Utility Intelligence Framework: Turning Water Data into Revenue, Insight & Action

An Analytics-Driven Approach to Segmentation, Forecasting & Customer Optimization.

Analyst: Elizabeth Nyadimo

Date: 2025

Water access, billing compliance, and service efficiency remain pressing challenges for many utility companies in Kenya. Despite infrastructure investments and digital payment integration, utilities often struggle with revenue leakage, dormant accounts, and inconsistent usage patterns. This analysis aims to uncover customer behavior at a granular level—examining how consumption, telco-linked payments, and account activity interact. The goal is to move beyond reporting and toward actionable insights that support smarter segmentation, improved collections, and strategic decision-making.

Phase 1: Data Preprocessing

Objectives:

- Handle missing values
- Ensure proper data types
- Engineer relevant features for analysis and modeling
- Categorize customers into meaningful segments

Step 1: Load the Dataset

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# Load the dataset
df = pd.read_excel("df_anonymous_data.xlsx")

# Quick look
print(df.shape)
df.head()
```

```
(1301, 50)
```

Out[1]:

	name	telco_category	meter_type	status	customer_type	zone	line	january_pr
0	Person_1	Safaricom	REGULAR	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
1	Person_2	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_2	Line_2	
2	Person_3	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
3	Person_4	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
4	Person_5	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	

5 rows × 50 columns

Step 2: Handle Missing Values

- Check for Nulls

In [2]:
```
missing = df.isnull().sum()
missing[missing > 0].sort_values(ascending=False)
```

Out[2]:
```
Series([], dtype: int64)
```

Step 3: Ensure Correct Data Types

In [3]:
```
# Check dtypes
print(df.dtypes)

# Convert columns to float (if needed)
numeric_cols = df.columns[df.columns.str.contains('reading|consumption|tariff|billi
df[numeric_cols] = df[numeric_cols].astype(float)
```

```
name                          object
telco_category                object
meter_type                    object
status                        object
customer_type                 object
zone                          object
line                          object
january_previous_reading     float64
january_current_reading      float64
january_consumption          float64
january_tariff                 int64
january_billing                int64
january_payment                int64
january_previous_balance     float64
january_current_balance      float64
february_previous_reading    float64
february_current_reading     float64
february_consumption         float64
february_tariff                int64
february_billing               int64
february_payment               int64
february_previous_balance    float64
february_current_balance     float64
march_previous_reading       float64
march_current_reading        float64
march_consumption            float64
march_tariff                   int64
march_billing                  int64
march_payment                  int64
march_previous_balance       float64
march_current_balance        float64
april_previous_reading       float64
april_current_reading        float64
april_consumption            float64
april_tariff                   int64
april_billing                  int64
april_payment                  int64
april_previous_balance       float64
april_current_balance        float64
may_previous_reading         float64
may_current_reading          float64
may_consumption              float64
may_tariff                     int64
may_billing                    int64
may_payment                    int64
may_previous_balance         float64
may_current_balance          float64
total_consumption            float64
total_billed                   int64
total_payment                  int64
dtype: object
```

Step 4: Feature Engineering

- Average Monthly Consumption

In [5]:
```
df['avg_monthly_consumption'] = df['total_consumption'] / 5
df['avg_monthly_consumption'].head()
```

Out[5]:
```
0    0.800
1    2.134
2    2.296
3    1.334
4    3.334
Name: avg_monthly_consumption, dtype: float64
```

- Average Monthly Payment

In [6]:
```
df['avg_monthly_payment'] = df['total_payment'] / 5
df['avg_monthly_payment'].head()
```

Out[6]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: avg_monthly_payment, dtype: float64
```

- Payment Ratio (How much of what was billed has been paid)

In [7]:
```
df['total_billed_safe'] = df['total_billed'].replace(0, 1)  # Avoid division by 0
df['payment_ratio'] = df['total_payment'] / df['total_billed_safe']
df['payment_ratio'].head()
```

Out[7]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: payment_ratio, dtype: float64
```

- Compliance Label (1 if paid ≥ 90%)

In [8]:
```
df['is_compliant_payer'] = (df['payment_ratio'] >= 0.9).astype(int)
df['is_compliant_payer'].head()
```

Out[8]:
```
0    0
1    0
2    0
3    0
4    0
Name: is_compliant_payer, dtype: int32
```

- Active Flag (1 if account is active)

In [9]:
```
df['is_active'] = (df['status'] == 'ACTIVE').astype(int)
df['is_active'].head()
```

Out[9]:
```
0    1
1    1
2    1
3    1
4    1
Name: is_active, dtype: int32
```

Step 5: Customer Segmentation (Categorization)

Let's define meaningful customer segments.

- Option A: Based on Monthly Consumption

In [10]:
```
def segment_usage(x):
    if x < 5:
        return 'Low'
    elif x < 15:
        return 'Moderate'
    else:
        return 'High'

df['usage_segment'] = df['avg_monthly_consumption'].apply(segment_usage)
df['usage_segment'].value_counts()
```

Out[10]:
```
usage_segment
Low         1106
Moderate     145
High          50
Name: count, dtype: int64
```

- Option B: Based on Payment Behavior

In [11]:
```
def segment_compliance(ratio):
    if ratio >= 1.0:
        return "Overpayer"
    elif ratio >= 0.9:
        return "Compliant"
    elif ratio >= 0.5:
        return "Partial"
    else:
        return "Defaulting"

