Major Project Report

On

CUSTOMER CHURN PREDICTION

Training and Internship Program at SKILLVERTEX
April batch (05 April 2022 – 05 June 2022)

MACHINE LEARNING WITH PYTHON

Submitted by: Group 1
Kriti Bhardwaj
Prakshita Agrawal
Anjali
Anil Kumar Barik

1. INTRODUCTION

Churn prediction is most common use case in machine learning domain. Churn means "leaving the company", It is very critical for business to have an idea about why and when customers are likely to churn. Having a robust and accurate churn prediction model helps businesses to take actions to prevent customers from leaving the company.

The main objective of our project is to:

- Analyse the data and identify the main indicators of churn.
- Build a predictive model to predict customer churn.
- Use different classification models for the given dataset and find the best model based on the model performances i.e. accuracy and f1_score.
- Model optimisation using GridSearchCV method.
- Predict “churn” for given test dataset

2. DATA DESCRIPTION

Each row and column of the Customer Churn Dataset represents a customer and customer’s attributes respectively. The following describes the dataset columns:

- state: string
- account_length: integer
- area_code: integer
- international_plan: string
- voice_mail_plan: string
- number_vmail_messages: integer
- total_day_minutes: float
- total_day_calls: integer
- total_day_charge: float
- total_eve_minutes: float
- total_eve_calls: integer
- total_eve_charge: float
- total_night_minutes: float
- total_night_calls: integer
- total_night_charge: float
- total_intl_minutes: float
- total_intl_calls: integer
- total_intl_charge: float
- customer_service_calls: integer
- churn: string

The "Customer churn" dataset contains 4250 rows (customers) and 20 columns (features). The "churn" column is the target to predict.

3. APPROACH

Our approach to the solution of this project is as follows:

- Importing the necessary libraries and the dataset
- Performing Exploratory Data Analysis, Visualization and Pre-processing of the training and testing dataset
- Modelling using Random Forest Classifier, Decision Tree Classifier, KNN Classifier, Logistic Regression, Naive Bayes and Support Vector Machine Algorithms
- The model with the best accuracy and f1_score closest to 1.0 will be the best model for churn prediction
- After the model is finalised, it is optimised using GridSearchCV model optimisation method
- "churn" value is predicted for the given test dataset with the optimised model

4. VISUALISATION, EDA AND PRE-PROCESSING

A detailed analysis and pre-processing are done in the dataset. It gave us a better idea of contribution of features towards the target variable. This is an overview of the given original training data, with its original features:

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_char
0	OH	107	area_code_415	no	yes	26	161.6	123	27.
1	NJ	137	area_code_415	no	no	0	243.4	114	41.
2	OH	84	area_code_408	yes	no	0	299.4	71	50.
3	OK	75	area_code_415	yes	no	0	166.7	113	28.
4	MA	121	area_code_510	no	yes	24	218.2	88	37.
...	...	...	...	...	...	...	...	...	
4245	MT	83	area_code_415	no	no	0	188.3	70	32.
4246	WV	73	area_code_408	no	no	0	177.9	89	30.
4247	NC	75	area_code_408	no	no	0	170.7	101	29.
4248	HI	50	area_code_408	no	yes	40	235.7	127	40.
4249	VT	86	area_code_415	no	yes	34	129.4	102	22.

4250 rows × 20 columns

- **Checking null values:**

As we can see from the above plot that we don't have any missing data in our dataset. Although if missing values are present in a dataset, then it can be managed using the following ways:

(a) Fill the missing value with the mean of the values present of the feature if it stores numerical data.
(b) Fill the missing value with the mode of the values present of the feature if it stores categorical data.
(c) If there are so many missing data in a row then it would be better to drop the row.
(d) For features with high number of missing values, it will be better to drop the column.

- **Statistical Data:**

```
df.describe()
```

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	total_eve_charge	total_
count	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	
mean	100.236235	7.631765	180.259600	99.907294	30.644682	200.173906	100.176471	17.015012	
std	39.698401	13.439882	54.012373	19.850817	9.182096	50.249518	19.908591	4.271212	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	73.000000	0.000000	143.325000	87.000000	24.365000	165.925000	87.000000	14.102500	
50%	100.000000	0.000000	180.450000	100.000000	30.680000	200.700000	100.000000	17.060000	
75%	127.000000	16.000000	216.200000	113.000000	36.750000	233.775000	114.000000	19.867500	
max	243.000000	52.000000	351.500000	165.000000	59.760000	359.300000	170.000000	30.540000	

- **Analysing target variable:**

```
df['churn'].value_counts()/len(df['churn'])*100
```

```
no     85.929412
yes    14.070588
Name: churn, dtype: float64
```

  - Data is **imbalanced** since the ratio is 86:14

- **Univariate Analysis:**

The first step in data exploration is usually a univariate analysis. It explores each variable(attribute) in a dataset separately. Variables could be either categorical or numerical.

