MAVEN CLOUD LIMITED

CALL RECORDS DATA ANALYSIS

BY LULE NURU
DATA SCIENTIST

Background

- Maven cloud limited is a company that provides call services and is looking to better its company by gaining meaningful insights using its existing records.
- The company would like to know what are some of the issues affecting its customer experience and therefore predict the future using the available data.
- The company believes that with the availed two-week data and the help of a data scientist, it should be able to give insight on how their company can be improved and hence better services in the long run, increase profits.

- Some of the questions Maven cloud wishes to be answered include;

1. When do we experience the most incoming call traffic? How does this vary over the course of the day? How does this compare to the number of outgoing calls being made over the course of the day?
2. How many customers do we talk to each day? How many customers called maven cloud limited multiple times in the two week period? What were we discussing with the customers who contacted us multiple times?
3. Which customers are more like to churn and why?
4. Which call type are we most likely to expect in the future predictably from the available data we have at hand?
5. Are there areas where we can improve customer experience based on call data?

- Exploratory data analysis was done using visual studio code to investigate the data and answer the questions the company wishes to find solutions to.
- The data analysis process included thoroughly understanding the data set by cleaning it and checking for all the quality checks.
- Different statistical and visualization tools were employed to identify patterns and trends (like matplotlib and seaborn).
- More advanced techniques were used for hypothesis testing and future engineering to create predicting models for example logistic regression and decision tree model.

- From the analysis, an example of the top 5 records are shown below;

```
data_copy.head()
data_copy.tail()
```

0.0s Python

	call_id	destination_person_id	source_person_id	account_age	account_state	occupation	language	call_direction	call_duration	call_outcome	call_end_
32023	2855830	338414	361079	68	In Repayment	Small Business Owner	Lunyankore	Incoming	6.55	Call Resolved	11-01
32024	2855832	157397	302921	183	In Repayment	Farmer	Ma'di	Incoming	12.90	Call Resolved	11-01
32025	2855835	80646	229800	386	In Repayment	Farmer	Luganda	Incoming	1.25	Call Resolved	11-01
32026	2855836	338372	8029	1146	Cancelled	Boda Boda	English	Incoming	2.98	Call Resolved	11-01
32027	2855837	124255	387315	0	In Repayment	Small Business Owner	Luganda	Incoming	3.08	Call Resolved	11-01

The data included the following information on data types of each column and the number of rows (32028) and columns (13)

```
[5]  data_copy.info()
     ✓ 0.5s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32028 entries, 0 to 32027
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   call_id               32028 non-null  int64
 1   destination_person_id 32028 non-null  int64
 2   source_person_id      32028 non-null  int64
 3   account_age           32028 non-null  int64
 4   account_state         32028 non-null  object
 5   occupation            31933 non-null  object
 6   language              32027 non-null  object
 7   call_direction        32028 non-null  object
 8   call_duration         32028 non-null  float64
 9   call_outcome          32028 non-null  object
 10  call_end_date         32028 non-null  object
 11  call_topics           32028 non-null  object
 12  call_topic_group      32028 non-null  object
dtypes: float64(1), int64(4), object(8)
memory usage: 3.2+ MB


[6]  print(f'There are {data_copy.shape[0]} rows and {data_copy.shape[1]} columns in the dataset')
     ✓ 0.0s

There are 32028 rows and 13 columns in the dataset
```

Further went to explore the data by checking for duplicates and missing values and as seen in the picture below, there were no duplicates and there were only 96 missing values

```
[7]  print(f'There are a total of {data_copy.duplicated().sum()} duplicate records')
     ✓ 0.4s

...  There are a total of 0 duplicate records

[8]  data_copy.isnull().sum()
     ✓ 0.0s

...  call_id                 0
     destination_person_id   0
     source_person_id        0
     account_age             0
     account_state           0
     occupation             95
     language                1
     call_direction          0
     call_duration           0
     call_outcome            0
     call_end_date           0
     call_topics             0
     call_topic_group        0
     dtype: int64
```

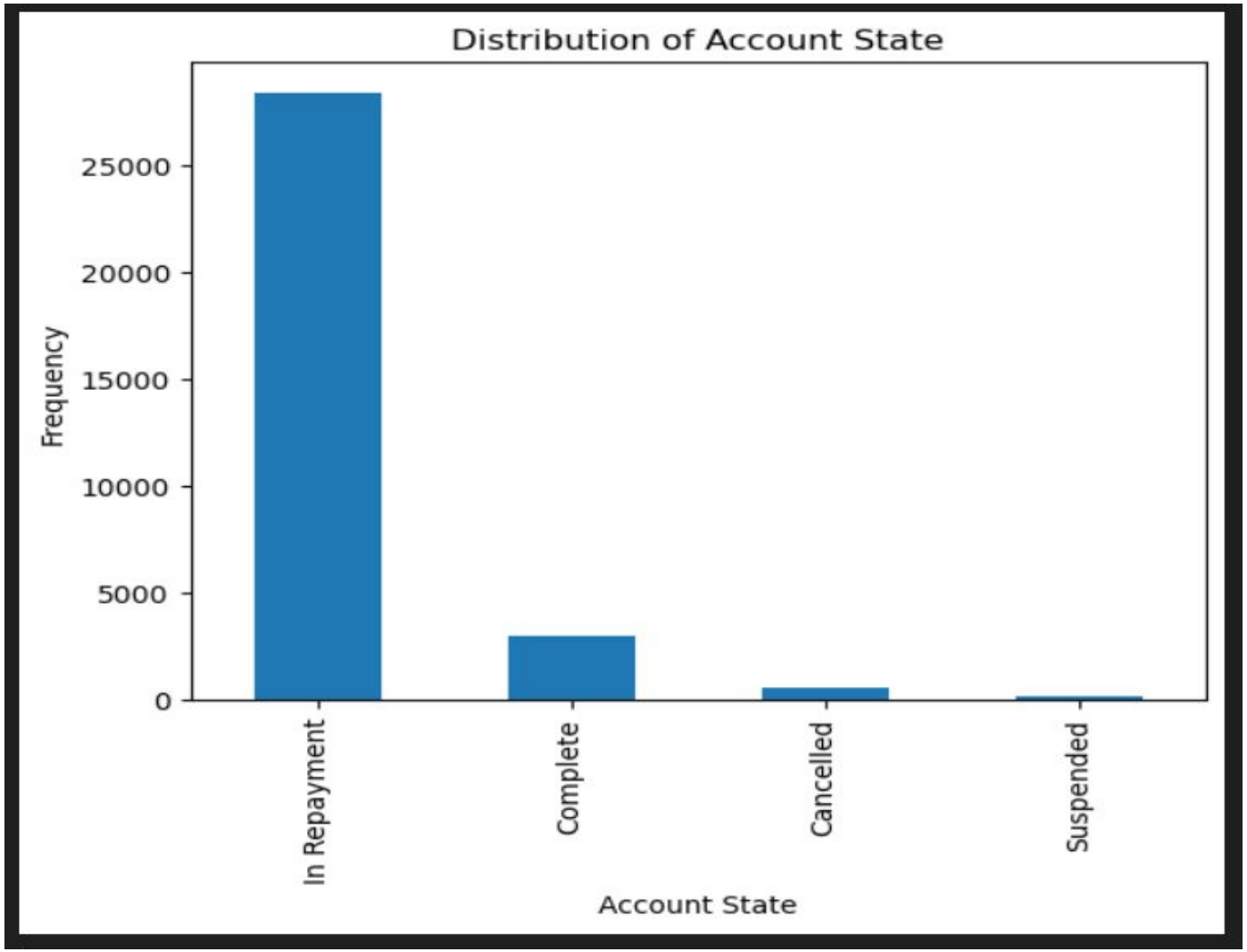
The statistical description of the numerical columns in the dataset is as shown below;

```
[12]  data_copy.describe()
      ✓ 0.2s
```