df['compliance_segment'] = df['payment_ratio'].apply(segment_compliance)
df['compliance_segment'].value_counts()
```

Out[11]:
```
compliance_segment
Defaulting    1217
Overpayer       45
Partial         30
Compliant        9
Name: count, dtype: int64
```

- Option C: Combine Usage + Compliance

In [13]:
```
df['customer_segment'] = df['usage_segment'] + " / " + df['compliance_segment']
df['customer_segment'].value_counts()
```

Out[13]:
```
customer_segment
Low / Defaulting         1058
Moderate / Defaulting     112
High / Defaulting          47
Low / Overpayer            28
Moderate / Overpayer       16
Low / Partial              15
Moderate / Partial         13
Low / Compliant             5
Moderate / Compliant        4
High / Partial              2
High / Overpayer            1
Name: count, dtype: int64
```

In [14]:
```
df[['name', 'avg_monthly_consumption', 'avg_monthly_payment', 'payment_ratio',
    'is_compliant_payer', 'is_active', 'usage_segment',
    'compliance_segment', 'customer_segment']].head(10)
```

Out[14]:

	name	avg_monthly_consumption	avg_monthly_payment	payment_ratio	is_complian
0	Person_1	0.800	0.0	0.000000	
1	Person_2	2.134	0.0	0.000000	
2	Person_3	2.296	0.0	0.000000	
3	Person_4	1.334	0.0	0.000000	
4	Person_5	3.334	0.0	0.000000	
5	Person_6	4.000	0.0	0.000000	
6	Person_7	2.332	0.0	0.000000	
7	Person_8	4.600	300.0	0.434783	
8	Person_9	0.000	0.0	0.000000	
9	Person_10	261.600	43010.0	1.096075	

Phase 2: KPI Dashboard (Initial Metrics)

KPIs Computation

In [16]:
```
import pandas as pd
from prophet import Prophet
import matplotlib.pyplot as plt
```

In [17]:
```
kpi = {}
```

- ## Water Consumption

In [27]:
```
# Total & Average Monthly Consumption
cons_columns = ['january_consumption', 'february_consumption', 'march_consumption',
kpi['Total Monthly Consumption'] = df[cons_columns].sum()
kpi['Average Monthly Consumption'] = df[cons_columns].mean()

# Combine the total and average consumption into one DataFrame
consumption_table = pd.DataFrame({
    'Total Consumption': kpi['Total Monthly Consumption'],
    'Average Consumption': kpi['Average Monthly Consumption']
})

# Display the table nicely
print("Total & Average Monthly Consumption")
display(consumption_table)
```

```
Total & Average Monthly Consumption
```

	Total Consumption	Average Consumption
january_consumption	6129.09	4.711061
february_consumption	6965.70	5.354112
march_consumption	5017.50	3.856649
april_consumption	2765.91	2.125988
may_consumption	3042.90	2.338893

- ## Billing

In [28]:
```
# Total & Average Billing
bill_columns = ['january_billing', 'february_billing', 'march_billing', 'april_bill
kpi['Total Monthly Billing'] = df[bill_columns].sum()
kpi['Average Monthly Billing'] = df[bill_columns].mean()

billing_table = pd.DataFrame({
    'Total Billing': kpi['Total Monthly Billing'],
    'Average Billing': kpi['Average Monthly Billing']
})

# Display the table nicely
```

```
print("Total & Average Monthly Billing")
display(billing_table)
```

Total & Average Monthly Billing

	Total Billing	Average Billing
january_billing	231300.0	177.786318
february_billing	239100.0	183.781706
march_billing	192900.0	148.270561
april_billing	163440.0	125.626441
may_billing	182550.0	140.315142

- ## Payments

In [29]:
```
# Total & Average Payment
pay_columns = ['january_payment', 'february_payment', 'march_payment', 'april_payme
kpi['Total Monthly Payment'] = df[pay_columns].sum()
kpi['Average Monthly Payment'] = df[pay_columns].mean()

payment_table = pd.DataFrame({
    'Total Payment': kpi['Total Monthly Payment'],
    'Average Payment': kpi['Average Monthly Payment']
})

print(" Total & Average Monthly Payment")
display(payment_table)
```

Total & Average Monthly Payment

	Total Payment	Average Payment
january_payment	192670.0	148.093774
february_payment	202350.0	155.534204
march_payment	64480.0	49.561875
april_payment	168000.0	129.131437
may_payment	21400.0	16.448885

- ## Payment Compliance Ratio

In [30]:
```
# Payment Compliance Ratio
total_payment = kpi['Total Monthly Payment'].sum()
total_billed = kpi['Total Monthly Billing'].sum()
kpi['Payment Compliance Ratio'] = round(total_payment / total_billed, 4)

compliance_table = pd.DataFrame({
    'Metric': ['Payment Compliance Ratio'],
    'Value': [f"{kpi['Payment Compliance Ratio']:.2%}"]
```

```
})

print(" Payment Compliance")
display(compliance_table)
```

```
Payment Compliance
```

	Metric	Value
0	Payment Compliance Ratio	64.29%

64.3% of the total billed amount has been paid across the five months. This indicates a gap in collection efficiency.

- ## Account Status

In [31]:
```
# Active vs Inactive
kpi['Account Status Counts'] = df['status'].value_counts()

status_table = pd.DataFrame({
    'Account Status': kpi['Account Status Counts'].index,
    'Count': kpi['Account Status Counts'].values
})

print(" Account Status Breakdown")
display(status_table)
```

```
Account Status Breakdown
```

	Account Status	Count
0	ACTIVE	852
1	DORMANT	449

65.5% of accounts are active. Dormancy might impact revenue potential.

- ## Telco-based Revenue Contribution

In [32]:
```
# Telco Revenue Contribution
kpi['Telco Revenue Contribution'] = df.groupby('telco_category')['total_payment'].s

telco_table = pd.DataFrame({
    'Telco Category': kpi['Telco Revenue Contribution'].index,
    'Total Payment': kpi['Telco Revenue Contribution'].values
})

print(" Telco Revenue Contribution")
display(telco_table)
```

```
Telco Revenue Contribution
```

	Telco Category	Total Payment
0	Airtel	2400.0
1	No Number	74900.0
2	Safaricom	571600.0

Safaricom users contribute ~87.6% of total revenue.

Time Series Forecast using Prophet

- ## Prepare time series data

In [33]:
```
monthly_consumption = kpi['Total Monthly Consumption'].reset_index()
monthly_consumption.columns = ['month', 'consumption']