(a) <u>Univariate Analysis for categorical data:</u>

```
df[['area_code','churn']].groupby(['area_code']).mean()
```

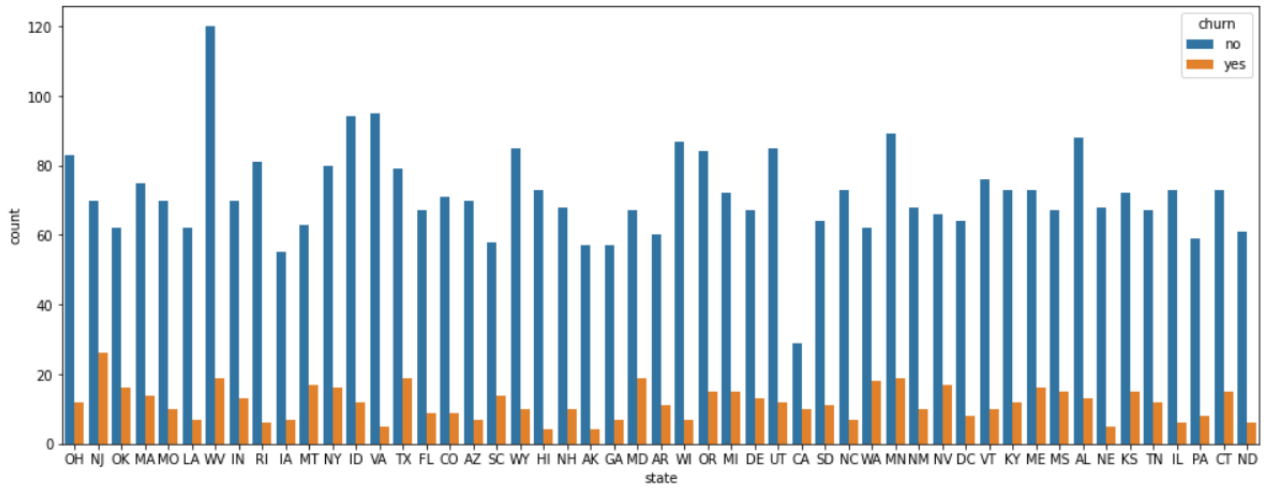
area_code	churn
area_code_408	0.139963
area_code_415	0.136148
area_code_510	0.150568

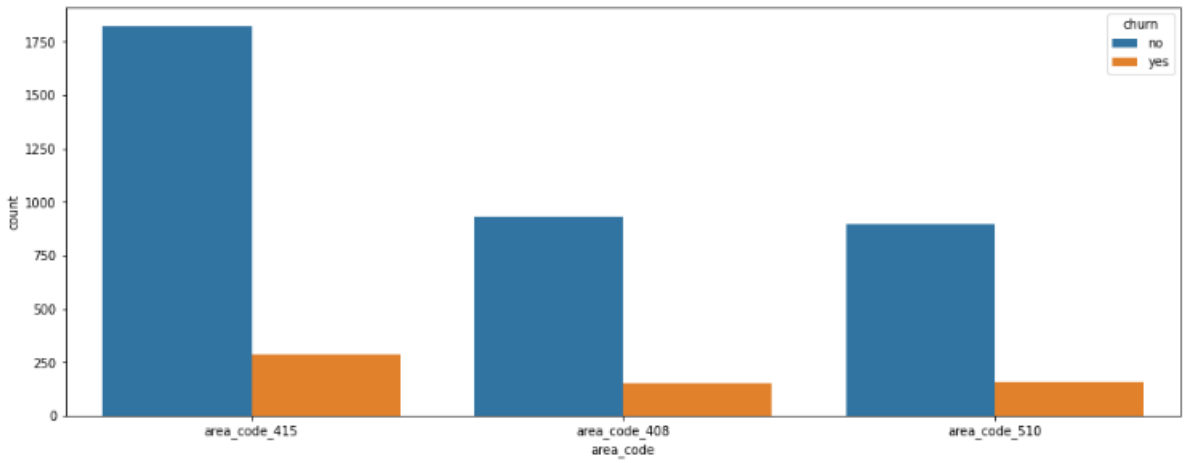
```
df[['international_plan','churn']].groupby(['international_plan']).mean()
```

international_plan	churn
no	0.111832
yes	0.421717

```
df[['voice_mail_plan','churn']].groupby(['voice_mail_plan']).mean()
```

