CAPSTONE PROJECT -

CHURN MODELLING

-By Nitisha Tyagi, Vanshika Sethi, R Venkat, Amit and Santhosh Kumar

INTRODUCTION

In the intricate realm of commerce, a puzzling occurrence lurks within the corridors of large corporations, particularly in the Telecom industry: the elusive specter of customer churn. This mysterious presence presents a notable threat, sparking worries about profitability and triggering a vital quest to pinpoint those on the brink of leaving. This sets the stage for a demanding effort to avert this imminent outcome by implementing strategic interventions and proactive steps.

The task is to build for the Telecom company a Logistic Regression Machine Learning model that predicts which of their customers are likely to churn (stop using their service in future)

Tools Used - Python, Canva

DATA DESCRIPTION

The dataset provided for this activity consists of 11 features where 10 are independent features and 1 is a target variable. Features in this dataset are described as below:

- **Churn** : 1 if customer cancelled service, 0 if not (Target)
- **AccountWeeks** : number of weeks customer has had active account
- **ContractRenewal** : 1 if customer recently renewed contract, 0 if not
- **DataPlan** : 1 if customer has data plan, 0 if not
- **DataUsage** : gigabytes of monthly data usage
- **CustServCalls** : number of calls into customer service
- **DayMins** : average daytime minutes per month
- **DayCalls** : average number of daytime calls
- **MonthlyCharge** : average monthly bill
- **OverageFee** : largest overage fee in last 12 months
- **RoamMins** : average number of roaming minutes

