In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
```

In [2]:
```
df = pd.read_excel('/mnt/c/Users/Vinay/Desktop/Fittlyf/dataset.xlsx')
df
```

Out[2]:

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India	Level 1	Level 2	Level 3	Level 4
0	2020	Jan	Desktop_Website	Existing_Customer	Came_From_LinkedIn	Bengaluru	NaN	NaN	56892	17178
1	2020	Jan	Desktop_Website	Existing_Customer	Came_From_LinkedIn	Hyderabad	NaN	NaN	41460	11916
2	2020	Jan	Desktop_Website	Existing_Customer	Came_From_LinkedIn	Dehradun	NaN	NaN	55561	19461
3	2020	Jan	Desktop_Website	Existing_Customer	Came_From_LinkedIn	Indore	NaN	NaN	320923	110667
4	2020	Jan	Desktop_Website	Existing_Customer	Came_From_LinkedIn	Pune	NaN	NaN	220937	46033
...	...	...	...	...	...	...	...	...	...	...
2155	2022	Dec	Laptop_Website	New_Customer	Unidentified_Sources	Bengaluru	67299.0	21255.0	6984	1882
2156	2022	Dec	Laptop_Website	New_Customer	Unidentified_Sources	Hyderabad	430294.0	156510.0	46676	16703
2157	2022	Dec	Laptop_Website	New_Customer	Unidentified_Sources	Dehradun	48713.0	27770.0	7515	2089
2158	2022	Dec	Laptop_Website	New_Customer	Unidentified_Sources	Indore	593021.0	310836.0	161575	78465
2159	2022	Dec	Laptop_Website	New_Customer	Unidentified_Sources	Pune	372897.0	123057.0	48802	19441

2160 rows × 10 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Year', 'Month', 'Laptop/Desktop', 'Type_of_Customers?', 'Coming from',
       'Place_in_India', 'Level 1', 'Level 2', 'Level 3', 'Level 4'],
      dtype='object')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2160 entries, 0 to 2159
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Year                2160 non-null   int64  
 1   Month               2160 non-null   object 
 2   Laptop/Desktop      2160 non-null   object 
 3   Type_of_Customers?  2160 non-null   object 
 4   Coming from         2160 non-null   object 
 5   Place_in_India      2160 non-null   object 
 6   Level 1             1081 non-null   float64
 7   Level 2             1081 non-null   float64
 8   Level 3             2160 non-null   int64  
 9   Level 4             2160 non-null   int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 168.9+ KB
```

In [5]:
```
df['Place_in_India'].unique()
```

Out[5]:
```
array(['Bengaluru', 'Hyderabad', 'Dehradun', 'Indore', 'Pune'],
      dtype=object)
```

In [6]:
```
(df.columns[df.isnull().any()])
```

Out[6]:
```
Index(['Level 1', 'Level 2'], dtype='object')
```

In [7]:
```
columns = ['Type_of_Customers?', 'Coming from','Place_in_India']

[df[column].unique() for column in columns]
```

Out[7]:
```
[array(['Existing_Customer', 'New_Customer'], dtype=object),
 array(['Came_From_LinkedIn', 'Landed_Directly', 'Unidentified_Sources'],
       dtype=object),
 array(['Bengaluru', 'Hyderabad', 'Dehradun', 'Indore', 'Pune'],
       dtype=object)]
```

Based on the list of column names provided, it looks like the data may be related to customer information for a business that provides services or products that can be accessed online. Some possible interpretations of the column names are:

1. **"Year" and "Month"** could refer to the time period in which the customer made a purchase or accessed the business's services.
2. **"Laptop/Desktop"** could indicate the device or method that the customer used to access the business's services or products, such as a laptop, desktop, or mobile device.
3. **"Type_of_Customers"** could describe the type of customer, such as individual or business.
4. **"Coming_from"** could refer to the location or source of the customer, such as an online referral or a physical store.
5. **"Place_in_India"** could indicate the region or city in India where the customer is located.

6. **"Level 1-4"** could be categories or labels used to describe the customer, such as their level of experience with the business's products or services or their status as a repeat customer.

Without more context or information, it is difficult to know for certain what these column names represent or which company the data might belong to. It might be helpful to try to gather more information about the data and the context in which it is being used to narrow down the possible companies that the data could belong to.

For Flipkart it could describe Number of customers which went from browsing a product(Level 1) to Checkout(Level 4). Loking at the pattern we could see that for every Level, as we move from 1 to 4, number of customers are reducing significantly and so it compliments our assumptions as not every customer who browse a product, necessarily buys it.
Level 1: *This metric could refer to the number of visitors who landed on Flipkart's website or a specific page on the website. It could be used to measure the overall traffic to the website or a particular page.*

Level 2: *This metric could refer to the number of visitors who took a specific action on the website, such as clicking a button or looked for a Product. It could be used to measure engagement with the website or a particular feature.*

Level 3: *This metric could refer to the number of visitors who started filling out a form on the website, such as a contact form or order form. It could be used to measure the number of visitors who are interested in taking a specific action on the website.*

Level 4: *This metric could refer to the number of visitors who completed and submitted a form on the website. It could be used to measure the effectiveness of the website in converting visitors into customers or leads.*

In [8]:
```
df2 = df[df['Place_in_India'] == 'Pune']
print("Customers coming from Pune: ", len(df2),'\n')

df_linkedin = (df2[df2['Coming from'] == "Came_From_LinkedIn"])
print("Customers coming from LinkedIn: ", len(df_linkedin),'\n')
```

```
Customers coming from Pune:  432

Customers coming from LinkedIn:  144
```

Part 1: Data Cleaning

In [9]:
```
def data_cleaning():
    df2['inc/dec percentage'] = (df['Level 1'] - df['Level 4']) / df['Level 1'] * 100

    # Create a mapping dictionary & eplace the month names using it
    month_map = {'Jan': 'January', 'Feb': 'February', 'Mar': 'March', 'Apr': 'April', 'May': 'May', 'Jun': 'Jun
    df['Month'] = df['Month'].replace(month_map)

    # Replace the values using the df.replace() method
    df["Coming from"] = df['Coming from'].replace({"Came_From_LinkedIn": "From LinkedIn", "Landed_on_the_page_D

    # Print the resulting dataframe
    display(df)
```

In [10]:
```
data_cleaning()
```

```
/tmp/ipykernel_22750/969343209.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df2['inc/dec percentage'] = (df['Level 1'] - df['Level 4']) / df['Level 1'] * 100
```

