AM41UD Coursework

Sowmya Perumalla
ID: 210295656

August 8, 2022

1 Abstract

Churn is a problem that affects practically every sector. However, given that many people think the telecoms sector has peaked, it is particularly pertinent to that industry. In order to reduce customer churn rates and effectively compete in the current environment of intense competition, many telcos are realizing the need of improving customer experience and service. Understanding the causes of churn in the telecom industry is crucial for doing this.

Lu's Communications has a huge problem with churn (loss of customers to competition). It is expensive to acquire new customers and therefore retaining existing customers is much more appealing.

To address this issue, we used classification Machine Learning Algorithms such as logistic regression, Naive Bayes, K-Nearest Neighbor, Random Forest, Gradient boosting classification and SVM on Lu's Communications dataset.

After executing all the above-mentioned algorithms Random Forest Classifier is showing promising results in terms of overall accuracy, precision,Recall, and F1 Score.

Contents

2 Introduction

Customer churn is a real and challenging problem across all service industries, and can be expensive too. Some customer churn is inevitable, although it is possible to predict customers likely to churn in advance and mitigate the problem with incentives. There could be many reasons for customer churn, Lu's communication has a huge churn ratio is due to losing customers to competitors which needs to be addressed at correct time. The core objective is to find a targeted approach to identify in advance customers who are likely to churn accurately, such that loss incurred due to invalid prediction are reduced.

As part of this report, various classifier models are tried for high accuracy and recall score by using evaluation metrics to support robustness of the best fit model identified.

2.1 Data Collection:

Lu's communication has provided the data in CSV format. However, this data is not perfect: it may contain outliers, missing values and irrelevant (not related to churn) information.Moving forward need to deal and analyse the provided data.

2.2 Data Preparation:

Given dataset contains in total 7350 records and 11 features.After reading the dataset we came to know that there are 2 numerical features those are monthlycost and Tenure ,where as Customer id, gender, location, partner, dependents, senior, package, survey and Class are Categorical values as most of them are binary classified and location contains string.

Here we considered "Class" feature as a target or dependent feature as our main objective is to find the root cause of the churn, So "class" is binary class feature which tells about whether customer is churned or not.
Rest of the columns (gender,,customerid,location,partner,dependents,senior,package,survey) are considered as independent features.

Summary model's Performance			
Feature	Description	Type	Value
Customer id	Unique Id	Categorical	Unique numbers
Gender	Male or Female	Categorical	Male or female
Partner	0=no patner and 1=patner	Categorical	0 or1
Dependents	0=no dependent and 1=dependent	Categorical	0,1 and Unknown
senior	0=not senior and 1=senior	Categorical	0 or 1
Tenure senior	It tells about tenure of customer with company in months 0=not senior and 1=senior	Numerical Categorical	Min=-4.6 and Max=30 0 or 1
Monthly cost	package cost	Numerical	Min=26 Max=47
Package	Packge type	Categorical	1,2,3,4
Survey	rating given by customers	Categorical	1,2,3,4,5,6,7,8,9,10
Location	Customers location	Categorical	Names of states
Class	It tell about customer churned or not	Categorical	churned is 1 not churned is 0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7350 entries, 0 to 7349
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   customer_id   7350 non-null   object 
 1   gender        7350 non-null   object 
 2   location      7350 non-null   object 
 3   partner       7350 non-null   int64  
 4   dependents    7350 non-null   object 
 5   senior        7350 non-null   int64  
 6   Tenure        7350 non-null   float64
 7   monthly_cost  67 non-null     object 
 8   package       7350 non-null   int64  
 9   survey        7350 non-null   object 
 10  Class         7301 non-null   object 
dtypes: float64(1), int64(3), object(7)
memory usage: 631.8+ KB
None
```

```
customer_id        0
gender             0
location           0
partner            0
dependents         0
senior             0
Tenure             0
monthly_cost    7283
package            0
survey             0
Class             49
dtype: int64
```

	partner	senior	Tenure	package
count	6752.000000	6752.000000	6752.000000	6752.000000
mean	0.547393	0.173134	8.722439	2.427725
std	0.497786	0.378391	6.408113	1.152222
min	0.000000	0.000000	-4.690416	1.000000
25%	0.000000	0.000000	3.000000	1.000000
50%	1.000000	0.000000	8.000000	2.000000
75%	1.000000	0.000000	14.000000	4.000000
max	1.000000	1.000000	30.000000	4.000000

Above images tells about the summary of the data types of features and null values in the each feature we can see that monthly cost feature have 7283 and Class feature has 49 missing values in the data set.

Before proceeding into EDA we have to handel null values and duplicates if any in the dataset.

We found that there are "598" duplicates based on the customer ID as it is mentioned as customer unique id we removed the all the duplicate row in dataset, So after deleting dataset contains 6752 records

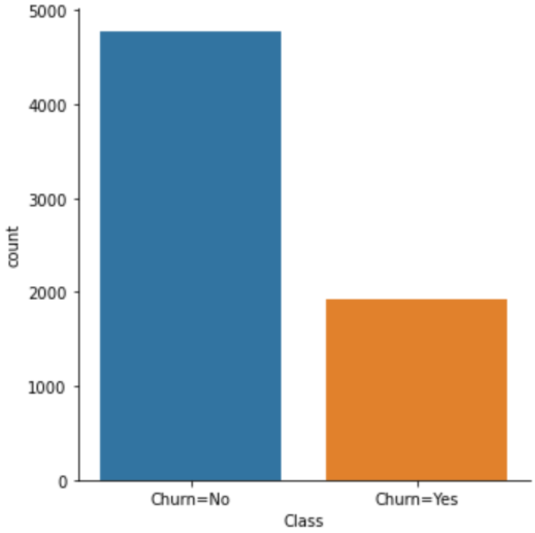
To handle NaN values in "monthly cost" we used "package" column, against each package we filled the monthly price of each package in monthly cost feature.
Monthly cost of the package are given by Lu's Communication but there is one missing value for the fourth package which is interpolated and considered as £47/month. Considered that second add-ons as £2 each and 1st time add-on as £3. There for package 4 is considered as £40 + £2 (Internet speed ad-on) +£2 (Tv-additional)+£3 (Landline talks ad-on).

Whereas NaN rows in "Class" feature are deleted. After deleting the Nan we finally have 6704 records in dataset.

3 Exploratory data analysis

Before we start EDA we found that there are few special charters in "Class" feature those are converted Churn-YES to analyse properly.

3.1 Target Feature Analysis

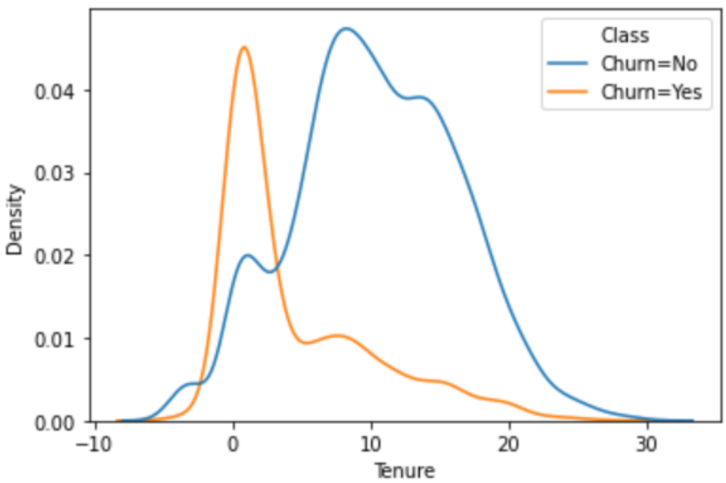


Above plot for the Count of churn Yes and No. It tells that there are total 4778 customers not churned and 1926 customers are churned. We can see that about 28 percentage of customers are endup churning.

3.2 Numerical Features Analysis

3.2.1 Tenure Feature Analysis

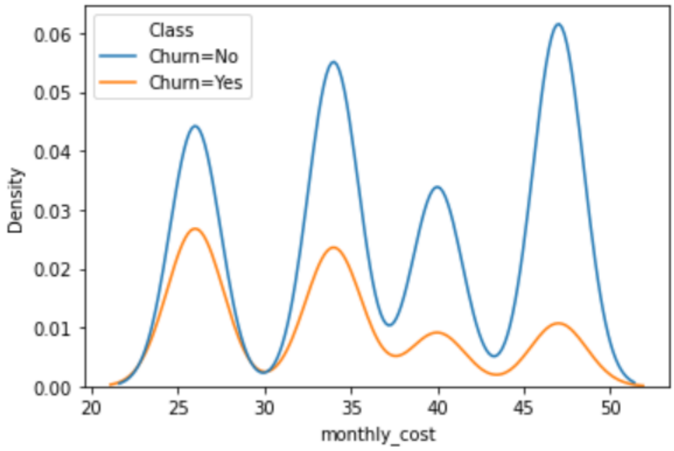
We used Kernel-Density-Estimation plot in order to visualise the probability distributions of the variables.



Above tenure plot for monthly Tenure tells that the customers tenure between 0 to 5 months are more likely to churn. And we also see that there are few outliers which are negative tenure in months.

3.2.2 Monthly Cost Feature Analysis

We used Kernel-Density-Estimation plot for monthly cost feature in order to visualise the probability distributions of the variables.



From above KDE plot we can tell that customer churn is getting decreased as the monthly cost is increasing. We see that the more churn is happening at package "1" of £26.

3.3 Categorical Features Analysis