STEPS TAKEN-

Data Understanding and Preparationd

EDA

Choosing the Model

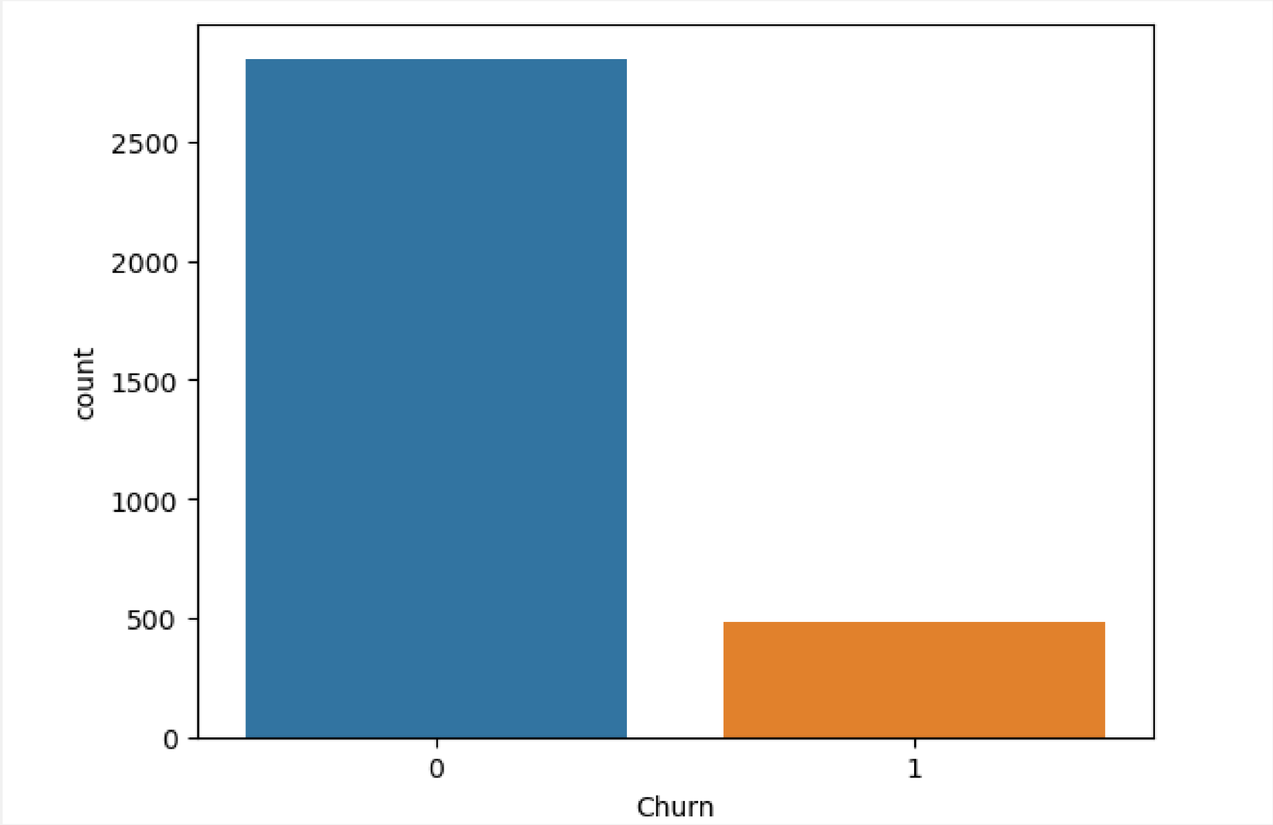
Model Building

Comparing the Models

Model Deployment

Bi- Variant Plots

Churn Rate Visualization

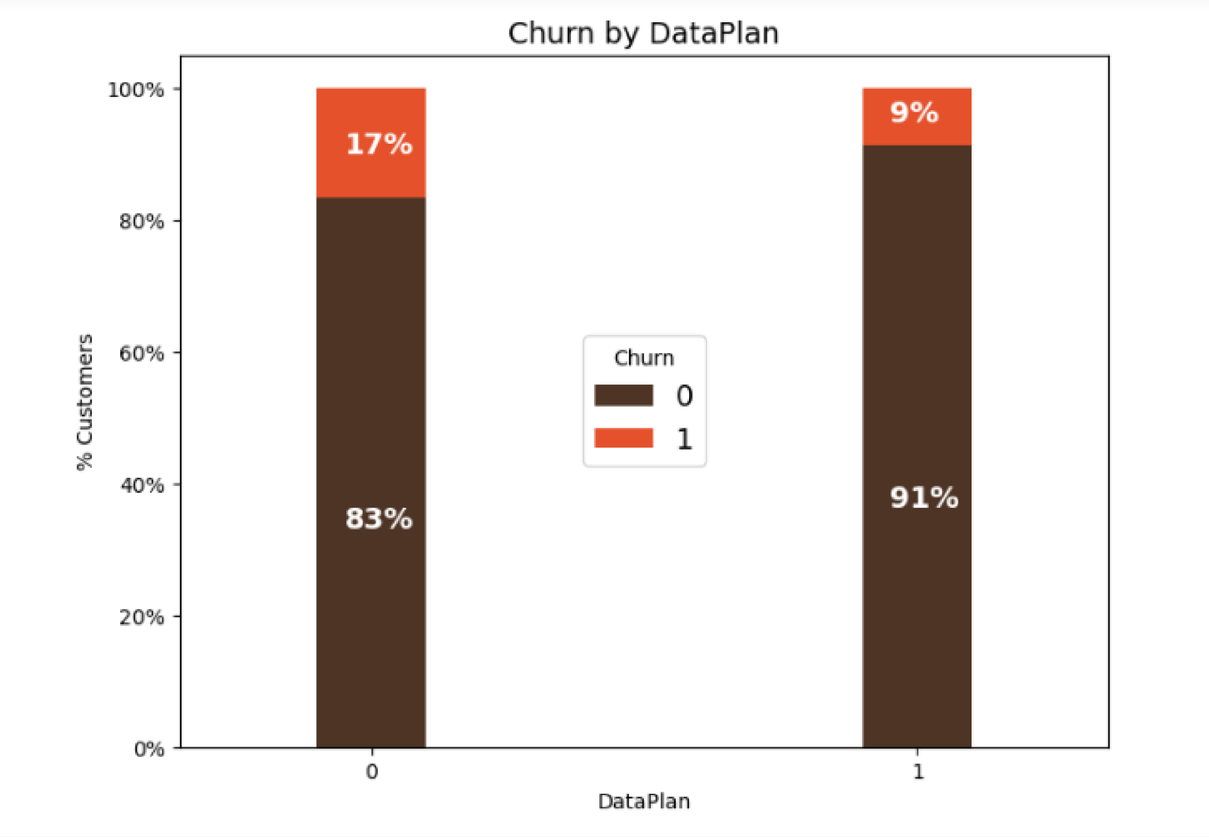


The bar chart depicting Churn value counts offers a clear visualization of the churn rate. It is imperative to delve into the reasons behind churn and formulate strategies to enhance customer retention.

Reasons for churn may include:

- Poor customer service
- Perceived lack of value
- Superior competitive offerings
- Product or service dissatisfaction

