In [1]:
```
import pandas as pd
import eeweather as ee
from folium import plugins
import folium
from folium.plugins import HeatMap
import plotly.express as px
import warnings
import matplotlib.pyplot as plt
%matplotlib inline
```

Read in all of the `csv` files as pandas DataFrames.

In [2]:
```
df1 = pd.read_csv('Dataset_1.csv')
df2 = pd.read_csv('Dataset_2.csv')
df3 = pd.read_csv('Dataset_3.csv')
```

Assignment #1

All of the code for this and the other assignments can be found at the Github repository below. [https://github.com/grantaguinaldo/bi-work/blob/main/bi-work-assignment-01.ipynb (https://github.com/grantaguinaldo/bi-work/blob/main/bi-work-assignment-01.ipynb)](https://github.com/grantaguinaldo/bi-work/blob/main/bi-work-assignment-01.ipynb)

For `Dataset_1` the following assumptions were made based on the material presented:

- The `Net Score` is defined as the following `100*(%Excellent - sum(%Good, %Fair, %Poor)`
- The units of the `Net Score` is greater than 100 since the component values of the metric are all initially expressed as a percentage, and is by definition, not a percentage.
- The `Net Score` value will be expressed as a `float` value with `one` decimal place, and all other values will be expressed as a `float` value with `four` decimal place.

General Approach

When handed with a data problem, my approach is to first understand what is in the data at a high level. That is to answer the following:

- What is the data actually representing? **The data is representing results from two survey questions that was sent out to a set of customers**
- How many features exists in the raw data? **Five, features exsits including the survey responses, date and time of the results, among others.**
- Are the feature nominal, ordinal, or interval? **The survey results are ordinal, since a `5` means excellent and a `1` means poor.**
- What is the "level of completeness" of the data? **In other words, does the dataset contain missing data?**
- What are the different datatypes contained in the dataset? **There are two different types of data in the dataset, `int` and `char`.**
- Can I determine a high level distribution of the values in the dataset?: **Yes, most of the responses from both questions are a `5`.**

For this project, I conducted an exploratory data analysis (EDA) to answer these questions to have a better understanding of the dataset and problem that I have.

What are the different datatypes contained in the dataset?

From the command below, we see that the six fields provided, are comprised of a mixture of datatypes, including `int`, `obj`. The most notable observation from this, is that `date_of_survey` and `time_of_survey` are **not** time-series objects.

Further, we see that the shape of the dataframe is 2,386 which suggests that we have survey responses from 2,386 customers. In addition, since we see that there are 2,386 values in each column that are `non-null` we can assume that there are no missing values/data in the dataset.

In [3]:
```
print('The dataset contains {} rows, and {} columns.'.format(df1.shape[0], df1.shape[1]))
```

```
The dataset contains 2386 rows, and 6 columns.
```