# Map to datetime
month_mapping = {
    'january_consumption': '2024-01',
    'february_consumption': '2024-02',
    'march_consumption': '2024-03',
    'april_consumption': '2024-04',
    'may_consumption': '2024-05',
}
monthly_consumption['ds'] = monthly_consumption['month'].map(month_mapping)
monthly_consumption['ds'] = pd.to_datetime(monthly_consumption['ds'])
monthly_consumption['y'] = monthly_consumption['consumption']
```

- ## Fit the model

In [34]:
```
model = Prophet()
model.fit(monthly_consumption[['ds', 'y']])
```

```
09:41:49 - cmdstanpy - INFO - Chain [1] start processing
09:41:50 - cmdstanpy - INFO - Chain [1] done processing
```

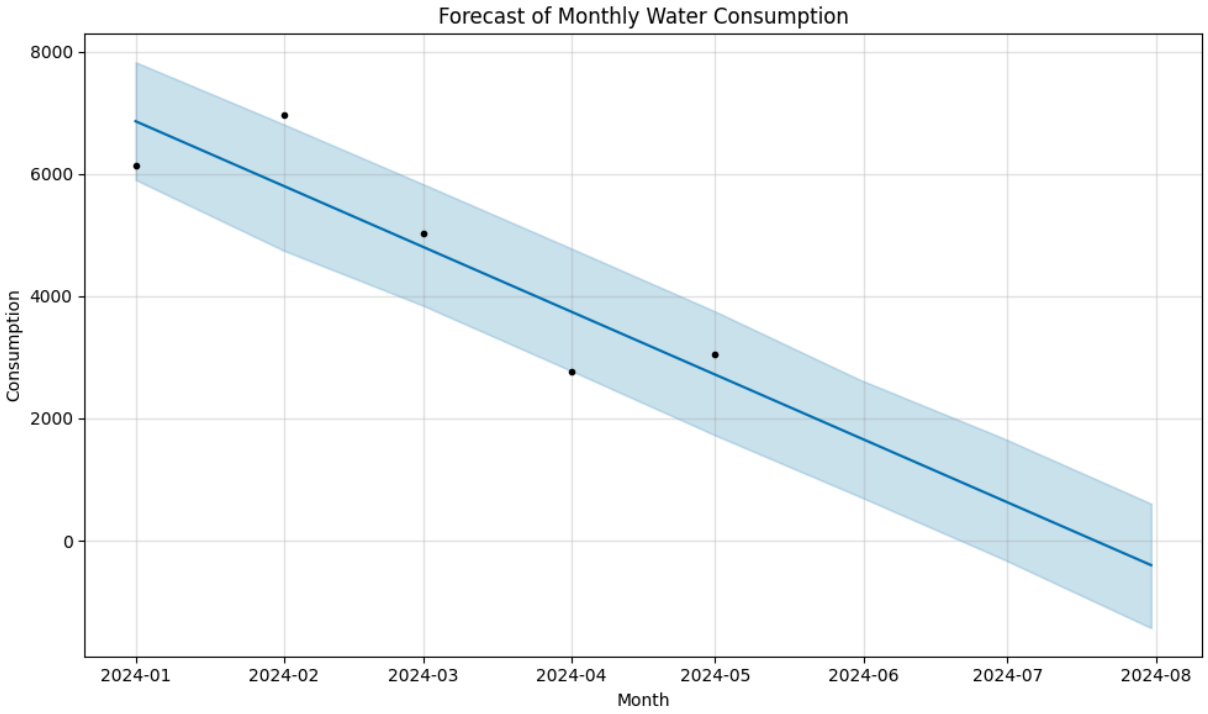
Out[34]:
```
<prophet.forecaster.Prophet at 0x16805fdc1d0>
```

- ## Forecast next 3 months

In [36]:
```
future = model.make_future_dataframe(periods=3, freq='M')
forecast = model.predict(future)
```

In [37]:
```
# Plot forecast
model.plot(forecast)
plt.title('Forecast of Monthly Water Consumption')
plt.xlabel('Month')
plt.ylabel('Consumption')
```