Data Plan Effect

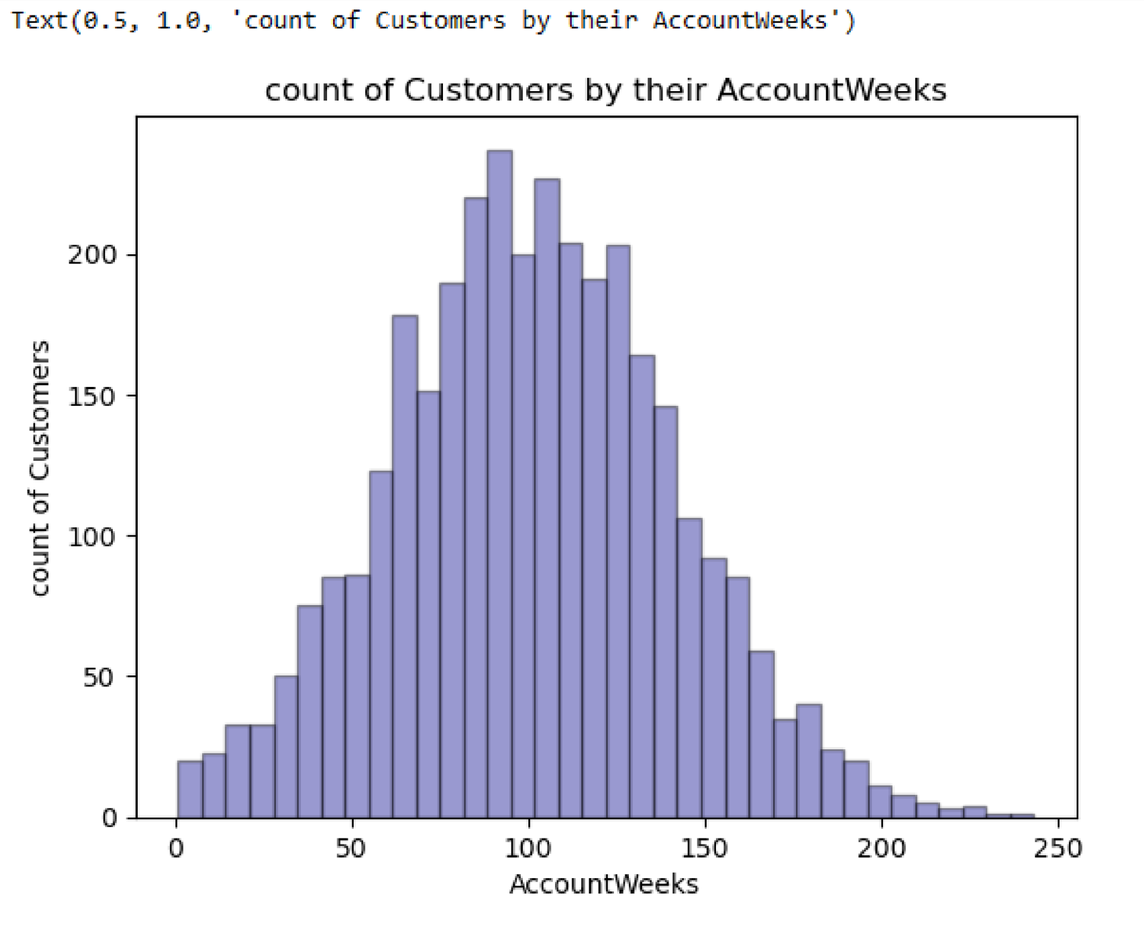


Analyzing the churn percentage among customers with and without a data plan can reveal the value customers place on data services.

Recommendation to influence data plan retention:

- Enhance data plan features or pricing to influence customer decisions to stay. This might include offering more data at the same price or bundling additional services with the data plan

Customer Tenure Analysis

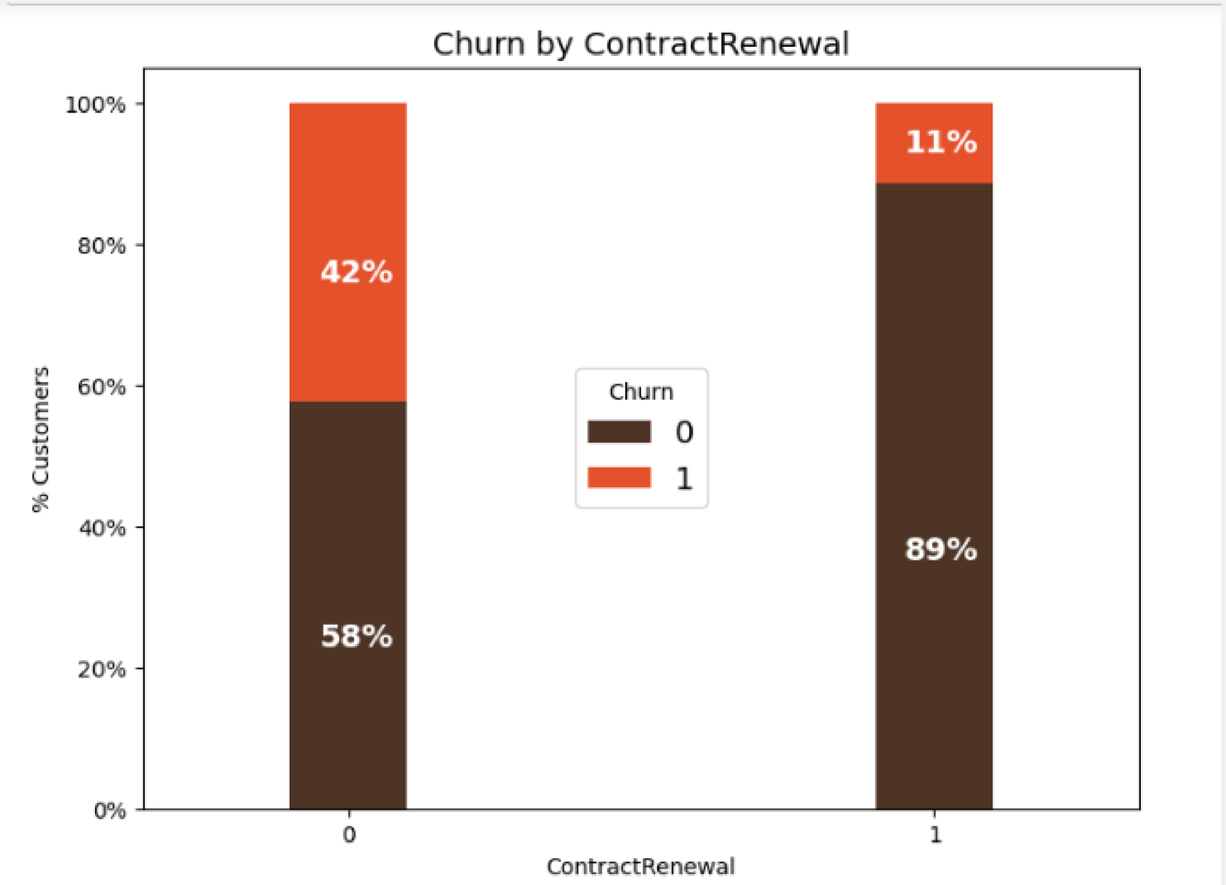


Analyzing customer tenure, represented by 'AccountWeeks', can offer insights into customer loyalty and satisfaction.

