**Customer Churn Prediction Pipeline – Detailed Project Documentation**

---

## 1. Problem Formulation

**Business Problem:**
Customer churn is a major challenge for telecom companies. Losing a customer incurs revenue loss and increases acquisition costs for new customers. The goal of this project is to predict which customers are likely to churn so that the company can implement retention strategies proactively.

**Business Objectives:**

- Reduce churn rate by identifying high-risk customers.
- Allocate resources for targeted retention campaigns.
- Improve customer lifetime value (CLV) through actionable insights.

**Key Data Sources and Attributes:**

1. **CSV Dataset:** WA_Fn-UseC_-Telco-Customer-Churn.csv
    - Columns include customerID, gender, SeniorCitizen, tenure, MonthlyCharges, TotalCharges, Churn, and other demographic & service-related features.
2. **API Data Source:** JSON endpoint at https://raw.githubusercontent.com/AnalyticsKnight/youtube/master/data/telecom_churn.json
    - Contains similar features as CSV, providing an alternative ingestion path.

**Expected Pipeline Outputs:**

- **Clean datasets** ready for EDA and preprocessing.
- **Transformed and feature-engineered datasets** suitable for machine learning.
- **A deployable churn prediction model** with versioning and performance reports.

**Evaluation Metrics:**

- Accuracy, Precision, Recall, F1 score, ROC-AUC.

**Deliverables:**

- PDF/Markdown document outlining business problem, objectives, data sources, pipeline outputs, and evaluation metrics.

---

**2. Data Ingestion**

**Implementation:**

- **Scripts:** src/ingestion/ingest.py
- **Sources ingested:** CSV file and REST API.
- **Mechanism:**
    - CSV ingestion reads the file from local storage.
    - API ingestion fetches JSON data and converts it to a DataFrame.
- **Storage:** Data saved to raw_data/ with timestamped filenames.

**Error Handling and Logging:**

- Network errors during API ingestion are caught and logged.
- Logging tracks successful ingestion events in ingestion.log.

**Deliverables:**

- Python ingestion scripts.
- Log file demonstrating ingestion success.
- Screenshots of ingested data in raw_data/.

---

**3. Raw Data Storage**

**Implementation:**

- **Storage type:** Local filesystem as a pseudo data lake.
- **Folder structure:**

raw_data/

raw_churn_csv_<timestamp>.csv

raw_churn_api_<timestamp>.csv

- Enables **easy partitioning by source and timestamp**.

**Deliverables:**

- Documentation describing folder structure.
- Python code demonstrating ingestion and storage.

---

**4. Data Validation**

**Implementation:**

- **Script:** src/validation/validate.py
- **Validation checks:**
    - Missing values
    - Duplicated rows
    - Data type validation
    - Outlier detection using z-scores
    - Simple range checks for numeric values

**Reporting:**

- Generates PDF report: data_quality_report_<timestamp>.pdf
- Summarizes issues, including counts of missing values, duplicates, and outliers.

**Deliverables:**

- Python validation script.
- Sample PDF/CSV data quality report.

---

**5. Data Preparation**

**Implementation:**

- **Script:** src/preprocessing/preprocess.py
- **Steps performed:**
    - Numeric missing values imputed with mean.

    - Categorical missing values imputed with mode.
    - One-hot encoding for categorical variables.
    - Standardization of numeric features using StandardScaler.
- **Rationale:** Standardization improves model convergence and one-hot encoding allows tree and linear models to process categorical data efficiently.

**Exploratory Data Analysis (EDA):**

- Summary statistics, histograms, and boxplots to understand distributions, outliers, and trends.

**Deliverables:**

- Python preprocessing scripts.
- Clean datasets (clean_churn.csv) ready for feature engineering.
- Visualizations and summary statistics for EDA.

---

**6. Data Transformation and Storage**

**Implementation:**

- **Script:** src/feature_engineering/features.py
- **Feature engineering performed:**
    - total_spend = MonthlyCharges * tenure (captures total revenue from customer)
    - tenure_years = tenure / 12 (numeric scaling for ML algorithms)
    - long_term_customer = (tenure > 24) (binary indicator for retention)
- **Storage:** SQLite database transformed_churn.db for feature persistence.
- Enables easy querying and retrieval for model training.

**Deliverables:**

- SQL schema automatically created by SQLite.
- Sample queries to fetch transformed features.
- Summary of transformation logic.

---

**7. Feature Store**

**Implementation:**

- **Script:** src/feature_store/export.py
- **Mechanism:**
    - Export features from SQLite to transformed_churn.csv for use in training or deployment.
    - Timestamp metadata event_timestamp added to each record.

**Sample Queries:**

```
# CSV retrieval

import pandas as pd

df = pd.read_csv("transformed_churn.csv")

high_value_customers = df[df["total_spend"] > 1000]


# SQLite retrieval

import sqlite3, pandas as pd

con = sqlite3.connect("transformed_churn.db")

df = pd.read_sql_query("SELECT * FROM customer_features WHERE total_spend > 1000", con)

con.close()
```

**Deliverables:**

- Feature store CSV and SQLite examples.
- Metadata capturing feature source, version, and timestamp.

---

**8. Data Versioning**

**Implementation:**

- **Script:** src/versioning/dvc_versioning.py

- **Tools:** DVC + Git
- **Versioning approach:**
    - Raw datasets and cleaned datasets are tracked.
    - Metadata files store source, timestamp, and change description.

**Deliverables:**

- DVC-tracked GitHub repository.
- Documentation of versioning strategy and workflow.

---

**9. Model Building**

**Implementation:**

- **Script:** src/modeling/train.py
- **Models trained:** Logistic Regression, Random Forest, Gradient Boosting, SVM, Decision Tree, KNN, XGBoost.
- **Evaluation:** Accuracy, Precision, Recall, F1 score, ROC-AUC.
- **Best model selection:** By highest F1 score.
- **Versioning:** MLflow logs model, parameters, and metrics.

**Deliverables:**

- Python training script.
- Model performance reports (.txt and .csv).
- Versioned saved model (models/<best_model_name>_churn_model.pkl).

---

**10. Orchestrating the Data Pipeline**

**Implementation:**

- **Orchestration tool:** Apache Airflow
- **DAG:** dags/churn_pipeline_dag.py
- **Task flow:**

    - Ingestion → Validation → DVC Versioning → Preprocessing → Feature Engineering → Feature Store Export → Model Training

- **Monitoring & Logging:** Airflow UI for task monitoring; ingestion.log and modeling.log for step-level logging.
- **Retries:** Each task can retry once on failure.

**Deliverables:**

- DAG script demonstrating automation.
- Logs and screenshots of successful DAG runs (Airflow UI).

---

**Additional Features and Considerations**

- Modularity: Each stage has a dedicated script.
- Logging: Ingestion, preprocessing, and modeling stages logged separately.
- Error Handling: Try-except blocks implemented where necessary.
- Documentation: This PDF describes pipeline design, data sources, transformations, feature engineering, and model evaluation.
- Video Demonstration: Recommended 5–10 minute walkthrough showing end-to-end pipeline execution.