Assignment

Imports needed for the analysis.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

Data cleaning and Pre-processing

Combining all the sheets together. We have created a new column 'Employee_Name' and combined all the sheets together with the help of this column. We could have avaoided this step altogether, but I have done this as there can be more associates as well, and having a large number of sheets to perform operations on can create problems.

```
file_path = '/content/Data Assignment.xlsx'  # Adjust if your path differs

# Read all sheets
xls = pd.ExcelFile(file_path)
sheet_names = xls.sheet_names

# Combine all sheets into a single DataFrame
dataframes = []
for sheet in sheet_names:
    df = pd.read_excel(xls, sheet_name=sheet)
    df['Employee_Name'] = sheet  # Add employee name column
    dataframes.append(df)

combined_df = pd.concat(dataframes, ignore_index=True)
```

Displaying the name of the sheets in the dataset.

```
# Display sheet names (optional)
print("Sheet Names:", xls.sheet_names)
```

```
Sheet Names: ['Raj', 'Arya', 'Ali']
```

The first rows of the dataset. Here we can see the addition of the new column 'Employee_name'.

```
combined_df.head()
```

	Day	Date	Leads	Time spent on LG (mins)	Avg Time Per Lead (mins)	Daily Team Review	No. of Incomplete Leads	Employee_Name
0	Mon	2023-06-12	7.0	240.0	34.0	Attended	0.0	Raj
1	Tue	2023-06-13	10.0	360.0	36.0	Attended	3.0	Raj
2	Wed	2023-06-	10.0	270.0	27.0	Attended	0.0	Raj

Next steps: Generate code with combined_df | View recommended plots | New interactive sheet

The datatypes of the column.

```
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183 entries, 0 to 182
```

```
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Day                       183 non-null    object
 1   Date                      183 non-null    datetime64[ns]
 2   Leads                     123 non-null    float64
 3   Time spent on LG (mins)   123 non-null    float64
 4   Avg Time Per Lead (mins)  123 non-null    float64
 5   Daily Team Review         124 non-null    object
 6   No. of Incomplete Leads   124 non-null    float64
 7   Employee_Name             183 non-null    object
dtypes: datetime64[ns](1), float64(4), object(3)
memory usage: 11.6+ KB
```

```
# Check data types
print("Data Types:\n", combined_df.dtypes)
```

```
Data Types:
 Day                              object
Date                      datetime64[ns]
Leads                            float64
Time spent on LG (mins)          float64
Avg Time Per Lead (mins)         float64
Daily Team Review                 object
No. of Incomplete Leads          float64
Employee_Name                     object
dtype: object
```

Finding the missing values in the dataset.

```
# Check for missing values
print("\nMissing Values:\n", combined_df.isnull().sum())
```

```
Missing Values:
 Day                         0
Date                         0
Leads                       60
Time spent on LG (mins)     60
Avg Time Per Lead (mins)    60
Daily Team Review           59
No. of Incomplete Leads     59
Employee_Name                0
dtype: int64
```

Most of the rows which had missing values were the rows with day as 'Sat' and 'Sun', we can consider these as holidays.

There was one row which had only one missing value in its column. I could have imputed the value but I did not do so as it could have added a bias.Therefore removing the row altogether felt like a good option.

```
# Remove rows with any missing values
cleaned_df = combined_df.dropna()

# Optionally reset index after dropping
cleaned_df.reset_index(drop=True, inplace=True)

# Confirm no missing values remain
print("Missing Values After Cleaning:\n", cleaned_df.isnull().sum())
```

```
Missing Values After Cleaning:
 Day                         0
Date                         0
Leads                        0
Time spent on LG (mins)      0
Avg Time Per Lead (mins)     0
Daily Team Review            0
No. of Incomplete Leads      0
Employee_Name                0
dtype: int64
```

Total columns available.

```
# Check column names for reference
print("Columns available:", cleaned_df.columns)
```

```
Columns available: Index(['Day', 'Date', 'Leads', 'Time spent on LG (mins)',
       'Avg Time Per Lead (mins)', 'Daily Team Review',
       'No. of Incomplete Leads', 'Employee_Name'],
      dtype='object')
```

Identifying the outliers in the dataset with the help of IQR method. with this method we find outliers in columns 'Leads' and 'No. of Incomplete Leads'.

Assessing the Outliers.