Recommendation based on customer tenure:

-Tailor services and offers based on customer tenure. For instance, long-term customers could be offered loyalty rewards, while new customers could benefit from introductory discounts or trial periods to enhance their experience.

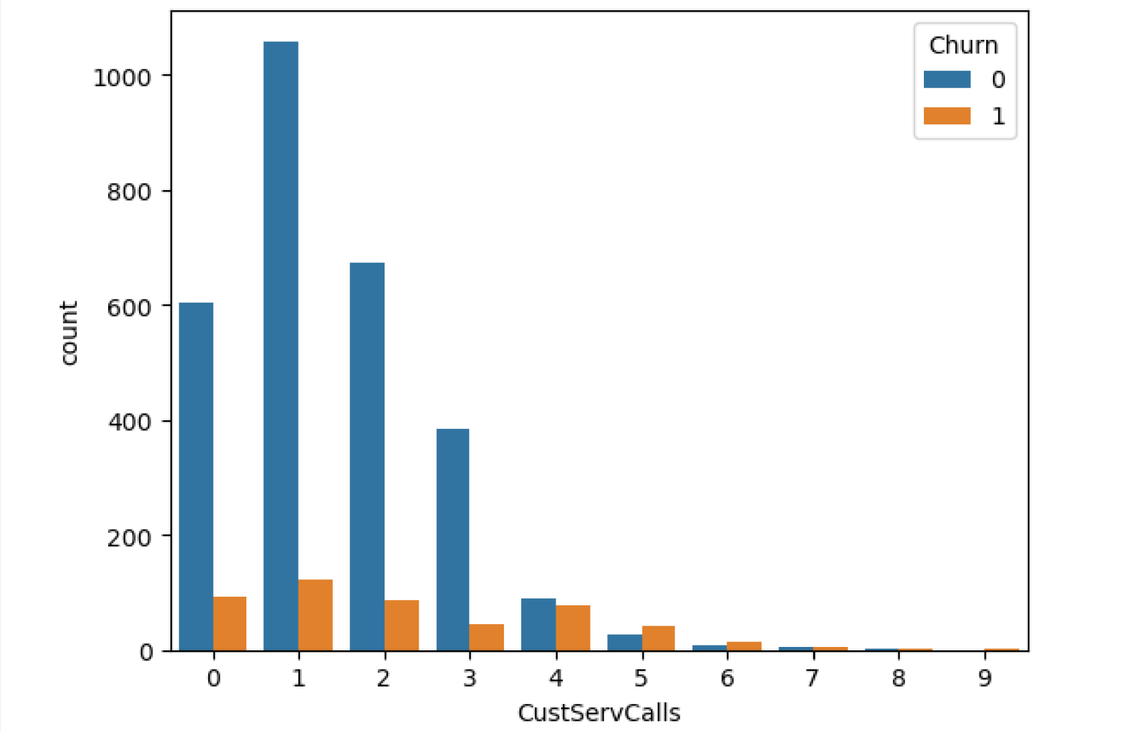
Contract Renewal Impact



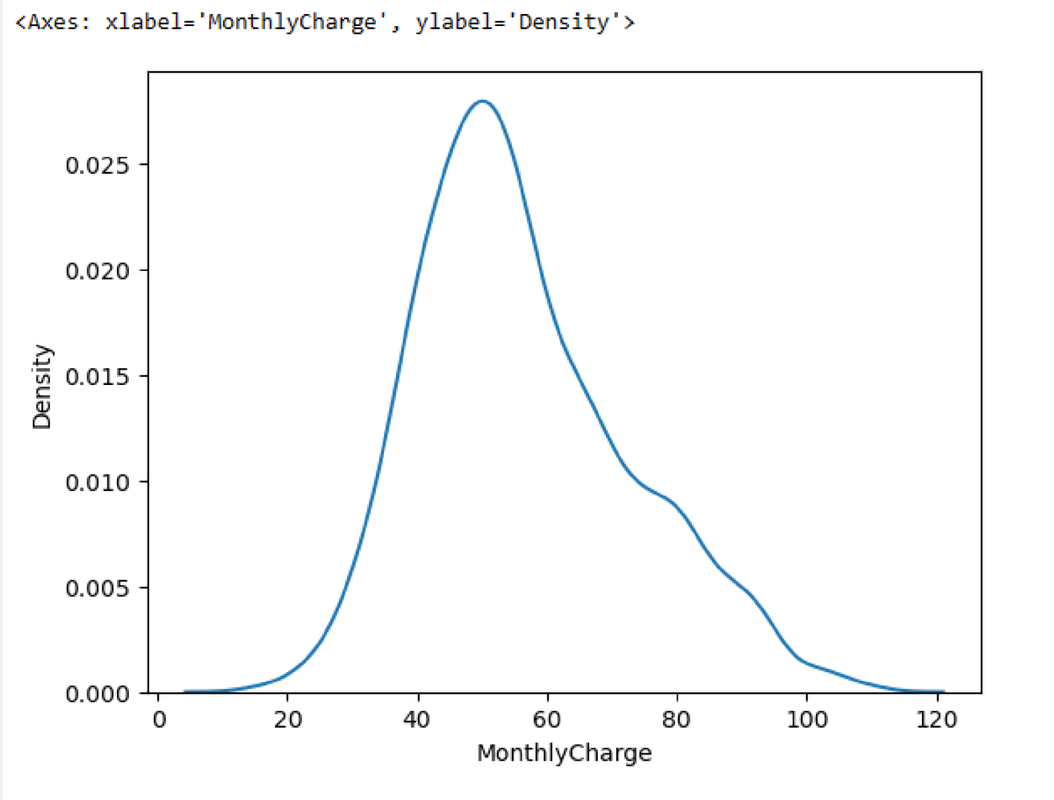
A higher churn rate is observed among customers who opt not to renew their contracts. This underscores the significance customers place on continuity and the assurance provided by contract renewal.