```
plt.tight_layout()
plt.show()
```



Phase 3: Exploratory Data Analysis (EDA)

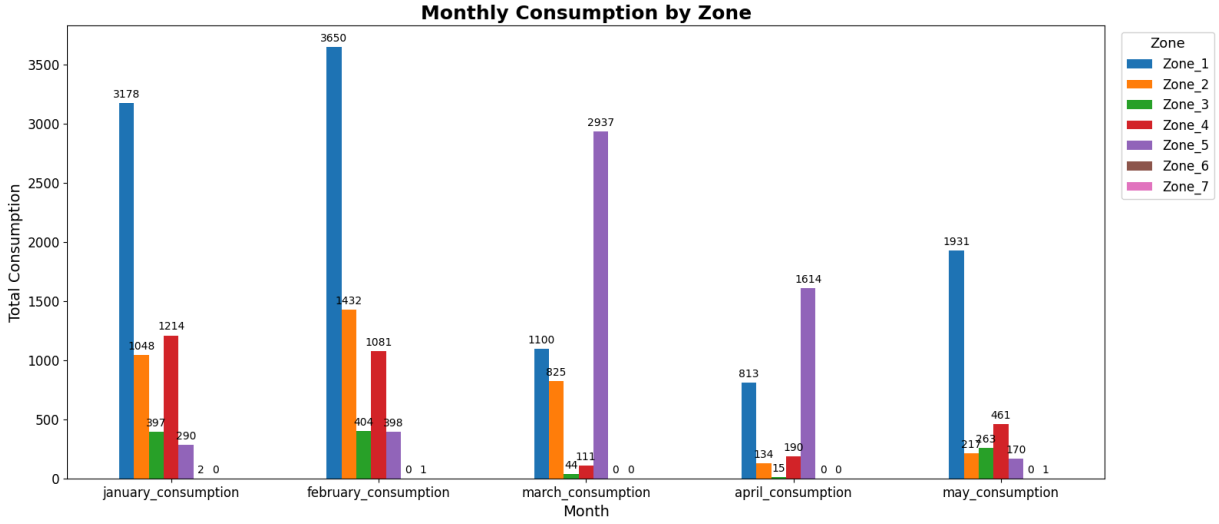
Monthly Trends by Zone / Line / Customer Type / Telco

We'll focus on consumption, billing, and payment per month, grouped by each category.

- a) Monthly Consumption by Zone

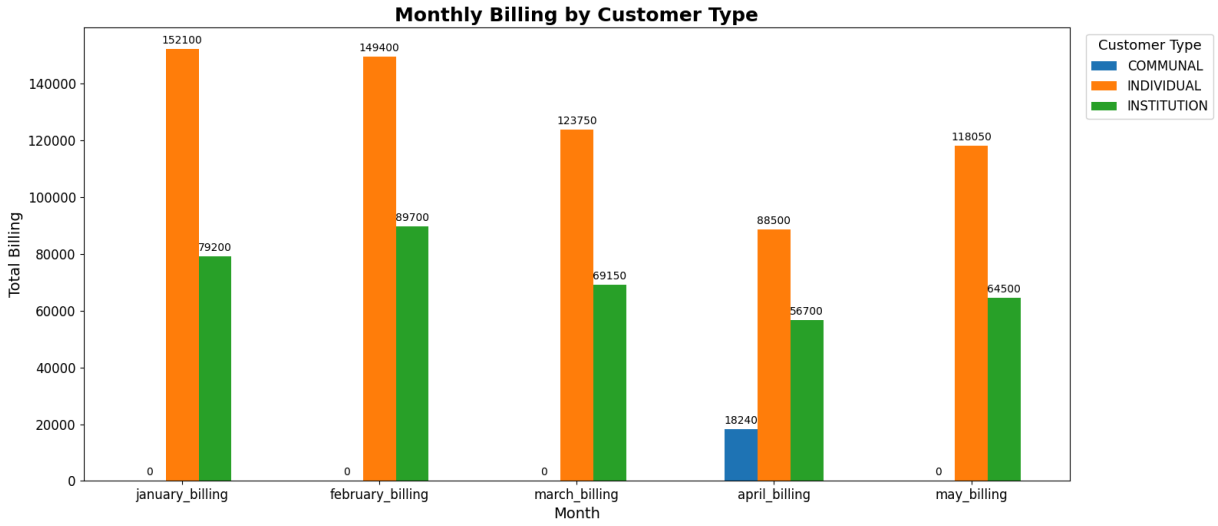
In [39]:
```
# Group and transpose
monthly_zone = df.groupby('zone')[[
    'january_consumption', 'february_consumption', 'march_consumption', 'april_cons
]].sum().T

# Plot
fig, ax = plt.subplots(figsize=(16, 7))
monthly_zone.plot(kind='bar', ax=ax)

# Title and labels
ax.set_title("Monthly Consumption by Zone", fontsize=18, fontweight='bold')
ax.set_ylabel("Total Consumption", fontsize=14)
ax.set_xlabel("Month", fontsize=14)
ax.tick_params(axis='x', labelrotation=0, labelsize=12)
ax.tick_params(axis='y', labelsize=12)

# Add value labels to each bar
for container in ax.containers:
```

```
    ax.bar_label(container, fmt='%.0f', label_type='edge', fontsize=10, padding=3)

# Adjust legend
ax.legend(title='Zone', fontsize=12, title_fontsize=13, bbox_to_anchor=(1.01, 1), l

plt.tight_layout()
plt.show()
```



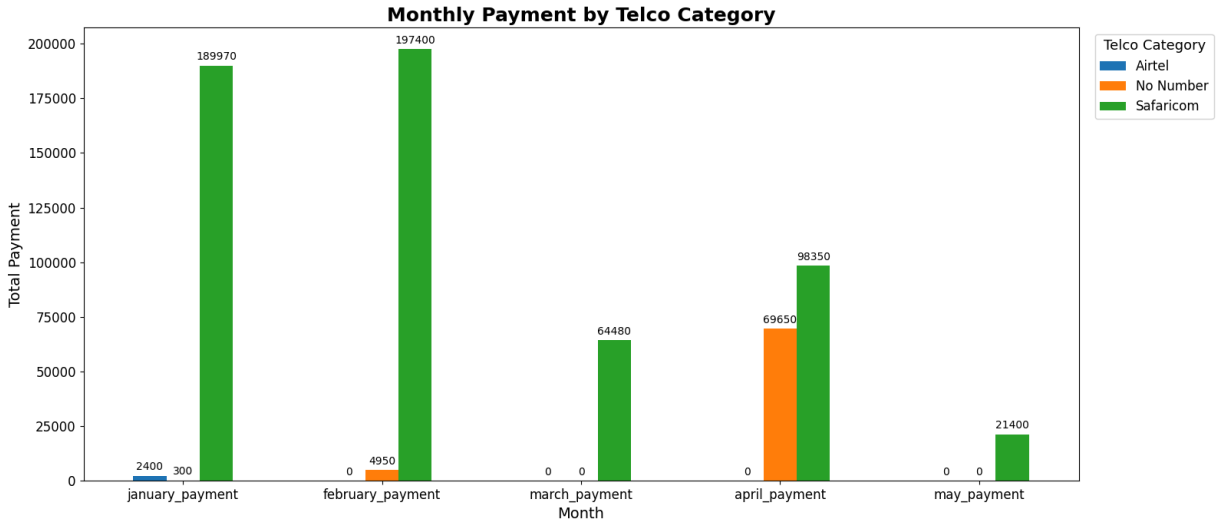
- **b) Monthly Billing by Customer Type**

In [41]:
```
# Group and transpose
monthly_custtype = df.groupby('customer_type')[[
    'january_billing', 'february_billing', 'march_billing', 'april_billing', 'may_b
]].sum().T

