CAPSTONE PROJECT ANALYSIS-

Telecom Churn

By-Anurag Nirwan

Problem Statement

- In the telecom sector, customers have numerous options when it comes to service providers, and they frequently switch between operators. This intense competition has led to an industry churn rate of 15-25% annually. Since acquiring new customers can be 5-10 times more costly than retaining current ones, customer retention has become a higher priority than customer acquisition.

- For many established telecom companies, the top business objective is to retain their most profitable customers.

- To address customer churn, telecom companies must predict which customers are most likely to leave.

- In this project, we will analyze customer-level data from a leading telecom company, build predictive models to forecast high-risk churn customers, and identify the key drivers of churn.

Steps to Approach

- Step 1:
  - Data reading and understanding
  - Data cleaning and imputing missing values
- Step 2:
  - Filter out high-value customers
- Step 3:
  - Derive the churn target variable
- Step 4:
  - Data preparation, including creating derived variables and performing exploratory data analysis (EDA)
  - Split the data into training and testing sets
  - Apply scaling
- Step 5:
  - Address class imbalance
  - Perform dimensionality reduction using PCA
  - Build classification models to predict churn using various algorithms
- Step 6:
  - Model evaluation
  - Prepare models for predictor variable selection, testing multiple models and choosing the best one
  - The goal is to provide the company with a well-rounded summary and insights based on the best-performing model.

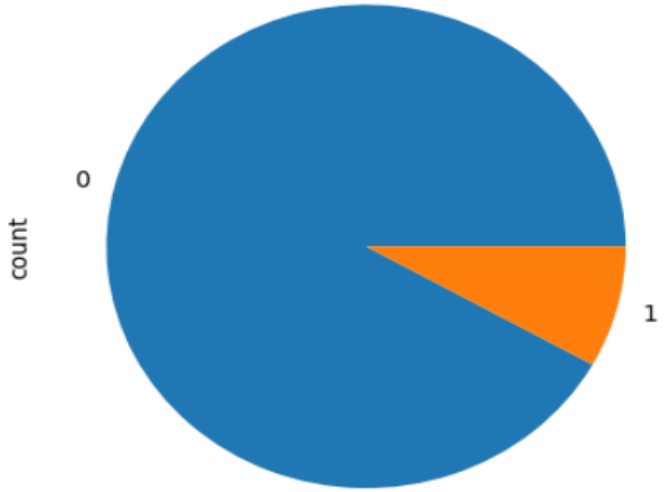
Analysis

- DATA INFO- <class 'pandas.core.frame.DataFrame'> RangeIndex: 99999 entries, 0 to 99998 Columns: 226 entries, mobile_number to sep_vbc_3g dtypes: float64(179), int64(35), object(12) memory usage: 172.4+ MB.
- This telecom dataset has 99999 rows and 226 columns
- DUPLICATES- (99999, 226)- There are no duplicates
- CHECKING NULL VALUES- As We can see more then 74% values for recharge related data are missing.
- In churn prediction, we assume that there are three phases of customer lifecycle :
  - The ‘good’ phase [Month 6 & 7] . The ‘action’ phase [Month 8] . The ‘churn’ phase [Month 9] In this case, since you are working over a four-month window, the first two months are the ‘good’ phase, the third month is the ‘action’ phase, while the fourth month is the ‘churn’ phase.

FINDING OUT CHURN %AGE

[52]:
```
# lets find out churn/non churn percentage
print((churn_filtered['churn'].value_counts()/len(churn))*100)
((churn_filtered['churn'].value_counts()/len(churn))*100).plot(kind="pie")
plt.show()
```

```
churn
0    27.560276
1     2.441024
Name: count, dtype: float64
```