Recommendation to encourage contract renewal:

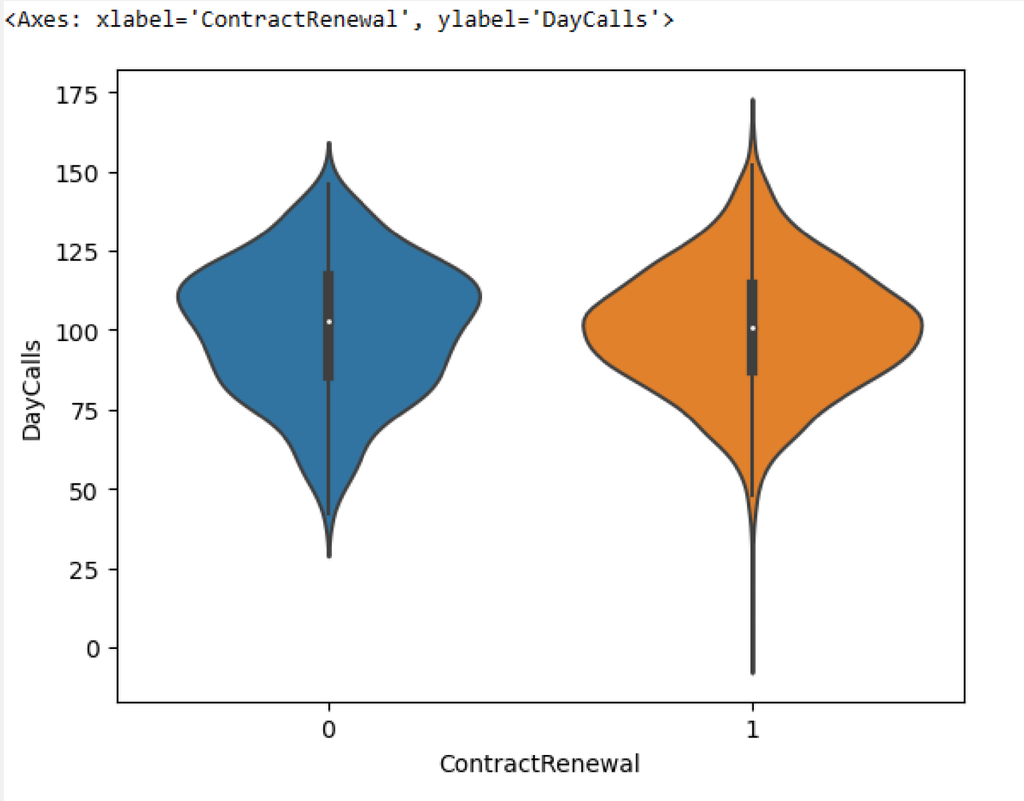
-Offer incentives such as discounts or additional services. Effective communication of the benefits of contract renewal to customers is also crucial.



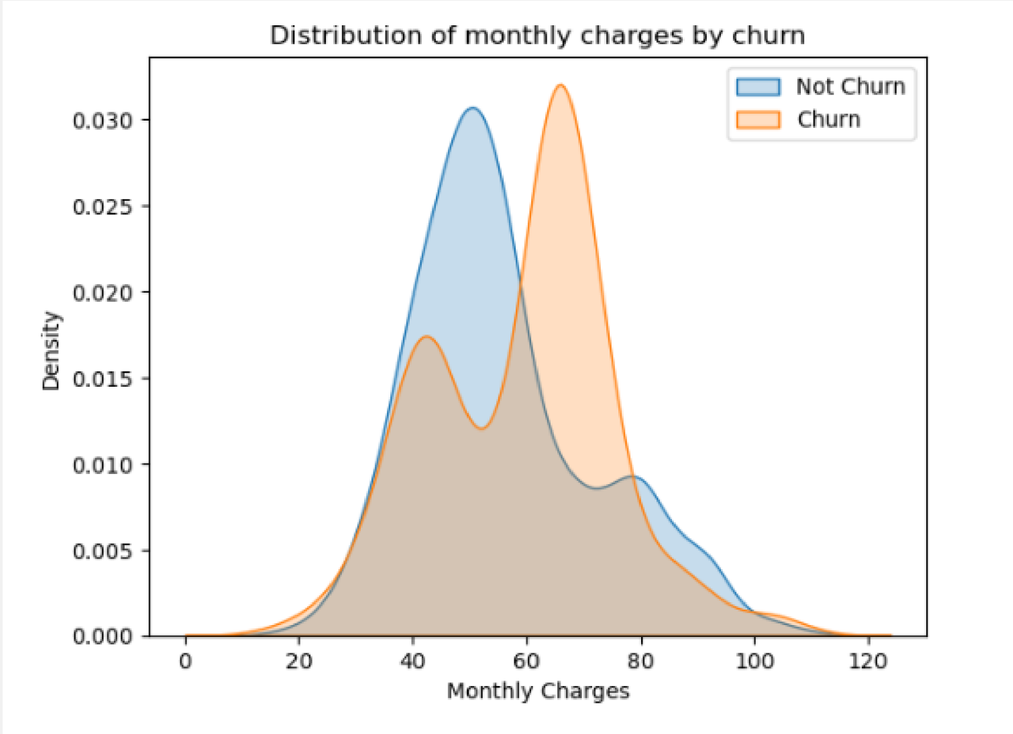
Countplot to understand relationship between churn rate and Customer service calls