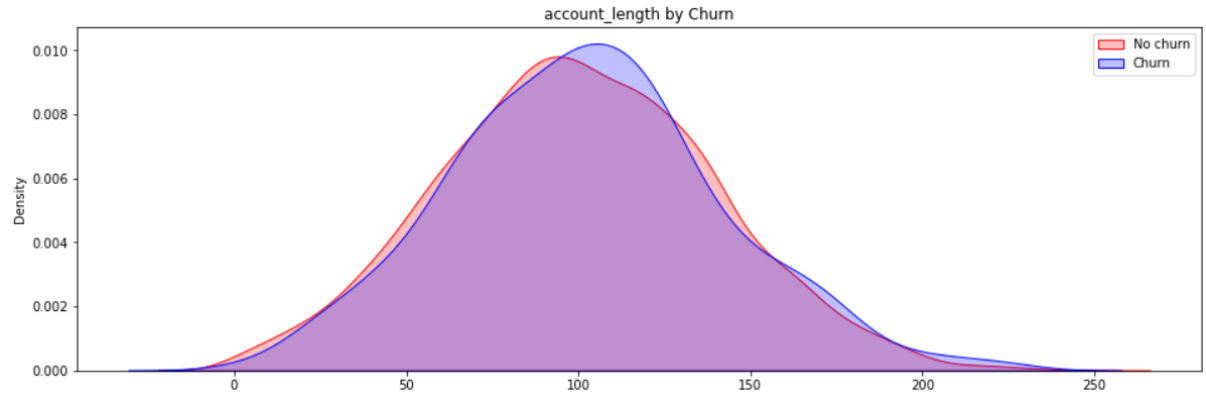
voice_mail_plan	churn
no	0.164436
yes	0.073741

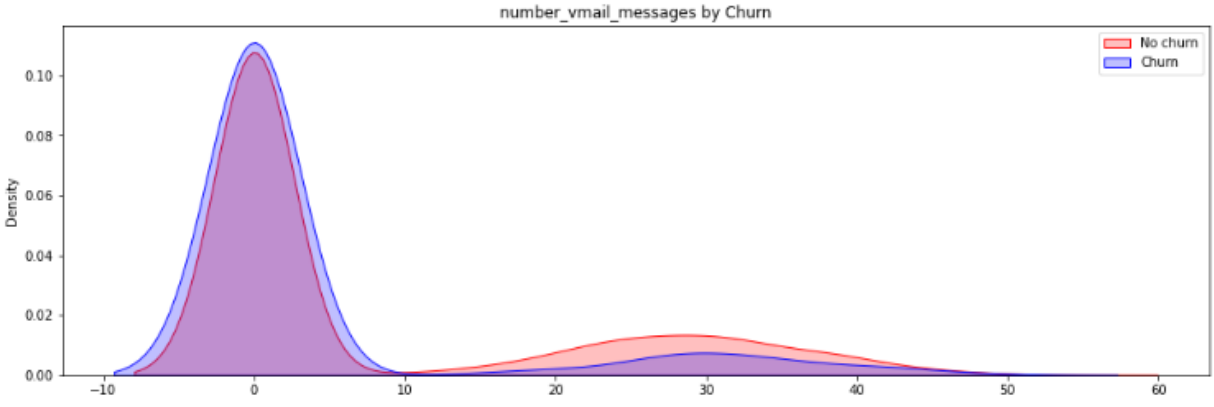


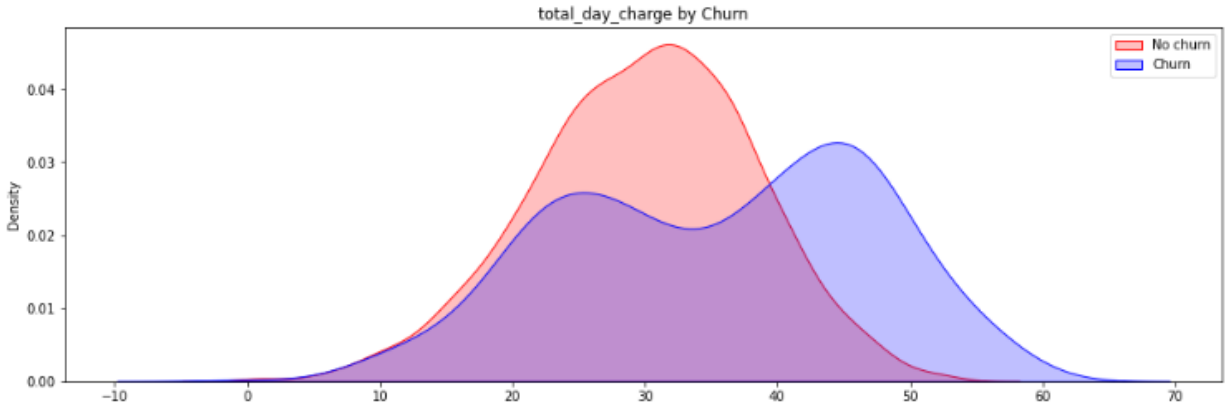


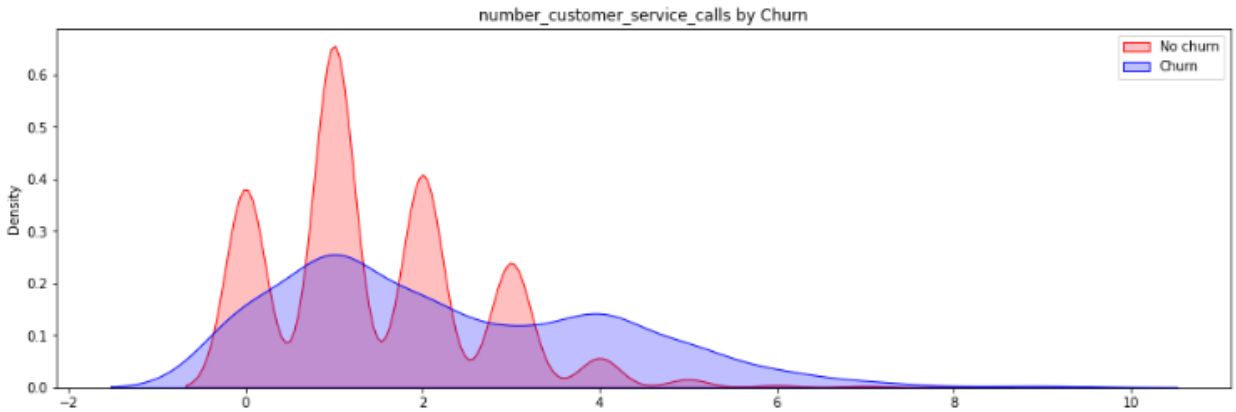
Similarly, other categorical columns are analysed using plot visualisation.

(b) Univariate Analysis for numerical data:





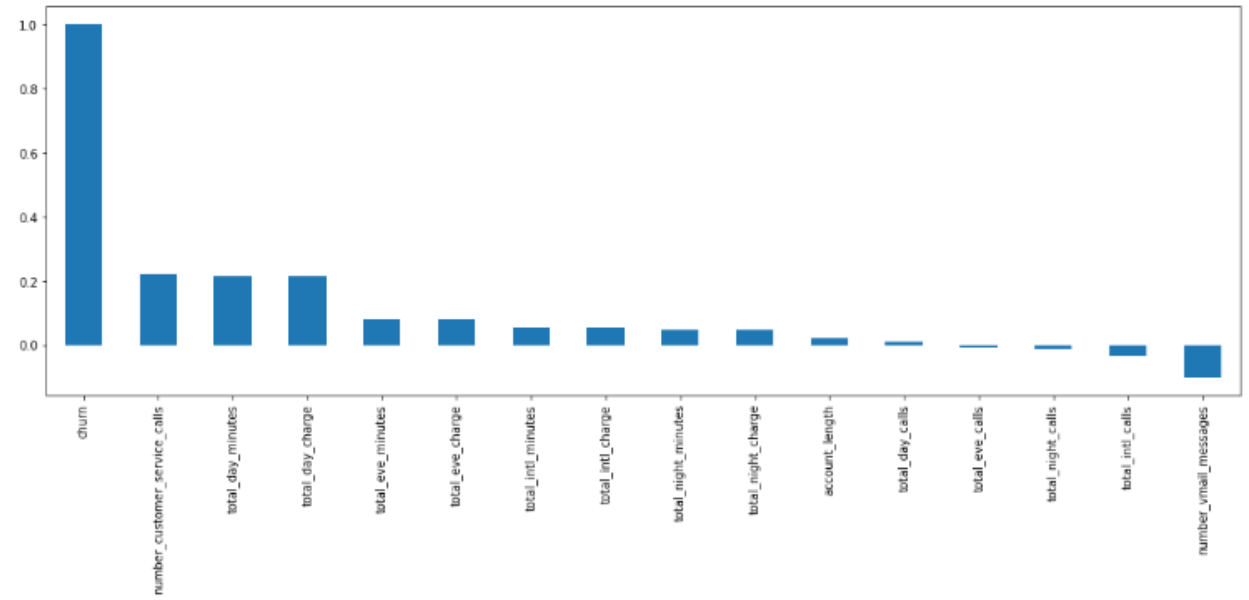




Similarly, other numerical columns are analysed using plot visualisation.