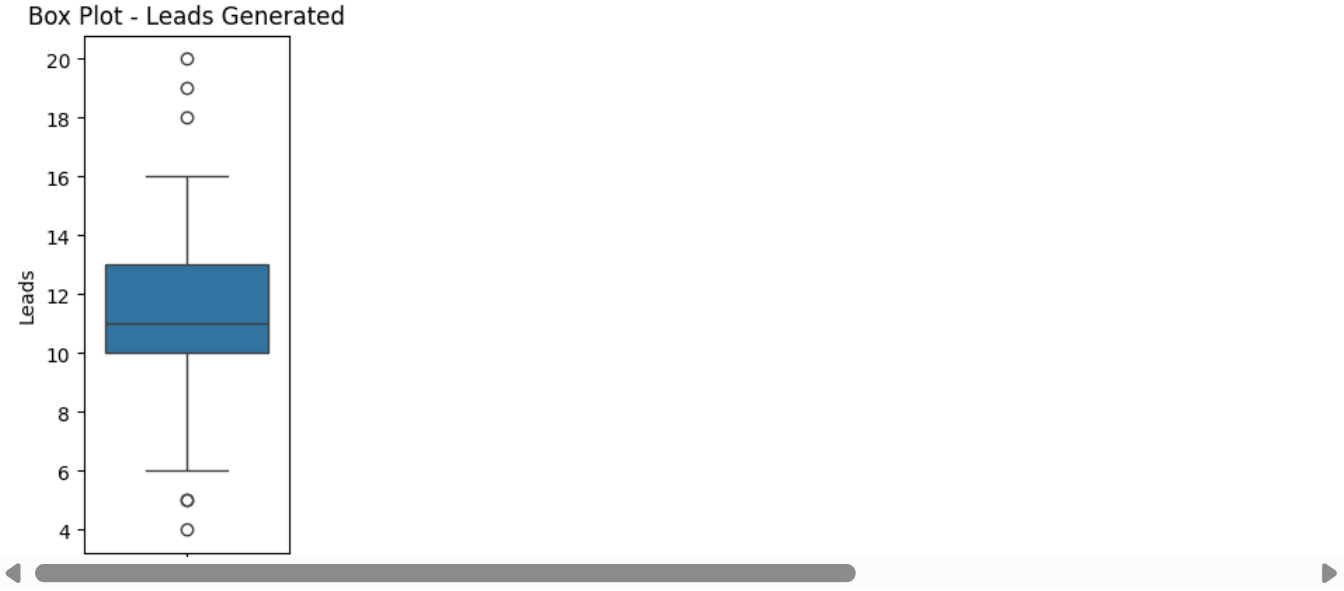
```
# Identify outliers using IQR for Leads_Generated and Time_Spent
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers

outliers_leads = detect_outliers_iqr(cleaned_df, 'Leads')
outliers_time = detect_outliers_iqr(cleaned_df, 'Time spent on LG (mins)')
outliers_average_time = detect_outliers_iqr(cleaned_df, 'Avg Time Per Lead (mins)')
outliers_incomplete_leads= detect_outliers_iqr(cleaned_df, 'No. of Incomplete Leads')

print(f"\nOutliers in Leads_Generated:\n{outliers_leads}")
print(f"\nOutliers in Time_Spent (in mins):\n{outliers_time}")
print(f"\nOutliers in Avg_Time_Per_Lead (in mins):\n{outliers_average_time}")
print(f"\nOutliers in No. of Incomplete Leads:\n{outliers_incomplete_leads}")
```

```
Outliers in Leads_Generated:
     Day       Date  Leads  Time spent on LG (mins)  Avg Time Per Lead (mins)  \
11   Wed 2023-06-28   19.0                    360.0                      19.0
12   Fri 2023-06-30   18.0                    370.0                      21.0
32   Wed 2023-08-02    4.0                     45.0                      11.0
70   Fri 2023-07-28    5.0                     46.0                       9.0
81   Mon 2023-06-12    5.0                    135.0                      27.0
111  Fri 2023-07-28   20.0                    360.0                      18.0

    Daily Team Review  No. of Incomplete Leads Employee_Name
11           Attended                     0.0           Raj
12           Attended                     1.0           Raj
32           Attended                     0.0           Raj
70           Attended                     0.0          Arya
81           Attended                     0.0           Ali
111          Attended                     0.0           Ali

Outliers in Time_Spent (in mins):
Empty DataFrame
Columns: [Day, Date, Leads, Time spent on LG (mins), Avg Time Per Lead (mins), Daily Team Review, No. of Incomplete Leads, Em
Index: []

Outliers in Avg_Time_Per_Lead (in mins):
Empty DataFrame
Columns: [Day, Date, Leads, Time spent on LG (mins), Avg Time Per Lead (mins), Daily Team Review, No. of Incomplete Leads, Em
Index: []

Outliers in No. of Incomplete Leads:
    Day       Date  Leads  Time spent on LG (mins)  Avg Time Per Lead (mins)  \
1   Tue 2023-06-13   10.0                    360.0                      36.0
7   Wed 2023-06-21    8.0                    240.0                      30.0
8   Fri 2023-06-23    9.0                    230.0                      26.0
9   Mon 2023-06-26   14.0                    280.0                      20.0
10  Tue 2023-06-27    7.0                    260.0                      37.0
12  Fri 2023-06-30   18.0                    370.0                      21.0
48  Fri 2023-06-23   10.0                    110.0                      11.0
82  Tue 2023-06-13    9.0                    210.0                      23.0

   Daily Team Review  No. of Incomplete Leads Employee_Name
1           Attended                     3.0           Raj
7           Attended                     2.0           Raj
8           Attended                     1.0           Raj
9           Attended                     1.0           Raj
10          Attended                     6.0           Raj
12          Attended                     1.0           Raj
48          Attended                     9.0          Arya
82          Attended                     3.0           Ali
```

Displaying the box plot of the column 'Leads'. We can observe the outliers found by the IQR range.

```
# Leads_Generated box plot
plt.subplot(1, 3, 1)
sns.boxplot(data=cleaned_df, y='Leads')
plt.title('Box Plot - Leads Generated')
```

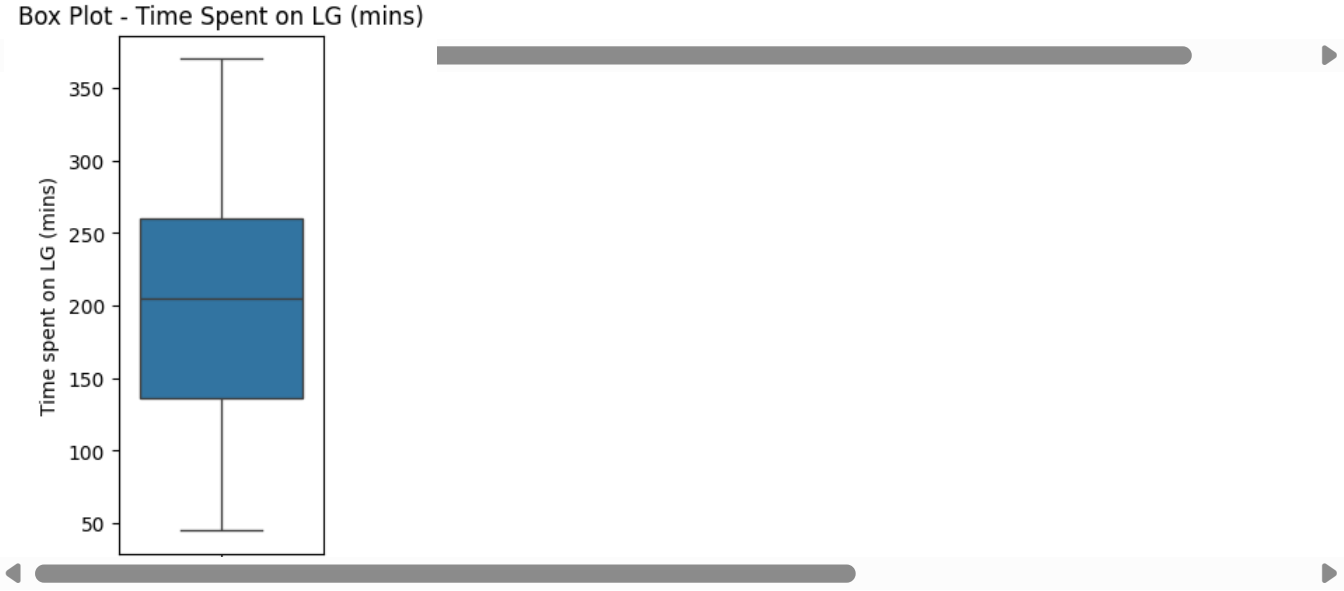
```
Text(0.5, 1.0, 'Box Plot - Leads Generated')
```



Displaying the box plot for the column 'Time Spent on LG(mins)'.

```
# Time Spent box plot
plt.subplot(1, 3, 2)
sns.boxplot(data=cleaned_df, y='Time spent on LG (mins)')
plt.title('Box Plot - Time Spent on LG (mins)')
```

```
Text(0.5, 1.0, 'Box Plot - Time Spent on LG (mins)')
```