Kde plot to understand the average amount of Monthly Recharge done by a customer



Violin to understand relationship between Contract Renewal and Average number of daytime calls



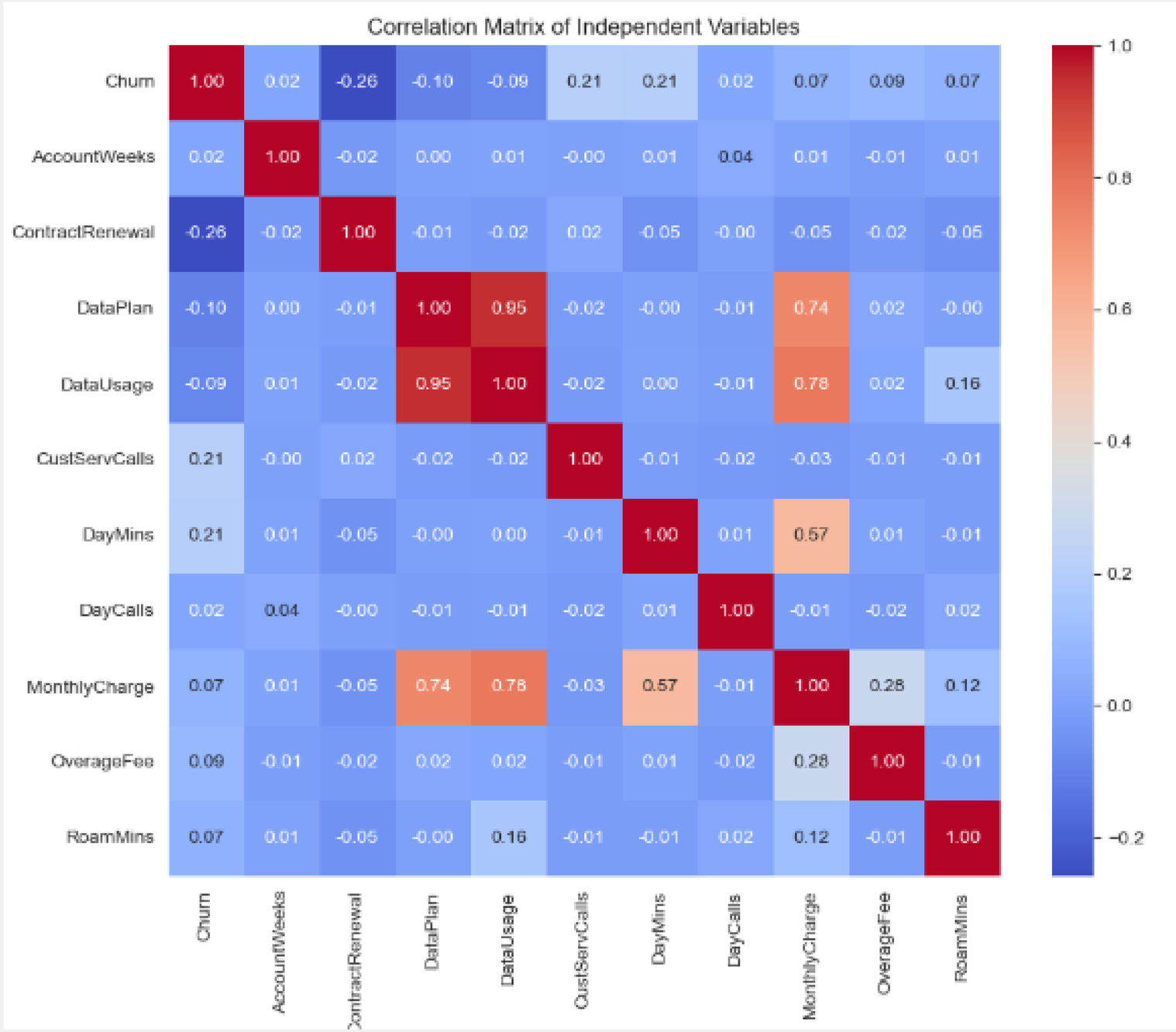
Distplot to understand the distribution of monthly charge by churn

Service Usage Patterns

Service Usage Patterns:
Examining the usage patterns of services such as 'DayCalls', 'MonthlyCharge', 'RoamMins', and 'OverageFee' can provide indications of customer satisfaction and their propensity to churn.

Recommendation based on service usage patterns:

- Adjust pricing plans or provide better value for these services to enhance customer satisfaction and reduce churn. For example, customers making frequent 'DayCalls' might benefit from a plan offering unlimited day calls.


