
#imputing the column which has greater than 10 empty values with mean based on their respective classes
class_column = "Class"     #declaring the class_column
cols_to_impute = df.columns[df.isnull().sum()>10]    #considering the columns which has null values more than 10
for col in cols_to_impute:
  df[col] = df.groupby(class_column)[col].transform(lambda x: x.fillna(x.mean()))    #filling the null values with mean by grouping class
```

```
df.isnull().sum()
```

```
ID       0
RI       2
Na       2
Mg       0
Al       0
Si       1
K        5
CA       0
Fe       0
Class    0
dtype: int64
```

```
#imputing the column which has less than 10 empty values with previous row values based on their respective classes
class_column = "Class"         #declaring the class_column
cols_to_impute = df.columns[df.isnull().sum()<10]    #considering the columns which has null values less than 10
for col in cols_to_impute:
  df[col] = df.groupby(class_column)[col].transform(lambda x: x.fillna(method = 'ffill'))     #filling the null values with previous values t
```

```
#finding if there are any null values present in any column in the dataframe
df.isnull().sum()
```

```
ID       0
RI       0
Na       0
Mg       0
Al       0
Si       0
K        0
CA       0
Fe       0
Class    0
dtype: int64
```

```
df
```

	ID	RI	Na	Mg	Al	Si	K	CA	Fe	Class
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	1
2	3	1.51761	13.53	3.55	1.54	72.99	0.39	7.78	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.61	0.00	8.62	0.0	7

214 rows × 10 columns

Hint: Consider using one or a combinition of [fillna](), [groupby](), [transform](), and [mean]() to compete this task

TODO3: Check if all the missing values are handled

```
#finding if there are any null values present in any column in the dataframe
df.isnull().sum()
```

```
ID       0
RI       0
Na       0
Mg       0
Al       0
Si       0
K        0
CA       0
Fe       0
Class    0
dtype: int64
```

Hint: If you have done all the above mentioned steps properly, you shouldnt be getting NAN values

TODO4: Get the descriptive statistics of the predictors for each class and present the information in a table/matrix format

Also, what will you do if your data has non-numerical columns. How will you generate the summary for all columns of a DataFrame regardless of data type?

```
#descriptive statistics of the predictors by grouping class
df.groupby('Class').describe(include='all')
```

	ID								RI		...	CA		Fe		
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std
Class																
1	70.0	35.5	20.351085	1.0	18.25	35.5	52.75	70.0	70.0	1.518739	...	9.0525	10.17	70.0	0.057000	0.0890
2	76.0	108.5	22.083176	71.0	89.75	108.5	127.25	146.0	76.0	1.518619	...	8.9150	16.19	76.0	0.079737	0.1064
3	17.0	155.0	5.049752	147.0	151.00	155.0	159.00	163.0	17.0	1.517964	...	8.9300	9.65	17.0	0.057059	0.1078
5	13.0	170.0	3.894440	164.0	167.00	170.0	173.00	176.0	13.0	1.518928	...	11.5300	12.50	13.0	0.060769	0.1555
6	9.0	181.0	2.738613	177.0	179.00	181.0	183.00	185.0	9.0	1.517456	...	9.9500	11.22	9.0	0.000000	0.0000
7	29.0	200.0	8.514693	186.0	193.00	200.0	207.00	214.0	29.0	1.517146	...	8.9500	9.76	29.0	0.013448	0.0297

6 rows × 72 columns

Hint: How to calculate summary statistics with Pandas?

Problem 2 (30 points)

```
#Import the built-in Titanic dataset for this problem
import seaborn as sns
titanic = sns.load_dataset('titanic')
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

TODO1: What is the mean age of female and male survivors respectively?

```
#calculating the mean of female and male survivors using pivot table
pivot_table = titanic[titanic['survived'] > 0].pivot_table(index='sex', values='age', aggfunc='mean')
pivot_table
```

	age
sex	
female	28.847716
male	27.276022

Hint: Apart from the aforementioned function groupby, creating a pivot table is also a way to go.

TODO2: Among all the survivors, what is the gender distribution? (You are expected to present the percentage of each gender in a pivot table.)

```
#calculating the distribution of survivors among genders
pivot_table = titanic[titanic['survived'] > 0].pivot_table(index='sex', aggfunc= 'size')
survivor = pivot_table/pivot_table.sum()*100      #changing the calculations into percentages
survivor
```

```
sex
female    68.128655
male      31.871345
dtype: float64
```

Hint: The pivot table can help with complex aggregation.