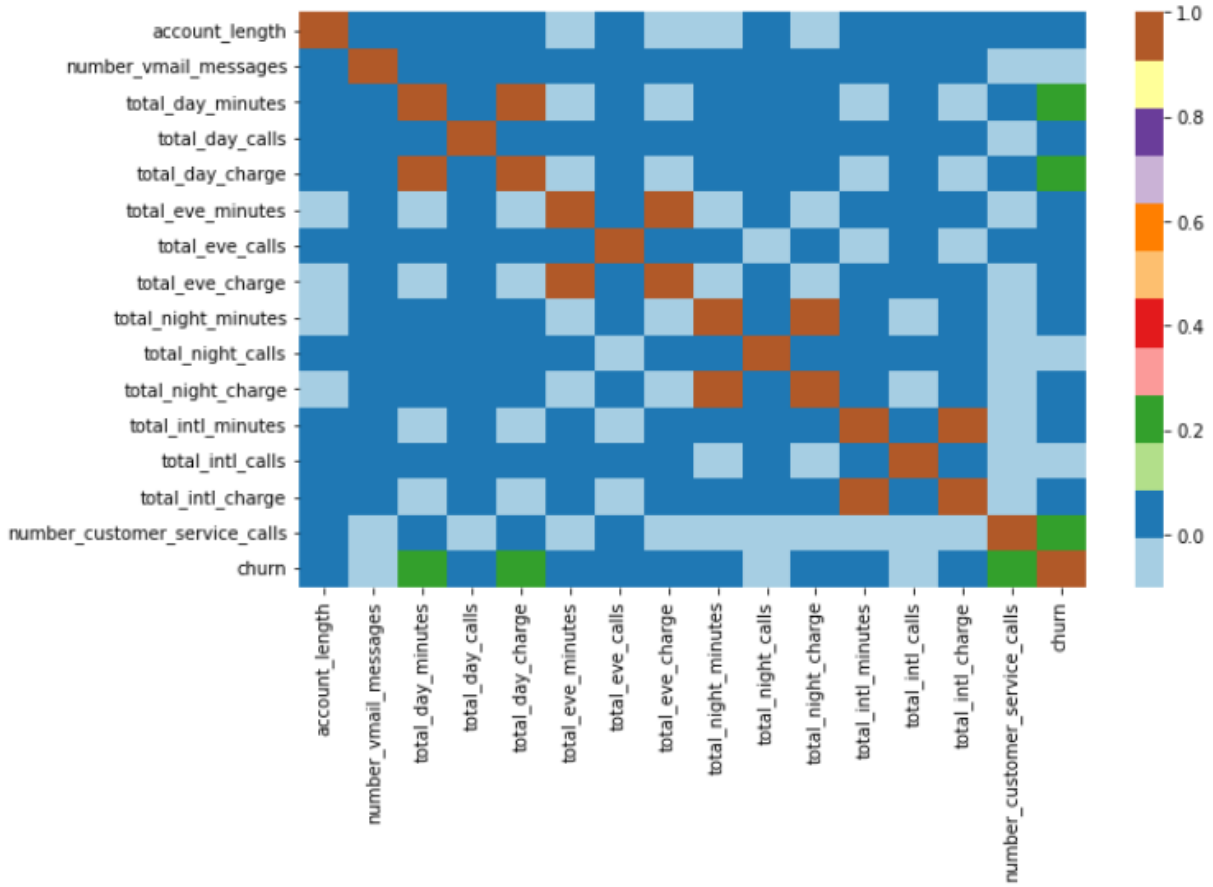
- **Correlation and Heatmap of all variables:**

```
plt.figure(figsize=(18,6))
df.corr()['churn'].sort_values(ascending = False).plot(kind='bar')
plt.show()
```



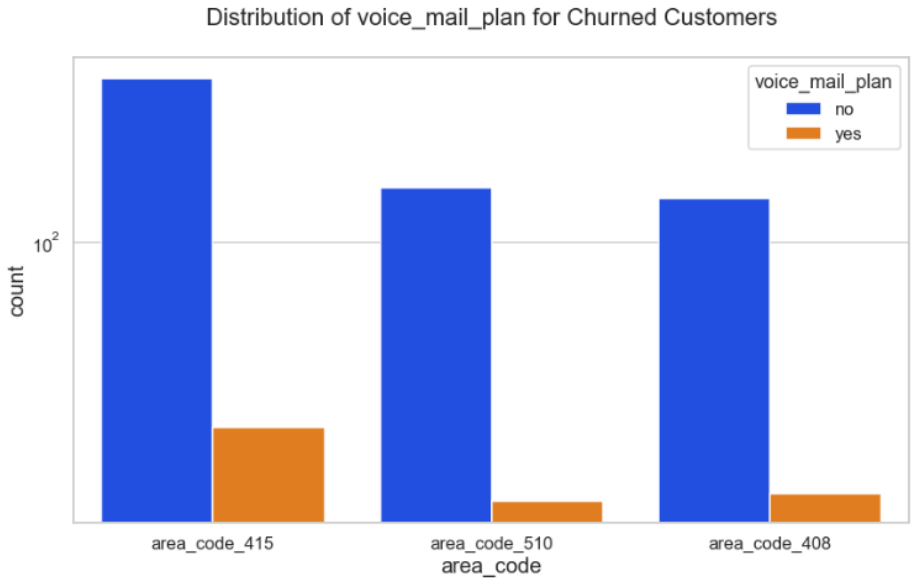

```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), cmap="Paired")
```

