```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

Data Understanding

```
df_time = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Time spent Analysis")
df_key = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Key Accounts")
df_meet = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Meeting Log")
df_inquiry = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Inquiry Tracker")

```

```
df_time.describe()
df_time.info()
df_time.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142 entries, 0 to 141
Data columns (total 6 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Company                      142 non-null    object 
 1   Category                     142 non-null    object 
 2   Total Hours                  142 non-null    float64
 3   Estimated Cost               142 non-null    int64  
 4   3500                         0 non-null      float64
 5   inr per hour is cost assumed  0 non-null      float64
dtypes: float64(3), int64(1), object(2)
memory usage: 6.8+ KB
```

	Company	Category	Total Hours	Estimated Cost	3500	inr per hour is cost assumed
0	Chennai	Unknown	32.00	112000	NaN	NaN
1	Pidilite	Distributor	23.75	83125	NaN	NaN
2	Kalpataru Projects	Contractor	23.50	82250	NaN	NaN
3	Asg win	Applicator	15.50	54250	NaN	NaN
4	Internal/Review (MRM, Hubspot, etc.)	Admin	12.50	43750	NaN	NaN

Next steps: Generate code with df_time | New interactive sheet

```
df_key.head(10)
df_key.info()
df_key.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 6 columns):
 #   Column                                Non-Null Count  Dtype 
---  ------                                --------------  ----- 
 0   Location                              42 non-null     object
 1   Key Account Names (company name)      42 non-null     object
 2   Types of Projects                     42 non-null     object
 3   Number of Decision Makers/ Sub Teams  41 non-null     object
 4   Estimated Vol of Boq Per Year with EJC  42 non-null     object
 5   Target number of Boq from the account  42 non-null     int64 
dtypes: int64(1), object(5)
memory usage: 2.1+ KB
```

	Target number of Boq from the account
count	42.000000
mean	2.500000
std	2.520646
min	1.000000
25%	1.000000
50%	1.000000
75%	3.000000
max	13.000000

```
df_meet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 468 entries, 0 to 467
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date of Activity          467 non-null    object
 1   Activity Type             159 non-null    object
 2   Company                   466 non-null    object
 3   Project                   450 non-null    object
 4   Hours spent               447 non-null    object
 5   Accomplishment / Summary  404 non-null    object
 6   Remarks                   301 non-null    object
 7   Unnamed: 7                0 non-null      float64
 8   Unnamed: 8                0 non-null      float64
 9   Unnamed: 9                2 non-null      object
 10  Unnamed: 10               12 non-null     object
 11  Unnamed: 11               2 non-null      object
 12  Unnamed: 12               1 non-null      object
dtypes: float64(2), object(11)
memory usage: 47.7+ KB
```

```
df_inquiry.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Inquiry Date           26 non-null     object
 1   Projects               25 non-null     object
 2   Number of Hours        15 non-null     float64
 3   Value/Hours            8 non-null      float64
 4   Feasability Ratio      0 non-null      float64
 5   Expected Close date    25 non-null     object
 6   Unnamed: 6             0 non-null      float64
 7   Construction type      25 non-null     object
 8   Stage                  25 non-null     object
 9   Source                 25 non-null     object
 10  Quantity in LM         9 non-null      object
 11  Value ($)              26 non-null     float64
 12  Value (INR)            26 non-null     float64
 13  Quoted Via Applicator  25 non-null     object
 14  Applicator             23 non-null     object
 15  Lost Reason            0 non-null      float64
 16  Unnamed: 16            0 non-null      float64
 17  Unnamed: 17            2 non-null      object
 18  Unnamed: 18            7 non-null      object
 19  Unnamed: 19            7 non-null      float64
 20  Unnamed: 20            2 non-null      object
 21  Unnamed: 21            2 non-null      object
dtypes: float64(9), object(13)
memory usage: 5.3+ KB
```

Data Cleaning

```
def normalize_name(s):
  if pd.isna(s):
    return s
  s = str(s).lower().replace("&", " and ")
  s = re.sub(r'[\(\).,;:/\-]', ' ', s)
  # collapse multiple spaces into one
  s = re.sub(r'\s+', ' ', s).strip()
  return s
```

```
#Here we have to clean the data of the first sheet that is Time spent analysis.
#Here we have to clean the data of the first sheet that is Time spent analysis.
df_time = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Time spent Analysis")
df_time = df_time.dropna(axis=1, how='all')
df_time.columns = ['company', 'category', 'total_hours', 'estimated_cost']

df_time['category']=df_time['category'].str.strip().str.lower().str.replace(r'\s+', ' ', regex=True)
df_time['company']=df_time['company'].str.strip().str.lower().str.replace(r'\s+', ' ', regex=True)

category_mapping = {
    'consultant/architect': 'Consultant',
    'developer': 'Developer',
    'developers/projects': 'Developer',
    'developers/office/projects': 'Developer',
    'developers/groups': 'Developer',
    'developers/group': 'Developer',
    'estates/developers': 'Developer',
```

```
    'distributor': 'Distributor',
    'contractor': 'Contractor',
    'applicator': 'Applicator',
    'admin': 'Admin',
    'unknown': 'Unknown'
}

df_time['category'] = df_time['category'].map(category_mapping).fillna(df_time['category'])

# Calculate cost per hour
df_time['cost_per_hour'] = df_time['estimated_cost'] / df_time['total_hours']

# Apply the normalize function to the column
df_time['company'] = df_time['company'].apply(normalize_name)
df_time['category'] = df_time['category'].apply(normalize_name)
```

```
# Cleaning of the second sheet
df_key = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Key Accounts")

df_key_Raw = df_key.copy()
rename_columns = {
      'Key Account Names (company name)': 'company_name',
    'Types of Projects': 'types_of_projects',
    'Number of Decision Makers/ Sub Teams': 'num_decision_makers',
    'Estimated Vol of Boq Per Year with EJC': 'estimated_boq_per_year',
    'Target number of Boq from the account': 'target_boq'
}
df_key = df_key.rename(columns=rename_columns)

def normalize_to_lower(s):
  if pd.isna(s):
    return s
  # Moved the processing lines outside the 'if' block so they are always executed for non-NaN values
  s = str(s).lower().replace('or',' ').replace('to',' ').strip()
  s = re.sub(r'\s+', ' ', s).strip()

  match = re.search(r'(\d+)', s)
  if match:
      return int(match.group(1))
  return s


df_key['estimated_boq_per_year'] = df_key['estimated_boq_per_year'].apply(normalize_to_lower)
df_key['target_boq'] = df_key['target_boq'].apply(normalize_to_lower)

df_key['estimated_boq_int'] = pd.to_numeric(df_key['estimated_boq_per_year'], errors='coerce')
df_key['target_boq'] = pd.to_numeric(df_key['target_boq'], errors='coerce').astype('Int64')

df_key['account_norm'] = df_key['company_name'].apply(normalize_name)
```

```
df_key[['company_name', 'account_norm', 'estimated_boq_per_year', 'estimated_boq_int']].head(10)

```

	company_name	account_norm	estimated_boq_per_year	estimated_boq_int
0	AECOM	aecom	5	5
1	AEDBM	aedbm	1	1
2	Aedium Designs	aedium designs	2	2
3	Archetype	archetype	3	3
4	Ashok Narayan & Eshwar	ashok narayan and eshwar	5	5
5	Base	base	1	1
6	Besten Consultants	besten consultants	1	1
7	Bureau Engineers (structural)	bureau engineers structural	5	5
8	Bureau Hepolt	bureau hepolt	1	1
9	Chandavarkar and Thacker Architects	chandavarkar and thacker architects	2	2

```
df_meet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 468 entries, 0 to 467
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date of Activity         467 non-null    object
 1   Activity Type            159 non-null    object
```

```
 2   Company                  466 non-null    object
 3   Project                  450 non-null    object
 4   Hours spent              447 non-null    object
 5   Accomplishment / Summary  404 non-null    object
 6   Remarks                  301 non-null    object
 7   Unnamed: 7               0 non-null      float64
 8   Unnamed: 8               0 non-null      float64
 9   Unnamed: 9               2 non-null      object
 10  Unnamed: 10              12 non-null     object
 11  Unnamed: 11              2 non-null      object
 12  Unnamed: 12              1 non-null      object
dtypes: float64(2), object(11)
memory usage: 47.7+ KB
```

```
df_meet = pd.read_excel("Sample Month End Report for Assesment.xlsx", sheet_name="Meeting Log")

df_meet_raw = df_meet.copy()
df_meet = df_meet.dropna(axis=1, how='all')
df_meet = df_meet.loc[:, ~df_meet.columns.str.contains('^Unnamed')]

keep_cols = [
    'Date of Activity',
    'Activity Type',
    'Company',
    'Project',
    'Hours spent',
    'Accomplishment / Summary',
    'Remarks'
]

df_meet = df_meet[keep_cols]

df_meet['Date of Activity'] = pd.to_datetime(df_meet['Date of Activity'], errors='coerce')


df_meet['Hours spent'] = pd.to_numeric(df_meet['Hours spent'], errors='coerce')

df_meet = df_meet[~df_meet['Project'].isna()]

df_meet['Activity Type'] = df_meet['Activity Type'].fillna('Unknown')

df_meet['project_norm'] = df_meet['Project'].apply(normalize_name)
df_meet['company_norm'] = df_meet['Company'].apply(normalize_name)

```

```
#SHEET-4 CLEANING


df_inquiry = pd.read_excel("Sample Month End Report for Assesment.xlsx",
                           sheet_name="Inquiry Tracker")

df_inquiry_raw = df_inquiry.copy()  # backup


# ---------- 3. Remove totally empty columns + header junk rows ----------
# Drop completely empty columns
df_inquiry = df_inquiry.dropna(axis=1, how='all')

# Drop first 2 rows if they are header/notes (as you observed earlier)
df_inquiry = df_inquiry.iloc[2:].reset_index(drop=True)

# Drop Unnamed columns (Excel artifacts)
df_inquiry = df_inquiry.loc[:, ~df_inquiry.columns.str.contains('^Unnamed')]


# ---------- 4. Quick check of columns (just to see real names) ----------
print("Columns in df_inquiry:")
print(df_inquiry.columns)


# ---------- 5. Convert dates ----------
# Inquiry Date
if 'Inquiry Date' in df_inquiry.columns:
    df_inquiry['Inquiry Date'] = pd.to_datetime(df_inquiry['Inquiry Date'], errors='coerce')


# ---------- 6. Clean Expected Close date ----------
def clean_expected_close(v):
    """
    Handle values like:
    - '2026' -> 2026-12-31
    - 'oct 2026' -> 2026-10-01
```

```
    - 'project on hold' -> NaT, on_hold=True
    """
    if pd.isna(v):
        return pd.NaT, False

    s = str(v).strip().lower()

    # On hold / no date
    if 'hold' in s:
        return pd.NaT, True

    # Pure year '2026'
    if re.fullmatch(r'\d{4}', s):
        year = int(s)
        return pd.Timestamp(year=year, month=12, day=31), False

    # Month + year like 'oct 2026'
    try:
        parts = s.split()
        if len(parts) == 2 and re.search(r'[a-zA-Z]', parts[0]) and re.fullmatch(r'\d{4}', parts[1]):
            s2 = '1 ' + s  # assume first day of month
            dt = pd.to_datetime(s2, errors='coerce', dayfirst=True)
        else:
            dt = pd.to_datetime(s, errors='coerce', dayfirst=True)
    except Exception:
        dt = pd.NaT

    return dt, False

if 'Expected Close date' in df_inquiry.columns:
    tmp = df_inquiry['Expected Close date'].apply(clean_expected_close)
    df_inquiry['expected_close_clean'] = tmp.apply(lambda x: x[0])
    df_inquiry['on_hold_flag'] = tmp.apply(lambda x: x[1])
else:
    df_inquiry['expected_close_clean'] = pd.NaT
    df_inquiry['on_hold_flag'] = False


# ---------- 7. Convert numeric columns ----------
numeric_cols = [
    'Number of Hours',
    'Value/Hours',
    'Feasability Ratio',
    'Quantity in LM',
    'Value ($)',
    'Value (INR)'
]

for col in numeric_cols:
    if col in df_inquiry.columns:
        df_inquiry[col] = pd.to_numeric(df_inquiry[col], errors='coerce')


# ---------- 8. Basic missing value handling ----------
# Drop rows with missing Projects (no project = not useful for project-level analysis)
if 'Projects' in df_inquiry.columns:
    df_inquiry = df_inquiry[~df_inquiry['Projects'].isna()]

# Fill Stage nulls with 'Unknown'
if 'Stage' in df_inquiry.columns:
    df_inquiry['Stage'] = df_inquiry['Stage'].fillna('Unknown')

# If you have a Construction type column, fill nulls
for col in df_inquiry.columns:
    if col.strip().lower() == 'construction type':
        df_inquiry[col] = df_inquiry[col].fillna('Unknown')


# ---------- 9. Normalize project name for joining ----------
if 'Projects' in df_inquiry.columns:
    df_inquiry['project_norm'] = df_inquiry['Projects'].apply(normalize_name)
else:
    df_inquiry['project_norm'] = pd.NA

```

```
Columns in df_inquiry:
Index(['Inquiry Date', 'Projects', 'Number of Hours', 'Value/Hours',
       'Expected Close date', 'Construction type', 'Stage', 'Source',
       'Quantity in LM', 'Value ($)', 'Value (INR)', 'Quoted Via Applicator',
       'Applicator'],
      dtype='object')
```

Data Analysis

```
# ============================
# SIMPLE KPIs — SHEET 1 (TIME SPENT)
# ============================

# KPI 1: Companies consuming the most hours
kpi_hours_by_company = (
    df_time.groupby('company')['total_hours']
    .sum()
    .sort_values(ascending=False)
)
print("KPI 1: Hours by company")
print(kpi_hours_by_company.head(10), "\n")


# KPI 2: Per-hour return per company
df_time['return_per_hour'] = df_time['estimated_cost'] / df_time['total_hours']
kpi_return_per_hour = (
    df_time.groupby('company')['return_per_hour']
    .mean()
    .sort_values(ascending=False)
)
print("KPI 2: Return per hour by company")
print(kpi_return_per_hour.head(10), "\n")


# KPI 3: Hours by category
kpi_hours_by_category = (
    df_time.groupby('category')['total_hours']
    .sum()
    .sort_values(ascending=False)
)
print("KPI 3: Hours by category")
print(kpi_hours_by_category, "\n")


# KPI 4: Revenue by category
kpi_revenue_by_category = (
    df_time.groupby('category')['estimated_cost']
    .sum()
    .sort_values(ascending=False)
)
print("KPI 4: Revenue by category")
print(kpi_revenue_by_category, "\n")


# KPI 5: Number of entries per company
kpi_entries_by_company = df_time['company'].value_counts()
print("KPI 5: Entry count per company")
print(kpi_entries_by_company.head(10), "\n")


# KPI 6: Revenue per company
kpi_revenue_by_company = (
    df_time.groupby('company')['estimated_cost']
    .sum()
    .sort_values(ascending=False)
)
print("KPI 6: Revenue by company")
print(kpi_revenue_by_company.head(10), "\n")

# KPI 7
avg_cost_per_hour = df_time['cost_per_hour'].mean()
df_time['CEI'] = df_time['cost_per_hour'] / avg_cost_per_hour

kpi_cei = df_time[['company','cost_per_hour','CEI']].sort_values('CEI', ascending=False)
print("KPI 7: Cost Efficiency Index")
print(kpi_cei.head(10))

#KPI 8
total_hours = df_time['total_hours'].sum()
pareto_hours_pct = (df_time.groupby('category')['total_hours'].sum() / total_hours) * 100

print("\nKPI 8: Pareto Distribution")
print(pareto_hours_pct.sort_values(ascending=False))

#KPI 9

total_revenue = df_time['estimated_cost'].sum()
category_revenue_pct = (df_time.groupby('category')['estimated_cost'].sum() / total_revenue) * 100
```

```
category_revenue_pct = (df_time.groupby('category')['estimated_cost'].sum() / total_revenue) * 100

print("\nKPI 9: Revenue Distribution")
print(category_revenue_pct.sort_values(ascending=False))

#kpi10
df_time['efficiency'] = df_time['estimated_cost'] / df_time['total_hours']

high_hours = df_time['total_hours'].quantile(0.75)
low_eff = df_time['efficiency'].quantile(0.25)

time_draining_accounts = df_time[
    (df_time['total_hours'] >= high_hours) &
    (df_time['efficiency'] <= low_eff)
][['company','total_hours','estimated_cost','efficiency']]

print("\nKPI 10: Time Draining Accounts")
print(time_draining_accounts)
```

```
                         assetz home         7.25           25375
                brigade and prestige         7.00           24500
                   sriram properties         7.00           24500
                             innotech         6.50           22750
                symmetric architects         6.50           22750
                                 bps         6.25           21875
                       rmz developers         6.00           21000
              snn estates developers          6.00           21000
             sattva developers groups         5.75           20125
                    abb india pvt ltd         5.25           18375
                      prestige groups         5.25           18375
                              optimus         5.00           17500
                  dhruva engineering          4.50           15750
shapoorji and pallonji constructions          4.50           15750
                  sundaram architects         4.50           15750
                         dsp designs          4.00           14000
      hubspot discussion with jayanth         4.00           14000
                     inform architects         4.00           14000
                        magcor pvt ltd         4.00           14000
                     somerset pharma           4.00           14000

    efficiency
0       3500.0
1       3500.0
2       3500.0
3       3500.0
4       3500.0
5       3500.0
6       3500.0
7       3500.0
8       3500.0
9       3500.0
10      3500.0
11      3500.0
12      3500.0
13      3500.0
14      3500.0
15      3500.0
16      3500.0
17      3500.0
18      3500.0
19      3500.0
20      3500.0
21      3500.0
22      3500.0
23      3500.0
24      3500.0
25      3500.0
26      3500.0
27      3500.0
28      3500.0
29      3500.0
30      3500.0
31      3500.0
32      3500.0
33      3500.0
34      3500.0
35      3500.0
```

```
# ======================================
# SHEET 2 — KEY ACCOUNTS KPI ANALYSIS
# ======================================

# Ensure 'num_decision_makers' is numeric before calculations
df_key['num_decision_makers'] = df_key['num_decision_makers'].apply(normalize_to_lower)
df_key['num_decision_makers'] = pd.to_numeric(df_key['num_decision_makers'], errors='coerce').fillna(0).astype(int)

```

```
# KPI 1 — Estimated BOQ per account (highest potential accounts)
kpi_estimated_boq = (
    df_key.groupby('company_name')['estimated_boq_int']
    .sum()
    .sort_values(ascending=False)
)
print("\nKPI 1 — Estimated BOQ per account:")
print(kpi_estimated_boq.head(10))


# KPI 2 — Target BOQ per account (highest assigned goals)
kpi_target_boq = (
    df_key.groupby('company_name')['target_boq']
    .sum()
    .sort_values(ascending=False)
)
print("\nKPI 2 — Target BOQ per account:")
print(kpi_target_boq.head(10))


# KPI 3 — BOQ Gap (Estimated - Target)
df_key['gap_boq'] = df_key['estimated_boq_int'] - df_key['target_boq']

kpi_gap_boq = (
    df_key.groupby('company_name')['gap_boq']
    .mean()
    .sort_values(ascending=False)
)
print("\nKPI 3 — BOQ Gap (Estimated - Target):")
print(kpi_gap_boq.head(10))


# KPI 4 — Complexity Score (Estimated BOQ × # decision makers)
df_key['opportunity_index'] = (
    df_key['estimated_boq_int'] / df_key['num_decision_makers']
)
df_key['complexity'] = df_key['estimated_boq_int'] * df_key['num_decision_makers']
df_key['opportunity_index'].replace([np.inf, -np.inf], np.nan, inplace=True)

kpi_complexity = (
    df_key.groupby('company_name')['complexity']
    .sum()
    .sort_values(ascending=False)
)
print("\nKPI 4 — Complexity Score:")
print(kpi_complexity.head(10))


# KPI 5 — Opportunity Index (Estimated BOQ / # decision makers)


kpi_opportunity = (
    df_key.groupby('company_name')['opportunity_index']
    .mean()
    .sort_values(ascending=False)
)
print("\nKPI 5 — Opportunity Index:")
print(kpi_opportunity.head(10))


# KPI 6 — BOQ by project type
kpi_project_type = (
    df_key.groupby('types_of_projects')['estimated_boq_int']
    .sum()
    .sort_values(ascending=False)
)
print("\nKPI 6 — BOQ by Project Type:")
print(kpi_project_type)


# KPI 7 — Tier Classification (High / Medium / Low potential accounts)
boq = df_key.groupby('company_name')['estimated_boq_int'].sum()

high_potential = boq.quantile(0.75)
low_potential = boq.quantile(0.25)

df_segments = boq.apply(
    lambda x: 'High' if x >= high_potential else
              ('Low' if x <= low_potential else 'Medium')
)

```

```
print("\nKPI 7 — Account Potential Tiers:")
print(df_segments.head(20))
```

```
KPI 1 — Estimated BOQ per account:
company_name
Design Tree                       15
CRN Architects                    10
Innotech                          10
Ashok Narayan & Eshwar             5
Bureau Engineers (structural)      5
AECOM                              5
RSB Consultants                    5
Natraj And Venkat.                 5
Archetype                          3
Jayam Consultants                  3
Name: estimated_boq_int, dtype: int64

KPI 2 — Target BOQ per account:
company_name
Design Tree                       13
Innotech                           9
CRN Architects                     8
Ashok Narayan & Eshwar             5
Bureau Engineers (structural)      5
AECOM                              5
RSB Consultants                    5
Jayam Consultants                  3
Archetype                          3
Natraj And Venkat.                 3
Name: target_boq, dtype: Int64

KPI 3 — BOQ Gap (Estimated - Target):
company_name
CRN Architects                          2.0
Design Tree                             2.0
Natraj And Venkat.                      2.0
Innotech                                1.0
Ducon                                   1.0
Geostructural                           1.0
Chandavarkar and Thacker Architects     1.0
Gensler.                                1.0
Murthy and Manyam.                      1.0
Nadig consultants                       1.0
Name: gap_boq, dtype: Float64

KPI 4 — Complexity Score:
company_name
Design Tree                       1050
Bureau Engineers (structural)       80
AECOM                               75
Ashok Narayan & Eshwar              60
RSB Consultants                     40
Dius Consultants                    36
Innotech                            30
CRN Architects                      30
cheralathan Architect               30
Archetype                           21
Name: complexity, dtype: int64
```

```
# ==============================
# SHEET 3 — MEETING LOG KPIs

# safety checks
required = ['Date of Activity', 'Activity Type', 'Company', 'Project', 'Hours spent']
missing = [c for c in required if c not in df_meet.columns]
if missing:
    raise KeyError(f"Missing columns in df_meet: {missing}")

# DATE CHECKING
df_meet['Date of Activity'] = pd.to_datetime(df_meet['Date of Activity'], errors='coerce')

# KPI M1: Total hours per company
hours_by_company = df_meet.groupby('Company')['Hours spent'].sum().sort_values(ascending=False)
print("\nKPI M1 — Total hours by company (top 10):")
print(hours_by_company.head(10))

# KPI M2: Total hours per project
hours_by_project = df_meet.groupby('Project')['Hours spent'].sum().sort_values(ascending=False)
print("\nKPI M2 — Total hours by project (top 10):")
print(hours_by_project.head(10))

# KPI M3: Total hours per activity type
hours_by_activity = df_meet.groupby('Activity Type')['Hours spent'].sum().sort_values(ascending=False)
print("\nKPI M3 — Total hours by activity type:")
```

```
print(hours_by_activity)

# KPI M4: Number of meetings per company
meetings_by_company = df_meet.groupby('Company').size().sort_values(ascending=False)
print("\nKPI M4 — Number of meetings per company (top 10):")
print(meetings_by_company.head(10))

# KPI M5: Number of meetings per project
meetings_by_project = df_meet.groupby('Project').size().sort_values(ascending=False)
print("\nKPI M5 — Number of meetings per project (top 10):")
print(meetings_by_project.head(10))

# KPI A (advanced): Average hours per meeting type
avg_hours_by_activity = df_meet.groupby('Activity Type')['Hours spent'].mean().sort_values(ascending=False)
print("\nKPI A — Average hours per activity type:")
print(avg_hours_by_activity)

# KPI B (advanced): Meeting intensity per company (hours / meetings)
hours = df_meet.groupby('Company')['Hours spent'].sum()
counts = df_meet.groupby('Company').size()
meeting_intensity = (hours / counts).sort_values(ascending=False)
print("\nKPI B — Meeting intensity (avg hours per meeting) per company (top 10):")
print(meeting_intensity.head(10))

# KPI C: Meetings & hours by month (trend)
df_meet['month'] = df_meet['Date of Activity'].dt.to_period('M')
meetings_per_month = df_meet.groupby('month').size().sort_index()
hours_per_month = df_meet.groupby('month')['Hours spent'].sum().sort_index()
print("\nKPI C — Meetings per month (chronological):")
print(meetings_per_month)
print("\nKPI C — Hours per month (chronological):")
print(hours_per_month)

# End of Sheet 3 KPIs

```

```
KPI M1 — Total hours by company (top 10):
Company
Chennai                                     32.00
Pidilite                                    22.75
Bagmane Developers                          18.25
Kalpataru projects International Limited    17.75
Hyderabad Travel                            16.00
Design Tree                                 13.25
CRN Architects                              12.75
Birla Groups                                12.00
Innotech                                    10.75
Colliers                                    10.40
Name: Hours spent, dtype: float64

KPI M2 — Total hours by project (top 10):
Project
Multiple projects    45.25
Multiple Project     32.00
Multiple Projects    18.75
Event                16.00
Chennai              16.00
Pidilite             15.00
Birla Tisya          14.00
Sap Labs             12.00
MRM                  11.00
Somerset pharma       8.50
Name: Hours spent, dtype: float64

KPI M3 — Total hours by activity type:
Activity Type
Unknown                          448.65
Introduction pitch                57.00
Techno commercial                 42.00
Site Visit                        32.25
Techno - Commercial               22.75
Internal team meeting             21.75
Techno - commercial               21.75
Site Visit                        11.00
Introduction pitch                10.00
Introduction Pitch                 8.50
Technical presentation             8.00
Admin                              2.25
Applicator Meeting                 2.00
Group Presntation                  2.00
Applicator Techno Commercial       1.50
Internal meeting                   1.25
Site visit                         1.00
Activity Type                      0.00
Leave                              0.00
Festival - Diwali                  0.00
```

```
Name: Hours spent, dtype: float64

KPI M4 — Number of meetings per company (top 10):
Company
Kalpataru projects International Limited    12
Bagmane Developers                          11
Birla Groups                                10
```

```
# ==============================
# SHEET 4 — INQUIRY TRACKER KPIs


# safety checks
required = ['Inquiry Date', 'Projects', 'Stage', 'Value (INR)']
missing = [c for c in required if c not in df_inquiry.columns]
if missing:
    raise KeyError(f"Missing required columns in df_inquiry: {missing}")

# Ensure dates and numeric types
df_inquiry['Inquiry Date'] = pd.to_datetime(df_inquiry['Inquiry Date'], errors='coerce')
if 'expected_close_clean' not in df_inquiry.columns:
    if 'Expected Close date' in df_inquiry.columns:
        df_inquiry['expected_close_clean'] = pd.to_datetime(df_inquiry['Expected Close date'], errors='coerce')
    else:
        df_inquiry['expected_close_clean'] = pd.NaT

df_inquiry['Value (INR)'] = pd.to_numeric(df_inquiry['Value (INR)'], errors='coerce')

# KPI I1: Inquiry count per Stage
inquiries_by_stage = df_inquiry['Stage'].value_counts()
inquiries_by_stage_pct = df_inquiry['Stage'].value_counts(normalize=True) * 100
print("\nKPI I1 — Inquiry count by Stage:")
print(inquiries_by_stage)
print("\nKPI I1 — Inquiry distribution (%) by Stage:")
print(inquiries_by_stage_pct)

# KPI I2: Inquiry count per Construction type
if 'Construction type' in df_inquiry.columns:
    inquiries_by_construction = df_inquiry['Construction type'].value_counts()
    print("\nKPI I2 — Inquiry count by Construction type:")
    print(inquiries_by_construction)
else:
    print("\nKPI I2 — 'Construction type' column not found.")

# KPI I3: Total Value (INR) per Stage
value_by_stage = df_inquiry.groupby('Stage')['Value (INR)'].sum().sort_values(ascending=False)
print("\nKPI I3 — Total Value (INR) by Stage:")
print(value_by_stage)

# KPI I4: Inquiry volume per month & KPI I5: Inquiry value per month
df_inquiry['month'] = df_inquiry['Inquiry Date'].dt.to_period('M')
inquiries_per_month = df_inquiry.groupby('month').size().sort_index()
value_per_month = df_inquiry.groupby('month')['Value (INR)'].sum().sort_index()
print("\nKPI I4 — Inquiries per month (chronological):")
print(inquiries_per_month)
print("\nKPI I5 — Value (INR) per month (chronological):")
print(value_per_month)

# KPI I6: Stage conversion - counts and percent
stage_counts = df_inquiry['Stage'].value_counts()
stage_percent = df_inquiry['Stage'].value_counts(normalize=True) * 100
print("\nKPI I6 — Stage funnel counts:")
print(stage_counts)
print("\nKPI I6 — Stage funnel %:")
print(stage_percent)

# KPI I7: Mean lead duration (expected_close_clean - Inquiry Date)
df_inquiry['lead_duration_days'] = (df_inquiry['expected_close_clean'] - df_inquiry['Inquiry Date']).dt.days
mean_lead_duration = df_inquiry['lead_duration_days'].mean()
print("\nKPI I7 — Mean lead duration (days):")
print(mean_lead_duration)

# KPI I8: High-value inquiries stuck in early stages (top 25% by Value)
early_stages = ['New', 'Feasibility', 'Commercial Review']  # adjust if needed
value_series = df_inquiry['Value (INR)'].dropna()
if len(value_series)>0:
    high_value_threshold = value_series.quantile(0.75)
else:
    high_value_threshold = np.nan

mask_stage = df_inquiry['Stage'].isin(early_stages)
mask_value = df_inquiry['Value (INR)'] >= high_value_threshold
df_high_stuck = df_inquiry[mask_stage & mask_value].copy()
```

```
print("\nKPI I8 — High-value inquiries stuck in early stages (summary):")
print("High-value threshold (75th percentile):", high_value_threshold)
print("Count high-value & early-stage:", len(df_high_stuck))
print("Total value stuck (INR):", df_high_stuck['Value (INR)'].sum())

print("\nTop high-value stuck rows (if any):")
show_cols = [c for c in ['Inquiry Date','Projects','project_norm','Stage','Value (INR)','Number of Hours','expected_clo
print(df_high_stuck.sort_values('Value (INR)', ascending=False)[show_cols].head(20))

# KPI I10: Overdue high-value inquiries (>60 days)
if 'lead_duration_days' in df_inquiry.columns:
    df_overdue = df_inquiry[(df_inquiry['lead_duration_days'] > 60) & (df_inquiry['Value (INR)'] >= high_value_thresho
    print("\nKPI I10 — Overdue (>60 days) high-value inquiries:")
    print("Count:", len(df_overdue))
    print("Total overdue value (INR):", df_overdue['Value (INR)'].sum())
    show_cols2 = [c for c in ['Inquiry Date','Projects','Stage','Value (INR)','lead_duration_days','expected_close_clea
    print(df_overdue.sort_values('lead_duration_days', ascending=False)[show_cols2].head(20))
else:
    print("\nKPI I10 — lead_duration_days not available; compute KPI I7 first.")

# End of Sheet 4 KPIs

```

```
KPI I1 — Inquiry count by Stage:
Stage
Commercial Review    20
Consultant Review     1
Deal won              1
Technical Review      1
Tender Awarded        1
Name: count, dtype: int64

KPI I1 — Inquiry distribution (%) by Stage:
Stage
Commercial Review    83.333333
Consultant Review     4.166667
Deal won              4.166667
Technical Review      4.166667
Tender Awarded        4.166667
Name: proportion, dtype: float64

KPI I2 — Inquiry count by Construction type:
Construction type
New    24
Name: count, dtype: int64

KPI I3 — Total Value (INR) by Stage:
Stage
Commercial Review    132345000.0
Consultant Review     10200000.0
Deal won               8500000.0
Technical Review       3995000.0
Tender Awarded         3400000.0
Name: Value (INR), dtype: float64

KPI I4 — Inquiries per month (chronological):
month
2024-11    2
2024-12    2
2025-01    2
2025-03    1
2025-04    2
2025-05    3
2025-06    2
2025-07    1
2025-08    2
2025-09    2
2025-10    3
Freq: M, dtype: int64

KPI I5 — Value (INR) per month (chronological):
month
2024-11    12325000.0
2024-12    15895000.0
2025-01    21930000.0
2025-03    10200000.0
2025-04     8585000.0
2025-05    23630000.0
2025-06    10795000.0
2025-07     8500000.0
```

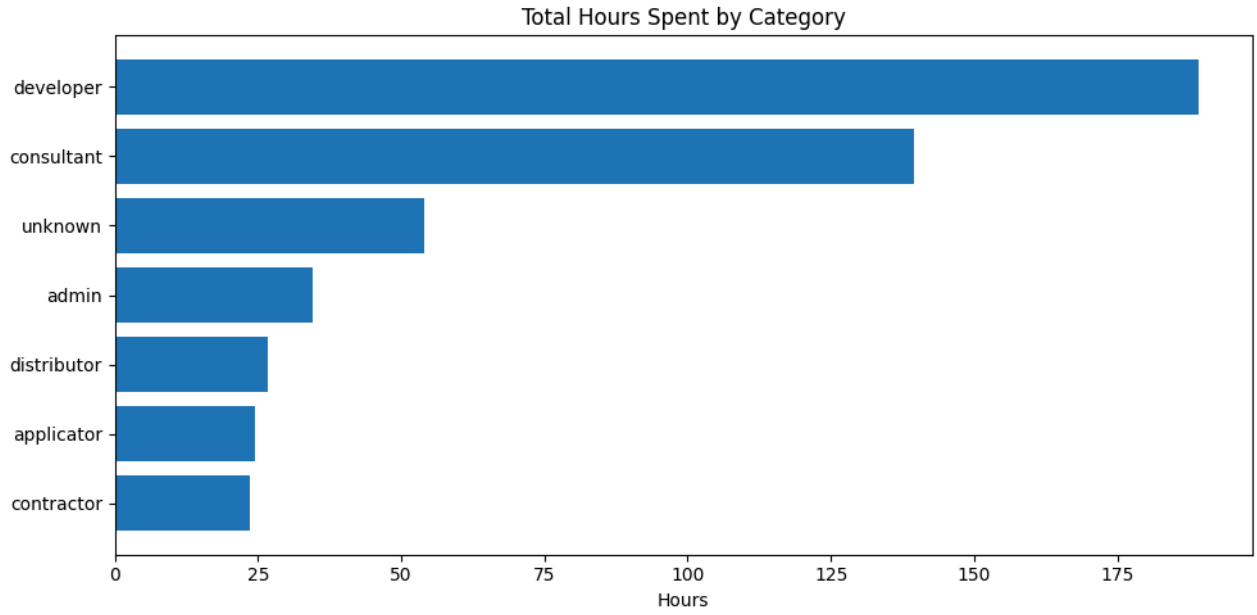
Data Problem

```
# assume existing cleaned DataFrames from your notebook:
# df_time (has 'company'), df_key (has 'account_norm'), df_meet (has 'company_norm','project_norm'), df_inquiry (has 'p

# Basic sizes
print("df_time rows:", len(df_time))
print("df_key rows:", len(df_key))
print("df_meet rows:", len(df_meet))
print("df_inquiry rows:", len(df_inquiry))

# Exact overlap — company-level between df_time and df_key
time_vs_key = pd.merge(
    df_time[['company']].drop_duplicates().rename(columns={'company':'key'}),
    df_key[['account_norm']].drop_duplicates().rename(columns={'account_norm':'key'}),
    on='key',
    how='outer',
    indicator=True
)
print("Company join counts (df_time <-> df_key):")
print(time_vs_key['_merge'].value_counts())

# Exact overlap — company-level between df_time and df_meet
time_vs_meet = pd.merge(
    df_time[['company']].drop_duplicates().rename(columns={'company':'key'}),
    df_meet[['company_norm']].drop_duplicates().rename(columns={'company_norm':'key'}),
    on='key',
    how='outer',
    indicator=True
)
print("Company join counts (df_time <-> df_meet):")
print(time_vs_meet['_merge'].value_counts())

# Exact overlap — project-level between df_meet and df_inquiry
proj_vs_inq = pd.merge(
    df_meet[['project_norm']].drop_duplicates().rename(columns={'project_norm':'key'}),
    df_inquiry[['project_norm']].drop_duplicates().rename(columns={'project_norm':'key'}),
    on='key',
    how='outer',
    indicator=True
)
print("Project join counts (df_meet <-> df_inquiry):")
print(proj_vs_inq['_merge'].value_counts())

```

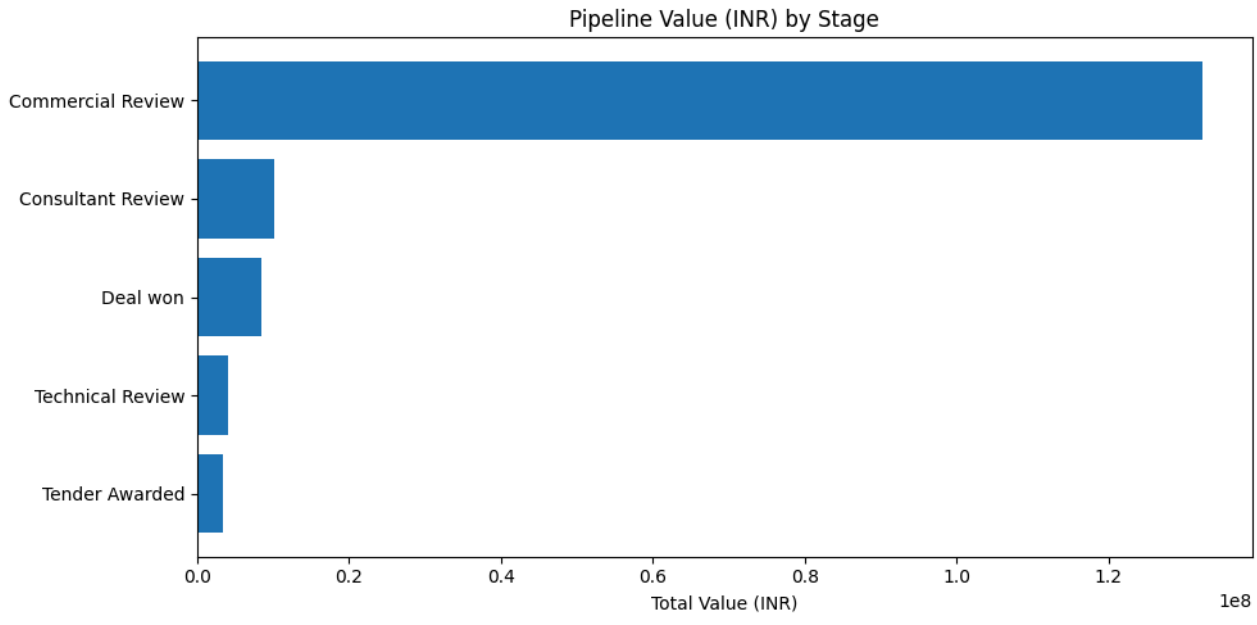
```
df_time rows: 142
df_key rows: 42
df_meet rows: 450
df_inquiry rows: 24
Company join counts (df_time <-> df_key):
_merge
left_only     131
right_only     34
both            8
Name: count, dtype: int64
Company join counts (df_time <-> df_meet):
_merge
both          106
right_only    101
left_only      33
Name: count, dtype: int64
Project join counts (df_meet <-> df_inquiry):
_merge
left_only     254
right_only     19
both            5
Name: count, dtype: int64
```

Data Visualization

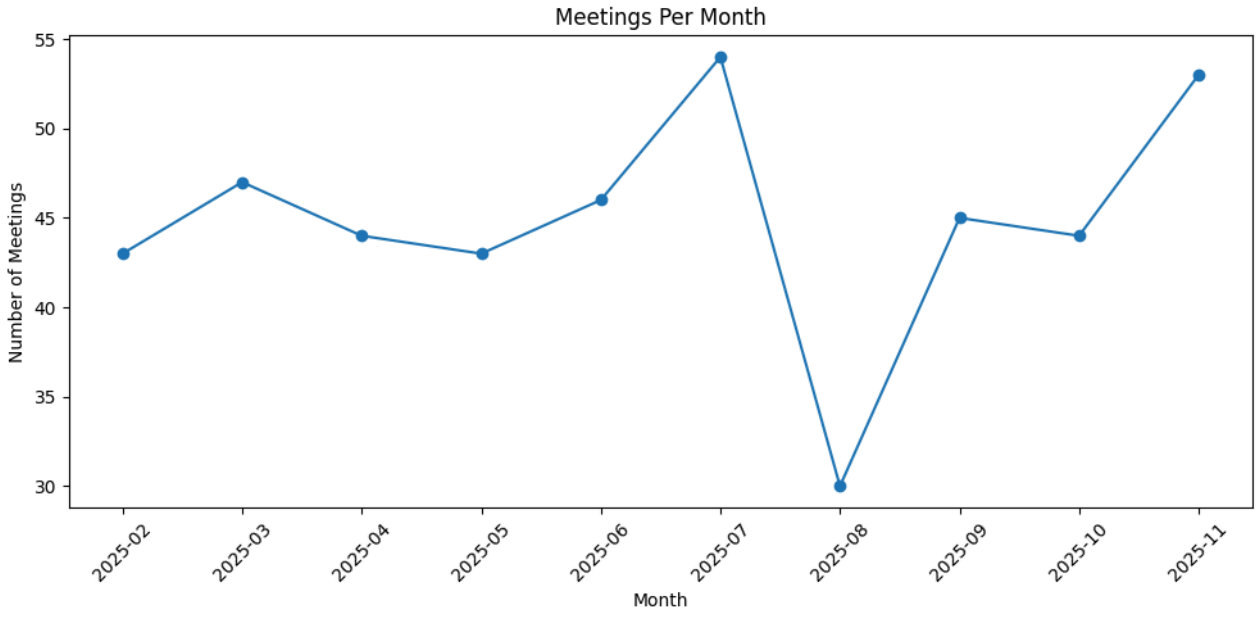
```
import matplotlib.pyplot as plt

category_hours = df_time.groupby('category')['total_hours'].sum().sort_values()

plt.figure(figsize=(10,5))
plt.barh(category_hours.index, category_hours.values)
plt.title("Total Hours Spent by Category")
plt.xlabel("Hours")
plt.tight_layout()
plt.show()

```



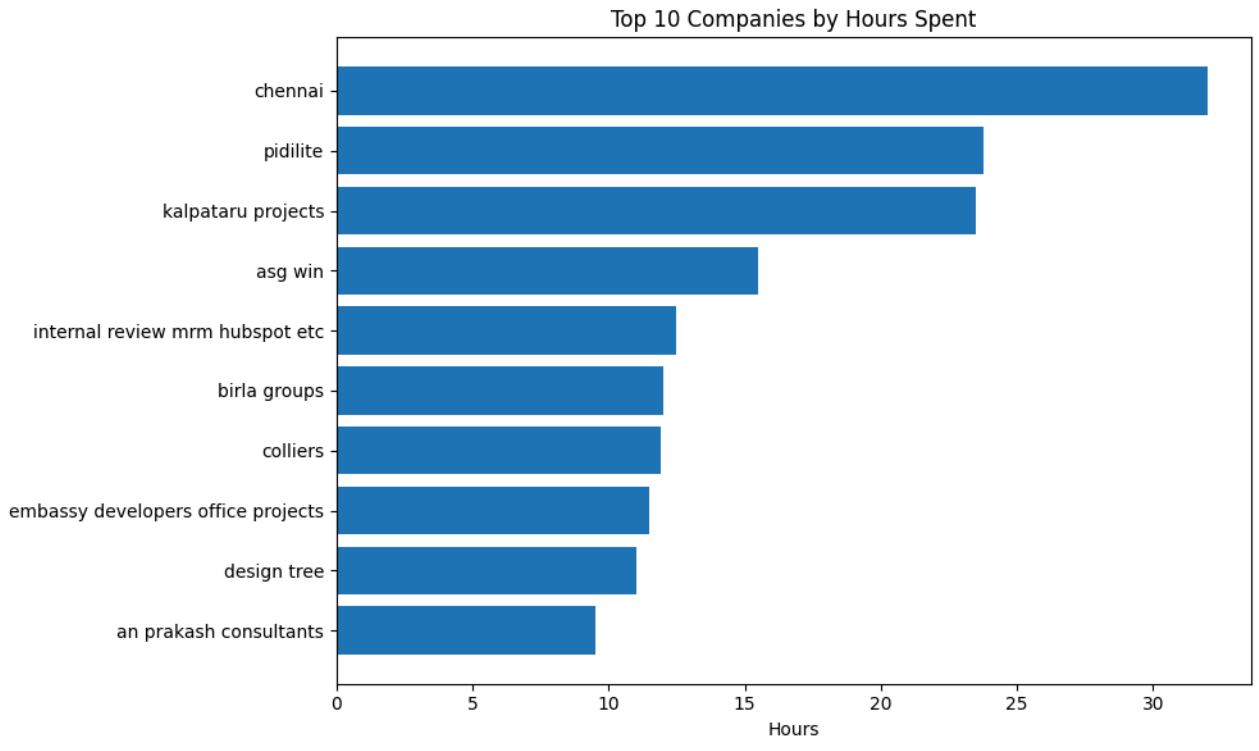

```
value_by_stage = df_inquiry.groupby('Stage')['Value (INR)'].sum().sort_values()

plt.figure(figsize=(10,5))
plt.barh(value_by_stage.index, value_by_stage.values)
plt.title("Pipeline Value (INR) by Stage")
plt.xlabel("Total Value (INR)")
plt.tight_layout()
plt.show()

```



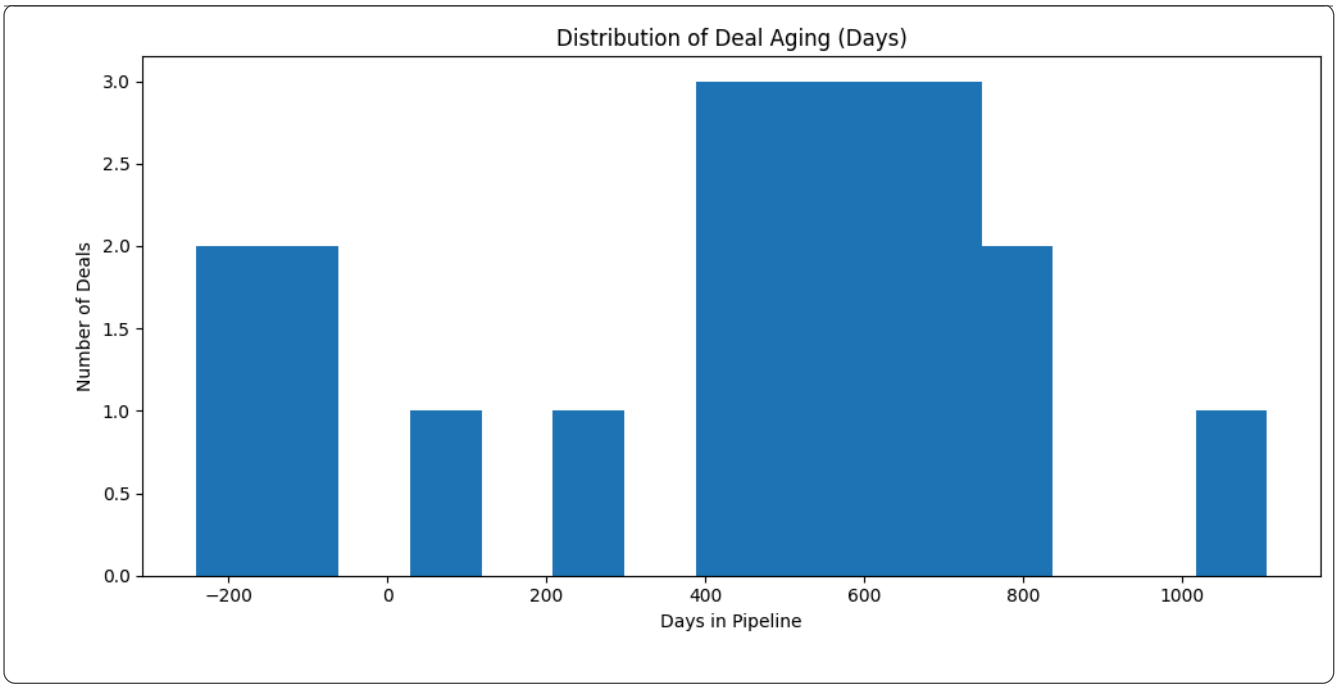

```
df_meet['month'] = df_meet['Date of Activity'].dt.to_period('M')
meetings_per_month = df_meet.groupby('month').size()

plt.figure(figsize=(10,5))
plt.plot(meetings_per_month.index.astype(str), meetings_per_month.values, marker='o')
plt.title("Meetings Per Month")
plt.xlabel("Month")
plt.ylabel("Number of Meetings")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```




```
top_companies = df_time.groupby('company')['total_hours'].sum().nlargest(10)

plt.figure(figsize=(10,6))
plt.barh(top_companies.index[::-1], top_companies.values[::-1])
plt.title("Top 10 Companies by Hours Spent")
plt.xlabel("Hours")
plt.tight_layout()
plt.show()

```




```
plt.figure(figsize=(10,5))
plt.hist(df_inquiry['lead_duration_days'].dropna(), bins=15)
plt.title("Distribution of Deal Aging (Days)")
plt.xlabel("Days in Pipeline")
plt.ylabel("Number of Deals")
plt.tight_layout()
plt.show()

```


Distribution of Deal Aging (Days)
Number of Deals
Days in Pipeline
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−200
0
200
400
600
800
1000