In [4]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2386 entries, 0 to 2385
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   ResponseID       2386 non-null   int64 
 1   BranchNum        2386 non-null   int64 
 2   Date_of_Survey   2386 non-null   object
 3   Time_of_Survey   2386 non-null   object
 4   SurveyQuestion1  2386 non-null   int64 
 5   SurveyQuestion2  2386 non-null   int64 
dtypes: int64(4), object(2)
memory usage: 112.0+ KB
```

Can I determine a high level distribution of the values in the dataset?

From the commands below, we see that we have collected responses that are from each of the five possible responses. We also see that most of the customers respond with a five (5) for each survey. Finally, we see that the cardinality of the survey results is five (5) which suggests that the data does not have high cardinality.

In [5]:
```
df1.SurveyQuestion1.value_counts()
```

Out[5]:
```
5    1528
4     586
3     177
2      52
1      43
Name: SurveyQuestion1, dtype: int64
```

In [6]:
```
df1.SurveyQuestion2.value_counts()
```

Out[6]:
```
5    1388
4     764
3     176
1      30
2      28
Name: SurveyQuestion2, dtype: int64
```

In [7]:
```
df1.head()
```

Out[7]:

	ResponseID	BranchNum	Date_of_Survey	Time_of_Survey	SurveyQuestion1	SurveyQuestion2
0	1000	38	3/29/2020 0:00	2:36:00 PM	5	4
1	1001	2	3/29/2020 0:00	6:54:00 PM	5	5
2	1002	9	3/28/2020 0:00	12:21:00 AM	5	5
3	1003	42	3/28/2020 0:00	1:08:00 AM	5	4
4	1004	23	3/27/2020 0:00	11:43:00 AM	5	4

Understanding the survey results relative to the problem statement.

From the problem statement, it is understood that a "Good" response is issue the label of `3`, "Fair" response is issue the label of `2`, and the "Poor" response is issue the label of `1`. Since the `4` label is not part of the evaluation criteria, all customers that responded with *at least one* `4` will be removed from the dataset.

Datetime conversion.

Further, since we want to group the findings by month and year, we will also need to convert the `date_of_survey` column into a time-series object. The commands below convert the format of the `date_of_survey` column to a normalized format `YYYY-MM` that is represented in the `dateyrmo` column.

In [8]:
```
# Convert all column names to lower case.
df1.columns = map(str.lower, df1.columns)
```

In [9]:
```
df1['dateyrmo'] = pd.to_datetime(df1['date_of_survey']).dt.to_period('m')
df1.head()
```

Out[9]:

	responseid	branchnum	date_of_survey	time_of_survey	surveyquestion1	surveyquestion2	dateyrmo
0	1000	38	3/29/2020 0:00	2:36:00 PM	5	4	2020-03
1	1001	2	3/29/2020 0:00	6:54:00 PM	5	5	2020-03
2	1002	9	3/28/2020 0:00	12:21:00 AM	5	5	2020-03
3	1003	42	3/28/2020 0:00	1:08:00 AM	5	4	2020-03
4	1004	23	3/27/2020 0:00	11:43:00 AM	5	4	2020-03

In [10]:
```
#Verify that `dateyrmo` is a date-time object so that we can aggregate by month.
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2386 entries, 0 to 2385
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   responseid       2386 non-null   int64
 1   branchnum        2386 non-null   int64
 2   date_of_survey   2386 non-null   object
 3   time_of_survey   2386 non-null   object
 4   surveyquestion1  2386 non-null   int64
 5   surveyquestion2  2386 non-null   int64
 6   dateyrmo         2386 non-null   period[M]
dtypes: int64(4), object(2), period[M](1)
memory usage: 130.6+ KB
```

Filtering out responses with a `4`.

Because we will remove all customers that responded with *at least one* `4` we will create a flag (`ques_1` and `ques_2`) to know if the survey response contains a `4`. If a `1` appears in either of these columns, a `4` was reported by the customer. Once we have these flags, we will only return the rows where `ques_1 = 0 AND ques_2 = 0` so that we only get the rows that do not have a `4` as a survey response.

In [11]:
```
ques_1 = ''
ques_2 = ''

for index, row in df1.iterrows():
    if (row['surveyquestion1'] == 4):
        df1.at[index, 'ques_1'] = 1
    else:
        df1.at[index, 'ques_1'] = 0

    if (row['surveyquestion2'] == 4):
        df1.at[index, 'ques_2'] = 1
    else:
        df1.at[index, 'ques_2'] = 0

df1['ques_1'] = df1['ques_1'].astype(int)
df1['ques_2'] = df1['ques_2'].astype(int)

# Create a filter (bool mask) dataframe that only contains the normalized date
# and the responses to questions 1 and 2.
df1_filter = df1[(df1['ques_1'] == 0) & (df1['ques_2'] == 0)].reset_index(False)\
                [['dateyrmo', 'surveyquestion1', 'surveyquestion2']]

# Return the head from the resulting dataframe.
df1_filter.head()
```

Out[11]:

	dateyrmo	surveyquestion1	surveyquestion2
0	2020-03	5	5
1	2020-03	5	5
2	2020-03	5	5
3	2020-03	5	5
4	2020-03	5	5

In [12]:
```
print('The number of remaining rows that don\'t contain a `4` is: {}'.format(df1_filter.shape[0])
)
```