# Plot
fig, ax = plt.subplots(figsize=(16, 7))
monthly_custtype.plot(kind='bar', ax=ax)

# Title and labels
ax.set_title("Monthly Billing by Customer Type", fontsize=18, fontweight='bold')
ax.set_ylabel("Total Billing", fontsize=14)
ax.set_xlabel("Month", fontsize=14)
ax.tick_params(axis='x', labelrotation=0, labelsize=12)
ax.tick_params(axis='y', labelsize=12)

# Add value labels to each bar
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', fontsize=10, padding=3)

# Adjust legend
ax.legend(title='Customer Type', fontsize=12, title_fontsize=13, bbox_to_anchor=(1.

plt.tight_layout()
plt.show()
```



- c) Monthly Payment by Telco Category

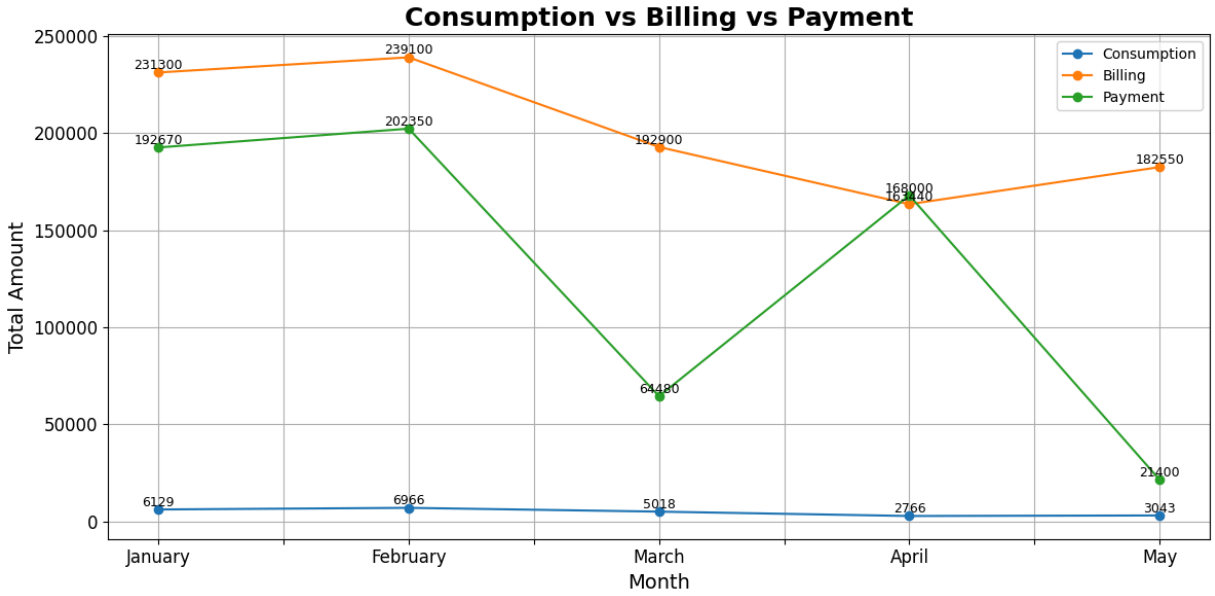
In [43]:
```
# Group and transpose
monthly_telco = df.groupby('telco_category')[[
    'january_payment', 'february_payment', 'march_payment', 'april_payment', 'may_p
]].sum().T

# Plot
fig, ax = plt.subplots(figsize=(16, 7))
monthly_telco.plot(kind='bar', ax=ax)

# Titles and labels
ax.set_title("Monthly Payment by Telco Category", fontsize=18, fontweight='bold')
ax.set_ylabel("Total Payment", fontsize=14)
ax.set_xlabel("Month", fontsize=14)
ax.tick_params(axis='x', labelrotation=0, labelsize=12)
ax.tick_params(axis='y', labelsize=12)

