```
# For categorical columns, use a mode imputation or a separate category
df['payment_status'].fillna(df['payment_status'].mode()[0], inplace=True)

#Verify missing values are handled
print(df.isnull().sum())
```

2. **Rule 2: Convert `bill_date` to datetime:**

```python
df['bill_date'] = pd.to_datetime(df['bill_date'], errors='coerce')  #errors='coerce' handles invalid dates

#Check for any remaining invalid dates after conversion.
print(df[df['bill_date'].isnull()])
#If any invalid dates exist, you may need further investigation or removal
```

3. **Rule 3: Identify and Handle Outliers:**

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize distributions to identify outliers (example with boxplots)
numerical_cols = ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']
for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()

#Strategies for handling outliers (choose one or a combination):
#1. Capping: Set values above/below thresholds to the thresholds
#   Example: df.loc[df['amount_due'] > 200, 'amount_due'] = 200
#2. Removal: Remove rows with outliers (use cautiously)
#   Example: df = df[(df['amount_due'] < 200) & (df['data_usage_gb']<100)]
#3. Transformation (log, square root, etc.): Less common for this type of data but may be helpful in certain cases
```

4. **Rule 4: Remove Duplicate Records:**

```python
# Identify duplicates
duplicates = df[df.duplicated(keep=False)]
print(f"Number of duplicate rows: {len(duplicates)}")

# Remove duplicates
```

df.drop_duplicates(inplace=True)
```

**Important Considerations:**

- **Missing Value Strategy:** The choice of imputation (mean, median, mode, etc.) depends on the data and the analysis goals. Dropping rows is generally a last resort as it can lead to information loss.
- **Outlier Handling:** The methods for handling outliers (capping, removal, transformation) should be selected based on the nature of the outliers and the context. It's often helpful to investigate the reasons behind outliers before deciding on a course of action.
- **Domain Expertise:** Consult with telecom experts to understand the possible reasons for unusual values and the best way to handle them. For example, very high data usage might indicate legitimate business needs or could signal fraudulent activity.

Remember to adapt these code snippets and strategies to your specific data and analysis requirements. Always back up your original dataset before applying any cleaning transformations. Thoroughly test your cleaning process to ensure its accuracy.

---

- 'bill_date' column converted from object to datetime64[ns, UTC].
- Standardized 'payment_status' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/website_activity_20250603163122.parquet (website) ---

--- GenAI Cleaning Suggestions for website ---
**Identified Issues:**

- **Issue 1: Incorrect Data Types:** `customer_id` is likely an identifier and should be treated as a categorical variable. `visit_timestamp` is currently an object but should be converted to a datetime object for easier analysis and manipulation.

- **Issue 2: Missing Values:** The data profile doesn't explicitly state the presence of missing values, but it's crucial to check for them in real data. We'll assume potential missing values in all columns and handle them accordingly.

- **Issue 3: Outliers:** `time_on_page_seconds` might contain outliers. We'll need to investigate this further.

- **Issue 4: Duplicate Records:** The profile doesn't mention duplicates, but we should check for and handle them.

**Suggested Cleaning Rules & Code:**

Assuming your data is in a Pandas DataFrame called `df`:

1. **Rule 1: Convert `visit_timestamp` to datetime:**

```python

```
import pandas as pd

# Assuming 'visit_timestamp' is already in your DataFrame
df['visit_timestamp'] = pd.to_datetime(df['visit_timestamp'])
```

2. **Rule 2: Handle Missing Values:** We'll use a simple strategy of dropping rows with any missing values. For more sophisticated handling (imputation), consider techniques like mean/median imputation or using more advanced models.

```python
df.dropna(inplace=True) #Drops rows with any NA values
```

3. **Rule 3: Identify and Handle Outliers in `time_on_page_seconds`:** We'll use the Interquartile Range (IQR) method to identify outliers.

```python
Q1 = df['time_on_page_seconds'].quantile(0.25)
Q3 = df['time_on_page_seconds'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Option 1: Remove outliers (be cautious with this!)
df = df[(df['time_on_page_seconds'] >= lower_bound) & (df['time_on_page_seconds'] <=
upper_bound)]

# Option 2: Cap outliers (less data loss)
df['time_on_page_seconds'] = df['time_on_page_seconds'].clip(lower=lower_bound,
upper=upper_bound)
```

4. **Rule 4: Remove Duplicate Records:**

```python
df.drop_duplicates(inplace=True)
```

5. **Rule 5: Convert `customer_id` to categorical (optional but recommended):**

```python
df['customer_id'] = df['customer_id'].astype('category')
```

Complete Example (Illustrative):

```python
import pandas as pd
import numpy as np

# Sample data (replace with your actual data)
```

```
data = {
    'customer_id': ['C0024', 'C0195', 'C0160', 'C0036', 'C0148', 'C0024'],
    'page_visited': ['faq', 'support', 'plans', 'contact_us', 'homepage', 'faq'],
    'visit_timestamp': ['2025-06-03T15:59:22.428394', '2025-06-03T16:12:22.428404',
'2025-06-03T16:09:22.428407', '2025-06-03T15:48:22.428409',
'2025-06-03T16:22:22.428412', '2025-06-03T15:59:22.428394'],
    'time_on_page_seconds': [120, 50, 300, 80, 150, 120],
    'browser': ['Safari', 'Firefox', 'Edge', 'Chrome', 'Safari', 'Safari']
}
df = pd.DataFrame(data)

# Apply cleaning rules
df['visit_timestamp'] = pd.to_datetime(df['visit_timestamp'])
df.dropna(inplace=True)

Q1 = df['time_on_page_seconds'].quantile(0.25)
Q3 = df['time_on_page_seconds'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df['time_on_page_seconds'] = df['time_on_page_seconds'].clip(lower=lower_bound,
upper=upper_bound)

df.drop_duplicates(inplace=True)
df['customer_id'] = df['customer_id'].astype('category')


print(df)
```

Remember to adapt these code snippets and outlier handling strategies to your specific dataset and requirements. Always examine your data before and after cleaning to ensure the transformations are appropriate and haven't introduced unintended consequences. Consider visualizing your data (e.g., box plots for `time_on_page_seconds`) to better understand its distribution and identify potential issues.

- 'visit_timestamp' column converted from object to datetime64[ns, UTC].
- Standardized 'page_visited' to title case.
- Standardized 'browser' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/usage_logs_20250603161825.parquet (usage) ---

--- GenAI Cleaning Suggestions for usage ---
Identified Issues:

1. **Missing Values:** A significant portion of `data_volume_mb` (30.19%) and `duration_seconds` (70.07%) columns contain missing values. This needs to be addressed through imputation or removal, depending on the impact on analysis.

2. **Incorrect Data Type:** The `timestamp` column is of `object` type, not a datetime type. This needs to be converted for proper temporal analysis.

3. **Potential Outliers:** While the min/max values for `data_volume_mb` and `duration_seconds` don't immediately suggest outliers, further investigation using visualization or statistical methods (e.g., box plots, IQR) might be necessary.

4. **Duplicate Records:** The data profile doesn't explicitly mention duplicates, but it's a standard check to perform on any dataset.

Suggested Cleaning Rules & Code:

Import necessary libraries:

```python
import pandas as pd
import numpy as np
```

Assume the data is loaded into a pandas DataFrame called `df`:

```python
# Sample data (replace with your actual data loading)
data = {'customer_id': ['C0537', 'C0564', 'C0919', 'C0483', 'C0788'] * 2000,
        'timestamp': ['2025-06-03T11:10:24.990277Z'] * 1000 + ['2025-06-02T22:59:24.990294Z']
* 1000 + ['2025-06-02T22:38:24.990298Z'] * 8000,
        'activity_type': ['app_login', 'video_stream', 'sms_sent', 'call', 'data_usage'] * 2000,
        'data_volume_mb': np.random.uniform(1, 100, 10000),
        'duration_seconds': np.random.uniform(10, 600, 10000)}
df = pd.DataFrame(data)
df.loc[df.index % 3 == 0, 'data_volume_mb'] = np.nan
df.loc[df.index % 2 == 0, 'duration_seconds'] = np.nan
```

1. **Handle Missing Values:** Impute missing `data_volume_mb` with the median and drop rows with missing `duration_seconds` (due to high percentage of missing data). A more sophisticated imputation method might be used if dropping rows is undesirable.