TODO3: How many children (age <= 12) survived and which class ticket they had?

```
#aggregating the number of survivors where age is less than or equal to 12 and displaying according to classes
pivot_table = titanic[(titanic['survived'] > 0) & (titanic['age'] <= 12)].pivot_table(index='class', values='s
pivot_table
```

	survived
class	
First	3
Second	17
Third	20

Hint:

[Ways to filter pandas dataframe based on column values]

[Using pandas groupby count()]

TODO4: How many first class seated girls (children) DID NOT survive?

```
#calculating the number of girls(childern) who did not survive
pivot_table = titanic[(titanic['survived'] == 0) & (titanic['age'] <= 12) & (titanic['class'] == 'First') & (t:
pivot_table
```

	sex
class	
First	1.0

TODO5: Check whether variable 'survived' and 'alive' are consistent (contains the same information). Is there any other redundant variable existing in this dataset? Drop all the redundant variables and present the updated dataset.

```
#mapping alive values with 0 and 1 inorder to match with survive column to chevk consitency
titanic['alive_encoded'] = titanic['alive'].map({'yes': 1, 'no': 0})


le1 = LabelEncoder()      #declaring labelEncoder as le1
le2 = LabelEncoder()      #declaring labelEncoder as le2
titanic['survive_encoded'] = le1.fit_transform(titanic['survived'])      #transforming survived column into label encoder as survive_encoded
titanic['alive_encoded1'] = le2.fit_transform(titanic['alive_encoded'])   #transforming alive_encoded column into label encoded as alive_enco
result = titanic['survive_encoded'] == titanic['alive_encoded1']     #checking all the values whether they are consistent in both the columns

if all(result):
  print("The columns are consistent")         #print if the columns are consistent
else:
  print("The columns are not consistent")     #print if the columns are not consistent
```

```
The columns are consistent
```

```
#redundant columns
titanic.drop(columns = 'pclass', inplace = True)    #dropping pclass column as there is already one more column with same values i.e., class
titanic.drop(columns = 'embarked', inplace = True)  #dropping embarked column as there is already one more column with same values i.e., emba
titanic.drop(columns = 'who', inplace = True)     ##dropping who column as there is already one more column with same values i.e., sex
titanic.drop(columns = 'alive', inplace = True)    #dropping alive column as there is already one more column with same values i.e., survived
titanic.drop(columns = 'alive_encoded', inplace = True)    #dropping alive_encoded column as there is already one more column with same value
titanic.drop(columns = 'survive_encoded', inplace = True)   ##dropping survive_encoded column as there is already one more column with same v
titanic.drop(columns = 'alive_encoded1', inplace = True)    #dropping alive_encoded1 column as there is already one more column with same val
titanic.drop(columns = 'adult_male', inplace = True)    #dropping adult_male column as there is already one more column with same values i.e.

#displaying the table
titanic
```

```
survived    sex   age  sibsp  parch    fare   class  deck  embark_town  alone
```

Hint:

You might want to encode two variables to 0 and 1 with LabelEncoder and check if two columns contain the same value.

Or else you can use Replace

```
3     1  female  35.0   1   0  53.1000   First   C  Southampton  False
```

TODO6: What other insights can you draw from this dataset? Present one finding through pivot table.

```
...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
```

```
#from the below pivot table we can interpret the percentage of male and female who have survived who had boarde
pivot_table1 = titanic[titanic['survived'] > 0].pivot_table(values = 'survived', columns = 'sex'  , index='emba
survivor = pivot_table1/pivot_table1.sum()*100      #changing the calculations into percentages
survivor
```

sex	female	male
embark_town		
Cherbourg	27.705628	26.605505
Queenstown	11.688312	2.752294
Southampton	60.606061	70.642202

Part 2: Data Visualization

Before you start: Read the book chapter "Data Visualization".

Note: Please make sure your plots are complete and presentable with a title, proper axis names and legends if applicable.

Problem 3 (25 points)

Dataset: Forest fires

Source: https://archive.ics.uci.edu/ml/datasets/Forest+Fires

The file forestfires.csv includes data from Cortez and Morais (2007).

Number of instances and attributes are 517 and 13 respectively.

Attribute Information:

1. X - x-axis spatial coordinate within the Montesinho park map: 1 to 9
2. Y - y-axis spatial coordinate within the Montesinho park map: 2 to 9
3. month - month of the `year`: 'jan' to 'dec'
4. day - day of the week: 'mon' to 'sun'
5. FFMC - FFMC index from the FWI system: 18.7 to 96.20
6. DMC - DMC index from the FWI system: 1.1 to 291.3
7. DC - DC index from the FWI system: 7.9 to 860.6
8. ISI - ISI index from the FWI system: 0.0 to 56.10
9. temp - temperature in Celsius degrees: 2.2 to 33.30
10. RH - relative humidity in %: 15.0 to 100
11. wind - wind speed in km/h: 0.40 to 9.40
12. rain - outside rain in mm/m2 : 0.0 to 6.4
13. area - the burned area of the forest (in ha): 0.00 to 1090.84

(this output variable is very skewed towards 0.0, thus it may make sense to model with the logarithm transform).

```
#Importing libraries and loading the dataset 'forestfires.csv'

import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
file = files.upload()  #upload file into google colab session
df1 = pd.read_csv("forestfires.csv")
df1.head()
```

Choose Files forestfires.csv

- **forestfires.csv**(text/csv) - 25478 bytes, last modified: 2/2/2023 - 100% done

```
Saving forestfires.csv to forestfires (4).csv
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

TODO1: Plot a stacked bar chart to show the number of forest fires grouped by months and days of the week. *(Make sure the month are in Months chronological order i.e attribute values are sorted starting with January and ending with December)*

```
#displaying number of forest fires grouped by months and days
df1.pivot_table(index='month',columns='day')
```

	DC							DMC			.
day	**fri**	**mon**	**sat**	**sun**	**thu**	**tue**	**wed**	**fri**	**mon**	**sat**	.
month											
apr	85.300000	41.600000	97.100000	19.700000	55.200000	NaN	43.500000	23.300000	24.900000	27.400000	
aug	665.200000	632.046667	652.062069	616.082500	659.057692	639.992857	636.000000	161.419048	158.806667	170.134483	
dec	352.600000	349.700000	NaN	353.500000	352.000000	349.700000	354.600000	26.700000	25.400000	NaN	
feb	39.640000	46.333333	46.925000	122.700000	26.600000	16.200000	18.700000	8.360000	7.800000	10.325000	
jan	NaN	NaN	9.300000	171.400000	NaN	NaN	NaN	NaN	NaN	3.700000	
jul	381.500000	470.950000	433.937500	481.940000	505.000000	429.633333	472.333333	110.466667	104.575000	88.525000	
jun	299.166667	275.066667	243.200000	303.100000	333.450000	NaN	324.200000	81.533333	94.266667	66.050000	
mar	81.190909	85.975000	79.050000	98.157143	46.800000	59.200000	42.125000	34.463636	40.225000	35.930000	
may	73.700000	NaN	113.800000	NaN	NaN	NaN	NaN	25.400000	NaN	28.000000	
nov	NaN	NaN	NaN	NaN	NaN	106.700000	NaN	NaN	NaN	NaN	
oct	682.600000	680.150000	686.900000	691.800000	NaN	669.100000	673.800000	41.500000	40.650000	43.700000	
sep	738.889474	737.439286	729.480000	736.318519	739.400000	718.136842	738.442857	125.613158	109.853571	116.148000	

12 rows × 77 columns

```
acking the column month and day. finding the value counts that is the number of forest fires which took place o
roupby('month').day.value_counts().unstack()
```

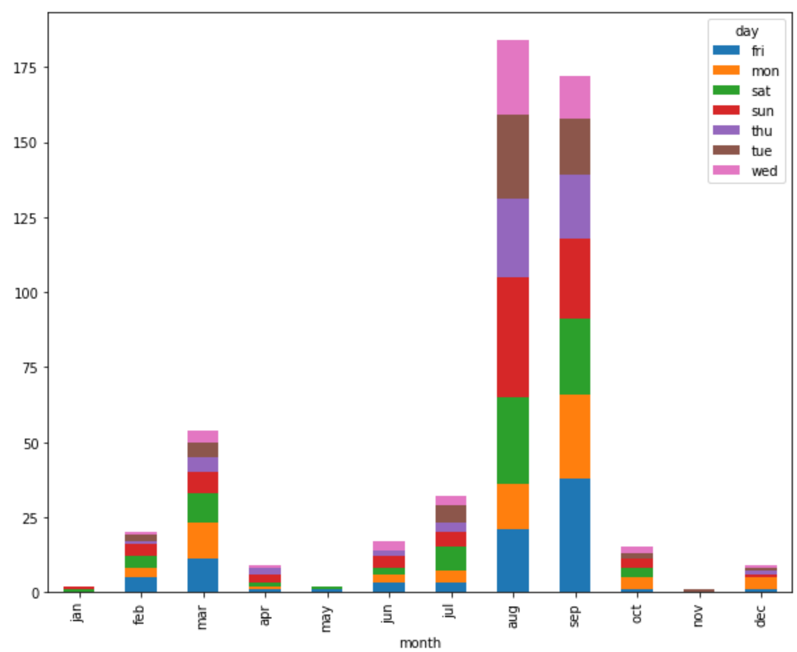
day	fri	mon	sat	sun	thu	tue	wed
month							
apr	1.0	1.0	1.0	3.0	2.0	NaN	1.0
aug	21.0	15.0	29.0	40.0	26.0	28.0	25.0
dec	1.0	4.0	NaN	1.0	1.0	1.0	1.0
feb	5.0	3.0	4.0	4.0	1.0	2.0	1.0
jan	NaN	NaN	1.0	1.0	NaN	NaN	NaN
jul	3.0	4.0	8.0	5.0	3.0	6.0	3.0

```
month=['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
df1['month']=pd.Categorical(df1['month'],categories=month,ordered=True)   #declaring the list of month inorder
grouped=df1.groupby(['month','day']).size().reset_index(name='count')   #grouping the dataframe based on month
grouped=grouped.sort_values(["month","day"])              #sorting
grouped
```

	month	day	count
0	jan	fri	0
1	jan	mon	0
2	jan	sat	1
3	jan	sun	1
4	jan	thu	0
...	...	...	...
79	dec	sat	0
80	dec	sun	1
81	dec	thu	1
82	dec	tue	1
83	dec	wed	1

84 rows × 3 columns

```
#grouping it by month and plotting bar chart by hueing day
df3=df1.groupby('month').day.value_counts().unstack().plot.bar(stacked=True,figsize=(10,8))
```



Hint: Before creating the bar chart, use aforementioned data aggregation tools to transform the original dataset to the data frame you need for this section. To be more specific, you need to compute the count of forest fires by months and days before plotting.

Then build a stacked bar chart with Pandas

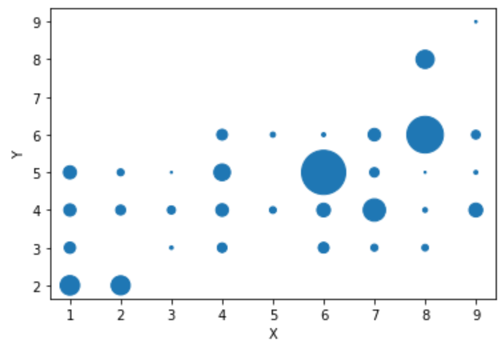
TODO2 (not graded): Do you notice any problem with the stacked bar chart? How do you plan to remedy this problem?

```
#In some particular months the number of forest fires are more when compared to that of others which is leading to the problem in scaling alc
```

TODO3: Create a scatter plot of the fires with the location(X & Y) as the X and Y axis, and the size of the point indicating the area burnt.

```
import matplotlib.pyplot as plt
#sns.scatterplot (x = 'X', y = 'Y', size = 'area', data= df1)
df1.plot.scatter(x='X',y='Y',s='area')    #plotting a scatter plot between X and Y and finding the density of p
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd25a26de50>
```



Hint: [Build a scatter plot with Pandas]

TODO4: Plot the scatter matrix for temp, RH, DC and DMC. How do you interpret the result in terms of correlation among the variables?

```
sns.pairplot(data=df1,vars=['temp','RH','DC','DMC'])
# From the below graphs, we can find out that, there is a posiitive correlation between 'Temp' vs 'DMC' and 'DMC' vs 'DC' whereas we can see
```

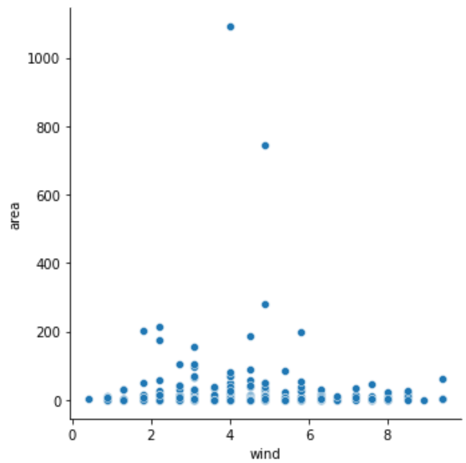
```
<seaborn.axisgrid.PairGrid at 0x7fd25e190cd0>
```

Hint: [Creat a scatter matrix with Seaborn]()

TODO5: Does the wind speed affect the spread of wildfire? Use visualization to back up your answer.

```
#wind vs area burnt
sns.relplot(data=df1,x='wind',y='area')
#There is less correlation between wind and area.Eventhough the wind speed increase there is no high spread of forest fires so we can say tha
```

```
<seaborn.axisgrid.FacetGrid at 0x7fd25e0e0550>
```



Problem 4 (20 points)

Dataset: Graduate School Admission

This dataset was created for Graduate Admissions prediction.

The purpose is to help students with shortlisting target universities according to their profiles.

The predicted output gives them a fair idea about their chances of admission for a particular university.

Attribute Information:

Serial.No.: application number: 1 to 500

GRE.Score: GRE score: 290 to 340

TOEFL.Score: TOEFL score: 92 to 120

University.Rating: undergraduate school's rating: A to E

SOP: Statement of Purpose score: 1 to 5

LOR: Letter of Recommendation score: 1 to 5

CGPA: Undergraduate GPA: 6.8 to 9.92

Research: Research experience: Yes or No

Chance.of.Admit: Chance of getting admitted: 0.34 to 0.97

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
from google.colab import files
file = files.upload()  #upload file into google colab session
df = pd.read_csv("Admission_Predict.csv")
df.head()
```

Choose Files Admission_Predict.csv

- **Admission_Predict.csv**(text/csv) - 13524 bytes, last modified: 2/2/2023 - 100% done

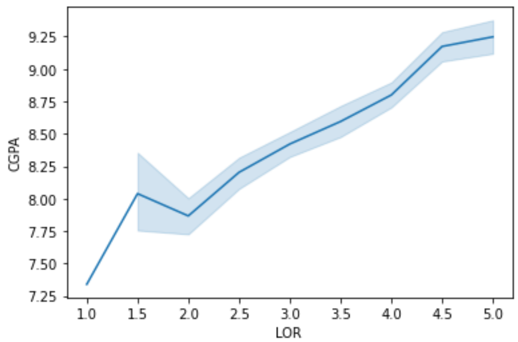
```
Saving Admission_Predict.csv to Admission_Predict (1).csv
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	yes	0.92
1	2	324	107	4	4.0	4.5	8.87	yes	0.76
2	3	316	104	3	3.0	3.5	8.00	yes	0.72
3	4	322	110	3	3.5	2.5	8.67	yes	0.80

TODO1: Is LOR score related to CGPA? Use visualization to back up your answer.

```
#df.plot.scatter(x='LOR ',y='CGPA')# As both the variables 'lor and cgpa are numerical tried to plot scatterplot but the plot is difficult tc
sns.lineplot(data=df,x='LOR ',y='CGPA')#As LOR score increases we can observe the CGPA of students is also increasing So students with higher
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd25a1d2c40>
```



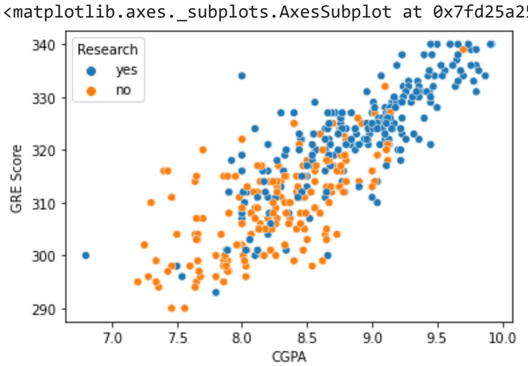
Hint: Use the visualization that is used to compare 2 numerical variables

TODO2:

- Create a scatterplot of CGPA and GRE. Use color to indicate research experience. Interpret the plot.
- Create a scatterplot of University.Rating vs Research. Why is the plot not useful? Pick an appropriate chart type to reveal the relationship between University.Rating and Research.

```
sns.scatterplot(data=df,x=df.CGPA,y='GRE Score',hue='Research')# posiitve correlation between gre score and CGPA.We can find that maximum stu
```

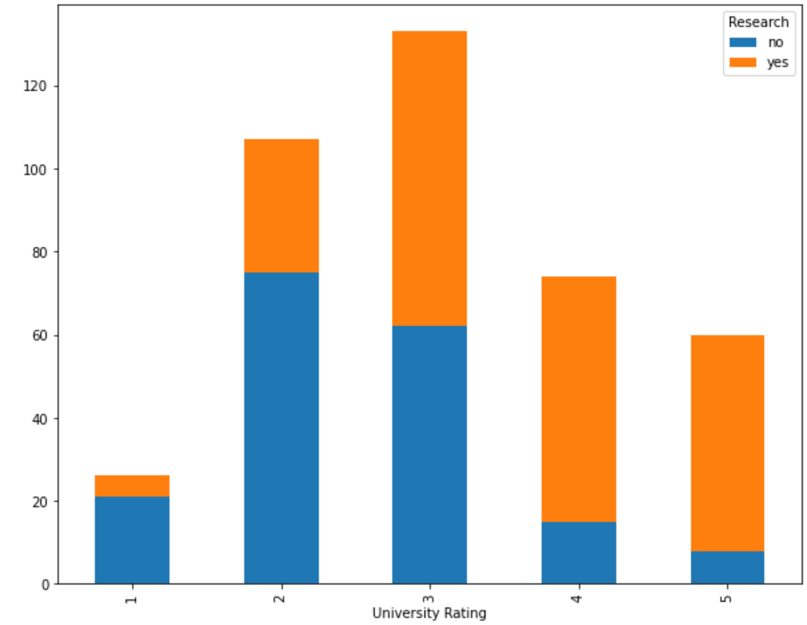
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd25a2548b0>
```




```
sns.scatterplot(data=df,x='University Rating',y='Research')#not informative we can interpret the relation between research and university rat
```

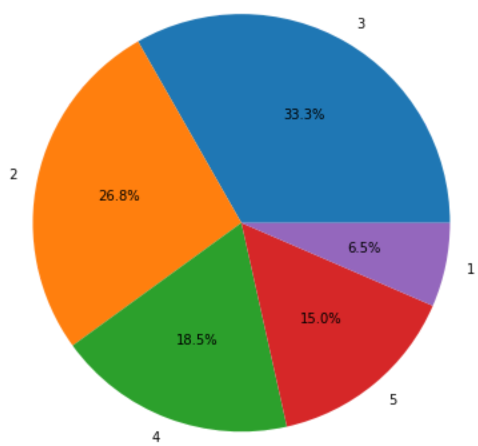
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd25a11d1f0>
```


```
df=df.groupby('University Rating').Research.value_counts().unstack().plot.bar(stacked=True,figsize=(10,8))#Stacked bar is more informative as
```



TODO3: Plot a pie chart of University Rating. The pie chart should also present the percentage of each slice. Explain your findings. *(Make sure you show data labels)*

```
rating_counts = df['University Rating'].value_counts()
plt.pie(rating_counts, labels=rating_counts.index, radius = 1.8, autopct='%1.1f%%')
plt.show()
#From the pie chart we can observe that the rank 3 universities are in higher percentage then followed by rank 2 universities.So we can inter
```



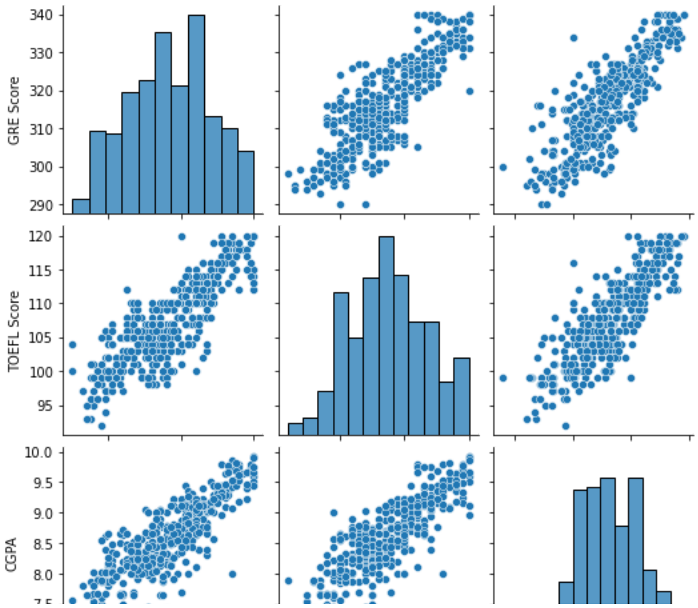
Hint: [Build a pie chart with Matplotlib]

[Build a pie chart with Pandas]

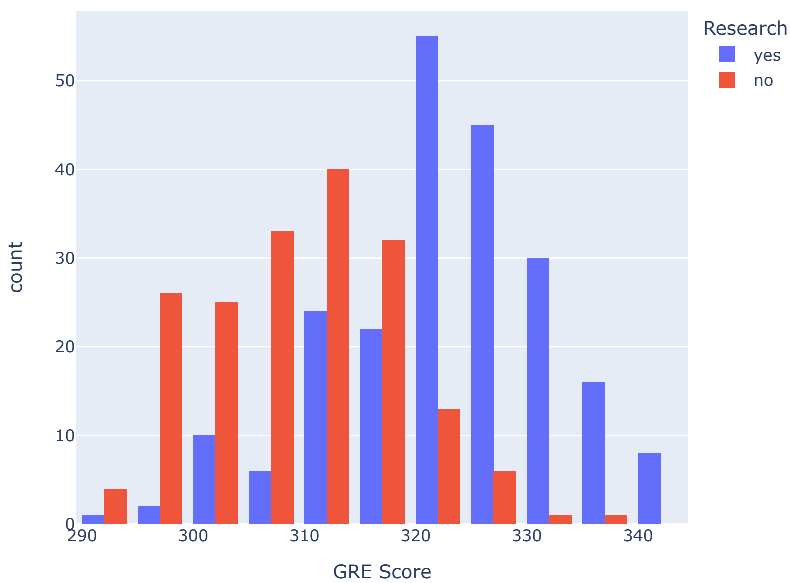
TODO4: What other insights can you draw from this dataset? Present one finding with visualization.

```
#gre score ,university rating,chance of admit
sns.pairplot(data=df,vars=['GRE Score','TOEFL Score','CGPA'])
```

```
<seaborn.axisgrid.PairGrid at 0x7fd25a2544f0>
```




```
import plotly.express as px
from matplotlib.pyplot import figure
figure(figsize=(6,4), dpi=80)
px.histogram(data_frame=df, x="GRE Score", color="Research", barmode="group")# from this we can observe that st
```




```
<Figure size 480x320 with 0 Axes>
```

```
!jupyter nbconvert --to html /content/Group25Homework_1 (2).ipynb
```

```
/bin/bash: -c: line 0: syntax error near unexpected token `('
/bin/bash: -c: line 0: `jupyter nbconvert --to html /content/Group25Homework_1 (2).ipynb'
```

Dataset: Graduate School Admission

This dataset was created for Graduate Admissions prediction.

The purpose is to help students with shortlisting target universities according to their profiles.

The predicted output gives them a fair idea about their chances of admission for a particular university.

Attribute Information:

Serial.No.: application number: 1 to 500

GRE.Score: GRE score: 290 to 340

TOEFL.Score: TOEFL score: 92 to 120

University.Rating: undergraduate school's rating: A to E

SOP: Statement of Purpose score: 1 to 5

LOR: Letter of Recommendation score: 1 to 5

CGPA: Undergraduate GPA: 6.8 to 9.92

Research: Research experience: Yes or No

Chance.of.Admit: Chance of getting admitted: 0.34 to 0.97