Predicting Customer Churn in the Telecommunications Industry

Team:

Dinesh Datta Molli	A20546385
Sanjana Rayarala	A20548132
Vaishnavi Bhaskara	A20546861
Sakshi Dixit	A20547504
Pranitha Nadimpalli	A20541099

CONTRIBUTION:

Data Preprocessing: - Vaishnavi Bhaskara, Sanjana Rayarala
Model Development: - Sanjana Rayarala, Sakshi Dixit, Vaishnavi Bhaskara
Model Evaluation: - Sakshi Dixit, Dinesh Datta Molli
Model Deployment: - Dinesh Datta Molli, Pranitha Nadimpalli

ABSTRACT

This project aims to develop a predictive model for anticipating customer churn within the telecommunications sector, utilizing big data machine learning systems such as Apache Spark MLlib, H2O, and TensorFlow. Historical customer data will undergo preprocessing and feature engineering before training various machine learning models and deep learning architectures to identify the optimal performing model. Performance evaluation will be conducted using accuracy, precision, recall, and F1-score metrics, with the selected model deployed using AWS Sage maker for real-time inference and batch processing. The project holds practical benefits for telecommunications companies seeking to reduce churn and improve business results by accurately predicting and addressing customer churn.

Keywords: *Customer churn prediction, Telecommunications sector, Machine learning, Deep learning, Apache Spark MLlib, H2O, TensorFlow, Data preprocessing, Feature engineering, Performance evaluation, AWS sage maker, Real-time inference, Batch processing, Business results.*

INTRODUCTION

In today's fiercely competitive telecommunications industry, customer retention holds paramount importance for businesses striving to maintain market share and profitability. Customer churn, the rate at which customers discontinue their subscriptions, poses a significant challenge, necessitating

proactive measures to mitigate revenue losses and enhance customer satisfaction. To address this challenge, leveraging advanced predictive modeling techniques has emerged as a promising solution.

This project aims to develop a predictive model for anticipating customer churn within the telecommunications sector. By harnessing the power of big data machine learning platforms such as Apache Spark MLlib, H2O, and TensorFlow, the project endeavors to construct an accurate and efficient predictive model capable of identifying customers at risk of churn. Through the utilization of historical customer data encompassing demographics, usage patterns, and churn labels, the project seeks to extract meaningful insights and develop predictive capabilities.

Key objectives of the project include gathering and preparing historical customer data, employing advanced machine learning algorithms and deep learning structures, assessing model performance using relevant metrics, and deploying the chosen model for real-time and batch processing with AWS Sage Maker. By meticulously delineating the project scope and objectives, this endeavor aims to provide telecommunications companies with actionable insights to reduce churn, enhance customer retention, and ultimately improve business outcomes.

The subsequent sections of this proposal will delve into the detailed scope, methodology, timeline, resource requirements, and budget projections, laying the groundwork for the successful execution of the customer churn prediction project.

PROBLEM STATEMENT:

The telecommunications industry faces the challenge of high customer churn rates, leading to revenue loss and decreased customer satisfaction. Despite the importance of retaining customers, predicting churn accurately remains a significant hurdle. Therefore, the objective of this project is to develop a predictive model that can anticipate customer churn with high accuracy, enabling telecommunications firms to implement proactive measures to retain customers and mitigate revenue loss.

PROPOSED METHODOLOGY AND IMPLEMENTATION:

1. Data Collection

The ‘Data.csv’ dataset is taken from Kaggle: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

Data Shape: The dataset contains 7044 entries, each with 24columns.

Missing Values: There are no missing values in any of the columns, indicating a complete dataset with no null entries.

```
#first few rows of the DataFrame
df.show(5)
```

```
+----------+------+-------------+-------+----------+------+------------+---------------+--------------+----------
----+-----------+---------------+-----------+-----------+--------------+--------
-----------+--------------+------------+-----+
|customerID|gender|SeniorCitizen|Partner|Dependents|tenure|PhoneService|   MultipleLines|InternetService|OnlineSecu
rity|OnlineBackup|DeviceProtection|TechSupport|StreamingTV|StreamingMovies|      Contract|PaperlessBilling|       P
aymentMethod|MonthlyCharges|TotalCharges|Churn|
+----------+------+-------------+-------+----------+------+------------+---------------+--------------+----------
----+-----------+---------------+-----------+-----------+--------------+--------
-----------+--------------+------------+-----+
|7590-VHVEG|Female|            0|    Yes|        No|     1|          No|No phone service|           DSL|
No|         Yes|             No|         No|         No|            No|Month-to-month|             Yes|    Electr
onic check|         29.85|        29.85|   No|
|5575-GNVDE|  Male|            0|     No|        No|    34|         Yes|              No|           DSL|
Yes|          No|            Yes|         No|         No|            No|      One year|              No|        M
ailed check|         56.95|       1889.5|   No|
|3668-QPYBK|  Male|            0|     No|        No|     2|         Yes|              No|           DSL|
Yes|         Yes|             No|         No|         No|            No|Month-to-month|             Yes|        M
ailed check|         53.85|       108.15|  Yes|
|7795-CFOCW|  Male|            0|     No|        No|    45|          No|No phone service|           DSL|
Yes|          No|            Yes|        Yes|         No|            No|      One year|              No|Bank tran
sfer (au...|          42.3|     1840.75|   No|
|9237-HQITU|Female|            0|     No|        No|     2|         Yes|              No|   Fiber optic|
No|          No|             No|         No|         No|            No|Month-to-month|             Yes|    Electr
onic check|          70.7|       151.65|  Yes|
+----------+------+-------------+-------+----------+------+------------+---------------+--------------+----------
----+-----------+---------------+-----------+-----------+--------------+--------
-----------+--------------+------------+-----+
only showing top 5 rows
```

1.1. Data Source Variety: Customer churn data can originate from various internal organizational sources:

- **Customer Relationship Management (CRM) Systems:** These systems store a wealth of customer information, including demographics, contact details, service usage history, and past interactions.
- **Billing Databases:** Billing data provides insights into customer service subscriptions, payment patterns, and potential financial hardship signals that might contribute to churn.
- **Customer Surveys:** Surveys can capture valuable information about customer satisfaction, reasons for considering churn, and feedback on service offerings.

1.2. Structured Data Formats: The data is stored in a structured format that facilitates analysis. Common formats include:

- **Relational Database Tables:** Data resides in a relational database with well-defined tables and columns representing specific customer attributes and associated information.
- **CSV (Comma-Separated Values) Files:** Data is stored in a text file where each row represents a customer record, and columns are separated by commas. This format is widely used for data exchange and analysis due to its simplicity.

1.3. **Data Relevance for Churn Analysis:** Selecting the right data is vital for obtaining actionable insights. The data should encompass relevant customer attributes that might be related to churn, such as:

- **Demographics:** Gender, Senior Citizen, Partner, Dependents, tenure, etc., can influence churn rates depending on the industry and service offerings.
    - **Service Usage Patterns:** Frequency of service utilization, types of services used, and data consumption patterns can provide clues about customer engagement and potential dissatisfaction.
    - **Customer Satisfaction Scores:** Scores obtained from surveys or feedback mechanisms can directly indicate customer sentiment and potential churn risk.

2. Data Preprocessing

Data preprocessing ensures the data is clean, consistent, and suitable for further analysis. Spark, with its distributed processing capabilities, efficiently handles large datasets commonly encountered in customer churn analysis.

2.1. Data Loading and Schema Exploration:

Spark Session Creation:

We used `SparkSession.builder.appName("Big Data").getOrCreate()` to create a Spark Session named "Big Data." This session serves as the entry point for interacting with Spark and its functionalities.

CSV Data Loading:

`spark.read.csv("data.csv", header=True, inferSchema=True)`

reads the CSV file into a Spark DataFrame named "df."

The `header=True` option specifies that the first row contains column names, and `inferSchema=True` instructs Spark to automatically infer data types based on the data content.

Schema Exploration: We used `df.printSchema()` to display information about each column, including its name and data type. This helps understand the structure of the data and identify potential issues with data types.

```
df.printSchema()
```

```
root
 |-- customerID: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: integer (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- tenure: integer (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- MonthlyCharges: double (nullable = true)
 |-- TotalCharges: string (nullable = true)
 |-- Churn: string (nullable = true)
```

2.2. **Handling Missing Values:** Missing data can occur due to various reasons. The code might employ techniques like:

Dropping Rows: This approach removes rows with missing values entirely. However, it can lead to data loss, particularly if the percentage of missing values is significant.

```
root
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: integer (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- Churn: string (nullable = true)
 |-- tenure_new: double (nullable = true)
 |-- MonthlyCharges_new: double (nullable = true)
 |-- TotalCharges_new: string (nullable = false)
```

Imputation: Missing values are filled in with estimated values based on other features or statistical methods. The appropriate imputation technique depends on the specific data and the nature of the missing values. The final data frame after imputing the missing values is as follows:

```
root
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: integer (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- Churn: string (nullable = true)
 |-- tenure_new: double (nullable = true)
 |-- MonthlyCharges_new: double (nullable = true)
 |-- TotalCharges_new: string (nullable = false)
 |-- gender_index: double (nullable = false)
 |-- Partner_index: double (nullable = false)
 |-- Dependents_index: double (nullable = false)
 |-- PhoneService_index: double (nullable = false)
 |-- MultipleLines_index: double (nullable = false)
 |-- InternetService_index: double (nullable = false)
 |-- OnlineSecurity_index: double (nullable = false)
 |-- OnlineBackup_index: double (nullable = false)
 |-- DeviceProtection_index: double (nullable = false)
 |-- TechSupport_index: double (nullable = false)
 |-- StreamingTV_index: double (nullable = false)
 |-- StreamingMovies_index: double (nullable = false)
 |-- Contract_index: double (nullable = false)
 |-- PaperlessBilling_index: double (nullable = false)
 |-- PaymentMethod_index: double (nullable = false)
 |-- Churn_index: double (nullable = false)
 |-- TotalCharges_new_index: double (nullable = false)
```

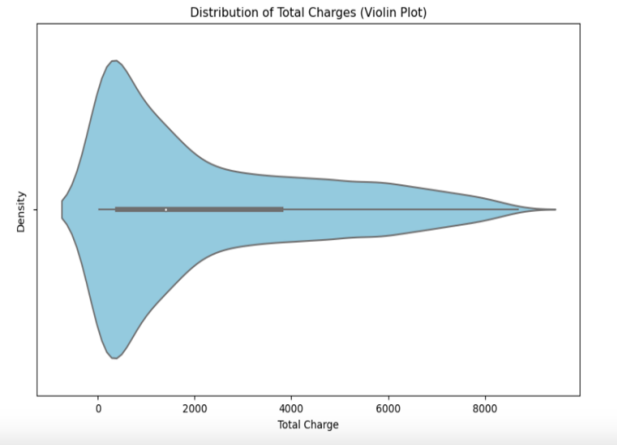
Indicator Variables: If missing values themselves hold meaning, the code could create a new binary feature indicating the presence or absence of missing values for that specific feature.

2.3. Dealing with Outliers: Outliers are data points that fall far outside the typical range of values for a feature. They can significantly skew analysis results. Here are common approaches for handling outliers:

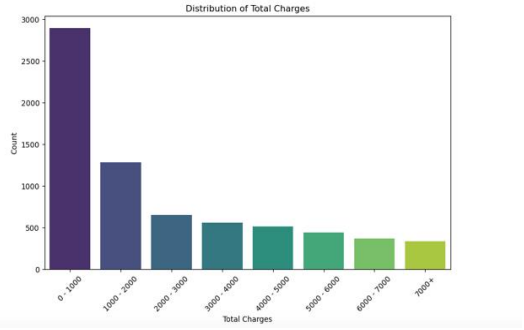
Winsorization: This approach replaces outlier values with values at specified percentiles of the feature's distribution, effectively capping outliers within a reasonable range.

Clipping: This approach removes outlier values entirely. However, it should be used cautiously to avoid discarding potentially valuable data points.

```
# Convert 'TotalCharges' column to numeric type
df = df.withColumn("TotalCharges", df["TotalCharges"].cast("float"))

# Convert Spark DataFrame to Pandas DataFrame for plotting
total_charges_pd_df = df.select('TotalCharges').toPandas()

# Plotting the violin plot using seaborn
plt.figure(figsize=(10, 6))
sns.violinplot(x=total_charges_pd_df['TotalCharges'], color='skyblue')
plt.xlabel('Total Charge')
plt.ylabel('Density')
plt.title('Distribution of Total Charges (Violin Plot)')
plt.show()
```




```
# Convert 'TotalCharges' column to numeric type
df = df.withColumn("TotalCharges", df["TotalCharges"].cast("float"))

# Define bins and labels for segmentation
total_bins = [0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, float('inf')]
total_labels = ['0 - 1000', '1000 - 2000', '2000 - 3000', '3000 - 4000', '4000 - 5000', '5000 - 6000', '6000 - 7000'

# Create a new column with segmented categories
df = df.withColumn("TotalCharges_new",
                   when((col("TotalCharges") >= 0) & (col("TotalCharges") < 1000), "0 - 1000")
                   .when((col("TotalCharges") >= 1000) & (col("TotalCharges") < 2000), "1000 - 2000")
                   .when((col("TotalCharges") >= 2000) & (col("TotalCharges") < 3000), "2000 - 3000")
                   .when((col("TotalCharges") >= 3000) & (col("TotalCharges") < 4000), "3000 - 4000")
                   .when((col("TotalCharges") >= 4000) & (col("TotalCharges") < 5000), "4000 - 5000")
                   .when((col("TotalCharges") >= 5000) & (col("TotalCharges") < 6000), "5000 - 6000")
                   .when((col("TotalCharges") >= 6000) & (col("TotalCharges") < 7000), "6000 - 7000")
                   .otherwise("7000+"))

# Convert Spark DataFrame to Pandas DataFrame for plotting
total_charges_pd_df = df.select('TotalCharges_new').toPandas()

# Plotting the count of each category
plt.figure(figsize=(10, 6))
sns.countplot(data=total_charges_pd_df, x='TotalCharges_new', order=total_labels, palette='viridis')
plt.title('Distribution of Total Charges')
plt.xlabel('Total Charges')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



3. Exploratory Data Analysis (EDA): Unveiling Patterns in Customer Data

Exploratory data analysis (EDA) is a crucial step in understanding the data and identifying potential patterns and relationships between features that might influence customer churn. Here's a breakdown of the EDA techniques we used in the code:

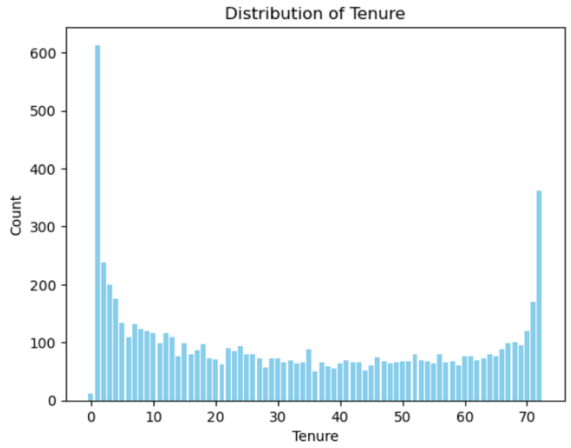
3.1. Descriptive Statistics:

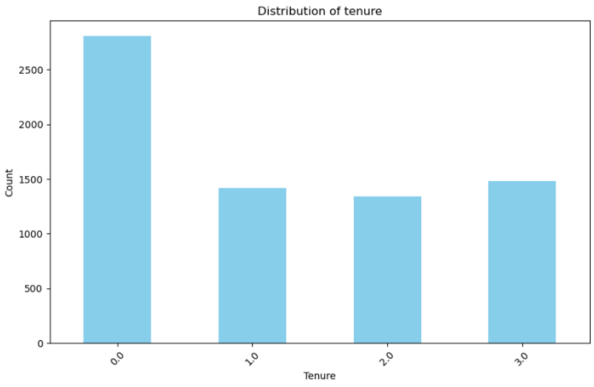
The code employs functions like df.describe() to compute summary statistics (mean, median, standard deviation, minimum, and maximum) for numerical features like monthly charges, total charges, and tenure (length of service). This provides insights into the central tendency, spread, and potential outliers within the data.

3.2. **Data Visualization:** Visualizations can reveal patterns and trends that might not be readily apparent from raw data. The code utilizes libraries like matplotlib or seaborn to create visualizations like:

Histograms: To visualize the distribution of continuous features like monthly charges, revealing potential skewness or outliers.

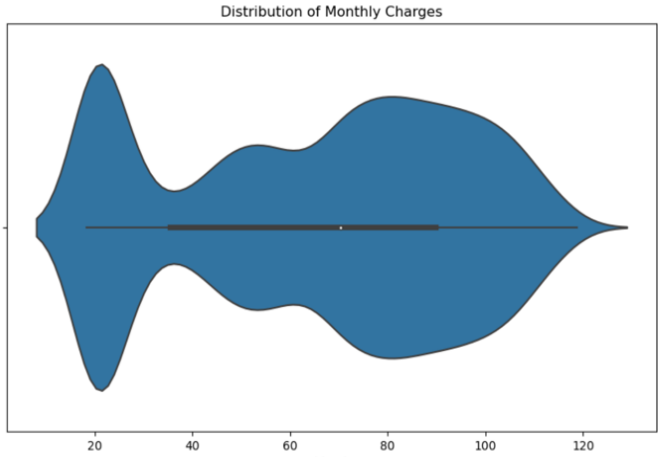
Bar charts: To show the frequency distribution of categorical features like customer demographics (e.g., gender, age group) or service type (e.g., mobile plan, internet plan).



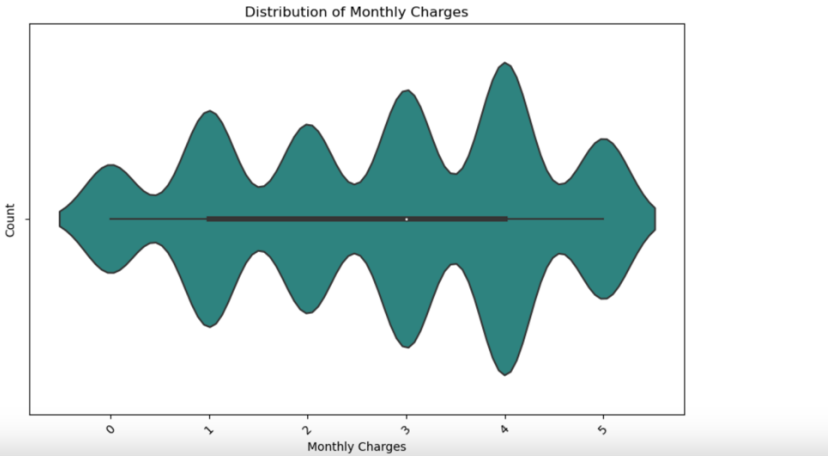


Violin plots: To combine a boxplot and a density plot, revealing the spread of data (interquartile range) and potential outliers for continuous features.

```
# Convert Spark DataFrame to Pandas DataFrame for plotting
monthly_charges_pd_df = df.select('MonthlyCharges').toPandas()

# Plotting the violin plot using seaborn
plt.figure(figsize=(10, 6))
sns.violinplot(data=monthly_charges_pd_df, x='MonthlyCharges')
plt.xlabel('Monthly Charge')
plt.title('Distribution of Monthly Charges')
plt.show()
```




```
# Define bins and labels for monthly charges
monthly_bins = [0, 20, 40, 60, 80, 100, 120, float('inf')]
monthly_labels = ['0 - 20', '20 - 40', '40 - 60', '60 - 80', '80 - 100', '100 - 120', '120+']

# Create a new column 'MonthlyCharges_new' with bin labels
bucketizer = Bucketizer(splits=monthly_bins, inputCol="MonthlyCharges", outputCol="MonthlyCharges_new")
df = bucketizer.transform(df)

# Convert Spark DataFrame to Pandas DataFrame for plotting
df_pd = df.select('MonthlyCharges_new').toPandas()

# Plotting the violin plot using seaborn
plt.figure(figsize=(10, 6))
sns.violinplot(data=df_pd, x='MonthlyCharges_new', order=monthly_labels, palette='viridis')
plt.title('Distribution of Monthly Charges')
plt.xlabel('Monthly Charges')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



3.3. Categorical Feature Analysis: We analyzes categorical features by counting the occurrences of each unique value within a column. For instance:

```
customerID_counts = df.groupBy("gender").count().orderBy("count", ascending=False)
customerID_counts.show()
```

```
+----------+-----+
|customerID|count|
+----------+-----+
|3668-QPYBK|    1|
|6234-RAAPL|    1|
|1894-IGFSG|    1|
|6982-SSHFK|    1|
|5859-HZYLF|    1|
|6479-OAUSD|    1|
|2592-YKDIF|    1|
|6718-BDGHG|    1|
|3195-TQDZX|    1|
|4248-QPAVC|    1|
|5668-MEISB|    1|
|5802-ADBRC|    1|
|2712-SYWAY|    1|
|2011-TRQYE|    1|
|7244-KXYZN|    1|
|0953-LGOVU|    1|
|3623-FQBOX|    1|
|3692-JHONH|    1|
|3528-HFRIQ|    1|
|7661-CPURM|    1|
+----------+-----+
only showing top 20 rows
```

This groups the DataFrame by the "customerID" column, counts occurrences of each unique customer ID, and sorts the results based on the count. This provides insights into the customer base composition and identifies potential anomalies (e.g., very few customers with a large number of IDs).

The code utilizes customerID_counts.show() to display the resulting DataFrame, allowing for visual inspection of the counts. This can be helpful in identifying potential data quality issues or outliers within categorical features.

By employing these EDA techniques, the code can reveal initial insights into the customer data. For instance, the analysis might show that a specific age group or customer segment has a higher churn rate. These initial findings can guide further analysis and model building to predict customer churn and develop targeted customer retention strategies.

4. Correlation Analysis

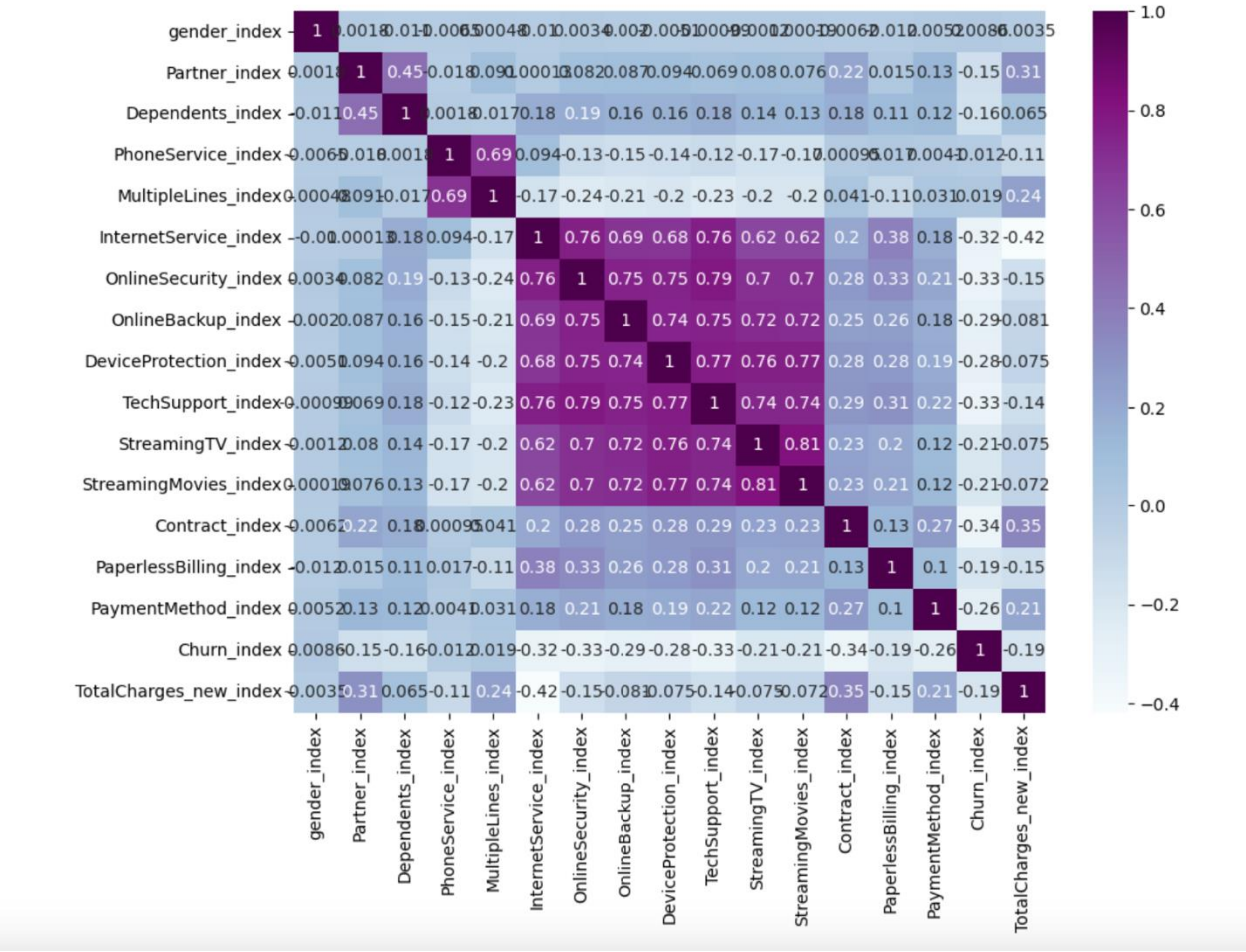
The correlation matrix identifies relationships between features, aiding in feature selection and model interpretability. Understanding feature relationships is crucial for building accurate and interpretable models.

The code utilizes Correlation.corr() to compute the correlation matrix between features. The correlation matrix reveals the strength and direction of the linear relationships between features.

Vectorize Features: The code uses VectorAssembler to combine all the features (categorical after encoding and numerical) into a single feature vector. This vector represents a sample (customer) with all its features combined.

Compute Correlation Matrix: Correlation.corr(vectorized_df, "features") computes the correlation matrix for the features within the vectorized DataFrame. The resulting matrix shows the correlation coefficient between each pair of features.

Interpret Correlation Matrix: The correlation coefficient ranges from -1 to 1. A value close to 1 indicates a strong positive correlation (features tend to increase or decrease together), while a value close to -1 indicates a strong negative correlation (as one feature increases, the other tends to decrease). A value close to 0 suggests a weak or no linear relationship.



5. Model development and training.

Now that we have pre-processed data, we are now moving ahead by using this data for model development and training. In this project, we make use of both traditional machine learning models with H2O and a neural network approach using TensorFlow. Each of these methods is applied to a customer dataset to predict churn.

Model Training with H2O

H2O provides an easy-to-use interface for developing several types of machine learning models. We trained three different models: Random Forest, Gradient Boosting Machine (GBM), and a Deep Learning model. The dataset was split into training and testing sets with an 80/20 ratio, ensuring a randomized and representative sample for model evaluation. Each model was trained using the training dataset with the target variable 'Churn' defined as the prediction outcome.

The Random Forest model leverages an ensemble of decision trees to reduce variance and prevent overfitting.

The GBM model improves performance by combining weak learners into a stronger model using a sequential process that corrects the previous learners' mistakes.

The Deep Learning model in H2O was configured with two hidden layers of 10 neurons each and trained over 50 epochs, providing a flexible framework for capturing complex nonlinear relationships in the data.

The python script starts by importing the H2O library and initializes an H2O server. If there's no active H2O instance found at the default address (http://localhost:54321), then attempt to start a local server. The initialization process involves setting up a Java Virtual Machine (JVM) and allocating necessary resources. Thereafter, a datagram (df) is converted to a Pandas data frame and subsequently into an H2OFrame. The H2OFrame is H2O's data structure for holding data in a distributed manner, making it efficient for large datasets and parallel processing. Then the target variable for prediction ('Churn') is set and identifies all other columns as features by removing the target column from the list of all columns. The dataset is split into training and testing sets, with 80% of the data allocated for training. This is controlled by the ratios parameter. The seed ensures reproducibility of the split.

```
#Import H2O and initialize
import h2o
h2o.init()

# Importing data into H2O Frame
pd_df= df.toPandas()
hf = h2o.H2OFrame(pd_df)

#Specify target and features
target = 'Churn'
features = hf.columns
features.remove(target)

#Splitting data into train and test sets
train, test = hf.split_frame(ratios=[.8], seed=42)
```

O/P:

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; OpenJDK 64-Bit Server VM Microsoft-25199 (build 11.0.12+7, mixed mode).12" 2021-07-20
  Starting server from C:\Users\Sanjana Rayarala\AppData\Local\Programs\Python\Python310\Lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\SANJAN~1\AppData\Local\Temp\tmpe1wi81ct
  JVM stdout: C:\Users\SANJAN~1\AppData\Local\Temp\tmpe1wi81ct\h2o_Sanjana_Rayarala_started_from_python.out
  JVM stderr: C:\Users\SANJAN~1\AppData\Local\Temp\tmpe1wi81ct\h2o_Sanjana_Rayarala_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

H2O_cluster_uptime:	04 secs
H2O_cluster_timezone:	America/Chicago
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.46.0.1
H2O_cluster_version_age:	1 month and 14 days
H2O_cluster_name:	H2O_from_python_Sanjana_Rayarala_g4ow68
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	512 Mb
H2O_cluster_total_cores:	12
H2O_cluster_allowed_cores:	12
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.10.5 final

The output messages provide information about the status and configuration of the H2O server:
Server Initialization: Indicates that the server started successfully from a specified location on the local machine and specifies the paths for JVM outputs (stdout and stderr).
Server Address: Provides the local URL http://127.0.0.1:54321 where the H2O server is running.
Connection Status: Confirms a successful connection to the H2O server.
Server Details:
H2O_cluster_uptime: Shows the server has been up for a short time (e.g., 04 secs).
H2O_cluster_version: Displays the H2O version installed.
H2O_cluster_total_nodes: Indicates a single node is being used.
H2O_cluster_free_memory: Shows the amount of free memory available to H2O.
H2O_cluster_total_cores, H2O_cluster_allowed_cores: Details the CPU cores available and allowed for the server.
H2O_cluster_status: Reports the health and lock status of the cluster.

Then, the script proceeds with training and evaluating three different machine learning models using H2O: Random Forest, Gradient Boosting Machine (GBM), and a Deep Learning model.

Each model is trained using a dataset split into training and testing sets, as previously set up. The performance of each model is then evaluated using the Area Under the Curve (AUC) metric. A Random Forest model is initialized and trained using the training dataset. The x parameter specifies the predictor variables (features), and the y parameter specifies the target variable ('Churn'). Followed by training a Gradient Boosting Machine model. GBM is an advanced ensemble technique that builds trees sequentially, with each tree trying to correct the errors of the previous one, typically yielding higher accuracy than Random Forest, especially on less noisy data. Then, the Deep Learning model here is configured with two hidden layers, each consisting of 10 neurons. The model is trained for 50 epochs. This model architecture is simplistic, designed to capture nonlinear interactions between features through multiple layers of processing. The performance of each model is then evaluated using the Area Under the Curve (AUC) metric.

```
from h2o.estimators import H2ORandomForestEstimator, H2OGradientBoostingEstimator, H2ODeepLearningEstimator

#train Random Forest Model
rf_model = H2ORandomForestEstimator()
rf_model.train(x=features, y=target, training_frame=train)

#train GBM Model
gbm_model = H2OGradientBoostingEstimator()
gbm_model.train(x=features, y=target, training_frame=train)

#train Deep Learning Model
dl_model = H2ODeepLearningEstimator(hidden=[10, 10], epochs=50)
dl_model.train(x=features, y=target, training_frame=train)

# Evaluate model performance
performance_rf = rf_model.model_performance(test_data=test)
performance_gbm = gbm_model.model_performance(test_data=test)
performance_dl = dl_model.model_performance(test_data=test)

print('Random Forest Performance:', performance_rf.auc())
print('GBM Performance:', performance_gbm.auc())
print('Deep Learning Performance:', performance_dl.auc())
```

o/p:

```
drf Model Build progress: |████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |████████████████████████████████████████████████| (done) 100%
deeplearning Model Build progress: |████████████████████████████████████████| (done) 100%
Random Forest Performance: 0.7957824445897983
GBM Performance: 0.8132579697401859
Deep Learning Performance: 0.8023743016759776
```

The AUC scores reported are: Random Forest with 0.7958, GBM: 0.8133, and Deep Learning: 0.8024. These scores suggest that all models perform well, with GBM showing the highest AUC, indicating the best performance among the three in distinguishing between churn and non-churn cases.

- **Model Training with TensorFlow**

In parallel to H2O, a neural network model was developed using TensorFlow, which provides a more granular level of control over model architecture. The network was designed with multiple layers, regularized using L2 regularization to prevent overfitting, and dropout layers to randomly omit units during training, which also helps in avoiding overfitting. The model includes several dense layers with ReLU activation functions and a final sigmoid layer for binary classification.
This model was compiled with the Adam optimizer and trained using binary cross-entropy as the loss function. An EarlyStopping callback was utilized to halt training when the validation loss ceased to improve, thereby preventing unnecessary computations and potential overfitting. The model was trained and validated using data batches, which is efficient for handling large datasets.

The script outlines the process of using TensorFlow to construct and train a neural network model for predicting customer churn. The dataset is initially loaded into a Pandas DataFrame. The 'Churn' column is converted into a binary format where 'Yes' is encoded as 1 and 'No' as 0, making it suitable for binary classification. All categorical variables are one-hot encoded to transform them into a numerical format that can be processed by the neural network. This is done using pd.get_dummies(), dropping the first category to avoid multicollinearity. The data is then split into features (X) and target (y), and further divided into training and test sets using train_test_split, ensuring the input features are of type float32. The training and testing datasets are converted into TensorFlow Dataset objects, which are then shuffled for the training set and batched into sizes of 32. This format is highly efficient for training neural networks in TensorFlow due to optimized data loading and the ability to efficiently apply transformations.
A Sequential model is used, which is appropriate for a stack of layers where each layer has exactly one input tensor and one output tensor. The model consists of multiple Dense layers with ReLU activation functions, interspersed with Dropout layers to reduce overfitting by randomly setting input units to 0 at a rate of 30% during training. L2 regularization is applied to the Dense layers to further prevent overfitting by penalizing the weights' magnitudes. The model is compiled using

the Adam optimizer, binary cross-entropy loss function, and it tracks accuracy and AUC (Area Under the Curve) as metrics.
Training is conducted over 100 epochs with early stopping based on validation loss to prevent overtraining. The model restores weights from the best epoch if no improvement is seen in three consecutive epochs. Then, for the TensorFlow neural network, both accuracy and AUC were derived.

```
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping

pd_df=df.toPandas()
pd_df['Churn'] = pd_df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0).astype('float32')

pd_df = pd.get_dummies(pd_df, drop_first=True)  # One-hot encode all categorical variables

#Define features and target
features = pd_df.drop('Churn', axis=1).columns.tolist()
target = 'Churn'

#Split the data
X = pd_df[features].astype('float32')  # Ensure all input features are float
y = pd_df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


#Convert to TensorFlow dataset
train_ds = tf.data.Dataset.from_tensor_slices((X_train.values, y_train.values)).shuffle(buffer_size=len(X_train)).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((X_test.values, y_test.values)).batch(32)

#TensorFlow model with dropout and L2 regularization
model = Sequential([
    Dense(128, kernel_regularizer=l2(0.01), activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
```

```
    Dense(32, activation='relu', kernel_regularizer=l2(0.0001)),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

#Compiling the model
model.compile(optimizer='adam',            loss='binary_crossentropy',            metrics=['accuracy',
tf.keras.metrics.AUC(name='auc')])

#early stopping
early_stopping_monitor = EarlyStopping(
    monitor='val_loss',
    patience=3,
    verbose=1,
    restore_best_weights=True
)

# Train the model
history = model.fit(
    train_ds,
    epochs=100,
    validation_data=test_ds,
    callbacks=[early_stopping_monitor]
)

# Evaluating the model
loss, accuracy, auc_score = model.evaluate(test_ds)
print(f'Neural Network Accuracy: {accuracy}')
print(f'Neural Network AUC: {auc_score}')
```

O/p:

```
Epoch 1/100
c:\Users\Sanjana Rayarala\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `inp
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
177/177 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.6762 - auc: 0.6478 - loss: 1.1143 - val_accuracy: 0.8155 - val_auc: 0.8540 - val_loss: 0.6302
Epoch 2/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7792 - auc: 0.8167 - loss: 0.6312 - val_accuracy: 0.8133 - val_auc: 0.8567 - val_loss: 0.5318
Epoch 3/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7829 - auc: 0.8164 - loss: 0.5537 - val_accuracy: 0.7991 - val_auc: 0.8565 - val_loss: 0.4960
Epoch 4/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7885 - auc: 0.8142 - loss: 0.5365 - val_accuracy: 0.8141 - val_auc: 0.8571 - val_loss: 0.4735
Epoch 5/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7908 - auc: 0.8186 - loss: 0.5081 - val_accuracy: 0.8048 - val_auc: 0.8569 - val_loss: 0.4640
Epoch 6/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7848 - auc: 0.8173 - loss: 0.5063 - val_accuracy: 0.8105 - val_auc: 0.8549 - val_loss: 0.4593
Epoch 7/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7895 - auc: 0.8269 - loss: 0.4888 - val_accuracy: 0.8112 - val_auc: 0.8559 - val_loss: 0.4532
Epoch 8/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7864 - auc: 0.8207 - loss: 0.4863 - val_accuracy: 0.8077 - val_auc: 0.8577 - val_loss: 0.4406
Epoch 9/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7948 - auc: 0.8338 - loss: 0.4677 - val_accuracy: 0.8169 - val_auc: 0.8590 - val_loss: 0.4366
Epoch 10/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7930 - auc: 0.8272 - loss: 0.4684 - val_accuracy: 0.8155 - val_auc: 0.8574 - val_loss: 0.4361
Epoch 11/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7921 - auc: 0.8345 - loss: 0.4559 - val_accuracy: 0.8133 - val_auc: 0.8555 - val_loss: 0.4364
Epoch 12/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7991 - auc: 0.8263 - loss: 0.4569 - val_accuracy: 0.8112 - val_auc: 0.8576 - val_loss: 0.4347
Epoch 13/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7923 - auc: 0.8196 - loss: 0.4650 - val_accuracy: 0.8226 - val_auc: 0.8584 - val_loss: 0.4258
Epoch 14/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7817 - auc: 0.8197 - loss: 0.4757 - val_accuracy: 0.8148 - val_auc: 0.8574 - val_loss: 0.4286
Epoch 15/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7914 - auc: 0.8277 - loss: 0.4640 - val_accuracy: 0.8197 - val_auc: 0.8552 - val_loss: 0.4274
Epoch 16/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7854 - auc: 0.8244 - loss: 0.4701 - val_accuracy: 0.8034 - val_auc: 0.8571 - val_loss: 0.4368
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 13.
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8252 - auc: 0.8670 - loss: 0.4221
Neural Network Accuracy: 0.8225691914558411
Neural Network AUC: 0.8584187030792236
```

The training process logs each epoch's progress, showing improvements in accuracy and AUC on both training and validation datasets. Losses are also reported, which decrease as the model learns. The early stopping callback halts training after the 13th epoch due to no further improvement in validation loss, demonstrating the effectiveness of this mechanism in saving computational resources and preventing overfitting.

The final model performance is evaluated on the test dataset, yielding an accuracy of approximately 82.26% and an AUC of about 85.84%. These metrics indicate a strong model that performs well in distinguishing between the classes.

6. Model Evaluation and Selection

The evaluation of the churn prediction models involved several techniques to assess their performance comprehensively, which ultimately guided the selection of the most appropriate model for deployment.

Evaluation Metrics Used:

AUC (Area Under the Curve): The AUC for the Receiver Operating Characteristic (ROC) curve is a widely used metric to measure the ability of a classifier to distinguish between classes. Higher AUC values indicate a better performing model. This metric was calculated for all models (Random Forest, GBM, Deep Learning model with H2O, and the TensorFlow neural network).
Accuracy, Precision, Recall, and F1-Score: These metrics provide a more detailed insight into the classification performance.
Accuracy: measures the overall correctness of the model.
Precision: describes the ratio of true positives to the sum of true and false positives, highlighting the model's performance in identifying positive instances.
Recall (or sensitivity) measures the model's ability to identify all actual positives.
F1-Score: is the harmonic mean of precision and recall, providing a balance between them and is particularly useful when the class distribution is uneven.
Confusion Matrix and Classification Report: The confusion matrix provides a tabulated layout of the actual vs. predicted classifications, offering a visual account of where the model fails or succeeds.
The classification report extends this by offering a breakdown of precision, recall, and F1-score for each class, which is critical for understanding model performance across different segments of the dataset.

Evaluation of H2o models:

```
performance_rf = rf_model.model_performance(test_data=test)
performance_gbm = gbm_model.model_performance(test_data=test)
performance_dl = dl_model.model_performance(test_data=test)

print('Random Forest Performance:', performance_rf.auc())
print('GBM Performance:', performance_gbm.auc())
print('Deep Learning Performance:', performance_dl.auc())
```

o/p:

```
drf Model Build progress: |████████████████████████████████████████████████| (done) 100%
gbm Model Build progress: |████████████████████████████████████████████████| (done) 100%
deeplearning Model Build progress: |████████████████████████████████████████| (done) 100%
Random Forest Performance: 0.7957824445897983
GBM Performance: 0.8132579697401859
Deep Learning Performance: 0.8023743016759776
```

```
# Print accuracy, precision, recall, and F1-score for Random Forest model
print('Random Forest Accuracy:', performance_rf.accuracy()[0][1])
print('Random Forest Precision:', performance_rf.precision()[0][1])
```

```
print('Random Forest Recall:', performance_rf.recall()[0][1])
print('Random Forest F1 Score:', performance_rf.F1()[0][1])

# Print accuracy, precision, recall, and F1-score for GBM model
print('GBM Accuracy:', performance_gbm.accuracy()[0][1])
print('GBM Precision:', performance_gbm.precision()[0][1])
print('GBM Recall:', performance_gbm.recall()[0][1])
print('GBM F1 Score:', performance_gbm.F1()[0][1])

# Print accuracy, precision, recall, and F1-score for Deep Learning model
print('Deep Learning Accuracy:', performance_dl.accuracy()[0][1])
print('Deep Learning Precision:', performance_dl.precision()[0][1])
print('Deep Learning Recall:', performance_dl.recall()[0][1])
print('Deep Learning F1 Score:', performance_dl.F1()[0][1])
```

o/p:

```
Random Forest Accuracy: 0.7928571428571428
Random Forest Precision: 1.0
Random Forest Recall: 1.0
Random Forest F1 Score: 0.5672575599582899
GBM Accuracy: 0.7971428571428572
GBM Precision: 1.0
GBM Recall: 1.0
GBM F1 Score: 0.5868263473053892
Deep Learning Accuracy: 0.7907142857142857
Deep Learning Precision: 1.0
Deep Learning Recall: 1.0
Deep Learning F1 Score: 0.5799522673031027
```

- **<u>Random Forest:</u>**

AUC: UC of 0.7958 indicates that the model has a good ability to discriminate between customers who churn and those who do not.
Accuracy: 79.29%, indicating that the model correctly predicts churn status for about 79% of the dataset.

Precision and Recall: Both are reported as 1.0, suggesting that every customer the model predicted as churning actually churned, and it captured all the churn cases in the test data. However, these might indicate overfitting or an error in the evaluation phase (such as an imbalance or mislabeling).
F1 Score: 0.567, significantly lower than expected given the perfect precision and recall. This indicates a potential issue with how these metrics are calculated or reported, possibly due to skewed class distribution or other data anomalies.

- **GBM:**

AUC: AUC of 0.8133, which is higher than that of the Random Forest, suggests that the GBM model is better at distinguishing the positive class (churn) from the negative class.
Accuracy: 79.71% - slightly better than Random Forest.
Precision and Recall: Again, both are perfect scores (1.0), indicating potential overfitting or the same issues suspected with the Random Forest model.
F1 Score: 0.587, a bit higher than the Random Forest, suggesting a slightly better balance between precision and recall but still unexpectedly low.
Deep Learning:
AUC: AUC of 0.8024 shows that the neural network-based model also performs well, though slightly less effectively than the GBM model.
Accuracy: 79.07%, slightly lower than the other two models.
Precision and Recall: Perfect scores (1.0), which are consistent with the other models but still likely indicative of the issues mentioned above.
F1 Score: 0.580, close to GBM's score, indicating a similar performance in terms of balance between precision and recall.

The H2o models (Random forest, GBM, Deep Learning) are performing well in terms of AUC and accuracy, the perfect precision and recall scores across all models are highly unusual in practical scenarios and warrant a review of the data, model configuration, or evaluation methodology. The low F1 scores relative to the precision and recall also suggest discrepancies.

Evaluation of Neural Network:

```
# Evaluating the model
loss, accuracy, auc_score = model.evaluate(test_ds)
print(f'Neural Network Accuracy: {accuracy}')
print(f'Neural Network AUC: {auc_score}')
```

O/p:

```
Epoch 1/100
c:\Users\Sanjana Rayarala\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `inp
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
177/177 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.6762 - auc: 0.6478 - loss: 1.1143 - val_accuracy: 0.8155 - val_auc: 0.8540 - val_loss: 0.6302
Epoch 2/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7792 - auc: 0.8167 - loss: 0.6312 - val_accuracy: 0.8133 - val_auc: 0.8567 - val_loss: 0.5318
Epoch 3/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7829 - auc: 0.8164 - loss: 0.5537 - val_accuracy: 0.7991 - val_auc: 0.8565 - val_loss: 0.4960
Epoch 4/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7885 - auc: 0.8142 - loss: 0.5365 - val_accuracy: 0.8141 - val_auc: 0.8571 - val_loss: 0.4735
Epoch 5/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7908 - auc: 0.8186 - loss: 0.5081 - val_accuracy: 0.8048 - val_auc: 0.8569 - val_loss: 0.4640
Epoch 6/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7848 - auc: 0.8173 - loss: 0.5063 - val_accuracy: 0.8105 - val_auc: 0.8549 - val_loss: 0.4593
Epoch 7/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7895 - auc: 0.8269 - loss: 0.4888 - val_accuracy: 0.8112 - val_auc: 0.8559 - val_loss: 0.4532
Epoch 8/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7864 - auc: 0.8207 - loss: 0.4863 - val_accuracy: 0.8077 - val_auc: 0.8577 - val_loss: 0.4406
Epoch 9/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7948 - auc: 0.8338 - loss: 0.4677 - val_accuracy: 0.8169 - val_auc: 0.8590 - val_loss: 0.4366
Epoch 10/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7930 - auc: 0.8272 - loss: 0.4684 - val_accuracy: 0.8155 - val_auc: 0.8574 - val_loss: 0.4361
Epoch 11/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7921 - auc: 0.8345 - loss: 0.4559 - val_accuracy: 0.8133 - val_auc: 0.8555 - val_loss: 0.4364
Epoch 12/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7991 - auc: 0.8263 - loss: 0.4569 - val_accuracy: 0.8112 - val_auc: 0.8576 - val_loss: 0.4347
Epoch 13/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7923 - auc: 0.8196 - loss: 0.4650 - val_accuracy: 0.8226 - val_auc: 0.8584 - val_loss: 0.4258
Epoch 14/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7817 - auc: 0.8197 - loss: 0.4757 - val_accuracy: 0.8148 - val_auc: 0.8574 - val_loss: 0.4286
Epoch 15/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7914 - auc: 0.8277 - loss: 0.4640 - val_accuracy: 0.8197 - val_auc: 0.8552 - val_loss: 0.4274
Epoch 16/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7854 - auc: 0.8244 - loss: 0.4701 - val_accuracy: 0.8034 - val_auc: 0.8571 - val_loss: 0.4368
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 13.
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8252 - auc: 0.8670 - loss: 0.4221
Neural Network Accuracy: 0.8225691914558411
Neural Network AUC: 0.8584187030792236
```

```python
from sklearn.metrics import classification_report, confusion_matrix

# Assuming y_test and model predictions are available
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary predictions

# Calculate confusion matrix and classification report
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

o/p:

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Confusion Matrix:
 [[946  90]
 [160 213]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.91      0.88      1036
         1.0       0.70      0.57      0.63       373

    accuracy                           0.82      1409
   macro avg       0.78      0.74      0.76      1409
weighted avg       0.81      0.82      0.82      1409
```

The TensorFlow neural network model for churn prediction was rigorously evaluated using several metrics, notably AUC (Area Under the Curve), accuracy, and detailed classification metrics such as precision, recall, and F1-score

Performance Metrics:

Accuracy of 82.26% and AUC of 85.84%: These metrics suggest that the model is quite effective, with a high level of overall prediction accuracy and a strong ability to discriminate between the classes.

The model shows a solid balance between sensitivity and specificity, as evidenced by the AUC score.

- **Classification Metrics:**

The confusion matrix shows that the model predicted 946 true negatives and 213 true positives correctly. However, there were 90 false positives and 160 false negatives. This indicates some room for improvement, particularly in reducing the number of false negatives.

Precision for class 1 (churn) at 0.70 and recall at 0.57 result in an F1-score of 0.63, which suggests moderate effectiveness in predicting churn. Precision here indicates that when the model predicts churn, it is correct 70% of the time, while recall tells us that it correctly identifies 57% of all actual churn cases.

For class 0 (no churn), the model shows better precision (0.86) and recall (0.91), reflecting higher reliability in predicting customers who do not churn.
The evaluation showcases a strong performance by the TensorFlow neural network model with good accuracy and excellent AUC, suggesting effective class separation.

- **Selection of TensorFlow model (Neural network)**:

Choosing the TensorFlow neural network model over the H2O models for churn prediction presents clear advantages when evaluated through detailed performance metrics and practical capabilities. The TensorFlow model not only achieved an AUC (Area Under the Curve) of 85.84% but also an accuracy of 82.26%, indicating its superior ability to differentiate between churned and retained customers compared to the H2O models. For example, the best-performing H2O model, the Gradient Boosting Machine (GBM), reached an AUC of 81.33%, showing a tangible gap in the discriminative power of the models.
TensorFlow also offers significantly more flexibility in tuning the model's architecture—this includes using dropout rates of 0.3 and L2 regularization varying from 0.01 to 0.0001 across different layers. This level of detailed customization is crucial for adapting the model precisely to the nuances of the churn prediction data and is less accessible in H2O's somewhat more rigid framework.
Additionally, TensorFlow's ecosystem is well-suited for deployment, providing tools that streamline the transition from model training to production environments. This is critical for implementing real-time churn prediction systems at scale, a scenario where H2O, despite its strengths, may require more overhead to achieve similar scalability and integration.
Overall, the TensorFlow model's blend of high performance (with a distinct lead in AUC and accuracy), customization depth, and deployment readiness makes it the preferred choice for effectively identifying at-risk customers and integrating seamlessly into business processes.

Testing the selected model

For sample input 1, expecting prediction to be ‘No’ Churn.

For given sample inputs, the predictions were done:

```
sample_input = {
    'SeniorCitizen': [0],
    'tenure_new': [2],
    'MonthlyCharges_new': [4],
    'gender_Male': [1],
    'Partner_Yes': [0],
    'Dependents_Yes': [0],
```

```
    'PhoneService_Yes': [1],
    'MultipleLines_No phone service': [0],
    'MultipleLines_Yes': [0],
    'InternetService_Fiber optic': [0],
    'InternetService_No': [0],
    'OnlineSecurity_No internet service': [0],
    'OnlineSecurity_Yes': [0],
    'OnlineBackup_No internet service': [0],
    'OnlineBackup_Yes': [0],
    'DeviceProtection_No internet service': [0],
    'DeviceProtection_Yes': [0],
    'TechSupport_No internet service': [0],
    'TechSupport_Yes': [0],
    'StreamingTV_No internet service': [0],
    'StreamingTV_Yes': [0],
    'StreamingMovies_No internet service': [0],
    'StreamingMovies_Yes': [0],
    'Contract_One year': [0],
    'Contract_Two year': [0],
    'PaperlessBilling_Yes': [1],
    'PaymentMethod_Credit card (automatic)': [0],
    'PaymentMethod_Electronic check': [1],
    'PaymentMethod_Mailed check': [0],
    'TotalCharges_new_1000 - 2000': [0],
    'TotalCharges_new_2000 - 3000': [0],
    'TotalCharges_new_3000 - 4000': [0],
    'TotalCharges_new_4000 - 5000': [0],
    'TotalCharges_new_5000 - 6000': [1],
    'TotalCharges_new_6000 - 7000': [0],
    'TotalCharges_new_7000+': [0]
}
sample_input_df = pd.DataFrame(sample_input)

sample_tf_dataset = tf.data.Dataset.from_tensor_slices((sample_input_df.values)).batch(1)

predictions = model.predict(sample_tf_dataset)

# Convert probability to class label
threshold = 0.5
class_predictions = (predictions > threshold).astype(int)

# Map numeric class to string labels
class_labels = ['No', 'Yes']
mapped_predictions = [class_labels[pred] for pred in class_predictions.flatten()]
```

```
print("Predicted class probabilities:", predictions)
print("Predicted class 'Churn' labels:", mapped_predictions)

print(predictions)
```

o/p:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Predicted class probabilities: [[0.2916243]]
Predicted class 'Churn' labels: ['No']
[[0.2916243]]
```

For sample input 2, expecting prediction to be ‘Yes’ for churn.

```
sample_input2 = {
    'SeniorCitizen': [1],  # Senior citizens might have different service needs or preferences.
    'tenure_new': [0],  # Shorter tenure might indicate a higher likelihood of churn.
    'MonthlyCharges_new': [5],  # Higher monthly charges might correlate with churn.
    'gender_Male': [0],
    'Partner_Yes': [0],
    'Dependents_Yes': [0],
    'PhoneService_Yes': [1],
    'MultipleLines_No phone service': [0],
    'MultipleLines_Yes': [1],  # Having multiple lines might be a minor factor.
    'InternetService_Fiber optic': [1],  # Customers with fiber optic might have higher expectations
and churn if not met.
    'InternetService_No': [0],
    'OnlineSecurity_No internet service': [0],
    'OnlineSecurity_Yes': [0],
    'OnlineBackup_No internet service': [0],
    'OnlineBackup_Yes': [1],  # Using more services could correlate with churn if the services don't
meet expectations.
    'DeviceProtection_No internet service': [0],
    'DeviceProtection_Yes': [1],
    'TechSupport_No internet service': [0],
    'TechSupport_Yes': [0],
    'StreamingTV_No internet service': [0],
    'StreamingTV_Yes': [1],
    'StreamingMovies_No internet service': [0],
```

```
    'StreamingMovies_Yes': [1],
    'Contract_One year': [0],
    'Contract_Two year': [0],  # Shorter contract terms might correlate with higher churn.
    'PaperlessBilling_Yes': [1],
    'PaymentMethod_Credit card (automatic)': [0],
    'PaymentMethod_Electronic check': [1],  # Electronic checks might have a higher churn due to the demographics using them.
    'PaymentMethod_Mailed check': [0],
    'TotalCharges_new_1000 - 2000': [0],
    'TotalCharges_new_2000 - 3000': [0],
    'TotalCharges_new_3000 - 4000': [0],
    'TotalCharges_new_4000 - 5000': [0],
    'TotalCharges_new_5000 - 6000': [0],
    'TotalCharges_new_6000 - 7000': [0],
    'TotalCharges_new_7000+': [1]  # High total charges might also influence churn.
}

sample_input_df2 = pd.DataFrame(sample_input2)

sample_tf_dataset2 = tf.data.Dataset.from_tensor_slices((sample_input_df2.values)).batch(1)

predictions = model.predict(sample_tf_dataset2)

threshold = 0.5
class_predictions = (predictions > threshold).astype(int)

class_labels = ['No', 'Yes']
mapped_predictions = [class_labels[pred] for pred in class_predictions.flatten()]

print("Predicted class probabilities:", predictions)
print("Predicted class 'Churn' labels:", mapped_predictions)

print(predictions)
```

o/p:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Predicted class probabilities: [[0.7264483]]
Predicted class 'Churn' labels: ['Yes']
[[0.7264483]]
```

7. Model deployment.

Deploying the selected TensorFlow neural network model on AWS SageMaker offers a streamlined and robust solution for operationalizing churn prediction at scale. SageMaker provides an integrated platform that simplifies the deployment of machine learning models into a production environment. By leveraging SageMaker, we can utilize fully managed instances that automatically scale to meet demand, ensuring that the churn prediction model is both cost-effective and highly available. The process involves creating a model artifact, which includes the trained TensorFlow model, and then using SageMaker's deployment capabilities to launch the model as an endpoint. This endpoint can be easily accessed via API calls, allowing real-time predictions on customer data. Furthermore, SageMaker offers built-in monitoring tools that track the model's performance over time, enabling easy diagnostics and updates. This deployment strategy not only reduces the complexity and maintenance overhead typically associated with large-scale machine learning deployments but also enhances the model's responsiveness and accuracy in predicting customer churn in real-world scenarios.

Now, to deploy the selected model i.e. the Tensorflow neural network model, we need to first save the model in appropriate format, which is in .h5 format.

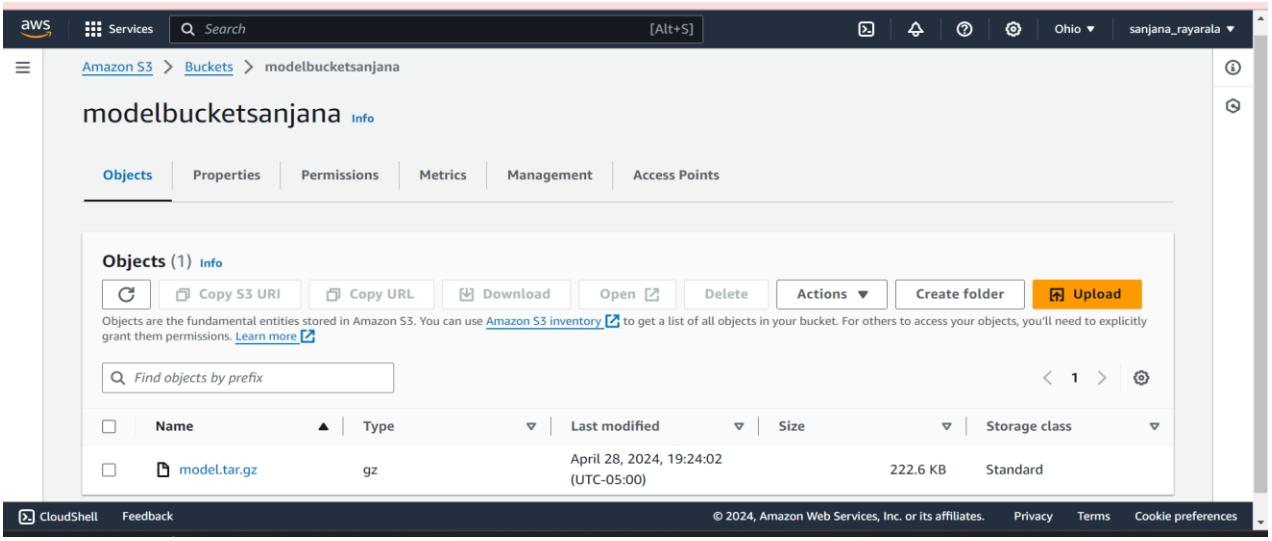
```
model.save('model.h5')
```

Then we convert it into.tar.gz so that it will be stored in an S3 bucket in AWS.

```
import tarfile
import os

# Define the name of the tar.gz archive
tar_filename = "model.tar.gz"
```

```
# Files to include in the tar.gz archive
files_to_archive = ["model.h5"]

# Create a tar.gz archive
with tarfile.open(tar_filename, "w:gz") as tar:
    # Add each file to the archive
    for file in files_to_archive:
        tar.add(file)

# Optionally, delete the original files
for file in files_to_archive:
    os.remove(file)
```

This file is then uploaded into an S3 bucket created, in out case it is ‘modelbucketsanjana’.

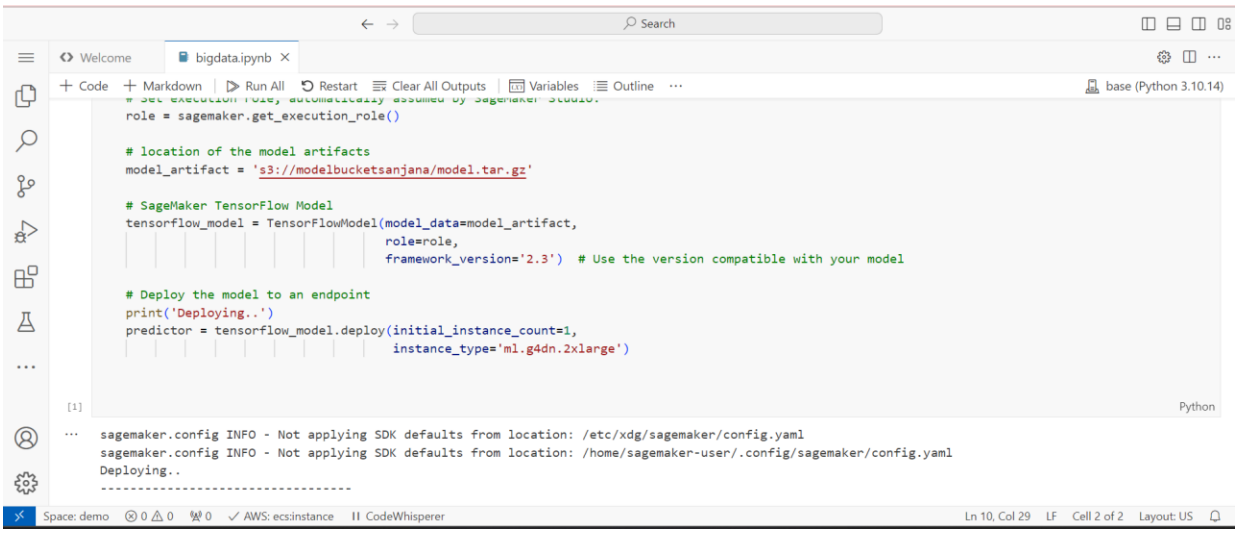


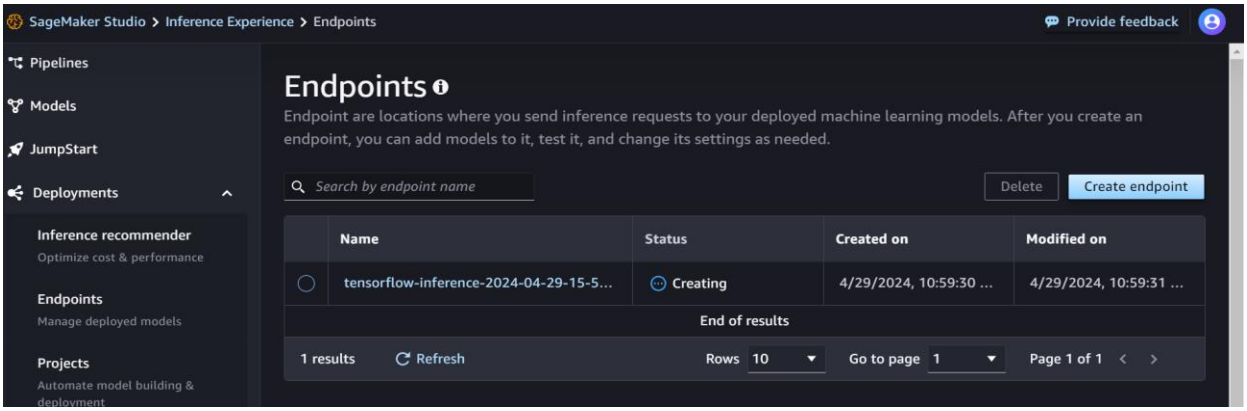
Now we write a script in AWS Sage maker studio, that utilizes this bucket which consists of the model file (model artifact) to establish endpoints and make predictions.

The script begins by importing the necessary modules from the sage maker Python library. sagemaker.tensorflow.TensorFlowModel is specifically used for deploying TensorFlow models on Sage Maker. Then, it retrieves the AWS Identity and Access Management (IAM) role that Sage Maker will assume when executing operations on our behalf. This role needs to have the necessary permissions to access resources like S3 buckets for model artifacts and to deploy instances for model endpoints. The location of the serialized model artifacts in Amazon S3 is specified. These artifacts include the trained TensorFlow model files packaged into a tar.gz file. This path tells

SageMaker where to fetch the model data for deployment. A TensorFlowModel object is created with several parameters:
model_data: The S3 path to the model artifacts.
role: The IAM role SageMaker will use, which we retrieved earlier.
framework_version: Specifies the version of TensorFlow used with this model, ensuring compatibility between the training environment and the deployment environment.
The deploy() method is called on the TensorFlowModel object to create a prediction endpoint. This endpoint will be hosted on a specified type of machine, in this case, ml.g4dn.2xlarge, which is a GPU-accelerated instance suitable for computationally intensive inference tasks.
initial_instance_count: Indicates the number of instances to launch for the endpoint. Here, it is set to 1, starting with a single instance for model deployment.
instance_type: Specifies the type of machine on which the endpoint will be hosted.
The script concludes with deploying the TensorFlow model as a fully managed endpoint on AWS SageMaker, capable of handling real-time inference requests. This endpoint acts as a scalable and secure web server that applications can interact with to make predictions based on new data.

```
import sagemaker
from sagemaker.tensorflow import TensorFlowModel

# Set execution role, automatically assumed by SageMaker Studio.
role = sagemaker.get_execution_role()

# location of the model artifacts
model_artifact = 's3://modelbucketsanjana/model.tar.gz'

# SageMaker TensorFlow Model
tensorflow_model = TensorFlowModel(model_data=model_artifact,
                                   role=role,
                                   framework_version='2.3')  # Use the version compatible with your model

# Deploy the model to an endpoint
print('Deploying..')
predictor = tensorflow_model.deploy(initial_instance_count=1,
                                    instance_type='ml.g4dn.2xlarge')
```





We see that the model has been deployed and endpoint was established.

Now we give a sample input and use the deployed model to predict the Churn %.

data = {"SeniorCitizen": [1], "tenure_new": [0], "MonthlyCharges_new": [5], "gender_Male": [0], "Partner_Yes": [0], "Dependents_Yes": [0], "PhoneService_Yes": [1], "MultipleLines_No phone service": [0], "MultipleLines_Yes": [1], "InternetService_Fiber optic": [1], "InternetService_No": [0], "OnlineSecurity_No internet service": [0], "OnlineSecurity_Yes": [0], "OnlineBackup_No internet service": [0], "OnlineBackup_Yes": [1], "DeviceProtection_No internet service": [0], "DeviceProtection_Yes": [1], "TechSupport_No internet service": [0], "TechSupport_Yes": [0], "StreamingTV_No internet service": [0], "StreamingTV_Yes": [1], "StreamingMovies_No internet service": [0], "StreamingMovies_Yes": [1], "Contract_One year": [0], "Contract_Two year": [0], "PaperlessBilling_Yes": [1], "PaymentMethod_Credit card (automatic)": [0], "PaymentMethod_Electronic check": [1], "PaymentMethod_Mailed check": [0], "TotalCharges_new_1000 - 2000": [0], "TotalCharges_new_2000 - 3000": [0], "TotalCharges_new_3000 - 4000": [0], "TotalCharges_new_4000 - 5000": [0],

```
"TotalCharges_new_5000 - 6000": [0], "TotalCharges_new_6000 - 7000": [0],
"TotalCharges_new_7000+": [1]}

# Deploy the model to an endpoint
prediction = predictor.predict(data)
print(prediction)
```

Output was 72%, anything above 50% is considered as 'YES' according to out model, so the prediction is 'YES'.

With this, we conclude our deployment using AWS Sage maker.

8. Conclusion:

In conclusion, the project on predicting customer churn in the telecommunications industry has been a comprehensive endeavor, this project has addressed the imperative challenge of predicting customer churn within the telecommunications industry through the lens of advanced data analytics and machine learning methodologies.

Leveraging advanced machine learning techniques such as H2O, TensorFlow, and Apache Spark MLlib, our project team successfully developed a churn prediction model capable of analyzing large volumes of customer data to anticipate churn behavior. The deployment of this model on AWS Sage maker or the designated platform ensures its accessibility and scalability, empowering organizations to integrate it seamlessly into their operational workflows. This ensures the dissemination of insights and methodologies essential for organizational learning and continuous improvement. By equipping stakeholders with actionable intelligence derived from predictive analytics, telecommunications companies can refine their retention strategies, minimize churn rates, and foster long-term customer loyalty and satisfaction.

The churn prediction model developed as part of this project equips telecommunications companies with a proactive tool to mitigate customer attrition. By identifying at-risk customers

early and implementing targeted retention strategies informed by machine learning insights, organizations can minimize churn rates and enhance customer satisfaction and loyalty.

In summary, this research underscores the transformative potential of data-driven approaches in addressing complex business challenges, particularly in the realm of customer churn prediction within the telecommunications sector. By embracing innovation and leveraging insights from advanced analytics, organizations can navigate the dynamic landscape of customer relationships with confidence, resilience, and strategic foresight.

9. References:

1. Ahmad, A.K., Jafar, A. & Aljoumaa, K. Customer churn prediction in telecom using machine learning in big data platform. J Big Data 6, 28 (2019). https://doi.org/10.1186/s40537-019-0191-6
2. Mustafa N, Sook Ling L, Abdul Razak SF. Customer churn prediction for telecommunication industry: A Malaysian Case Study. F1000Res. 2021 Dec 13;10:1274. doi: 10.12688/f1000research.73597.1. PMID: 35528953; PMCID: PMC9051585.