```python
#Impute missing values for data_volume_mb with median
median_data_volume = df['data_volume_mb'].median()
df['data_volume_mb'].fillna(median_data_volume, inplace=True)

#Drop rows with missing duration_seconds
df.dropna(subset=['duration_seconds'], inplace=True)
```

2. **Correct Data Type for Timestamp:** Convert the 'timestamp' column to datetime.

```python
df['timestamp'] = pd.to_datetime(df['timestamp'])
```

```

3. **Identify and Remove Duplicates:**

```python
df.drop_duplicates(inplace=True)
```

4. **Basic Outlier Detection and Handling (example using IQR for `data_volume_mb`):** This is a basic example; more robust methods exist.

```python
Q1 = df['data_volume_mb'].quantile(0.25)
Q3 = df['data_volume_mb'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df['data_volume_mb'] >= lower_bound) & (df['data_volume_mb'] <= upper_bound)]

```

5. **Standardize Formats (Example: Capitalize `activity_type`):**

```python
df['activity_type'] = df['activity_type'].str.capitalize()
```

Remember to adapt these code snippets to your specific data and needs. You might need to explore other outlier detection techniques and imputation methods based on your data's characteristics and the goals of your analysis. Always carefully consider the implications of dropping data or imputing values. Visual inspection of your data before and after cleaning is highly recommended.

- 'timestamp' column converted from object to datetime64[ns, UTC].
- Standardized 'activity_type' to title case.
- Imputing missing 'data_volume_mb' values with 0.

/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:220: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

df[col].fillna(0, inplace=True)
- Imputing missing 'duration_seconds' values with 0.
```

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/website_activity_20250603163535.parquet (website) ---

--- GenAI Cleaning Suggestions for website ---
Identified Issues:

- **Issue 1: Incorrect Data Type for `visit_timestamp`:** The `visit_timestamp` column is currently an object type, but it should be a datetime type for proper analysis and manipulation.

- **Issue 2: Potential Missing Values:** Although the data profile doesn't explicitly state the percentage of missing values in each column, it's crucial to check for and handle them. Missing values could be present and need to be addressed (e.g., imputation or removal).

- **Issue 3: Inconsistent capitalization in categorical columns:** The `page_visited` and `browser` columns might have inconsistent capitalization (e.g., "Chrome", "chrome").

- **Issue 4: Potential Outliers in `time_on_page_seconds`:** While the min and max values (6 and 293 seconds) don't seem extraordinarily large, further investigation might reveal outliers requiring handling (e.g., capping, removal).

- **Issue 5: Duplicate Records:** The data profile doesn't mention duplicates, but checking for and removing them is a standard data cleaning step.

Suggested Cleaning Rules & Code:

This code assumes your data is in a Pandas DataFrame called `df`.

1. **Convert `visit_timestamp` to datetime:**

```python
import pandas as pd

# Assuming your data is in a dataframe called 'df'
df['visit_timestamp'] = pd.to_datetime(df['visit_timestamp'])
```

2. **Handle Missing Values (Example: Imputation with median for `time_on_page_seconds`):**

```python
# Check for missing values
print(df.isnull().sum())

# Impute missing values in 'time_on_page_seconds' with the median
median_time = df['time_on_page_seconds'].median()
df['time_on_page_seconds'].fillna(median_time, inplace=True)

#Alternative: Remove rows with missing values if appropriate
#df.dropna(inplace=True)
```

3. **Standardize capitalization in categorical columns:**

```python
df['page_visited'] = df['page_visited'].str.lower()
df['browser'] = df['browser'].str.lower()
```

4. **Identify and handle potential outliers in `time_on_page_seconds` (Example: using IQR):**

```python
Q1 = df['time_on_page_seconds'].quantile(0.25)
Q3 = df['time_on_page_seconds'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

#Identify outliers
outliers = df[(df['time_on_page_seconds'] < lower_bound) | (df['time_on_page_seconds'] >
upper_bound)]
print("Outliers:", outliers)

#Options for handling outliers:
#1. Remove outliers:
#df = df[(df['time_on_page_seconds'] >= lower_bound) & (df['time_on_page_seconds'] <=
upper_bound)]

#2. Cap outliers:
df['time_on_page_seconds'] = df['time_on_page_seconds'].clip(lower=lower_bound,
upper=upper_bound)
```

5. **Remove duplicate records:**

```python
df.drop_duplicates(inplace=True)
```

Important Notes:

* The code snippets above provide examples. The best approach for handling missing values and outliers depends on the nature of your data and the goals of your analysis. Consider carefully whether to impute, remove, or cap outliers.
* Before running any data cleaning steps, it's essential to create a backup copy of your original DataFrame to avoid accidental data loss.
* Always visually inspect your data (e.g., using histograms and boxplots) to understand its distribution and identify potential issues. The sample size of 50 is small, so be cautious in your interpretations and outlier handling. More robust methods might be needed with larger datasets.

Remember to adapt these code snippets to your specific data and needs. This comprehensive approach addresses the identified potential issues and provides a foundation for more sophisticated data cleaning if needed.

———————————————————————

```
df['age'] = pd.to_numeric(df['age'], errors='coerce') #convert to numeric, non-numeric become NaN
df['age'] = df['age'].fillna(df['age'].median()) #Impute NaN with median age
```

2. **Rule 2: Convert data types:** Convert 'age' to integer and 'signup_date' to datetime.

```python
df['age'] = df['age'].astype(int)
df['signup_date'] = pd.to_datetime(df['signup_date'])
```

3. **Rule 3: Handle missing values (general):** This needs to be adapted based on the actual missing value percentages and the column. Here's an example for a simple strategy:

```python
# Check for missing values
print(df.isnull().sum())

# For example, if a column has a few missing values, drop rows
df.dropna(subset=['column_name'], inplace=True)  #Replace 'column_name' with the column

# For other strategies consider imputing using median/mode/more sophisticated techniques
```

4. **Rule 4: Identify and handle outliers in 'monthly_charges' and 'total_charges':** Use IQR method.

```python
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_filtered

df = remove_outliers_iqr(df, 'monthly_charges')
df = remove_outliers_iqr(df, 'total_charges')

```

5. **Rule 5: Remove duplicate records:**

```python
df.drop_duplicates(inplace=True)
```

Important Considerations:

* **Missing Value Strategy:** The choice of imputation method (mean, median, mode, k-NN, etc.) depends heavily on the data distribution and the impact of missing values on the analysis. Dropping rows might lead to significant data loss.

* **Outlier Treatment:** Removing outliers might be too aggressive. Consider transforming the data (e.g., log transformation) or using robust statistical methods less sensitive to outliers.
* **Domain Knowledge:** The best cleaning rules often depend on understanding the business context. For instance, unusual values in 'age' might be valid (very young or old customers). Consult with domain experts.

Remember to replace the sample data with your actual DataFrame. Before applying any cleaning rule to the entire dataset, it's highly recommended to test it on a smaller sample to ensure its correctness and impact. Always back up your original dataset before performing any data cleaning operations.

  - Imputing missing/invalid 'age' values with median (44.0).
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:187: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

  df['age'].fillna(df['age'].median(), inplace=True)
  - 'age' column converted from object to int64.
  - 'signup_date' column converted from object to datetime64[ns, UTC].
  - Standardized 'gender' to title case.
  - Standardized 'location' to title case.
  - Standardized 'contract_type' to title case.
  - Standardized 'service_plan' to title case.
  - Standardized 'payment_method' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/usage_logs_20250603162325.parquet (usage) ---

--- GenAI Cleaning Suggestions for usage ---
Identified Issues:

- **Issue 1: Missing Values:** A significant percentage of `data_volume_mb` (29.96%) and `duration_seconds` (69.94%) are missing. This needs to be addressed through imputation or removal, depending on the impact on analysis.

- **Issue 2: Incorrect Data Type:** `timestamp` is an object, not a datetime object. This prevents easy date/time-based analysis.

- **Issue 3: Potential Outliers:** While the min/max values for `data_volume_mb` and `duration_seconds` don't immediately scream outliers, further investigation (e.g., box plots) might be needed. Extreme values could skew analysis.

- **Issue 4: Duplicate Records:** The data profile doesn't explicitly mention duplicates, but it's a standard data quality check that should be performed.

Suggested Cleaning Rules & Code:

Import Libraries

```python
import pandas as pd
import numpy as np
```

1. Load Data (Assuming your data is in a CSV file named 'telecom_usage.csv')

```python
df = pd.read_csv('telecom_usage.csv')
```

2. Handle Missing Values:

We'll impute missing `data_volume_mb` with the median and drop rows where `duration_seconds` is missing due to the high percentage of missing values. A more sophisticated imputation method could be used if dropping rows is too drastic.