```
The number of remaining rows that don't contain a `4` is: 1498
```

Filtering out responses with a `4`.

From the code below, we have verified that the survey responses to questions 1 and 2 do not contain the value of `4`.

In [13]:
```
# Check the cardinality of `surveyquestion1` to ensure that all `4` have been removed.
set(df1_filter.surveyquestion1.to_list())
```

Out[13]:
```
{1, 2, 3, 5}
```

In [14]:
```
#Check distribution of responses to ensure that all `4` have been removed.
df1_filter.surveyquestion1.value_counts()
```

Out[14]:
```
5    1319
3     100
1      41
2      38
Name: surveyquestion1, dtype: int64
```

In [15]:
```
# Check the cardinality of `surveyquestion2` to ensure that all `4` have been removed.
set(df1_filter.surveyquestion2.to_list())
```

Out[15]:
```
{1, 2, 3, 5}
```

In [16]:
```
#Check distribution of responses to ensure that all `4` have been removed.
df1_filter.surveyquestion2.value_counts()
```

Out[16]:
```
5    1312
3     131
1      30
2      25
Name: surveyquestion2, dtype: int64
```

In [17]:
```
#Display the head of the dataframe before going into the aggregation steps.
df1_filter.head()
```

Out[17]:

	dateyrmo	surveyquestion1	surveyquestion2
0	2020-03	5	5
1	2020-03	5	5
2	2020-03	5	5
3	2020-03	5	5
4	2020-03	5	5

Aggregation using a cross-tabulation

The main aggregation here will be a cross-tabulation, which means that for each period, we're going to take a count of how many times we see a five, how many times we see a two, a one, and so forth, and we're going to do that for each month and each question type.

In [18]:
```
#User-defined function to caluclate the percentage from each response in the Crosstab.
def pct(x):
    return 100*(x/x.sum())
```

In [19]:
```
# Crosstab for the filtered resonses for Surveyquestion1
d1 = pd.crosstab(index=df1_filter['dateyrmo'],
                 columns=[df1_filter['surveyquestion1']]).apply(pct, axis=1)

# Crosstab for the filterd resonses for Surveyquestion2
d2 = pd.crosstab(index=df1_filter['dateyrmo'],
                 columns=[df1_filter['surveyquestion2']]).apply(pct, axis=1)

# Concatenate the two dataframes containing the survey responses into one.
dfsummary = pd.concat({d.columns.name: d for d in [d1, d2]}, axis=1)
dfsummary.head()
```

Out[19]:

	surveyquestion1				surveyquestion2			
surveyquestion1	**1**	**2**	**3**	**5**	**1**	**2**	**3**	**5**
dateyrmo								
2019-05	2.500000	5.000000	15.000000	77.500000	2.500000	5.000000	7.500000	85.000000
2019-06	2.608696	1.739130	6.956522	88.695652	1.739130	3.478261	6.956522	87.826087
2019-07	4.255319	1.595745	6.914894	87.234043	2.659574	2.659574	9.574468	85.106383
2019-08	3.517588	2.512563	5.025126	88.944724	2.512563	1.005025	7.537688	88.944724
2019-09	0.649351	3.896104	3.246753	92.207792	1.298701	1.298701	5.844156	91.558442

Calculaing the `net scores` from the aggregated data.

The code below computes the net scores from the aggregated data. We like to point out that per the assignment instructions, the units of the net score are in terms of 100*% since the components that make up the net score are already in terms of a percent.

In [20]:
```
#Calculate the netscore for Survey Question 1 and create a dataframe.
netScore1 = 100*(dfsummary['surveyquestion1'][5] - (dfsummary['surveyquestion1'][3] +\
            dfsummary['surveyquestion1'][2] + dfsummary['surveyquestion1'][1]))

# Misc. formatting and renaming due to multi-index.
dfnetscore1 = pd.DataFrame(netScore1).round(decimals=1)
dfnetscore1['dateyrmo'] = dfnetscore1.index
dfnetscore1.columns.values[0] = 'net_score_question_1'
dfnetscore1.reset_index(drop=True)[['net_score_question_1']]
dfnetscore1 = pd.DataFrame(dfnetscore1['net_score_question_1'])
dfnetscore1.head()
```

Out[20]:

dateyrmo	net_score_question_1
2019-05	5500.0
2019-06	7739.1
2019-07	7446.8
2019-08	7788.9
2019-09	8441.6

In [21]:
```
#Calculate the netscore for Survey Question 2 and create a dataframe.
netScore2 = 100*(dfsummary['surveyquestion2'][5] - (dfsummary['surveyquestion2'][3] +\
            dfsummary['surveyquestion2'][2] + dfsummary['surveyquestion2'][1]))

# Misc. formatting and renaming due to multi-index.
dfnetscore2 = pd.DataFrame(netScore2).round(decimals=1)
dfnetscore2['dateyrmo'] = dfnetscore2.index
dfnetscore2.columns.values[0] = 'net_score_question_2'
dfnetscore2.reset_index(drop=True)[['net_score_question_2']]
dfnetscore2 = pd.DataFrame(dfnetscore2['net_score_question_2'])
dfnetscore2.head()
```

Out[21]:

dateyrmo	net_score_question_2
2019-05	7000.0
2019-06	7565.2
2019-07	7021.3
2019-08	7788.9
2019-09	8311.7

Preparation of Final Tables

In this code block, we're doing some clean up en route to the final table. For all of the fields except the net scores, we've decided to report two decimals.

In this code, we are joining net score values to the cross-tabulation data. In this final table, you can see that all of the values except the net score values are two decimals and are being reported as a percentage. The net score values are to one decimal, but that's a hundred times a percent.

In [22]:
```
# Join both `dfnetscore1` and `dfnetscore2` to create a single DataFrame.
dfnetscore = dfnetscore1.join(dfnetscore2)
dfnetscore.rename_axis(['Year-Month'], inplace=True)

# Rename columns for final delivery so that it's interpretable.
dfnetscore.rename(columns={'net_score_question_1': 'Net Score Question 1',
                           'net_score_question_2': 'Net Score Question 1'}, level=0, inplace=True)
```

In [23]:
```
# Rename columns for final delivery so that it's interpretable.
dfsummary.rename(columns={'surveyquestion1': 'Survey Question 1',
                          'surveyquestion2': 'Survey Question 2'}, level=0, inplace=True)

dfsummary.rename(columns={1: '%Poor',
                          2: '%Fair',
                          3: '%Good',
                          5: '%Excellent'}, level=1, inplace=True)

dfsummary.rename_axis(['Year-Month'], inplace=True)
dfsummary.columns.names = ['Question', 'Responses']

#Round final table to three decimals and store in new variable.
dfsummary_round = dfsummary.round(decimals=2)
dfsummary_round
```

Out[23]:

Question	Survey Question 1				Survey Question 2			
Responses	%Poor	%Fair	%Good	%Excellent	%Poor	%Fair	%Good	%Excellent
Year-Month								
2019-05	2.50	5.00	15.00	77.50	2.50	5.00	7.50	85.00
2019-06	2.61	1.74	6.96	88.70	1.74	3.48	6.96	87.83
2019-07	4.26	1.60	6.91	87.23	2.66	2.66	9.57	85.11
2019-08	3.52	2.51	5.03	88.94	2.51	1.01	7.54	88.94
2019-09	0.65	3.90	3.25	92.21	1.30	1.30	5.84	91.56
2019-10	0.00	0.00	6.78	93.22	0.56	0.00	6.78	92.66
2019-11	3.25	1.63	3.25	91.87	3.25	0.81	4.88	91.06
2019-12	4.00	4.00	8.00	84.00	2.40	0.80	15.20	81.60
2020-01	3.50	2.80	8.39	85.31	2.10	1.40	11.89	84.62
2020-02	2.90	5.07	8.70	83.33	1.45	3.62	12.32	82.61
2020-03	3.12	2.08	8.33	86.46	2.08	1.04	7.29	89.58

Final Table

In this final table, you can see that all of the values except the net score values are two decimals and are being reported as a percentage. The net score values are to one decimal, but that's a hundred times a percent.

In [24]:
```
#Join both tables to produce a final table for delivery.
#The highest net score for both questions is in Oct 2019.
warnings.filterwarnings('ignore')
dfsummary_round.join(dfnetscore)
```

Out[24]:

Year-Month	(Survey Question 1, %Poor)	(Survey Question 1, %Fair)	(Survey Question 1, %Good)	(Survey Question 1, %Excellent)	(Survey Question 2, %Poor)	(Survey Question 2, %Fair)	(Survey Question 2, %Good)	(Survey Question 2, %Excellent)	Net Score Question 1	Net Score Question 1
2019-05	2.50	5.00	15.00	77.50	2.50	5.00	7.50	85.00	5500.0	7000.0
2019-06	2.61	1.74	6.96	88.70	1.74	3.48	6.96	87.83	7739.1	7565.2
2019-07	4.26	1.60	6.91	87.23	2.66	2.66	9.57	85.11	7446.8	7021.3
2019-08	3.52	2.51	5.03	88.94	2.51	1.01	7.54	88.94	7788.9	7788.9
2019-09	0.65	3.90	3.25	92.21	1.30	1.30	5.84	91.56	8441.6	8311.7
2019-10	0.00	0.00	6.78	93.22	0.56	0.00	6.78	92.66	8644.1	8531.1
2019-11	3.25	1.63	3.25	91.87	3.25	0.81	4.88	91.06	8374.0	8211.4
2019-12	4.00	4.00	8.00	84.00	2.40	0.80	15.20	81.60	6800.0	6320.0
2020-01	3.50	2.80	8.39	85.31	2.10	1.40	11.89	84.62	7062.9	6923.1
2020-02	2.90	5.07	8.70	83.33	1.45	3.62	12.32	82.61	6666.7	6521.7
2020-03	3.12	2.08	8.33	86.46	2.08	1.04	7.29	89.58	7291.7	7916.7