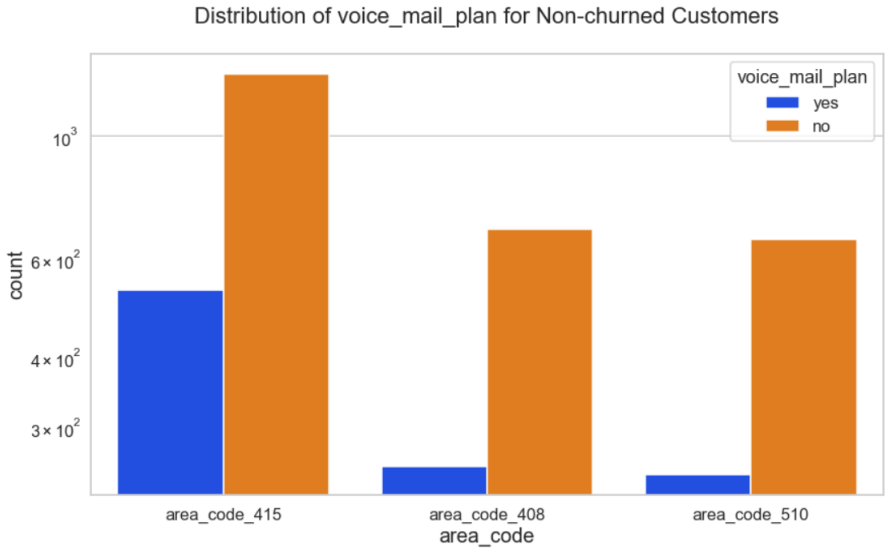
```
<AxesSubplot:>
```

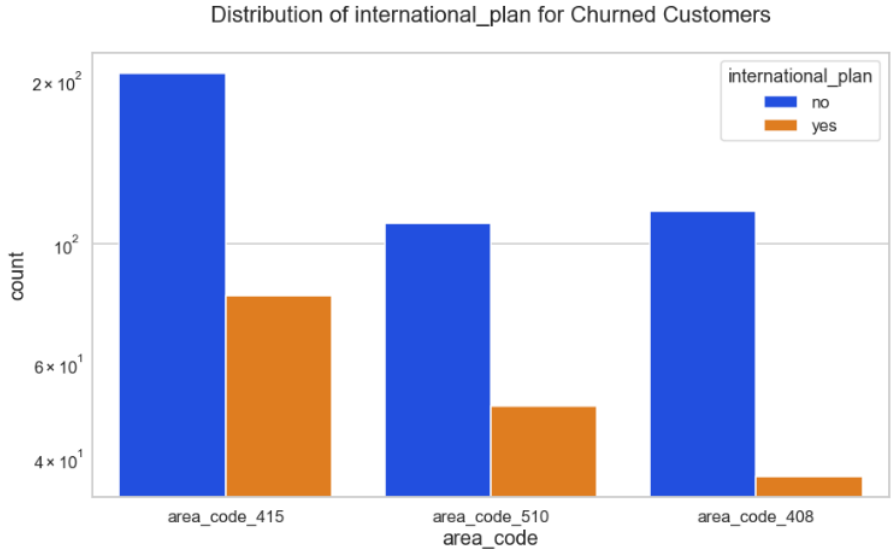


- **Bivariate Analysis:**

Analysing two variables at a time is termed as bivariate analysis. We have analysed the distribution of categorical columns for churned customers.







- **Observations from the data analysis:**

  - ✓ Combining all the observations from the above plots, **total_day_charge, total_day_minutes and number_customer_service_calls** are contributing majorly to the target variable.
  - ✓ **HIGH churning** is observed with **international_plan, number_customer_service_calls, total_day_minutes** and **total_day_charge** since they are positively well correlated with the target variable.
  - ✓ **LOW churning** is observed with **no international_plan, voice_mail_plan** and **number_vmail_messages** since they are negatively well correlated with the target variable.
  - ✓ It can be seen that features like **area_code, total_day_calls, total_night_calls** and **total_eve_calls** aren't contributing much to the target variable; hence it can be concluded that they have almost **NO impact** on the target variable, churn.

- **Encoding of categorical columns:**

We have used One Hot Encoding to produce binary integers of 0 and 1 to encode our categorical features 'churn' because categorical features that are in string format cannot be understood by the machine and needs to be converted to numerical format. Also, we have created dummy variables for all categorical variables and dropped the "state" column as it is not contributing much to our target variable "churn".

5. ALGORITHMS

Since Churn Prediction is a Classification problem so we have used different classification algorithms on this dataset and they all have different accuracy and other performance measures. Feature Selection is the process where you automatically or manually select those features which contribute the best to the target variable.