```python
# Impute missing data_volume_mb with the median
df['data_volume_mb'] = df['data_volume_mb'].fillna(df['data_volume_mb'].median())

# Drop rows where duration_seconds is missing
df.dropna(subset=['duration_seconds'], inplace=True)
```

3. Correct Data Type:

Convert the `timestamp` column to datetime objects. Error handling is included to manage potential parsing issues.

```python
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
# Drop rows with invalid timestamps after conversion
df.dropna(subset=['timestamp'], inplace=True)

```

4. Identify and Remove Duplicates:

```python
# Identify and remove duplicate rows
df.drop_duplicates(inplace=True)
```

5. Basic Outlier Detection and Handling (Illustrative):

This uses the IQR method. More sophisticated outlier detection methods exist, but this is a basic example.

```python
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data < lower_bound) | (data > upper_bound)]

# Detect outliers in 'data_volume_mb'
outliers_data_volume = detect_outliers_iqr(df['data_volume_mb'])
print("Outliers in data_volume_mb:\n", outliers_data_volume)

# Detect outliers in 'duration_seconds'
outliers_duration = detect_outliers_iqr(df['duration_seconds'])
print("\nOutliers in duration_seconds:\n", outliers_duration)

#Decide how to handle outliers (remove, cap, etc). Example: capping
df['data_volume_mb'] = np.clip(df['data_volume_mb'], outliers_data_volume.min(),
outliers_data_volume.max())
df['duration_seconds'] = np.clip(df['duration_seconds'], outliers_duration.min(),
outliers_duration.max())
```

6. (Optional) Standardize Formats:

If needed, you can standardize the case of categorical variables:

```python
# Standardize activity_type to lowercase
df['activity_type'] = df['activity_type'].str.lower()
```

Remember to adapt these code snippets to your specific data loading and file paths. Also, carefully consider the implications of each cleaning step, especially imputation and outlier handling, as they can affect the results of your analysis. Always back up your original data before applying cleaning transformations.

  - 'timestamp' column converted from object to datetime64[ns, UTC].
  - Standardized 'activity_type' to title case.
  - Imputing missing 'data_volume_mb' values with 0.
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:220: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

  df[col].fillna(0, inplace=True)
  - Imputing missing 'duration_seconds' values with 0.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/usage_logs_20250603160412.parquet (usage) ---

--- GenAI Cleaning Suggestions for usage ---
Identified Issues:

- **Issue 1: Missing Values:** A significant percentage of `data_volume_mb` (29.11%) and a very high percentage of `duration_seconds` (70.47%) are missing. This needs to be addressed through imputation or removal, depending on the impact on analysis.

- **Issue 2: Incorrect Data Type:** The `timestamp` column is an object, not a datetime object. This prevents easy date/time-based analysis.

- **Issue 3: Potential Outliers:** While the min/max values for `data_volume_mb` and `duration_seconds` don't immediately suggest outliers, further investigation using descriptive statistics (e.g., box plots, IQR) is needed.

- **Issue 4: Duplicate Records:** The data profile doesn't explicitly mention duplicates, but it's a standard data quality check that should be performed.

Suggested Cleaning Rules & Code:

Import necessary libraries:

```python
import pandas as pd
import numpy as np
from datetime import datetime
```

1. Load the data (replace 'your_file.csv' with your actual file name):

```python
df = pd.read_csv('your_file.csv')
```

2. Handle Missing Values:

Given the high percentage of missing `duration_seconds`, imputation might introduce significant bias. We'll remove rows with missing `duration_seconds`, and impute `data_volume_mb` using the median.

```python
#Remove rows with missing duration_seconds
```

```
df.dropna(subset=['duration_seconds'], inplace=True)

#Impute missing data_volume_mb with the median
df['data_volume_mb'].fillna(df['data_volume_mb'].median(), inplace=True)
```

3. Correct Data Type:

Convert the `timestamp` column to datetime objects. Error handling is included to manage potential parsing issues.

```python
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')

# Drop rows where timestamp conversion failed (if any)
df.dropna(subset=['timestamp'], inplace=True)
```

4. Identify and Remove Duplicates:

```python
#Identify and remove duplicate rows.  Keep the first occurrence.
df.drop_duplicates(inplace=True)
```

5. Basic Outlier Detection and Handling (for data_volume_mb and duration_seconds):

We'll use the IQR method to identify outliers. This is a basic approach; more sophisticated methods might be needed depending on the data distribution and analysis goals.

```python
def detect_outliers_iqr(data):
    q1, q3 = data.quantile([0.25, 0.75])
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return data[(data < lower_bound) | (data > upper_bound)]


outliers_data_volume = detect_outliers_iqr(df['data_volume_mb'])
outliers_duration = detect_outliers_iqr(df['duration_seconds'])

print("Outliers in data_volume_mb:\n", outliers_data_volume)
print("\nOutliers in duration_seconds:\n", outliers_duration)


#Choose a strategy for handling outliers (e.g., capping, removal):
#Example: Capping outliers at the upper and lower bounds:

df['data_volume_mb'] = np.clip(df['data_volume_mb'], outliers_data_volume.min(),
outliers_data_volume.max())
df['duration_seconds'] = np.clip(df['duration_seconds'], outliers_duration.min(),
outliers_duration.max())
```

```

**6. (Optional) Standardize Text Formats:**

If needed, you can standardize the `activity_type` column to lowercase:

```python
df['activity_type'] = df['activity_type'].str.lower()
```


Remember to adapt the outlier handling (capping, removal) strategy based on your understanding of the data and the impact on your analysis. You might also want to explore more advanced outlier detection techniques if needed. After cleaning, it's crucial to re-examine the data profile to assess the effectiveness of the cleaning steps.

  - 'timestamp' column converted from object to datetime64[ns, UTC].
  - Standardized 'activity_type' to title case.
  - Imputing missing 'data_volume_mb' values with 0.
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:220: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

  df[col].fillna(0, inplace=True)
  - Imputing missing 'duration_seconds' values with 0.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/billing_20250603163122.parquet (billing) ---

--- GenAI Cleaning Suggestions for billing ---
Identified Issues:

- **Issue 1: Incorrect Data Type for `bill_date`:** The `bill_date` column is of object type, but should be datetime for easier analysis and manipulation.

- **Issue 2: Potential Missing Values:** Although the profile doesn't explicitly state the percentage of missing values for each column, it's crucial to check and handle them appropriately. We'll assume some missing values exist.

- **Issue 3: Outliers in `amount_due`, `data_usage_gb`, `call_minutes`, and `sms_count`:** While the min/max values don't immediately scream outliers, further investigation with descriptive statistics (e.g., box plots) might reveal extreme values needing attention.
```

- **Issue 4: Potential Duplicate Records:** The data profile doesn't mention duplicates, but it's a standard data quality check.

- **Issue 5: Inconsistent formatting (potential):** The data profile doesn't highlight it, but we should check for inconsistent formatting within text columns (e.g., inconsistent capitalization in `payment_status`).

Suggested Cleaning Rules & Code:

Import necessary libraries

```python
import pandas as pd
import numpy as np
```

1. Convert `bill_date` to datetime:

```python
def clean_bill_date(df):
    df['bill_date'] = pd.to_datetime(df['bill_date'], errors='coerce')
    return df

#Example Usage: Assuming your dataframe is called 'df'
#df = clean_bill_date(df)
```

`errors='coerce'` handles potential errors during conversion by setting invalid dates to `NaT` (Not a Time).

2. Handle Missing Values:

We'll use a simple imputation strategy for numeric columns (mean imputation) and a mode imputation for categorical columns. Missing `bill_date` entries will be dropped as they're crucial.

```python
def handle_missing_values(df):
    #Drop rows with missing bill_dates
    df.dropna(subset=['bill_date'], inplace=True)

    for col in ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']:
        df[col] = df[col].fillna(df[col].mean())

    df['payment_status'] = df['payment_status'].fillna(df['payment_status'].mode()[0])
    return df

#Example Usage
#df = handle_missing_values(df)
```

3. Identify and Handle Outliers: We'll use the IQR method for outlier detection and removal. This is a basic example and more sophisticated methods might be needed depending on the data distribution.