Displaying the box plot for the column 'Avg Time Per Lead(mins)'.

```
plt.subplot(1, 3, 2)
sns.boxplot(data=cleaned_df, y='Avg Time Per Lead (mins)')
plt.title('Box Plot - Avg Time Per Lead (mins)')
```

```
Text(0.5, 1.0, 'Box Plot - Avg Time Per Lead (mins)')
```

Box plot for column 'No. of Incomplete Leads'. We can observe the outliers found through the outliers method.

```
plt.subplot(1, 3, 3)
sns.boxplot(data=cleaned_df, y='No. of Incomplete Leads')
plt.title('Box Plot - No. of Incomplete Leads')
```

```
Text(0.5, 1.0, 'Box Plot - No. of Incomplete Leads')
```

Assessing the outliers through the Z-score method. We find no outliers in the dataset though this method. Therefore removing outliers found through the IQR method isn't the best option anymore. Also after observing the dataset we can observe that there aren't values which can introduce much of a bias.

```
from scipy.stats import zscore

# Apply Z-score calculation
z_scores = combined_df[['Leads', 'Time spent on LG (mins)', 'No. of Incomplete Leads']].apply(zscore)

# Define threshold
threshold = 3

# Boolean mask for rows where any column has a Z-score > threshold
outliers_z = (z_scores.abs() > threshold).any(axis=1)

# Extract outlier rows
z_outliers_df = combined_df[outliers_z]

# Display the outliers
print("Outliers detected using Z-score method:")
```

```
print(z_outliers_df[['Date', 'Employee_Name', 'Leads', 'Time spent on LG (mins)', 'No. of Incomplete Leads']])
```

```
Outliers detected using Z-score method:
Empty DataFrame
Columns: [Date, Employee_Name, Leads, Time spent on LG (mins), No. of Incomplete Leads]
Index: []
```

Initial Observation.

The data shape after cleaning.

```
# Resulting cleaned data
print("Cleaned data shape:", cleaned_df.shape)
print(cleaned_df.head())
```

```
Cleaned data shape: (122, 8)
   Day       Date  Leads  Time spent on LG (mins)  Avg Time Per Lead (mins)  \
0  Mon 2023-06-12    7.0                    240.0                      34.0
1  Tue 2023-06-13   10.0                    360.0                      36.0
2  Wed 2023-06-14   10.0                    270.0                      27.0
3  Thu 2023-06-15   10.0                    260.0                      26.0
4  Fri 2023-06-16   10.0                    240.0                      24.0

  Daily Team Review  No. of Incomplete Leads Employee_Name
0          Attended                     0.0           Raj
1          Attended                     3.0           Raj
2          Attended                     0.0           Raj
3          Attended                     0.0           Raj
4          Attended                     0.0           Raj
```

The five point summary and the standard deviation of the dataset.

```
print(cleaned_df.describe())
```

```
                                Date       Leads  Time spent on LG (mins)  \
count                            122  122.000000               122.000000
mean   2023-07-12 04:19:40.327868928   11.459016               206.467213
min              2023-06-12 00:00:00    4.000000                45.000000
25%              2023-06-27 00:00:00   10.000000               136.500000
50%              2023-07-12 00:00:00   11.000000               205.000000
75%              2023-07-28 00:00:00   13.000000               260.000000
max              2023-08-11 00:00:00   20.000000               370.000000
std                              NaN    2.663334                73.365403

       Avg Time Per Lead (mins)  No. of Incomplete Leads
count                122.000000               122.000000
mean                  18.540984                 0.213115
min                    9.000000                 0.000000
25%                   11.250000                 0.000000
50%                   17.500000                 0.000000
75%                   23.750000                 0.000000
max                   40.000000                 9.000000
std                    7.092246                 1.061961
```

Saving the file, as this file can be used to create the Power BI Dashboard.

```
# Save to Excel
output_excel_path = 'cleaned_lead_data.csv'
cleaned_df.to_csv(output_excel_path, index=False)
print(f"Cleaned data saved to: {output_excel_path}")
```

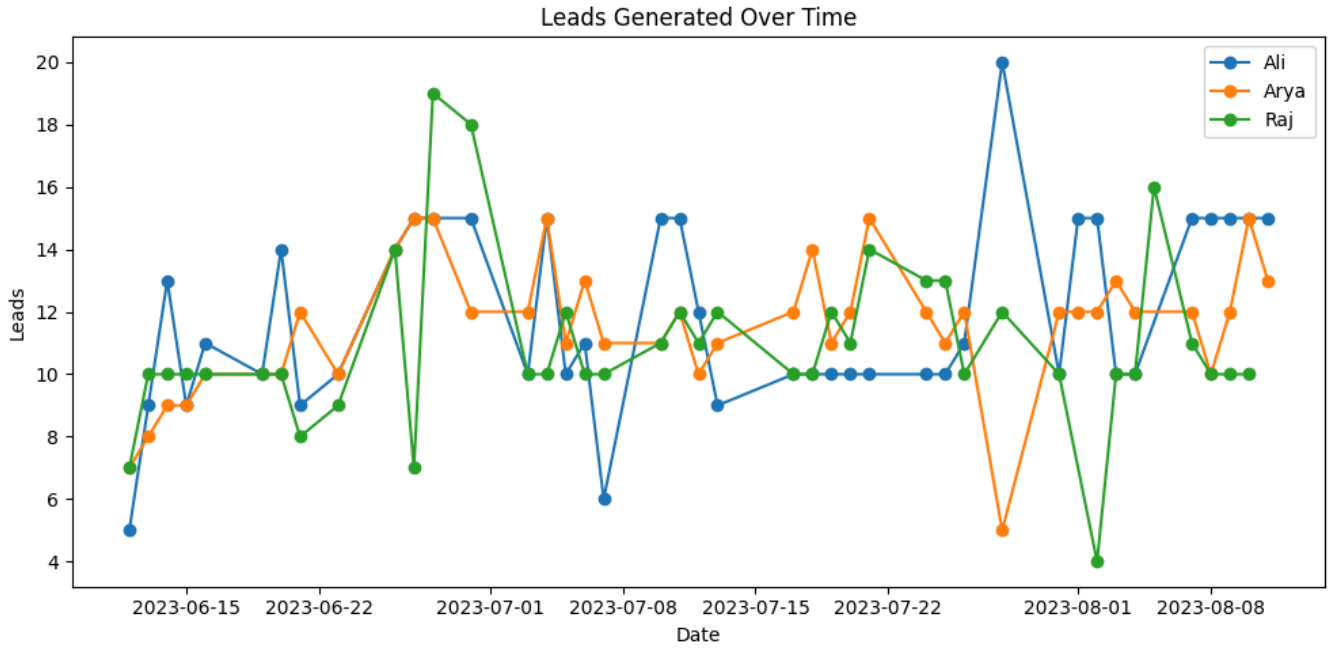
```
Cleaned data saved to: cleaned_lead_data.csv
```

