**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**



**DATA STRUCTURE AND ALGORITHM**

**FINAL REPORT**

**TOPIC : EXPLORATORY DATA ANALYSIS**

**MEMBER LIST**

| **ID** | **Name** | **Contribution** |
| --- | --- | --- |
| ITDSIU22166 | Nguyễn Minh Đạt | 25% |
| ITDSIU22131 | Nguyễn Hoàng Thiện | 25% |
| ITDSIU22157 | Nguyễn Tất Bách | 25% |
| ITDSIU22128 | Hà Anh Khoa | 25% |

**Table of Contents**

[**CHAPTER I. INTRODUCTION**](#_fib4qlhguvoj)

[1. Abstract](#_p4ht1eo2gr02)

[2. Goal](#_yprj93m067m3)

[3. The tools and techniques used](#_gg8uhxg25woo)

[**CHAPTER II. PROJECT ANALYSIS**](#_48h6431dq11c)

1. Data Preparation

2[. Exploratory Data Analysis (EDA)](#_mwsc1t1htwam)

3. Feature Engineering

3.1. Create Additional Features……………………………………………………...

3.1.1. Tenure Grouping………………………………………………………………

3.1.2. Total Charges Outline…………………………………………………………

3.1.3. Engagement Metric……………………………………………………………

3.2. Perform Feature Selection………………………………………………………

4. Model Development and Interpretation…………………………………………...

4.1. Preprocessing and Split the Train-Test Dataset…………………………………

4.1.1. Handling Categorical Features………………………………………………..

4.1.2. Combining Features…………………………………………………………..

4.1.3. Handling Missing Values……………………………………………………..

4.1.4. Handling the Target Variable…………………………………………………

4.1.5. Building Pipeline for Machine Learning Model………………………………

4.1.6. Split the Dataset after preprocessing into Train-Test…………………………

4.2. Decision Tree Classification……………………………………………………

4.2.1. Import necessary library and Create a Decision Tree Classifier Object……..

4.2.2. Define hyperparameter grid for tuning………………………………………

4.2.3. Set up the evaluator and Cross Validator…………………………………….

4.2.4. Train model and make predictions……………………………………………

4.3. Logistic Regression……………………………………………………………

4.3.1. Import necessary library and Create a Logistic Regression object………….

4.3.2. Define hyperparameter grid for tuning………………………………………

4.3.3. Set up the evaluator and Cross Validator……………………………………...

4.3.4. Train model and make predictions…………………………………………….

4.4 Model Interpretation……………………………………………………………

4.4.1. Accuracy, Precision, Recall and F1-Score…………………………………..

4.4.2. The PR AUC and ROC AUC……………………………………………….

[**CHAPTER III. CONCLUSION**](#_dbrqvypwzx0e)

[**CHAPTER V. REFERENCE**](#_sct6pnniv3ar)

# INTRODUCTION

# ABSTRACT

Customer retention is a critical focus for subscription-based services, with churn prediction playing a vital role in understanding and addressing customer behavior. This project leverages Apache Spark to analyze customer data and identify key factors contributing to subscription cancellations. The dataset includes customer profiles, service interactions, and churn outcomes, enabling comprehensive exploratory data analysis (EDA) to uncover patterns and trends. Key tasks involve visualizing metrics such as churn rates over time, demographic influences on churn, and feature correlations. Insights derived from the analysis will inform predictive modeling efforts and strategies to mitigate churn, enhancing customer satisfaction and business sustainability.

# Goal

The goal that our group wants to achieve when choosing this topic:

* Analyze customer data using Apache Spark to identify factors influencing subscription cancellations (churn).
* Conduct exploratory data analysis (EDA) to uncover patterns, trends, and correlations related to customer churn.
* Build predictive models to detect customers at risk of churning.
* Provide actionable insights to develop strategies for reducing churn rates.
* Support improved customer retention and long-term business growth.

# The tools and techniques used

**Tools**:

* **Apache Spark**: For distributed data processing and scalable analytics.
* **Python**: For data manipulation, visualization, and model development.
* **Jupyter Notebook**: For interactive coding and documentation.
* **Libraries**: Pandas, NumPy, Matplotlib, Seaborn, and PySpark for EDA and data visualization.

**Techniques**:

* **Exploratory Data Analysis (EDA)**: To identify patterns, trends, and correlations in the dataset.
* **Data Visualization**: To illustrate metrics such as churn rate trends, demographic influences, and feature correlations.
* **Feature Engineering**: To preprocess and transform data for predictive modeling.
* **Predictive Modeling**: Utilizing machine learning algorithms (e.g., Logistic Regression, Random Forest, Gradient Boosting) to predict customer churn.
* **Model Evaluation**: Using metrics like accuracy, precision, recall, F1-score, and AUC to assess model performance.

