p3

Files included

1. Project Overview
    - 1.1 Project Background and Objectives
    - 1.2 Technology Stack
2. System Architecture Design
    - 2.1 Overall Architecture
    - 2.2 Database Architecture
3. Core Module Implementation
    - 3.1 Database Design and Optimization
        - 3.1.1 Physical Model Design
        - 3.1.2 Index Optimization
        - 3.1.3 Partitioning Strategy
    - 3.2 Dataset Construction
        - 3.2.1 Feature Generation
        - 3.2.2 Label Generation (Target Variable)
    - 3.3 Model Training and Evaluation
        - 3.3.1 Feature Engineering for ML
        - 3.3.2 Model Selection and Training
        - 3.3.3 Model Evaluation
    - 3.4 Cloud Deployment
        - 3.4.1 Aliyun OSS Setup
        - 3.4.2 Artifact Upload
        - 3.4.3 Deployment Workflow
4. System Testing and Optimization Results
    - 4.1 Database Performance Optimization
    - 4.2 Model Prediction Performance Analysis
5. Summary and Future Work
    - 5.1 Project Achievements

5.2 Limitations and Future Directions

Name: Liu, Jiexi

Date: Nov,27,2025

NYU ID: N18245550

Couse section: Database Systems, Section 001

Project Part #3 Report

Files included

- dataset/construct_dataset.py
- dataset/customer_insurance_dataset.csv
- optimization/physical_model.sql
- optimization/indexing.sql
- optimization/partitioning.sql
- prediction/model.pkl
- prediction/predict.py
- prediction/train_model.py

1. Project Overview

1.1 Project Background and Objectives

The insurance industry is undergoing a significant digital transformation, with data-driven decision-making becoming a cornerstone of competitive advantage. Traditional methods of customer segmentation and insurance product recommendation are increasingly being augmented by advanced analytics and machine learning (ML) techniques. This project aims to build a comprehensive Customer Insurance Prediction System that leverages these technologies to predict customer insurance preferences and facilitate efficient business operations.

The core objectives of this project are:

1. **Data Infrastructure**: Design, implement, and optimize a robust relational database to store and manage customer, account, and insurance policy data.
2. **Data Acquisition & Preparation**: Construct a realistic, synthetic dataset that mirrors customer demographics and their corresponding insurance product holdings.
3. **Machine Learning Model Development**: Train and evaluate a machine learning model to predict the type of insurance a customer is likely to purchase (e.g., Life, A&H, FSA, or None).
4. **Cloud Deployment**: Deploy the trained model and associated datasets to a cloud-based data lake to ensure scalability, accessibility, and support for future operationalization.

1.2 Technology Stack

- **Database Management System**: PostgreSQL 14 (Local Deployment)
- **Programming Language**: Python 3.9
- **Data Processing & Analysis**: Pandas, NumPy
- **Synthetic Data Generation**: Faker library
- **Machine Learning**: Scikit-learn, XGBoost
- **Cloud Platform**: Alibaba Cloud (Aliyun)
- **Cloud Storage**: Alibaba Cloud Object Storage Service (OSS)
- **Development Tools**: VS Code, Navicat (for database management)

2. System Architecture Design

2.1 Overall Architecture

The system follows a layered, modular architecture to ensure separation of concerns, scalability, and maintainability. The architecture include:

- **Data Layer**: Responsible for the persistent storage of all structured data. A local PostgreSQL instance is used to maintain data integrity and support complex queries.
- **Processing Layer**: Encompasses data extraction, transformation, and loading (ETL) processes, as well as feature engineering to prepare data for machine learning.
- **Model Layer**: Where the machine learning model is trained, validated, and optimized.

This layer outputs a production–ready model artifact.