Model performances can be evaluated by various metrics such as:

(a) Confusion Matrix: The confusion matrix is a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label that the model predicted, and the other axis is the actual label.

(b) Precision: Precision is the ratio of correct positive predictions to the overall number of positive predictions: TP/TP+FP

(c) Recall: Recall is the ratio of correct positive predictions to the overall number of positive examples in the set: TP/FN+TP

(d) Accuracy: Accuracy is given by the number of correctly classified examples divided by the total number of classified examples. In terms of the confusion matrix, it is given by: TP+TN/TP+TN+FP+FN

(e) f1_score: The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

Combining all performance measures together, all the classification models have their classification reports. These are the following machine learning algorithms on our dataset:

5.1 RANDOM FOREST CLASSIFIER

It is based on the concept of **ensemble learning,** which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. Instead of relying on one decision tree, the random forest takes the prediction from multiple trees and based on the majority votes of predictions, and it predicts the final output.

- **Training and testing accuracy**

```
rf.score(X_train, y_train)*100 # training accuracy
```

```
99.97058823529412
```

```
rf.score(X_test, y_test)*100 # testing accuracy
```

```
96.47058823529412
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[719,   2],
       [ 28, 101]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
96.47058823529412
```

```
precision_score(y_test, y_pred)
```

```
0.9805825242718447
```

```
recall_score(y_test, y_pred)
```

```
0.7829457364341085
```

```
f1_score(y_test, y_pred)
```

```
0.8706896551724137
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       721
           1       0.98      0.78      0.87       129

    accuracy                           0.96       850
   macro avg       0.97      0.89      0.93       850
weighted avg       0.97      0.96      0.96       850
```

5.2 DECISION TREE CLASSIFIER

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier with two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

- **Training and testing accuracy**

```
dt.score(X_train, y_train)*100 # training accuracy
```

```
100.0
```

```
dt.score(X_test, y_test)*100 # testing accuracy
```

```
91.05882352941177
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[680,  41],
       [ 35,  94]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
91.05882352941177
```

```
precision_score(y_test, y_pred)
```

```
0.6962962962962963
```

```
recall_score(y_test, y_pred)
```

```
0.7286821705426356
```

```
f1_score(y_test, y_pred)
```

```
0.7121212121212122
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       721
           1       0.70      0.73      0.71       129

    accuracy                           0.91       850
   macro avg       0.82      0.84      0.83       850
weighted avg       0.91      0.91      0.91       850
```

5.3 LOGISTIC REGRESSION

Logistic regression is a classification algorithm which predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

- **Training and testing accuracy**

```
log_reg.score(X_train, y_train)*100 # training accuracy
```

```
86.88235294117646
```

```
log_reg.score(X_test, y_test)*100 # testing accuracy
```

```
85.88235294117646
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[703,  18],
       [102,  27]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
85.88235294117646
```

```
precision_score(y_test, y_pred)
```

```
0.6
```

```
recall_score(y_test, y_pred)
```

```
0.20930232558139536
```

```
f1_score(y_test, y_pred)
```

```
0.3103448275862069
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       721
           1       0.60      0.21      0.31       129

    accuracy                           0.86       850
   macro avg       0.74      0.59      0.62       850
weighted avg       0.83      0.86      0.83       850
```

5.4 K-NN CLASSIFIER

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

- **Training and testing accuracy**

```
knn.score(X_train, y_train)*100 # training accuracy
```

```
88.79411764705883
```

```
knn.score(X_test, y_test)*100 # testing accuracy
```

```
87.52941176470588
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[718,   3],
       [103,  26]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
87.52941176470588
```

```
precision_score(y_test, y_pred)
```

```
0.896551724137931
```

```
recall_score(y_test, y_pred)
```

```
0.20155038759689922
```

```
f1_score(y_test, y_pred)
```

```
0.32911392405063294
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       721
           1       0.90      0.20      0.33       129

    accuracy                           0.88       850
   macro avg       0.89      0.60      0.63       850
weighted avg       0.88      0.88      0.84       850
```

5.5 NAIVE BAYES

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

- **Training and testing accuracy**

```
nb.score(X_train, y_train)*100 # training accuracy
```

```
86.8529411764706
```

```
nb.score(X_test, y_test)*100 # testing accuracy
```

```
86.58823529411764
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[668,  53],
       [ 61,  68]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
86.58823529411764
```

```
precision_score(y_test, y_pred)
```

```
0.5619834710743802
```

```
recall_score(y_test, y_pred)
```

```
0.5271317829457365
```

```
f1_score(y_test, y_pred)
```

```
0.5440000000000002
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       721
           1       0.56      0.53      0.54       129

    accuracy                           0.87       850
   macro avg       0.74      0.73      0.73       850
weighted avg       0.86      0.87      0.86       850
```