Correlation Matrix of Independent Variables
Churn 1.00 0.02 -0.26 -0.10 -0.09 0.21 0.21 0.02 0.07 0.09 0.07
AccountWeeks 0.02 1.00 -0.02 0.00 0.01 -0.00 0.01 0.04 0.01 -0.01 0.01
ContractRenewal -0.26 -0.02 1.00 -0.01 -0.02 0.02 -0.05 -0.00 -0.05 -0.02 -0.05
DataPlan -0.10 0.00 -0.01 1.00 0.95 -0.02 -0.00 -0.01 0.74 0.02 -0.00
DataUsage -0.09 0.01 -0.02 0.95 1.00 -0.02 0.00 -0.01 0.78 0.02 0.16
CustServCalls 0.21 -0.00 0.02 -0.02 -0.02 1.00 -0.01 -0.02 -0.03 -0.01 -0.01
DayMins 0.21 0.01 -0.05 -0.00 0.00 -0.01 1.00 0.01 0.57 0.01 -0.01
DayCalls 0.02 0.04 -0.00 -0.01 -0.01 -0.02 0.01 1.00 -0.01 -0.02 0.02
MonthlyCharge 0.07 0.01 -0.05 0.74 0.78 -0.03 0.57 -0.01 1.00 0.28 0.12
OverageFee 0.09 -0.01 -0.02 0.02 0.02 -0.01 0.01 -0.02 0.28 1.00 -0.01
RoamMins 0.07 0.01 -0.05 -0.00 0.16 -0.01 -0.01 0.02 0.12 -0.01 1.00
Churn
AccountWeeks
ContractRenewal
DataPlan
DataUsage
CustServCalls
DayMins
DayCalls
MonthlyCharge
OverageFee
RoamMins
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Model Building

Generalised Linear Model

It's a flexible framework for modeling relationships between a response variable and one or more predictor variables. GLMs are an extension of linear regression that allow for non-normal error distributions and non-linear relationships between the predictors and the response..

There are three components to any GLM:

- Random Component - specifies the probability distribution of the response variable.
- Systematic Component - specifies the explanatory variables in the model, more specifically, their linear combination
- Link Function- specifies the link between the random and the systematic components. It indicates how the expected value of the response relates to the linear combination of explanatory variables

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	3333
Model:	GLM	**Df Residuals:**	3325
Model Family:	Gaussian	**Df Model:**	7
Link Function:	identity	**Scale:**	0.10337
Method:	IRLS	**Log-Likelihood:**	-943.37
Date:	Sun, 14 Apr 2024	**Deviance:**	343.72
Time:	12:52:24	**Pearson chi2:**	344.
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.1822
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0482	0.048	-0.997	0.319	-0.143	0.047
CustServCalls	0.0584	0.004	13.787	0.000	0.050	0.067
ContractRenewal	-0.2997	0.019	-15.871	0.000	-0.337	-0.263
MonthlyCharge	0.0062	0.001	11.201	0.000	0.005	0.007
DataPlan	-0.2478	0.019	-12.716	0.000	-0.286	-0.210
OverageFee	0.0022	0.002	0.935	0.350	-0.002	0.007
RoamMins	0.0030	0.002	1.479	0.139	-0.001	0.007
DayCalls	0.0004	0.000	1.283	0.200	-0.000	0.001

GLM Results

Logistic Regression Model

Logistic Regression is a supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event or observation

Logical regression analyzes the relationship between one or more independent variables and classifies data into discrete classes. It is extensively used in predictive modeling, where the model estimates the mathematical probability of whether an instance belongs to a specific category or not.

Key Advantages of Logistic Regression

- Easier to implement machine learning methods
- Suitable for linearly separable datasets
- Provides valuable insights

SMOTE Analysis - For Oversampling

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(x_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
Before OverSampling, counts of label '1': 382
Before OverSampling, counts of label '0': 2284

After OverSampling, the shape of train_X: (4568, 12)
After OverSampling, the shape of train_y: (4568,)

After OverSampling, counts of label '1': 2284
After OverSampling, counts of label '0': 2284
```

SMOTE is specifically designed to tackle imbalanced datasets by generating synthetic samples for the minority class. It helps to overcome the overfitting problem posed by random oversampling.

Near Miss Technique - For Undersampling

```
print("Before Undersampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before Undersampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# apply near miss
from imblearn.under_sampling import NearMiss
nr = NearMiss()

x_train_miss, y_train_miss = nr.fit_resample(x_train, y_train.ravel())

print('After Undersampling, the shape of train_X: {}'.format(x_train_miss.shape))
print('After Undersampling, the shape of train_y: {} \n'.format(y_train_miss.shape))

print("After Undersampling, counts of label '1': {}".format(sum(y_train_miss == 1)))
print("After Undersampling, counts of label '0': {}".format(sum(y_train_miss == 0)))
```

```
Before Undersampling, counts of label '1': 382
Before Undersampling, counts of label '0': 2284

After Undersampling, the shape of train_X: (764, 12)
After Undersampling, the shape of train_y: (764,)

After Undersampling, counts of label '1': 382
After Undersampling, counts of label '0': 382
```

Near-miss helps in balancing an imbalanced dataset. It can be grouped under under sampling algorithms and is an efficient way to balance the data. The algorithm does this by looking at the class distribution and randomly eliminating samples from the larger class

Lazy Classifier (Lazy Predict)

To deal with the problem of choosing a correct model that suits your problem statement, python has an amazing in-built library called ‘Lazy Predict’.
It provides support for a diverse range of supervised machine learning models, enabling an efficient comparison and choosing the optimal model for a specific task.

```
from lazypredict.Supervised import LazyClassifier
```

```
clf=LazyClassifier()
```

```
models,predictions=clf.fit(x_train_res,x_test,y_train_res,y_test)
```

```
 97%|▉| 28/29 [00:08<00:00,  4.08it/s]

[LightGBM] [Info] Number of positive: 2284, number of negative: 2284
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000731 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1851
[LightGBM] [Info] Number of data points in the train set: 4568, number of used features: 12
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000

100%|| 29/29 [00:08<00:00,  3.27it/s]
```

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score
NuSVC	0.87	0.86	0.86	0.88
AdaBoostClassifier	0.87	0.85	0.85	0.88
SVC	0.90	0.84	0.84	0.90
RandomForestClassifier	0.90	0.83	0.83	0.90
ExtraTreesClassifier	0.90	0.83	0.83	0.91
KNeighborsClassifier	0.85	0.82	0.82	0.86
LGBMClassifier	0.90	0.82	0.82	0.90
GaussianNB	0.85	0.81	0.81	0.86
XGBClassifier	0.89	0.80	0.80	0.89
BaggingClassifier	0.90	0.80	0.80	0.90
ExtraTreeClassifier	0.85	0.80	0.80	0.86
DecisionTreeClassifier	0.87	0.78	0.78	0.87
LogisticRegression	0.80	0.78	0.78	0.82
CalibratedClassifierCV	0.80	0.77	0.77	0.82
LinearSVC	0.80	0.77	0.77	0.82
RidgeClassifierCV	0.81	0.77	0.77	0.83
NearestCentroid	0.80	0.77	0.77	0.82
BernoulliNB	0.79	0.77	0.77	0.82
LinearDiscriminantAnalysis	0.80	0.77	0.77	0.82
RidgeClassifier	0.80	0.77	0.77	0.82
LabelSpreading	0.84	0.75	0.75	0.85
LabelPropagation	0.84	0.75	0.75	0.85
SGDClassifier	0.77	0.75	0.75	0.79
Perceptron	0.64	0.65	0.65	0.69
QuadraticDiscriminantAnalysis	0.85	0.50	0.50	0.78
DummyClassifier	0.85	0.50	0.50	0.78
PassiveAggressiveClassifier	0.36	0.49	0.49	0.41

Model	Time Taken
NuSVC	0.92
AdaBoostClassifier	0.46
SVC	0.64
RandomForestClassifier	1.04
ExtraTreesClassifier	0.46
KNeighborsClassifier	0.12
LGBMClassifier	0.48
GaussianNB	0.00
XGBClassifier	0.19
BaggingClassifier	0.40
ExtraTreeClassifier	0.03
DecisionTreeClassifier	0.05
LogisticRegression	0.06
CalibratedClassifierCV	0.89
LinearSVC	0.27
RidgeClassifierCV	0.02
NearestCentroid	0.02
BernoulliNB	0.05
LinearDiscriminantAnalysis	0.06
RidgeClassifier	0.06
LabelSpreading	1.44
LabelPropagation	1.05
SGDClassifier	0.03
Perceptron	0.02
QuadraticDiscriminantAnalysis	0.00
DummyClassifier	0.01
PassiveAggressiveClassifier	0.04

Model 1 - Random Forest Classifier

- Random forests are a popular supervised machine learning algorithm. It is used for solving regression (numeric target variable) and classification (categorical target variable) problems. It is an ensemble method, means they combine predictions from other models.
- Each of the smaller models in the random forest ensemble is a decision tree.

```
ACCURACY OF THE MODEL: 0.9055472263868066
```

```
predictions = clf.predict(x_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       566
           1       0.67      0.74      0.70       101

    accuracy                           0.91       667
   macro avg       0.81      0.84      0.82       667
weighted avg       0.91      0.91      0.91       667
```

Model 2 - XGB Model

- XGBoost is a robust machine-learning algorithm that can help you understand your data and make better decisions.It's a highly efficient and scalable implementation of the boosting algorithm, with performance comparable to that of other state-of-the-art machine learning algorithms in most cases.
- It takes in training data, uses it to train a model, and then evaluates the model on new data. This process repeats until the model stops improving.

```
Accuracy: 0.8935532233883059
```

```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       566
           1       0.64      0.67      0.66       101

    accuracy                           0.89       667
   macro avg       0.79      0.80      0.80       667
weighted avg       0.90      0.89      0.89       667
```

Model 3 - LGB Model

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

```
predictions = model.predict(x_test)
print(classification_report(y_test, predictions))
```

```
[LightGBM] [Warning] min_data_in_leaf is set=20, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=20
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=3
1).
              precision    recall  f1-score   support

           0       0.94      0.93      0.93       566
           1       0.63      0.68      0.65       101

    accuracy                           0.89       667
   macro avg       0.78      0.81      0.79       667
weighted avg       0.89      0.89      0.89       667
```

Selection and Deployment of Model

On Application of Lazy Classifier, 3 models with great accuracy were tested for the purpose of deployment.

Random Forest Method was selected as it showed great levels of accuracy and recall value

Final Model deployment was done with the help of Flask framework as it helps in providing a better environment

THANK YOU
VERY MUCH