3.3.1 Gender Feature Analysis

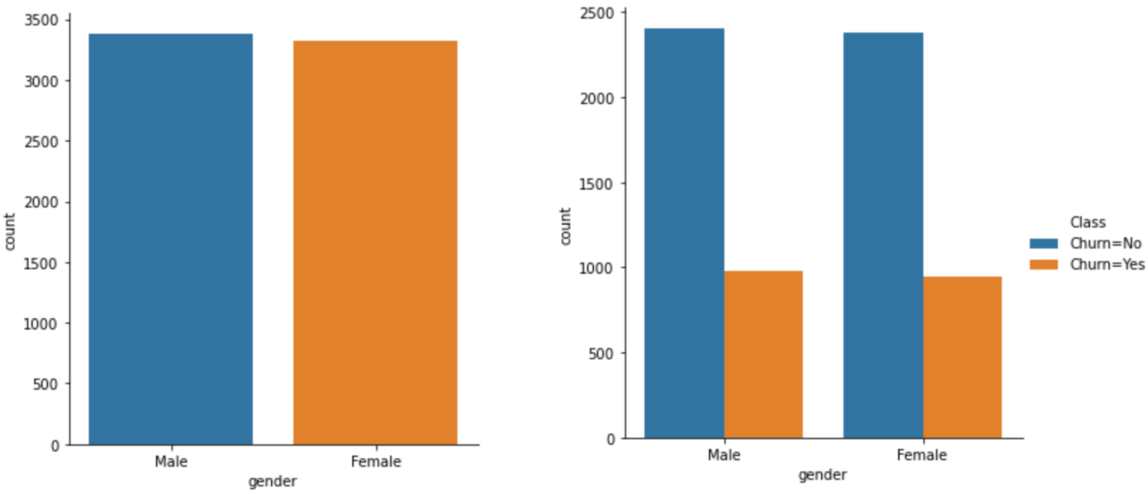


Figure 1: Visualization of Gender count and count against Class

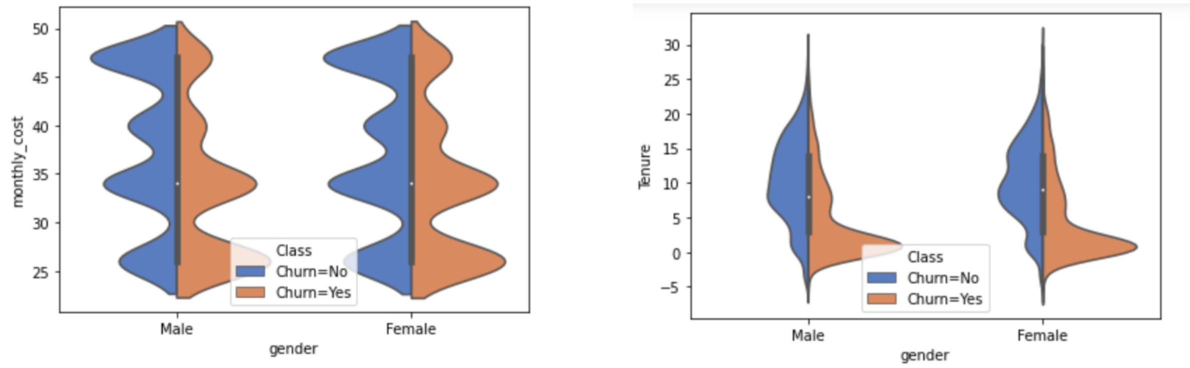


Figure 2: Visualization of Gender vs Tenure and Gender vs Monthly cost

From the Figure 1 and figure 2 we can say that gender doesn't show much effect on target class.As the count of both gender are same.

3.3.2 Partner Feature Analysis

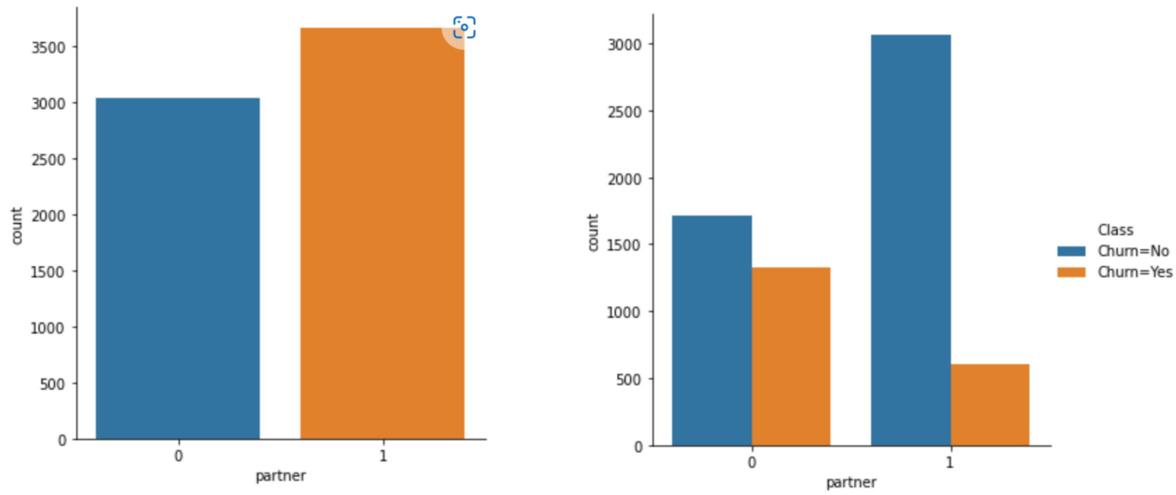


Figure 3: Visualization of partner count and count against Class

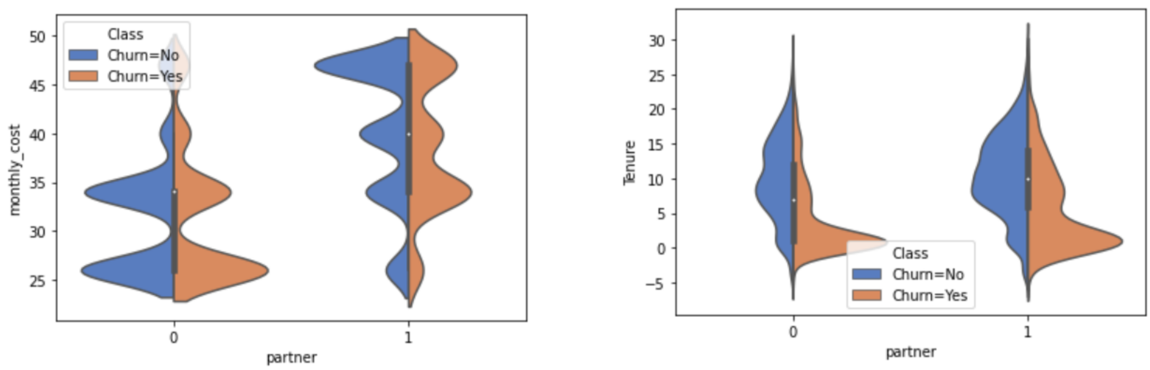


Figure 4: Visualization of partner vs Tenure and partner vs Monthly cost

From Figure 3 plot we observe that customers with no partners are end up churning-up more. We can also say that from Figure 4 customers with no partners and tenure between 0 to 5 months are churning up more.

3.3.3 Dependents Feature Analysis

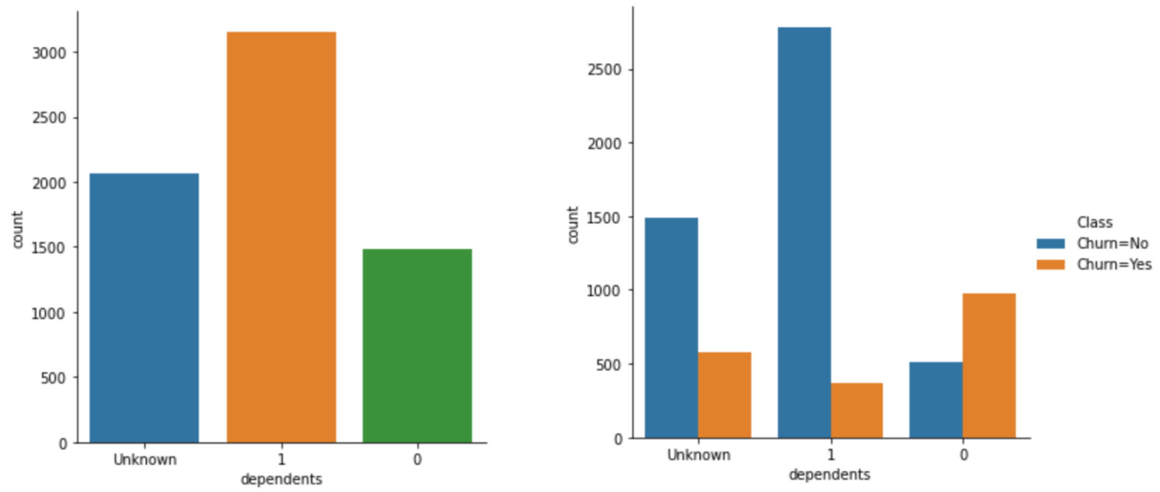


Figure 5: Visualization of dependents count and count against Class

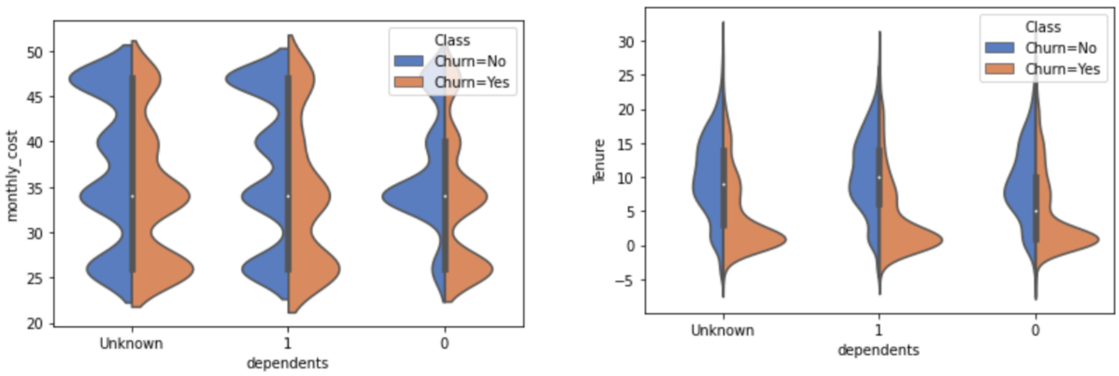


Figure 6: Visualization of dependents vs Tenure and partner vs Monthly cost

From Figure 5 plot we observe that customers with no dependents are more likely churning-up more.We can also say that from Figure 6 customers with no dependents and tenure between 0 to 5 months are churning up more.And also we can see there are few ”Unknown” class in dependents which need to be handled in the Data-pre processing.

3.3.4 Senior Feature Analysis

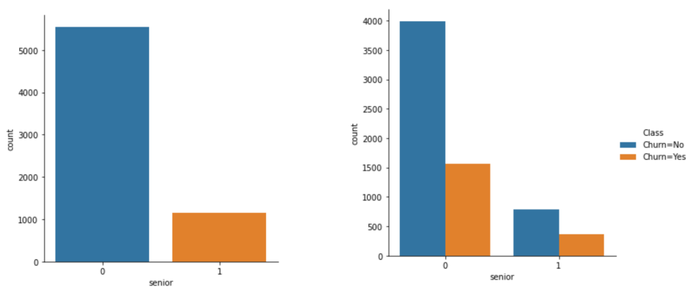


Figure 7: Visualization of Senior count and count against Class

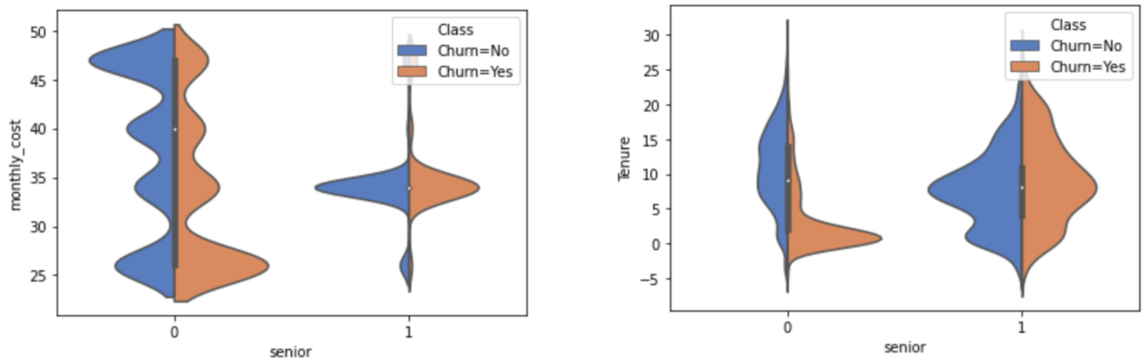


Figure 8: Visualization of senior vs Tenure and partner vs Monthly cost

From the above figure 7 says that there more non senior customers and they are churned more.From figure 8 we can observe that non seniors from tenure 0 to 5 months are end up churning more.

3.3.5 Location and Survey Feature Analysis

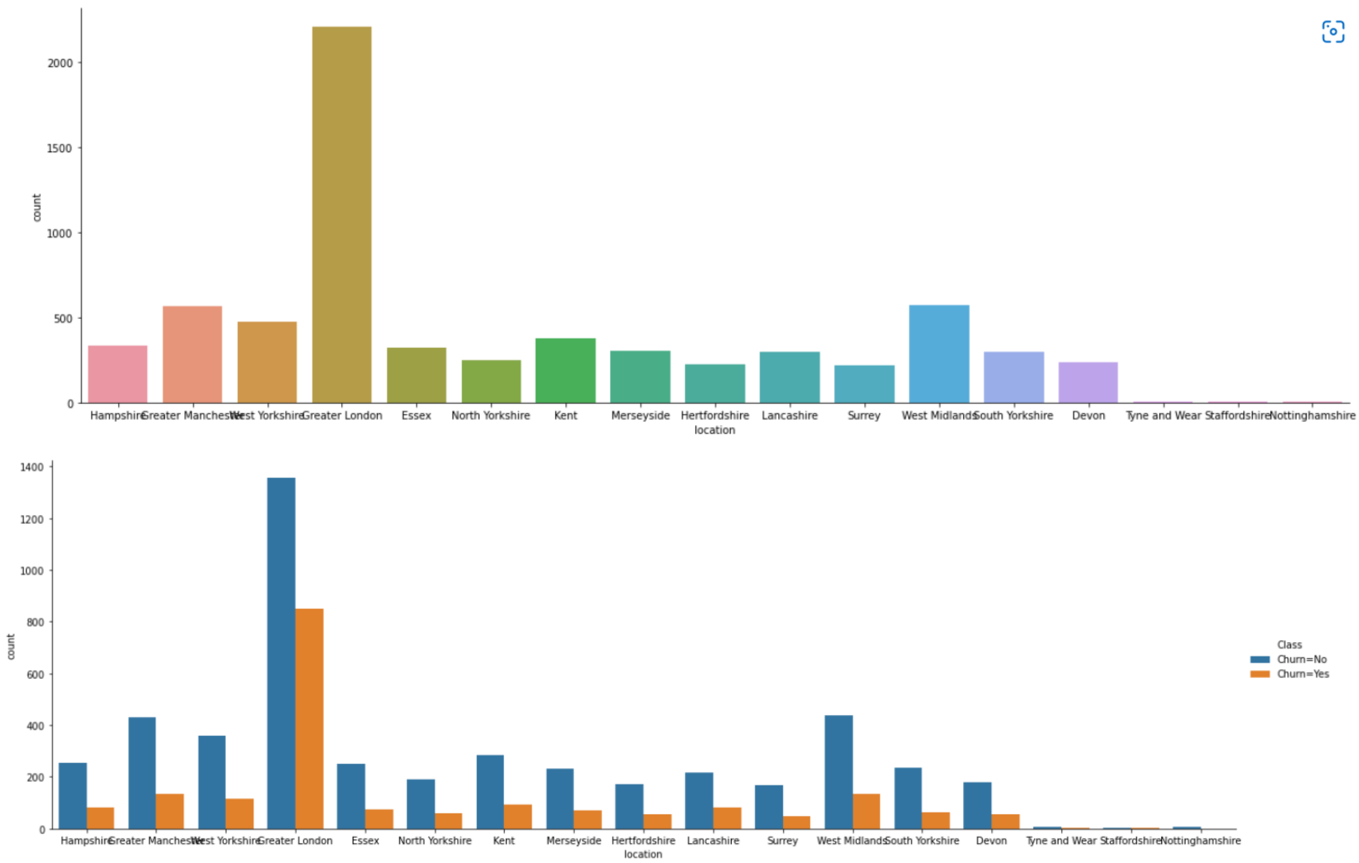


Figure 9: Location feature visualizations

Figure 9 tells that there are more customers in greater london area and churn is also more in the greater london area.

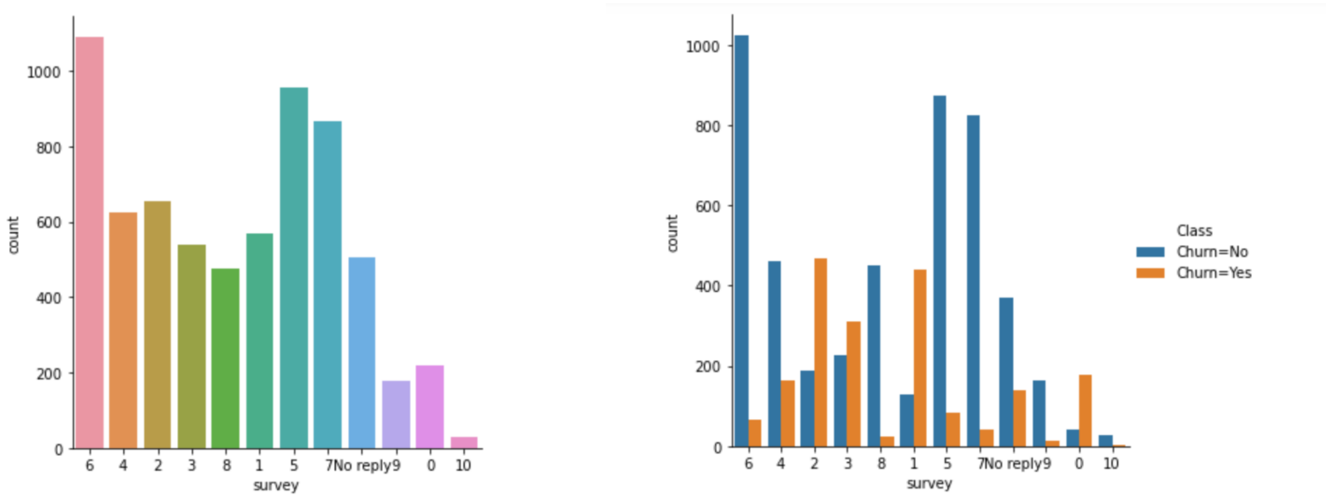


Figure 10: Visualization of survey feature

figure 10 says that customers who gave the survey ratings from 0 to 5 are churned up more than 5 to 10.And also observed few "No-reply" rows which need to be handled later.

4 Initial hypothesis

Based on EDA my initial hypothesis is as follows:

H0:Null Hypothesis: Customers with no dependents and less tenure are more likely to churn

Alternate Hypothesis:Customers with no dependents and less tenure are not churning.

5 Data Pre-Processing

Data pre-processing is data mining technique which is used to convert the raw data into useful data.

5.1 Handling outliers

Observed there few negative entries in the **Tenure** feature which was replaced by "0" months in dataset.

5.2 Handling missing values

