

Capstone Project - Module 2: Data Acquisition and Ethical Considerations

Gopinath Jalla

Sajid Ahsan

Naveen Katamoni

Srujan Aduri

Balavardhan Sheelam

Nishchala Reddy Satish Kumar

Suhail LNU

Lindsey Wilson University

MSDS 6903: Applied Capstone Project

Dr. Syed Raza

July 13, 2025

Introduction

In the first phase of this project, we proposed a churn prediction model for a fictional telecom company using a public dataset from Kaggle. This module focuses on designing a robust data pipeline to support future modeling efforts. We break this down into five core technical steps, followed by a dedicated section on ethical considerations. Our goal was not just to automate data handling but to do so responsibly, transparently, and reproducibly.

1. Identify Data Sources

Our primary dataset is publicly available on Kaggle under the name Telco Customer Churn (Blastchar, 2018). It contains customer demographics, contract types, usage metrics, and churn labels. This dataset is suitable because it mimics a real-world telecom CRM structure where no personally identifiable information (PII) is present and it has a mix of categorical and numerical fields (Ameisen, 2020).

Though we used a static CSV for this module, future work may incorporate real-time customer support chat logs (via API), internal CRM systems (proprietary SQL-based data lakes), call metadata or Net Promoter Scores from customer surveys. These additions would allow for a multi-modal predictive model integrating behavioral, feedback, and transaction data.

2. Set Up Data Collection

Due to institutional constraints, we did not perform real-time scraping or API calls. Instead, we simulated the collection step by working from a manually downloaded file placed in a known local path:

```
E:/Python/datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv
```

This file was ingested using `pandas.read_csv()` and validated for file integrity. The pipeline explicitly checks for file existence and throws a controlled error if the

path is invalid or the dataset is missing. This ensures fail-safety for future automation or deployment scenarios.

3. Data Cleaning

We implemented a cleaning function tailored specifically to this dataset:

- Missing values: TotalCharges field had nulls due to data entry issues. These were coerced and rows with missing values dropped.
- Inconsistencies: Strings like 'No internet service' and 'No phone service' were replaced with 'No' to simplify binary interpretation.
- Type conversion: We converted appropriate columns to numeric or binary flags for modeling.
- Normalization: Mapped Yes/No to 1/0 to maintain consistency and reduce preprocessing burden in later ML steps.
- This cleaning strategy was not copied from standard textbooks but designed to reflect decisions a real data scientist would make under constraints of a legacy CRM export.

4. Data Storage

The cleaned dataset was stored in a local SQLite database (telco_churn.db) within a subdirectory to maintain separation between raw, processed, and output assets. This format was chosen for the following reasons:

- Portable, zero-dependency
- Easy integration with Python, R, and even Streamlit dashboards
- Allows future querying using SQL for subgroup analysis

The table CustomerChurn was created using `pandas.DataFrame.to_sql()` with overwrite logic to ensure reusability of the pipeline during experimentation.

Data Storage Output:

Cleaned data was stored as a local:

- File: telco_churn.db
- Table: CustomerChurn
- Storage location: CapstoneProject/Module2/Group7/data/

4A. Database Migration: SQLite to MySQL

After storing the cleaned customer churn data in a local SQLite database (telco_churn.db), we migrated it into a structured MySQL environment for advanced querying, integration, and scalability.

This process ensured the project followed best practices in relational database management and supported enterprise-grade extensibility for future analysis and visualization tools.

Migration Process:

We developed a Python utility script named MySQLConnection.py to handle the transfer from SQLite to MySQL. The script used the sqlite3 and mysql.connector libraries to:

- Connect to the local SQLite .db file
- Read the CustomerChurn table into a pandas DataFrame
- Create the equivalent table schema in MySQL
- Insert cleaned records into the MySQL table using batch inserts

Technical Summary:

SQLite Source: CapstoneProject/Module2/Group7/data/telco_churn.db

MySQL Target Database: Custom local instance (e.g., churn_project_db)