# Add value labels
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', fontsize=10, padding=3)

# Adjust legend
ax.legend(title='Telco Category', fontsize=12, title_fontsize=13, bbox_to_anchor=(1

plt.tight_layout()
plt.show()
```



- d) Consumption vs Billing vs Payment (Line Plot Overview)

In [44]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Define month order
month_order = ['January', 'February', 'March', 'April', 'May']

# Define columns
cons_columns = ['january_consumption', 'february_consumption', 'march_consumption',
bill_columns = ['january_billing', 'february_billing', 'march_billing', 'april_bill
pay_columns  = ['january_payment', 'february_payment', 'march_payment', 'april_paym

# Fill missing values with 0 to avoid posx/posy errors
df[cons_columns] = df[cons_columns].fillna(0)
df[bill_columns] = df[bill_columns].fillna(0)
df[pay_columns]  = df[pay_columns].fillna(0)

# Sum for each month
cons = df[cons_columns].sum().values
bill = df[bill_columns].sum().values
pay  = df[pay_columns].sum().values

# Create ordered DataFrame
monthly_compare = pd.DataFrame({
    'Month': month_order,
    'Consumption': cons,
    'Billing': bill,
    'Payment': pay
})

# Plot
fig, ax = plt.subplots(figsize=(12, 6))
monthly_compare.set_index('Month').plot(kind='line', marker='o', ax=ax)

# Style
ax.set_title("Consumption vs Billing vs Payment", fontsize=18, fontweight='bold')
```

```
ax.set_ylabel("Total Amount", fontsize=14)
ax.set_xlabel("Month", fontsize=14)
ax.tick_params(axis='x', labelrotation=0, labelsize=12)
ax.tick_params(axis='y', labelsize=12)
ax.grid(True)

# Add value labels only for valid numbers
for line in ax.lines:
    for x, y in zip(line.get_xdata(), line.get_ydata()):
        if np.isfinite(y):
            ax.text(x, y + 100, f'{y:.0f}', ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()
```



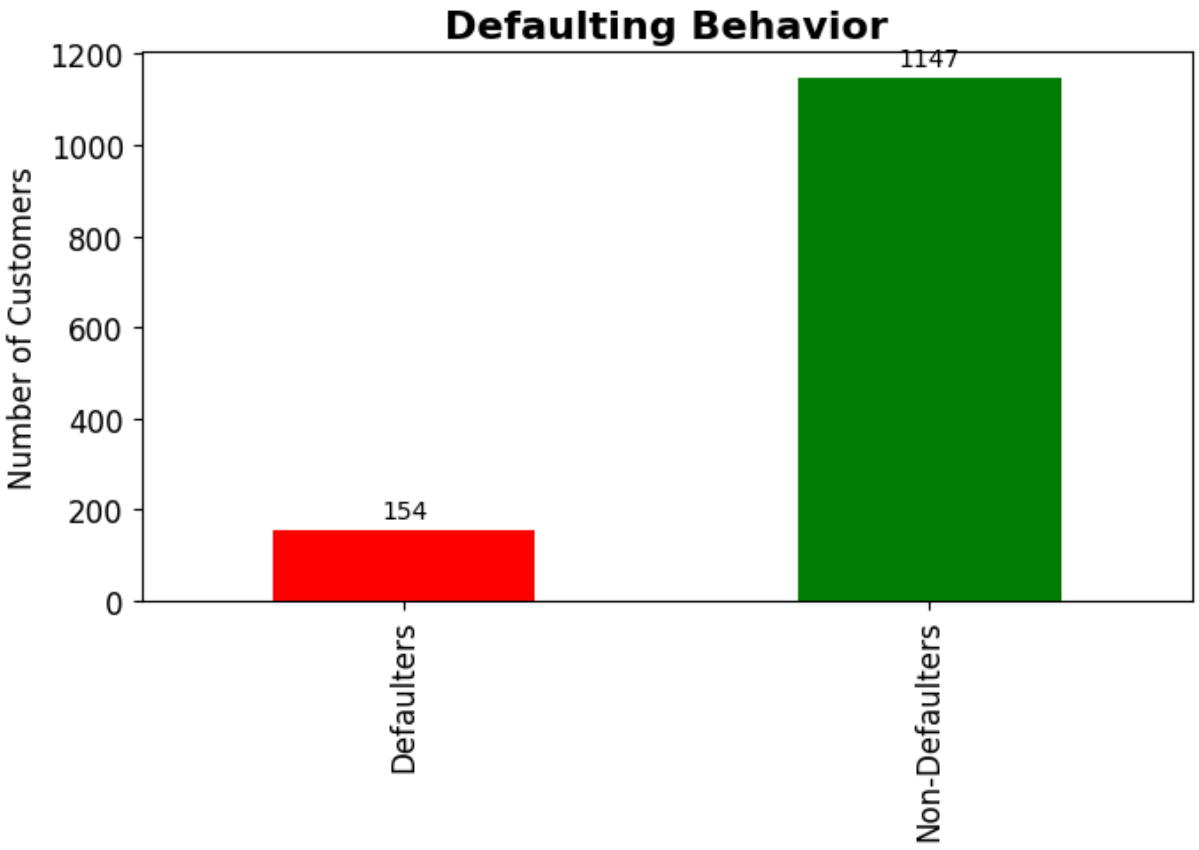
- e) Defaulting Behavior (Payment < Billing)

In [45]:
```
# Step 1: Define defaulting
df['total_billing_5m'] = df[bill_columns].sum(axis=1)
df['total_payment_5m'] = df[pay_columns].sum(axis=1)
df['defaulted'] = df['total_payment_5m'] < df['total_billing_5m']

# Summary table
default_table = df['defaulted'].value_counts().rename({
    True: 'Defaulters', False: 'Non-Defaulters'
}).to_frame('Count')

# Display table
print(" Defaulting Behavior")
display(default_table)

# Step 2: Count and label
default_counts = df['defaulted'].value_counts()
default_table = pd.Series({
    'Defaulters': default_counts.get(True, 0),
    'Non-Defaulters': default_counts.get(False, 0)
```

```
})

# Step 3: Plot with fixed colors
colors = ['red', 'green']  # Red for Defaulters, Green for Non-Defaulters

fig, ax = plt.subplots(figsize=(7, 5))
bars = default_table.plot(kind='bar', color=colors, ax=ax)

# Styling
ax.set_title("Defaulting Behavior", fontsize=16, fontweight='bold')
ax.set_ylabel("Number of Customers", fontsize=12)
ax.set_xlabel("")
ax.tick_params(axis='x', labelsize=12)
ax.tick_params(axis='y', labelsize=12)

# Add value labels
for container in ax.containers:
    ax.bar_label(container, fmt='%d', fontsize=10, padding=3)

plt.tight_layout()
plt.show()
```

