FuDgetron Assignment

Q.1 Write python code that would do the following:

```
#importing necessary libraries

import pandas as pd
from geopy.distance import geodesic
from geopy.geocoders import ArcGIS
```

I. Create a data frame from the input (File: Delhi-Electricity-SubStation.csv).

```
#createing a data frame

df = pd.read_csv('/content/Delhi-Electricity-SubStations.csv')
```

```
#displaying the first few rows of a DataFrame

df.head()
```

	_id	Substations	Telephone Numbers	Address	Voltage Class	Coordinates
0	1	Bawana	27791190/1193/1210	400kV Sub-Station Bawana, Sector-5, DSIIDC Baw...	400 kV	28.79568, 77.0723
1	2	Bamnauli	25314199/25314204	Village-Bamnauli, P.O. Dhul Sirus, Near Chhawa...	400 kV	28.5447, 77.03269
2	3	Harsh Vihar	0120-6500138	Harsh Vihar, Loni Road (Near Bhopura Chowk), D...	400 kV	28.71185, 77.29044
		Tikri		Neewala Village Road, Near Vaishno Devi		28.67671,

```
#getting summary of the DataFrame

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43 entries, 0 to 42
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   _id                43 non-null     int64
 1   Substations        43 non-null     object
 2   Telephone Numbers  40 non-null     object
 3   Address            43 non-null     object
 4   Voltage Class      43 non-null     object
 5   Coordinates        41 non-null     object
dtypes: int64(1), object(5)
memory usage: 2.1+ KB
```

```
#dropping the rows with missing values (NaN/null values) from a DataFrame

df.dropna(subset=['Telephone Numbers','Coordinates'], inplace=True)
```

II. In Telephone Numbers there are multiple numbers. Create separate columns for these Telephone1, Telephone2 etc.

```
# Spliting Telephone Numbers into separate columns

split_numbers = df['Telephone Numbers'].str.split('/', expand=True)


df['Telephone1'] = split_numbers[0]
df['Telephone2'] = split_numbers[1]
df['Telephone3'] = split_numbers[2]


df.head()
```

	_id	Substations	Telephone Numbers	Address	Voltage Class	Coordinates	Telephone1	Telephone2	Telephone3
0	1	Bawana	27791190/1193/1210	400kV Sub-Station Bawana, Sector-5, DSIIDC Baw...	400 kV	28.79568, 77.0723	27791190	1193	1210
1	2	Bamnauli	25314199/25314204	Village-Bamnauli, P.O. Dhul Sirus, Near Chhawa...	400 kV	28.5447, 77.03269	25314199	25314204	None
				Harsh Vihar, Lopi					

III. Determine the accuracy of the Coordinates with Address. Add another column GeoAccuracy. Find the coordinates using the address. If the coordinates match exactly with address, then accuracy is 100%, otherwise for every 10 meters off consider a drop in accuracy of 1 %. For example, if the coordinate of the address is 30 meters away from the given coordinates, the accuracy would be 70%.

```
#converting addresses into geographic coordinates


def geocode_address(address):
    nom = ArcGIS()
    coordinate = nom.geocode(address)
    if coordinate:
        rounded_tuple = tuple(round(value, 5) for value in coordinate[1])
        return rounded_tuple
    else:
        return None
```

```
# calculating accuracy
def calculate_accuracy(row):
    if pd.notnull(row['Coordinates']):
        given_coords = tuple(map(float, row['Coordinates'].split(',')))
        address_coords = geocode_address(row['Address'])

        if address_coords:
            distance = geodesic(given_coords, address_coords).meters

            if distance == 0:
                return 100.0
            else:
                accuracy = max(0, 100 - (distance / 10))
                return round(accuracy, 2)
        else:
            return None
    else:
        return None



# Applying the calculate_accuracy function to each row

df['GeoAccuracy'] = df.apply(calculate_accuracy, axis=1)


df
```

	_id	Substations	Telephone Numbers	Address	Voltage Class	Coordinates	Telephone1	Telephone2	Telephone3	GeoAccuracy
0	1	Bawana	27791190/1193/1210	400kV Sub-Station Bawana, Sector-5, DSIIDC Baw...	400 kV	28.79568, 77.0723	27791190	1193	1210	0.00
1	2	Bamnauli	25314199/25314204	Village-Bamnauli, P.O. Dhul Sirus, Near Chhawa...	400 kV	28.5447, 77.03269	25314199	25314204	None	23.70
2	3	Harsh Vihar	0120-6500138	Harsh Vihar, Loni Road (Near Bhopura Chowk), D...	400 kV	28.71185, 77.29044	0120-6500138	None	None	0.00
3	4	Tikri Kalan(Mundka)	65108444/7290010893	Neewala Village Road, Near Vaishno Devi Mandir...	400 kV	28.67671, 76.98639	65108444	7290010893	None	0.00
4	5	BTPS	26948637	Badarpur Thermal Power Station Complex, Badarp...	220 kV	28.50765, 77.30015	26948637	None	None	89.41
5	6	DSIDC Bawana	65005603/7290013476	Near H Block, J.J. colony, Bawana Hanuman Mand...	220 kV	28.80361, 77.05284	65005603	7290013476	None	87.53
6	7	DIAL	25655090	Near IGI Airport, Delhi	220 kV	28.55616, 77.09995	25655090	None	None	0.00
7	8	Electric Lane	23329790	HCM Lane, Behind BSNL Building, Janpath, New D...	220 kV	28.62174, 77.21969	23329790	None	None	9.78

Near Sai

Q.2 The file MH-Veh-Reg.csv contains data of vehicles registered in Maharashtra for the period 2000-2018. Write python code to do the following

```
#creating the dataframe


df_vehicles = pd.read_csv('/content/MH-Vehicle-Reg.csv',sep=';')
```

Delhi-11...

```
df_vehicles.head()
```

	_id	Sr No.	Year	Region	Sub Region	Motor Cycles	Scooters	Moped	Cars	Jeeps	...	Private Service Vehicles	Ambulances	Articulated/Multi.	T Lo
0	1	1	2000-2001	Greater Mumbai	Mumbai(C)	84289	62444	7289	164758	12727	...	829	902	0	
1	2	2	2000-2001	Greater Mumbai	Mumbai(W)	80320	96297	15230	110397	5465	...	605	298	0	
2	3	3	2000-2001	Greater Mumbai	Mumbai(E)	39930	44932	9786	42945	4439	...	279	158	0	
3	4	4	2000-2001	Greater Mumbai	Borivali	0	0	0	0	0	...	0	0	0	
4	5	5	2000-2001	Thane Region	Thane	130448	104028	9304	96933	23755	...	1030	206	0	

```
df_vehicles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 27 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   _id                            900 non-null    int64 
 1   Sr No.                         900 non-null    int64 
 2   Year                           900 non-null    object
 3   Region                         900 non-null    object
 4   Sub Region                     900 non-null    object
 5   Motor Cycles                   900 non-null    int64 
 6   Scooters                       900 non-null    int64 
 7   Moped                          900 non-null    int64 
 8   Cars                           900 non-null    int64 
 9   Jeeps                          900 non-null    int64 
 10  Stn. Wagons                    900 non-null    int64 
 11  Taxis meter fited              900 non-null    int64 
 12  Luxury /Turist Cabs/           900 non-null    int64 
 13  Auto-rikshaws                  900 non-null    int64 
 14  Stage carriages                900 non-null    int64 
 15  Contract carriages /Mini Bus   900 non-null    int64 
 16  School Buses                   900 non-null    int64 
 17  Private Service Vehicles       900 non-null    int64 
 18  Ambulances                     900 non-null    int64 
 19  Articulated/Multi.             900 non-null    int64 
 20  Trucks & Lorries               900 non-null    int64 
 21  Tanker                         900 non-null    int64 
 22  Delivery Van (4 wheelers)      900 non-null    int64 
 23  Delivery Van (3 wheelers)      900 non-null    int64 
 24  Tractors                       900 non-null    int64 
 25  Trailors                       900 non-null    int64 
 26  Others                         900 non-null    int64 
dtypes: int64(24), object(3)
memory usage: 190.0+ KB
```

i. Merge the data of each year (2000-2001, 2001-2002,….) to a single period (2000-2018) by the region and sub region.

```
start_year = 2000
end_year = 2018

# Creating  new 'Period' column with the entire range (2000-2018)
df_vehicles['Period'] = f"{start_year}-{end_year}"


# Groupping the data by 'Region', 'Sub Region', and 'Period', summing the values
merged_df = df_vehicles.groupby(['Region', 'Sub Region', 'Period']).sum().reset_index()
```

```
<ipython-input-24-339487c163e0>:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify
  merged_df = df_vehicles.groupby(['Region', 'Sub Region', 'Period']).sum().reset_index()
```

```
merged_df
```

	Region	Sub Region	Period	_id	Sr No.	Motor Cycles	Scooters	Moped	Cars	Jeeps	...	Private Service Vehicles	Ambulances	Articu
0	Amrawati Region	Akola	2000-2018	8388	8388	1831354	662478	533725	194733	67539	...	520	1862	
1	Amrawati Region	Amaravati	2000-2018	888	888	411115	89261	77774	47343	5528	...	139	324	
2	Amrawati Region	Amrawati	2000-2018	7446	7446	2752398	776194	1181966	284367	78668	...	883	2564	
3	Amrawati Region	Buldhana	2000-2018	8352	8352	2146589	207743	391400	100699	70979	...	571	2000	
4	Amrawati Region	Washim	2000-2018	8406	8406	1026814	199646	256012	56151	40523	...	13	764	
5	Amrawati Region	Yavatmal	2000-2018	890	890	251054	51685	56859	19969	7351	...	103	131	
6	Amrawati Region	Yawatmal	2000-2018	7480	7480	1712991	456532	632721	140025	69535	...	928	1147	
7	Aurangabad Region	Aurangabad	2000-2018	8172	8172	6732394	1172472	821358	580450	305997	...	15024	4740	
8	Aurangabad Region	Beed	2000-2018	8208	8208	1389296	174268	276431	88063	166834	...	317	161	
9	Aurangabad Region	Jalna	2000-2018	8190	8190	1840802	132054	168691	100090	66852	...	209	1539	
10	Dhule Region	Dhule	2000-2018	8118	8118	2121258	388461	430375	220423	79878	...	945	1607	
11	Dhule Region	Jalgaon	2000-2018	8136	8136	4683708	922367	929245	363146	136294	...	4520	2537	
12	Dhule Region	Nandurbar	2000-2018	8154	8154	754941	103184	36595	93707	34666	...	263	483	
13	Greater Mumbai	Borivali	2000-2018	7722	7722	585828	246798	610	428629	422	...	76	119	
14	Greater Mumbai	Mumbai(C)	2000-2018	7668	7668	3496882	2028388	133665	4013416	184449	...	13476	14558	
15	Greater Mumbai	Mumbai(E)	2000-2018	7704	7704	2803926	1261837	193574	1610609	85715	...	5232	3804	
16	Greater Mumbai	Mumbai(W)	2000-2018	7686	7686	4609117	2862439	267636	4105896	196838	...	5541	8135	
17	Kolhapur Region	Karad	2000-2018	7938	7938	488153	29072	29820	80425	10684	...	129	888	
18	Kolhapur Region	Kolhapur	2000-2018	7884	7884	7667144	1361203	908320	943127	310053	...	2016	3980	
19	Kolhapur Region	Sangli	2000-2018	7902	7902	4582228	884316	731573	557522	244737	...	676	2301	
20	Kolhapur	Satara	2000-	7920	7920	4460445	640547	648146	477699	224397	...	688	2342	

	Region		2018										
21	Latur Region	Ambejogai	2000-2018	8262	8262	530243	29930	28646	43780	29520	...	24	136
22	Latur Region	Latur	2000-2018	8226	8226	1935520	283385	458239	232831	127225	...	599	839
23	Latur Region	Osmanabad	2000-2018	8244	8244	1165907	93303	73993	102398	64726	...	93	581
24	Nagpur(R) Region	Bhandara	2000-2018	8550	8550	1075760	283359	292742	72511	29237	...	942	986
25	Nagpur(R) Region	Chandrapur	2000-2018	8514	8514	2348452	817589	758372	251764	101659	...	1644	4593
26	Nagpur(R) Region	Gadchiroli	2000-2018	8496	8496	668815	146900	33211	30785	21382	...	558	1685
27	Nagpur(R) Region	Gondia	2000-2018	8532	8532	1351058	399799	290913	84322	43268	...	712	887
28	Nagpur(R) Region	Nagpur	2000-2018	7582	7582	1458847	387980	262777	122459	67970	...	2367	1116
29	Nagpur(R) Region	Nagpur Rural	2000-2018	896	896	292925	83106	32112	29634	14171	...	335	224
30	Nagpur(U) Region	Nagpur	2000-2018	7531	7531	5421260	4392213	4330897	1258410	377915	...	19297	7349
31	Nagpur(U) Region	Nagpur East	2000-2018	895	895	200416	30162	1877	28788	8441	...	19	150
32	Nagpur(U) Region	Nagpur(City)	2000-2018	893	893	487388	330401	282821	115988	31517	...	1317	663
33	Nagpur(U) Region	Nagpur(East)	2000-2018	7565	7565	487170	70449	2728	66933	19657	...	75	388
34	Nagpur(U) Region	Wardha	2000-2018	8442	8442	1264993	300651	545151	98666	40102	...	438	638
35	Nanded Region	Hingoli	2000-2018	8316	8316	771119	58882	103949	57946	45703	...	265	648
36	Nanded Region	Nanded	2000-2018	8280	8280	2485649	236640	383760	187063	111353	...	797	1708
37	Nanded Region	Parbhani	2000-2018	8298	8298	1481494	77120	179856	102992	93784	...	540	1555
38	Nashik Region	Ahmednagar	2000-2018	8064	8064	3769474	708183	656633	314276	145288	...	860	2287
39	Nashik Region	Malegaon	2000-2018	8100	8100	2109084	109412	89280	154747	37835	...	121	817
40	Nashik Region	Nashik	2000-2018	8046	8046	8270884	2186858	899582	1352076	411851	...	2991	6545

ii. Write a function that would take year as parameter and plot a graph. Classify the vehicles into 3 categories Private, Commercial and Others. Commercial vehicles will have 2 subcategories Light Motor Vehicles (LMV) and Heavy Motor Vehicles (HMV). Use different colors for each type

of vehicle and also show the legend. The graph should show both the values and percentages.

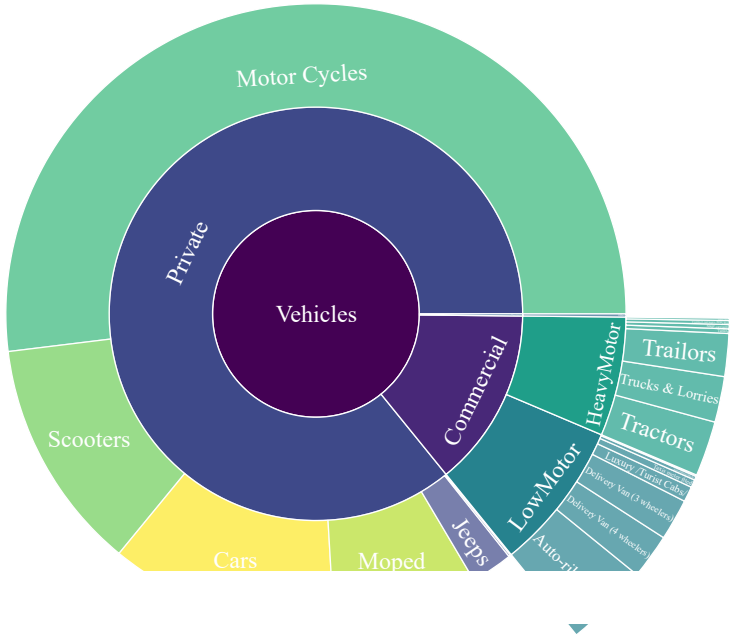
43	Panvel	Panvel	7812	7812	836591	37488	79	451690	2809	...	1588	1640

```
import re
import pandas as pd
import plotly.graph_objects as go
import re
import plotly.express as px



def get_valid_year_range():
    year_range_pattern = re.compile(r'^20(01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18)-20(02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18)$')

    while True:
        user_input = input("Please enter a year range in the format YYYY-YYYY (2001-2018): ")

        if year_range_pattern.match(user_input):
            start_year, end_year = map(int, user_input.split('-'))

            if start_year + 1 == end_year and 2001 <= start_year <= 2018:
                return start_year, end_year
            else:
                print("Years must be consecutive and in the range 2001-2018. Please try again.")
        else:
            print("Invalid format. Please enter a year range in the format YYYY-YYYY (2001-2018).")

def pie_graph(year):
    data = pd.read_csv('MH-Vehicle-Reg.csv', delimiter=';')

    year_data = data[data['Year'] == year]

    # Defining vehicle categories
    private_vehicles = ['Motor Cycles', 'Scooters', 'Moped', 'Cars', 'Jeeps', 'Stn. Wagons', 'Private Service Vehicles']
    commercial_vehicles = ['Taxis meter fited', 'Luxury /Turist Cabs/', 'Auto-rikshaws', 'Stage carriages', 'Contract carriages /Mini Bus', 'School Buses', 'Ambulances', 'Articulated/Multi.', 'Trucks &
    others = ['Others']
    hmvs = ['Stage carriages', 'Contract carriages /Mini Bus', 'School Buses', 'Trucks & Lorries', 'Tanker', 'Tractors', 'Trailors']
    lmvs = ['Taxis meter fited', 'Luxury /Turist Cabs/', 'Auto-rikshaws','Ambulances', 'Articulated/Multi.','Delivery Van (4 wheelers)', 'Delivery Van (3 wheelers)']

    # Calculating the total number of vehicles in each category
    private_total = year_data[private_vehicles].sum().sum()
    commercial_total = year_data[commercial_vehicles].sum().sum()
    other_total = year_data[others].sum().sum()
    lmvs_total = year_data[lmvs].sum().sum()
    hmvs_total = year_data[hmvs].sum().sum()

    private_all_category_total = year_data[private_vehicles].sum()
    private_all_category_names = list(private_all_category_total.index)
    p_len = len(private_all_category_names)

    others_all_category_total = year_data[others].sum()
    others_all_category_names = list(others_all_category_total.index)
    o_len = len(others_all_category_names)

    lmvs_all_category_total = year_data[lmvs].sum()
    lmvs_all_category_names = list(lmvs_all_category_total.index)
    l_len = len(lmvs_all_category_names)
```

```
        hmvs_all_category_total = year_data[hmvs].sum()
        hmvs_all_category_names = list(hmvs_all_category_total.index)
        h_len = len(hmvs_all_category_names)

        total = private_total + commercial_total + other_total

            #creating figure

        fig = go.Figure(go.Sunburst(
            labels=['Vehicles', 'Commercial', 'Private', 'OthersVehicles', 'LowMotor', 'HeavyMotor'] +
                    private_all_category_names + others_all_category_names + lmvs_all_category_names + hmvs_all_category_names,
            parents=['', 'Vehicles', 'Vehicles', 'Vehicles', 'Commercial', 'Commercial'] +
                    (['Private'] * p_len) + (['OthersVehicles'] * o_len) + (['LowMotor'] * l_len) + (['HeavyMotor'] * h_len),
            values=[total, commercial_total, private_total, other_total, lmvs_total, hmvs_total] +
                    list(private_all_category_total.values) + list(others_all_category_total.values) +
                    list(lmvs_all_category_total.values) + list(hmvs_all_category_total.values),
            branchvalues='total',
            marker=dict(
                colors=px.colors.sequential.Viridis
            ),
            hoverinfo='label+value+percent parent',
            textfont=dict(
                size=18,
                color='white',
                family='Open Sans',
            )
        ))

        fig.update_layout(
            title=f"Vehicle Categories Distribution for {year}",
            height=800, width=800,
            legend=dict(title='Vehicle Categories', traceorder='reversed'),
            plot_bgcolor='black',
        )

        fig.show()
        print(f"Plot for the year range: {year}")


    start_year, end_year = get_valid_year_range()
    selected_year_range = f"{start_year}-{end_year}"

    pie_graph(selected_year_range)
```

```
Please enter a year range in the format YYYY-YYYY (2001-2018): 2011-2012
```

Vehicle Categories Distribution for 2011-2012




```
Plot for the year range: 2011-2012
```