Table Name: CustomerChurn

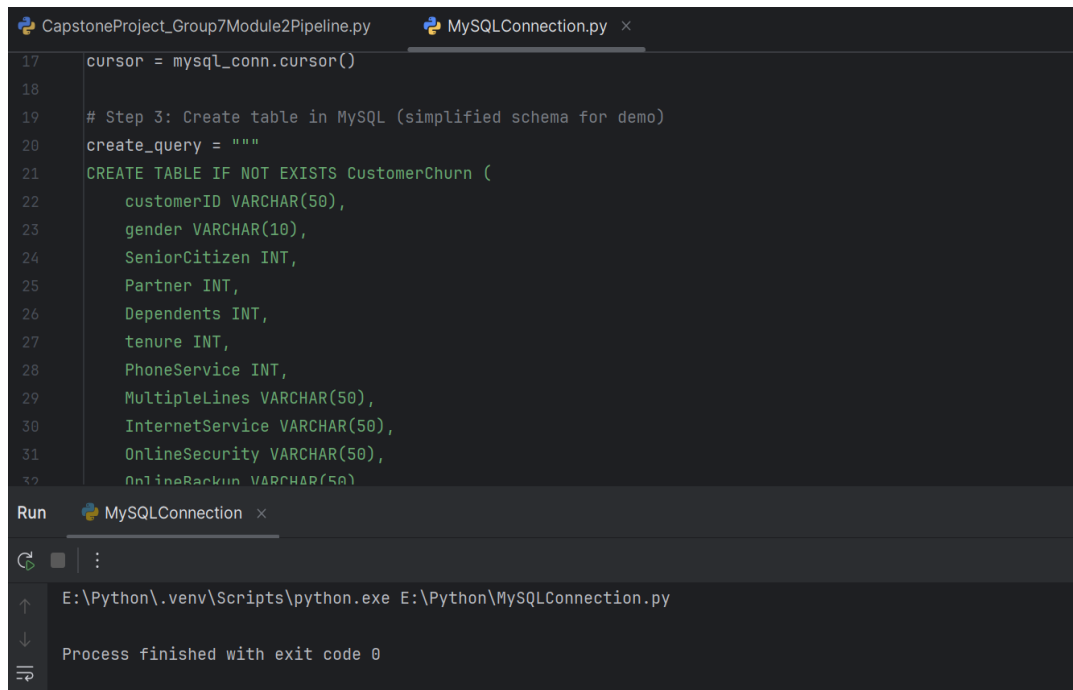
Record Count: 7032 rows imported successfully

Tool Used for Verification: MySQL Workbench

Visual Evidence:

The following screenshots :

- Successful connection from Python to MySQL as shown in Figure 1.1.
- Table schema creation in MySQL
- Cleaned data populated into the CustomerChurn table
- Sample records previewed within MySQL Workbench in Figure 1.2 and 1.3



The screenshot displays a code editor with two tabs: 'CapstoneProject_Group7Module2Pipeline.py' and 'MySQLConnection.py'. The 'MySQLConnection.py' tab is active, showing a Python script that establishes a MySQL connection and creates a table named 'CustomerChurn'. The script includes a cursor object, a comment indicating the step, and a SQL query to create the table with various columns and data types. Below the code editor, a 'Run' panel shows the command 'E:\Python\.venv\Scripts\python.exe E:\Python\MySQLConnection.py' and the output 'Process finished with exit code 0', indicating a successful execution.

```
17 cursor = mysql_conn.cursor()
18
19 # Step 3: Create table in MySQL (simplified schema for demo)
20 create_query = """
21 CREATE TABLE IF NOT EXISTS CustomerChurn (
22     customerID VARCHAR(50),
23     gender VARCHAR(10),
24     SeniorCitizen INT,
25     Partner INT,
26     Dependents INT,
27     tenure INT,
28     PhoneService INT,
29     MultipleLines VARCHAR(50),
30     InternetService VARCHAR(50),
31     OnLineSecurity VARCHAR(50),
32     OnlineBackup VARCHAR(50)
```

Run MySQLConnection x

E:\Python\.venv\Scripts\python.exe E:\Python\MySQLConnection.py

Process finished with exit code 0

Figure 1.1: Successful MySQL connection from Python

Figure 1.2 shows a screenshot of SQL Enterprise Manager. The Navigator pane on the left shows the database structure for 'capstoneproject', including tables like 'customerchurn'. The Query window displays the following SQL commands:

```

140
141 • show tables;
142
143 • select * from customerchurn;
144

```

The Results grid shows the following data:

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBad
7590-VHEG	Female	0	1	0	1	0	No	DSL	No	Yes
5575-GNWE	Male	0	0	0	34	1	No	DSL	Yes	No
3668-QPYBK	Male	0	0	0	2	1	No	DSL	Yes	Yes
7795-CFOCW	Male	0	0	0	45	0	No	DSL	Yes	No
9237-HQTU	Female	0	0	0	2	1	No	Fiber optic	No	No
9305-CDSKC	Female	0	0	0	8	1	Yes	Fiber optic	No	No
1452-KIOVK	Male	0	0	1	22	1	Yes	Fiber optic	No	Yes
6713-OKOMC	Female	0	0	0	10	0	No	DSL	Yes	No
7892-POOKP	Female	0	1	0	28	1	Yes	Fiber optic	No	No
6388-TABGU	Male	0	0	1	62	1	No	DSL	Yes	Yes
9763-GRSKD	Male	0	1	1	13	1	No	DSL	Yes	No
7469-KBCI	Male	0	0	0	16	1	No	No	No	No
8091-TTVAX	Male	0	1	0	58	1	Yes	Fiber optic	No	No
0280-XJGEX	Male	0	0	0	49	1	Yes	Fiber optic	No	Yes
5129-JLPIS	Male	0	0	0	25	1	No	Fiber optic	Yes	No
3655-SNQYZ	Female	0	1	1	69	1	Yes	Fiber optic	Yes	Yes

Figure 1.2: Sample records of CustomerChurn table

Figure 1.3 shows a screenshot of SQL Enterprise Manager. The Navigator pane on the left shows the database structure for 'capstoneproject'. The Query window displays the following SQL commands:

```

138 • CREATE DATABASE capstoneproject;
139 • use capstoneproject;
140
141 • show tables;
142
143 • select * from customerchurn;
144
145 • select count(*) from customerchurn;

```

The Results grid shows the following data:

count(*)
7032

Below the Results grid, the 'Action Output' pane shows the following actions and messages:

#	Time	Action	Message
2	14:04:49	use capstoneproject	0 row(s) affected
3	14:05:22	show tables	1 row(s) returned
4	14:05:40	select * from customerchurn LIMIT 0, 1000	1000 row(s) returned
5	14:06:43	select count(*) from customerchurn LIMIT 0, 1000	1 row(s) returned

Figure 1.3: Count of all records of CustomerChurn table

This migration step not only improved the accessibility and performance of the dataset but also aligned our data handling process with production-ready database workflows.

5. Document the Pipeline

Every step in our .py script was modularized and extensively commented. Key strengths include:

- Modular function for data cleaning (clean_telco())
- Centralized path management to avoid hardcoding
- Automated visual generation saved to a /visuals/ folder
- Final summary printout to confirm success at every stage

The script handles edge cases (e.g., missing file, null columns) and can be re-executed across environments without major edits.

6. Consider Ethical Implications

Even though our dataset is synthetic, it reflects real customer categories.

Ethical considerations include:

- Bias in features: Fields like gender, senior citizen status, and tenure could be proxies for age or social class. These could bias the model unfairly if not handled carefully.
- Lack of consent: Although the data is public, using such structure without actual opt-in from customers can lead to poor privacy norms in real-world applications.
- Predictive misuse: Models trained on churn might be used not just for retention but for exclusion (e.g., denying discounts to high-risk customers).
- Imbalanced labels: Overfitting to churned customers can propagate bias in marketing strategies.

In our project, we mitigate these by documenting all preprocessing logic, avoiding sensitive personal fields for modeling, and focusing on explainable analytics first.

7. Ethics Report

Data Ownership and Consent:

The dataset does not contain any customer names or contact information, which helps us stay compliant with GDPR and CCPA guidelines. However, if this were a real telecom system, explicit consent and opt-in records would be necessary for lawful usage (Mittelstadt et al, 2016).

Privacy-by-Design:

Our SQLite storage is local and not pushed to cloud platforms, meaning no unauthorized access risks exist in this phase. If the project were to scale, we'd implement access controls, encryption at rest, and user-level anonymization.

Fairness and Transparency:

Customer churn models often carry the risk of being "black boxes." Our visualizations (tenure, charges, contract types) were selected to communicate model-relevant features early in the pipeline. This ensures human-in-the-loop verification before any ML modeling is deployed.

Bias Mitigation:

For example, longer contracts naturally show lower churn, not because of satisfaction but because of binding terms. This insight should lead to fairer segmentation and not penalizing month-to-month customers in loyalty programs.

8. Visualizations

We included five key visualizations to uncover early insights before modeling:

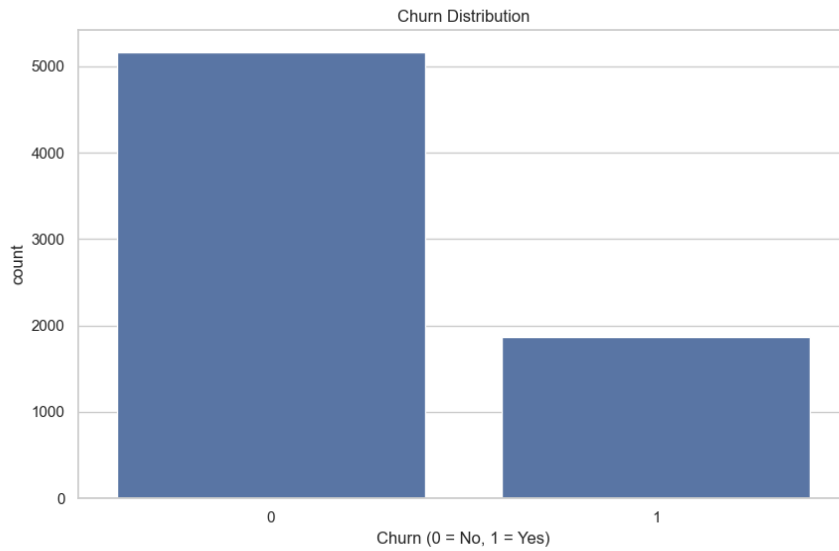


Figure 2.1: Churn Distribution

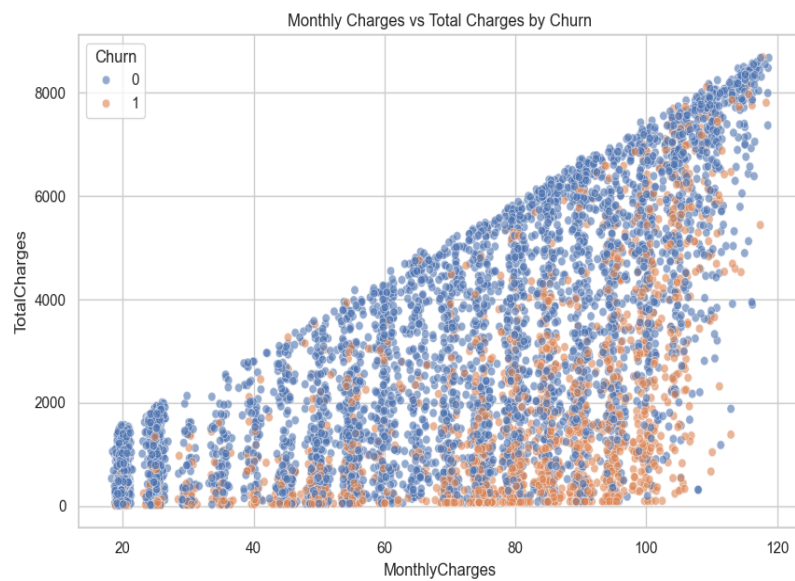


Figure 2.2: Monthly vs Total Charges by Churn

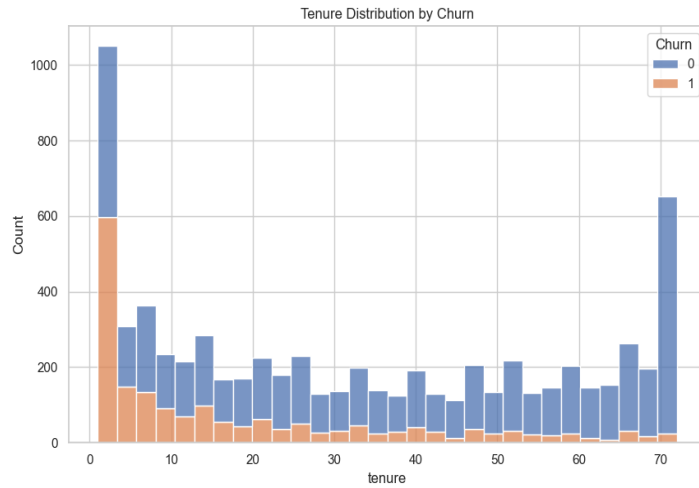


Figure 2.3: Tenure Distribution by Churn

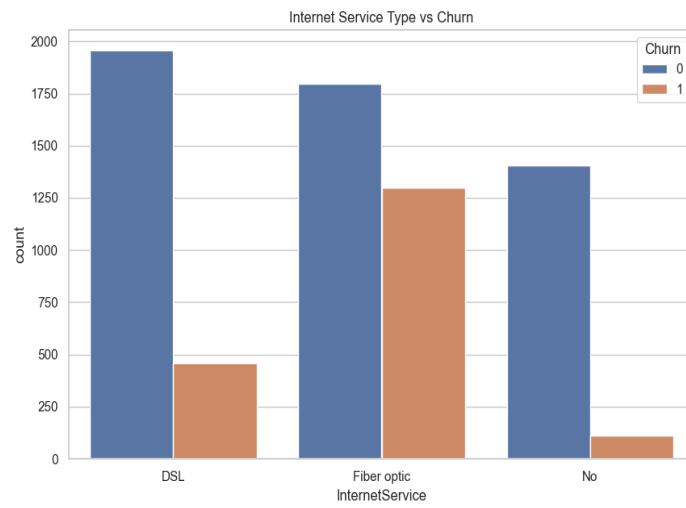


Figure 2.4: Internet Service vs Churn

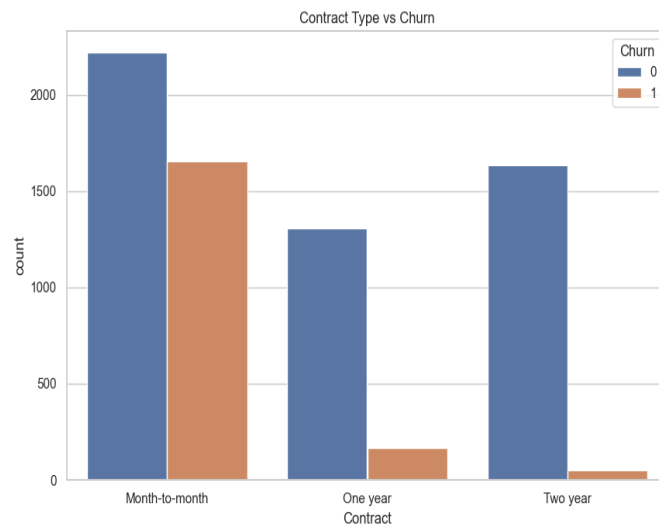


Figure 2.5: Contract Type vs Churn

9. Conclusion

This module provided a full-stack simulation of real-world data engineering for churn modeling. We identified a valid dataset, set up local ingestion and cleaning, stored results in a reusable format, and produced visual insights for decision-makers. Ethical analysis reinforced that even well-structured pipelines must be governed by fairness, consent, and clarity.

References

- Ameisen, E. (2020). Building Machine Learning Powered Applications (1st ed.). O'Reilly Media, Inc. <https://www.oreilly.com/library/view/building-machine-learning/9781492045106/>
- Blastchar. (2018). Telco Customer Churn. Kaggle. <https://www.kaggle.com/datasets/blastchar/telco-customer-churn>
- Mittelstadt, B. D., et al. (2016). The ethics of algorithms: Mapping the debate. Big Data & Society, 3(2). <http://dx.doi.org/10.1177/2053951716679679>