Survey Feature: Observed there are **507** records which are having **"no reply"** in the **Survey** feature. There are many ways to fill the categorical missing values but we chooses to fill up with "mode" of that feature.

Dependents Feature: Observed that there are **2067** records which are having **Unknown**. We used Logistic regression model to predict the values of the "Unknown" in dependents column. We considered **Gender, partner, senior, Tenure, Class** are used as independent feature and **Dependents** feature as target value.

5.3 Handling the different scales of the data

We used **Normalization** on the feature "Tenure and "Monthly charges". Normalization is technique which changes the numerical values of column to 0 to 1 so that it improves the numerical stability of model and reduces the training if the data is large. We Converted male and female in gender feature to binary class(i.e Female=0 and male = 1)

5.4 Feature Selection

Feature selection is very important as it reduces the number of input variables to improve the model performance.
Location: In location feature we observed that "Greater London" is having more churn and more customers, so we created dummies for Greater london and rest of states as non greater london in dataset. **Customer id**: Customer id feature is the removed as it is unique value and there will not be any relation between the target and the feature.

To remove less significant association features with target value we used **P value** test. Test results are as follows:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.7123	0.041	17.470	0.000	0.632	0.792
gender	0.0041	0.007	0.609	0.543	-0.009	0.017
partner	-0.0679	0.008	-8.244	0.000	-0.084	-0.052
dependents	-0.4766	0.009	-55.949	0.000	-0.493	-0.460
senior	-0.0825	0.011	-7.633	0.000	-0.104	-0.061
Tenure	-0.3715	0.017	-21.650	0.000	-0.405	-0.338
monthly_cost	-0.1766	0.190	-0.931	0.352	-0.549	0.195
package	0.0412	0.063	0.657	0.512	-0.082	0.164
survey	-0.0620	0.002	-38.377	0.000	-0.065	-0.059
location_Greater London	0.3867	0.021	18.653	0.000	0.346	0.427
location_Non london area	0.3256	0.021	15.744	0.000	0.285	0.366

Omnibus:	355.252	Durbin-Watson:	2.022
Prob(Omnibus):	0.000	Jarque-Bera (JB):	831.467
Skew:	0.328	Prob(JB):	2.81e-181
Kurtosis:	4.596	Cond. No.	4.15e+16

Based on the above p values we removed **gender and package** feature as there p values ¿= 0.05 which tells that they are less significant.

5.5 Class Imbalance

We observed that there is class imbalance between churn=1926 and not churn=4778 in the labeled "Class" feature. This may effect the model performance.So before we feed it into model we upsampled churn class by using random sampling.

6 Developing and testing Machine learning Models

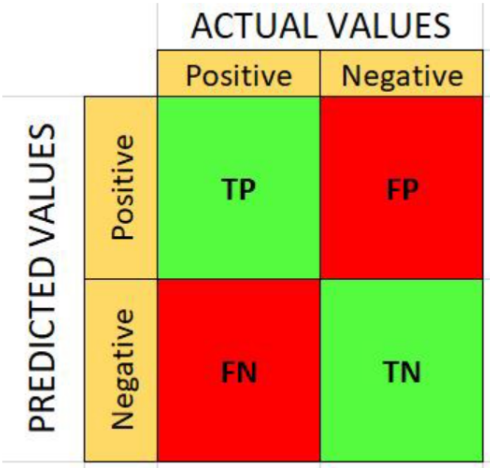
6.1 Splitting Data

Splitting the data into training and testing so that we can train the model by using the training data and test the model using test data. Here we split the data into 80:20 ratio(i.e, 80 percentage of records are for training and 20 percentage for testing). To split the data we used **test train split**, which will split the data by choosing randomly.

6.2 Evaluation metrics to evaluate model

In Binary classification model there many ways to evaluate the model but most effective metrics is Confusion matrix.
Based on the confusion matrix we can calculate the **accuracy, precision, recall, F1 score and etc**.

Confusion Matrix Confusion matrix comprises of TP,TN,FP and FN.
TP:When actual positive and predicted positive
TN:When actual Negative and predicted Negative
FP:When actual Negative and predicted positive
FN:When actual positive and predicted Negative



Accuracy:Accuracy is ration of correctly predicted by total no.of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision:It measures correctness in the prediction.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision:It measures actual observations which are predicted correctly.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score:It is hormonic mean of precision and recall.

$$\text{F1-Score} = 2* \frac{(\text{Recall}*\text{Precision})}{(\text{Recall} + \text{Precision})}$$

6.3 Logistic Regression model

Logistic model is mostly commonly used binary classification algorithm as exactly predicts the binary class outputs.It perform good when the data is linearly separable and it ll interpet model coefficient which tell about the importance of the features in dataset

Logistic Regression				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	89.33%	89.86%	89.68%	89.77%

6.4 K Nearest Neighbor Classification

KNN Classification model is widely used classification algorithm as it is very simple and it classifies the data point based on the similarity in the group of neighbouring data points.

KNN model				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	91.47%	93.53%	89.88%	90.67%

6.5 Random forest Classifier

Random forest Classifier is a bootstrapping algorithm with decision tree model.It assumes number of cases in training set and samples of cases taken at random.It is based on the majority voting.Hence it give better accuracy by overcoming over-fitting issue which we observe in decision tree.

Random Forest model				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	94.61%	93.92%	95.89%	94.89%

6.6 Naive Bayes

Navie Bayes is a classification algorithm based on bayes theorem with an assumption of Independence among algorithms.It is widely used model for the binary and multi classification

Naive Bayes				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	89.07%	89.33%	89.78%	89.56%

6.7 Support Vector Machine Classifier

SVM algoritm uses classifies the data within the degree of polarity

Support Vector Machine Classifier				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	90.22%	91.17%	89.98%	90.57%

6.8 Gradient Boosting Classifier

This model is group weak learner models one among them is decision tree.This model is very effective in high dimensional data also.

Gradient Boosting Classifier				
Data	Accuracy	Precision	Recall	F1 Score
Test Data	92.89%	93.45%	92.89%	93.17%

7 Conclusion

After fitting and testing the data in all above mentioned models, We got different accuracy's , precision ,Recall and F1 score. The summary of all the models will give us the change to get into conclusion in terms of the evaluation metrics.

Summary model's Performance				
Model/Algorithm	Accuracy	Precision	Recall	F1 Score
Logistic Regression	89.33%	89.86%	89.68%	89.77%
KNN	91.47%	93.53%	89.88%	90.67%
Random Forest	94.61%	93.92%	95.89%	94.89%
Naive Bayes	89.07%	89.33%	89.78%	89.56%
Support Vector Machine	90.22%	91.17%	89.98%	90.57%
Gradient Boosting	92.89%	93.45%	92.89%	93.17%

From the above table we can say that **Random Forest** algorithm give more accuracy, precision, recall and F1 Score.For the data we need more Recall percentage as it minimize the False Negative. If the Recall score is more it says that there are less False Negative which means less miss-prediction of likely to churn customer as not.

Confusion Matrix of the models:

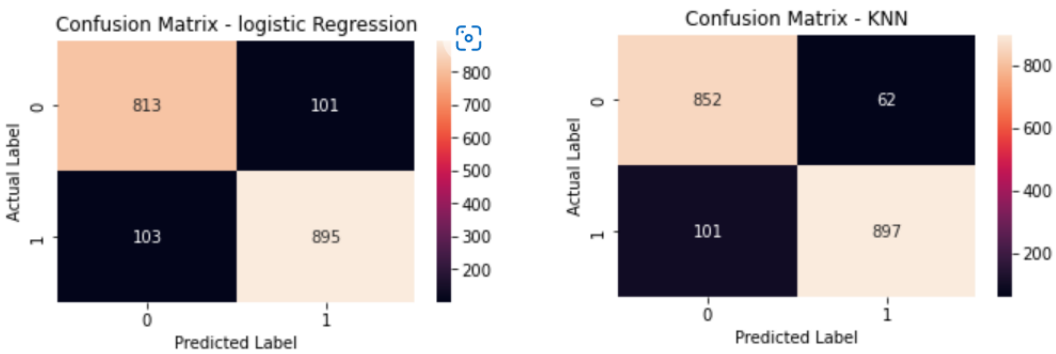


Figure 11: Confusion Matrix of Logistic Regression and KNN

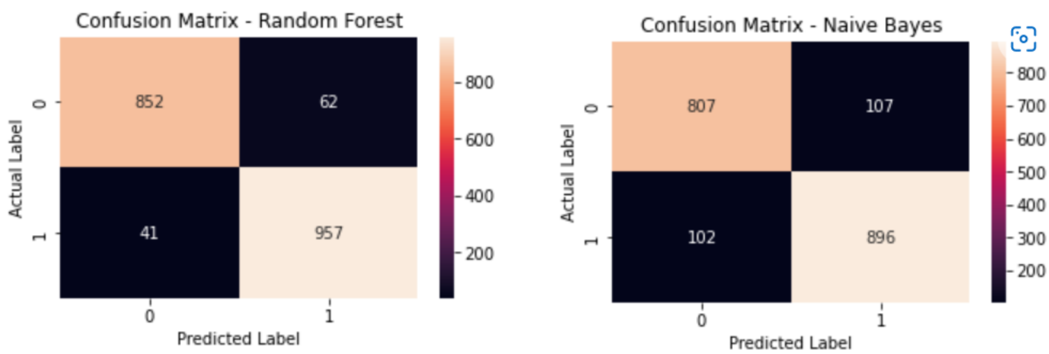


Figure 12: Confusion Matrix of Random Forest and Naive Bayes

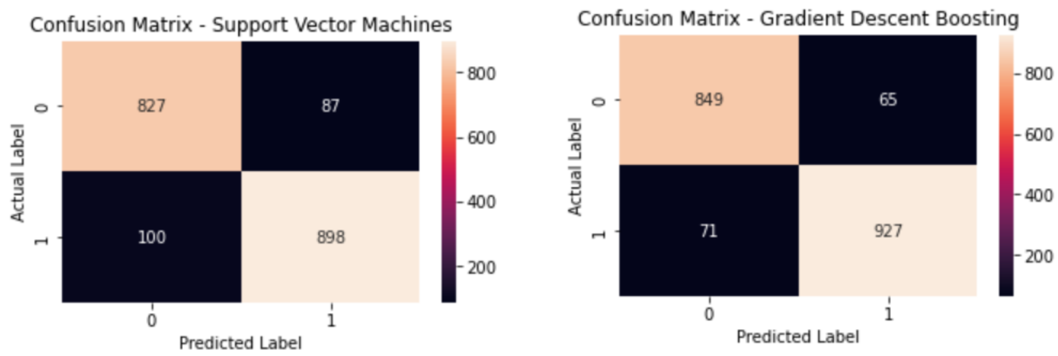
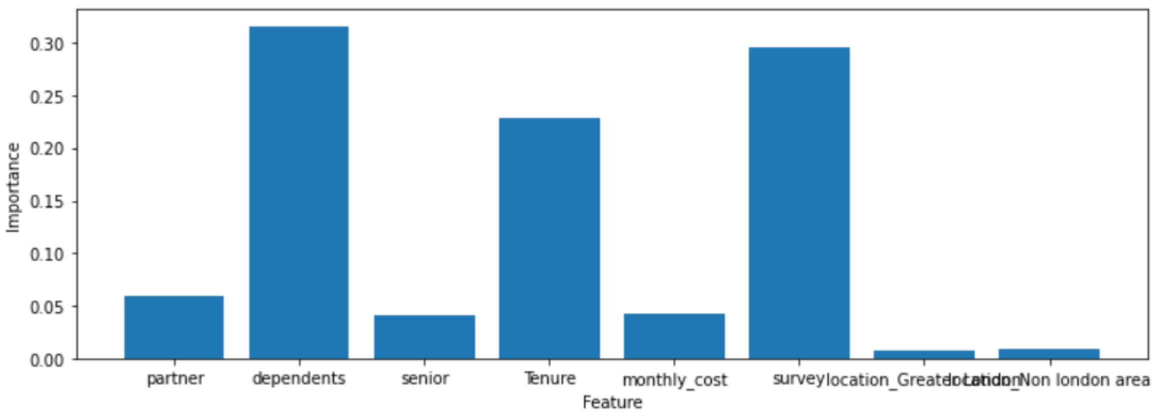


Figure 13: Confusion Matrix of SVM and Gradient Boosting

Feature Importance plot of Random forest model:



From the above plot we can say that Dependents and Survey and Tenure are more important and playing major role in getting the accuracy and recall of model. As my **hypothesis** says Customers with no dependents and less tenure are more likely to churn can be accepted. Finally our model says **Customer with no dependents are more likely to churn**

References

[1] Medium. 2022. Churn Prediction with Machine Learning. [online] Available at: ¡https://towardsdatascience.com/churn-prediction-with-machine-learning-ca955d52bd8c¿ [Accessed 8 August 2022].

[2] Medium. 2022. Customer Churn Analysis: EDA. [online] Available at: ¡https://towardsdatascience.com/customer-churn-analysis-eda-a688c8a166ed¿ [Accessed 8 August 2022].

[3] Brownlee, J., 2022. Feature Importance and Feature Selection With XGBoost in Python. [online] Machine Learning Mastery. Available at: ¡https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/¿ [Accessed 8 August 2022].

[4] Kaggle.com. 2022. Telco Churn Prediction — Feature Engineering[EDA]. [online] Available at: ¡https://www.kaggle.com/code/mechatronixs/telco-churn-prediction-feature-engineering-eda¿ [Accessed 8 August 2022].

[5] Medium. 2022. What is a confusion matrix?. [online] Available at: ¡https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5¿ [Accessed 8 August 2022].

A Appendix

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LinearRegression
import sklearn.metrics as sm
import seaborn as sns
import statsmodels.api as sms
from scipy import stats
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# import plotly.express as px
```

In [2]:
```
df = pd.read_csv("Group 2 (2).csv")
```

In [3]:
```
del df[df.columns[0]]
```

In [4]:
```
df.shape
```

Out[4]:
```
(7350, 11)
```

In [5]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7350 entries, 0 to 7349
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   customer_id   7350 non-null   object
 1   gender        7350 non-null   object
 2   location      7350 non-null   object
 3   partner       7350 non-null   int64
 4   dependents    7350 non-null   object
 5   senior        7350 non-null   int64
 6   Tenure        7350 non-null   float64
 7   monthly_cost  67 non-null     object
 8   package       7350 non-null   int64
 9   survey        7350 non-null   object
 10  Class         7301 non-null   object
dtypes: float64(1), int64(3), object(7)
memory usage: 631.8+ KB
None
```

In [6]:
```
df.head()
```

Out[6]:

	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class
0	K3713	Male	Hampshire	0	Unknown	0	12.0	NaN	1	6	Churn=No
1	D9048	Male	Greater Manchester	1	1	0	21.0	NaN	4	6	Churn=No
2	K8227	Female	West Yorkshire	0	Unknown	0	0.0	NaN	1	4	Churn=Yes
3	H3533	Male	Greater London	1	1	1	11.0	NaN	2	4	Churn=No
4	J4501	Male	Greater London	0	0	0	7.0	NaN	4	2	Churn=Yes

In [7]:
```
print(df.isnull().sum())
```

```
customer_id        0
gender             0
location           0
partner            0
dependents         0
senior             0
Tenure             0
monthly_cost    7283
package            0
survey             0
Class             49
dtype: int64
```

```
duplicateRowsDF = df[df['customer_id'].duplicated()]
duplicateRowsDF
df = df.drop_duplicates(subset=["customer_id"], keep='first')
df.shape
```

Out[8]:
```
(6752, 11)
```

In [9]:
```
df.shape
```

Out[9]:
```
(6752, 11)
```

In [10]:
```
df.dtypes
```

Out[10]:
```
customer_id      object
gender           object
location         object
partner           int64
dependents       object
senior            int64
Tenure          float64
monthly_cost     object
package           int64
survey           object
Class            object
dtype: object
```

In [11]:
```
df.describe()
```

Out[11]:

	partner	senior	Tenure	package
count	6752.000000	6752.000000	6752.000000	6752.000000
mean	0.547393	0.173134	8.722439	2.427725
std	0.497786	0.378391	6.408113	1.152222
min	0.000000	0.000000	-4.690416	1.000000
25%	0.000000	0.000000	3.000000	1.000000
50%	1.000000	0.000000	8.000000	2.000000
75%	1.000000	0.000000	14.000000	4.000000
max	1.000000	1.000000	30.000000	4.000000

In [12]:
```
df['monthly_cost'][df.package==1]=26
df['monthly_cost'][df.package==2]=34
df['monthly_cost'][df.package==3]=40
df['monthly_cost'][df.package==4]=47
```

In [13]:
```
df.head()
```

Out[13]:

	customer_id	gender	location	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class
0	K3713	Male	Hampshire	0	Unknown	0	12.0	26	1	6	Churn=No
1	D9048	Male	Greater Manchester	1	1	0	21.0	47	4	6	Churn=No
2	K8227	Female	West Yorkshire	0	Unknown	0	0.0	26	1	4	Churn=Yes
3	H3533	Male	Greater London	1	1	1	11.0	34	2	4	Churn=No
4	J4501	Male	Greater London	0	0	0	7.0	47	4	2	Churn=Yes

In [14]:
```
df = df.dropna()
```

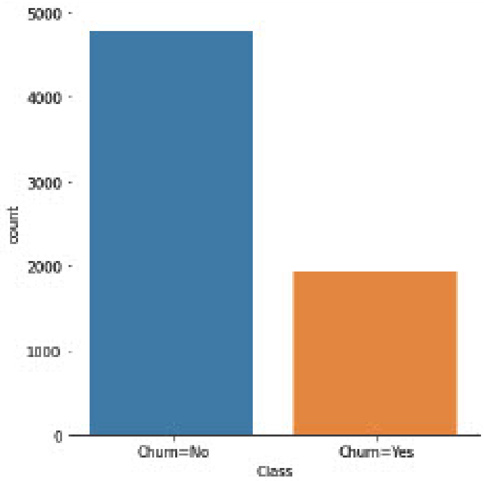
In [15]:
```
df.shape
```

Out[15]:
```
(6704, 11)
```

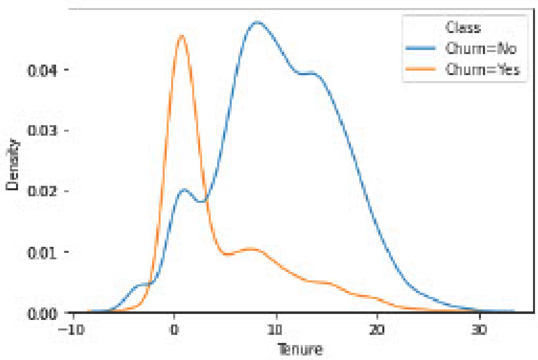
In [16]:
```
print(df['Class'].value_counts())
```