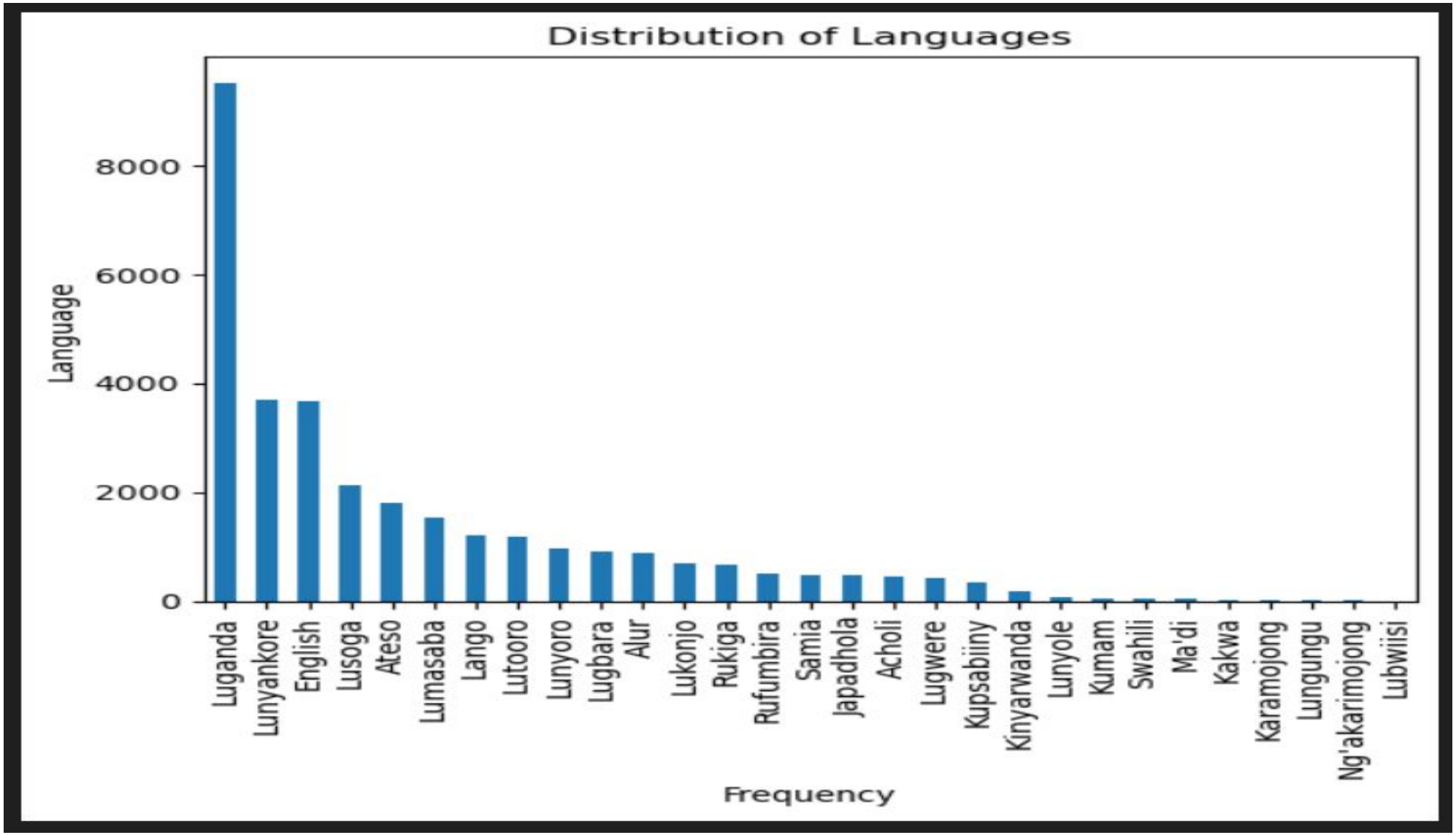
	call_id	destination_person_id	source_person_id	account_age	call_duration
count	3.202800e+04	32028.000000	32028.000000	32028.000000	3.202800e+04
mean	2.831672e+06	176398.621956	238985.845354	342.367085	-1.565586e+03
std	1.374674e+04	97753.450380	106046.655869	306.645479	1.987238e+05
min	2.807990e+06	73.000000	89.000000	-10.000000	-2.514926e+07
25%	2.819686e+06	108351.000000	155159.000000	64.000000	1.550000e+00
50%	2.831604e+06	134577.000000	239082.000000	298.000000	3.580000e+00
75%	2.843481e+06	239005.000000	338348.250000	573.000000	6.200000e+00
max	2.855837e+06	387319.000000	387407.000000	1430.000000	5.787000e+01

Visual representation of the categorical columns in the data set

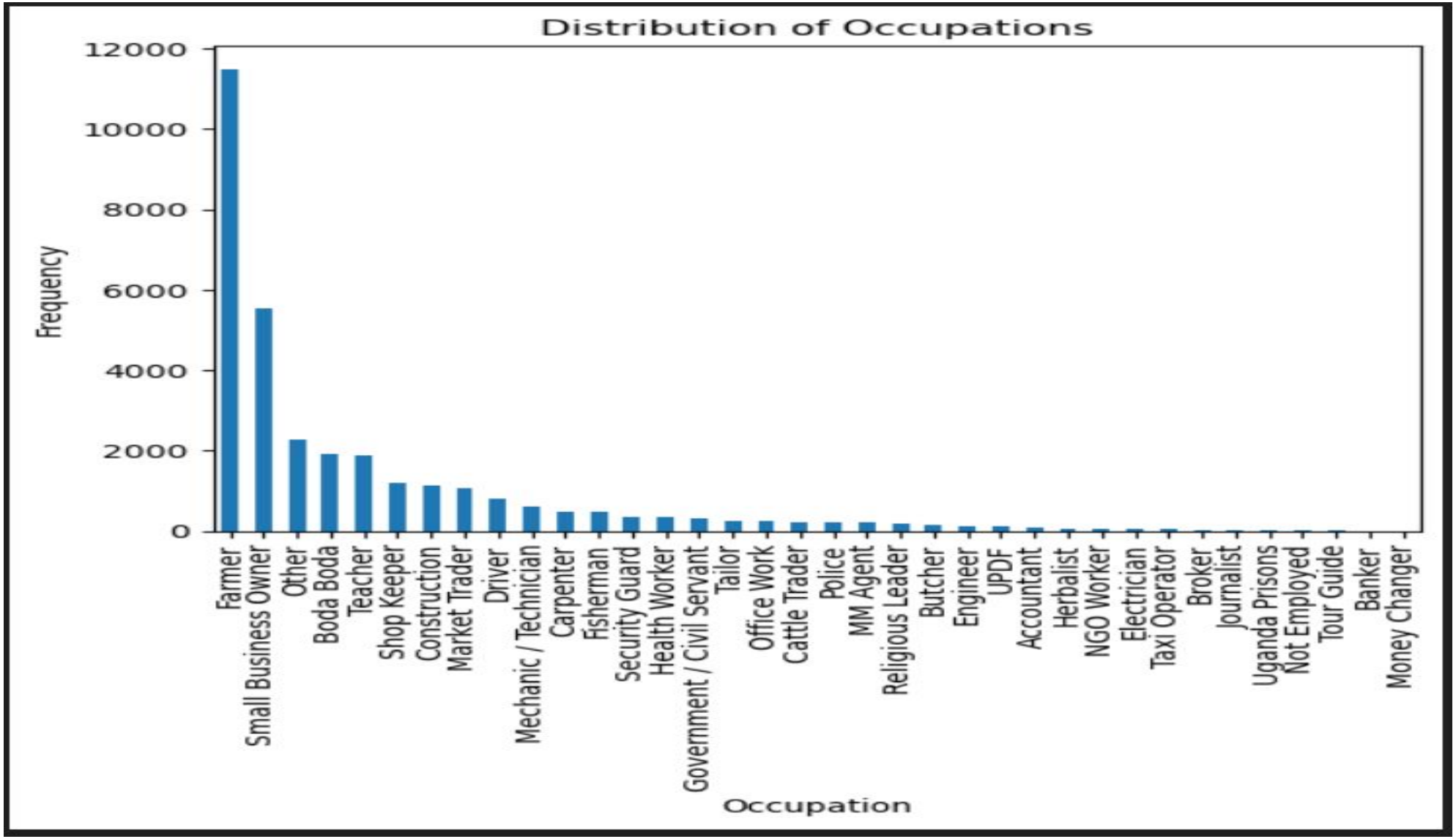
- A bar graph showing the distribution of account_state



A bar graph showing the distribution of languages



A bar graph showing the distribution of occupation



1. When do we experience the most incoming call traffic? How does this vary over the course of the day? How does this compare to the number of outgoing calls being made over the course of the day?

- Maven cloud limited experienced the most incoming call traffic at 16:00 hours with 2194 calls.
- This was obtained by focusing on two columns that is the call_direction and call_end_date columns.
- The call_direction has both incoming and outgoing calls whereas the call_end_direction has the date and hour of the calls.
- As seen from the data information above, I realized that the call_end_date had a data type of ‘object’ which had to be changed to the ‘datetime’ format as shown below.

Changing the call_end_date column datatype from ‘object’ to ‘datetime’ as well as checking and changing the consistency of how the dates are written. (ensuring all dates are written with a ‘-’ and not ‘/’

```
data_copy['call_end_date'].unique()
```

✓ 0.0s

```
array(['10/21/17 4:01', '10/21/17 4:04', '10/21/17 4:05', ...,
       '11-03-17 15:09', '11-03-17 15:10', '11-03-17 15:11'], dtype=object)
```

```
# Replace '/' with '-' for consistency
data_copy['call_end_date'] = data_copy['call_end_date'].str.replace('-', '/')

# Define the parsing function
def parse_dates(date_str):
    for fmt in ('%m/%d/%y %I:%M', '%m/%d/%y %H:%M'):
        try:
            return pd.to_datetime(date_str, format=fmt)
        except ValueError:
            continue
    return pd.NaT  # Return NaT if no format matches

# Apply the parsing function
data_copy['call_end_date'] = data_copy['call_end_date'].apply(parse_dates)

# Check and handle NaT values if needed
print(data_copy['call_end_date'].isna().sum())
data_copy = data_copy.dropna(subset=['call_end_date'])
```

✓ 10.0s

```
0
```

```
data_copy['call_end_date'] = pd.to_datetime(data_copy['call_end_date'], format='%m/%d/%y %H:%M')
```