```python
def handle_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df

# Example usage (apply to relevant columns):
#for col in ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']:
#    df = handle_outliers_iqr(df, col)
```

4. Remove Duplicate Records:

```python
def remove_duplicates(df):
    df.drop_duplicates(inplace=True)
    return df

# Example usage:
#df = remove_duplicates(df)

```

5. Standardize Text Formats (example for `payment_status`):

```python
def standardize_text(df):
    df['payment_status'] = df['payment_status'].str.strip().str.title()
    return df

# Example usage:
#df = standardize_text(df)
```

Complete Example (Illustrative):

```python
import pandas as pd
import numpy as np

data = {'customer_id': ['C0590', 'C0679', 'C0490', 'C0624', 'C0527', 'C0590'],
        'bill_date': ['2025-04-11', '2025-04-01', '2025-01-01', '2024-11-04',
'2024-12-22','2025-04-11'],
        'amount_due': [100.11, 150.50, 50.22, 120.75, 200.00, 100.11],
```

```
        'payment_status': ['paid', 'Due', 'paid', 'Overdue', 'due', 'Paid'],
        'data_usage_gb': [10, 20, 5, 15, 100, 10],
        'call_minutes': [100, 200, 50, 150, 500, 100],
        'sms_count': [50, 100, 25, 75, 200, 50],
        'promo_applied': [False, True, False, False, True, False]}
df = pd.DataFrame(data)


df = clean_bill_date(df)
df = handle_missing_values(df)
for col in ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']:
    df = handle_outliers_iqr(df, col)
df = remove_duplicates(df)
df = standardize_text(df)


print(df)
```

Remember to adapt the outlier handling and missing value imputation strategies based on your specific data and understanding of the domain. Always visually inspect your data (histograms, box plots) before and after cleaning to ensure the process is effective. Consider more advanced techniques if needed (e.g., KNN imputation, outlier detection using clustering).

  - 'bill_date' column converted from object to datetime64[ns, UTC].
  - Standardized 'payment_status' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/customers_csv_20250603163122.parquet (customers) ---

--- GenAI Cleaning Suggestions for customers ---
Identified Issues:

- **Issue 1: Non-numeric values in 'age' column:** The 'age' column contains non-numeric strings, preventing it from being treated as a numerical variable.
- **Issue 2: Missing values:** While the profile doesn't explicitly state the percentage, the presence of problematic data suggests the possibility of missing values (represented as blanks or other placeholders).
- **Issue 3: Inconsistent data types:** The 'age' column should be numeric, and the 'signup_date' column should be datetime.
- **Issue 4: Duplicate records:** The profile doesn't mention duplicates, but it's a standard data quality check to perform.
- **Issue 5: Outliers in 'monthly_charges' and 'total_charges':** While the min and max values seem reasonable, a more thorough check for outliers is needed.

Suggested Cleaning Rules & Code:

1. Rule: Handle non-numeric values in 'age' column. Replace non-numeric values with NaN (Not a Number) and then impute with the median age.

```python
import pandas as pd
import numpy as np

# Sample DataFrame (replace with your actual DataFrame)
data = {'customer_id': ['C0001', 'C0002', 'C0003', 'C0004', 'C0005'],
        'age': ['25', '30', 'sell', '40', '50'],
        'gender': ['Male', 'Female', 'Male', 'Female', 'Male'],
        'location': ['Miami', 'New York', 'Los Angeles', 'Chicago', 'Houston']}
df = pd.DataFrame(data)

#Convert age to numeric,coercing errors to NaN
df['age'] = pd.to_numeric(df['age'], errors='coerce')

# Impute missing age values with the median age.
median_age = df['age'].median()
df['age'].fillna(median_age, inplace=True)

print(df)
```

2. Rule: Convert 'signup_date' column to datetime.

```python
# Assuming 'signup_date' column is already in a consistent format (YYYY-MM-DD)
df['signup_date'] = pd.to_datetime(df['signup_date'])

print(df)

```

3. Rule: Handle missing values in other columns (if any). This step depends on the nature of missing values in other columns. Different strategies (e.g., imputation with mode, mean, or a constant value, or removal of rows with missing data) might be necessary. Here's an example of imputing missing values in a categorical column using the mode:

```python
# Example: Imputing missing values in 'gender' column (if any exist).
if df['gender'].isnull().any():
    df['gender'].fillna(df['gender'].mode()[0], inplace=True)

print(df)
```

4. Rule: Identify and remove duplicate records.

```python
# Identify and remove duplicate rows based on all columns
df.drop_duplicates(inplace=True)
print(df)

#Alternatively, remove duplicates based on specific columns if needed
```

#df.drop_duplicates(subset=['customer_id'], inplace=True)
```

**5. Rule: Identify and handle outliers in 'monthly_charges' and 'total_charges'.** Use IQR method to identify outliers.

```python
def detect_outliers_iqr(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

monthly_outliers = detect_outliers_iqr(df['monthly_charges'])
total_outliers = detect_outliers_iqr(df['total_charges'])

print("Monthly Charges Outliers:\n", monthly_outliers)
print("\nTotal Charges Outliers:\n", total_outliers)

# Options for handling outliers:  remove rows, cap values, or transform data (log transformation etc.)

# Example: Removing rows with outliers in 'monthly_charges'
# df = df[~df['monthly_charges'].isin(monthly_outliers)]

```

Remember to replace the sample DataFrame with your actual data. Adapt the imputation and outlier handling methods based on your specific data characteristics and business requirements. You might need to explore other outlier detection methods (e.g., Z-score) depending on your data distribution. Always carefully review the impact of your cleaning steps to avoid unintended data loss or bias.

- Imputing missing/invalid 'age' values with median (43.0).
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:187: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

df['age'].fillna(df['age'].median(), inplace=True)
- 'age' column converted from object to int64.
```

- 'signup_date' column converted from object to datetime64[ns, UTC].
- Standardized 'gender' to title case.
- Standardized 'location' to title case.
- Standardized 'contract_type' to title case.
- Standardized 'service_plan' to title case.
- Standardized 'payment_method' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/customers_csv_20250603163534.parquet (customers) ---

--- GenAI Cleaning Suggestions for customers ---
Identified Issues:

- **Issue 1: Incorrect Data Types:** The `age` column is currently an object type, but should be numeric. It also contains non-numeric values. The `signup_date` column is an object, but should be datetime.

- **Issue 2: Missing Values:** The data profile doesn't explicitly state the presence of missing values, but it's good practice to check for them.

- **Issue 3: Outliers:** The `monthly_charges` and `total_charges` columns might contain outliers that need investigation.

- **Issue 4: Inconsistent Data:** The `age` column contains non-numeric entries which need to be handled.

Suggested Cleaning Rules & Code:

1. Handle Incorrect Data Types:

```python
import pandas as pd

# Sample data (replace with your actual dataframe)
data = {'customer_id': ['C0001', 'C0002', 'C0003'],
        'age': ['19', '66', 'piece'],
        'signup_date': ['2023-03-26', '2024-07-23', '2025-01-06']}
df = pd.DataFrame(data)

# Convert age to numeric, handling errors
df['age'] = pd.to_numeric(df['age'], errors='coerce') #Coerce non-numeric values to NaN

# Convert signup_date to datetime
df['signup_date'] = pd.to_datetime(df['signup_date'], errors='coerce')

print(df)
```

2. Handle Missing Values:

```python
#Check for missing values and decide how to handle them. Example using imputation for age:
```

df['age'].fillna(df['age'].median(), inplace=True) # Fill NaN values in 'age' with the median age

#For signup_date, it depends on the context. You could drop rows with missing signup dates, or use a more sophisticated method like forward/backward fill depending on your needs.
#df.dropna(subset=['signup_date'], inplace=True) # Option to drop rows with missing signup dates

print(df)

```

**3. Identify and Remove Duplicate Records:**

```python
#Identify and drop duplicate rows
duplicates = df[df.duplicated()]
print("Duplicate Rows:\n", duplicates)
df.drop_duplicates(inplace=True)
print("After removing duplicates:\n", df)
```

4. Identify and Handle Outliers (Monthly Charges):

```python
import numpy as np

#Simple outlier detection using IQR method for monthly_charges
Q1 = df['monthly_charges'].quantile(0.25)
Q3 = df['monthly_charges'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df['monthly_charges'] < lower_bound) | (df['monthly_charges'] > upper_bound)]
print("Outliers in monthly_charges:\n", outliers)

#Decide how to handle outliers (remove, cap, or transform).  Example: capping outliers
df['monthly_charges'] = np.clip(df['monthly_charges'], lower_bound, upper_bound)

print(df)
```

5. Standardize Formats (if needed):

This step might involve changing the case of text columns or applying consistent date formats. If needed, you can add steps using `.str.lower()` or `.str.upper()` for string columns, or `.dt.strftime()` for datetime columns to achieve format consistency. For example, if location names were inconsistently capitalized:

```python
df['location'] = df['location'].str.title() #Capitalize the first letter of each word
```
```

Remember to replace the sample dataframe `df` with your actual loaded dataframe. Before implementing any cleaning rule, it's crucial to carefully analyze the data and understand the implications of each decision. Always back up your original data before making any changes. The outlier handling is just a basic example; more sophisticated techniques may be necessary depending on the nature of your data and the goals of your analysis.

  - Imputing missing/invalid 'age' values with median (43.0).
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:187: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

  df['age'].fillna(df['age'].median(), inplace=True)
  - 'age' column converted from object to int64.
  - 'signup_date' column converted from object to datetime64[ns, UTC].
  - Standardized 'gender' to title case.
  - Standardized 'location' to title case.
  - Standardized 'contract_type' to title case.
  - Standardized 'service_plan' to title case.
  - Standardized 'payment_method' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/billing_20250603163535.parquet (billing) ---

--- GenAI Cleaning Suggestions for billing ---
Identified Issues:

- **Issue 1: Missing Values:** The data profile doesn't specify the percentage of missing values, but we should assume there might be some based on real-world data. We need to handle these missing values appropriately.

- **Issue 2: Incorrect Data Type:** `bill_date` is currently an object. It should be converted to datetime format for easier analysis and manipulation.

- **Issue 3: Outliers:** While the min/max values for numerical columns seem reasonable, a visual inspection (histograms, box plots) would be necessary to identify outliers definitively. Extremely high or low values in `amount_due`, `data_usage_gb`, `call_minutes`, or `sms_count` could indicate errors or unusual customer behavior.

- **Issue 4: Duplicate Records:** The profile doesn't mention duplicates, but it's a common data quality issue. We need to check for and handle duplicate rows.

Suggested Cleaning Rules & Code:

Assumptions: The data is in a pandas DataFrame called `df`.

1. **Handle Missing Values (Imputation with median for numerical):** We will impute missing values in numerical columns using the median to minimize the impact of potential outliers. Categorical values (payment status) will be handled using a placeholder ('Unknown').

```python
import pandas as pd
import numpy as np

# Sample DataFrame (replace with your actual data loading)
data = {'customer_id': ['C0361', 'C0316', 'C0831', 'C0859', 'C0388', 'C0361', np.nan],
        'bill_date': ['2024-07-30', '2025-02-05', '2025-02-06', '2024-06-19', '2025-01-30',
'2024-07-30', '2024-08-15'],
        'amount_due': [100.0, 50.0, 150.0, 75.0, 200.0, 100.0, np.nan],
        'payment_status': ['Paid', 'Due', 'Paid', 'Overdue', 'Paid', 'Paid', np.nan],
        'data_usage_gb': [10.0, 20.0, 30.0, 15.0, 50.0, 10.0, np.nan],
        'call_minutes': [100, 200, 300, 150, 400, 100, np.nan],
        'sms_count': [50, 100, 150, 75, 200, 50, np.nan],
        'promo_applied': [False, True, False, True, False, False, True]}
df = pd.DataFrame(data)


for col in ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']:
    df[col] = df[col].fillna(df[col].median())

df['payment_status'] = df['payment_status'].fillna('Unknown')

```

2. **Convert `bill_date` to datetime:**

```python
df['bill_date'] = pd.to_datetime(df['bill_date'], errors='coerce') #errors='coerce' handles invalid
dates

```

3. **Identify and Remove Duplicates:**

```python
df.drop_duplicates(inplace=True)

```

4. **Basic Outlier Detection and Handling (using IQR):** This is a simple method; more sophisticated techniques may be needed. We'll focus on `amount_due` as an example.

```python
Q1 = df['amount_due'].quantile(0.25)
Q3 = df['amount_due'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

upper_bound = Q3 + 1.5 * IQR

#Filter out outliers or replace them with the upper or lower bound
df = df[(df['amount_due'] >= lower_bound) & (df['amount_due'] <= upper_bound)]
#or
#df.loc[(df['amount_due'] < lower_bound) | (df['amount_due'] > upper_bound), 'amount_due'] = np.nan #and then impute
```

Remember to apply similar outlier detection and handling to other numerical columns (`data_usage_gb`, `call_minutes`, `sms_count`) as needed. Visual inspection (e.g., box plots) is highly recommended to guide outlier treatment.

**Further Considerations:**

* **Data Validation:** After cleaning, add checks to ensure data integrity (e.g., are `amount_due` values always positive?).
* **Advanced Outlier Detection:** Explore techniques like Z-score or DBSCAN for more robust outlier identification.
* **Missing Value Imputation for Categorical Variables:** For `payment_status`, using the mode is a reasonable imputation method. However, if a substantial number of `payment_status` is missing, consider using more sophisticated techniques like KNN imputation (for categorical and numerical features).
* **Error Handling:** The provided code includes `errors='coerce'` in `pd.to_datetime`. This sets invalid dates to `NaT` (Not a Time). Handle these `NaT` values appropriately.

This enhanced response provides a more complete and robust data cleaning pipeline. Remember to adapt the outlier detection and handling steps based on your analysis of the data distribution.

---

- 'bill_date' column converted from object to datetime64[ns, UTC].
- Standardized 'payment_status' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/billing_20250603161825.parquet (billing) ---

--- GenAI Cleaning Suggestions for billing ---
**Identified Issues:**

- **Issue 1: Incorrect Data Type for `bill_date`:** The `bill_date` column is of object type, but it represents dates. This needs to be converted to datetime.

- **Issue 2: Missing Values:** The data profile doesn't specify the percentage of missing values, but we should check for and handle them appropriately (e.g., imputation or removal).

- **Issue 3: Potential Outliers:** Extreme values in `amount_due`, `data_usage_gb`, `call_minutes`, and `sms_count` might exist and need investigation.

- **Issue 4: Duplicate Records:** Duplicate customer bills might exist, especially based on `customer_id` and `bill_date`.
```

- **Issue 5: Inconsistent Formatting (potential):** Although not explicitly stated, there's a potential for inconsistent formatting in `bill_date` (e.g., different separators) or other string columns.

Suggested Cleaning Rules & Code:

We'll assume the data is in a Pandas DataFrame called `df`.

1. **Rule 1: Convert `bill_date` to datetime:**

```python
import pandas as pd

# Assuming your data is in a DataFrame called 'df'
df['bill_date'] = pd.to_datetime(df['bill_date'], errors='coerce')  #errors='coerce' handles invalid dates
```

2. **Rule 2: Handle Missing Values:** We'll impute missing values in numerical columns with the median and drop rows with missing values in the `bill_date` column (since it's crucial).

```python
# Identify columns with numerical data for imputation
numerical_cols = ['amount_due', 'data_usage_gb', 'call_minutes', 'sms_count']

for col in numerical_cols:
    df[col] = df[col].fillna(df[col].median())

#Drop rows with missing bill_dates (result of to_datetime error handling)
df.dropna(subset=['bill_date'], inplace=True)
```

3. **Rule 3: Identify and Handle Outliers:** We'll use the Interquartile Range (IQR) method to detect outliers. This is a basic approach; more sophisticated methods might be needed depending on the data distribution. Here, we'll just flag them for further investigation.

```python
def detect_outliers_iqr(data):
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

for col in numerical_cols:
    outliers = detect_outliers_iqr(df[col])
    print(f"Outliers in {col}: {outliers}")
    #Further action: Investigate and decide whether to remove, cap or transform these outliers
```

4. **Rule 4: Remove Duplicate Records:** We'll remove duplicate rows based on `customer_id` and `bill_date` (assuming a customer can't have two bills on the same date).

```python
df.drop_duplicates(subset=['customer_id', 'bill_date'], inplace=True)
```

5. **Rule 5: Check and Standardize String Formats (if necessary):** This step depends on the actual inconsistencies found in the data. For example, if `payment_status` has inconsistent capitalization, you could use:

```python
df['payment_status'] = df['payment_status'].str.lower() # or .str.title() for title case
```

Important Note: The outlier handling in Rule 3 only identifies them. The best course of action (remove, cap, transform) depends on the domain knowledge and the reason for the outliers. You might need to investigate those flagged outliers before deciding how to handle them. Similarly, data cleaning steps should be iterative. You may need to run these steps multiple times, potentially adjusting them based on the results you observe after each step. Always backup your original data before performing any cleaning operations.

- 'bill_date' column converted from object to datetime64[ns, UTC].
- Standardized 'payment_status' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/website_activity_20250603160412.parquet (website) ---

--- GenAI Cleaning Suggestions for website ---
Identified Issues:

- **Issue 1: Incorrect Data Types:** `visit_timestamp` is an object type, but should be datetime. `customer_id` might benefit from being numeric if feasible (depending on its nature and if the leading 'C' is consistently present).

- **Issue 2: Missing Values:** The data profile doesn't explicitly state the presence of missing values, but it's crucial to check for them in real data before proceeding.

- **Issue 3: Inconsistent Formatting (Potential):** While not explicitly mentioned, the `browser` column might have inconsistencies in capitalization (e.g., "safari" vs. "Safari").

- **Issue 4: Outliers (Potential):** Extreme values in `time_on_page_seconds` should be investigated. While the range (10-299) seems reasonable in this small sample, further analysis with the full dataset is necessary.

- **Issue 5: Duplicate Records:** The profile doesn't mention duplicates, but it's essential to check for them.

Suggested Cleaning Rules & Code:

Assuming your DataFrame is called `df`:

1. **Convert `visit_timestamp` to datetime:**

```python
import pandas as pd

df['visit_timestamp'] = pd.to_datetime(df['visit_timestamp'])
```

2. **Handle Missing Values (Illustrative):** This assumes you'll have to address missing values in real data. This example shows different strategies:

```python
# Check for missing values
print(df.isnull().sum())

# Example: Impute missing values in 'time_on_page_seconds' with the median
df['time_on_page_seconds'] =
df['time_on_page_seconds'].fillna(df['time_on_page_seconds'].median())

# Example: Drop rows with missing values in 'browser' (use cautiously)
df.dropna(subset=['browser'], inplace=True)
```

3. **Standardize `browser` capitalization:**

```python
df['browser'] = df['browser'].str.title() # Capitalizes the first letter of each word
```

4. **Identify and remove duplicate rows:**

```python
duplicates = df[df.duplicated(keep=False)] # Shows all duplicates
df.drop_duplicates(inplace=True) # Removes duplicates (keeping the first occurrence)

#Optional: Check to see if any duplicates remain.
print("Remaining duplicates after removal:",df[df.duplicated(keep=False)])
```

5. **Basic Outlier Detection and Handling (Illustrative):** This uses the Interquartile Range (IQR) method. Adjust thresholds based on your domain knowledge.

```python
Q1 = df['time_on_page_seconds'].quantile(0.25)
Q3 = df['time_on_page_seconds'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

outliers = df[(df['time_on_page_seconds'] < lower_bound) | (df['time_on_page_seconds'] > upper_bound)]
print("Potential outliers:", outliers)

#Options for handling outliers:
#1. Remove outliers:
#df = df[(df['time_on_page_seconds'] >= lower_bound) & (df['time_on_page_seconds'] <= upper_bound)]

#2. Cap outliers:
#df['time_on_page_seconds'] = df['time_on_page_seconds'].clip(lower=lower_bound, upper=upper_bound)

#3. Transform (log, etc.)
```

6. **Convert `customer_id` to numeric (if appropriate):** Only do this if the 'C' prefix is always present and doesn't carry meaningful information.

```python
#Only do this if the 'C' prefix is always present and carries no extra information.
df['customer_id'] = pd.to_numeric(df['customer_id'].str.replace('C', ''))
```

Important Notes:

- Before running any of these cleaning steps, **always back up your original data**.
- The outlier handling strategy should be chosen carefully based on the context and potential impact on your analysis. Removing outliers might discard valuable data, while capping them might distort the distribution.
- Missing value imputation methods should also be carefully considered (e.g., mean, median, mode, more advanced techniques).
- This code provides a starting point. You might need to adapt it based on the specifics of your data and the analysis you intend to perform. Always examine your data visually (e.g., using histograms, boxplots) to identify potential issues and validate the cleaning steps.

Remember to install pandas: `pip install pandas`

- 'visit_timestamp' column converted from object to datetime64[ns, UTC].
- Standardized 'page_visited' to title case.
- Standardized 'browser' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/customers_csv_20250603161825.parquet (customers) ---
--- GenAI Cleaning Suggestions for customers ---
Identified Issues:

- **Issue 1: Incorrect Data Types:** The `age` column is an object type instead of numeric, hindering numerical analysis. It also contains non-numeric values. The `signup_date` column is an object instead of datetime.
```

- **Issue 2: Missing Values:** The data profile doesn't explicitly state the presence of missing values, but it's good practice to check for them.

- **Issue 3: Inconsistent Data:** The `age` column contains non-numeric entries like 'student', 'consumer', etc. These need to be handled.

- **Issue 4: Outliers:** Extreme values in `monthly_charges` and `total_charges` might be outliers warranting investigation.

- **Issue 5: Duplicate Records:** The profile doesn't mention duplicates, but it's important to check for them.

Suggested Cleaning Rules & Code:

1. Handle Incorrect Data Types and Non-numeric Values in `age`:

```python
import pandas as pd
import numpy as np

# Sample DataFrame (replace with your actual data)
data = {'customer_id': ['C0001', 'C0002', 'C0003', 'C0004', 'C0005'],
        'age': ['46', '37', 'student', '20', '40'],
        'gender': ['Other', 'Male', 'Female', 'Male', 'Female'],
        'signup_date': ['2022-12-23', '2024-04-21', '2024-05-21', '2023-10-17', '2024-10-05'],
        'contract_type': ['Monthly', 'Annual', 'Two Year', 'Monthly', 'Annual'],
        'service_plan': ['Standard', 'Premium', 'Basic', 'Standard', 'Premium'],
        'monthly_charges': [60.0, 80.5, 75.2, 50.0, 90.1],
        'total_charges': [1000, 2000, 1500, 600, 2500],
        'payment_method': ['E-Wallet', 'Mail', 'Credit Card', 'Bank Transfer', 'E-Wallet'],
        'churn': [0, 1, 0, 0, 1]}
df = pd.DataFrame(data)


#Convert age to numeric, coercing errors to NaN
df['age'] = pd.to_numeric(df['age'], errors='coerce')

#Impute missing ages (NaN) with the median age.  Consider other imputation strategies as needed.
median_age = df['age'].median()
df['age'].fillna(median_age, inplace=True)


#Convert signup_date to datetime
df['signup_date'] = pd.to_datetime(df['signup_date'])


```

2. Check for and Handle Missing Values:

```python
```

```
# Check for missing values
print(df.isnull().sum())

#Handle missing values (example: drop rows with any missing values)
#df.dropna(inplace=True) #Or impute values as done above for age.

```

3. Identify and Remove Duplicates:

```python
# Identify duplicates
duplicates = df[df.duplicated()]
print("Duplicate Rows:\n", duplicates)

#Remove duplicates
df.drop_duplicates(inplace=True)
```

4. Basic Outlier Detection and Handling (for `monthly_charges` and `total_charges`):

```python
# Using IQR method to detect outliers

def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

monthly_outliers = detect_outliers_iqr(df['monthly_charges'])
total_outliers = detect_outliers_iqr(df['total_charges'])

print("Monthly Charges Outliers:\n", monthly_outliers)
print("Total Charges Outliers:\n", total_outliers)

#Handle outliers (example: cap values at upper and lower bounds)
df['monthly_charges'] = np.clip(df['monthly_charges'], monthly_outliers.min(),
monthly_outliers.max())
df['total_charges'] = np.clip(df['total_charges'], total_outliers.min(), total_outliers.max())

#Alternatively, you can remove the outliers:
#df = df[~df['monthly_charges'].isin(monthly_outliers)]
#df = df[~df['total_charges'].isin(total_outliers)]


```

5. Standardize Formats (Example: Capitalization):

```python
```

```
# Standardize text to lowercase (adjust as needed)
for col in ['gender', 'location', 'contract_type', 'service_plan', 'payment_method']:
    df[col] = df[col].str.lower()
```

Remember to adapt these code snippets to your specific dataset and requirements. You may need to explore more sophisticated outlier detection and handling techniques, as well as more nuanced missing value imputation strategies, depending on the characteristics of your data and the goals of your analysis. Always examine your data carefully before and after each cleaning step to ensure you're achieving the desired results and not inadvertently introducing errors.

  - Imputing missing/invalid 'age' values with median (45.0).
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data_cleaning_and_unification.py:187: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

  df['age'].fillna(df['age'].median(), inplace=True)
  - 'age' column converted from object to int64.
  - 'signup_date' column converted from object to datetime64[ns, UTC].
  - Standardized 'gender' to title case.
  - Standardized 'location' to title case.
  - Standardized 'contract_type' to title case.
  - Standardized 'service_plan' to title case.
  - Standardized 'payment_method' to title case.

--- Processing /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/staging/website_activity_20250603162325.parquet (website) ---

--- GenAI Cleaning Suggestions for website ---
Identified Issues:

- **Issue 1: Incorrect Data Types:** `customer_id` should likely be of a categorical type. `visit_timestamp` is an object but should be datetime.

- **Issue 2: Missing Values:** The data profile doesn't explicitly state the presence of missing values, but it's crucial to check for them in the actual dataset before proceeding.

- **Issue 3: Inconsistent Formatting:** While not explicitly mentioned, `browser` names might have inconsistent capitalization (e.g., "safari" vs. "Safari").

- **Issue 4: Potential Outliers:** The `time_on_page_seconds` column might contain outliers. A value of 295 seconds (almost 5 minutes) on a single page could be unusually high depending on the website's context. Further analysis is needed to determine if it's truly an outlier.

- **Issue 5: Duplicate Records:** The data profile doesn't mention duplicates, but it's important to check for them.

Suggested Cleaning Rules & Code:

Assuming your DataFrame is named `df`:

1. **Rule 1: Convert Data Types:**
   ```python
   import pandas as pd

   df['customer_id'] = df['customer_id'].astype('category')
   df['visit_timestamp'] = pd.to_datetime(df['visit_timestamp'])
   ```

2. **Rule 2: Handle Missing Values (Example - Imputation with median):**
   ```python
   # Check for missing values first
   print(df.isnull().sum())

   # Impute missing values in 'time_on_page_seconds' (if any) with the median
   if df['time_on_page_seconds'].isnull().any():
       median_time = df['time_on_page_seconds'].median()
       df['time_on_page_seconds'] = df['time_on_page_seconds'].fillna(median_time)


   #If other columns have missing values, you might drop the rows or use a different imputation method
   #depending on the characteristics of the data.  For example:
   # df.dropna(subset=['browser'], inplace=True) #Drop rows with missing browser data
   ```

3. **Rule 3: Standardize Browser Names:**
   ```python
   df['browser'] = df['browser'].str.title() #Capitalize the first letter of each word
   ```

4. **Rule 4: Identify and Handle Outliers (Example - using IQR):**
   ```python
   import numpy as np

   Q1 = df['time_on_page_seconds'].quantile(0.25)
   Q3 = df['time_on_page_seconds'].quantile(0.75)
   IQR = Q3 - Q1
   lower_bound = Q1 - 1.5 * IQR
   upper_bound = Q3 + 1.5 * IQR

   outliers = df[(df['time_on_page_seconds'] < lower_bound) | (df['time_on_page_seconds'] > upper_bound)]
   print("Outliers in 'time_on_page_seconds':\n", outliers)

   #Choose one option:
   ```

```
#Option 1: Remove outliers:
#df = df[(df['time_on_page_seconds'] >= lower_bound) & (df['time_on_page_seconds'] <= upper_bound)]

#Option 2: Cap outliers
#df['time_on_page_seconds'] = np.clip(df['time_on_page_seconds'], lower_bound, upper_bound)

#Option 3: Impute with median
#df.loc[(df['time_on_page_seconds'] < lower_bound) | (df['time_on_page_seconds'] > upper_bound), 'time_on_page_seconds'] = df['time_on_page_seconds'].median()
```

5. **Rule 5: Remove Duplicate Records:**
```python
df.drop_duplicates(inplace=True)
```

Remember to adapt these code snippets to your specific dataset and cleaning needs. You might need to adjust outlier detection thresholds or missing value handling strategies based on your understanding of the data and business context. Always inspect your data before and after applying cleaning rules to ensure that the changes are correct and haven't introduced new errors.

- 'visit_timestamp' column converted from object to datetime64[ns, UTC].
- Standardized 'page_visited' to title case.
- Standardized 'browser' to title case.

Identified a master customer dataframe (shape: (1000, 5)). Merging other data sources.
- Merged a dataframe with customer_id. Unified shape: (1000, 15)
- Merged a dataframe with customer_id. Unified shape: (1000, 19)
- Merged a dataframe with customer_id. Unified shape: (1000, 19)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)
- Merged a dataframe with customer_id. Unified shape: (1000, 26)

Dropped rows with missing customer_id. Final shape: (1000, 26)
Removed 0 duplicate customer IDs from unified data.

Initial cleaning and unification complete. Saved to /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/processed/unified_customers_20250603163811.parquet

--- GenAI Generated Data Dictionary for Unified Data ---
Telecom Customer Churn Data Dictionary and Metadata

This data dictionary describes the columns in a Pandas DataFrame used for telecom customer churn prediction. The data is a unified view from various sources, post-initial cleaning.

Column Name	Description	Data Type	Source	Example Values	Potential Use Cases	Initial Cleaning/Transformation Notes
`customer_id`	Unique identifier for each customer.	object	CRM	C12345, B67890, A01234	Primary key, joining data from different sources.	Ensure uniqueness, handle potential duplicates.
`timestamp`	Timestamp of the data record (likely representing a specific event).	datetime64[ns, UTC]	Various	2024-03-08 10:30:00+00:00, 2024-03-08 14:15:00+00:00	Feature engineering (time-based features), event sequencing.	Handle potential timezone inconsistencies, missing values.
`activity_type`	Type of customer activity (e.g., call, data usage, web login).	object	Usage Logs, Web Activity	Call, Data, Web Login, SMS	Feature engineering (one-hot encoding).	Standardize values, handle inconsistencies (case, spelling).
`data_volume_mb`	Data volume consumed in megabytes.	float64	Usage Logs	500.2, 1200.5, 25.7, 0.0	Predictor for churn (high data usage might indicate higher value customer or vice-versa).	Handle potential outliers (extremely high or low values).
`duration_seconds`	Duration of a specific activity (e.g., call duration).	float64	Usage Logs	3600.0, 120.5, 0.0	Feature engineering (binning, normalization).	Handle outliers, potential negative values.
`age`	Customer's age.	int64	CRM	25, 45, 62, 30	Predictor for churn (age might correlate with usage patterns and churn likelihood).	Handle outliers (unrealistic ages), missing values (potentially impute or remove).
`gender`	Customer's gender.	object	CRM	Male, Female, Other	Predictor for churn (potential for gender-based differences in churn).	Standardize values, handle missing or unknown values.
`location`	Customer's geographic location.	object	CRM	New York, London, Paris, Tokyo	Predictor for churn (location-based differences in service quality or pricing).	Standardize location names, handle missing values.
`signup_date`	Date when the customer signed up for the service.	datetime64[ns, UTC]	CRM	2023-01-15, 2024-02-20	Feature engineering (time since signup), cohort analysis.	Handle missing values.

`contract_type`	Type of contract (e.g., monthly, yearly).	object	Billing System	Monthly, Yearly, Two-year	Strong predictor for churn (longer contracts reduce churn).	Standardize values.
`service_plan`	Service plan subscribed to by the customer.	object	Billing System	Basic, Premium, Enterprise	Predictor for churn (plan features and price impact churn).	Standardize values.
`monthly_charges`	Monthly charges for the service.	float64	Billing System	50.0, 100.5, 25.7	Predictor for churn (price sensitivity).	Handle outliers, potential negative values.
`total_charges`	Total charges incurred by the customer.	float64	Billing System	600.0, 1206.0, 308.4	Predictor for churn (lifetime value).	Handle missing values (potentially impute based on monthly charges and tenure).
`payment_method`	Customer's payment method.	object	Billing System	Credit Card, Debit Card, Bank Transfer, Electronic Check	Predictor for churn (payment method might correlate with churn).	Standardize values.
`churn`	Churn indicator (1 for churned, 0 for not churned).	int64	CRM, Billing System	1, 0	Target variable for churn prediction.	Ensure binary values (0, 1).
`page_visited`	Web page visited by the customer.	object	Web Activity	Homepage, Billing, Support, Account Settings	Feature engineering (one-hot encoding), understanding customer engagement.	Standardize values, handle missing values.
`visit_timestamp`	Timestamp of the web page visit.	datetime64[ns, UTC]	Web Activity	2024-03-08 11:00:00+00:00, 2024-03-08 15:30:00+00:00	Feature engineering (time-based features).	Handle potential timezone inconsistencies, missing values.
`time_on_page_seconds`	Time spent on the web page.	float64	Web Activity	60.5, 300.0, 1200.0	Feature engineering (binning, normalization).	Handle outliers, potential negative values.
`browser`	Browser used by the customer for web visits.	object	Web Activity	Chrome, Firefox, Safari, Edge	Potential predictor for churn (less common browsers might indicate technical issues).	Standardize values, handle missing values.
`bill_date`	Date of the bill.	datetime64[ns, UTC]	Billing System	2024-03-15, 2024-02-28	Feature engineering (time-based features), linking with payment status.	Handle missing values.
`amount_due`	Amount due on the bill.	float64	Billing System	75.0, 150.5, 0.0	Predictor for churn (late payments or high amounts might increase churn risk).	Handle outliers, potential negative values.
`payment_status`	Status of the payment (e.g., paid, overdue).	object	Billing System	Paid, Overdue, Pending	Strong predictor for churn (overdue payments strongly correlate with churn).	Standardize values.
`data_usage_gb`	Total data usage in gigabytes.	float64	Usage Logs	0.5, 1.2, 2.5, 0.0	Predictor for churn (high data

usage might indicate higher value customer or vice-versa).| Handle potential outliers (extremely high or low values). |
`call_minutes`	Total call minutes.	float64	Usage Logs	120.5, 600.0, 0.0	Predictor for churn (call minutes might indicate engagement or frustration).	Handle outliers, potential negative values.
`sms_count`	Total SMS messages sent/received.	float64	Usage Logs	10, 100, 500, 0.0	Predictor for churn (SMS usage might indicate engagement or frustration).	Handle outliers, potential negative values.
`promo_applied`	Whether a promotion was applied to the customer's account.	object	Billing System	Yes, No	Predictor for churn (promotions might influence churn).	Standardize values ("Yes", "No").

This dictionary provides a starting point. Further analysis may reveal additional cleaning or transformation needs. Data quality checks (e.g., for consistency, completeness, and outliers) should be performed before model building.

Generated data dictionary saved to /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/processed/unified_data_dictionary.md
GenAI model initialized successfully for Feature Engineering.
Starting feature engineering process directly...
Starting Feature Engineering...
Loading latest processed data from: /Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/data/processed/unified_customers_20250603163811.parquet
--- GenAI Feature Ideation Suggestions ---
New Feature Engineering Suggestions for Telecom Churn Prediction

Here are some creative features that can be engineered from the provided DataFrame schema to improve churn prediction:

Feature Name	Description	Derivation Logic/Formula	Expected Impact
CustomerTenureMonths	Number of months the customer has been subscribed.	`(df['timestamp'] - df['signup_date']).dt.days / 30` (Approximate, consider more robust methods for leap years etc.)	Longer tenure might indicate lower churn risk, but it could also suggest customers less likely to change.
AvgMonthlyDataUsageGB	Average data usage per month.	`df['data_usage_gb'] / ((df['timestamp'] - df['signup_date']).dt.days / 30)`	High average data usage could signify higher satisfaction and lower churn, but could also lead to higher bills, potentially increasing churn.
AvgMonthlyCallMinutes	Average call minutes per month.	`df['call_minutes'] / ((df['timestamp'] - df['signup_date']).dt.days / 30)`	Similar to data usage, moderate to high average call minutes could indicate higher engagement and lower churn.

| AvgMonthlySMS | Average SMS messages per month.
| `df['sms_count'] / ((df['timestamp'] - df['signup_date']).dt.days / 30)`
| Similar to data and call minutes, moderate to high average SMS could indicate higher engagement and lower churn. |
| PaymentConsistency | Measure of consistency in on-time payments. (e.g., percentage of on-time payments) | `df.groupby('customer_id')['payment_status'].apply(lambda x: (x == 'Paid').mean())`
| Higher consistency suggests a lower churn risk. |
| AvgBillIncreaseRate | Average percentage increase in monthly charges over time.
| Calculate monthly charges, then use a rolling window or linear regression to estimate the average increase rate. Requires more complex calculation. | High increase rates might be correlated with higher churn. |
| WebsiteEngagementScore | Composite score reflecting website visit frequency and time spent on pages. | `df.groupby('customer_id')['page_visited'].count() + df.groupby('customer_id')['time_on_page_seconds'].sum()` (This needs normalization/scaling).
| Higher score indicates greater engagement, potentially lowering churn.
|
| ContractTypeMultiplier | Multiplier based on contract type (e.g., 1 for month-to-month, 0.5 for 1-year, 0.25 for 2-year). | Create a mapping: {'Month-to-month': 1, '1-year': 0.5, '2-year': 0.25} and apply to 'contract_type' column. | Lower multiplier indicates higher churn risk due to less commitment. |
| RecencyDays | Number of days since last activity (any activity type).
| `(df['timestamp'].max() - df.groupby('customer_id')['timestamp'].max()).dt.days`
| Higher recency might indicate increased churn risk. |
| FrequencyOfActivity | Number of activities within a given period (e.g., last 3 months).
| Requires aggregation and grouping by customer and time window (e.g., using `pd.Grouper`).
| Higher frequency suggests more engagement, potentially lower churn.
|
| MonetaryValue | Total charges in a given period (e.g., last 3 months).
| Requires aggregation and grouping by customer and time window (e.g., using `pd.Grouper`).
| Higher value could indicate lower churn, but requires careful consideration of individual bill characteristics. |
| PromoImpact | Binary feature indicating if a promo was applied in the last month before churn (if churned). | Requires complex logic combining `promo_applied` flag and proximity to churn date. |
Promo applications may indicate attempts to retain customers at risk.
|

Note: These features require careful data cleaning, handling of missing values, and appropriate scaling before being used in a churn prediction model. The formulas provided are simplified illustrations and might need adjustments based on data characteristics and specific requirements. For example, the time-based calculations need to be adapted to handle edge cases effectively. Feature scaling (e.g., standardization or normalization) is crucial for most machine learning algorithms.

- Engineered 'tenure_months' and 'tenure_years'.
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/feature_engineering.py:110: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
  df['charges_per_month'].fillna(df['monthly_charges'], inplace=True) # Fill inf/NaN for new customers with monthly_charges
```
```
  - Engineered 'charges_per_month'.
```
```
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/feature_engineering.py:119: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

```
  df[new_col_name].fillna(0, inplace=True) # Fill NaNs for new customers
```
```
  - Engineered 'avg_data_usage_per_month'.
```
```
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/feature_engineering.py:119: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

```
  df[new_col_name].fillna(0, inplace=True) # Fill NaNs for new customers
```
```
  - Engineered 'avg_call_per_month'.
```
```
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/feature_engineering.py:119: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

```
  df[new_col_name].fillna(0, inplace=True) # Fill NaNs for new customers
```
```
  - Engineered 'avg_sms_per_month'.
  - Engineered 'is_payment_overdue'.
```
```
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/feature_engineering.py:141: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
  df['engagement_score'].fillna(0, inplace=True) # Fill for customers with no web activity
```

```
  - Engineered 'avg_time_on_page', 'total_pages_visited', and 'engagement_score'.
  - Engineered 'contract_duration_months'.
  - Engineered 'is_male'.
  - Engineered 'uses_credit_card'.
  - Cleaned up column names.
/Users/chiranjeeviboddu/Documents/Projects/Customer_Churn_Prediction_Project/
feature_engineering.py:197: FutureWarning: A value is trying to be set on a copy of a
DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the
intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation
inplace on the original object.
```

```
  ml_ready_df[col].fillna(0, inplace=True) # Or median: ml_ready_df[col].median()
```

```
Feature Engineering complete. ML-ready data saved to /Users/chiranjeeviboddu/Documents/
Projects/Customer_Churn_Prediction_Project/data/warehouse/
ml_ready_customers_20250603163836.parquet
List of ML features saved to /Users/chiranjeeviboddu/Documents/Projects/
Customer_Churn_Prediction_Project/data/warehouse/ml_ready_features.json
(veera) chiranjeeviboddu@Chiranjeevis-MacBook-Pro Customer_Churn_Prediction_Project %
```