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India	Level 1	Level 2	Level 3	Level 4
0	2020	January	Desktop_Website	Existing_Customer	From LinkedIn	Bengaluru	NaN	NaN	56892	17178
1	2020	January	Desktop_Website	Existing_Customer	From LinkedIn	Hyderabad	NaN	NaN	41460	11916
2	2020	January	Desktop_Website	Existing_Customer	From LinkedIn	Dehradun	NaN	NaN	55561	19461
3	2020	January	Desktop_Website	Existing_Customer	From LinkedIn	Indore	NaN	NaN	320923	110667
4	2020	January	Desktop_Website	Existing_Customer	From LinkedIn	Pune	NaN	NaN	220937	46033
...	...	...	...	...	...	...	...	...	...	...
2155	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Bengaluru	67299.0	21255.0	6984	1882
2156	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Hyderabad	430294.0	156510.0	46676	16703
2157	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Dehradun	48713.0	27770.0	7515	2089
2158	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Indore	593021.0	310836.0	161575	78465
2159	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Pune	372897.0	123057.0	48802	19441

2160 rows × 10 columns

Part 2: Descriptitve statistics:

In [11]:
```
def descriptive_stats(year_column=2020, month_column="January", mode_column="Laptop_Website", type_column="Exis
    # 1. Calculate the minimum value for each of the "Level" columns, filtered by all input columns
    level1_min = df[(df['Year'] == year_column) & (df["Month"] == month_column) & (df["Laptop/Desktop"] == mode
    level2_min = df[(df['Year'] == year_column) & (df["Month"] == month_column) & (df["Laptop/Desktop"] == mode
    level3_min = df[(df['Year'] == year_column) & (df["Month"] == month_column) & (df["Laptop/Desktop"] == mode
    level4_min = df[(df['Year'] == year_column) & (df["Month"] == month_column) & (df["Laptop/Desktop"] == mode
    # Print the minimum values
    display(f"1. Minimum value in Level 1, 2, 3, 4: {level1_min, level2_min, level3_min, level4_min}")



    # 2. Print the maximum value of “Level 2” / “Level 1” among those who came directly to the via desktop webs
    desktop_rows = df[df["Laptop/Desktop"] == 'Desktop_Website']
    desktop_rows['Level_ratio'] = desktop_rows['Level 2'] / desktop_rows['Level 1']
    max_ratio = desktop_rows['Level_ratio'].max()

    display(f"2. Maximum value of (Level 2 / Level 1) among customers who came via Desktop website: {max_ratio}


    # 3. New DF with default values:
    new_df = df[(df['Year'] == year_column) & (df["Month"] == month_column) & (df["Laptop/Desktop"] == mode_col
    display(new_df)


    # 4. Summary Statistics:
    display(new_df.describe())


    # 5.  list of all the unique values & data types present in the non-numeric columns in New DF
    filtered_df = df.select_dtypes(exclude='number')
    # Unique values and their Datatype:
    print('\n\n5. list of all the unique values & data types present in the non-numeric columns in New DF')
    for column in filtered_df.columns:
        print([filtered_df[column].unique()])
```

In [12]:
```
import warnings
warnings.filterwarnings("ignore")

# When passing the parameters:
descriptive_stats(2022, 'December', "Laptop_Website", 'New_Customer', 'Unidentified_Sources')
```

```
'1. Minimum value in Level 1, 2, 3, 4: (48713.0, 21255.0, 6984, 1882)'
'2. Maximum value of (Level 2 / Level 1) among customers who came via Desktop website: 0.8980720288554845'
```

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India	Level 1	Level 2	Level 3	Level 4
2155	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Bengaluru	67299.0	21255.0	6984	1882
2156	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Hyderabad	430294.0	156510.0	46676	16703
2157	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Dehradun	48713.0	27770.0	7515	2089
2158	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Indore	593021.0	310836.0	161575	78465
2159	2022	December	Laptop_Website	New_Customer	Unidentified_Sources	Pune	372897.0	123057.0	48802	19441

	Year	Level 1	Level 2	Level 3	Level 4
count	5.0	5.000000	5.000000	5.000000	5.000000
mean	2022.0	302444.800000	127885.600000	54310.400000	23716.000000
std	0.0	237390.412623	118011.375228	63292.808725	31659.759475
min	2022.0	48713.000000	21255.000000	6984.000000	1882.000000
25%	2022.0	67299.000000	27770.000000	7515.000000	2089.000000
50%	2022.0	372897.000000	123057.000000	46676.000000	16703.000000
75%	2022.0	430294.000000	156510.000000	48802.000000	19441.000000
max	2022.0	593021.000000	310836.000000	161575.000000	78465.000000

```
5. list of all the unique values & data types present in the non-numeric columns in New DF
[array(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype=object)]
[array(['Desktop_Website', 'Laptop_Website'], dtype=object)]
[array(['Existing_Customer', 'New_Customer'], dtype=object)]
[array(['From LinkedIn', 'Landed_Directly', 'Unidentified_Sources'],
      dtype=object)]
[array(['Bengaluru', 'Hyderabad', 'Dehradun', 'Indore', 'Pune'],
      dtype=object)]
```

In [13]:
```
import warnings
warnings.filterwarnings("ignore")

# When no parameters are given:
# Default parameters: (year_column=2020, month_column="December", mode_column="Desktop_Website", type_column="N
```

```
descriptive_stats()
```

```
'1. Minimum value in Level 1, 2, 3, 4: (nan, nan, 10151, 2272)'
'2. Maximum value of (Level 2 / Level 1) among customers who came via Desktop website: 0.8980720288554845'
```

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India	Level 1	Level 2	Level 3	Level 4
40	2020	January	Laptop_Website	Existing_Customer	Unidentified_Sources	Bengaluru	NaN	NaN	15108	3919
41	2020	January	Laptop_Website	Existing_Customer	Unidentified_Sources	Hyderabad	NaN	NaN	10151	2272
42	2020	January	Laptop_Website	Existing_Customer	Unidentified_Sources	Dehradun	NaN	NaN	13502	4463
43	2020	January	Laptop_Website	Existing_Customer	Unidentified_Sources	Indore	NaN	NaN	64611	17261
44	2020	January	Laptop_Website	Existing_Customer	Unidentified_Sources	Pune	NaN	NaN	50790	10397

	Year	Level 1	Level 2	Level 3	Level 4
count	5.0	0.0	0.0	5.000000	5.000000
mean	2020.0	NaN	NaN	30832.400000	7662.400000
std	0.0	NaN	NaN	25072.990653	6182.976128
min	2020.0	NaN	NaN	10151.000000	2272.000000
25%	2020.0	NaN	NaN	13502.000000	3919.000000
50%	2020.0	NaN	NaN	15108.000000	4463.000000
75%	2020.0	NaN	NaN	50790.000000	10397.000000
max	2020.0	NaN	NaN	64611.000000	17261.000000

```
5. list of all the unique values & data types present in the non-numeric columns in New DF
[array(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype=object)]
[array(['Desktop_Website', 'Laptop_Website'], dtype=object)]
[array(['Existing_Customer', 'New_Customer'], dtype=object)]
[array(['From LinkedIn', 'Landed_Directly', 'Unidentified_Sources'],
      dtype=object)]
[array(['Bengaluru', 'Hyderabad', 'Dehradun', 'Indore', 'Pune'],
      dtype=object)]
```

Part 3: Prescriptive Stats

In [14]:
```
df_2020 = df[df['Year'] == 2020]
# Group the data by "Place_in_India" and sum the values in the "Level 4" column
df_grouped_2020 = df_2020.groupby('Place_in_India')['Level 4'].sum()
df_all_2020 = df_grouped_2020.reset_index()
df_all_2020.insert(0, "Year", 2020, True)
df_all_2020.insert(1, "Rank", range(1, len(df_all_2020) + 1), True)


#No data for Year 2022 so calculating for 2022
df_2022 = df[df['Year'] == 2022]
df_grouped_2022 = df_2022.groupby('Place_in_India')['Level 4'].sum()
df_all_2022 = df_grouped_2022.reset_index()
df_all_2022.insert(0, "Year", 2022, True)
df_all_2022.insert(1, "Rank", range(1, len(df_all_2022) + 1), True)


df_agg = pd.concat([df_all_2020.head(3), df_all_2022.head(3)], axis=0)
df_agg
```

Out[14]:

	Year	Rank	Place_in_India	Level 4
0	2020	1	Bengaluru	3231524
1	2020	2	Dehradun	3685750
2	2020	3	Hyderabad	5156066
0	2022	1	Bengaluru	3752706
1	2022	2	Dehradun	2673864
2	2022	3	Hyderabad	8211936

In [15]:
```
# Create a pivot table with the cities as the index, the years as the columns, and the sum of "Level 4" as the
table = df.pivot_table(index=["Place_in_India", "Year"], values=["Level 1", "Level 2", "Level 3", "Level 4"], a

# Print the pivot table
for index, row in table.iterrows():
    table.at[index,('sumL2/sumL1')] = row[1]/row[0]
    table.at[index,('sumL3/sumL1')] = row[2]/row[0]
    table.at[index,('sumL4/sumL1')] = row[3]/row[0]
table = table.drop(["Level 1", "Level 2", "Level 3", "Level 4"], axis=1)
table
```

Out[15]:

Place_in_India	Year	sumL2/sumL1	sumL3/sumL1	sumL4/sumL1
Bengaluru	2020	0.620010	0.573013	0.242929
	2021	0.441488	0.390348	0.184470
	2022	0.398947	0.375267	0.179285
Dehradun	2020	0.562625	0.410828	0.197959
	2021	0.379406	0.304007	0.119051
	2022	0.329035	0.349966	0.114622
Hyderabad	2020	0.616914	0.487160	0.235361
	2021	0.419998	0.321613	0.162600
	2022	0.457987	0.394942	0.132555
Indore	2020	0.681462	0.563541	0.267239
	2021	0.464476	0.411337	0.182769
	2022	0.520902	0.472407	0.135817
Pune	2020	0.545058	0.518644	0.187607
	2021	0.322846	0.237307	0.099993
	2022	0.361984	0.273714	0.084704

In [16]:
```
table.sort_values("sumL4/sumL1", ascending=False)
```

Out[16]:

Place_in_India	Year	sumL2/sumL1	sumL3/sumL1	sumL4/sumL1
Indore	2020	0.681462	0.563541	0.267239
Bengaluru	2020	0.620010	0.573013	0.242929
Hyderabad	2020	0.616914	0.487160	0.235361
Dehradun	2020	0.562625	0.410828	0.197959
Pune	2020	0.545058	0.518644	0.187607
Bengaluru	2021	0.441488	0.390348	0.184470
Indore	2021	0.464476	0.411337	0.182769
Bengaluru	2022	0.398947	0.375267	0.179285
Hyderabad	2021	0.419998	0.321613	0.162600
Indore	2022	0.520902	0.472407	0.135817
Hyderabad	2022	0.457987	0.394942	0.132555
Dehradun	2021	0.379406	0.304007	0.119051
	2022	0.329035	0.349966	0.114622
Pune	2021	0.322846	0.237307	0.099993
	2022	0.361984	0.273714	0.084704

3. Bottom 3 for "Level4 / Level1":

Order: Ascending
Bottom 3 for 2021: *Pune Dehradun Hyderabad*

Bottom 3 for 2022: *Pune Dehradun Hyderabad*

From the above table we can conclude that all the cases are same with Pune performing worst followed by Dehradun and then Hyderabad.

In [17]:
```
max(df[df["Level 4"] > 150000]['Place_in_India'])
```

Out[17]:
```
'Pune'
```

4. “Level 4” value greater than 150000 most of the times:

The Answer is 'Pune'

In [18]:
```
min(df[df['Type_of_Customers?'] == "Existing_Customer"]['Place_in_India'])
```

Out[18]:
```
'Bengaluru'
```

5. least number of existing customers?:

least number of existing customers?:

The Answer is "Bengaluru"

Part 4: Machine Learning:

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
import seaborn as sn
import matplotlib.pyplot as plt
```

In [20]:
```
# Preprocessing data to feed into an Macine Learning model:
df2 = df.copy()
df2 = df2.drop(['Level 1', 'Level 2', 'Level 3'], axis=1)

# Encoding the Non-Numerical values:
from sklearn.preprocessing import LabelEncoder

# Create a dictionary to store the encoding maps
encoding_maps = {}

# Iterate over all the text columns
for col in df2.select_dtypes(include=['object']):
        # Encode the column using a LabelEncoder
        le = LabelEncoder()
        df2[col] = le.fit_transform(df2[col])
        # Store the encoding map in the dictionary
        encoding_maps[col] = le.classes_

# 1: January --- 12 : December
df2['Month'] = [(x+1) for x in df2.iloc[:]['Month']]

# Select the label column
labels = df2['Level 4']
features = df2.drop('Level 4', axis=1)
display(features, labels)
```

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India
0	2020	5	0	0	0	0
1	2020	5	0	0	0	2
2	2020	5	0	0	0	1
3	2020	5	0	0	0	3
4	2020	5	0	0	0	4
...	...	...	...	...	...	...
2155	2022	3	1	1	2	0
2156	2022	3	1	1	2	2
2157	2022	3	1	1	2	1
2158	2022	3	1	1	2	3
2159	2022	3	1	1	2	4

2160 rows × 6 columns

```
0        17178
1        11916
2        19461
3       110667
4        46033
         ...
2155      1882
2156     16703
2157      2089
2158     78465
2159     19441
Name: Level 4, Length: 2160, dtype: int64
```

In [21]:
```
# Train-test Split:
x_train,x_test,y_train,y_test = train_test_split(features, labels, test_size = 0.1, random_state = 7)
display(x_train, y_train)
```

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India
229	2020	1	1	1	0	4
1382	2021	3	0	0	0	1
34	2020	5	1	0	0	4
206	2020	1	0	1	2	2
37	2020	5	1	0	1	1
...	...	...	...	...	...	...
211	2020	1	1	0	0	2
1603	2022	8	1	0	2	3
537	2020	12	1	1	2	1
1220	2021	12	0	1	1	0
175	2020	8	1	1	2	0

1944 rows × 6 columns

```
229     122740
1382     17429
34       40931
206       7345
37       14367
         ...
211       4578
1603      9831
537       7081
1220     46196
175       4140
Name: Level 4, Length: 1944, dtype: int64
```

In [22]:
```
import random

# A set of default values for each month(0-january to 11-December) of the Year 2023
default = [(2023, int((_/50)+1) , random.randint(0, 1), random.randint(0, 1), random.randint(0, 2), random.rand
ds = pd.DataFrame(default, columns=x_test.columns)
ds
```

Out[22]:

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India
0	2023	1	1	1	0	0
1	2023	1	0	0	1	4
2	2023	1	0	1	0	1
3	2023	1	0	0	2	2
4	2023	1	0	0	0	0
...	...	...	...	...	...	...
595	2023	12	0	0	0	0
596	2023	12	1	1	2	2
597	2023	12	1	0	1	0
598	2023	12	0	0	2	4
599	2023	12	1	1	1	0

600 rows × 6 columns

In [23]:
```
from sklearn.metrics import mean_absolute_percentage_error, mean_squared_error
ypred = []
def Model(xtrain=x_train, ytrain=y_train, xtest=x_test, ytest=y_test, metrics=False):
    model = XGBClassifier(eval_metric='merror')
    model.fit(xtrain,ytrain)
    ypred = model.predict(xtest)

    if metrics == True:
        mape = mean_absolute_percentage_error(ytest, ypred)
        rmse = np.sqrt(mean_squared_error(ytest, ypred))
        print(f"MAPE: {mape}, RMSE: {rmse}")
    return ypred
```

In [24]:
```
''' Predictions for Year 2023 for 600 randomly created datapoints:
    Reason for choosing 600 Datapints: for 2020-2022 we have about 1944 datapoint,
    so an average of about 650 datapoints for each year. SO to match that we created
    a total of 600 datapoint for future prediction.
'''
predictions = Model(xtest=ds)
predictions
```

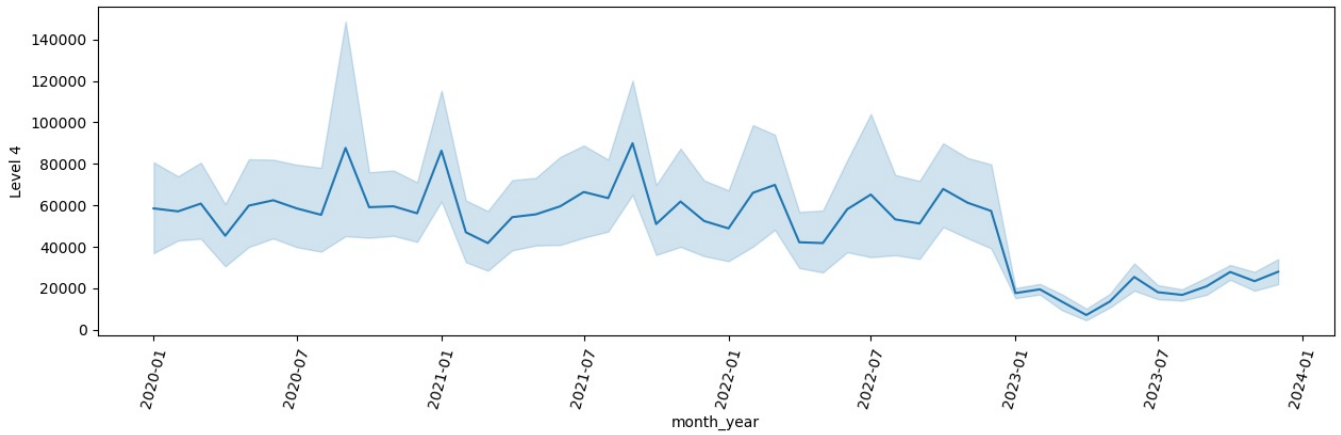
Out[24]:
```
array([30242, 19529, 15595,  2362, 15595, 19529, 19529, 19529,  2362,
        2362, 19529, 19529, 19529, 15595, 32233, 15595, 30242,  7079,
       32233, 15595, 19529, 15595, 15595, 36822,  2334, 19529, 15595,
       19529, 19529, 15595, 15595, 19529, 19529,  5405, 15595, 32233,
       15595, 15595, 32233,  2362, 19529,  2362, 19529,  7079, 35556,
       19529, 15595, 32233, 15595,  5405,  2334, 15595, 30242, 19529,
       32233, 15595, 15595, 19529, 19529, 32233, 15595, 15595, 19529,
       19529, 15595, 19529, 19529, 19529,  2362, 36822, 19529, 15595,
       15595, 15595, 15595, 15595,  2362, 36822, 35556, 32233, 19529,
       15595, 15595,  2362, 19529, 36822, 15595, 15595,  2334, 35556,
       32233, 19529, 15595, 36822, 15595, 15595, 19529, 15595, 15595,
       15595,  2362,  2362,  2362, 19529,  2362,  2362, 30962, 36822,
       26268, 30962,  2362,  2362,  2362, 26268,  2362, 36822,  2362,
       15595, 30962, 35556,  2362,  2362,  2362, 30962,  2362, 30962,
       30962, 26268,  5137,  2362,  2362, 36822,  2362,  2362,  2362,
       30962,  2334,  2362, 30962,  2362,  2362,  2362,  2362,  2362,
       30962, 35556, 26268,  2362,  2362,  2362,  2362,  2362,  2362,
       15595, 35556,  2362,  2362,  2362,  2362,  2944, 35556, 15595,
        2944, 35556,  2362,  2362, 19529,  2362,  2362,  2362,  2362,
        2362,  2334,  2920,  2362,  2362,  2362,  2362,  2362, 19529,
        2362,  3245,  2362,  2944,  2920, 19529,  2362,  2362,  2362,
        2362,  2362,  2362,  2362,  2362,  2362,  2362,  2362,  3245,
       35556, 19529, 19529, 19529, 19529, 21651, 21651, 19529, 21651,
        2944,  2362, 19529,  7079,  2944, 19529,  2944,  2334,  7079,
       19529,  5405, 19529, 19529, 13068, 19529,  5405, 15595,  2334,
       19529,  2362, 19529, 78976,  2944,  2334, 19529, 19529, 19529,
        5405, 19529, 19529, 19529,  7079,  5137,  2944,  2944, 19529,
        2334,  7079, 19529,  7079,  2944,  2362, 19529, 32233,  2944,
       21734, 21734,  2334, 21734, 78976,  5897,  2334, 21734,  2334,
       32233, 32233,  2334, 21734,  2334, 32233, 78976,  2944, 78976,
       21734, 21734,  2334, 78976, 21734, 32233,  2334, 32233,  2362,
        5897, 78976, 21734, 32233, 21734, 32233,  5897, 21734,  5897,
       21734,  2334,  2334, 78976, 21734, 32233, 32233, 78976,  2334,
       32233,  2334, 21734, 15595, 17741, 15595,  5897, 15595, 15595,
       15595, 36822, 15595, 15595, 15595, 15595, 36822, 15595, 15595,
       36822, 15595, 15595, 36822,  5405, 36822, 15595,  5172,  2334,
       15595, 15595,  5172, 36822,  2334,  5405, 41530, 36822,  5137,
       15595, 36822, 15595,  2362,  2334,  7079, 15595, 15595, 15595,
       41530,  2334, 15595, 21734, 15595, 15595, 36822, 21734, 36822,
       21734,  5172, 36822, 21734, 21734,  5172,  5172,  5172, 13068,
       21734, 21734, 13068,  5137, 15595, 13068, 21734, 21734, 21734,
       21734, 13068,  5172, 21734, 21734, 13068, 36822,  5172,  5405,
       36822, 13068, 21734,  5172, 21734,  5172,  5172, 21734, 21734,
        5172,  5172, 13068,  5137, 21734, 21734, 13068,  5172, 13068,
       36822, 36822, 21734, 13068, 36822, 36822,  5137,  2334, 36822,
       36822, 36822,  7079, 13068, 13068,  5137,  7079,  5137,  7079,
       36822, 21734,  7079,  7079, 36822,  7079, 36822, 15595, 36822,
       15595, 21734, 36822,  7079, 13068, 36822, 36822, 15595, 36822,
       36822, 36822,  4058, 36822,  7079, 36822,  7079,  4058, 36822,
       36822, 13068,  7079,  7079,  5137,  7079, 36822, 36822, 13068,
       36822, 36822, 15595, 36822, 36822, 36822, 26268, 15595, 36822,
       26268,  7079, 36822, 36822,  5137, 36822, 36822, 36822, 36822,
       36822,  2362, 36822, 36822, 36822, 15595,  7079, 13068,  5897,
       13068,  5137, 36822, 36822, 36822, 36822, 36822,  7079, 36822,
       36822, 36822, 13068, 36822, 36822,  7079, 13068, 36822, 36822,
       36822, 36822, 13068, 36822, 36822, 30242,  2334, 26268, 30242,
       36822, 13068, 30242, 30242,  2334, 30242,  5897, 26268, 30242,
        5897,  2334, 36822,  2334, 17741, 30242,  2334, 26268, 30242,
        2334, 30242, 17741, 26268, 36822, 30242, 13068, 17741, 30242,
        7079, 77847, 77847, 13068,  2334,  5897, 36822, 30242, 30242,
       26268, 30242, 36822,  2334, 30242,  5137, 13068, 36822, 21734,
       30242, 17741, 36822, 17741, 30242, 17741, 30242, 26268, 17741,
       26268, 30242, 13068, 13068, 15595, 77847, 17741, 77847,  2334,
       17741, 21734,  2334, 30242, 30242, 13068, 77847, 15595, 17741,
       17741, 77847, 30242, 17741,  7079, 77847, 36822, 17741, 30242,
       17741, 17741,  2334, 21734, 13068, 77847, 17741, 17741, 17741,
       17741, 13068, 17741, 26268, 77847, 30242])
```

In [25]:
```
df_2020_f, df_2020_l  = features[df2.Year == 2020], df2[df2.Year == 2020]['Level 4']
df_2021_f, df_2021_l  = features[df2.Year == 2021], df2[df2.Year == 2021]['Level 4']
df_2022_f, df_2022_l  = features[df2.Year == 2022], df2[df2.Year == 2022]['Level 4']

for x in [(df_2020_f, df_2020_l), (df_2021_f, df_2021_l), (df_2022_f, df_2022_l)]:
    Model(xtest=x[0], ytest=x[1], metrics=True)
```

```
MAPE: 0.8346466629136121, RMSE: 90121.12823597471
MAPE: 0.7982595349793671, RMSE: 88451.11783843128
MAPE: 1.0090725335476372, RMSE: 92570.29232635118
```

In [26]:
```
# create a new column with the month and year values formatted as strings
ds['Level 4'] = predictions
df_plt = df2.copy()
df_plt = pd.concat([df_plt, ds], ignore_index=True)

df_plt['month_year'] = df_plt['Month'].astype(str).str.zfill(2)+ '-' +df_plt['Year'].astype(str)

df_plt['month_year'] = pd.to_datetime(df_plt['month_year'], format='%m-%Y')

# sort the DataFrame by the month_year column in ascending order
```

```
df_plt = df_plt.sort_values(by='month_year')


plt.figure(figsize=(15, 4))
# create the line plot
sn.lineplot(x='month_year', y=df_plt['Level 4'], data=df_plt)
plt.xticks(rotation=75)

# show the plot
plt.show()
```



In [27]:
```
# Create a pivot table with the cities as the index, the years as the columns, and the sum of "Level 4" as the
places_decoded = le.inverse_transform(df_plt["Place_in_India"])
table = df_plt.pivot_table(index=[places_decoded], columns="Year", values=["Level 4"], aggfunc="sum")

# Print the pivot table
# for index, row in table.iterrows():
#     display(row[1])
#     table.at[index,('Level 4')] = row[3]/row[0]
#     table.at[index,('Level 4')] = row[2]/row[0]
#     table.at[index,('Level 4')] = row[3]/row[0]
# table = table.drop(["Level 1", "Level 2", "Level 3", "Level 4"], axis=1)
table["diff"] = table["Level 4"][2023] - table["Level 4"][2022]
table
```

Out[27]:

	Level 4				diff
Year	2020	2021	2022	2023	
Bengaluru	3231524	3140030	3752706	1733909	-2018797
Dehradun	3685750	2445091	2673864	2082056	-591808
Hyderabad	5156066	7836311	8211936	1931170	-6280766
Indore	20092071	17533698	15104408	3000648	-12103760
Pune	11039977	12805835	11208722	2807079	-8401643

*Looking at the above table we could conclude that in comparison to 2022, **Dehradun** in 2023 would do better than all the other Places*

Part 5: Visualizations:

In [28]:
```
# Iterate over all the text columns
for col in df.select_dtypes(include=['object']):
    if col == "Month":
        # Encode the column using a LabelEncoder
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        # Store the encoding map in the dictionary
        encoding_maps[col] = le.classes_

# 1: January --- 12 : December
df['Month'] = [(x+1) for x in df.iloc[:]['Month']]
df["Month"].unique()
```

Out[28]:
```
array([ 5,  4,  8,  1,  9,  7,  6,  2, 12, 11, 10,  3])
```

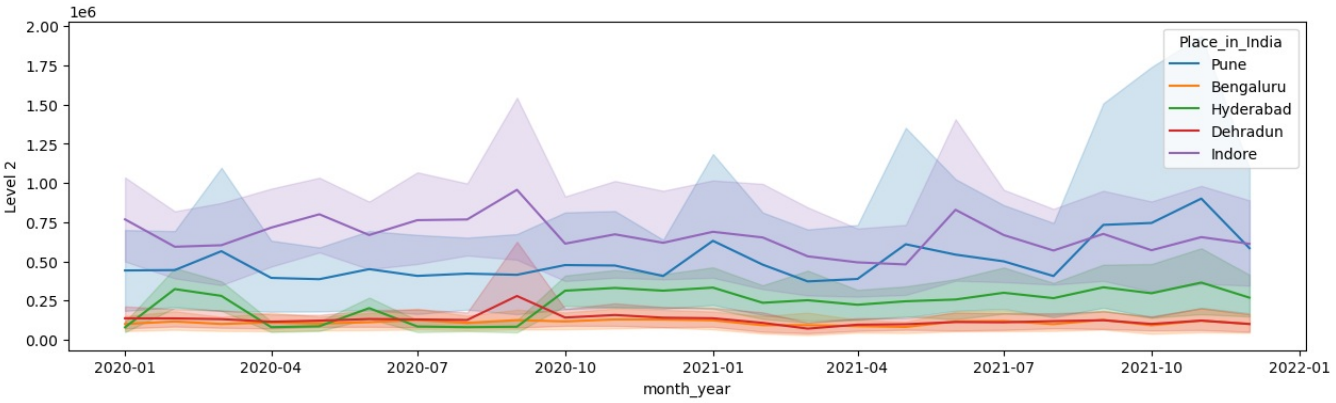
In [29]:
```
dfp = df.copy()
dfp['month_year'] = df['Month'].astype(str)+ '-' +df['Year'].astype(str).str.zfill(2)

dfp['month_year'] = pd.to_datetime(dfp['month_year'], format='%m-%Y')

# sort the DataFrame by the month_year column in ascending order
dfp = dfp.sort_values(by='month_year')
dfp
```

Out[29]:

	Year	Month	Laptop/Desktop	Type_of_Customers?	Coming from	Place_in_India	Level 1	Level 2	Level 3	Level 4	month_year
239	2020	1	Laptop_Website	New_Customer	Unidentified_Sources	Pune	201839.0	127107.0	37986	21543	2020-01-01
209	2020	1	Desktop_Website	New_Customer	Unidentified_Sources	Pune	182169.0	124750.0	43132	24175	2020-01-01
210	2020	1	Laptop_Website	Existing_Customer	From LinkedIn	Bengaluru	NaN	NaN	47007	12983	2020-01-01
211	2020	1	Laptop_Website	Existing_Customer	From LinkedIn	Hyderabad	NaN	NaN	18421	4578	2020-01-01
212	2020	1	Laptop_Website	Existing_Customer	From LinkedIn	Dehradun	NaN	NaN	34689	10200	2020-01-01
...	...	...	...	...	...	...	...	...	...	...	...
1953	2022	12	Laptop_Website	Existing_Customer	From LinkedIn	Indore	NaN	NaN	79808	26288	2022-12-01
1952	2022	12	Laptop_Website	Existing_Customer	From LinkedIn	Dehradun	NaN	NaN	26588	7986	2022-12-01
1951	2022	12	Laptop_Website	Existing_Customer	From LinkedIn	Hyderabad	NaN	NaN	79805	24035	2022-12-01
1949	2022	12	Desktop_Website	New_Customer	Unidentified_Sources	Pune	154252.0	58434.0	22978	11343	2022-12-01
1979	2022	12	Laptop_Website	New_Customer	Unidentified_Sources	Pune	321848.0	123688.0	48452	18697	2022-12-01

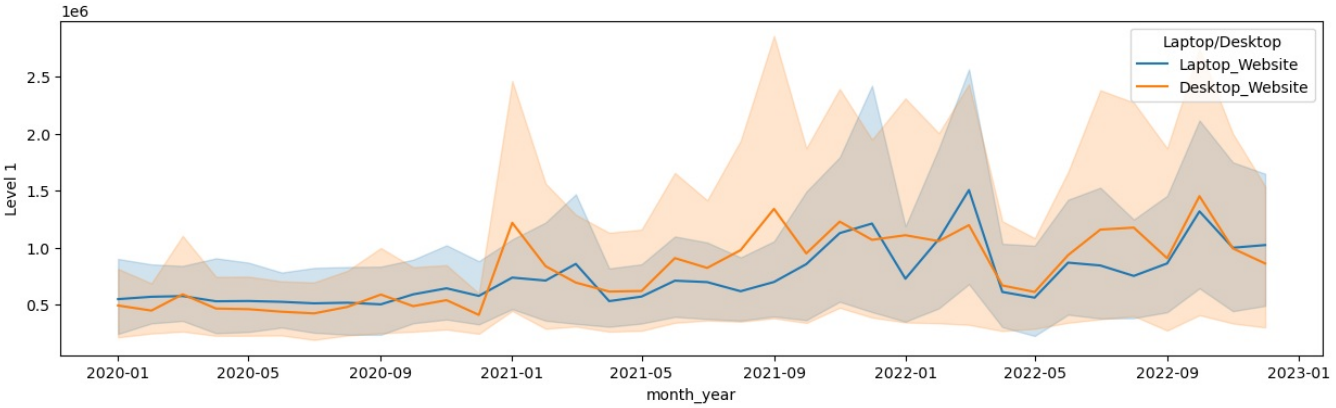
2160 rows × 11 columns

In [30]:
```
# 1.
# filter the DataFrame to only include rows with Year values of 2020 or 2021
plt.figure(figsize=(15, 4))
sn.lineplot(x='month_year', y='Level 2', hue='Place_in_India', data=dfp[dfp['Year'].isin([2020, 2021])])
plt.show()
''' Insights:
    For Year 2020, Indore was the ebst performing Place in India. As we move into 2021,
    Pune has been competing with Indore and outperforms others in 2nd half of the year.
'''
```



Out[30]:
```
' Insights:\n    For Year 2020, Indore was the ebst performing Place in India. As we move into 2021, \n    Pune has been competing with Indore and outperforms others in 2nd half of the year.\n'
```

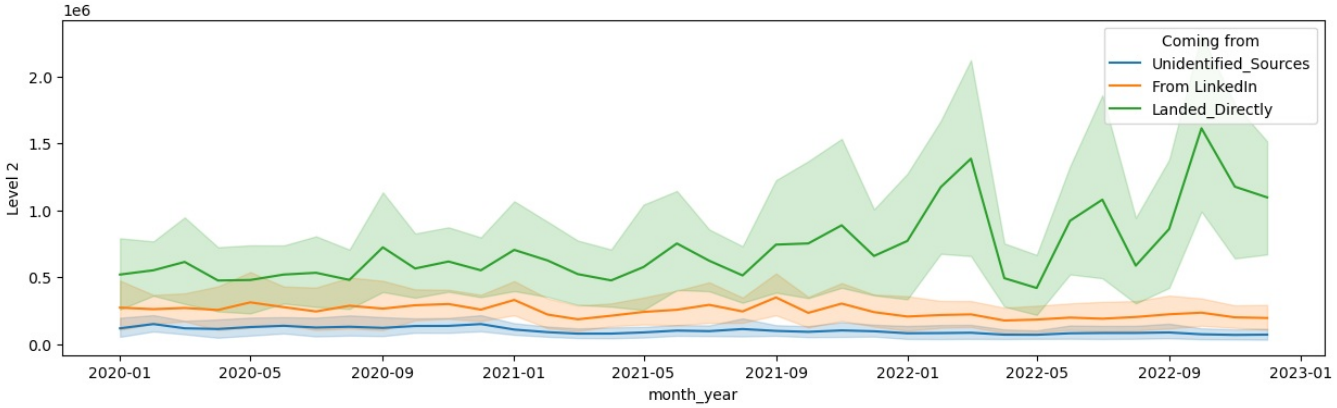
In [31]:
```
# 2
plt.figure(figsize=(15, 4))
sn.lineplot(x='month_year', y='Level 1', hue='Laptop/Desktop', data=dfp)
plt.show()
''' Insights:
    We have same nUmber of Users from both the sources.
'''
```



Out[31]:
```
' Insights:\n    We have same nUmber of Users from both the sources.\n'
```

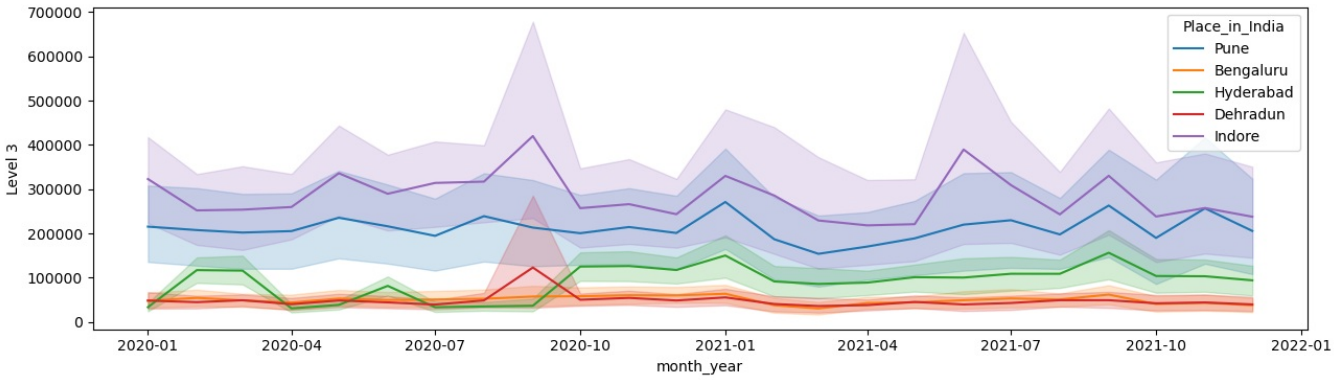
In [32]:
```
# 3
plt.figure(figsize=(15, 4))
sn.lineplot(x='month_year', y='Level 2', hue='Coming from', data=dfp)
plt.show()
''' Insights:
    Most of the customers have landed drectly to the website, indicating good marketting campaigns.
```

```
'''
```



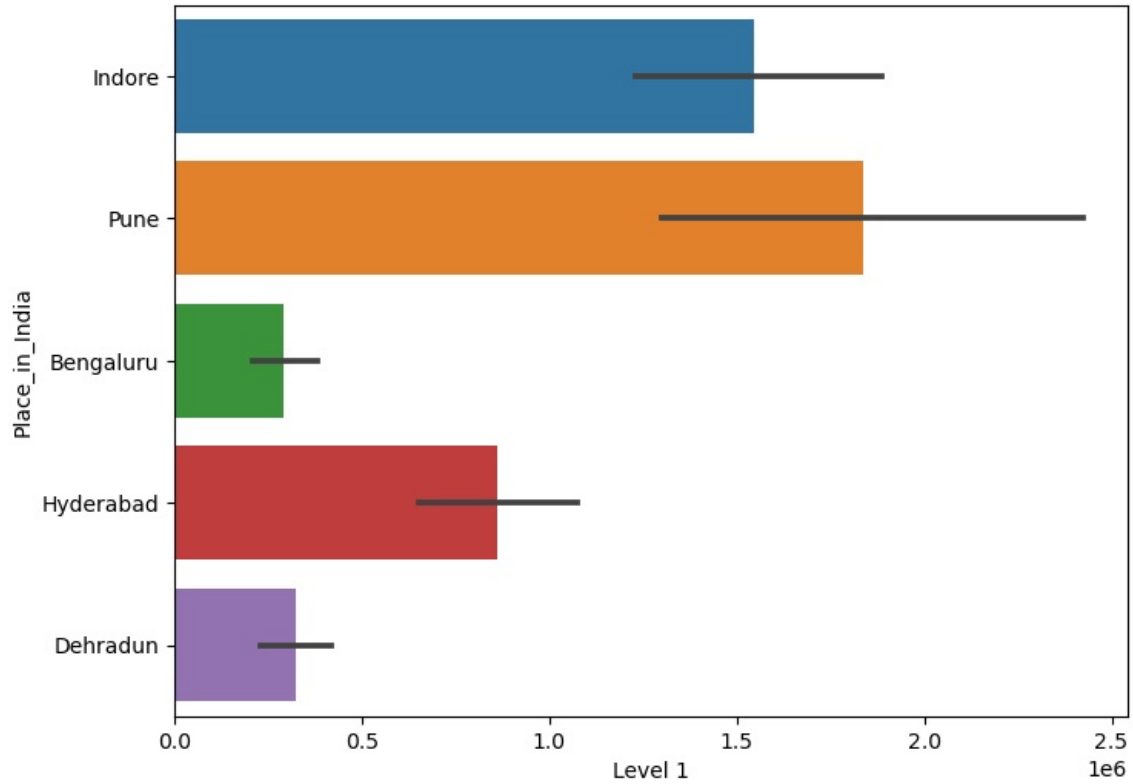
Out[32]:
```
' Insights:\n    Most of the customers have landed drectly to the website, indicating good marketting campaigns
. \n'
```

In [33]:
```
# 5
plt.figure(figsize=(15, 4))
sn.lineplot(x='month_year', y='Level 3', hue='Place_in_India', data=dfp[dfp['Year'].isin([2020, 2021])])
plt.show()
''' Insights:
    We can conclude that most most of our customers belong to INdore and Pune.
'''
```



Out[33]:
```
' Insights:\n    We can conclude that most most of our customers belong to INdore and Pune.\n'
```

In [34]:
```
# 6
plt.figure(figsize=(8, 6))
sn.barplot(x='Level 1', y='Place_in_India',data=dfp[dfp['Year'].isin([2022])])

plt.show()
```

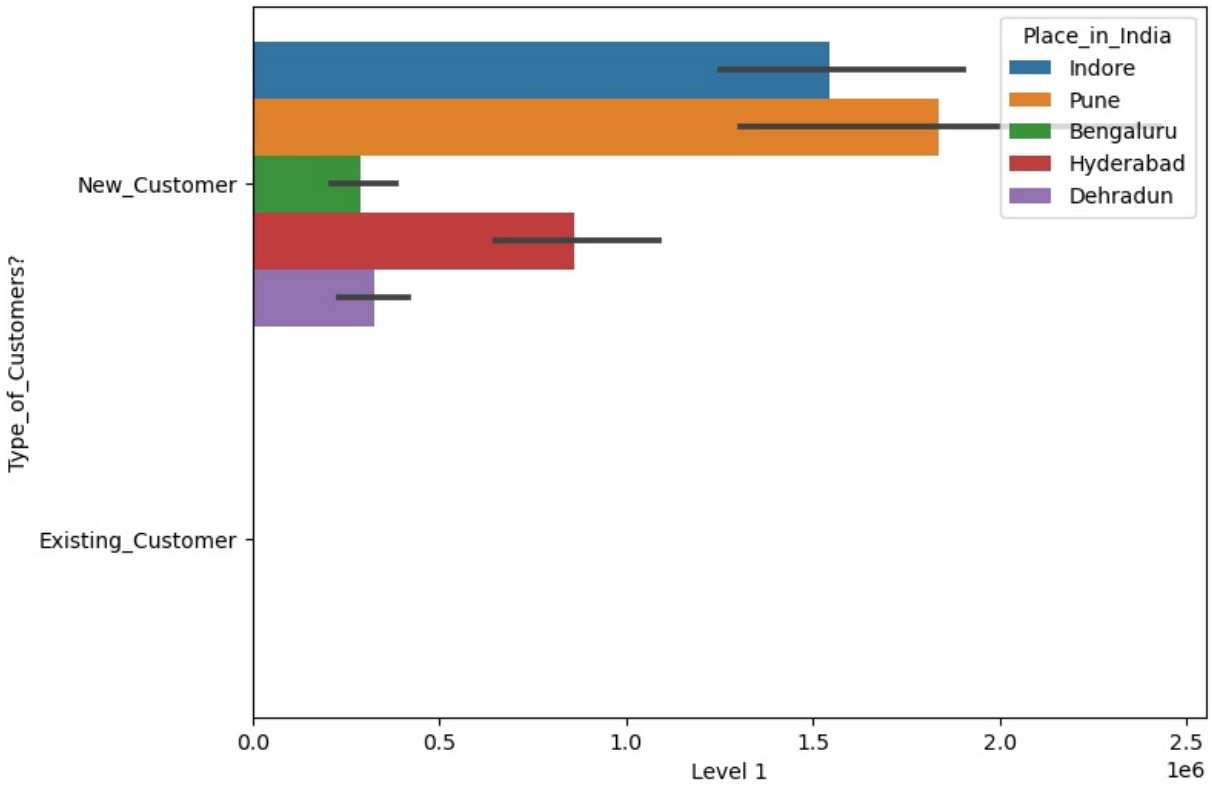


In [39]:
```
dfpp = dfp[dfp["Year"] == 2022]
```

```
dfpp[dfpp["Type_of_Customers?"] == "Existing_Customer"]["Level 1"].max()
# Result of this code demostrates why we dont have any data in existing customer section.
```

Out[39]:
```
nan
```

In [40]:
```
# 6
plt.figure(figsize=(8, 6))
sn.barplot(x='Level 1', y='Type_of_Customers?', hue="Place_in_India", data=dfp[dfp['Year'].isin([2022])])
plt.show()
```



Part 6: About the Projects:

Title: Comparison of Machine Learning Algorithms for Predicting Bioactivity of PKM2 Modulators

In this ongoing project, I have been working with my professor to develop machine learning algorithms for predicting the biological activity of various molecular targets. This research has the potential to be a significant advancement in cancer research, as the protein we are studying is a key target for drug discovery. Specifically, we have used a range of machine learning models and compared their performance in terms of accuracy and precision. To optimize the models' performance, we have applied various feature selection techniques to identify the most relevant features from a dataset of 1886 bioactive descriptors. Working with biological data can be challenging, as it often requires extensive cleaning, selection, and transformation. While we have achieved some promising results so far, I will not able to share specific details at this time due to the confidential nature of the project.

Title: Predicting Parkinson's Disease Outcomes Using Machine Learning Algorithms

In this side project, I applied the XGBoost algorithm to predict outcomes in Parkinson's disease data. I computed the accuracy of the model and generated visualizations to help understand the results. I also identified the most important features that contributed to the prediction. By using machine learning techniques, I was able to gain insights into the factors that may influence the course of the disease and potentially inform treatment decisions. Overall, this project was a valuable learning opportunity and allowed me to explore the potential of machine learning in the context of healthcare data.</i>

Part 7: Time Management:

If selected for this full-time internship, I am committed to maximizing my productivity and delivering high-quality work. To do this, I plan to prioritize my tasks and manage my time effectively by setting clear goals and allocating my time accordingly. I will also break larger tasks into smaller ones to make them more manageable. I am confident in my ability to effectively manage my time, as I have a strong track record in this area and the next six months to focus solely on this internship without any other commitments. I am grateful for the opportunity to intern with your company and am excited to contribute my skills and knowledge to your team. Thank you for considering me for this opportunity.