DSCI-352 Final Project

Lizzy Chen, David Wang, Harley Chen
Github Repository: https://github.com/wangyanzhao0423/DSCI352_final

Executive Summary

The two projects demonstrate the application of supervised machine learning across two distinct data domains—structured tabular data and time-series financial data. The bonus Part 3 further includes deploying the trained model from Part 1 onto AWS and creating a Streamlit application. This architecture allows a local frontend to accept user input and invoke the cloud backend to generate predictions.

- **Part 1 - Telco Customer Churn Prediction:** We constructed a set of binary classification models that help the providers identify customers who may cancel. By utilizing One hot encoding and MinMax scaling of demographic and account data, we fed four Scikit-Learn models and a Keras Multilayer Perceptron (MLP). The best-performing algorithm with respect to AUC is Gradient Boosting Classifier with an AUC of 0.843. This shows a strong capability to differentiate between churners and non-churners capturing non-linear relationships between customers.

- **Part 2 - Bitcoin Price Forecasting:** We used an RNN (Recurrent Neural Network) using LSTM (Long Short-Term Memory) architecture on unstructured temporal data created to predict daily high prices for Bitcoin using RNNs. The model was trained on data from 2016-2022. It had a lookback window of 30 days. The root mean square error (RMSE) value of the model was ~ $2,644 on the test data (2023–present). The LSTM effectively exploited short-to-medium-term market trends without overfitting. This result supports our hypothesis that deep learning can be more effective than static tabular models for sequential regression tasks.

- **Part 3 - Automated AWS Pipeline:** To connect modeling and production, we deployed an automated inference pipeline on AWS. It is a serverless architecture using the AWS stack. We created a lightweight JSON artifact from the Telco Churn model and hosted it on AWS S3. Furthermore, we built a custom inference engine using AWS Lambda and API Gateway. The interface of the system is powered by a Streamlit Dashboard which permits the executives to conduct risk analysis on a single customer and even process the batch data sets to prove low latency and cost-effective machine learning deployment within a corporate system.

Part 1: Telco Customer Churn Prediction & Automated AWS Pipeline

Approach & Methodology:

The objective of Part 1 was to build, train, and evaluate multiple supervised machine learning models to predict customer churn (binary classification).

The dataset used in the project is Telco Customer Churn. It contains information about the customer's demographic, service subscribed, and account info. Before modeling, we first cleaned and manipulated the dataset to better conduct model training.

- The TotalCharges column was originally read as object type and was converted to numeric type.

- We used MinMaxScaler (feature_range=(0, 1)) to normalize all continuous numerical features (e.g., tenure, MonthlyCharges, TotalCharges). This scales the data to a similar range so that features with larger values do not unduly influence the model's loss function.
- One-hot encoding was applied to all binary (Yes/No) and multi-class categorical features e.g. (Contract, PaymentMethod, Gender etc.). This results in formation of new binary columns for each of the new categories as machine learning models and neural networks cannot take text inputs as such.

We first create 4 distinct Scikit-Learn models – Logistic Regression, Random Forest, Gradient Boosting, and SGDClassifier – and 1 Keras Multiplayer Perceptron (MLP). All models were then compared and evaluated using Area Under the Curve (AUC) and Accuracy.

For the MLP model, a 2-layer feed-forward neural network was built.

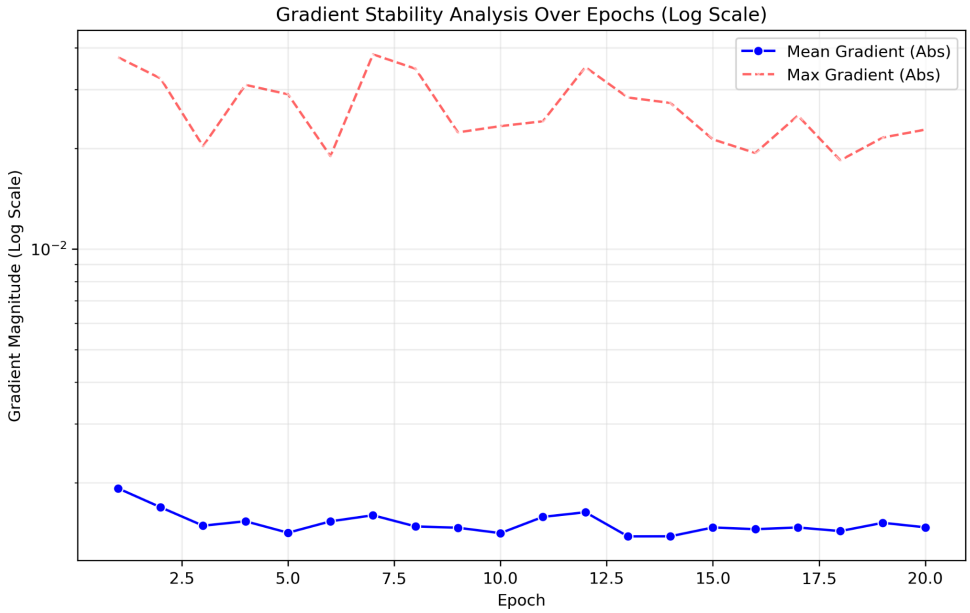
- Input Layer: Matches the number of features after one-hot encoding.
- Hidden Layer: Dense layer with 64 units and ReLU activation.
- Output Layer: Dense layer with 1 unit and Sigmoid activation.
- Optimizer: Adam (adaptive learning rate).
- Loss: BinaryCrossentropy.
- Epochs: 20 (with a validation split).
- Batch Size: 32.

Lastly, we incorporated a custom callback during training to monitor and log the average gradient magnitude of each layer per epoch. This diagnostic step was required to analyze model stability and interpret whether gradients have any vanishing or exploding behavior.

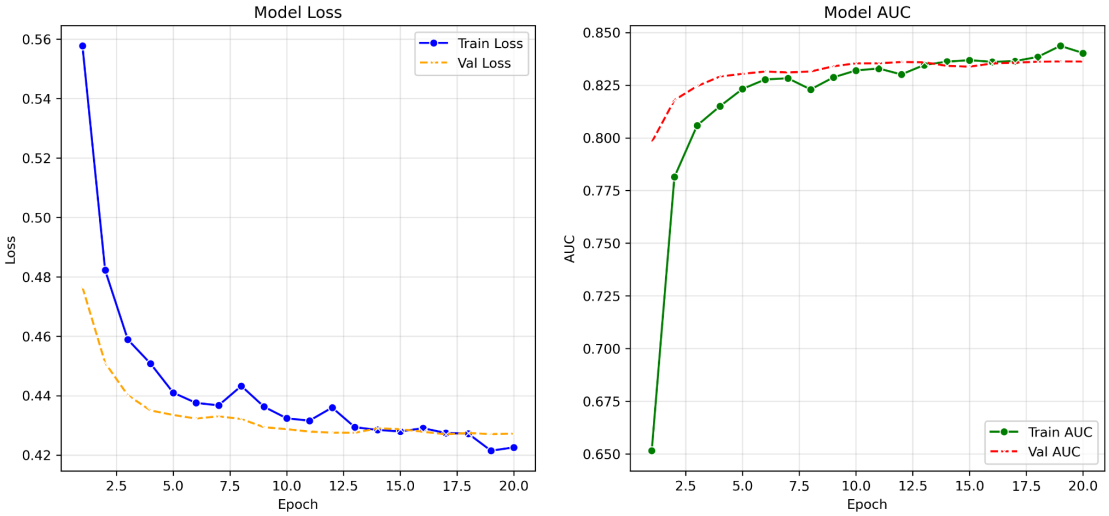
Results & Analysis:

Keras Multiplayer Perceptron (MLP)

Throughout all epochs of training, we tracked the mean absolute magnitudes of the gradients for each Keras MLP layer. The visualization is very important to check the “health” of the deep learning training process. The absence of vanishing gradients and exploding gradients is evident. The network is constantly capable of learning and updating weights at a consistent rate without saturation as the lines are stable.



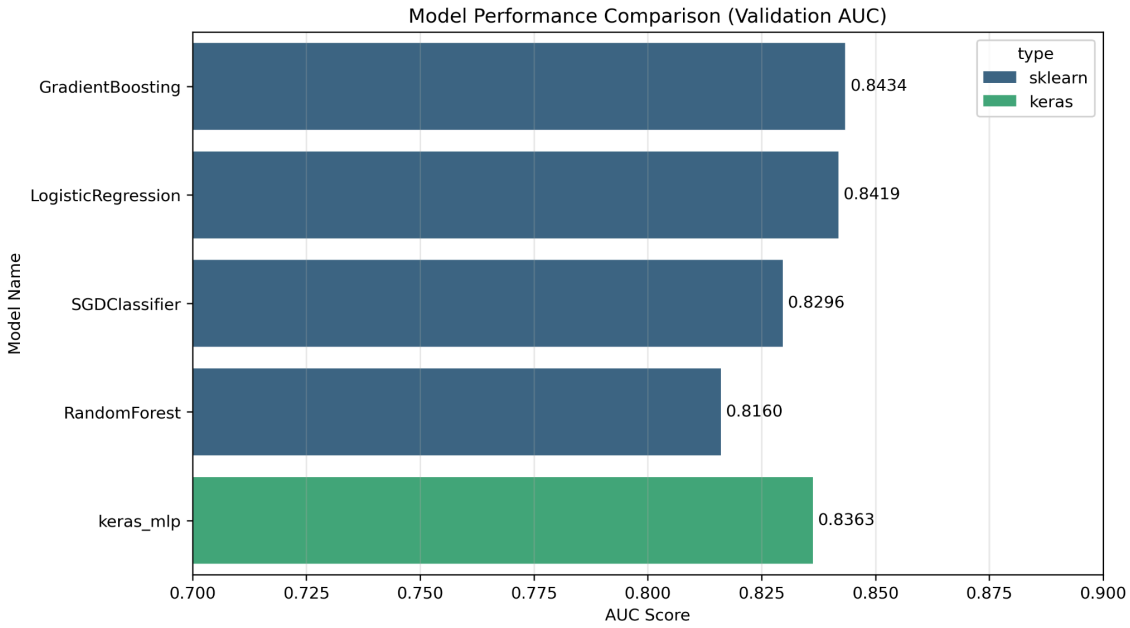
We also visualized the Training Loss vs. Validation Loss over the 20 epochs. This "Learning Curve" helps us identify underfitting or overfitting. During the process, the Validation Loss does not start to increase significantly. While Training Loss continues to drop.



Cross Model Comparison

We compared the validation AUC measure across the four models on the ROC. Because AUC is robust to class imbalance, it was selected as the primary metric. Furthermore, it measures the model's ability to rank a churning customer higher than a non-churning customer.

The performance of the Gradient Boosting Classifier was the best with AUC of 0.843. The superior performance of Gradient Boosting indicates that the customer characteristics and churn relationship has non-linear patterns which can be captured well using decision trees.



Outlook / Future Enhancements:

Our existing models are successful but there are a number of easy ways to improve them. Using our four base models (like Random Forest and Gradient Boosting) we could greatly improve performance by exploring new and larger hyperparameter ranges / implementing additional features. For instance, combining "Monthly Charges" and "Tenure" to get a "Total Lifetime Value". To make the Keras MLP (Neural Network) more stable and accurate, we can either add layers ("deeper") or train for more epochs with a smaller learning rate. It would be helpful to collect updated form fields from customers so that all models remain accurate.

Part 2: BTC Price Forecasting

Approach & Methodology:

We collected Bitcoin daily high prices from TradingView's Coinbase BTCUSD daily chart. The dataset spans 2016 to present and includes date, daily high, and other indicators. During preprocessing, we converted the date to pandas datetime, kept only the date and high columns, and dropped rows with N/A values to avoid chart issues.

The data was normalized using MinMaxScaler (feature_range=(0, 1)) to scale values between 0 and 1, which helps LSTM training. The dataset was split chronologically: training from 2019-01-01 to 2022-12-31, and testing from 2023-01-01 onward. We created sliding windows with a 30-day look back to predict the next day's price, which are sequences suitable for LSTM.

We used a two-layer LSTM:

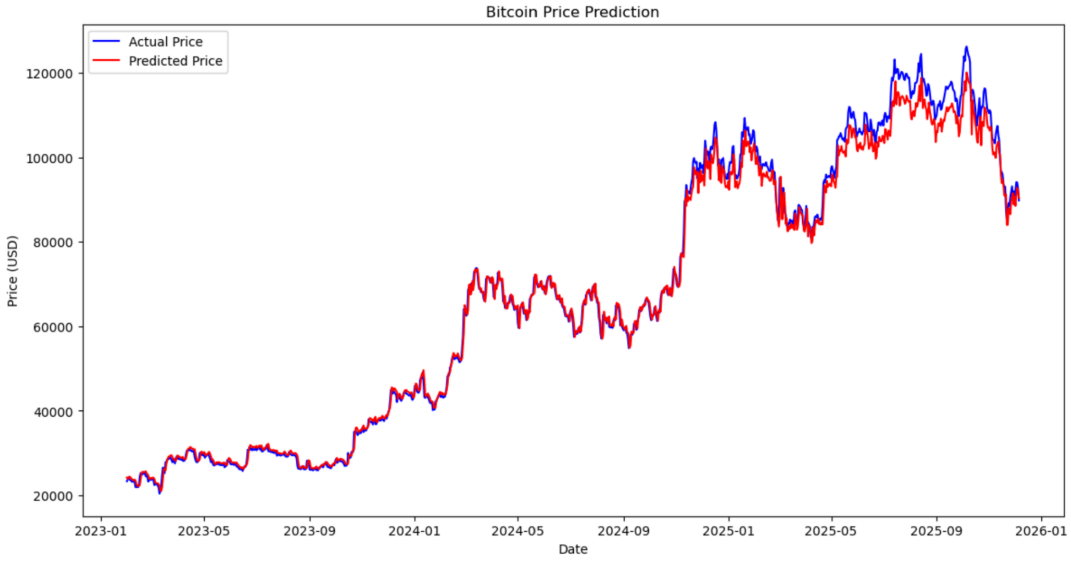
- First LSTM: 50 units, return_sequences=True
- Second LSTM: 50 units, return_sequences=False

- Dense output: 1 unit (regression)
- Optimizer: Adam (learning_rate=0.001)
- Loss: mean_squared_error
- Epochs: 20
- Batch size: 32

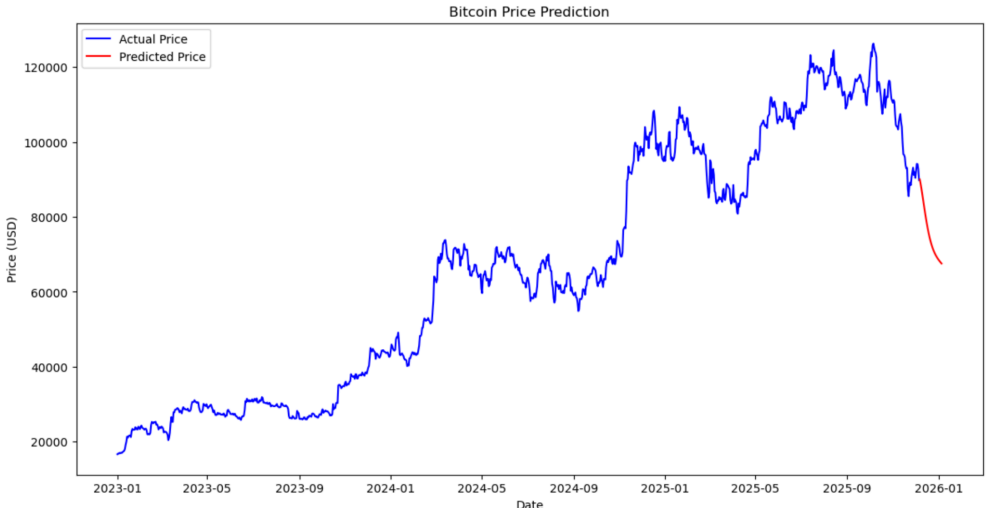
The architecture captures short- and medium-term patterns in Bitcoin prices.

Results & Analysis:

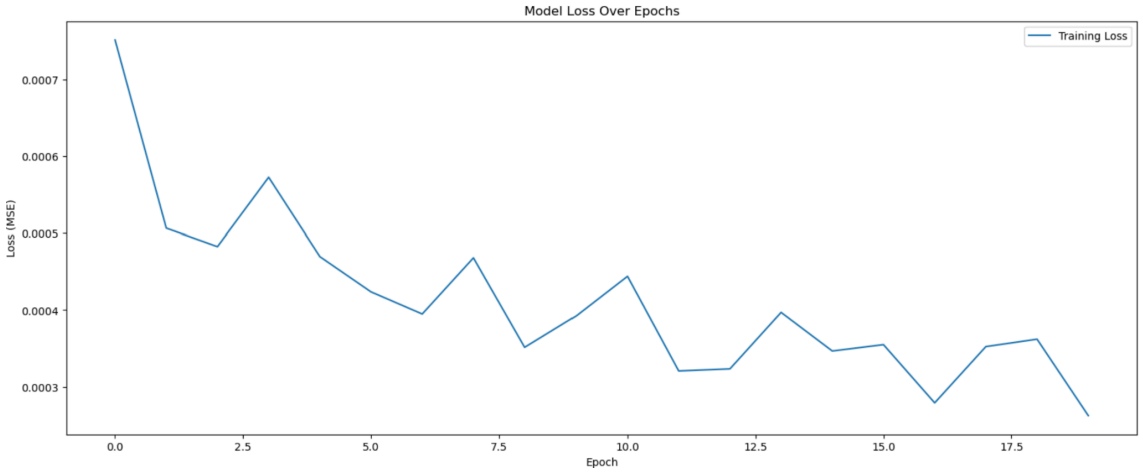
The model achieved a Root Mean Squared Error (RMSE) of approximately $2,644 on the test set (2023 onward). This indicates the average prediction error is within this range, which is reasonable given Bitcoin's volatility.



This is also an extra prediction to see what the price of the next 30 days would look like using the trained model and the past 30 days numbers.



The training loss curve shows a smooth, decreasing trend over 20 epochs with no signs of overfitting. The model learned temporal patterns without memorizing the training data.



Compared to tabular churn models:

- Data preparation: LSTM requires sequential sliding windows and chronological splits, while tabular models use independent rows with random splits.
- Architecture: LSTM captures temporal dependencies; tabular models learn static feature relationships.
- Evaluation: RMSE measures regression error in USD; AUC (used for churn) measures classification ranking.

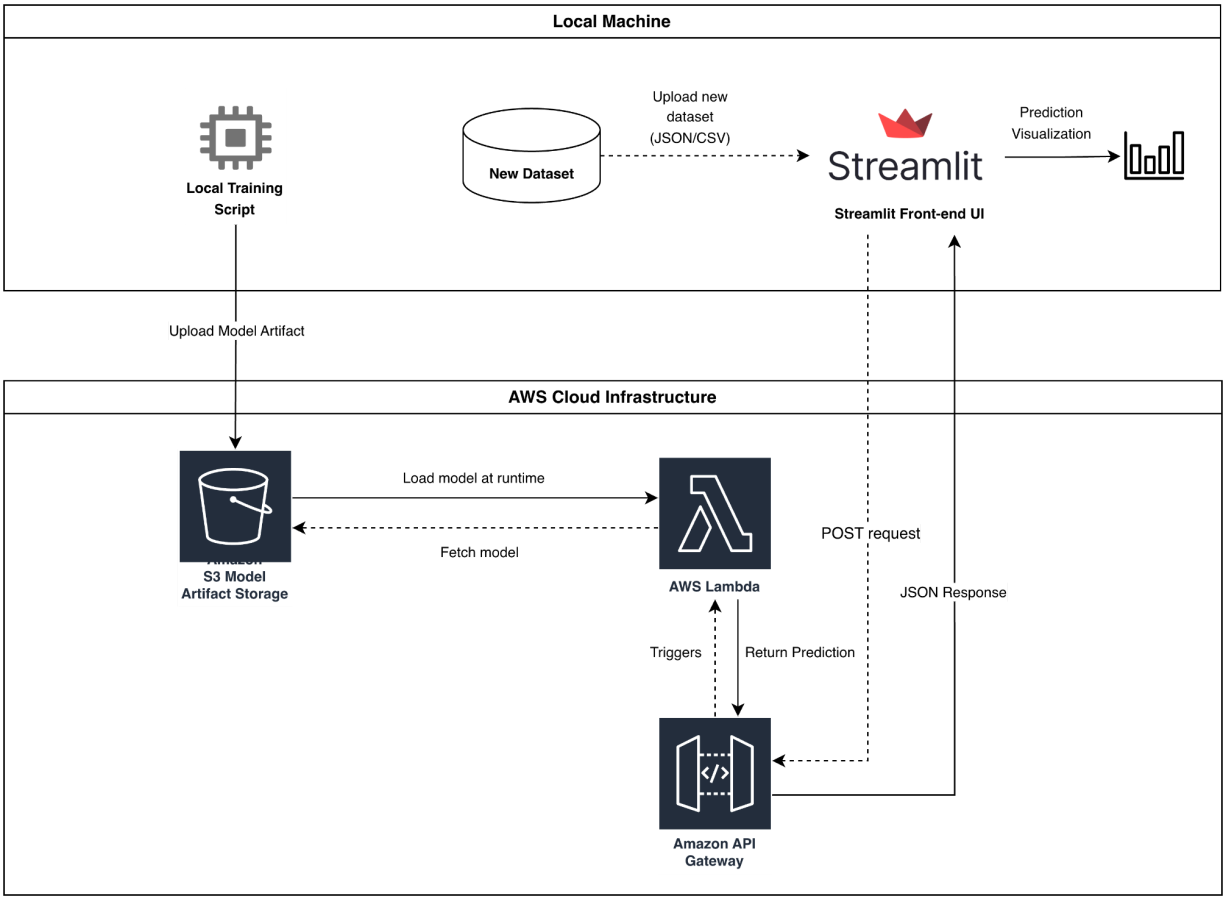
Outlook / Future Enhancements:

Within the current architecture, we could try to optimize for sequence length, layer count, units, learning rate, and batch size. Moreover, perhaps adding more technical indicators (RSI, MACD, Bollinger Bands), volume, and external factors (market sentiment, regulatory news) would make the model more accurate as it takes in consideration other than just the price. Furthermore, we can also completely change the architecture to a Transformer-based model to capture long-range dependencies as it is proven to be more efficient than vanilla LSTM.These enhancements could improve accuracy and make the model more practical for trading or risk management.

Part 3 Bonus: AWS Pipeline: Architecture, screenshots, workflow

Architecture Overview

The bonus objective was to deploy the trained Telco Churn model as an automated, end-to-end prediction system. We went with a Serverless Prediction Pipeline type of approach that hosts the inference logic on AWS and offers a Streamlit Web Dashboard.



System Workflow

The system workflow operates as follows:

1. User Input (Front-End): A user inputs the customer data into a Streamlit dashboard.
2. Request Submission: The dashboard sends the data to AWS API Gateway as a JSON POST payload.
3. Function Invocation: API Gateway triggers the AWS Lambda function.
4. Serverless Inference Logic
   - On the first invocation, Lambda downloads a lightweight model artifact from Amazon S3 and caches it in memory.
   - Subsequent predictions reuse the cached artifact to avoid repeated S3 calls and reduce latency.
   - Lambda performs manual preprocessing, one-hot encoding, scaling, and logistic prediction in pure Python, without external ML libraries.
5. Prediction Delivery: The JSON prediction returned by Lambda is displayed in real time on the Streamlit dashboard.

Steps to Reproduce

1. **Amazon S3 Model Registry**

- Create a private S3 bucket named dsci352-telco-churn-project in the us-east-1 region.
- This bucket serves as the model registry, hosting serialized model artifacts used by the inference service.
- Buckets remain private by default, preventing external access and ensuring the security of model parameters.

2. **IAM Authentication and Local Access**
   - Create a programmatic IAM user (telco-script-user) with the AmazonS3FullAccess policy.
   - Generate an Access Key ID and Secret Access Key, and configure them locally using aws configure.
   - The credentials allow the local training script to securely upload artifacts to the private S3 bucket without exposing public access.
3. **Model Training and Lightweight Serialization**
   - Execute the Telco churn training script to train an SGDClassifier with log_loss.
   - Extract the raw coefficients (weights) and preprocessing statistics (means, scales) into a lightweight JSON file.
   - Use the boto3 library to upload this JSON artifact to the S3 bucket automatically.
4. **AWS Lambda Deployment**
   - Create a Lambda function named DSCI352_PredictTelcoChurn using: Python 3.10 runtime & x86_64 architecture.
   - Implement a pure-Python inference script to avoid uploading libraries such as pandas or scikit-learn to Lambda. The function performs: manual one-hot encoding, scaling operations using stored stats, logit computation using model coefficients, sigmoid transformation to compute probability.
5. **Lambda IAM Execution Role**
   - Modify the Lambda execution role to include AmazonS3ReadOnlyAccess
   - This grants Lambda read-only access to the stored model artifact, without allowing file modification or external bucket access.
6. **Public API Endpoint**
   - Configure an HTTP API Gateway trigger for the Lambda function.
   - This publishes the inference function as a REST endpoint capable of handling JSON requests. Users can now submit a structured JSON payload representing a customer record and receive a churn probability score.
7. **Interactive Streamlit Dashboard**
   - Develop a real-time prediction interface using Streamlit.
   - Supported features:
     - Single-customer inference with real-time visualization using a Plotly gauge
     - Batch analytics: Upload CSV or JSON, perform row-level inference, compute churn KPIs, and download results

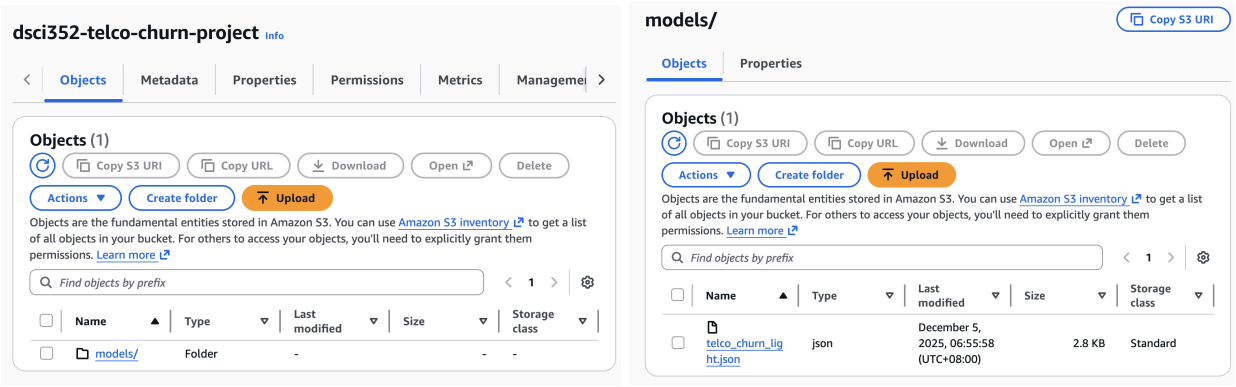
Deployed Components & Screenshots

Model Serialization (AWS S3)

To deploy a low latency serverless, we did not upload heavy machine learning libraries (involving Scikit-Learn, Pandas or similar). The local training script does not extract the whole SGDClassifier but only its useful mathematical components and writes them into a lightweight JSON artifact instead.

- Raw Model Parameters: The learned coefficients and intercept of the logistic regression classifier.
- Preprocessing Metadata: The specific means, minimums, and maximums required for the MinMaxScaler, and the ordered list of categories for One-Hot Encoding.

These values are written to a single lightweight JSON artifact, which is stored in a private S3 bucket and serves as the model registry. This design ensures the deployment package stays within AWS Lambda's size limits and has no complex dependencies.

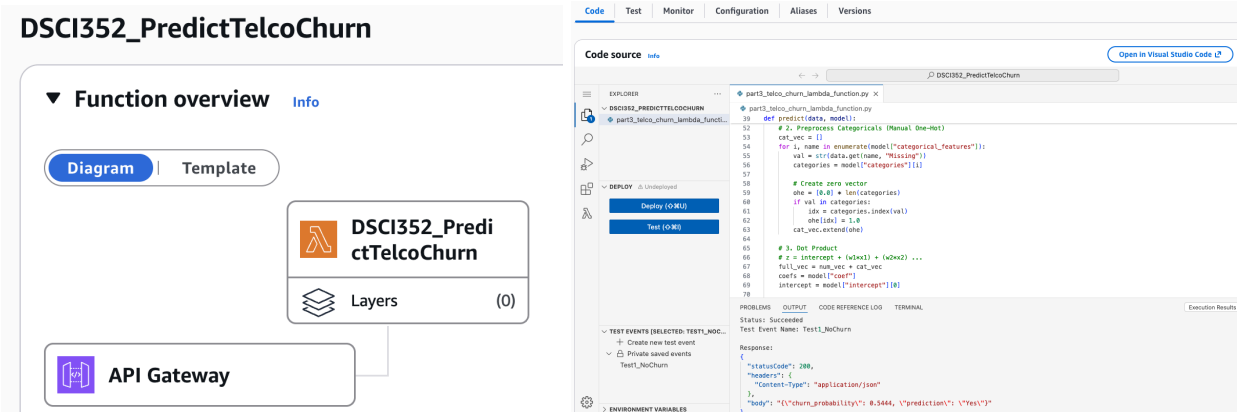


Serverless Inference Service (AWS Lambda)

The core prediction logic is managed by AWS Lambda. In order to optimize performance:

- Upon the first invocation, the function fetches the JSON artifact from S3 and caches it in memory. Subsequent requests use this cached model, significantly reducing latency.
- Takes control of the manual functions of linear algebra and sigmoid activation. The performance of the function on cold-start is better since it does not require large libraries.

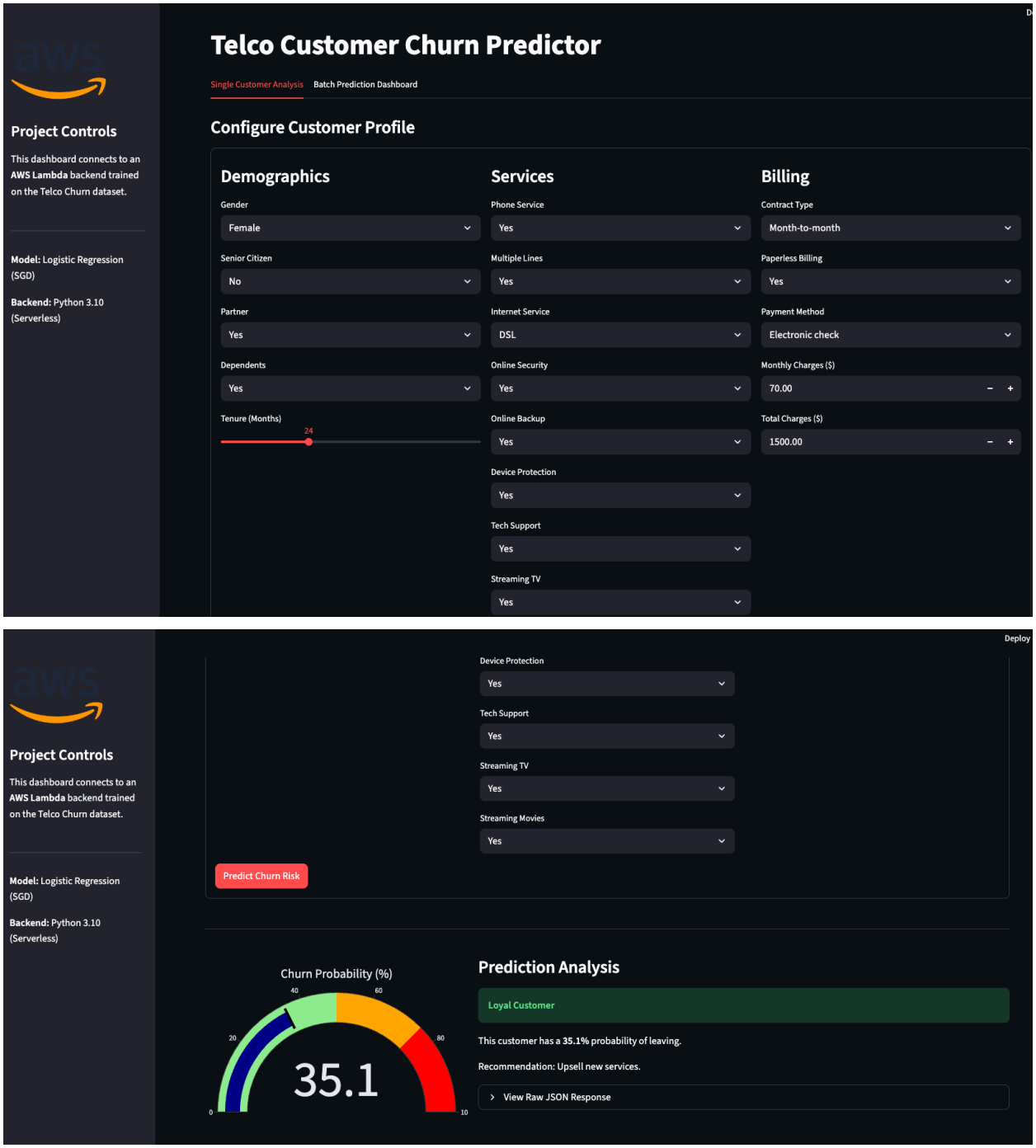
The AWS API Gateway (HTTP API) exposes the Lambda function as a public REST API endpoint, linking the backend logic with the front-end application.



User Interface (Streamlit Dashboard)

The executives will make any decisions based on the Streamlit Dashboard. The API Gateway accepts standard HTTP requests in two different ways:

- **Single Customer Analysis:** allow the user to change issue by issue (like increasing tenure) and see the immediate impact on churn.



- **Batch Prediction:** upload a CSV file with new customers. The Dashboard goes through the same procedure for all other rows. API is checked for each row, and a report is formed which will be downloaded in a single file.

Deploy

Project Controls

This dashboard connects to an **AWS Lambda** backend trained on the Telco Churn dataset.

Model: Logistic Regression (SGD)

Backend: Python 3.10 (Serverless)

Telco Customer Churn Predictor

Single Customer Analysis | Batch Prediction Dashboard

Batch Analytics Dashboard

Upload CSV or JSON file

Drag and drop file here
Limit 200MB per file • CSV, JSON
Browse files

Download Sample CSV Template

Deploy

Project Controls

This dashboard connects to an **AWS Lambda** backend trained on the Telco Churn dataset.

Model: Logistic Regression (SGD)

Backend: Python 3.10 (Serverless)

Drag and drop file here
Limit 200MB per file • CSV, JSON
Browse files

telco_test_sample.csv 6.2KB

Download Sample CSV Template

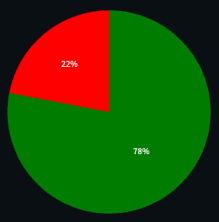
Loaded **50** customers.

Run Batch Prediction

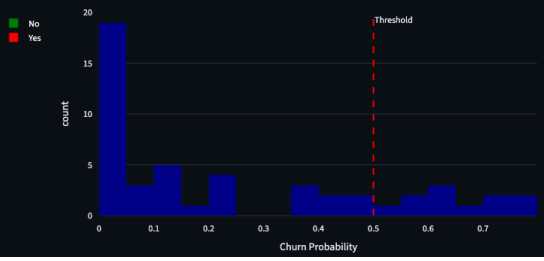
Analytics Report

Total Customers	Predicted Churners	Average Churn Risk
50	11	24.7%

Churn Distribution



Risk Probability Distribution



Model: Logistic Regression (SGD)

Backend: Python 3.10 (Serverless)

Detailed Results

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	Streami
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	No	No
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	No	No
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No
5	Female	0	No	No	8	Yes	Yes	Fiber optic	No	No	Yes	No	Yes	Yes
6	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	Yes	No	No	Yes	No
7	Female	0	No	No	10	No	No phone service	DSL	Yes	No	No	No	No	No
8	Female	0	Yes	No	28	Yes	Yes	Fiber optic	No	No	Yes	Yes	Yes	Yes
9	Male	0	No	Yes	62	Yes	No	DSL	Yes	Yes	No	No	No	No

Download Predictions (CSV)

AI Usage Statement

- Tool(s)used: Gemini Pro, Cursor
- How l used Al:
    - Generated code for creating the 30 days time window and using it to predict future price for BTC
    - Assisted in setting up AWS S3 configurations
    - Generated and debugged the Streamlit code and the Lambda function for Part 3
    - Polished language and tone for PDF report
- How l verified outputs:
    - I compared the loss to the visualization graphs. I have also validated AI suggestions against lecture material
- Approximate percentage of code written by AI: 30%