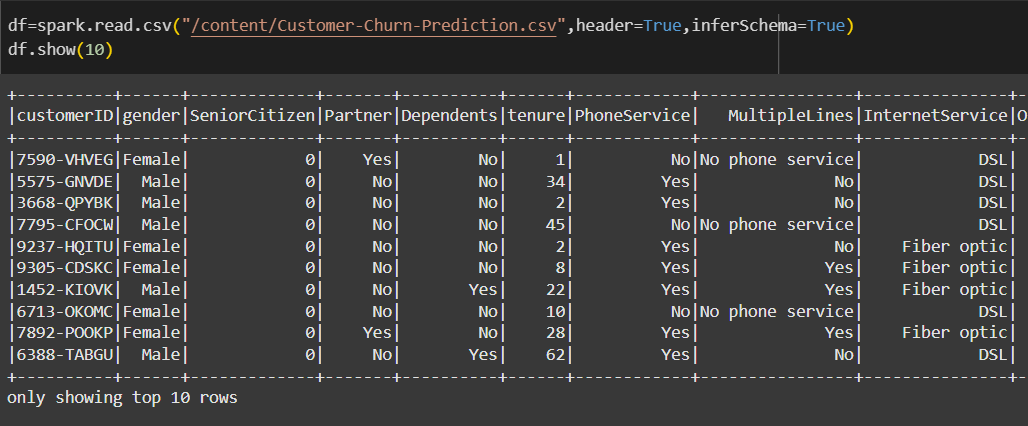
### 

# PROJECT ANALYSIS

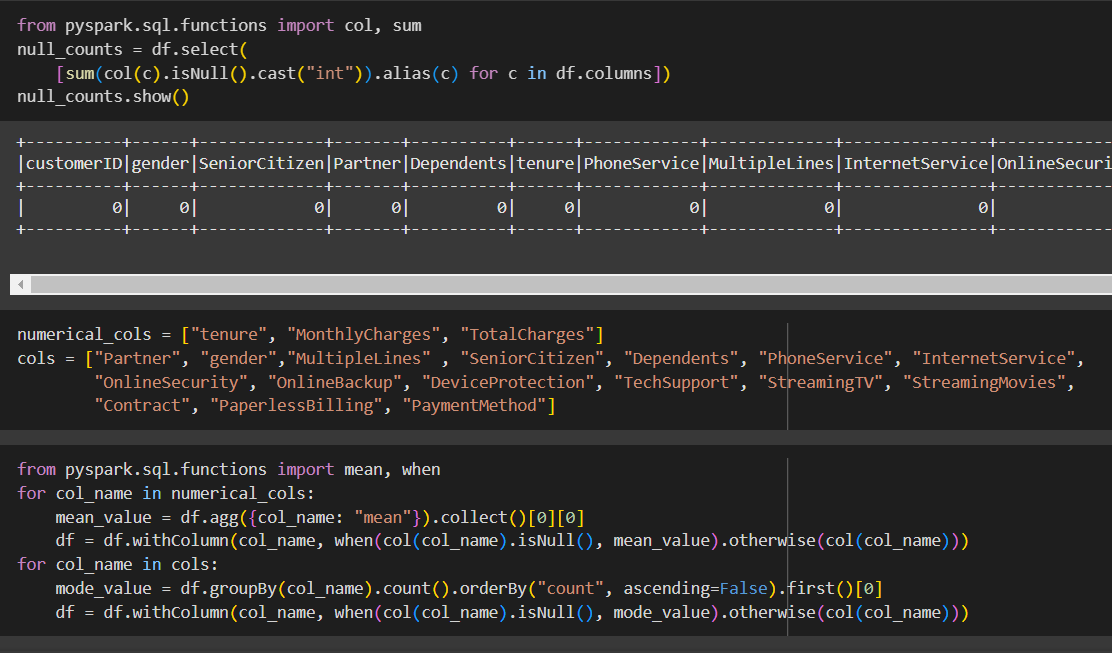
## 1.Data Preparation:

* First of all, we initialize a Spark session, which is the entry point for using Apache Spark to process big data. SparkSession allows access to all of Spark's features, such as SQL, DataFrame, and machine learning.
* Then read a CSV file into a Spark DataFrame. This loads the data into a distributed DataFrame, which is Spark's fundamental data structure for handling large datasets across multiple machines.

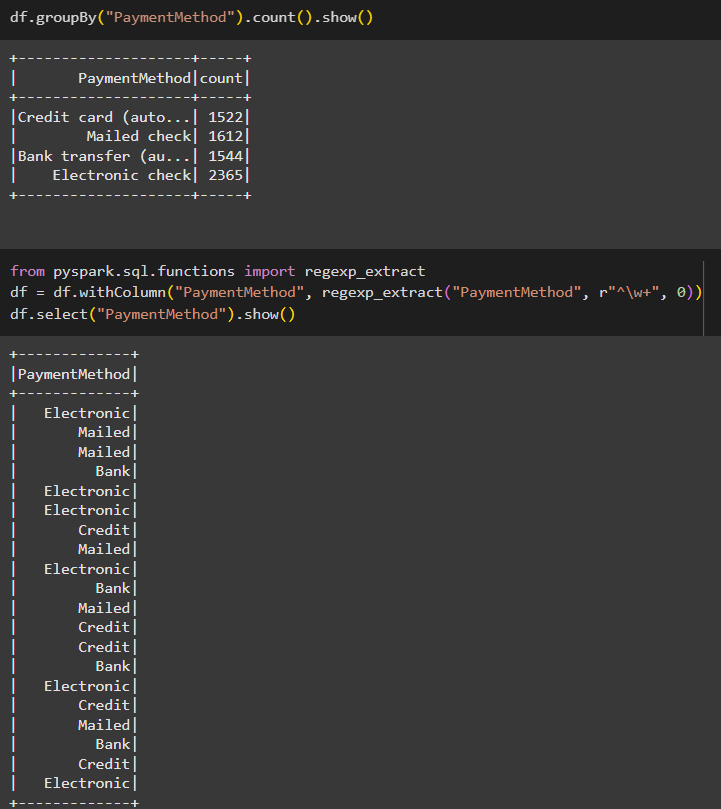




* Preprocess the data by handling missing values:

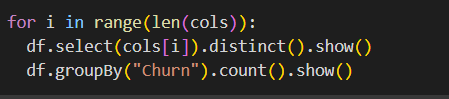


* Preprocessing regex for the variable PaymentMethod:

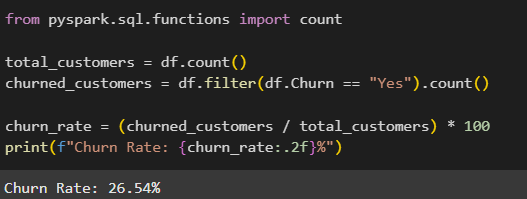


## 2.Exploratory Data Analysis (EDA):

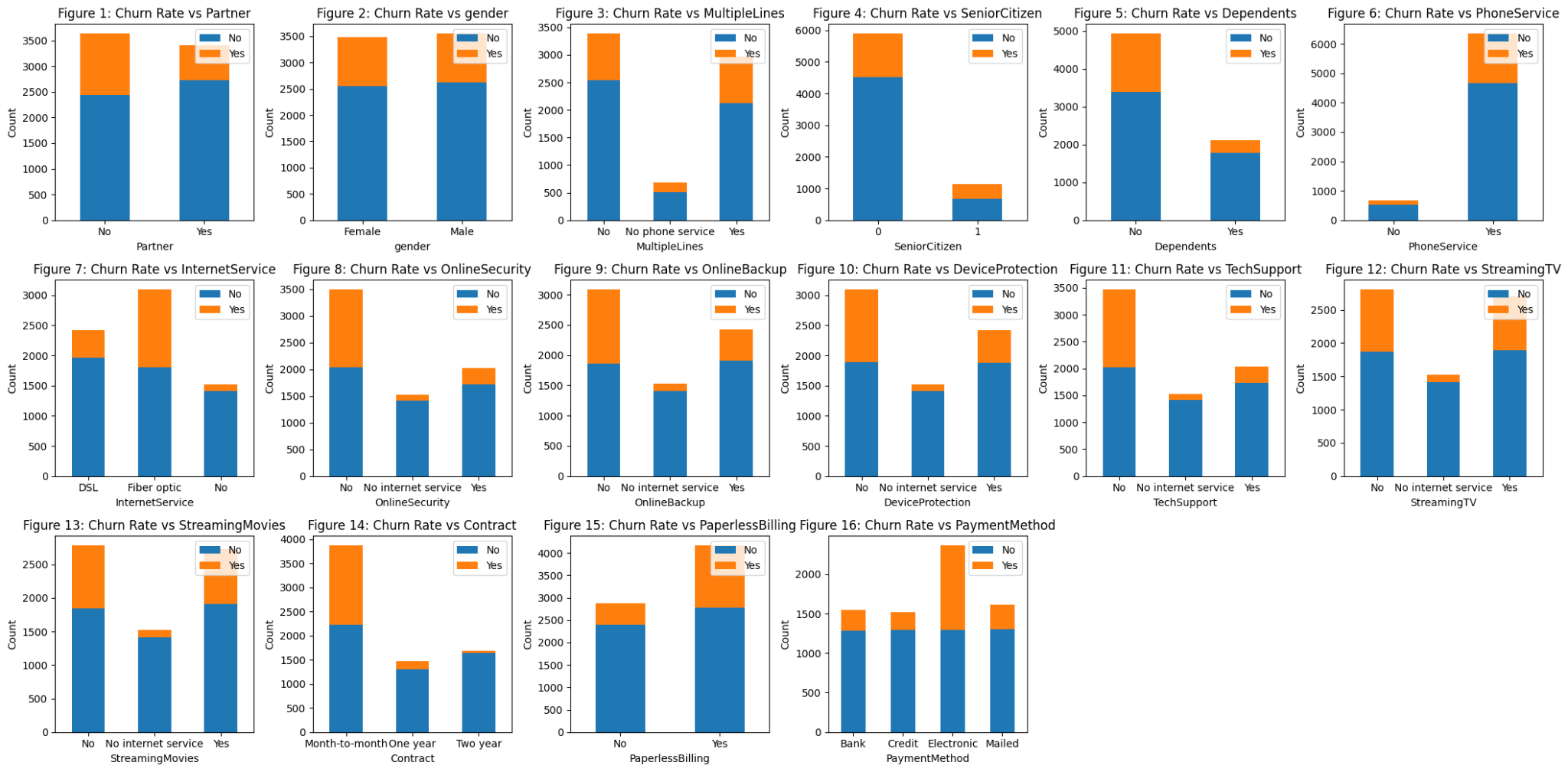
* Check for counts of each variable value:



* Calculate churn rate of “Yes”:

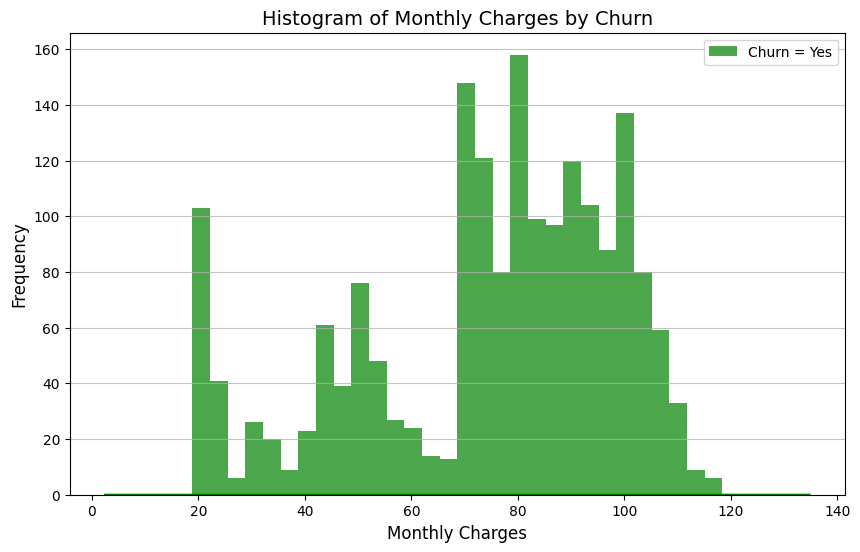


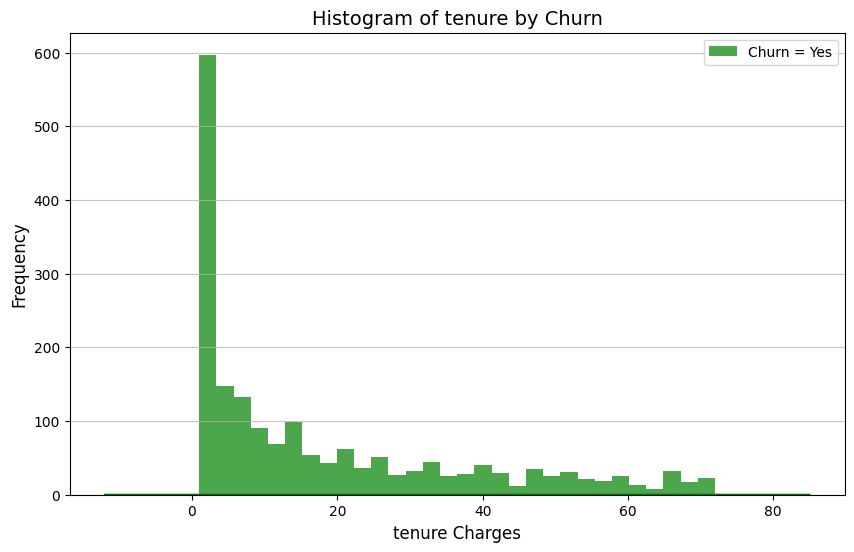
* Plot the bar chart of churn rate for each categorical column:



By plotting these charts, we can easily point out the correlation between churn rate and each categorical variable:

* **Partner & Dependents**: Churn rate is higher for customers without partners or dependents.
* **Gender**: Churn rate is the same for males and females.
* **MultipleLines & PhoneService**: Churn rate increases regardless of having these services.
* **Senior Citizen**: Younger customers show higher churn but also a higher rate of continued usage. Churn rate rises when using phone service.
* **InternetService**: Fiber optic has the highest churn rate, possibly due to complexity.
* **Online Services**: Lack of security, backup, protection, or support increases churn due to reduced safety and assistance.
* **Streaming Services**: Churn rises with or without streaming services unless internet service is absent.
* **Contract**: Longer contracts reduce churn.
* **Billing**: Paperless billing lowers churn compared to paper billing, which is slower and costlier.
* **PaymentMethod**: Electronic checks have the highest churn rate.
* Plot histograms for numerical columns:
* **MonthlyCharges**:



* **Tenure**:

## 3.Feature Engineering:

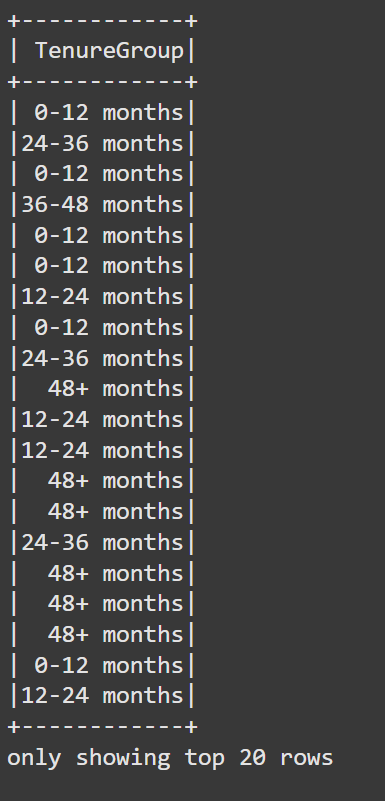
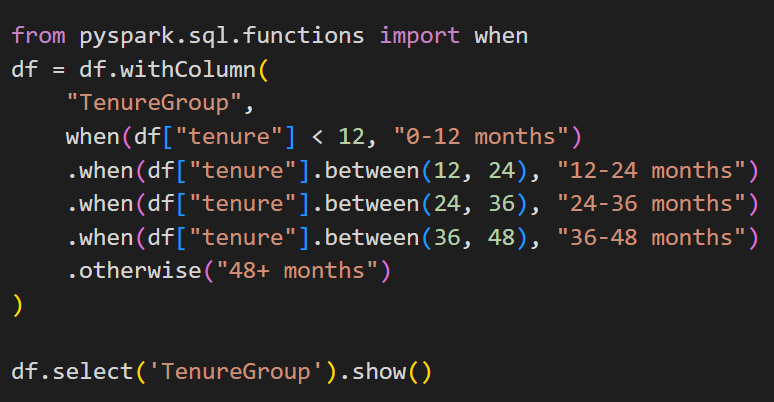
### 3.1.Create additional features:

#### 3.1.1. Tenure Grouping:

We will define a new column named "TenureGroup" based on the existing "tenure" column. The "tenure" column likely represents the length of time a customer has been using a service. The code groups customers into different tenure groups based on their tenure. These groups are:

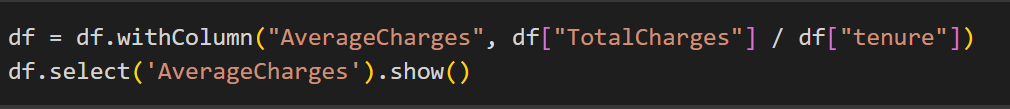
* 0-12 months
* 12-24 months
* 24-36 months
* 36-48 months
* 48+ months

This grouping can be useful for analyzing the relationship between customer tenure and churn (customer cancellation). By segmenting customers by tenure, businesses can identify groups with higher churn rates and target them with specific retention campaigns.



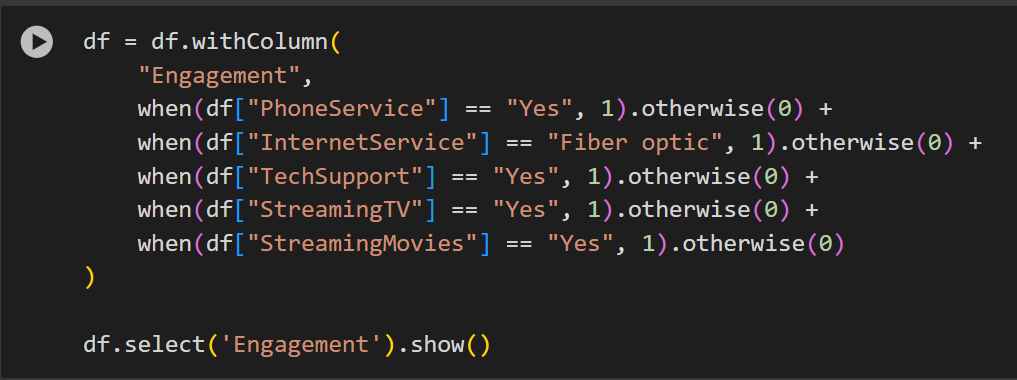
#### 3.1.2. Total Charges per Month (Monthly payment amount):

I'll calculate the average charges per month using the "TotalCharges" and "tenure" columns. This will create a new column called "AverageCharges" that represents the average monthly payment amount for each customer by dividing the "TotalCharges" column by the "tenure" column. This calculation can help identify customers who are paying more or less than average, which can be useful for customer segmentation and targeted marketing:

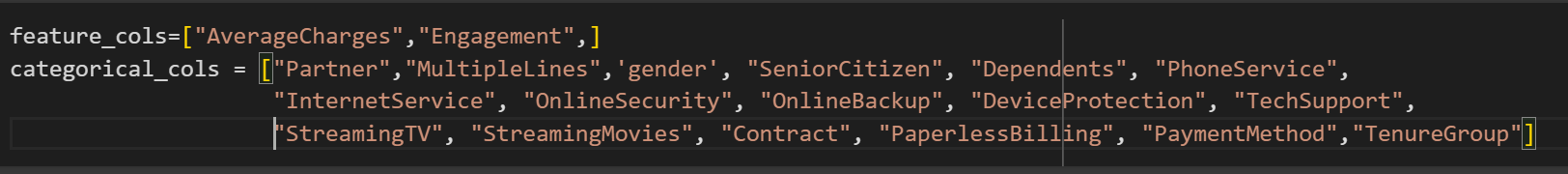


#### 3.1.3. Engagement Metric:

I'll create an "Engagement" metric by assigning a value of 1 for each service the customer uses ("PhoneService", "InternetService" where it's "Fiber optic", "TechSupport", "StreamingTV", "StreamingMovies") and 0 otherwise. Then, I'll sum these values to create an aggregated index representing the customer's overall engagement.



### 3.2.Perform feature selection:

I'll drop the features customerID, TotalCharges, MonthlyCharges, and tenure as they are not relevant for the model. Then, I'll select the numerical features and store them in feature\_cols and select the categorical features and store them in categorical\_cols. By selecting the Features, I can prepare the data for machine learning modeling later.  


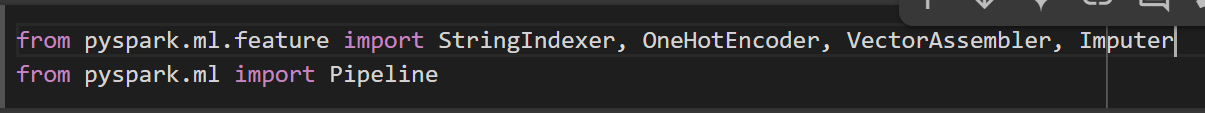
## 4.Model Development and Interpretation:

### 4.1.Preprocessing and split the Train-Test dataset:

We imports four classes used for feature engineering, including:

* **StringIndexer**: This class converts categorical string features into numerical indices. Machine learning algorithms often require numerical input, so StringIndexer prepares the data for these algorithms.
* **OneHotEncoder**: This class takes the indexed categorical features and creates a binary vector representation for each category. This is often necessary to avoid implying an ordinal relationship between categories.
* **VectorAssembler**: This class combines multiple feature columns into a single vector column, which is a standard input format for many machine learning models.
* **Imputer**: This class is used to fill in missing values in the dataset. It provides strategies like using the mean, median, or mode to impute missing values.

Then we build a machine learning Pipeline by importing the Pipeline class: This class helps chain together multiple stages of data transformations and model training into a single workflow. This promotes code organization, reproducibility, and easier model deployment.



#### 4.1.1. Handling Categorical Features:

* **stringindexer\_stages**: This line creates a list of StringIndexer objects.
  + StringIndexer is used to convert categorical string columns (like "InternetService" with values "Fiber optic", "DSL", etc.) into numerical indices (0, 1, 2, etc.).
  + It iterates through each column in categorical\_cols and creates a StringIndexer for it. The original column name is stored in inputCol, and the new column with numerical indices will be named strindexed\_ followed by the original column name (e.g., strindexed\_InternetService).
* **onehotencoder\_stages**: This line creates a list of OneHotEncoder objects.
  + OneHotEncoder takes the indexed categorical columns (strindexed\_...) and creates one-hot encoded vectors.
  + One-hot encoding represents each category as a binary vector (0s and 1s), where only one element is 1 (indicating the presence of that category) and the rest are 0s.
  + The inputCol is the indexed column, and the outputCol will be named onehot\_ followed by the original column name (e.g., onehot\_InternetService).

#### 4.1.2.Combining Features:

* **feature\_columns**: This line creates a list of all the feature columns that will be used in the model. It includes all the one-hot encoded columns (onehot\_...) and the original numerical features (feature\_cols).
* **vectorassembler\_stage**: This line creates a VectorAssembler object.
  + VectorAssembler combines all the selected features into a single vector column named "features."
  + Machine learning algorithms often expect input data in this format.

#### 4.1.3. Handling Missing Values:

* **imputer**: This creates an Imputer object.
  + Imputer is used to fill in missing values in the numerical columns (feature\_cols).
  + It uses the "mean" strategy, meaning it replaces missing values with the average value of that column.
  + inputCols specify the columns to impute, and outputCols specify the names of the imputed columns (which are the same in this case).

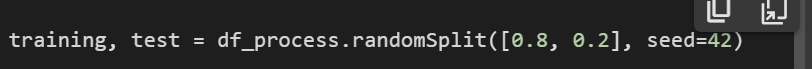
#### 4.1.4. Handling the Target Variable:

* **indexer**: This line creates a StringIndexer specifically for the target variable, "Churn."
  + It converts the "Churn" column (which likely has values like "Yes" and "No") into a numerical label column named "label."
  + This is necessary because most machine learning algorithms expect the target variable to be numerical.

#### 4.1.5. Building Pipeline for machine learning model:

* **all\_stages**: This variable is created by combining three lists: stringindexer\_stages, onehotencoder\_stages, and a list containing indexer, imputer, and vectorassembler\_stage. Each of these lists represents a specific transformation that will be applied to the data.
* **pipeline**: This variable is assigned an instance of the Pipeline class. The Pipeline object is designed to chain multiple data transformation stages together and execute them sequentially. In this case, it's initialized with all\_stages, which means all the transformations we discussed above will be executed in order when the pipeline is run.
* **df\_process:** This variable now stores the data that applies the trained pipeline model and contains only the selected columns. This data is now ready for use in machine learning models**.**

#### 4.1.6. Split the dataset after preprocessing into Train-Test:



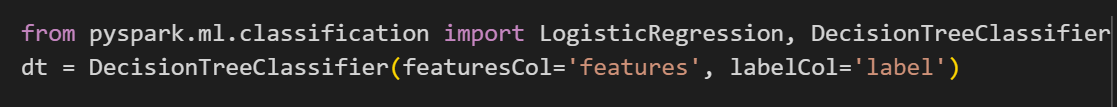
This line is responsible for splitting the preprocessed dataset (df\_process) into two parts: a training set (training) and a testing set (test). This is a crucial step in machine learning model development. We call the **randomSplit** function on the **df\_process** DataFrame. We specifies the weights or proportions for the split: In this case, 80% of the data will go into the training set and 20% into the test set. By doing this, it can help to assess the model's generalization ability and avoid overfitting (where the model performs well on training data but poorly on new data).

In essence, Preprocessing data is a crucial step in preparing the data for machine learning. It transforms categorical features, handles missing values, combines features into a single vector, prepares the target variable, builds and applies the pipeline model to the original data, and splits the after-preprocessing dataset into Train set and Test set– all essential steps before training a model.

### *4.2.Decision Tree Classification:*

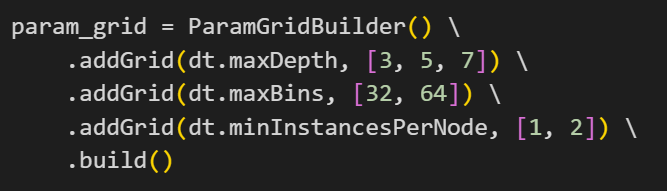
In this section, a decision tree model is implemented using Apache Spark’s MLlib library to predict customer churn. The process involves preparing the dataset by selecting relevant features and encoding categorical variables. The dataset is split into training and testing sets to ensure reliable evaluation. The decision tree algorithm is trained on the processed data to capture patterns and relationships influencing churn. Key hyperparameters, such as maximum depth and minimum instances per node, are fine-tuned to optimize performance. Finally, the model's accuracy, precision, recall, and F1-score are evaluated to assess its effectiveness in predicting customer churn.

#### **4.2.1. Import necessary library and Create a Decision Tree Classifier object:**



We import the DecisionTreeClassifier class which is the core for building the Decision Tree model, create a DecisionTreeClassifier object named dt. featuresCol='features' specifies that the column named 'features' in the dataset contains the input features for the model and labelCol='label' specifies that the column named 'label' contains the target variable (churn or not churn).

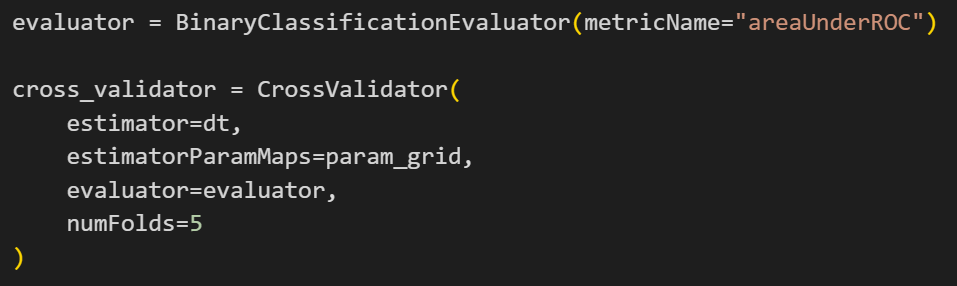
#### **4.2.2. Define hyperparameter grid for tuning:**



This is the code for defining the hyperparameter grid for Decision Tree Classifier. It contains a list of specified values for the tuning model:

* **dt.maxDepth**: Maximum depth of the decision tree (values: 3, 5, 7).
* **dt.maxBins**: Maximum number of bins used for discretizing continuous features (values: 32, 64).
* **dt.minInstancesPerNode**: Minimum number of instances required to be at a leaf node (values: 1, 2).

#### **4.2.3. Set up the evaluator and Cross Validator:**

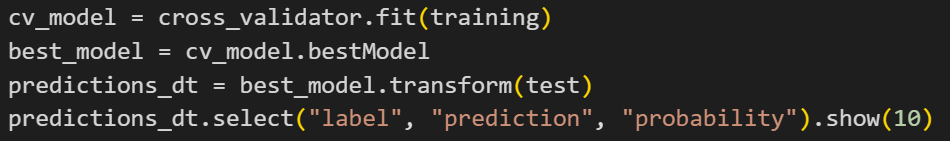


We created a BinaryClassificationEvaluator to assess the model's performance with the metricName: "areaUnderROC" indicating that the area under the ROC curve (AUC) will be used as the evaluation metric.

A CrossValidator is created to perform k-fold cross-validation, which helps in selecting the best model. In this validator, we have the setting like this:

* *estimator=dt:* The decision tree classifier (dt) is used as the base estimator.
* *estimatorParamMaps=param\_grid*: The parameter grid (param\_grid) is provided to test different hyperparameter combinations.
* *evaluator=evaluator*: The binary classification evaluator (evaluator) is used to measure performance.
* *numFolds=5*: The data will be split into 5 folds for cross-validation.

#### **4.2.4. Train model and make predictions:**

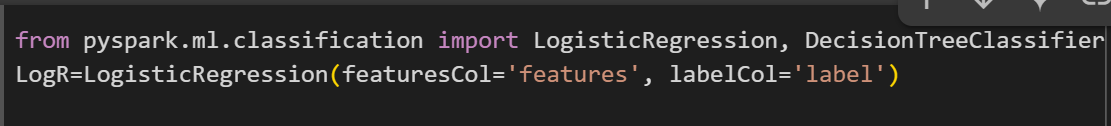


After making the Cross Validator, we will apply it to the training data and retrieve the decision tree model with the best performance based on the evaluation metric (AUC). Then we will use the best model to make predictions on the test data and display the actual labels ('label'), predicted labels ('prediction'), and prediction probabilities ('probability') for the first 10 rows of the test data.

### *4.3.Logistic Regression:*

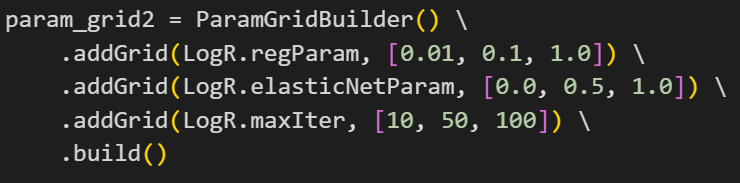
In this section, a logistic regression model is developed using Apache Spark’s MLlib library to predict customer churn. The process begins with data preprocessing, including feature selection, scaling, and encoding categorical variables. The dataset is divided into training and testing subsets to enable robust evaluation. The logistic regression algorithm is trained on the processed data to model the probability of customer churn. Hyperparameters, such as regularization parameters (e.g., L1 and L2 penalties) and the maximum number of iterations, are fine-tuned to enhance model performance. The model's effectiveness is evaluated using metrics such as accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC).

#### **4.3.1. Import necessary library and Create a Logistics Regression object:**



We import the LogisticRegression class which is the core for building the Logistic Regression model, and create a Logistic Regression object named Logr. The featuresCol='features' specifies that the column named 'features' in the dataset contains the input features for the model and labelCol='label' specifies that the column named 'label' contains the target variable (churn or not churn), same as the Decision Tree model.

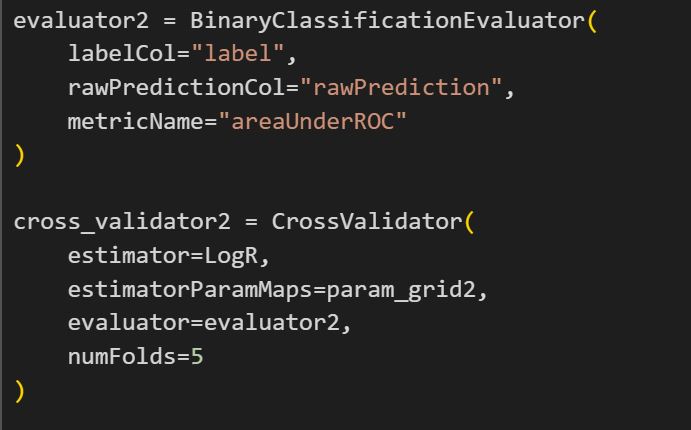
#### **4.3.2. Define hyperparameter grid search for tuning:**



This code defines a grid of hyperparameters that will be tested to find the best combination for the Logistic Regression model. The Parameter Grid Builder will adjust these value for the tuning of Regression model:

* LogR.regParam: Regularization parameter (values: 0.01, 0.1, 1.0)
* LogR.elasticNetParam: Elastic Net mixing parameter (values: 0.0, 0.5, 1.0)
* LogR.maxIter: Maximum number of iterations (values: 10, 50, 100)
* build(): finalizes the hyperparameter grid.

#### **4.3.3. Set up the evaluator and Cross Validator:**

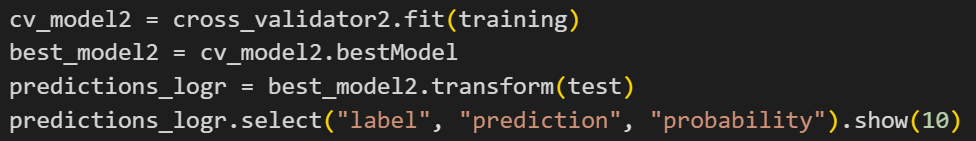


With this model, we will also use the metric ‘Area Under ROC’ for building the evaluator. The labelCol="label" specifies the column containing the true labels and rawPredictionCol="rawPrediction" specifies the column containing the model's raw predictions.

Then we also create a Cross Validator same as the Decision Tree model to perform k-fold cross-validation. This validator will contain:

* estimator=LogR: The Logistic Regression model to be evaluated.
* estimatorParamMaps=param\_grid2: The hyperparameter grid to search.
* evaluator=evaluator2: The evaluator to use for model assessment.
* numFolds=5: The number of folds for cross-validation (in this case, 5).

#### **4.3.4. Train model and make predictions:**



After making the Cross Validator, we will apply it to train the Logistic Regression model using cross-validation on the training data and retrieve the best-performing model from the cross-validation process. Then we will use the chosen model to make predictions on the test data and display the actual labels ('label'), predicted labels ('prediction'), and prediction probabilities ('probability') for the first 10 rows of the test data.

In summary, we just build a Logistic Regression model, tune its hyperparameters using cross-validation, evaluate its performance using the area under the ROC curve, and generate predictions on a test dataset.

### *4.4.Model Interpretation:*

#### 4.4.1. Accuracy, Precision and Recall of 2 Models:

* Decision Tree:
  + Accuracy (0.7866): This indicates that the model correctly predicted the target variable (Churn or Not Churn) for about 78.66% of the instances in the dataset.
  + Precision (0.7857): Out of all the customers predicted to churn, about 78.57% actually churned. This metric reflects the model's ability to avoid false positives.
  + Recall (0.7866): The model identified 78.66% of all actual churn cases correctly. This metric measures the ability to capture true positives.
* Logistic Regression:
  + Accuracy (0.8097): Logistic regression achieved a higher accuracy, correctly predicting about 80.97% of the instances.
  + Precision (0.8025): Out of all customers predicted to churn, 80.25% actually churned, showing better avoidance of false positives compared to the Decision Tree.
  + Recall (0.8097): Logistic regression identified 80.97% of all actual churn cases, demonstrating superior detection of true positives.

Logistic Regression appears to be slightly more accurate in predicting customer churn on this dataset. This suggests that, in this specific instance, the linear decision boundary learned by Logistic Regression might better capture the underlying patterns in the data compared to the tree-based approach of the Decision Tree.

The lower accuracy of the Decision Tree compared to Logistic Regression can be attributed to a few factors:

* Overfitting: Decision Trees tend to overfit the training data, especially when they are not properly pruned or regularized. This leads to lower generalization on unseen data, reducing accuracy and precision. Logistic Regression is less prone to overfitting because it assumes a linear relationship between the features and the target.
* Handling of Relationships and Noise: Logistic Regression captures linear relationships effectively, while Decision Trees can struggle with noisy data or features that do not have clear thresholds. Logistic Regression benefits from its probabilistic nature, which helps in smoother decision boundaries.
* Feature Importance: Decision Trees heavily rely on splits, which can sometimes lead to suboptimal decisions if the dataset contains features with minor but combined importance. Logistic Regression assigns weights to features, capturing even subtle contributions better.

#### 4.4.2. Identify actionable insights:

* Senior Citizens: Higher churn rates might be observed in this demographic. Tailored offerings or support for older customers can help reduce churn.
* Tenure: Customers with shorter tenure are more likely to churn. Providing incentives such as discounts or loyalty programs for new customers during the initial months may improve retention.
* Contract Type: Long-term contracts (e.g., annual) usually have lower churn rates. Encourage customers to opt for longer contracts through discounts or added benefits.
* Paperless Billing: Digital-savvy customers may prefer paperless billing, but those uncomfortable with technology might perceive it as inconvenient. Offering flexibility in billing preferences can improve customer satisfaction.
* Payment Method: Certain payment methods may correlate with churn. If customers using a particular method (e.g., manual payments) have higher churn, consider educating them on the benefits of automated payments.
* Internet Service and Add-Ons: Customers with certain internet services or without add-ons like Online Security or Tech Support might feel less valued or underserved. Bundling these services at a discount may increase perceived value.
* Monthly Charges and Total Charges: High monthly charges could lead to dissatisfaction. Consider tiered pricing or customized plans based on usage to improve affordability.

#### 4.4.3. Actionable Recommendations:

* Customer Engagement Strategies
  + Implement personalized retention campaigns for Senior Citizens and customers with shorter tenures.
  + Use predictive analytics to identify at-risk customers early and provide proactive outreach.
* Offer Customizable Plans
  + Provide flexible plans that cater to different customer needs, focusing on affordability and perceived value.
  + Introduce incentives for customers to opt for longer-term contracts.
* Enhance Value Through Bundling
  + Promote bundles that include services like Online Security, Tech Support, and Streaming options to increase satisfaction.
  + Offer free trials of add-on services to encourage adoption.
* Simplify the Payment Process
  + Encourage auto-payments or digital wallets to reduce churn associated with payment issues.
  + Offer incentives for using preferred payment methods.
* Improve Customer Support
  + Strengthen customer service to address pain points such as technical support and billing clarity.
  + Use feedback mechanisms to identify and resolve issues leading to dissatisfaction.
* Reward Loyalty
  + Introduce loyalty programs rewarding long-term customers with discounts, upgrades, or exclusive offers.
  + Offer tenure-based perks to motivate customers to remain longer.

# CONCLUSION

The project demonstrated the use of Apache Spark to analyze customer churn data and build predictive models, including decision tree and logistic regression algorithms. Through exploratory data analysis, significant factors contributing to customer churn, such as numerical features like TotalCharges and tenure, were identified. The logistic regression model outperformed the decision tree model in terms of accuracy, primarily because logistic regression effectively handles numerical features and captures linear relationships, whereas decision trees may struggle with such data without proper preprocessing. These findings highlight the importance of selecting appropriate models based on dataset characteristics. The insights derived can guide targeted strategies to reduce churn, improve customer retention, and support data-driven decision-making for subscription-based businesses.

1. **REFERENCES**
2. Apache Spark documentation:
3. Python libraries documentation:

* Pandas Documentation: https://pandas.pydata.org/docs/
* NumPy Documentation: https://numpy.org/doc/
* Matplotlib Documentation: https://matplotlib.org/stable/contents.html
* Seaborn Documentation: https://seaborn.pydata.org/

1. Dataset source:

* <https://www.kaggle.com/datasets/ranasarkar15/customerchurndatasets?fbclid=IwY2xjawH0n09leHRuA2FlbQIxMAABHWblVbCeyH8vc8-ikHzo79O_6tQNgB8DfPktbaSk17-5uvj61xZYHHM2TQ_aem_JgWSpeeXHunHe761Ds17TA>