5.6 SUPPORT VECTOR MACHINE

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

- **Training and testing accuracy**

```
sv.score(X_train, y_train)*100 # training accuracy
```

```
86.32352941176471
```

```
sv.score(X_test, y_test)*100 # testing accuracy
```

```
84.94117647058823
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[721,   0],
       [128,   1]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
84.94117647058823
```

```
precision_score(y_test, y_pred)
```

```
1.0
```

```
recall_score(y_test, y_pred)
```

```
0.007751937984496124
```

```
f1_score(y_test, y_pred)
```

```
0.015384615384615384
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       721
           1       1.00      0.01      0.02       129

    accuracy                           0.85       850
   macro avg       0.92      0.50      0.47       850
weighted avg       0.87      0.85      0.78       850
```

FINAL MODEL:

- ✓ After going through all the classification models, based on their performance measures, we conclude that **Random Forest Classifier** is the best model for our dataset.
- ✓ Although the high accuracy of the Random Forest Classifier can also be because of the model overfitting, hence our model also needs to be optimised.

6. MODEL OPTIMISATION USING GRIDSEARCHCV:

It is necessary to tune the hyperparameters of respective algorithms to improve accuracy and avoid overfitting when using tree-based models such as Random Forest Classifier. Hyperparameters are sets of information that are used to control the way of learning an algorithm.

Following methods can be used to hyperparameter tuning:

(a) RandomizedSearchCV → Fast
(b) GridSearchCV → More Accurate

Focusing on the accuracy, we have optimised our model using GridSearchCV model optimisation method.

- **Assigning hyperparameters and model fitting**

```
n_estimators = [20,60,100,120]
max_features = [0.2,0.6,1.0]
max_depth = [2,8,None]
max_samples = [0.5,0.75,1.0]
```

```
param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'max_samples': max_samples
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_de
pth': [2, 8, None], 'max_samples': [0.5, 0.75, 1.0]}
```

```
rfcf = RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV
```

```
rf_grid = GridSearchCV(estimator = rfcf,
                       param_grid = param_grid,
                       cv = 5,
                       verbose = 2,
                       n_jobs = -1)
```

```
rf_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

- **Best parameters and best score**

```
rf_grid.best_params_
```

```
{'max_depth': 8, 'max_features': 0.6, 'max_samples': 0.75, 'n_estimators': 100}
```

```
rf_grid.best_score_
```

```
0.9555882352941175
```

- **Performance measures**

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[712,   9],
       [ 28, 101]], dtype=int64)
```

```
accuracy_score(y_pred, y_test)*100
```

```
95.6470588235294
```

```
precision_score(y_test, y_pred)
```

```
0.9181818181818182
```

```
recall_score(y_test, y_pred)
```

```
0.7829457364341085
```

```
f1_score(y_test, y_pred)
```

```
0.8451882845188284
```

- **Classification Report**

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       721
           1       0.92      0.78      0.85       129

    accuracy                           0.96       850
   macro avg       0.94      0.89      0.91       850
weighted avg       0.96      0.96      0.96       850
```

7. SAVING MODEL (PICKLING):

```
import pickle
```

```
filename = 'rf_model.pkl'
pickle.dump(rf_grid, open(filename, 'wb'))
```

Our final model i.e. **Random Forest Classifier** after being optimised with **GridSearchCV**, is now ready and dumped in **rf_model.pkl**, which can be used to prepare **API's** so that our model can be accessed from **UI** also.

8. CONCLUSION:

Starting with the dataset loading, we have performed EDA, pre-processing, encoding of categorical columns, univariate and bivariate analysis, trained and tested different classification models, finalised the best model as per the accuracies among all the models and then optimised the model.

- We got the best accuracy and f1_score with the **Random Forest Classifier** and hence it is the **best model** for the churning dataset.
- To avoid overfitting of the model, it is optimised using the **GridSearchCV** model optimisation method which gave model best_score_ as **0.96**.
- After model optimisation, our model has the final **accuracy** of **95.65%** and **f1_score** of **0.85**
- Loaded the given test dataset and predicted the final churn value for the test dataset with the optimised model.

9. FUTURE WORK

Customer churn is a major problem and one of the most important concerns for large companies. Due to the direct effect on the revenues of the companies, especially in the telecom field, companies are seeking to develop means to predict potential customer to churn. Therefore, finding factors that increase customer churn is important to take necessary actions to reduce this churn. Customers' churn is a considerable concern in service sectors with highly competitive services. On the other hand, predicting the customers who are likely to leave the company will represent potentially large additional revenue source if it is done in the early phase.

Our model is trained and tested with all the features that can affect customers' churn. Also, it is well optimised to give accurate predictions. A model like this would be very valuable and helpful for a company to predict its customers' churn and based on the reasons of churning, it can then make an attempt so that its customer won't churn and continue with the company happily.