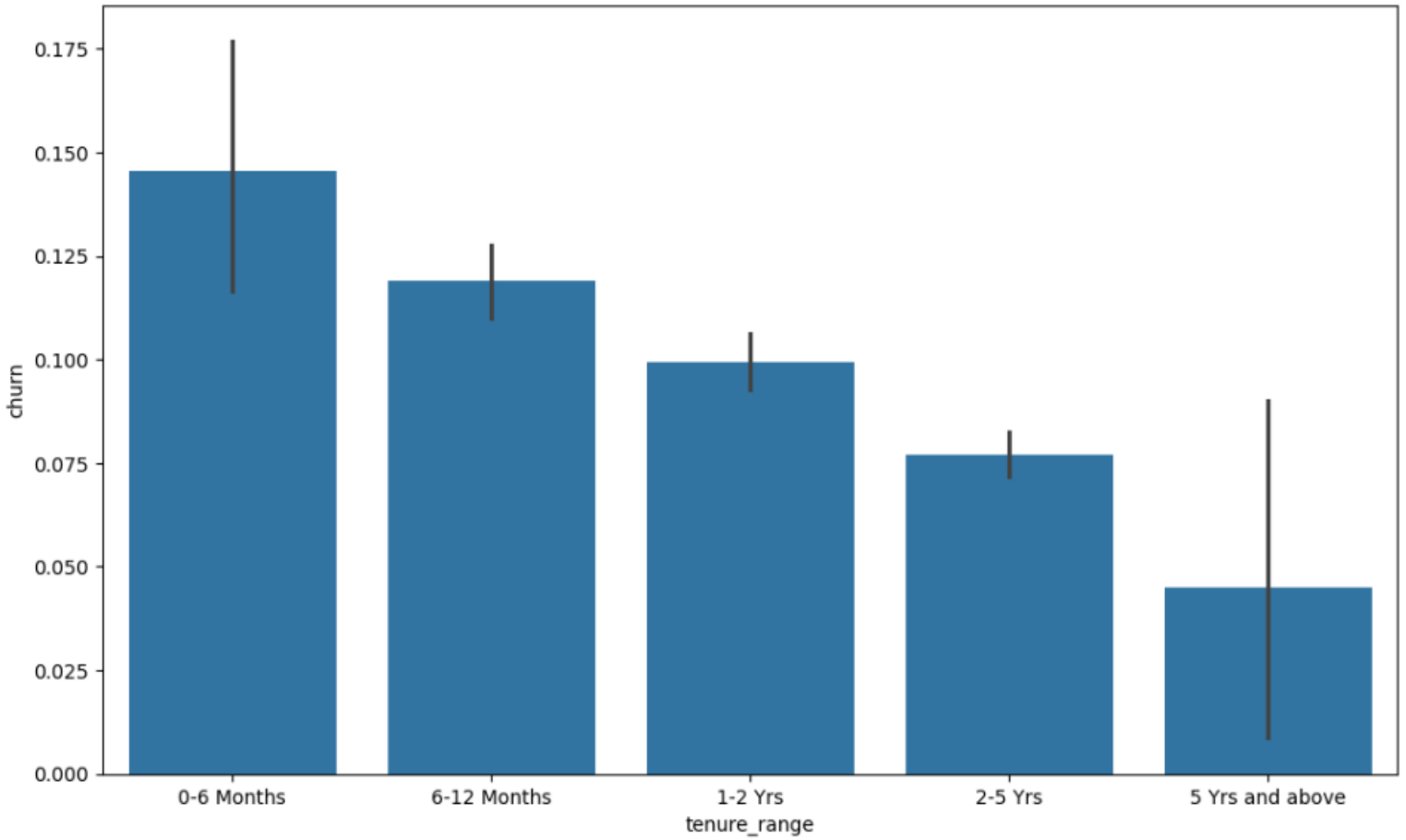


As we can see that 97% of the customers do not churn, there is a possibility of class imbalance

Since this variable churn is the target variable, all the columns relating to this variable(i.e. all columns with suffix _9) can be dropped forn the dataset.

TENURE RANGE

[481]:
```
# Plotting a bar plot for tenure range
plt.figure(figsize=[12,7])
sns.barplot(x='tenure_range',y='churn', data=churn_filtered)
plt.show()
```



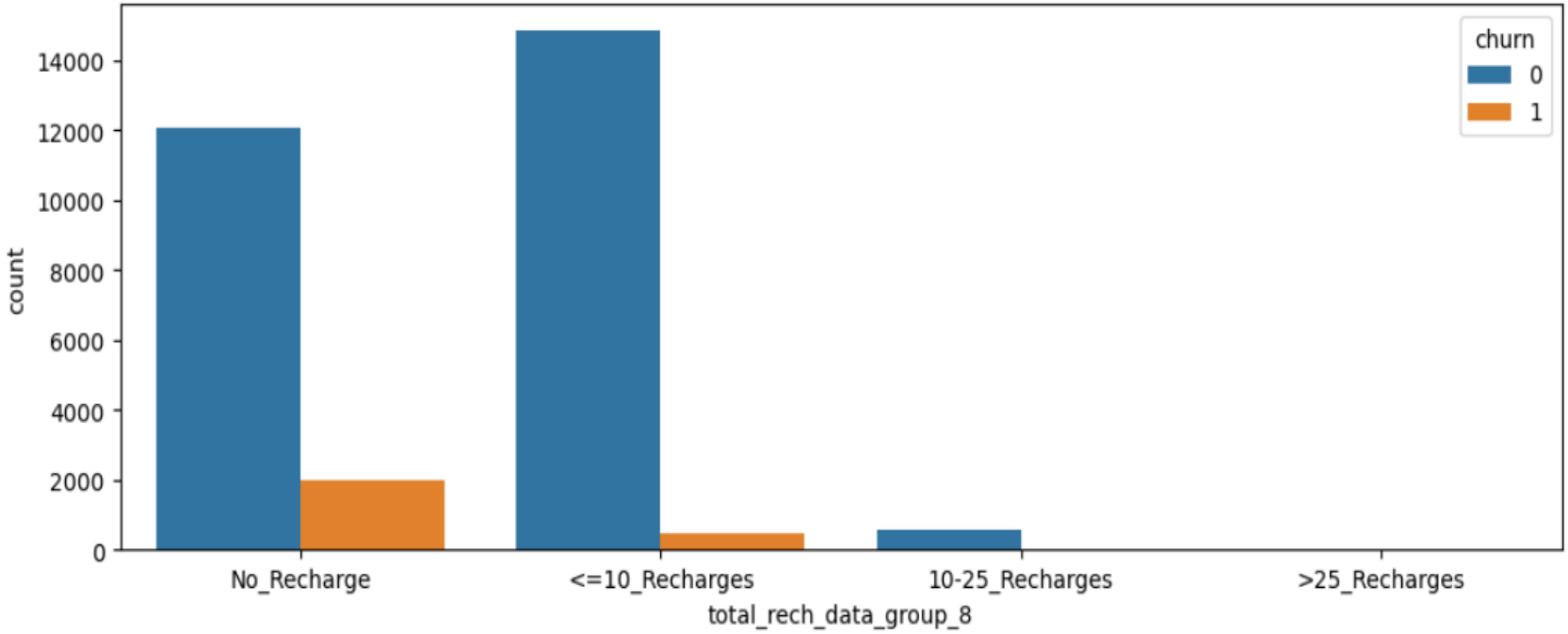
It can be seen that the maximum churn rate happens within 0-6 month, but it gradually decreases as the customer retains in the network.

The average revenue per user is good phase of customer is given by arpu_6 and arpu_7. since we have two seperate averages, lets take an average to these two and drop the other columns

RECHARGE DATA (I)

Distribution of total_rech_data_8 variable

```
 total_rech_data_group_8
<=10_Recharges      15307
No_Recharge         14048
10-25_Recharges       608
>25_Recharges          38
Name: count, dtype: int64
```

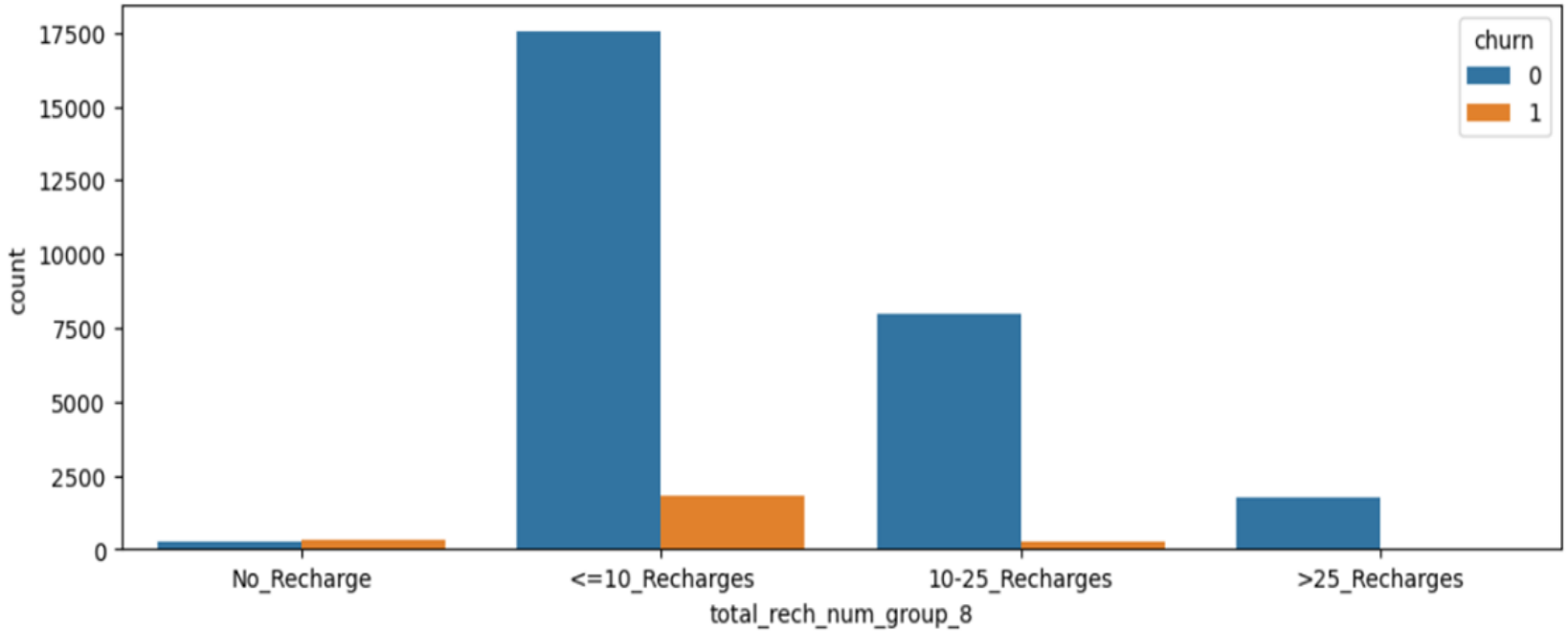


Distribution of total_rech_num_8 variable

RECHARGE DATA (II)

Distribution of total_rech_num_8 variable

```
 total_rech_num_group_8
<=10_Recharges      19349
10-25_Recharges      8245
>25_Recharges        1824
No_Recharge           583
Name: count, dtype: int64
```



As the number of recharge rate increases, the churn rate decreases clearly.

Data Set Split and Train

```
[501]: # divide data into train and test
       X = churn_filtered.drop("churn", axis = 1)
       y = churn_filtered.churn
       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 4, stratify = y)

[502]: # print shapes of train and test sets
       X_train.shape

[502]: (22500, 43)

[503]: y_train.shape

[503]: (22500,)

[504]: X_test.shape

[504]: (7501, 43)

[505]: y_test.shape

[505]: (7501,)
```

- In machine learning, one of the critical steps in model development is to divide the dataset into training and testing sets. This ensures the model's ability to generalize on unseen data and helps prevent issues like over fitting. The code snippet provided demonstrates how to implement this step for a dataset that predicts customer churn.

- # Step-by-Step Explanation:

1. ## Creating Feature Set (X) and Target Variable (y)

- The first two lines of code perform an essential task: separating the independent variables (features) and the dependent variable (target).
- Here, X represents the feature set. It is obtained by removing the churn column from the dataset churn_filtered using the drop() function. The axis=1 argument specifies that the column (not row) is being removed.
- The features in X might include attributes like customer demographics, service usage, and other behavioral indicators, all of which are used to predict churn.
- y is the target variable, which stores the churn column, representing whether a customer has churned (1) or not (0). This is the value the model will attempt to predict based on the features in X.

2. Splitting the Data

- The next line divides the dataset into training and testing sets using the train_test_split() function from the sklearn.model_selection module:
- Parameters:
  - test_size=0.25: This indicates that 25% of the data will be used as the testing set, while the remaining 75% will be used for training the model. This ratio is common, though it can be adjusted based on the dataset size and model type.
  - random_state=4: The random_state is a seed used to shuffle the data before splitting. Using a fixed random state ensures that the data is split the same way each time the code is run, which is critical for reproducibility.
  - stratify=y: This ensures that the class distribution (churn vs. non-churn) is preserved in both the training and testing sets. If the dataset is imbalanced (e.g., there are significantly more non-churn cases than churn cases), using stratify=y ensures that both the training and test sets maintain the same proportion of churn cases. This is important for ensuring fair model evaluation.

Data Imbalance Handling

Data Imbalance Handling

Using SMOTE method, we can balance the data w.r.t. churn variable and proceed further

[511]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_train_sm,y_train_sm = sm.fit_resample(X_train,y_train)
```

[512]:
```
print("Dimension of X_train_sm Shape:", X_train_sm.shape)
print("Dimension of y_train_sm Shape:", y_train_sm.shape)
```

```
Dimension of X_train_sm Shape: (41338, 43)
Dimension of y_train_sm Shape: (41338,)
```

Why Use SMOTE?

- **Class Imbalance**: In many datasets, one class may dominate, leading to biased models that perform poorly on the minority class.
- **Synthetic Data**: SMOTE generates new samples for the minority class by interpolating between existing examples, making it a useful technique for enhancing the model's ability to learn from the minority class without simply duplicating existing samples.
- Using SMOTE is particularly beneficial when you want to improve the predictive performance of models on imbalanced datasets, especially for classification tasks.

- The shapes of X_train_sm and y_train_sm after applying SMOTE provide insight into the structure of your training data and how it has changed due to the oversampling process. Let's break down the dimensions:
- X_train_sm Shape: (41338, 43)
- 41338: This is the number of samples (or observations) in the training dataset after applying SMOTE. It indicates that the dataset now contains 41,338 instances.
- 43: This number represents the number of features (or attributes) in each sample. So, each instance has 43 different characteristics that are used as input for the model.
- y_train_sm Shape: (41338,)
- 41338: This is the number of labels corresponding to the instances in X_train_sm. It confirms that there are 41,338 labels, one for each instance in the features dataset.

Implications

- Balanced Dataset: The fact that both X_train_sm and y_train_sm have the same number of samples (41,338) indicates that the dataset is now balanced. This means the number of instances for each class in y_train_sm is likely equal or closer to equal, which is the goal of using SMOTE.
- Model Training: With a balanced dataset, machine learning models can learn better representations of both classes, improving overall performance, especially for the minority class.
- Feature Importance: The presence of 43 features means you may have a rich set of information available for model training. This could allow for complex decision boundaries, depending on how informative those features are.
- Overall, the transformation applied by SMOTE has increased the number of samples in your training set, which can help mitigate issues related to class imbalance during model training.

Note- refer python file for more

Model Accuracy (few sample)

```
evaluate_model(dt_best)

Train Accuracy : 0.9315833333333333
Train Confusion Matrix:
[[21734   292]
 [ 1350   624]]
--------------------------------------------------
Test Accuracy : 0.9220129978336944
Test Confusion Matrix:
[[5430  104]
 [ 364  103]]
```

Decision tree with PCA

```
evaluate_model(rfc_model)

Train Accuracy : 0.91775
Train Confusion Matrix:
[[22026     0]
 [ 1974     0]]
--------------------------------------------------
Test Accuracy : 0.9221796367272121
Test Confusion Matrix:
[[5534    0]
 [ 467    0]]
```

Random forest with PCA

- **Details:**

After cleaning the data, we broadly employed three models as mentioned below including some variations within these models in order to arrive at the best model in each of the cases.

- **Logistic Regression :**

Logistic Regression with RFE Logistic regression with PCA Random Forest For each of these models, the summary of performance measures are as follows:

- **Logistic Regression:** Train Accuracy : ~79% . Test Accuracy : ~80%
- **Logistic regression with PCA :**Train Accuracy : ~91% . Test Accuracy : ~92%
- **Decision Tree with PCA:** Train Accuracy : ~93% . Test Accuracy : ~92%
- **Random Forest with PCA:** . Train Accuracy :~ 91% . Test Accuracy :~ 92%

Conclusion

Note:

- Note that the best parameters procured the accuracy of 91% which is not significantly deterred than the accuracy of original random forest, which is pegged around 92%

Conclusion :

- The best model to predict the churn is observed to be Random Forest based on the accuracy as performance measure.
- The incoming calls (with local same operator mobile/other operator mobile/fixed lines, STD or Special) plays a vital role in understanding the possibility of churn. Hence, the operator should focus on incoming calls data and has to provide some kind of special offers to the customers whose incoming calls turning lower.