Displaying the leads by the three associates.

```
cleaned_df['Date'] = pd.to_datetime(cleaned_df['Date'])
cleaned_df.sort_values('Date', inplace=True)

plt.figure(figsize=(10, 5))
for name, group in cleaned_df.groupby('Employee_Name'):
    plt.plot(group['Date'], group['Leads'], marker='o', label=name)
```

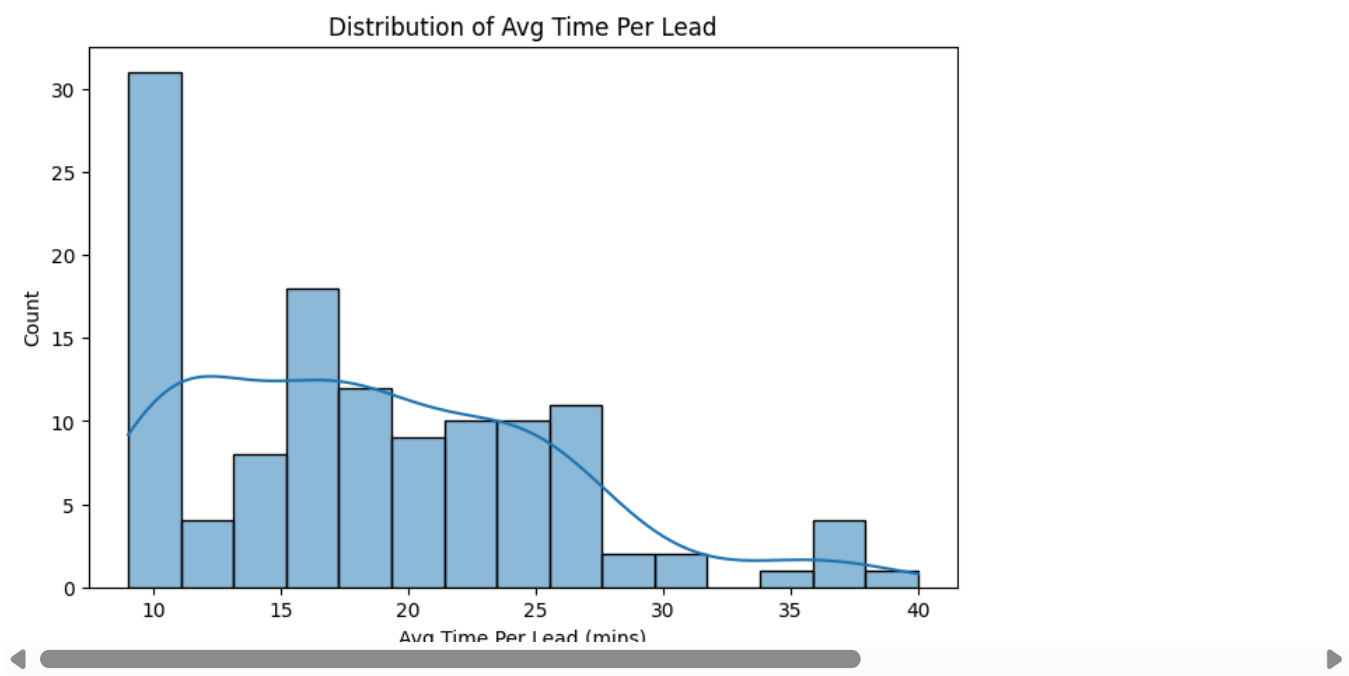
```
plt.title('Leads Generated Over Time')
plt.xlabel('Date')
plt.ylabel('Leads')
plt.legend()
plt.tight_layout()
plt.show()
```

```
<ipython-input-38-9017131e058b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  cleaned_df['Date'] = pd.to_datetime(cleaned_df['Date'])
<ipython-input-38-9017131e058b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  cleaned_df.sort_values('Date', inplace=True)
```



Displaying through a histogram the Average time invested per lead by the associates.

```
plt.figure(figsize=(8, 5))
sns.histplot(combined_df['Avg Time Per Lead (mins)'], kde=True, bins=15)
plt.title('Distribution of Avg Time Per Lead')
plt.xlabel('Avg Time Per Lead (mins)')
plt.show()
```



The mean, standard deviation and the five point summary of each associate for each column.

```
# Group data by employee
grouped = cleaned_df.groupby('Employee_Name')

# Summary statistics for each employee
summary_stats = grouped[['Leads', 'Time spent on LG (mins)', 'Avg Time Per Lead (mins)', 'No. of Incomplete Leads']].describe()
print(summary_stats)
```

```
               Leads                                                \
               count       mean       std  min   25%   50%   75%   max
Employee_Name
Ali             41.0  11.902439  3.039777  5.0  10.0  11.0  15.0  20.0
Arya            41.0  11.560976  2.156951  5.0  10.0  12.0  12.0  15.0
Raj             40.0  10.900000  2.687101  4.0  10.0  10.0  12.0  19.0

               Time spent on LG (mins)              ...  \
                                 count        mean  ...
Employee_Name                                       ...
Ali                               41.0  225.243902  ...
Arya                              41.0  135.829268  ...
Raj                               40.0  259.625000  ...

               Avg Time Per Lead (mins)       No. of Incomplete Leads  \
                                    75%   max                   count
Employee_Name
Ali                                20.0  40.0                    41.0
Arya                               12.0  28.0                    41.0
Raj                                26.0  37.0                    40.0


                    mean       std  min  25%  50%  75%  max
Employee_Name
Ali             0.073171  0.468521  0.0  0.0  0.0  0.0  3.0
Arya            0.219512  1.405564  0.0  0.0  0.0  0.0  9.0
Raj             0.350000  1.098951  0.0  0.0  0.0  0.0  6.0

[3 rows x 32 columns]
```

```
employee_summary = grouped.agg({
    'Leads': ['mean', 'median', 'count'],
    'Time spent on LG (mins)': ['mean', 'median'],
    'Avg Time Per Lead (mins)': ['mean', 'median'],
    'No. of Incomplete Leads': ['mean', 'median', 'sum']
})

print(employee_summary)
```

```
                   Leads             Time spent on LG (mins)         \
                    mean median count                    mean median
Employee_Name
```

```
Ali            11.902439   11.0    41              225.243902  240.0
Arya           11.560976   12.0    41              135.829268  130.0
Raj            10.900000   10.0    40              259.625000  260.0

              Avg Time Per Lead (mins)        No. of Incomplete Leads         \
                                  mean median                    mean median
Employee_Name
Ali                         19.439024   17.0                0.073171    0.0
Arya                        12.024390   11.0                0.219512    0.0
Raj                         24.300000   24.0                0.350000    0.0


                sum
Employee_Name
Ali             3.0
Arya            9.0
Raj            14.0
```

Initial Observations from the Dataset

1. Missing Data Patterns:

The dataset shows missing rows primarily on weekends—Saturday and Sunday—which aligns with the business being closed on those days.

Additionally, some weekdays (e.g., Thursday, 12th row) are also missing across all associates, suggesting company-wide holidays rather than data collection errors.

These missing entries appear consistently across all employees, reinforcing the idea of non-working days.

2. Incomplete Leads Analysis:

Raj has the highest number of incomplete leads spread across multiple days, which may indicate inconsistency in follow-ups or productivity issues.

Arya had incomplete leads on one day only, likely representing an anomaly or a bad day, as her overall pattern shows good follow-through.

Ali demonstrates the best performance in this regard—only 3 out of 9 incomplete leads, all concentrated on a single day—suggesting strong consistency.

3. Efficiency of Lead Handling:

Arya has the lowest average time per lead, implying a more efficient workflow or possibly handling simpler lead types.

Ali and Raj spend more time per lead, which may suggest either more complex lead interactions or lower efficiency.

Overall Impressions:

Arya appears to be the most consistent and time-efficient associate.

Ali performs strongly with both lead volume and incomplete lead reduction.

Raj, while generating a fair number of leads, struggles with incomplete leads and time management.

Questions

1. Lead Generation Efficiency: Calculate the lead generation efficiency for each associate as the ratio of total leads generated to the total time spent on lead generation. Which associate has the highest efficiency?

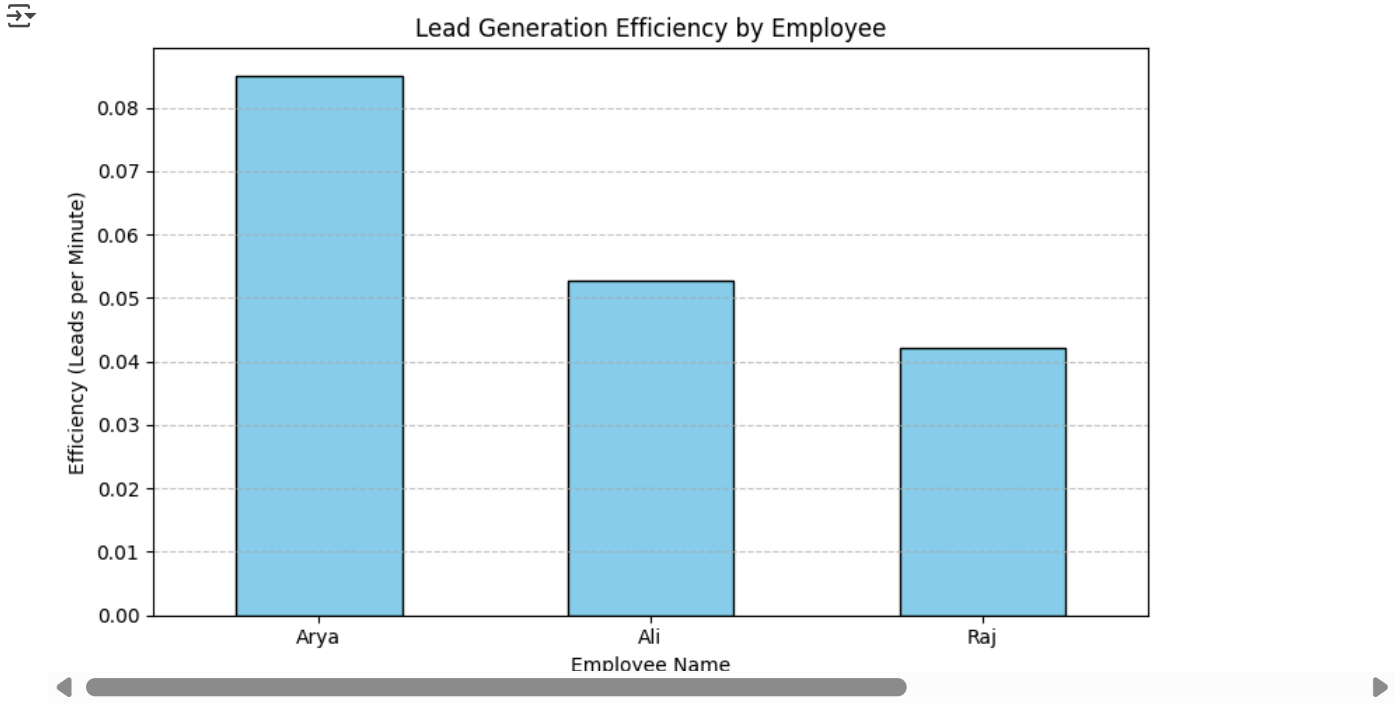
```
# Group by employee and calculate total leads and total time spent
efficiency_df = combined_df.groupby('Employee_Name').agg({
    'Leads': 'sum',
    'Time spent on LG (mins)': 'sum'
})

# Calculate efficiency
efficiency_df['Efficiency (Leads per Minute)'] = efficiency_df['Leads'] / efficiency_df['Time spent on LG (mins)']

# Sort by efficiency
efficiency_df = efficiency_df.sort_values('Efficiency (Leads per Minute)', ascending=False)

print(efficiency_df)
```

```
               Leads  Time spent on LG (mins)  Efficiency (Leads per Minute)
Employee_Name
Arya           474.0                   5569.0                       0.085114
```

```
Ali           488.0                 9235.0                      0.052842
Raj           447.0                10585.0                      0.042230
```

Displaying the efficiency through a bar chart for better analysis.

```
# Plot the efficiency bar chart
plt.figure(figsize=(8, 5))
efficiency_df['Efficiency (Leads per Minute)'].plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Lead Generation Efficiency by Employee')
plt.ylabel('Efficiency (Leads per Minute)')
plt.xlabel('Employee Name')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

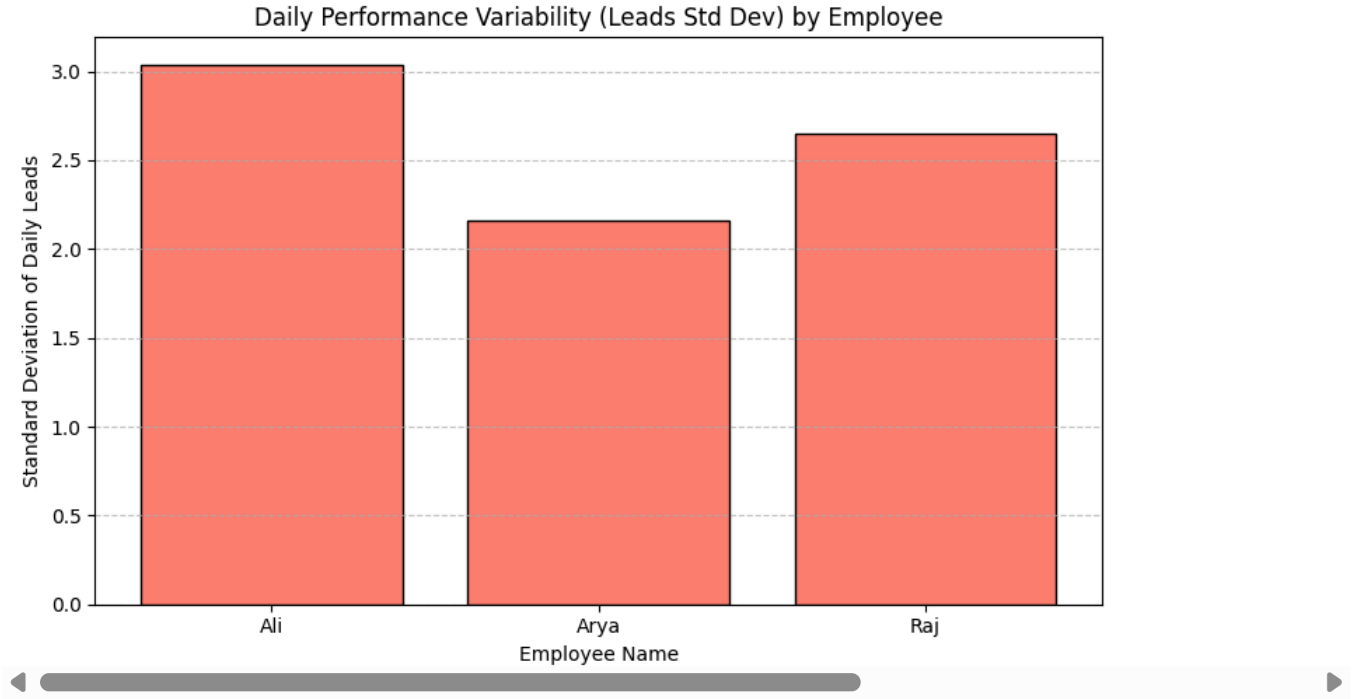


Therefore we can observe from the numbers and from the graph that Arya shows better efficiency among all the three associates. Which means she gets more leads in less amount of time.

2. Daily Performance Variability: Determine the standard deviation of the daily number of leads generated by each associate. Which associate shows the highest variability in daily performance?

```
# Group by Employee and calculate standard deviation of daily leads
variability_df = combined_df.groupby('Employee_Name')['Leads'].std().reset_index()
variability_df.columns = ['Employee_Name', 'Lead Std Dev']

# Identify the associate with highest variability
highest_var = variability_df.loc[variability_df['Lead Std Dev'].idxmax()]

print("Employee with highest daily performance variability:")
print(highest_var)

# Plot the bar chart
plt.figure(figsize=(8, 5))
plt.bar(variability_df['Employee_Name'], variability_df['Lead Std Dev'], color='salmon', edgecolor='black')

plt.title('Daily Performance Variability (Leads Std Dev) by Employee')
plt.xlabel('Employee Name')
plt.ylabel('Standard Deviation of Daily Leads')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

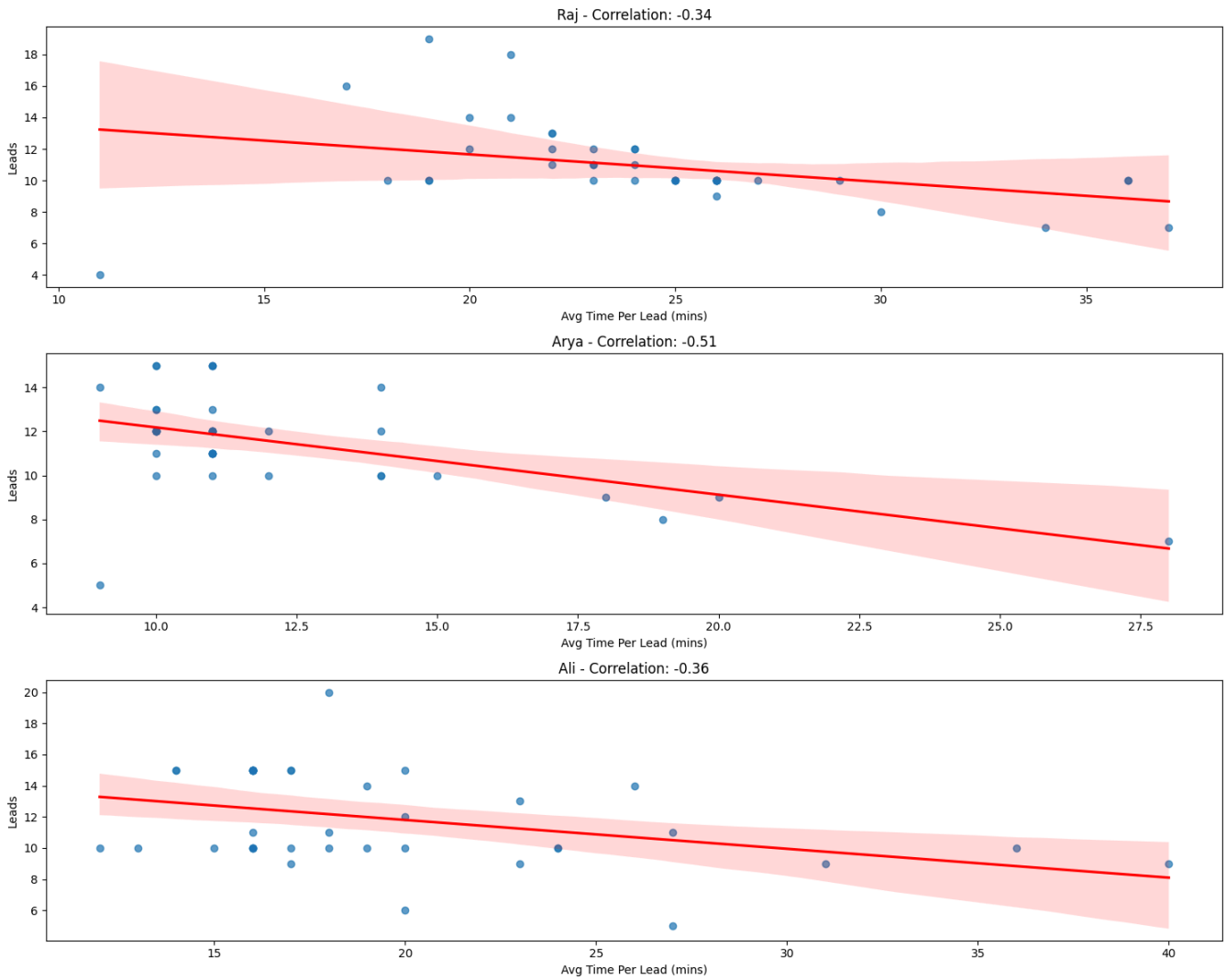
```
Employee with highest daily performance variability:
Employee_Name         Ali
Lead Std Dev     3.039777
Name: 0, dtype: object
```



Therefore, Ali shows the highest variability, which indicates that somedays Ali performs the very well while on the others he is not able to give his best.

3. Time Management Analysis: Analyze the relationship between the average time per lead and the total number of leads generated per day for each associate. Is there a significant correlation?

```
# Unique list of employees
employees = cleaned_df['Employee_Name'].unique()

# Set up the subplot layout
plt.figure(figsize=(15, 4 * len(employees)))

# For storing correlation values
correlations = {}

# Loop through each employee and plot
for i, employee in enumerate(employees):
    emp_data = cleaned_df[cleaned_df['Employee_Name'] == employee]

    # Calculate correlation
    corr = emp_data['Leads'].corr(emp_data['Avg Time Per Lead (mins)'])
    correlations[employee] = corr

    # Plot
    plt.subplot(len(employees), 1, i + 1)
    sns.regplot(
        data=emp_data,
        x='Avg Time Per Lead (mins)',
        y='Leads',
        scatter_kws={'alpha': 0.7},
        line_kws={'color': 'red'}
    )
    plt.title(f'{employee} - Correlation: {corr:.2f}')
    plt.xlabel('Avg Time Per Lead (mins)')
    plt.ylabel('Leads')

plt.tight_layout()
plt.show()

# Print correlation values
print("Correlation Coefficient (Avg Time Per Lead vs Leads) for each employee:")
for emp, coeff in correlations.items():
    print(f"{emp}: {coeff:.2f}")
```




```
Correlation Coefficient (Avg Time Per Lead vs Leads) for each employee:
Raj: -0.34
Arya: -0.51
Ali: -0.36
```

Yes, there is a significant relationship between the average time per lead and the Lead generated.

For each associates:

Raj (-0.34): Moderate negative relationship — he's less efficient when spending more time per lead.

Arya (-0.51): Strongest negative correlation — her productivity drops more significantly as average time per lead increases.

Ali (-0.36): Also shows a moderate decline in leads with more time per lead, similar to Raj.

4. Compare the average number of leads generated on days when daily team reviews were attended versus missed for each associate. What is the percentage difference in performance?

```
# Group by employee and daily team review attendance
review_group = cleaned_df.groupby(['Employee_Name', 'Daily Team Review'])['Leads'].mean().unstack()
```

```
# Calculate percentage difference
review_group['% Difference (Missed vs Attended)'] = (
    ((review_group.get('Missed', 0) - review_group.get('Attended', 0)) / review_group.get('Attended', 1)) * 100
)

# Round values for clarity
review_group = review_group.round(2)

print("Average Leads on Review Days (Attended vs Missed) and % Difference:")
print(review_group)
```

```
Average Leads on Review Days (Attended vs Missed) and % Difference:
Daily Team Review  Attended  Missed  % Difference (Missed vs Attended)
Employee_Name
Ali                   11.92    11.0                             -7.76
Arya                  11.56     NaN                               NaN
Raj                   10.92    10.0                             -8.45
```

Interpretation: Both Ali and Raj perform worse on days when they miss the daily team review.

Raj shows a larger drop (-8.45%) in performance compared to Ali (-7.76%).

Arya has no data for missed review days, so we can't compare her.

Business Insight: Team reviews appear to positively impact daily performance — consider reinforcing attendance or exploring what value the reviews bring (motivation, alignment, planning).

5. Incomplete Leads Reduction Over Time: Calculate the trend (using a linear regression model) of the number of incomplete leads over time for each associate. Are there any significant improvements or deteriorations?

```
from sklearn.linear_model import LinearRegression

# Function to calculate trends for incomplete leads
def calculate_trend(data):
    trends = {}
    for employee in data['Employee_Name'].unique():
        emp_data = data[data['Employee_Name'] == employee].dropna(subset=['No. of Incomplete Leads'])
        if not emp_data.empty:
            X = np.array((emp_data['Date'] - emp_data['Date'].min()).dt.days).reshape(-1, 1)
            y = emp_data['No. of Incomplete Leads'].values
            model = LinearRegression().fit(X, y)
            trends[employee] = model.coef_[0]  # Slope of the regression line
    return trends

# Calculate trends for each associate
incomplete_leads_trend = calculate_trend(cleaned_df)

incomplete_leads_trend
```

```
{'Raj': np.float64(-0.019788172326712147),
 'Arya': np.float64(-0.012477852030637442),
 'Ali': np.float64(-0.006304323562649105)}
```

Insight:

Raj is showing the most improvement.

Arya's trend is worsening slightly — may need support or process review.

Ali is improving but at a slower rate.

6. Performance Consistency: Calculate the coefficient of variation (CV) for the daily leads generated by each associate. Which associate has the most consistent performance?

```
# Coefficient of Variation (CV) = std / mean
cv_df = cleaned_df.groupby('Employee_Name')['Leads'].agg(['mean', 'std'])
cv_df['CV'] = (cv_df['std'] / cv_df['mean']).round(3)

print("Coefficient of Variation for Daily Leads (lower = more consistent):")
print(cv_df)

# Identify the most consistent performer (lowest CV)
most_consistent = cv_df['CV'].idxmin()
```

```
lowest_cv = cv_df['CV'].min()

print(f"\n Most consistent performer: {most_consistent} (CV = {lowest_cv})")
```

```
Coefficient of Variation for Daily Leads (lower = more consistent):
                      mean       std     CV
    Employee_Name
    Ali            11.902439  3.039777  0.255
    Arya           11.560976  2.156951  0.187
    Raj            10.900000  2.687101  0.247

     Most consistent performer: Arya (CV = 0.187)
```

Insight:

Arya is the most reliable performer—her daily lead counts don't fluctuate as much compared to others.

7. High-Performance Days: Identify the top 10% of days with the highest lead generation for each associate. What is the average time spent on lead generation during these high-performance days?

```
top_10_stats = {}

for emp in cleaned_df['Employee_Name'].unique():
    emp_data = cleaned_df[cleaned_df['Employee_Name'] == emp]
    threshold = emp_data['Leads'].quantile(0.90)
    top_days = emp_data[emp_data['Leads'] >= threshold]
    avg_time = top_days['Time spent on LG (mins)'].mean()
    top_10_stats[emp] = round(avg_time, 2)

print("\nAverage Time Spent on Top 10% Lead Days:")
for emp, time in top_10_stats.items():
    print(f"{emp}: {time} mins")
```

```
Average Time Spent on Top 10% Lead Days:
Raj: 314.0 mins
Arya: 161.2 mins
Ali: 251.43 mins
```

Insight: On their best performance days, associates are typically spending more time on lead generation. Raj spends the most time on such days.

8. Impact of Longer Lead Generation Time: Determine if there is a threshold in the time spent on lead generation beyond which the number of leads generated significantly increases. What is the optimal time spent on lead generation for maximizing leads?

```
# Define dictionary to store optimal thresholds
optimal_time_thresholds = {}

for employee in cleaned_df['Employee_Name'].unique():
    emp_data = cleaned_df[cleaned_df['Employee_Name'] == employee].copy()  # Fix 1

    # Bin time spent into equal-width bins
    emp_data['Time_Bin'] = pd.cut(emp_data['Time spent on LG (mins)'], bins=5)

    # Group by bin and calculate average leads
    bin_summary = emp_data.groupby('Time_Bin', observed=False)['Leads'].mean()  # Fix 2

    # Find the bin with maximum lead increase over the previous bin
    lead_diff = bin_summary.diff().fillna(0)
    max_bin = lead_diff.idxmax()

    # Get threshold as midpoint of the max increasing bin
    threshold = (max_bin.left + max_bin.right) / 2
    optimal_time_thresholds[employee] = round(threshold, 2)

# Display thresholds
print("Estimated Optimal Time Thresholds (per employee):")
for emp, th in optimal_time_thresholds.items():
    print(f"{emp}: {th} mins")
```

```
Estimated Optimal Time Thresholds (per employee):
Raj: 337.5 mins
Arya: 91.3 mins
Ali: 240.0 mins
```

Insight: This tells us roughly how much time each person needs to spend before seeing significant improvement in lead count. Arya is more efficient, needing less time

9. Comparative Day Analysis: Calculate the average leads generated on weekdays versus weekends for each associate. Are there any notable differences in performance based on the day of the week?

```
# Add day type
cleaned_df['Day_Type'] = cleaned_df['Day'].apply(lambda x: 'Weekend' if x in ['Sat', 'Sun'] else 'Weekday')

day_analysis = cleaned_df.groupby(['Employee_Name', 'Day_Type'])['Leads'].mean().unstack()

print("\nAverage Leads on Weekdays vs Weekends:")
print(day_analysis.round(2))
```

```
Average Leads on Weekdays vs Weekends:
Day_Type       Weekday  Weekend
Employee_Name
Ali              11.90      NaN
Arya             11.56      NaN
Raj              10.77     16.0
<ipython-input-51-446a020c626a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  cleaned_df['Day_Type'] = cleaned_df['Day'].apply(lambda x: 'Weekend' if x in ['Sat', 'Sun'] else 'Weekday')
```

Insight:

Raj performs significantly better on weekends, possibly due to fewer distractions or different strategy.

Arya and Ali have no weekend data, meaning they might not have worked on weekends or data wasn't collected.

10. Predictive Analysis: Using a simple linear regression model, predict the number of leads each associate is expected to generate based on their time spent on lead generation. How accurate is the model when compared to actual data?

```
predictions = {}

for emp in cleaned_df['Employee_Name'].unique():
    emp_data = cleaned_df[cleaned_df['Employee_Name'] == emp].copy()

    X = emp_data[['Time spent on LG (mins)']]
    y = emp_data['Leads']

    # Normalize the feature
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Fit the model
    model = LinearRegression()
    model.fit(X_scaled, y)

    y_pred = model.predict(X_scaled)

    r2 = r2_score(y, y_pred)
    mse = mean_squared_error(y, y_pred)

    predictions[emp] = {
        'R²': round(r2, 3),
        'MSE': round(mse, 2),
        'Intercept': round(model.intercept_, 2),
        'Slope': round(model.coef_[0], 2)
    }

# Display summary
print("\nLead Prediction Model Summary (per employee) [with Normalization]:")
```

```
for emp, metrics in predictions.items():
    print(f"{emp}: {metrics}")
```

```
Lead Prediction Model Summary (per employee) [with Normalization]:
Raj: {'R²': 0.426, 'MSE': 4.04, 'Intercept': np.float64(10.9), 'Slope': np.float64(1.73)}
Arya: {'R²': 0.1, 'MSE': 4.09, 'Intercept': np.float64(11.56), 'Slope': np.float64(0.67)}
Ali: {'R²': 0.272, 'MSE': 6.56, 'Intercept': np.float64(11.9), 'Slope': np.float64(1.57)}
```

Insight:

1. R² (R-squared) shows how well time spent explains lead count (closer to 1 = better).

Raj's model is the most accurate.

Arya's low R² means her performance isn't well explained by time alone (maybe quality > quantity).

2. Slope tells you how many leads increase per additional minute.

Raj gains ~1.73 leads per extra hour.

3. MSE (Mean Squared Error) shows average prediction error. Lower is better.

Conclusion:

Employee-Level Performance Overview

Arya:

Most consistent performer (lowest coefficient of variation – CV = 0.187).

Requires least time (91.3 mins) to achieve optimal lead generation.

Weak correlation between time spent and leads (r = -0.51), suggesting efficiency or other factors (like call quality) play a role.

No weekend performance data available.

Raj:

Shows better performance on weekends (avg. 16 leads vs 10.77 on weekdays).

Moderate consistency (CV = 0.247).

Stronger correlation between time spent and leads (r = -0.34).

Requires around 337.5 mins to reach optimal productivity.

Best linear regression model for predicting performance (R² = 0.426).

Ali:

Highest variability in daily lead performance (std dev = 3.04, CV = 0.255).

Needs around 240 mins to reach productive levels.

Moderate correlation (r = -0.36) between time and leads.

No weekend data, but performs consistently on weekdays.

General Observations Across All Employees

Time spent on lead generation is positively related to performance, but not always linearly—each associate has their own threshold.

On days when Daily Team Reviews were missed, lead generation dropped slightly:

Raj: ↓ 8.45%

Ali: ↓ 7.76%

Arya: No missed days.

On top 10% performance days, employees spent more time than average:

This shows time investment is linked to peak performance.

Incomplete Leads are decreasing over time for Raj and Ali, but slightly increasing for Arya (may require attention).

Variation exists in how effective each employee is at converting time into leads:

This could guide training focus or workflow optimization.

Assumptions for Next Month Projections Based on the Model

Let’s assume:

22 working days in the month

Average daily time spent by each associate:

Raj: 300 mins

Arya: 180 mins

Ali: 250 mins

Summary

Raj is expected to lead in monthly performance (~430 leads).

Ali will closely follow (~406 leads).

Arya, while most consistent, may have slightly lower output due to fewer hours (~299 leads).

These are estimates, assuming current behavior and time allocation trends remain stable.

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit