S L A   f o r   S a l e s   -   M o d u l e

Aging and interaction Function

Contents

Abstract
Objective of the Function
Introduction
Problem Statement
Data Overview
Installation
Codes
Implementation
Conclusion
Package Reference

Abstract

In the dynamic world of sales, understanding lead management and performance assessment is critical. This abstract presents a function designed for the sales industry, aimed at evaluating lead aging and interaction frequency to determine lead importance in a particular point in time.

The function calculates the aging of leads by measuring the number of days since their generation, providing historical context to assess their relevance. Additionally, it quantifies the number of times each lead has been contacted by the sales team, serving as a valuable indicator of lead engagement.

These two metrics, lead aging and interaction frequency, collectively contribute to the determination of lead importance.

Objective of the Function

The primary goal of this function is to provide information about leads, specifically in two aspects: aging and the number of times they have been contacted. These two pieces of information are essential for assessing the significance or importance of each lead once they are into the pipeline(system).

Aging of Leads:

- The function calculates the "aging" of a lead. In this context, aging refers to how many days have passed since the lead was generated.
- This aging metric helps evaluate how old a lead has become, indicating its historical context, older leads may require different strategies or attention compared to newer ones.

Number of Interactions:

- The function also determines the number of times a lead has been contacted by the sales team. This is a measure of how frequently the sales team has engaged with the lead.
- This metric helps assess the degree of interest or engagement with the lead. A lead that has been contacted multiple times and not being converted to sales can be considered less important and should have less priority to improve the performance of the floor.

Importance of Leads:

- This mentions that both aging and the number of interactions are used to gauge the "degree of importance" of each lead.
- Leads that are older and those which have been contacted more may be considered less important and may receive different levels of attention or prioritization in sales or marketing efforts.

In summary, this function is designed to provide valuable insights into leads by calculating their age(time since generation) and interactions(the number of times they've been contacted). These metrics help in evaluating and prioritizing leads based on their historical context and engagement level, which is crucial in effective lead management and decision-making within sales and marketing activities.

Introduction

In the context of sales or lead management, determining the degree of importance of leads is essential. Leads can vary in terms of their potential value to a business. Some leads may represent high-value opportunities, while others may have lower potential. By assessing the degree of importance, businesses can prioritize their efforts, allocate resources wisely, and focus on leads that are most likely to result in successful conversions or sales.

Factors that contribute to the degree of importance of leads may include their level of engagement, readiness to purchase, potential revenue they can bring, and their fit with the company's target audience or ideal customer profile. The more important a lead is perceived to be, the more attention and effort a business may allocate to nurturing and converting that lead into a customer.

Lead time aging was also analyzed to understand the optimum point of inflection from where possibility of sales conversions dips maximum. Hence, this metrics was used for prioritizing leads for sales process progression (following up, discussion, sharing premium computations and benefits etc.)

The optimum number of interactions for sales conversion was identified and the impact of improving sales performance of sales persons (good and bad performers) quantified. Based on this, incentive plan and training plan were formulated for the sales organization.

<u>Here are some of the key benefits of identifying aging and no.of interactions with lead:</u>

Prioritizing Leads: By knowing how long a lead has been in the pipeline (lead aging), businesses can prioritize their efforts. Older leads may require different follow-up strategies or a more personalized approach, helping sales teams allocate their time and resources effectively.

Customized Engagement: Understanding lead aging allows businesses to tailor their communication and engagement strategies. For example, newer leads may need more nurturing and education, while older leads may be closer to making a purchase decision.

Conversion Rate Improvement: Tracking the number of interactions with leads helps businesses gauge lead engagement and interest. Leads that have had multiple interactions are often less likely to convert into customers, making it possible to focus efforts on newer leads with higher potential.

Resource Allocation: Efficiently allocating sales and marketing resources is critical for cost-effectiveness. Data on lead aging and interactions allows businesses to allocate resources where they are needed most, reducing wastage and increasing ROI.

Lead Scoring Improvement: Lead scoring models can be enhanced with data on lead aging and interactions. This enables more accurate lead prioritization and scoring, ensuring that the most promising leads receive the highest attention.

Data-Driven Decision-Making: Data on lead aging and interactions provide valuable insights for strategic decision-making. Businesses can make data-driven decisions about lead management, sales strategies, and marketing campaigns.

Develop more effective sales training programs: Introducing new training methodologies based on aging and interactions can make the salesperson understands on which lead more time and labour needs to get invested. Customizing training programs for sales and customer support teams based on lead interaction data. Teams can receive specialized training on how to engage with leads at different stages of the customer journey.

Early Engagement Incentives: Offering incentives to salesperson for early-interaction with the leads to encourage them. This can create a sense of urgency and reward proactive engagement.

Problem Statement

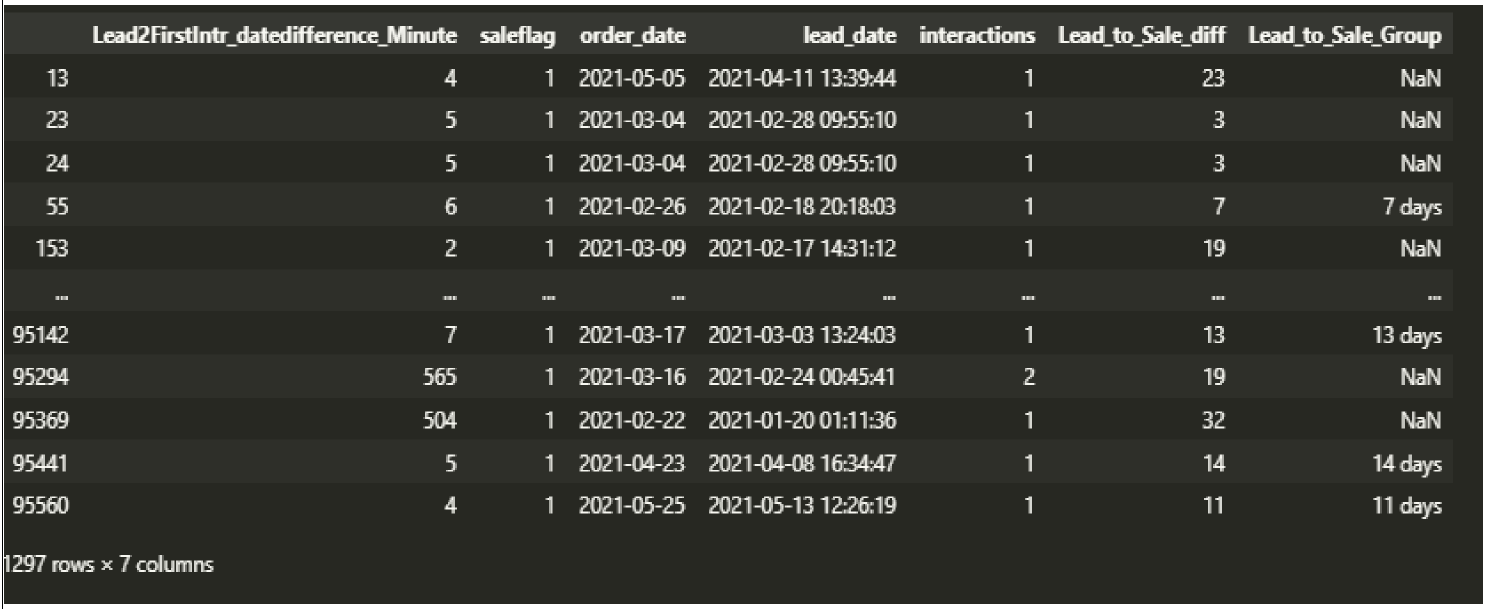
<u>The key challenges faced by sales management to identify the priority of leads</u>

Mostly its very difficult to determine how to prioritize leads, classifying them without any proper data driven solution is just beating around the bush, which is technically a absolute waste of time and efforts, thus to mitigate these problems more efficiently the module can be used for better understanding of lead prioritization.

Data Overview

The data used to demonstrate this module is basically a sales data having time taken to contact lead for the very first time units in minutes(Lead2FirstIntr_datedifference_Minute), Sale non-sale column(saleflag) with 0 referring to non-sale and 1 referring sale, date of order(order_date), lead generation date(lead_date), number of interactions done by sales team after the generation of lead(interactions) and difference of date between order date and lead generation date(Lead_to_Sale_diff). The columns are described below with their data types

Dataset column names	Description	Data type
Lead2FirstIntr_datedifference_Minute	time taken to contact lead for the very first time units in minutes	Continuous
saleflag	'0' stands for unsold and '1' for sold entries	Binary discrete
order_date	Sales Date	Date
lead_date	Date on which lead was generated	Date
interactions	Number of times a lead was contacted after it was generated	Discrete count
Lead_to_Sale_diff	Number of days taken for the lead to get converted to sale	Discrete
Lead_to_Sale_Group	Group of days	



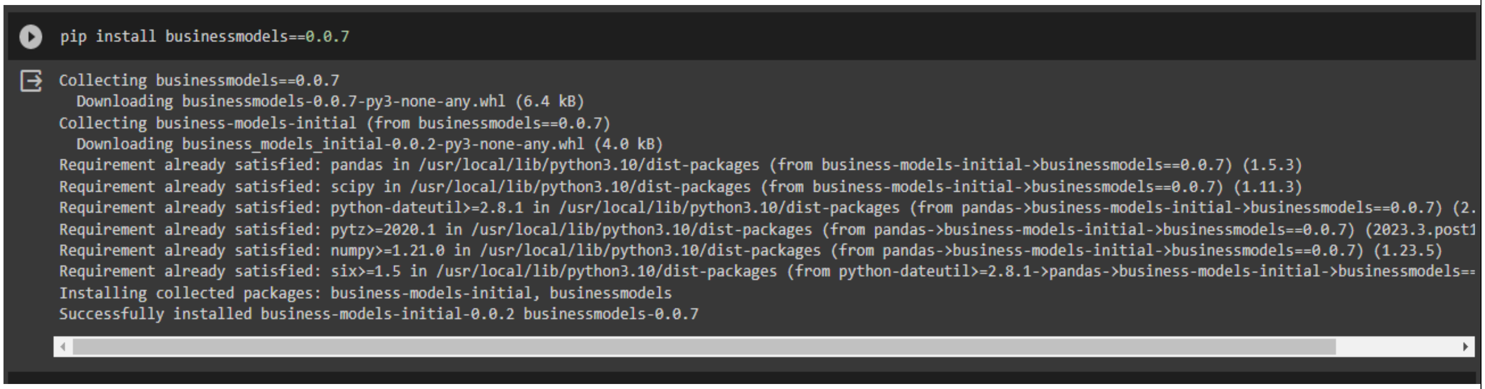
	Lead2FirstIntr_datedifference_Minute	saleflag	order_date	lead_date	interactions	Lead_to_Sale_diff	Lead_to_Sale_Group
13	4	1	2021-05-05	2021-04-11 13:39:44	1	23	NaN
23	5	1	2021-03-04	2021-02-28 09:55:10	1	3	NaN
24	5	1	2021-03-04	2021-02-28 09:55:10	1	3	NaN
55	6	1	2021-02-26	2021-02-18 20:18:03	1	7	7 days
153	2	1	2021-03-09	2021-02-17 14:31:12	1	19	NaN
...	...	...	...	...	...	...	...
95142	7	1	2021-03-17	2021-03-03 13:24:03	1	13	13 days
95294	565	1	2021-03-16	2021-02-24 00:45:41	2	19	NaN
95369	504	1	2021-02-22	2021-01-20 01:11:36	1	32	NaN
95441	5	1	2021-04-23	2021-04-08 16:34:47	1	14	14 days
95560	4	1	2021-05-25	2021-05-13 12:26:19	1	11	11 days

1297 rows × 7 columns

INSTALLATION

Installation of the package is very simple just typing the code referred bellow we can install our package into our python environment

```
pip install businessmodels==0.0.7
```



```
pip install businessmodels==0.0.7
```

```
Collecting businessmodels==0.0.7
  Downloading businessmodels-0.0.7-py3-none-any.whl (6.4 kB)
Collecting business-models-initial (from businessmodels==0.0.7)
  Downloading business_models_initial-0.0.2-py3-none-any.whl (4.0 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from business-models-initial->businessmodels==0.0.7) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from business-models-initial->businessmodels==0.0.7) (1.11.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->business-models-initial->businessmodels==0.0.7) (2.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->business-models-initial->businessmodels==0.0.7) (2023.3.post1
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->business-models-initial->businessmodels==0.0.7) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->business-models-initial->businessmodels==
Installing collected packages: business-models-initial, businessmodels
Successfully installed business-models-initial-0.0.2 businessmodels-0.0.7
```

The module needs to be imported from the businessmodels package using the following line of code

```
from businessmodels import get_sla
```

```
sla_determination = get_sla.SLADetermination(df)
```

In this line, the code is creating an instance of the "SLADetermination" class from the "get_sla" module and assigning it to a variable named "sla_determination". The instance of the "SLADetermination" class is assigned to the variable "sla_determination," which can be used later in the code to interact with the functionalities provided by the "SLADetermination" class. We will use this instance to call the all functions within the class.

The DataFrame df is passed as an argument to the "SLADetermination" class constructor. This suggests that the "SLADetermination" class expects a DataFrame as input.

```
sla_determination.filter_and_convert_dates(st,et)
```

This is the method call, and it take two arguments, “st” and “et”. Start date and end date considering the timeframe of the data.

```
sla_determination.calculate_lead_to_sale_diff()
```

The code sla_determination.calculate_lead_to_sale_diff() appears to be another method call on the sla_determination object, which is an instance of the "SLADetermination" class.

```
x,y = sla_determination.categorize_lead_to_sale_calculate()
```

The code x, y = sla_determination.categorize_lead_to_sale_calculate() is assigning the results of a method call to two variables, x and y.

CODES

```
import pandas as pd

class SLADetermination:
    def __init__(self, df):
        self.df = df

    def filter_and_convert_dates(self, st, et):
        selected_columns = ['Lead2FirstIntr_datedifference_Minute', 'lead_source', 'saleflag', 
'salesman_id', 'order_date', 'order_term', 'net_amount', 'matched_table', 'lead_date', 
'interactions']
```

```
        self.df = self.df[selected_columns]
      self.df = self.df[(self.df['saleflag'] == 1) & ((self.df['order_date'] >= st) &
(self.df['order_date'] < et))]
 self.df['order_date'] = pd.to_datetime(self.df['order_date'])
        self.df['lead_date'] = pd.to_datetime(self.df['lead_date'])

    def calculate_lead_to_sale_diff(self):
        self.df['Lead_to_Sale_diff'] = (self.df['order_date'] - self.df['lead_date']).dt.days
        self.df = self.df[self.df['Lead_to_Sale_diff'] > 0]
     def categorize_lead_to_sale_calculate(self):
        bins = [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
        labels = ['7 days', '8 days', '9 days', '10 days', '11 days', '12 days', '13 days', '14
days', '15 days', '16 days', '17 days']
        self.df['Lead_to_Sale_Group'] = pd.cut(self.df['Lead_to_Sale_diff'], bins=bins,
labels=labels, right=False)

        self.df_sale_inter = self.df.copy()
        bins_int = [2, 3, 4, 5, 6, 7, 8, 9, 10]
        labels = ['2 Interactions', '3 Interactions', '4 Interactions', '5 Interactions', '6
Interactions', '7 Interactions', '8 Interactions', '9 Interactions']
        self.df_sale_inter['Lead_to_Sale_Interactions_Group'] =
pd.cut(self.df_sale_inter['interactions'], bins=bins_int, labels=labels, right=False)
      self.df_sale_age_cum =
      self.df.groupby('Lead_to_Sale_Group').size().reset_index(name='Count')
        self.df_sale_age_cum['Cumulative_Count'] = self.df_sale_age_cum['Count'].cumsum()
        self.df_sale_age_cum['Cumulative_Percentage'] = (self.df_sale_age_cum['Count'].cumsum()
/ self.df_sale_age_cum['Count'].sum()) * 100
       self.df_sale_int_cum =
self.df_sale_inter.groupby('Lead_to_Sale_Interactions_Group').size().reset_index(name='Count')
        self.df_sale_int_cum['Cumulative_Count'] = self.df_sale_int_cum['Count'].cumsum()
        self.df_sale_int_cum['Cumulative_Percentage'] = (self.df_sale_int_cum['Count'].cumsum()
/ self.df_sale_int_cum['Count'].sum()) * 100

        return self.df_sale_age_cum, self.df_sale_int_cum
```

IMPLEMENTATION

Selecting Columns:

The code starts by selecting specific columns from a DataFrame (df) and stores them in a new DataFrame called df_selected. The selected columns are 'Lead2FirstIntr_datedifference_Minute', 'saleflag', 'order_date', 'lead_date', and 'interactions'

Filtering Data for Sales and Date Range:

It then filters the df_selected DataFrame to only keep rows where 'saleflag' is equal to 1 (indicating a sale) and where the 'order_date' falls within a specified date range (st to et).

Filtering Data for Sales and Date Range:

It then filters the df_selected DataFrame to only keep rows where 'saleflag' is equal to 1 (indicating a sale) and where the 'order_date' falls within a specified date range (st to et).

Calculating Lead-to-Sale Time:

Calculates the time difference between 'order_date' and 'lead_date' for each row and stores this difference in a new column called 'Lead_to_Sale_diff'. This difference is measured in days.

Copying Data for Interactions:

Creates a copy of the filtered data in df_sale_age and assigns it to df_sale_inter.

Filtering Negative Lead-to-Sale Times:

Filters the df_sale_age DataFrame to exclude rows where 'Lead_to_Sale_diff' is less than or equal to zero, meaning the sale occurred before or on the same day as the lead.

Categorizing Lead-to-Sale Time:

Groups the remaining data in df_sale_age into categories based on the 'Lead_to_Sale_diff' values. It creates categories like '7 days', '8 days', etc., and assigns each row to one of these categories based on the number of days it took for the sale to happen after the lead.

Categorizing Interaction Counts:

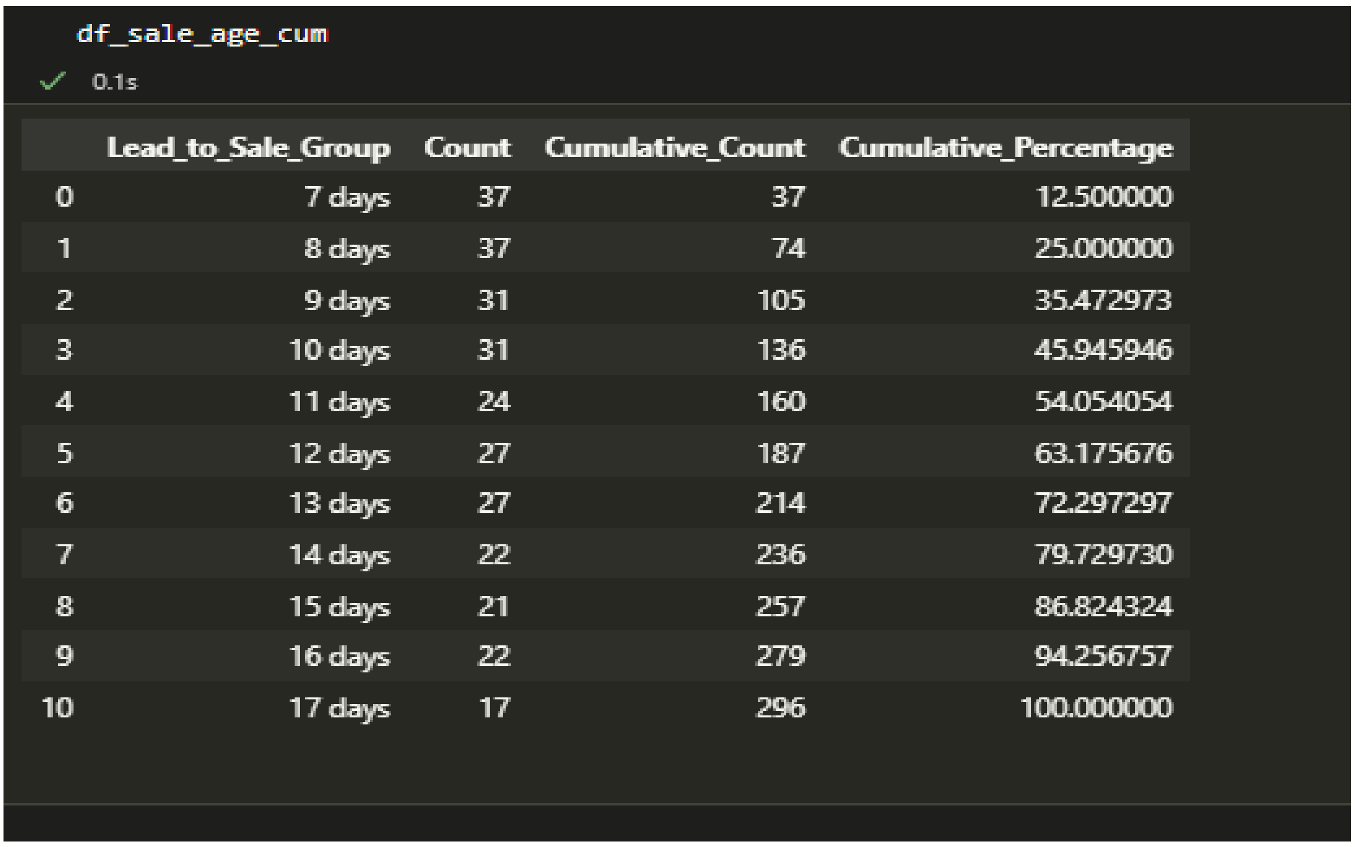
Groups the data in df_sale_inter into categories based on the 'interactions' column. It creates categories like '2 Interactions', '3 Interactions', etc., based on the number of interactions before the sale.

Calculating Cumulative Statistics for Interaction Counts:

Groups the data in df_sale_inter by the categories created in step 8 ('Lead_to_Sale_Interactions_Group') and calculates the count of rows in each category. It also calculates the cumulative count and cumulative percentage of rows up to each category.

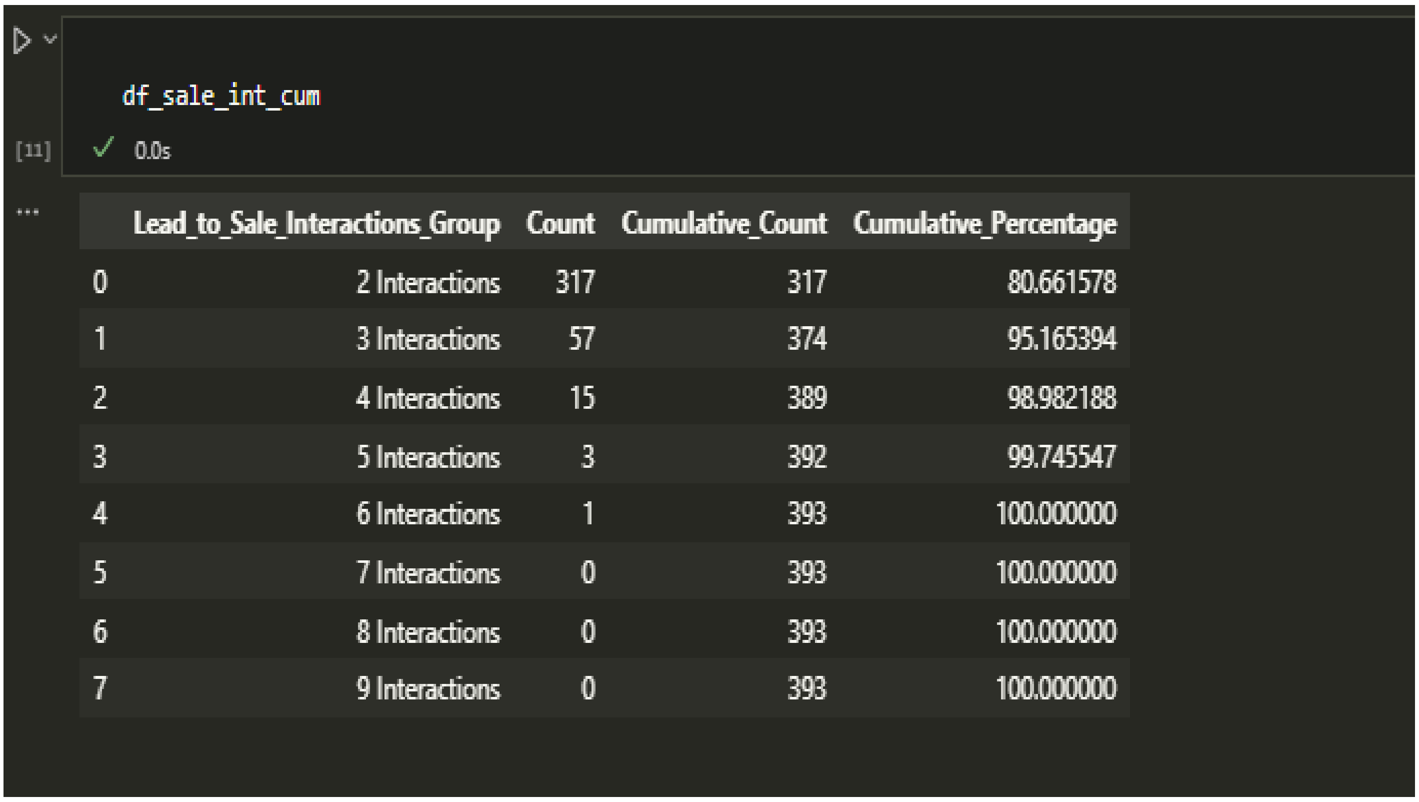
Returning Results:

Finally, the code returns two DataFrames, df_sale_age_cum and df_sale_int_cum, which contain the cumulative statistics for lead-to-sale times and interaction counts, respectively. These DataFrames provide insights into how long it takes for leads to convert into sales and how the number of interactions influences the conversion.



```
df_sale_age_cum
✓ 0.1s
```

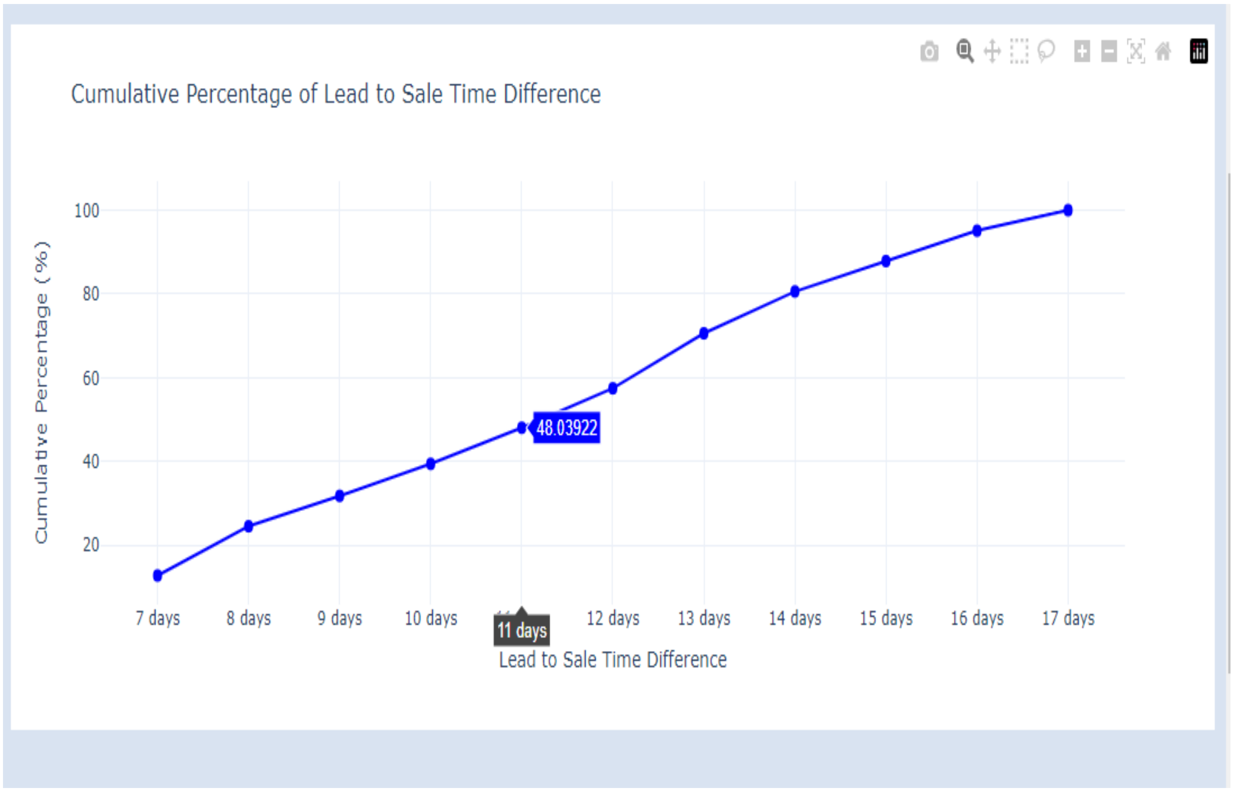
	Lead_to_Sale_Group	Count	Cumulative_Count	Cumulative_Percentage
0	7 days	37	37	12.500000
1	8 days	37	74	25.000000
2	9 days	31	105	35.472973
3	10 days	31	136	45.945946
4	11 days	24	160	54.054054
5	12 days	27	187	63.175676
6	13 days	27	214	72.297297
7	14 days	22	236	79.729730
8	15 days	21	257	86.824324
9	16 days	22	279	94.256757
10	17 days	17	296	100.000000

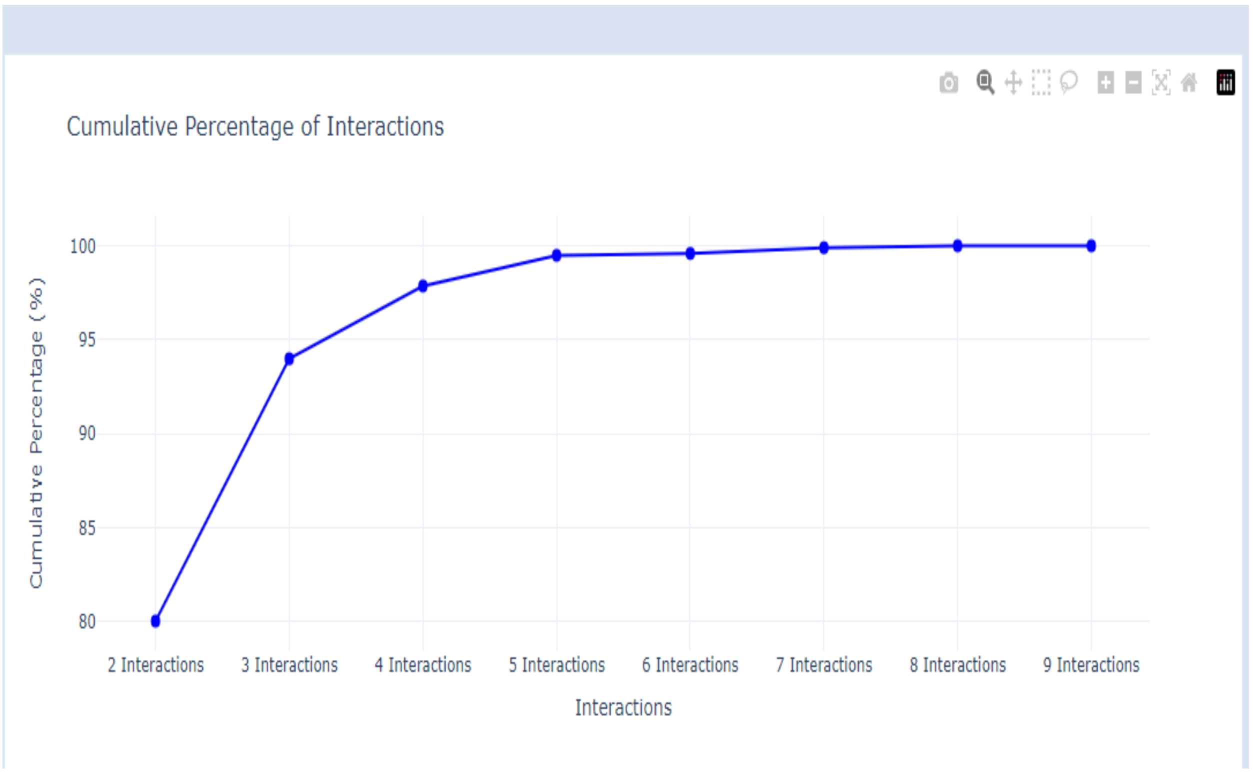


```
df_sale_int_cum
```

[11] ✓ 0.0s

	Lead_to_Sale_Interactions_Group	Count	Cumulative_Count	Cumulative_Percentage
0	2 Interactions	317	317	80.661578
1	3 Interactions	57	374	95.165394
2	4 Interactions	15	389	98.982188
3	5 Interactions	3	392	99.745547
4	6 Interactions	1	393	100.000000
5	7 Interactions	0	393	100.000000
6	8 Interactions	0	393	100.000000
7	9 Interactions	0	393	100.000000





CONCLUSION

The module named "get_sla" published under python package "businessmodels" makes it easy to get interpreted result to prioritize or understate leads while they are in the system. The output table with necessary fields like Lead_to_Sale_Group, Lead_to_Sale_Interactions_Group and their Cumulative_Percentage components helps to go for further analysis.

PACKAGE REFERENCE

For your reference you can visit the https://pypi.org/project/packageimportant/#description page where this python package is available.