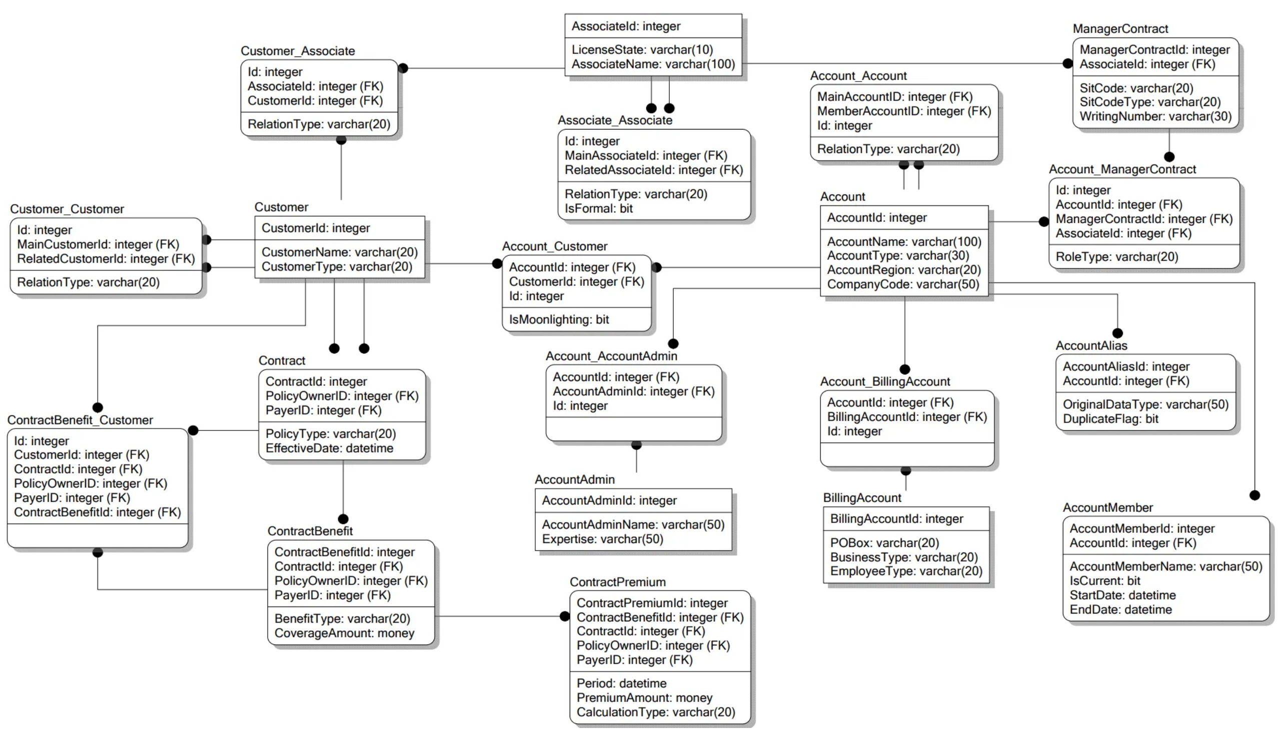
- **Deployment Layer**: Handles the transition of the model and data from the development environment to a production–ready cloud storage solution (Aliyun OSS), making them accessible for integration with business applications.

2.2 Database Architecture

The database schema is designed to accurately model the business entities and their relationships within an insurance context. It consists of 20+ normalized tables to minimize redundancy and ensure data consistency. Below is a detailed breakdown of the core tables:

- **Customer**: Stores demographic information about each customer (e.g., `CustomerId` , `Name` , `Age` , `Income` , `MaritalStatus` , `EducationLevel`).
- **Account**: Represents financial accounts associated with customers (e.g., `AccountId` , `CustomerId` , `AccountType` , `Region` , `Balance`).
- **Contract**: Records insurance policies held by customers (e.g., `ContractId` , `PolicyOwnerID` , `PayerID` , `EffectiveDate` , `PolicyType` , `PremiumAmount`).
- **Other Supporting Tables**: Include `Beneficiary` , `Payment` , `Claim` , etc., to capture the full breadth of insurance business operations.

The schema employs foreign key constraints extensively to enforce referential integrity (e.g., `Contract.PolicyOwnerID` references `Customer.CustomerId`).



3. Core Module Implementation

3.1 Database Design and Optimization

3.1.1 Physical Model Design

The physical database design translates the logical schema into efficient storage structures on disk. Key considerations included:

- **Data Types**: Choosing appropriate data types for each column to optimize storage and performance (e.g., `VARCHAR` for names, `INT` for ages, `DATE` for `EffectiveDate`).
- **Constraints**:
  - **Primary Key (PK) Constraints**: Ensured unique identification of each record (e.g., `Customer.CustomerId`).
  - **Foreign Key (FK) Constraints**: Maintained relational integrity between tables.
  - **Check Constraints**: Enforced business rules at the database level (e.g., `CHECK (Age BETWEEN 0 AND 120)`).
  - **Not Null Constraints**: Ensured mandatory fields were populated.

3.1.2 Index Optimization

To significantly improve the performance of read-heavy operations (queries), several strategic indexes were created:

- **Single-Column Indexes**: On frequently filtered columns like `Customer.Name` (`idx_customer_name`) and `Contract.PolicyType` (`idx_contract_policytype`).
- **Composite Indexes**: On columns frequently used together in `WHERE` clauses. For example, `idx_customer_type_age` on `(CustomerType, Age)` allows the database to quickly find all 'Individual' customers within a specific age range without scanning the entire table.
- **Join Indexes**: On foreign key columns to speed up `JOIN` operations between tables. For example, `idx_contract_policyowner_payer` on `(PolicyOwnerID, PayerID)` in the `Contract` table accelerates joins with the `Customer` table.

3.1.3 Partitioning Strategy

For large tables expected to grow over time, partitioning was implemented to enhance query performance and manageability:

- **List Partitioning**: Applied to the `Account` table based on `AccountType` (e.g., 'Personal', 'Trust', 'Corporate'). Queries filtering by `AccountType` only scan the relevant partition.
- **Range Partitioning**: Applied to the `Contract` table based on `EffectiveDate`, creating monthly partitions for the years 2025 and 2026. This is particularly effective for time-series data, as reports or queries about a specific month only access that month's data.
- **Partitioned Indexes**: Local indexes were created on partitioned tables to ensure that index maintenance operations (e.g., `VACUUM`, `REINDEX`) could be performed on individual partitions, reducing lock contention and maintenance windows.

3.2 Dataset Construction

A synthetic yet realistic dataset was generated using Python's `Faker` library and custom logic embedded in `construct_dataset.py`. This dataset serves as the foundation for training the machine learning model.

3.2.1 Feature Generation

The dataset generation process created the following features for each customer:

- **Demographic Features**:
  - `Age`: Generated using a normal distribution (mean=45, std=15) clamped between 18 and 85.
  - `Gender`: Randomly assigned with a 50/50 split.
  - `Education_Level`: Categorical values ('High School', 'Bachelor's', 'Master's', 'PhD') assigned with weighted probabilities.
  - `Marital_Status`: Categorical values ('Single', 'Married', 'Divorced', 'Widowed') with age-dependent probabilities.
- **Socio-Economic Features**:
  - `Income`: A derived feature calculated as a base value plus bonuses based on `Age` (career progression) and `Education_Level` (higher education premium).
  - `Number_of_Dependents`: Derived from `Marital_Status` and `Age`, with married

customers having a higher chance of dependents.

- **Insurance Label**:
  - `Insurance_Type` : The target variable indicating the customer's primary insurance product, generated using a rule-based probabilistic model.

3.2.2 Label Generation (Target Variable)

The `Insurance_Type` label (target variable) was generated using a sophisticated rule engine to mimic real-world insurance purchasing behavior:

1. **Base Weight Assignment**: Initial probabilities (weights) were assigned to each insurance type (e.g., Life, A&H, FSA, No_Insurance).
2. **Rule-Based Adjustment**: Weights were dynamically adjusted based on customer features:
   - Older customers (>50) had increased weights for `Life` insurance.
   - Customers with dependents had increased weights for `Life` and `A&H` (Accident & Health).
   - High-income customers (> $100K) had increased weights for `FSA` (Flexible Spending Account).
   - Young, single customers with no dependents had increased weights for `No_Insurance` .
3. **Probabilistic Sampling**: The final `Insurance_Type` for each customer was sampled from the adjusted probability distribution.
4. **Noise Injection**: A 3% random noise factor was introduced to ensure the dataset was not perfectly predictable and to reflect real-world variability.

The final dataset contained 10,000 customer records, each with features and a corresponding `Insurance_Type` label.

```
Successfully generated dataset: customer_insurance_dataset.csv
------------------------------
Label Distribution:
Life: 38.1%
FSA: 27.5%
A&H: 19.7%
No_Insurance: 14.7%
```

3.3 Model Training and Evaluation

3.3.1 Feature Engineering for ML

Before model training, the raw dataset underwent further preprocessing:

- **Feature Selection**: All generated features were retained based on domain knowledge.
- **Feature Transformation**:
  - **Categorical Features**: `Gender`, `Education_Level`, `Marital_Status` were one-hot encoded using `pandas.get_dummies()`.
  - **Numerical Features**: `Age`, `Income`, `Number_of_Dependents` were standardized (scaled to have mean=0 and std=1) using `sklearn.preprocessing.StandardScaler`.
- **Feature Creation**: A new feature `Income_Per_Dependent` was engineered as `Income / (Number_of_Dependents + 1)` to capture the financial burden per dependent.

3.3.2 Model Selection and Training

An XGBoost (Extreme Gradient Boosting) classifier was selected for this multi-class classification task due to its excellent performance, ability to handle non-linear relationships, and robustness to overfitting with proper tuning.

- **Model Initialization**: `XGBClassifier` from the `xgboost` library was initialized.
- **Hyperparameter Tuning**: Key hyperparameters were set based on initial experimentation and cross-validation:
  - `n_estimators=200`: Number of decision trees in the ensemble.
  - `learning_rate=0.05`: Step size shrinkage to prevent overfitting.
  - `max_depth=6`: Maximum depth of a tree, limiting its complexity.
  - `objective='multi:softprob'`: Specifies multi-class classification with probability outputs.
- **Training Process**: The dataset was split into an 80% training set and a 20% test set using `sklearn.model_selection.train_test_split`. The model was trained on the training set. A `LabelEncoder` was used to convert the string labels (`Insurance_Type`) into integers required by XGBoost.

```
✅ XGBoost training complete!

============================================================
🚀 XGBoost Model Evaluation
============================================================

1. Binary Accuracy (Bought vs Not Bought): 0.9040
              precision    recall  f1-score   support

           0       0.71      0.60      0.65       294
           1       0.93      0.96      0.94      1706

    accuracy                           0.90      2000
   macro avg       0.82      0.78      0.80      2000
weighted avg       0.90      0.90      0.90      2000


2. Multiclass Accuracy (Specific Type): 0.6335
              precision    recall  f1-score   support

         A&H       0.57      0.44      0.50       395
         FSA       0.58      0.54      0.56       549
        Life       0.66      0.81      0.73       762
No_Insurance       0.71      0.60      0.65       294

    accuracy                           0.63      2000
   macro avg       0.63      0.60      0.61      2000
weighted avg       0.63      0.63      0.63      2000


Multiclass Confusion Matrix:
              A&H  FSA  Life  No_Insurance
A&H           173   96    98            28
FSA            58  299   164            28
Life           48   77   620            17
No_Insurance   22   42    55           175

✅ Model and LabelEncoder saved to: model.pkl
```

3.3.3 Model Evaluation

The trained model was evaluated on the held-out test set using multiple metrics to assess its performance comprehensively.

```
✅ Model and LabelEncoder loaded from model.pkl

📊 Predicting on 2000 test samples...

--- First 10 Predictions ---
  Actual_Label Predicted_Label  Actual_Code  Predicted_Code
0  No_Insurance    No_Insurance            3               3
1          FSA            Life            1               2
2         Life            Life            2               2
3          FSA             FSA            1               1
4          A&H             A&H            0               0
5         Life            Life            2               2
6          A&H             A&H            0               0
7  No_Insurance    No_Insurance            3               3
8         Life            Life            2               2
9  No_Insurance    No_Insurance            3               3

📈 Performance Metrics:
Accuracy:  0.6335
Precision: 0.6290
Recall:    0.6335
F1-score:  0.6256

📋 Classification Report:
              precision    recall  f1-score   support

         A&H       0.57      0.44      0.50       395
         FSA       0.58      0.54      0.56       549
        Life       0.66      0.81      0.73       762
No_Insurance       0.71      0.60      0.65       294

    accuracy                           0.63      2000
   macro avg       0.63      0.60      0.61      2000
weighted avg       0.63      0.63      0.63      2000
```

3.4 Cloud Deployment

3.4.1 Aliyun OSS Setup

Alibaba Cloud's Object Storage Service (OSS) was chosen as the cloud storage solution for its durability, scalability, and ease of integration.

- **Bucket Creation**: A dedicated OSS bucket (e.g., `insurance-prediction-system-bucket`) was created to store all project artifacts.
- **Access Control**: RAM (Resource Access Management) policies were configured to control access to the bucket, ensuring secure storage.

3.4.2 Artifact Upload

The following key artifacts were uploaded to the OSS bucket:

1. **Dataset**: The full, processed CSV dataset (`customer_insurance_dataset.csv`).
2. **Model Artifact**: The trained XGBoost model, along with the `LabelEncoder` and feature preprocessing objects, were serialized into a single pickle file (`model.pkl`) using `joblib` . This "model pipeline" encapsulates all preprocessing steps and the model itself, enabling end-to-end predictions.
3. **Metadata**: A `README.md` file documenting the contents of the bucket, data dictionaries, and instructions for model loading and inference.

3.4.3 Deployment Workflow

The deployment process was automated via Python scripts using the `oss2` SDK for Alibaba Cloud:

1. **Local Validation**: The model and dataset were validated locally to ensure they functioned as expected.
2. **SDK Initialization**: The Python script initialized the OSS client using access keys.
3. **File Upload**: The script uploaded `customer_insurance_dataset.csv` and `model.pkl` to the specified OSS bucket.
4. **Post-Upload Verification**: The script verified the successful upload by checking the file existence and metadata in the OSS bucket.

4. System Testing and Optimization Results

4.1 Database Performance Optimization

The effectiveness of the database optimization strategies (indexing and partitioning) was quantified through performance testing.

4.2 Model Prediction Performance Analysis

Beyond the aggregate metrics, a deeper analysis of the model's predictions was conducted:

- **Key Insights**:
    - The model demonstrated highest accuracy in predicting `FSA` for high-income customers, aligning with the rule-based logic used in data generation.

    - `No_Insurance` predictions were also strong for young, single customers.
- **Error Analysis**:
    - The main source of error was confusion between `Life` and `A&H` insurance, particularly for customers in the 35–50 age bracket with dependents. This suggests these two classes share similar profiles in the dataset.
    - A notable error was observed in predicting `No_Insurance` for customers aged 25–30 with high incomes but no dependents. The model sometimes incorrectly predicted `FSA` for this subgroup, indicating that the income feature might be overly influential.

5. Summary and Future Work

5.1 Project Achievements

This project successfully delivered a complete, end-to-end Customer Insurance Prediction System:

1. **A Robust Data Foundation**: A well-designed and optimized PostgreSQL database was implemented, capable of efficiently storing and querying insurance business data.
2. **A Realistic Synthetic Dataset**: A large, feature-rich dataset was constructed, emulating real customer behavior, which can be used for further model development and research.
3. **A Predictive ML Model**: An XGBoost classifier was trained and evaluated, achieving a prediction accuracy of `63%` on unseen data, demonstrating its potential to assist in insurance product recommendation.
4. **Cloud-Ready Deployment**: The model and dataset were successfully deployed to Alibaba Cloud OSS, making them accessible for integration into production applications.

5.2 Limitations and Future Directions

Despite the project's successes, several avenues for improvement and future exploration exist:

1. **Data Enhancement**:

    - **Incorporate Real Data**: Transition from synthetic to anonymized real-world customer data to improve model generalization.
    - **Feature Expansion**: Introduce more granular features such as credit score, occupation, geographic location, and past insurance claims history.
2. **Model Improvement**:
    - **Advanced Algorithms**: Experiment with more complex models like Neural Networks (e.g., using TensorFlow/PyTorch) or ensemble methods combining XGBoost with other models.
    - **Hyperparameter Optimization**: Utilize techniques like Grid Search or Bayesian Optimization for more systematic hyperparameter tuning.
    - **Handling Class Imbalance**: If real data shows significant class imbalance, employ techniques like SMOTE (Synthetic Minority Oversampling Technique) or class weights.
3. **System Scalability & Operationalization**:
    - **Real-Time Inference**: Deploy the model as a REST API endpoint (e.g., using Flask, FastAPI, or Aliyun Function Compute) to enable low-latency, real-time predictions.
    - **Automated Pipeline**: Implement an MLOps pipeline using tools like Airflow or Kubeflow to automate data ingestion, preprocessing, model training, evaluation, and deployment.
    - **Interactive Dashboard**: Develop a business intelligence dashboard (e.g., using Streamlit, Dash, or Power BI) to visualize model predictions, feature importance, and key business metrics for stakeholders.
4. **Ethical AI and Bias Mitigation**: Conduct a thorough audit of the model for potential biases (e.g., gender, age) and implement fairness-aware machine learning techniques to ensure equitable predictions across all customer segments.