✓ 0.1s

From the call_direction column, two variables were generated that is ‘is_incoming’ and ‘is_outgoing’and there respective columns added at the end of the dataset as shown below.

```
data_copy['is_incoming'] = data_copy['call_direction'] == 'Incoming'
data_copy['is_outgoing'] = data_copy['call_direction'] == 'Outgoing'
```

7] ✓ 0.0s Python

```
data_copy.head()
```

9] ✓ 0.0s Python

account_age	account_state	occupation	language	call_direction	call_duration	call_outcome	call_end_date	call_topics	call_topic_group	is_incoming	is_outgoing
301	In Repayment	Farmer	Luganda	Incoming	0.78	Call Resolved	2017-10-21 04:01:00	Call Dropped	Other	True	False
122	In Repayment	Other	Luganda	Incoming	1.93	Call Resolved	2017-10-21 04:04:00	Code Not Received	Codes	True	False
695	In Repayment	Mechanic / Technician	Luganda	Incoming	4.90	Call Resolved	2017-10-21 04:05:00	RP Remote Technical	Accessory Technical Issue	True	False
22	In Repayment	Farmer	Lango	Incoming	3.77	Call Resolved	2017-10-21 04:06:00	Code Not Received	Codes	True	False
44	In Repayment	Electrician	Luganda	Incoming	2.62	Call Resolved	2017-10-21 04:07:00	Code Not Received	Codes	True	False

Went ahead to calculate the valuecounts of both is_incoming and is_outgoing calls to see the total number of calls made and its visualisation

```
print(data_copy['is_incoming'].value_counts())
print(data_copy['is_outgoing'].value_counts())
```

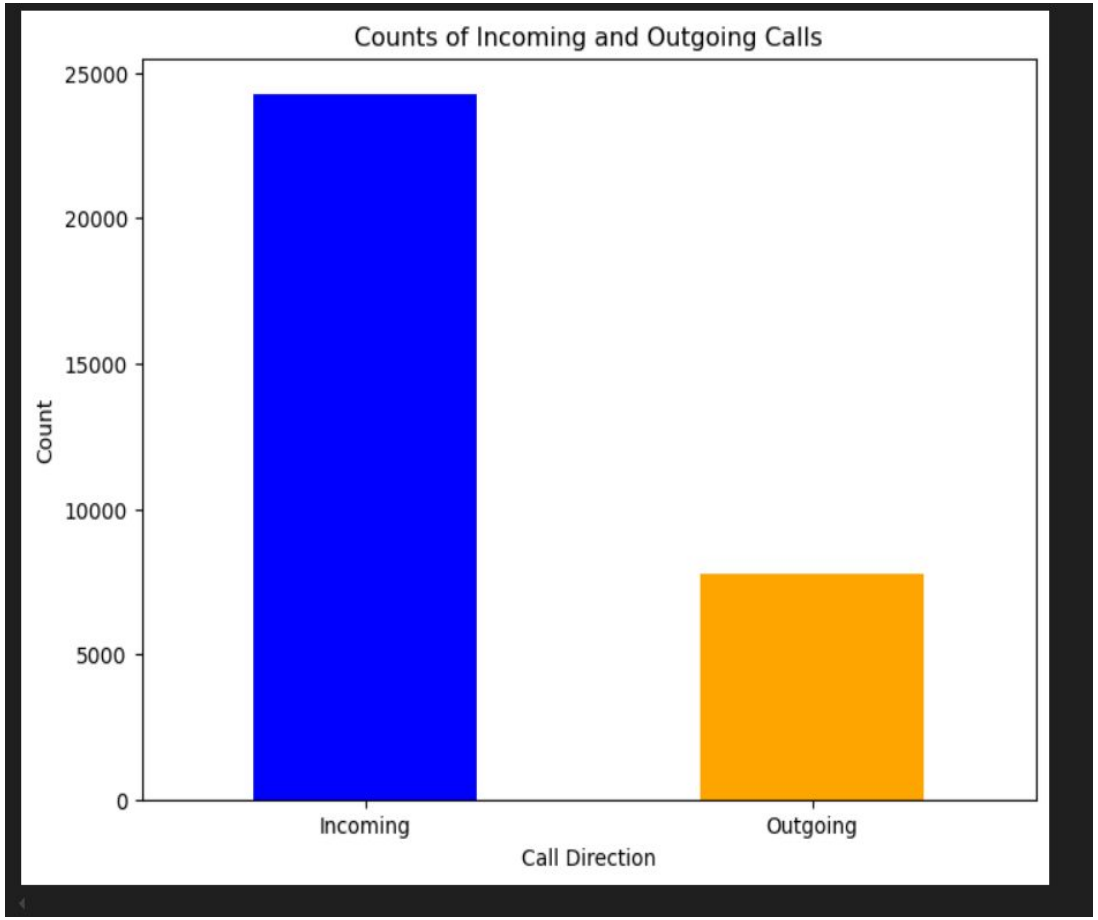
[23] ✓ 0.0s

```
is_incoming
True     24286
False     7742
Name: count, dtype: int64
is_outgoing
False    24286
True      7742
Name: count, dtype: int64
```

```
counts = data_copy[['is_incoming', 'is_outgoing']].sum()

plt.figure(figsize=(8, 6))
counts.plot(kind='bar', color=['blue', 'orange'])
plt.title('Counts of Incoming and Outgoing Calls')
plt.xlabel('Call Direction')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Incoming', 'Outgoing'], rotation=0)
plt.show()
```

[24] ✓ 2.7s



Therefore to answer the question of when does the company experience the most incoming call traffic, it was calculated as follows

```
[25]  data_copy['hour'] = data_copy['call_end_date'].dt.hour
      ✓ 0.0s

[26]  incoming_counts = data_copy[data_copy['is_incoming']].groupby('hour').size()
      ✓ 0.4s

[27]  peak_hour_incoming = incoming_counts.idxmax()
      peak_count_incoming = incoming_counts.max()
      ✓ 0.0s

[28]  print(f"Peak incoming call traffic is at {peak_hour_incoming}:00 with {peak_count_incoming} calls.")
      ✓ 0.0s

...   Peak incoming call traffic is at 16:00 with 2194 calls.
```

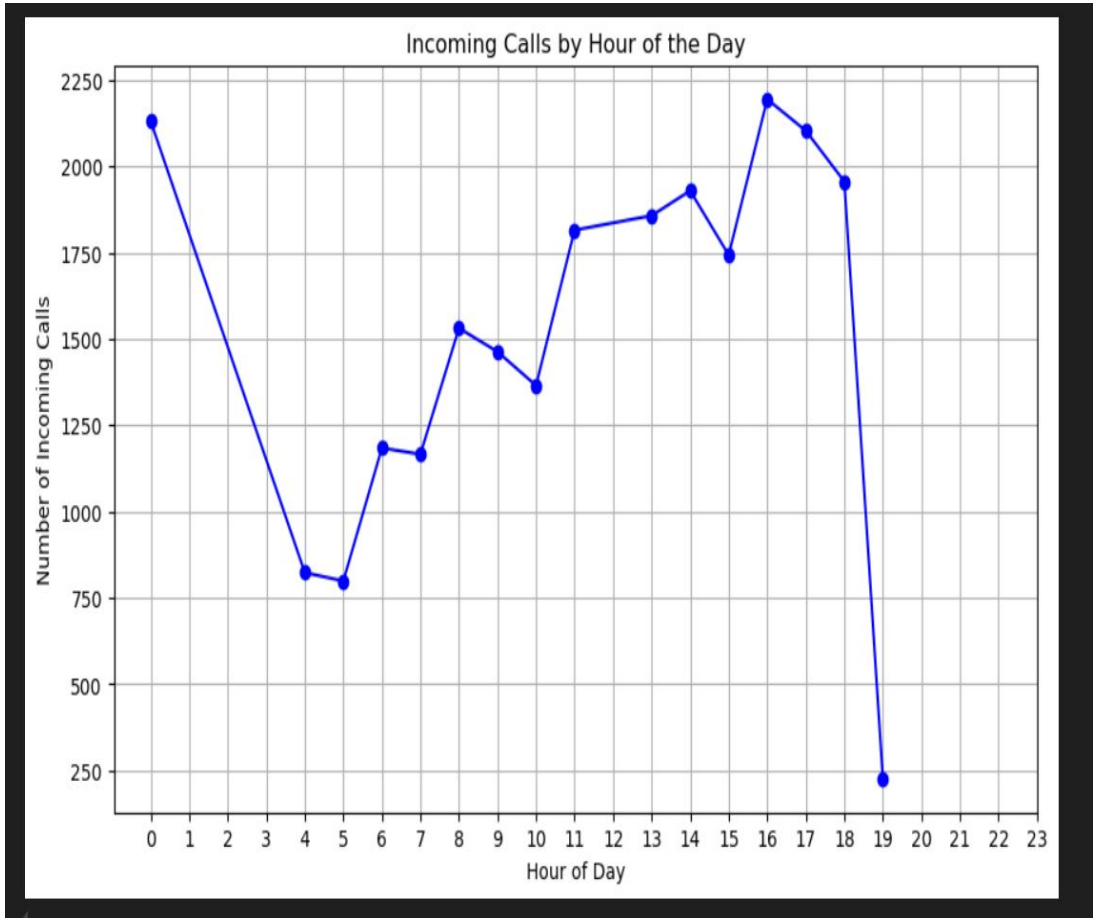
The summation of incoming call traffic variation by hour over the course of the day is as shown below along side its visualization

```
[29]  incoming_df = data_copy[data_copy['is_incoming']]
      ✓ 0.0s

[30]  incoming_counts_by_hour = incoming_df.groupby('hour').size()
      ✓ 0.0s

[31]  print(incoming_counts_by_hour)
      ✓ 0.0s

...   hour
      0     2131
      4      823
      5      799
      6     1184
      7     1166
      8     1531
      9     1463
      10    1365
      11    1815
      13    1857
      14    1929
      15    1743
      16    2194
      17    2103
      18    1956
      19     227
      dtype: int64
```



The total number of outgoing call traffic by hour is summarized below in the first diagram whereas the comparison between incoming and outgoing calls is shown in the second diagram

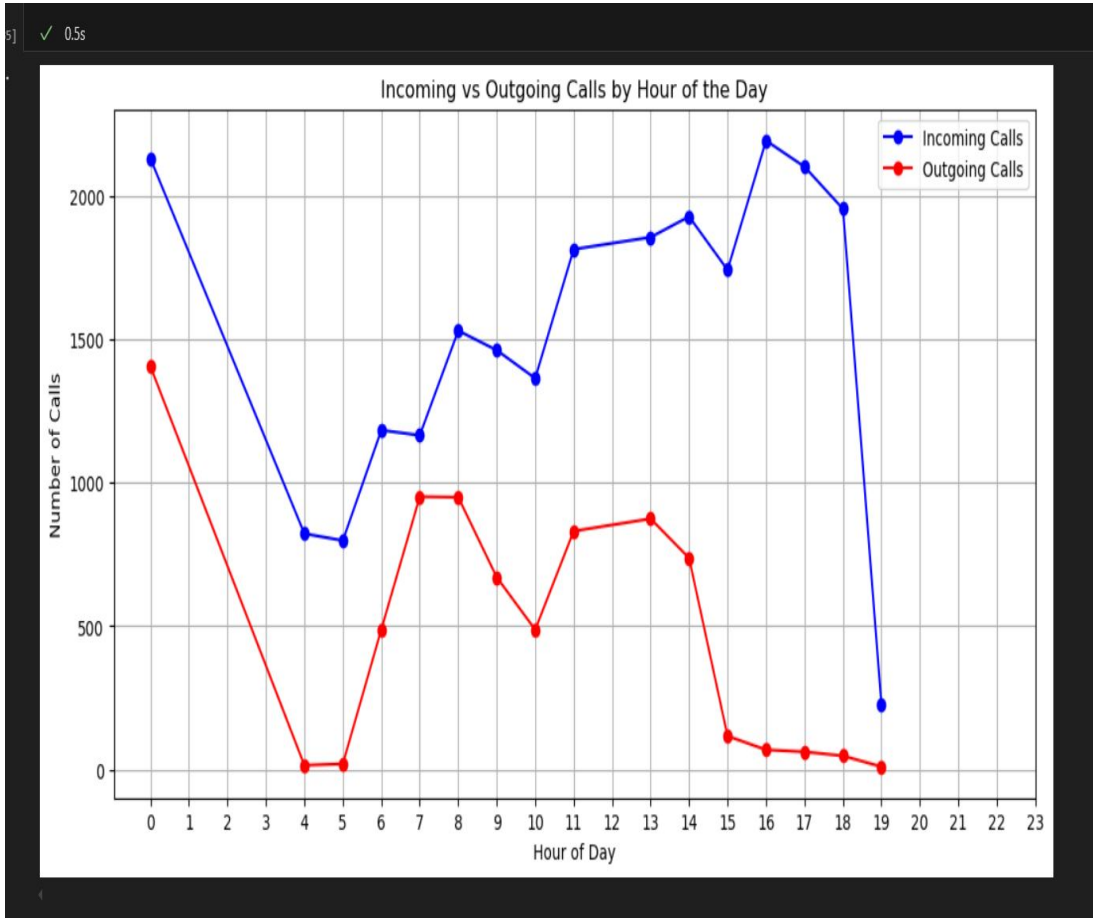
```
# Filter for outgoing calls
outgoing_df = data_copy[data_copy['is_outgoing']]

# Count outgoing calls by hour
outgoing_counts_by_hour = outgoing_df.groupby('hour').size()

# Print the counts for reference
print(outgoing_counts_by_hour)
```

34] ✓ 0.0s

```
hour
0     1406
4       15
5       20
6      489
7      952
8      950
9      671
10     488
11     832
13     875
14     738
15     117
16      69
17      62
18      48
19      10
dtype: int64
```



- From the above analysis and visualization, we noticed the most incoming call traffic is at 16:00 hours
- At midnight, many calls are made but they drastically drop till 4am and this is probably because most people are sleeping at this time or no data was recorded.
- From 4am, we notice calls further drop till 5am
- From 5am to 4pm, calls steadily increase till they reach the peak as these are working day hours where people call for different inquiries and then drop again drastically at 7pm.

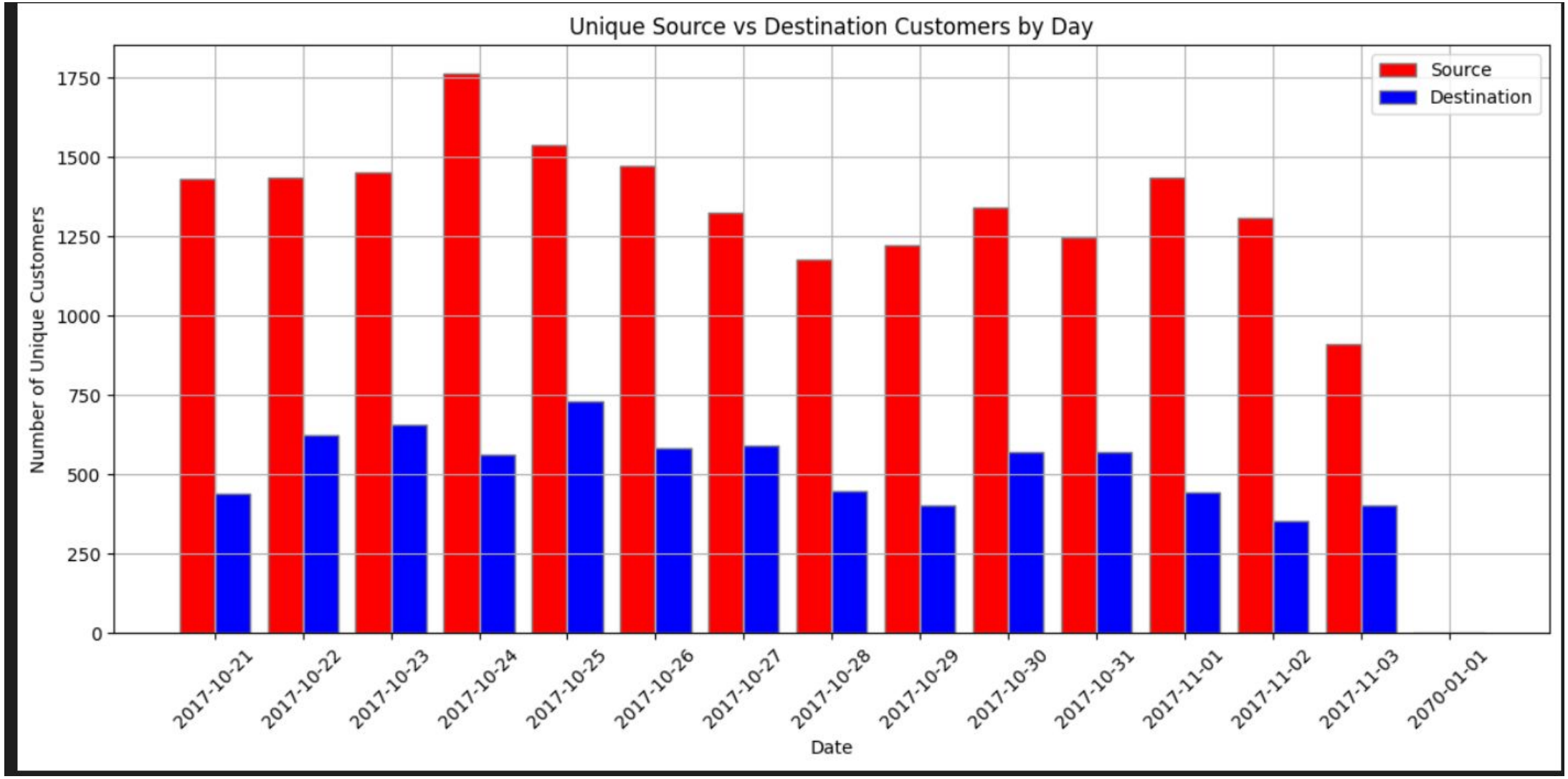
2. How many customers do we talk to each day? How many customers called maven cloud limited multiple times in the two week period? What were we discussing with the customers who contacted us multiple times?

- Maven cloud limited talks to more than 1000 clients each day.
- This was obtained by using the destination_person_id and source_person_id which represent the outgoing and incoming calls respectively thereby getting the unique customer number that we talked to on a daily person.
- We combined the unique customers from destination_person_id with the unique customers from source_person_id to come up with the totals as shown below;

Number of customers Maven clouds limited talks to per day

```
Total unique customers per day:
call_date
2017-10-21    1797
2017-10-22    1956
2017-10-23    1974
2017-10-24    2208
2017-10-25    2137
2017-10-26    1952
2017-10-27    1813
2017-10-28    1545
2017-10-29    1543
2017-10-30    1795
2017-10-31    1700
2017-11-01    1779
2017-11-02    1569
2017-11-03    1240
2070-01-01       4
Name: combined_ids, dtype: int64
```

A bar graph showing the incoming calls(source_id) and outgoing calls (destination_id) per day



- The number of clients who called maven cloud limited more than once were 7422 which was obtained by concating all the unique customers from source_id and destination_id that were greater than one. This is shown below;

```
# Extract customers that appear more than once
customers_more_than_once = customer_contact_counts[customer_contact_counts > 1]

# Print the result
print(customers_more_than_once)
```

✓ 0.0s

```
212968    758
124292    700
212954    689
108357    677
86394     639
          ...
101428      2
381389      2
180021      2
143709      2
334261      2
Name: count, Length: 7422, dtype: int64
```

```
# Get the number of customers that appear more than once
num_customers_more_than_once = customers_more_than_once.count()

# Print the result
print(f"Number of customers who called more than once: {num_customers_more_than_once}")
```

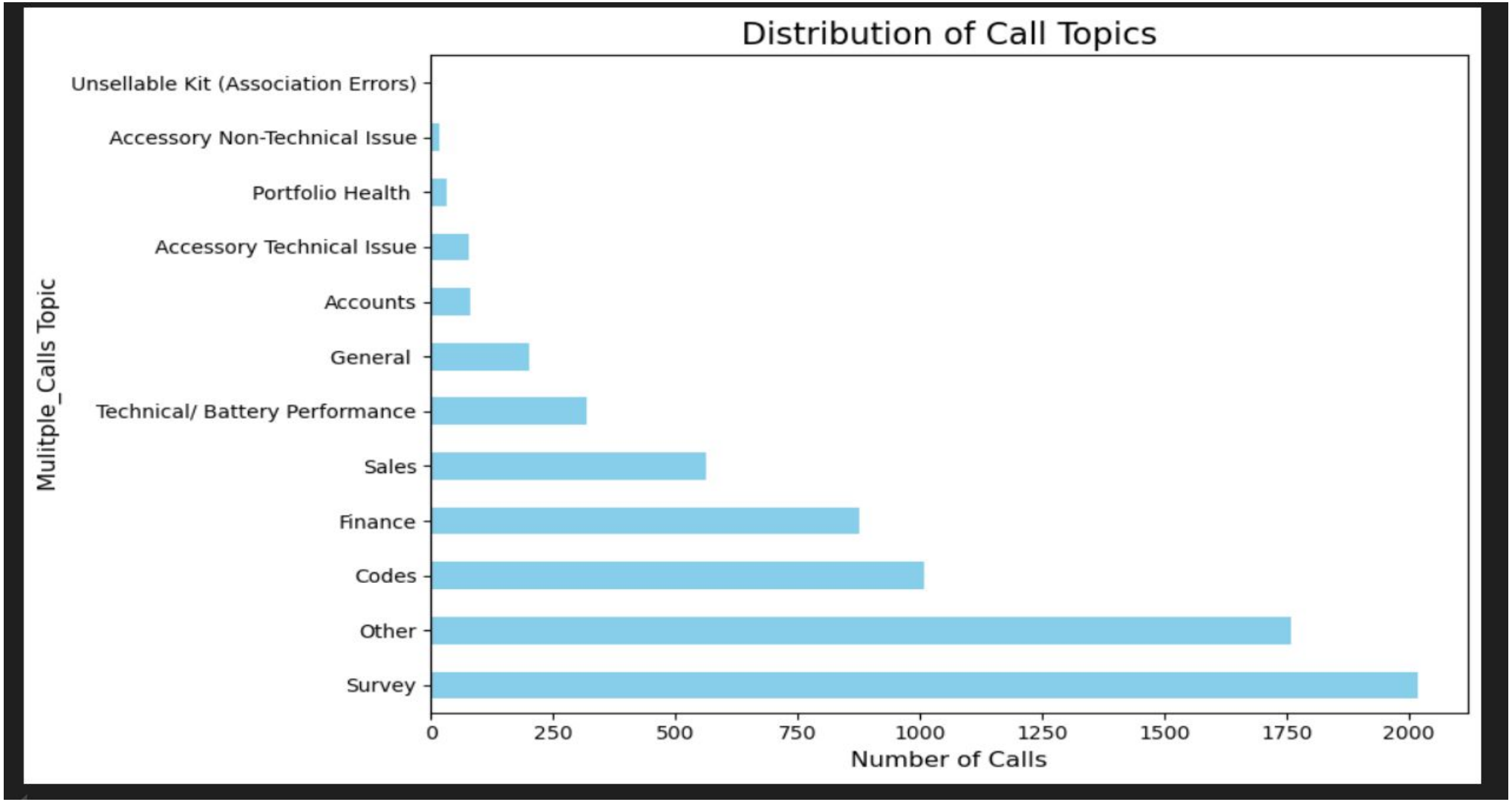
✓ 0.0s

```
Number of customers who called more than once: 7422
```

- Further analysis was done showing groups of people who called multiple times that is greater than 1, 2,3,5 and 10.

	count
>1	13473
>2	1677
>3	1266
>5	473
>10	160

What we discussed with the customers who contacted us multiple times is visually represented by a bar graph below;



- From the above visualization, we notice that the customers who called more than once in the two week window period were mostly discussing about a survey which has the highest number of call counts.
- This was followed by the ‘other’ category then codes. These were the top three topics
- The least topic discussed was the unsellable kit (association errors)

3. Which customers are more likely to churn and why?

- The number of customers expected to stop using maven cloud limited were analyzed using the call_outcome column which included a number of different outcomes as shown below;

```
Summary of call outcomes:
call_outcome
Call Resolved                                  27214
Customer Unreachable                            2417
Call Unresolved                                  786
Call Unresolved                                  729
Needs Repair / Replacement                       584
Vetting Incomplete                               128
Customer hung up on CSR / Call dropped            90
Voluntary Return Unit                             46
De-activation                                     13
Escalate                                          11
Unresolved call requires further action by CSR     9
Mistake Call                                       1
```

- With the above call_outcome summary, we observe that the customers most likely to churn are those with unsatisfied call outcomes which include;

1. Unresolved calls
2. Customer hang up on CSR / call dropped
3. Unresolved call requires further action by CSR
4. Deactivated calls
5. Vetting incomplete

- Some of the reasons clients are most likely to churn are;

1. Unresolved calls of their inquiries
2. Long and tedious calls
3. Transfer of their calls and no resolution
4. Dropped calls
5. No follow up calls on inquiries
6. Long vetting system

#4. Which call type (call_direction) are we most likely to expect in the future predictability from the available data we have at hand?

- The call type we are most likely to expect in the future was predicted by the use of three different model comparisons i.e. linear regression model, decision tree model and support vector model.
- All the models gave an accuracy nearly in the same range however, the support vector model gave us the most accurate prediction using the available data.
- The dependent variable (X) was the call_direction which consists of both the incoming and outgoing calls whereas the independent variables (y) were all the columns except the call_direction.
- In all the three models a 80-20 train-test function was used.

Linear regression model

- This model assumes a linear relationship between a dependent variable and one or more independent variables.
- From the available data, our linear regression model gave the report below;

Linear Regression Model Classification Report:

	precision	recall	f1-score	support
Incoming	0.85	0.99	0.91	4878
Outgoing	0.94	0.43	0.59	1528
accuracy			0.86	6406
macro avg	0.89	0.71	0.75	6406
weighted avg	0.87	0.86	0.84	6406

- From the above report, we notice the following;
- Precision: 85% of call_direction predicted as incoming were correct whereas 94% of call_direction predicted as outgoing were actually outgoing.
- Recall: the model correctly identified 99% of all actual as incoming while 43% of all were correctly identified as outgoing.
- The f1-score of incoming(91%) was higher than that of outgoing (59%) indicating better overall performance.
- Support shows the dataset has 4878 incoming calls and 1528 outgoing calls and the model has an overall accuracy of 86%.

Decision Tree Model

- This model predicts the value of a target variable by learning simple decision rules inferred from the data features.
- From the available data, our decion tree model gave the report below;

Decision Tree Classification Report:

	precision	recall	f1-score	support
Incoming	0.89	0.88	0.88	4878
Outgoing	0.63	0.64	0.64	1528
accuracy			0.82	6406
macro avg	0.76	0.76	0.76	6406
weighted avg	0.83	0.82	0.82	6406

- From the above report, we notice the following;
- Precision: 89% of call_direction predicted as incoming were correct whereas 63% of call_direction predicted as outgoing were actually outgoing.
- Recall: the model correctly identified 88% of all actual as incoming while 64% of all were correctly identified as outgoing.
- The f1-score of incoming(88%) was higher than that of outgoing (64%) indicating better overall performance.
- Support shows the dataset has 4878 incoming calls and 1528 outgoing calls and the model has an overall accuracy of 82%.

Support Vector Model

- This model is good for its ability to handle complex, non linear decision boundaries.
- From the available data, our support vector model gave the report below;

SVM Classification Report:

	precision	recall	f1-score	support
Incoming	0.85	1.00	0.92	4878
Outgoing	1.00	0.42	0.59	1528
accuracy			0.86	6406
macro avg	0.92	0.71	0.75	6406
weighted avg	0.88	0.86	0.84	6406

- From the above report, we notice the following;
- Precision: 85% of call_direction predicted as incoming were correct whereas 100% of call_direction predicted as outgoing were actually outgoing.
- Recall: the model correctly identified 100% of all actual as incoming while 42% of all were correctly identified as outgoing.
- The f1-score of incoming(92%) was higher than that of outgoing (59%) indicating better overall performance.
- Support shows the dataset has 4878 incoming calls and 1528 outgoing calls and the model has an overall accuracy of 86%.

- From the above analysis, the comparison between the three models i.e. linear regression, decision tree and support vector model is 86%, 82% and 86% respectively.
- In conclusion, all the three models accurately predicted the future call type but the support vector model was the most accurate at 86% with an f1-score of 92% for incoming calls and 59% for outgoing calls and so it is the most recommended model of the three.

#5 Are there areas where we can improve customer experience based on call data?

- We can improve customer experience by reducing the average call duration

Average Call Duration by Call Outcome:

	call_outcome	average_call_duration
0	Call Resolved	-1843.028286
1	Call Unresolved	2.316189
2	Call Unresolved	2.327476
3	Customer Unreachable	1.368146
4	Customer hung up on CSR / Call dropped	2.134685
5	De-activation	4.422308
6	Escalate	7.502273
7	Mistake Call	0.320000
8	Needs Repair / Replacement	8.306626
9	Unresolved call requires further action by CSR	9.887778
10	Vetting Incomplete	9.640773
11	Voluntary Return Unit	5.564130

- As we had seen earlier, some of the reasons why our customers were churning were because of unresolved call, dropped calls, escalated calls, deactivated calls, unresolved calls which require further action by CSR and incomplete vetting. These reasons are still seen to have the highest average call duration with highest upto 9.8.
- If consideration of these call outcomes is given more priority it would reduce the rate at which our customers churn and in the long run improve the overall customer experience.
- Also, considering the transfer calls in relation to its average, client occupation, language account age and call direction, these can give us insights on which clients need more help and get customer care teams that suit their needs. This is illustrated below;

```
Total calls: 1157

Average account age for transfered calls: 249.04days

Most common languages in transfered calls: n\language
Rufumbira    150
Luganda      140
Alur         135
Lugbara      105
Samia         93
Name: count, dtype: int64

Most common occupations in transfered calls: n\occupation
Farmer                  495
Small Business Owner    201
Other                    90
Boda Boda                68
Shop Keeper              42
Name: count, dtype: int64

Average duration of transfered calls: 1.29minutes

Call directions for transfered calls:
call_direction
Incoming    1155
Outgoing       2
```