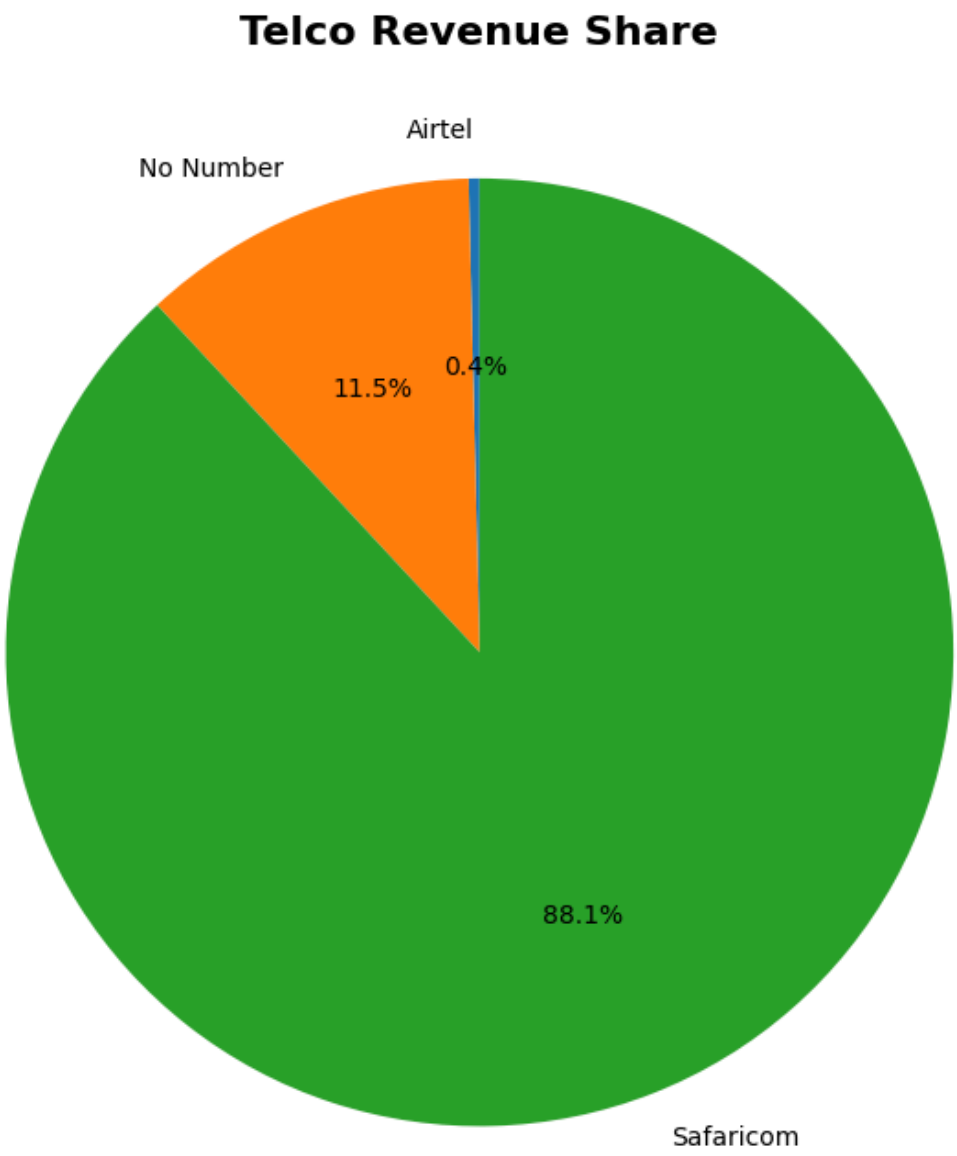
Defaulting Behavior

	Count
defaulted	
Non-Defaulters	1147
Defaulters	154



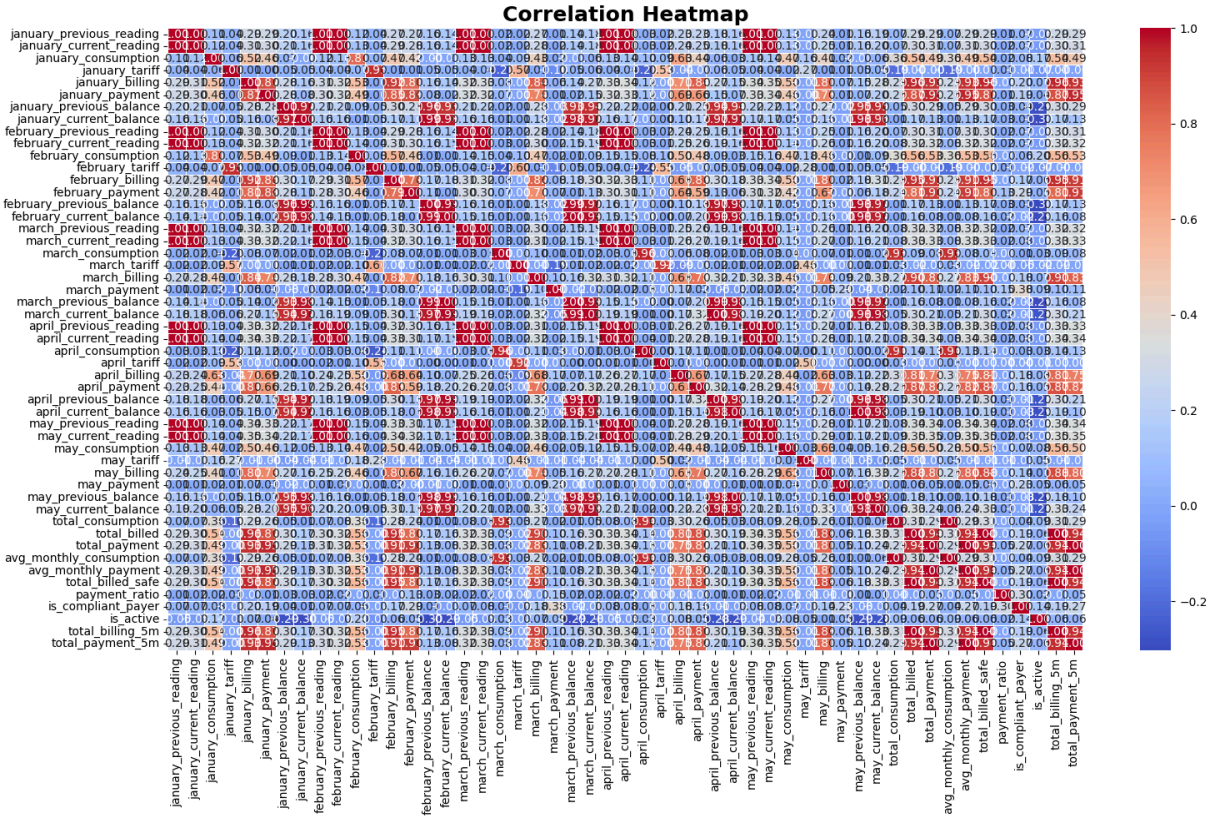
- f) Telco-Wise Segmentation (Pie of Payments)

In [46]:
```
# Group and sum
telco_pie = df.groupby('telco_category')['total_payment'].sum()

# Plot
fig, ax = plt.subplots(figsize=(7, 7))
telco_pie.plot(kind='pie', autopct='%1.1f%%', ax=ax, startangle=90)

# Styling
ax.set_title("Telco Revenue Share", fontsize=16, fontweight='bold')
ax.set_ylabel("")  # Remove y-label for cleaner pie

plt.tight_layout()
plt.show()
```



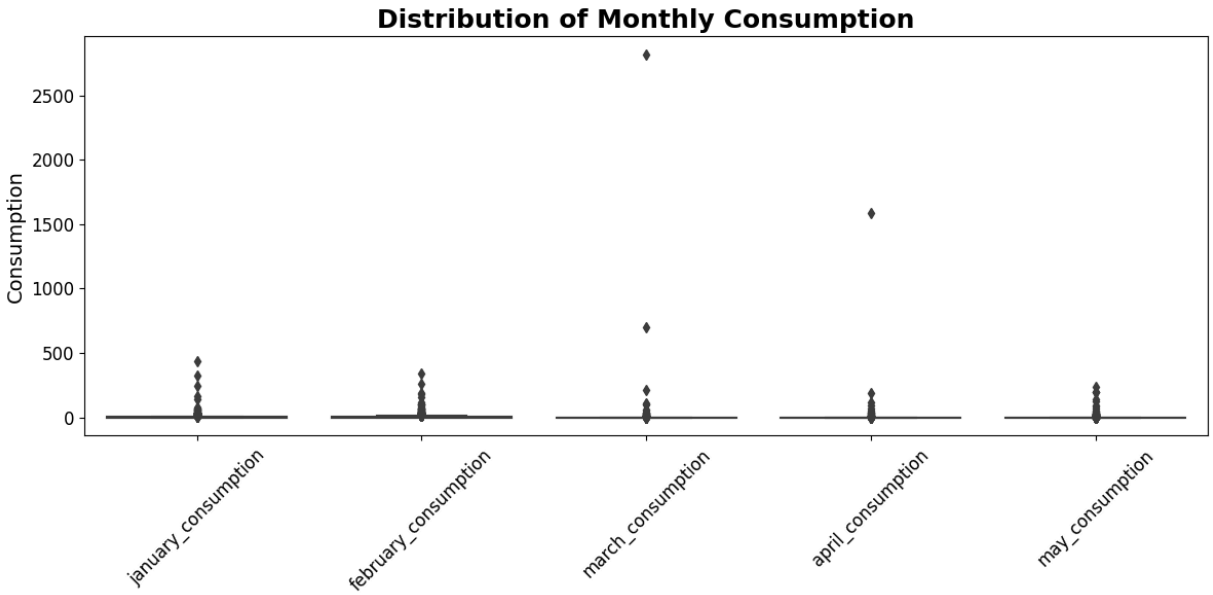
- g) Heatmap of Correlations

In [48]:
```
import seaborn as sns
# Correlation heatmap
plt.figure(figsize=(16, 10))
sns.heatmap(df.select_dtypes(include=['float', 'int']).corr(),
            cmap='coolwarm', annot=True, fmt='.2f', linewidths=0.5)

# Title
plt.title("Correlation Heatmap", fontsize=18, fontweight='bold')
plt.tight_layout()
plt.show()
```



- **h) Boxplots for Monthly Consumption**

In [49]:
```
# Boxplot for consumption
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[cons_columns])

# Styling
plt.title("Distribution of Monthly Consumption", fontsize=18, fontweight='bold')
plt.ylabel("Consumption", fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

plt.tight_layout()
plt.show()
```


Distribution of Monthly Consumption
Consumption
2500
2000
1500
1000
500
0
january_consumption
february_consumption
march_consumption
april_consumption
may_consumption

In [ ]: