Fyttlyf Data Science Team Test Solution

Part 0: Reading the data

In [1]:

```
# import the needed libraries
import pandas as pd
```

In [2]:

```
data_path = 'Fytlyff_DS_Interview.csv'

# read the dataset as a DataFrame
dataset = pd.read_csv(data_path)

dataset.head()    # show the first 5 rows from our dataset
```

Out[2]:

	Year	Month	MobileWeb_or_Web	Type_of_Customers?	Where_Are_They_comming_from?	Which_Place_in_India?	How_many_Landed_on_our_Page?	How_many_Landed_on_the_our_Pa
0	2019	Jan	Desktop_Website	Existing_Customer	Came_From_Google	Bangalore	NaN	
1	2019	Jan	Desktop_Website	Existing_Customer	Came_From_Google	Chennai	NaN	
2	2019	Jan	Desktop_Website	Existing_Customer	Came_From_Google	Dehradun	NaN	
3	2019	Jan	Desktop_Website	Existing_Customer	Came_From_Google	Indore	NaN	
4	2019	Jan	Desktop_Website	Existing_Customer	Came_From_Google	Pune	NaN	

Variables and their description for Fyttlyf DS Interview Test

We see that our data has 2160 rows (cases) and 10 columns (features), The columns, their data type and their description are shown in the table below:

Variable	Type	Description
Year (A)	Numerical (Discrete)	The year when the customer comming
Month (B)	Categorical (Nominal)	The month when the customer comming
MobileWeb_or_Web (C)	Categorical (Nominal)	The service requested by the customer
Type_of_Customers? (D)	Categorical (Nominal)	The customer status
Where_Are_They_comming_from? (E)	Categorical (Nominal)	The company/organization where the customer comming from
Which_Place_in_India? (F)	Categorical (Nominal)	Customer city
How_many_Landed_on_our_Page? (G)	Numerical (Discrete)	Number of our page visits
How_many_Landed_on_the_our_Page_and_clicked_on_a_button? (H)	Numerical (Discrete)	Number of clicks on a button when landing the page
How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form? (I)	Numerical (Discrete)	Number of clicks on a button when landing the page and filling the form
How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form_and_Completed_and_submited_the_form? (J)	Numerical (Discrete)	Number of clicks on a button when landing the page and filling the form and completed and submitted the form

Part 1: Data cleaning

In [3]:

```
def data_cleaning(dataframe):
    """
    This function will perform the following:
        1. Replace the NA values with 0s in the data
        2. In column 'B' replace Jan with 1, feb with 2, march with 3 and so on…
        3. In column 'E' Replace "Came_From_Google" with "Google" and "Landed_on_the_page_Directly" with "Direct_traffic"

    Parameters:
        dataframe (dataFrame): The DataFrame of the dataset we used

    Returns
        df (dataFrame): The cleaned data frame

    """
    df = dataframe.copy(deep=True)     # get a copy of the dataframe

    # renaming columns
    df.columns = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']

    # get the columns that have NaN values
    null_series = df.isnull().any()     # check if there is any missing across each column
    nan_cols = []     # to save the columns that have nan values
    for col, val in null_series.iteritems():
        if val:
```

```python
            nan_cols.append(col)

    # replace the nan values with zeros in the dataset
    # Note: (by exploring the data we know that the nan values are only in the numerical data)
    for nan_col in nan_cols:
        df[nan_col].fillna(0, inplace=True)

    # replace months with numbers
    months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
    df['B'] = df['B'].replace(months, nums)

    # replace "Came_From_Google" with "Google" and "Landed_on_the_page_Directly" with "Direct_traffic"
    old_vals = ['Came_From_Google', 'Landed_on_the_page_Directly']
    new_vals = ['Google', 'Direct_traffic']
    df['E'] = df['E'].replace(old_vals, new_vals)

    df.columns = dataframe.columns
    # print(dataframe.columns)

    return df
```

In [4]:

```python
# cleaning_data(dataset)
cleaned_data = data_cleaning(dataset)
cleaned_data.head()
```

Out[4]:

	Year	Month	MobileWeb_or_Web	Type_of_Customers?	Where_Are_They_comming_from?	Which_Place_in_India?	How_many_Landed_on_our_Page?	How_many_Landed_on_the_our_Pa
0	2019	1	Desktop_Website	Existing_Customer	Google	Bangalore	0.0	
1	2019	1	Desktop_Website	Existing_Customer	Google	Chennai	0.0	
2	2019	1	Desktop_Website	Existing_Customer	Google	Dehradun	0.0	
3	2019	1	Desktop_Website	Existing_Customer	Google	Indore	0.0	
4	2019	1	Desktop_Website	Existing_Customer	Google	Pune	0.0	

Part 2: Descriptive statistics

In [5]:

```python
def descriptive_stats(dataframe):
    """
    Generates the summary statistics (Mean, Median, Quartile, standard deviation)
    of all the numerical columns
    Produce a list of all the unique values & data types present in the non-numeric columns

    Parameters:
        dataframe (dataFrame): The cleaned DataFrame of the dataset we used

    Returns
        summary_stat (dataFrame): A dataframe contains the summary statistics of the numerical data
        unique_vals (list): A list of all the unique values in the categorical data and their data type
    """

    df = dataframe.copy(deep=True)     # get a copy of the dataframe

    # Generate the summary
    summary_stat = df.describe()

    # get the unique values of the non-numeric columns
    unique_vals = []
    for col, val in df.iteritems():
        if col not in summary_stat.columns:
            unique_vals.append([col, df[col].unique()])

    return summary_stat, unique_vals
```

In [6]:

```python
summary_statstics, unique_values = descriptive_stats(cleaned_data)
```

In [7]:

```python
summary_statstics
```

Out[7]:

	Year	Month	How_many_Landed_on_our_Page?	How_many_Landed_on_the_our_Page_and_clicked_on_a_button?	How_many_Landed_on_the_our_Page_and_clicked_on_
count	2160.000000	2160.000000	2.160000e+03	2.160000e+03	
mean	2020.000000	6.500000	3.922474e+05	1.792281e+05	
std	0.816686	3.452852	9.555773e+05	3.951562e+05	
min	2019.000000	1.000000	0.000000e+00	0.000000e+00	
25%	2019.000000	3.750000	0.000000e+00	0.000000e+00	
50%	2020.000000	6.500000	1.228350e+04	4.212500e+03	
75%	2021.000000	9.250000	3.816422e+05	1.730452e+05	
max	2021.000000	12.000000	1.127413e+07	4.079301e+06	

In [8]:

```
unique_values
```

Out[8]:

```
[['MobileWeb_or_Web',
  array(['Desktop_Website', 'Mobile_website'], dtype=object)],
 ['Type_of_Customers?',
  array(['Existing_Customer', 'New_Customer'], dtype=object)],
 ['Where_Are_They_comming_from?',
  array(['Google', 'Direct_traffic', 'Unidentified_Sources'], dtype=object)],
 ['Which_Place_in_India?',
  array(['Bangalore', 'Chennai', 'Dehradun', 'Indore', 'Pune'], dtype=object)]]
```

Part 3: Prescriptive statistics

1. “Which_Place_in_India?” has the highest “How_many_Landed_on_the_our_Page?”

In [9]:

```
cond_df = cleaned_data.loc[cleaned_data["How_many_Landed_on_our_Page?"] == cleaned_data["How_many_Landed_on_our_Page?"].max()]

cond_df
```

Out[9]:

	Year	Month	MobileWeb_or_Web	Type_of_Customers?	Where_Are_They_comming_from?	Which_Place_in_India?	How_many_Landed_on_our_Page?	How_many_Landed_on_the_our_
984	2020	5	Desktop_Website	New_Customer	Direct_traffic	Pune	11274131.0	

In [10]:

```
place = cond_df["Which_Place_in_India?"]
place
```

Out[10]:

```
984    Pune
Name: Which_Place_in_India?, dtype: object
```

In [11]:

```
print(f'The place in India which has the highest number of page visits is {place[984]}')
```

```
The place in India which has the highest number of page visits is Pune
```

2. “How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form_and_Completed_and_submited_the_fo divided by “How_many_Landed_on _our_Page?” is highest for “Which_Place_in_India?”

Before exploring this values, we know that the “How_many_Landed_on _our_Page?” column has 0 values which lead to infinty when finding the percentage of “How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form_and_Completed_and_submited_the_form?” from “How_many_Landed_on _our_Page?”. Thus, we need to fill 0s with more reasonable value, we can use the mean, the median, mode, KNN or MICE methods to handle these missing values

First we will explore filling data with the mean

In [12]:

```
# renaming columns to make it easier while using
cleaned_data.columns = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

In [13]:

```
mean_df = cleaned_data.copy(deep=True)

mean = mean_df['G'].mean()

# change 0s with the mean
mean_df['G'].replace(0, mean, inplace=True)

# get the percentage
mean_df['Percentage_submit_landed'] = mean_df['J'] / mean_df['G']
# mean_df.head()

# # get the rows that have the max percentage
high_percent_rows_mean = mean_df.loc[mean_df['Percentage_submit_landed'] == mean_df['Percentage_submit_landed'].max()]
high_percent_mean = high_percent_rows_mean['Percentage_submit_landed'][high_percent_rows_mean.index[0]] * 100
place_mean = high_percent_rows_mean["F"][high_percent_rows_mean.index[0]]

print(f'{place_mean} has the highest percentage of submitted forms which is {high_percent_mean}')
```

```
Dehradun has the highest percentage of submitted forms which is 61.8403568548801
```

Second, We will explore filling data with the median

In [14]:

```
median_df = cleaned_data.copy(deep=True)

median = median_df['G'].median()

if median < median_df['J'].max():
```

```
    print("This is not reasonable to have submitted forms greater than the number od visits of the page")

else:
    # change 0s with the median value
    median_df['G'].replace(0, median, inplace=True)

    # get the percentage
    median_df['Percentage_submit_landed'] = median_df['J'] / median_df['G']
#     median_df.head()

    # get the rows that have the max percentage
    high_percent_rows_median = median_df.loc[median_df['Percentage_submit_landed'] == median_df['Percentage_submit_landed'].max()]
#     high_percent_rows_median
    high_percent_median = high_percent_rows_median['Percentage_submit_landed'][high_percent_rows_median.index[0]] * 100
    place_median = high_percent_rows_median["F"][high_percent_rows_median.index[0]]

    print(f'{place_median} has the highest percentage of submitted forms which is {high_percent_median}')
```

```
This is not reasonable to have submitted forms greater than the number od visits of the page
```

By exploring filling the null data in “How_many_Landed_on _our_Page?” by the median value, we found that there are some cases who submitted the form but doesn't land on the page (number of submissions > number of visits), which is not reasonable in this case. Thus, we cannot handle this with the median value.

Third, We will explore filling data with the mode

In [15]:

```
mode_df = cleaned_data.copy(deep=True)

mode = mode_df['G'].mode()

if mode[0] == 0:
    print("The mode for this feature is 0, It's not useful for handling missing values")

else:
    # change 0s with the median value
    mode_df['G'].replace(0, mode, inplace=True)

    # get the percentage
    mode_df['Percentage_submit_landed'] = mode_df['J'] / mode_df['G']
#     mode_df.head()

    # get the rows that have the max percentage
    high_percent_rows_mode = mode_df.loc[mode_df['Percentage_submit_landed'] == mode_df['Percentage_submit_landed'].max()]
#     high_percent_rows_mode
    high_percent_mode = high_percent_rows_mode['Percentage_submit_landed'][high_percent_rows_mode.index[0]] * 100
    place_mode = high_percent_rows_mode["F"][high_percent_rows_mode.index[0]]

    print(f'{place_mode} has the highest percentage of submitted forms which is {high_percent_mode}')
```

```
The mode for this feature is 0, It's not useful for handling missing values
```

Fourth, We will explore filling data using KNN imputer

In [16]:

```
from sklearn.impute import KNNImputer
import numpy as np
```

In [17]:

```
knn_df = cleaned_data.copy(deep=True)

imputer = KNNImputer(missing_values=0, n_neighbors=5)    # initiate the imputer
g = np.array(knn_df['G']).reshape(-1, 1)
df_filled = imputer.fit_transform(g)    # Fill the missing values in G

knn_df['G'] = df_filled

# get the percentage
knn_df['Percentage_submit_landed'] = knn_df['J'] / knn_df['G']

# # get the rows that have the max percentage
high_percent_rows_knn = knn_df.loc[knn_df['Percentage_submit_landed'] == knn_df['Percentage_submit_landed'].max()]
high_percent_knn = high_percent_rows_knn['Percentage_submit_landed'][high_percent_rows_knn.index[0]] * 100
place_knn = high_percent_rows_knn["F"][high_percent_rows_knn.index[0]]

print(f'{place_knn} has the highest percentage of submitted forms which is {high_percent_knn}')
```

```
Dehradun has the highest percentage of submitted forms which is 61.8403568548801
```

We notice that the KNN method gives the same result as when using the mean

Part 4: Simple Machine learning questions

In [18]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_percentage_error
```

In [19]:

```
def create_date_frame(dataframe):
    date_frame = dataframe.copy(deep=True)

    date_frame['Date'] = pd.to_datetime(date_frame.A.astype(str) + '/' + date_frame.B.astype(str) + '/01')
```

```
    date_frame['Time'] = np.arange(len(date_frame.index)) # create time dummies for training

    return date_frame
```

In [20]:

```
df = cleaned_data.copy(deep=True)
df = create_date_frame(cleaned_data)

X = df.loc[:, 'Time'].to_numpy() # x_train
y = df.loc[:, 'J'].to_numpy()     # y_train

df.head()
```

Out[20]:

	A	B	C	D	E	F	G	H	I	J	Date	Time
0	2019	1	Desktop_Website	Existing_Customer	Google	Bangalore	0.0	0.0	56892	17178	2019-01-01	0
1	2019	1	Desktop_Website	Existing_Customer	Google	Chennai	0.0	0.0	41460	11916	2019-01-01	1
2	2019	1	Desktop_Website	Existing_Customer	Google	Dehradun	0.0	0.0	55561	19461	2019-01-01	2
3	2019	1	Desktop_Website	Existing_Customer	Google	Indore	0.0	0.0	320923	110667	2019-01-01	3
4	2019	1	Desktop_Website	Existing_Customer	Google	Pune	0.0	0.0	220937	46033	2019-01-01	4

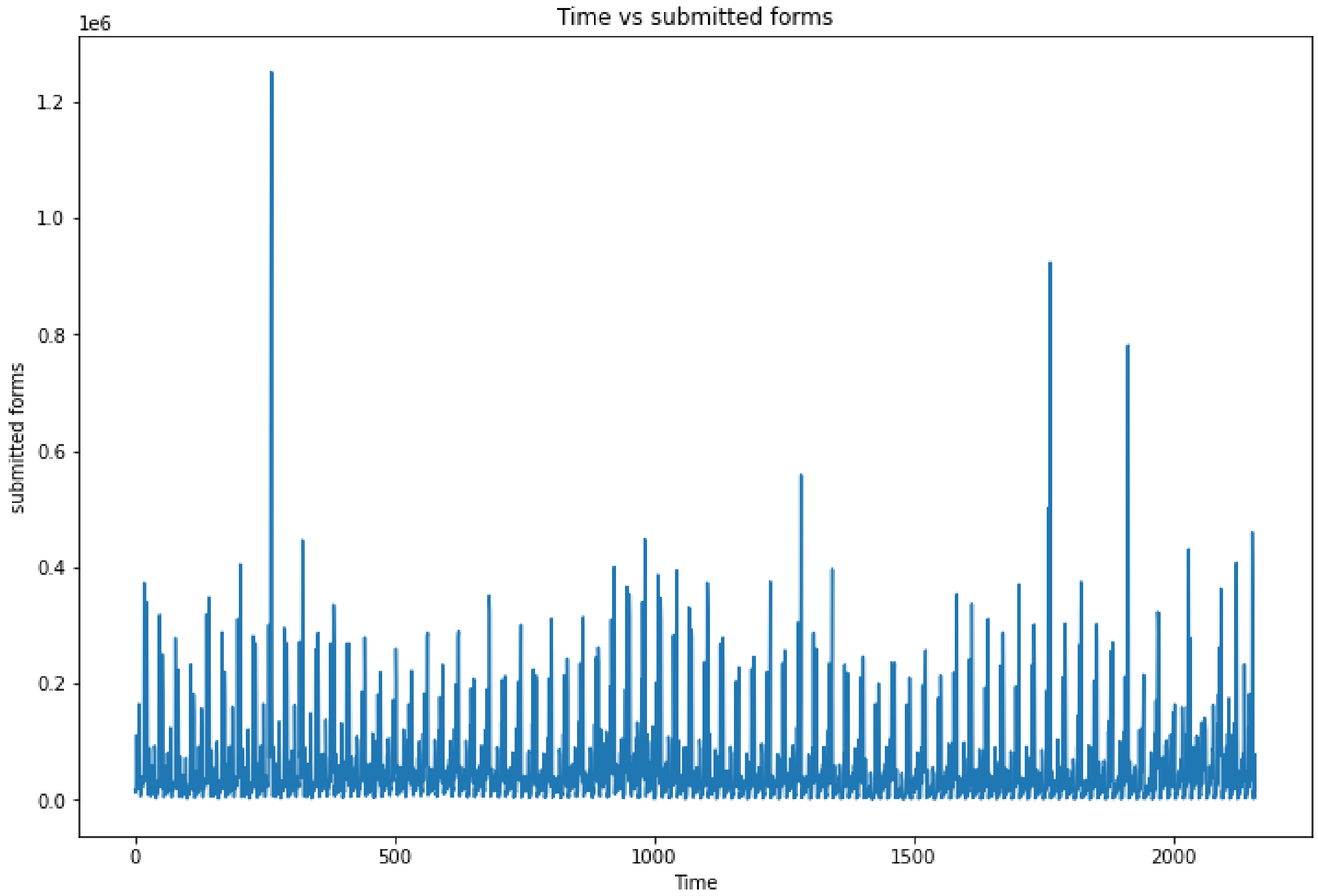
In [21]:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12, 8))

plt.plot(X, y)

# Set the title
plt.title("Time vs submitted forms")
# Set the y-axis label
plt.ylabel('submitted forms')
# Set the x-axis label
plt.xlabel('Time')
plt.show()
```



In [23]:

```
def pred_future(X_train, y_train):
    """
    Predicts "How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form_and_Completed_and_submited_the_
form?"
    for the complete year of 2022
    and generates the overall MAPE of your prediction for the year 2021.
    """
    linear_model = LinearRegression()
    linear_model.fit(X_train.reshape(-1,1), y_train)

    b = linear_model.intercept_
    w = linear_model.coef_

    # predictions for 2022
    x_test_2022 = np.arange(2160, 2880)
    y_pred_2022 = linear_model.predict(x_test_2022.reshape(-1,1))

    # predictions for 2021
    x_test_2021 = X_train[1439:]
    y_pred_2021 = linear_model.predict(x_test_2021.reshape(-1,1))
    mape_2021 = mean_absolute_percentage_error(y_train[1439:], y_pred_2021)

    return y_pred_2022, mape_2021

y_pred_2022, mape_2021 = pred_future(X, y)
```

```python
print(f'predictions for the complete year 2022:\n {y_pred_2022}')
print(f'MAPE of 2021 predictions: {MAPE}')
```

```
predictions for the complete year 2022:
 [58005.47020183 58004.34496258 58003.21972334 58002.0944841
 58000.96924485 57999.84400561 57998.71876637 57997.59352713
 57996.46828788 57995.34304864 57994.2178094  57993.09257015
 57991.96733091 57990.84209167 57989.71685243 57988.59161318
 57987.46637394 57986.3411347  57985.21589545 57984.09065621
 57982.96541697 57981.84017773 57980.71493848 57979.58969924
 57978.46446    57977.33922075 57976.21398151 57975.08874227
 57973.96350303 57972.83826378 57971.71302454 57970.5877853
 57969.46254605 57968.33730681 57967.21206757 57966.08682833
 57964.96158908 57963.83634984 57962.7111106  57961.58587135
 57960.46063211 57959.33539287 57958.21015363 57957.08491438
 57955.95967514 57954.8344359  57953.70919666 57952.58395741
 57951.45871817 57950.33347893 57949.20823968 57948.08300044
 57946.9577612  57945.83252196 57944.70728271 57943.58204347
 57942.45680423 57941.33156498 57940.20632574 57939.0810865
 57937.95584726 57936.83060801 57935.70536877 57934.58012953
 57933.45489028 57932.32965104 57931.2044118  57930.07917256
 57928.95393331 57927.82869407 57926.70345483 57925.57821558
 57924.45297634 57923.3277371  57922.20249786 57921.07725861
 57919.95201937 57918.82678013 57917.70154088 57916.57630164
 57915.4510624  57914.32582316 57913.20058391 57912.07534467
 57910.95010543 57909.82486618 57908.69962694 57907.5743877
 57906.44914846 57905.32390921 57904.19866997 57903.07343073
 57901.94819148 57900.82295224 57899.697713   57898.57247376
 57897.44723451 57896.32199527 57895.19675603 57894.07151679
 57892.94627754 57891.8210383  57890.69579906 57889.57055981
 57888.44532057 57887.32008133 57886.19484209 57885.06960284
 57883.9443636  57882.81912436 57881.69388511 57880.56864587
 57879.44340663 57878.31816739 57877.19292814 57876.0676889
 57874.94244966 57873.81721041 57872.69197117 57871.56673193
 57870.44149269 57869.31625344 57868.1910142  57867.06577496
 57865.94053571 57864.81529647 57863.69005723 57862.56481799
 57861.43957874 57860.3143395  57859.18910026 57858.06386101
 57856.93862177 57855.81338253 57854.68814329 57853.56290404
 57852.4376648  57851.31242556 57850.18718631 57849.06194707
 57847.93670783 57846.81146859 57845.68622934 57844.5609901
 57843.43575086 57842.31051161 57841.18527237 57840.06003313
 57838.93479389 57837.80955464 57836.6843154  57835.55907616
 57834.43383692 57833.30859767 57832.18335843 57831.05811919
 57829.93287994 57828.8076407  57827.68240146 57826.55716222
 57825.43192297 57824.30668373 57823.18144449 57822.05620524
 57820.930966   57819.80572676 57818.68048752 57817.55524827
 57816.43000903 57815.30476979 57814.17953054 57813.0542913
 57811.92905206 57810.80381282 57809.67857357 57808.55333433
 57807.42809509 57806.30285584 57805.1776166  57804.05237736
 57802.92713812 57801.80189887 57800.67665963 57799.55142039
 57798.42618114 57797.3009419  57796.17570266 57795.05046342
 57793.92522417 57792.79998493 57791.67474569 57790.54950644
 57789.4242672  57788.29902796 57787.17378872 57786.04854947
 57784.92331023 57783.79807099 57782.67283174 57781.5475925
 57780.42235326 57779.29711402 57778.17187477 57777.04663553
 57775.92139629 57774.79615704 57773.6709178  57772.54567856
 57771.42043932 57770.29520007 57769.16996083 57768.04472159
 57766.91948235 57765.7942431  57764.66900386 57763.54376462
 57762.41852537 57761.29328613 57760.16804689 57759.04280765
 57757.9175684  57756.79232916 57755.66708992 57754.54185067
 57753.41661143 57752.29137219 57751.16613295 57750.0408937
 57748.91565446 57747.79041522 57746.66517597 57745.53993673
 57744.41469749 57743.28945825 57742.164219   57741.03897976
 57739.91374052 57738.78850127 57737.66326203 57736.53802279
 57735.41278355 57734.2875443  57733.16230506 57732.03706582
 57730.91182657 57729.78658733 57728.66134809 57727.53610885
 57726.4108696  57725.28563036 57724.16039112 57723.03515187
 57721.90991263 57720.78467339 57719.65943415 57718.5341949
 57717.40895566 57716.28371642 57715.15847717 57714.03323793
 57712.90799869 57711.78275945 57710.6575202  57709.53228096
 57708.40704172 57707.28180248 57706.15656323 57705.03132399
 57703.90608475 57702.7808455  57701.65560626 57700.53036702
 57699.40512778 57698.27988853 57697.15464929 57696.02941005
 57694.9041708  57693.77893156 57692.65369232 57691.52845308
 57690.40321383 57689.27797459 57688.15273535 57687.0274961
 57685.90225686 57684.77701762 57683.65177838 57682.52653913
 57681.40129989 57680.27606065 57679.1508214  57678.02558216
 57676.90034292 57675.77510368 57674.64986443 57673.52462519
 57672.39938595 57671.2741467  57670.14890746 57669.02366822
 57667.89842898 57666.77318973 57665.64795049 57664.52271125
 57663.397472   57662.27223276 57661.14699352 57660.02175428
 57658.89651503 57657.77127579 57656.64603655 57655.5207973
 57654.39555806 57653.27031882 57652.14507958 57651.01984033
 57649.89460109 57648.76936185 57647.64412261 57646.51888336
 57645.39364412 57644.26840488 57643.14316563 57642.01792639
 57640.89268715 57639.76744791 57638.64220866 57637.51696942
 57636.39173018 57635.26649093 57634.14125169 57633.01601245
 57631.89077321 57630.76553396 57629.64029472 57628.51505548
 57627.38981623 57626.26457699 57625.13933775 57624.01409851
 57622.88885926 57621.76362002 57620.63838078 57619.51314153
 57618.38790229 57617.26266305 57616.13742381 57615.01218456
 57613.88694532 57612.76170608 57611.63646683 57610.51122759
 57609.38598835 57608.26074911 57607.13550986 57606.01027062
 57604.88503138 57603.75979213 57602.63455289 57601.50931365
 57600.38407441 57599.25883516 57598.13359592 57597.00835668
 57595.88311743 57594.75787819 57593.63263895 57592.50739971
 57591.38216046 57590.25692122 57589.13168198 57588.00644274
 57586.88120349 57585.75596425 57584.63072501 57583.50548576
 57582.38024652 57581.25500728 57580.12976804 57579.00452879
 57577.87928955 57576.75405031 57575.62881106 57574.50357182
 57573.37833258 57572.25309334 57571.12785409 57570.00261485
 57568.87737561 57567.75213636 57566.62689712 57565.50165788
```

```
 57568.87737561 57567.75213636 57566.62689712 57565.50165788
 57564.37641864 57563.25117939 57562.12594015 57561.00070091
 57559.87546166 57558.75022242 57557.62498318 57556.49974394
 57555.37450469 57554.24926545 57553.12402621 57551.99878696
 57550.87354772 57549.74830848 57548.62306924 57547.49782999
 57546.37259075 57545.24735151 57544.12211226 57542.99687302
 57541.87163378 57540.74639454 57539.62115529 57538.49591605
 57537.37067681 57536.24543756 57535.12019832 57533.99495908
 57532.86971984 57531.74448059 57530.61924135 57529.49400211
 57528.36876287 57527.24352362 57526.11828438 57524.99304514
 57523.86780589 57522.74256665 57521.61732741 57520.49208817
 57519.36684892 57518.24160968 57517.11637044 57515.99113119
 57514.86589195 57513.74065271 57512.61541347 57511.49017422
 57510.36493498 57509.23969574 57508.11445649 57506.98921725
 57505.86397801 57504.73873877 57503.61349952 57502.48826028
 57501.36302104 57500.23778179 57499.11254255 57497.98730331
 57496.86206407 57495.73682482 57494.61158558 57493.48634634
 57492.36110709 57491.23586785 57490.11062861 57488.98538937
 57487.86015012 57486.73491088 57485.60967164 57484.48443239
 57483.35919315 57482.23395391 57481.10871467 57479.98347542
 57478.85823618 57477.73299694 57476.60775769 57475.48251845
 57474.35727921 57473.23203997 57472.10680072 57470.98156148
 57469.85632224 57468.731083   57467.60584375 57466.48060451
 57465.35536527 57464.23012602 57463.10488678 57461.97964754
 57460.8544083  57459.72916905 57458.60392981 57457.47869057
 57456.35345132 57455.22821208 57454.10297284 57452.9777336
 57451.85249435 57450.72725511 57449.60201587 57448.47677662
 57447.35153738 57446.22629814 57445.1010589  57443.97581965
 57442.85058041 57441.72534117 57440.60010192 57439.47486268
 57438.34962344 57437.2243842  57436.09914495 57434.97390571
 57433.84866647 57432.72342722 57431.59818798 57430.47294874
 57429.3477095  57428.22247025 57427.09723101 57425.97199177
 57424.84675252 57423.72151328 57422.59627404 57421.4710348
 57420.34579555 57419.22055631 57418.09531707 57416.97007782
 57415.84483858 57414.71959934 57413.5943601  57412.46912085
 57411.34388161 57410.21864237 57409.09340313 57407.96816388
 57406.84292464 57405.7176854  57404.59244615 57403.46720691
 57402.34196767 57401.21672843 57400.09148918 57398.96624994
 57397.8410107  57396.71577145 57395.59053221 57394.46529297
 57393.34005373 57392.21481448 57391.08957524 57389.964336
 57388.83909675 57387.71385751 57386.58861827 57385.46337903
 57384.33813978 57383.21290054 57382.0876613  57380.96242205
 57379.83718281 57378.71194357 57377.58670433 57376.46146508
 57375.33622584 57374.2109866  57373.08574735 57371.96050811
 57370.83526887 57369.71002963 57368.58479038 57367.45955114
 57366.3343119  57365.20907265 57364.08383341 57362.95859417
 57361.83335493 57360.70811568 57359.58287644 57358.4576372
 57357.33239795 57356.20715871 57355.08191947 57353.95668023
 57352.83144098 57351.70620174 57350.5809625  57349.45572326
 57348.33048401 57347.20524477 57346.08000553 57344.95476628
 57343.82952704 57342.7042878  57341.57904856 57340.45380931
 57339.32857007 57338.20333083 57337.07809158 57335.95285234
 57334.8276131  57333.70237386 57332.57713461 57331.45189537
 57330.32665613 57329.20141688 57328.07617764 57326.9509384
 57325.82569916 57324.70045991 57323.57522067 57322.44998143
 57321.32474218 57320.19950294 57319.0742637  57317.94902446
 57316.82378521 57315.69854597 57314.57330673 57313.44806748
 57312.32282824 57311.197589   57310.07234976 57308.94711051
 57307.82187127 57306.69663203 57305.57139278 57304.44615354
 57303.3209143  57302.19567506 57301.07043581 57299.94519657
 57298.81995733 57297.69471808 57296.56947884 57295.4442396
 57294.31900036 57293.19376111 57292.06852187 57290.94328263
 57289.81804339 57288.69280414 57287.5675649  57286.44232566
 57285.31708641 57284.19184717 57283.06660793 57281.94136869
 57280.81612944 57279.6908902  57278.56565096 57277.44041171
 57276.31517247 57275.18993323 57274.06469399 57272.93945474
 57271.8142155  57270.68897626 57269.56373701 57268.43849777
 57267.31325853 57266.18801929 57265.06278004 57263.9375408
 57262.81230156 57261.68706231 57260.56182307 57259.43658383
 57258.31134459 57257.18610534 57256.0608661  57254.93562686
 57253.81038761 57252.68514837 57251.55990913 57250.43466989
 57249.30943064 57248.1841914  57247.05895216 57245.93371291
 57244.80847367 57243.68323443 57242.55799519 57241.43275594
 57240.3075167  57239.18227746 57238.05703821 57236.93179897
 57235.80655973 57234.68132049 57233.55608124 57232.430842
 57231.30560276 57230.18036351 57229.05512427 57227.92988503
 57226.80464579 57225.67940654 57224.5541673  57223.42892806
 57222.30368882 57221.17844957 57220.05321033 57218.92797109
 57217.80273184 57216.6774926  57215.55225336 57214.42701412
 57213.30177487 57212.17653563 57211.05129639 57209.92605714
 57208.8008179  57207.67557866 57206.55033942 57205.42510017
 57204.29986093 57203.17462169 57202.04938244 57200.9241432
 57199.79890396 57198.67366472 57197.54842547 57196.42318623]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [23], in <cell line: 27>()
     24 y_pred_2022, mape_2021 = pred_future(X, y)
     26 print(f'predictions for the complete year 2022:\n {y_pred_2022}')
---> 27 print(f'MAPE of 2021 predictions: {MAPE}')

NameError: name 'MAPE' is not defined
```

Part 5: Visualization

A line graph for “How_many_Landed_on_the_our_Page_and_clicked_on_a_button?” for the different “Which_Place_in_India?” over the months of the year 2019 & 2020.

(Hint : On x axis there should be months for 2019 & 2020 and Y axis should be the “How_many_Landed_on_the_our_Page_and_clicked_on_a_button?” and there should different lines depicting different regions of “Which_Place_in_India?”)

In [24]:

```
vis_df = cleaned_data.copy(deep=True)

months_2019_2020 = vis_df[(vis_df['A'] == 2019) | (vis_df['A'] == 2020)]  # get 2019 and 2020 from data

months_2019_2020.head()
```

Out[24]:

	A	B	C	D	E	F	G	H	I	J
0	2019	1	Desktop_Website	Existing_Customer	Google	Bangalore	0.0	0.0	56892	17178
1	2019	1	Desktop_Website	Existing_Customer	Google	Chennai	0.0	0.0	41460	11916
2	2019	1	Desktop_Website	Existing_Customer	Google	Dehradun	0.0	0.0	55561	19461
3	2019	1	Desktop_Website	Existing_Customer	Google	Indore	0.0	0.0	320923	110667
4	2019	1	Desktop_Website	Existing_Customer	Google	Pune	0.0	0.0	220937	46033

From the unique values, we know that we have 5 different places in indea:'Bangalore', 'Chennai', 'Dehradun', 'Indore', 'Pune', do we will make a dataframe for each for ease use

In [25]:

```
Bangalore = months_2019_2020[months_2019_2020['F'] == 'Bangalore']
Bangalore = create_date_frame(Bangalore)

Chennai  = months_2019_2020[months_2019_2020['F'] == 'Chennai']
Chennai  = create_date_frame(Chennai)


Dehradun = months_2019_2020[months_2019_2020['F'] == 'Dehradun']
Dehradun = create_date_frame(Dehradun)

Indore = months_2019_2020[months_2019_2020['F'] == 'Indore']
Indore = create_date_frame(Indore)

Pune = months_2019_2020[months_2019_2020['F'] == 'Pune']
Pune = create_date_frame(Pune)
```

In [26]:

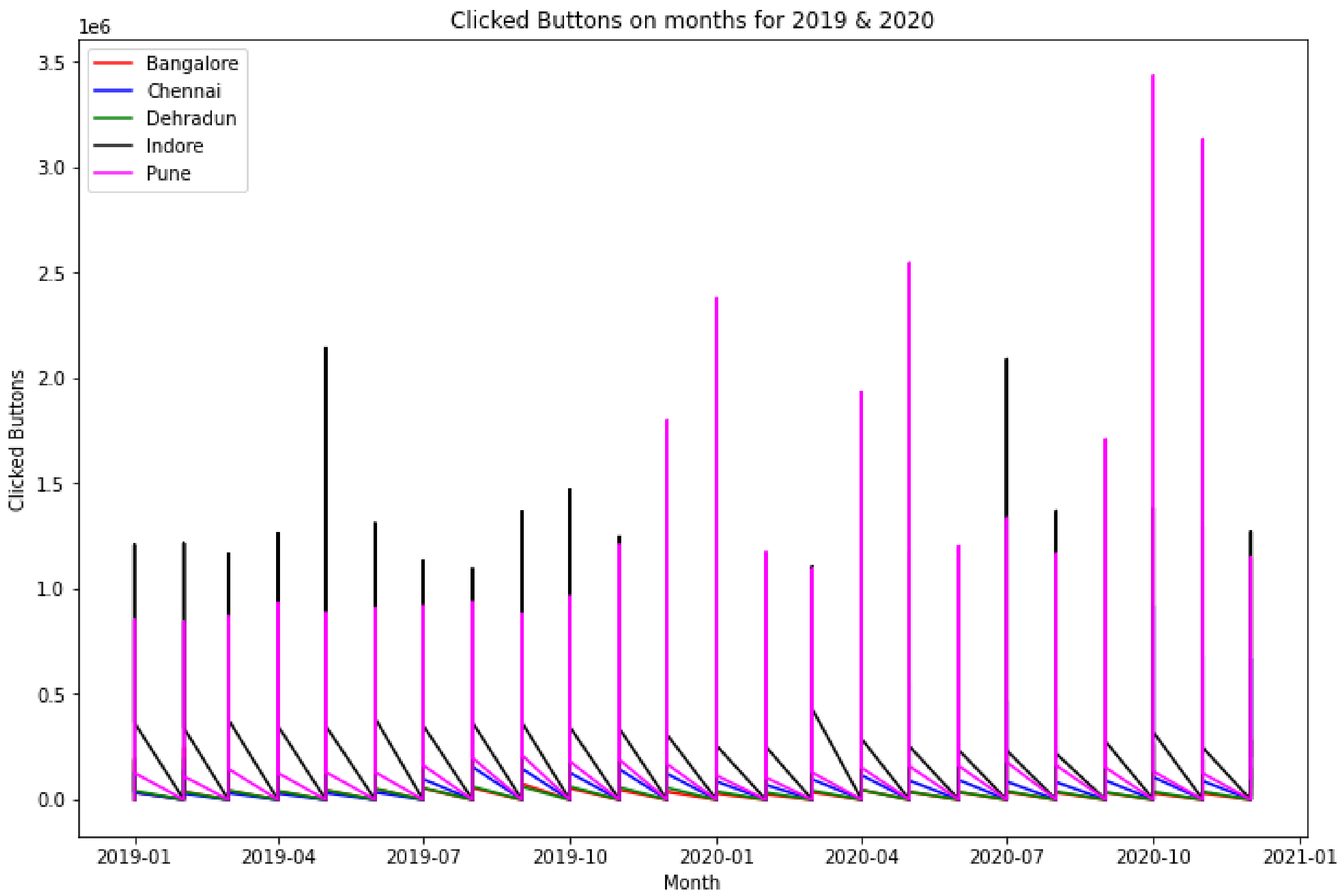
```
fig = plt.figure(figsize=(12, 8))

plt.plot(Bangalore['Date'], Bangalore['H'], color='red', label='Bangalore')
plt.plot(Chennai['Date'], Chennai['H'], color='blue', label='Chennai')
plt.plot(Dehradun['Date'], Dehradun['H'], color='green', label='Dehradun')
plt.plot(Indore['Date'], Indore['H'], color='black', label='Indore')
plt.plot(Pune['Date'], Pune['H'], color='magenta', label='Pune')


plt.title('Clicked Buttons on months for 2019 & 2020')
plt.xlabel('Month')
plt.ylabel('Clicked Buttons')

plt.legend()

plt.show()
```



A line graph of the actual and projected number of "How_many_Landed_on_the_our_Page_and_clicked_on_a_button_and_started_filling_the_Form_and_Completed_and_submited_the_form?" for the months of the year 2021(Actuals values) & 2022 (Predicted values). (Hint : It should be a line graph)

In [27]:

```
fig = plt.figure(figsize=(12, 8))
```

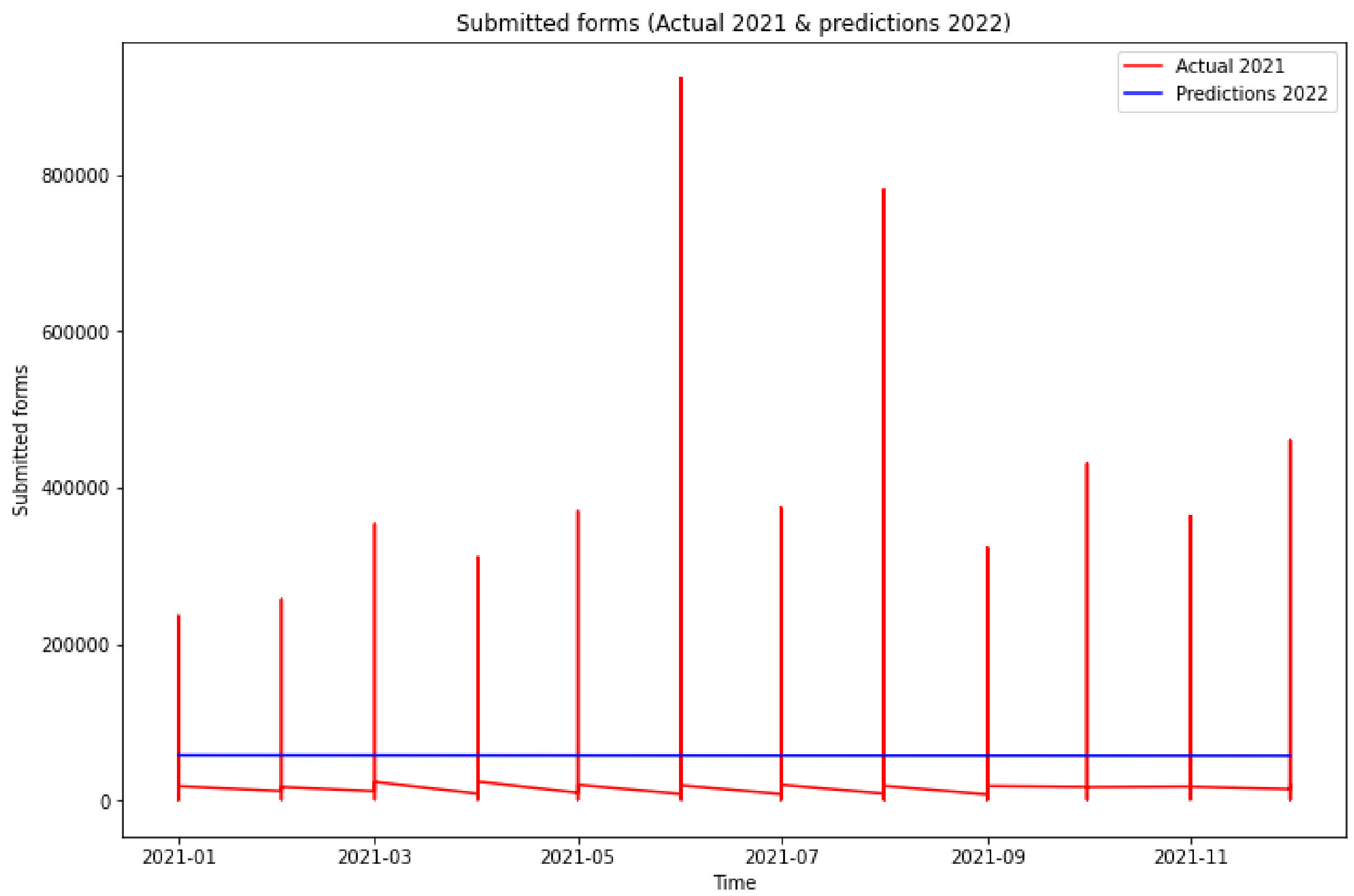
```
actual_2021 = y[1440:]
plt.plot(df.loc[1440:, 'Date'], actual_2021, color='red', label='Actual 2021')

predicted_2022 = y_pred_2022
plt.plot(df.loc[1440:, 'Date'], predicted_2022, color='blue', label='Predictions 2022')

plt.title('Submitted forms (Actual 2021 & predictions 2022)')
plt.xlabel('Time')
plt.ylabel('Submitted forms')

plt.legend()

plt.show()
```



Part 6: About the previous projects

1. **Investigating Netflix Movies and Guest Stars in The Office Project**

This project is just for data analysis. In this project, We invistigated Netflix Movies to get insights about the duration of the movies. We appplied EDA on two categories:

```
1. On friend's data: given the average duartions movie during the period from 2011 to 2020.
2. On a dataset for Netflix that contains large samples of the movies, tv shows, ...etc. during the period from 1925 to 2021
, and of different genres.
```

We also gained experience in Exploratory Data Analysis, allowing us to manipulate raw data, "Netflix data", and draw conclusions based on the visualizations of the data we created.

GitHub: https://github.com/raniaelhagin/Data-Analysis-Projects-and-Excercises-/tree/main/Investigating_Netflix_Movies_and_Guest_Stars_in_The_Office

1. **Dr.Semmelweis and the Discovery of Handwashing**

Also this project was in data analysis. This project is about reanalyzing the data behind one of the most important discoveries of modern medicine, handwashing, and how it was a major cause of childbed fever and by enforcing handwashing, hundreds of lives were saved.

This project have many topics such as Case Studying, Data Manipulation, Data Visualization, Probability and Statistics.

GitHub: https://github.com/raniaelhagin/Data-Analysis-Projects-and-Excercises-/tree/main/Dr._Semmelweis_and_the_Discovery_of_Handwashing

1. **Prediction using supervised ML**

This project was about using linear regression to predict student scores based on their study hours. I used linear regression using a from scratch approach and in another solution I used Scikit-learn. I explored the data and get the relationship between the features. And this project was in The Sparks Foudation Internship, I also will do more projects with them as soon as poossible.

GitHub: https://github.com/raniaelhagin/Data-Science-and-Business-analytics-Internship-TSF-GRIP/tree/main/Projects

Part 7: Time management

If I get selected in this full-time internship, I will try to manage my time as I have a job in company. Usually I work all the day but if I have a task I will give it more concentration in time, I mean I concentrate in the task all the time I have until I finish my work. I can spend over 10 hours working on a task. always taking notes and define the problem I work on save a lot of time and help me manage my thought. So, taking time at the first for taking notes, define our problem, and have an initial thought about how I will manage and prioritize tasks, and be aware of the deadlines to get the solutions save me a lot of time and organize my work.