```
Churn=No     4778
Churn=Yes    1911
Y$e$s$$        15
Name: Class, dtype: int64
```

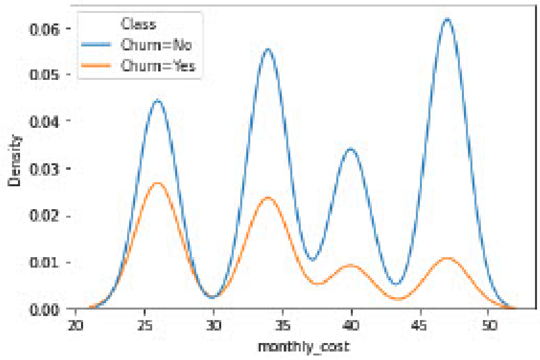

In [17]:
```
df['monthly_cost']=df['monthly_cost'].astype(int)
```

In [18]:
```
df.shape
```
Out[18]:
```
(6704, 11)
```
In [19]:
```
churn_numeric = {'Y$e$s$$':'Churn=Yes'}
df.Class.replace(churn_numeric, inplace=True)
```
In [20]:
```
df.dtypes
```
Out[20]:
```
customer_id      object
gender           object
location         object
partner           int64
dependents       object
senior            int64
Tenure          float64
monthly_cost      int32
package           int64
survey           object
Class            object
dtype: object
```
In [21]:
```
print(df['Class'].value_counts())
```
```
Churn=No     4778
Churn=Yes    1926
Name: Class, dtype: int64
```
In [22]:
```
ax6 = sns.catplot(x="Class", kind="count", data=df)
```


Numerical data

In [23]:
```
sns.kdeplot(data=df, x="Tenure", hue="Class")
```
Out[23]:
```
<AxesSubplot:xlabel='Tenure', ylabel='Density'>
```


In [24]:
```
sns.kdeplot(data=df, x="monthly_cost", hue="Class")
```
Out[24]:
```
<AxesSubplot:xlabel='monthly_cost', ylabel='Density'>
```



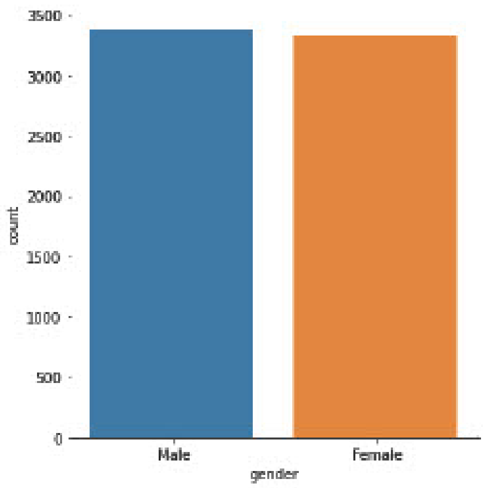
Gender:

In [25]:
```python
print(df['gender'].value_counts())
```

```
Male      3380
Female    3324
Name: gender, dtype: int64
```

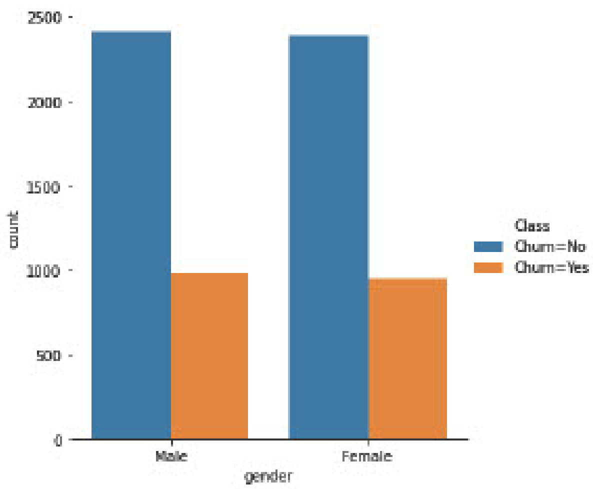
In [26]:
```python
sns.catplot(x="gender", kind="count", data=df)
```

Out[26]: <seaborn.axisgrid.FacetGrid at 0x1eabbb20280>



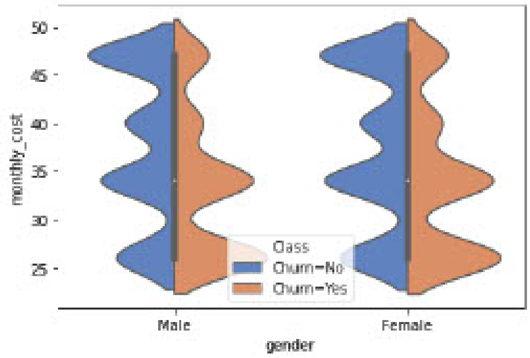
In [27]:
```python
sns.catplot(x="gender", kind="count", hue="Class", data=df)
```

Out[27]: <seaborn.axisgrid.FacetGrid at 0x1eaba3d25e0>



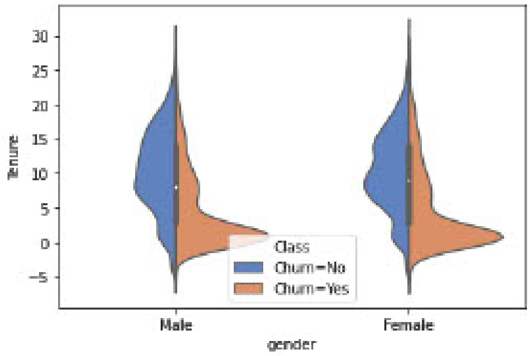
In [28]:
```python
sns.violinplot(x="gender", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

Out[28]: <AxesSubplot:xlabel='gender', ylabel='monthly_cost'>



In [29]:
```
sns.violinplot(x="gender", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

Out[29]: <AxesSubplot:xlabel='gender', ylabel='Tenure'>



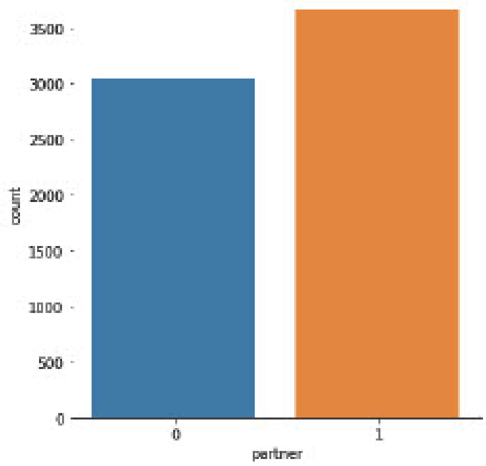
Partner

In [30]:
```
print(df['partner'].value_counts())
```

```
1    3667
0    3037
Name: partner, dtype: int64
```

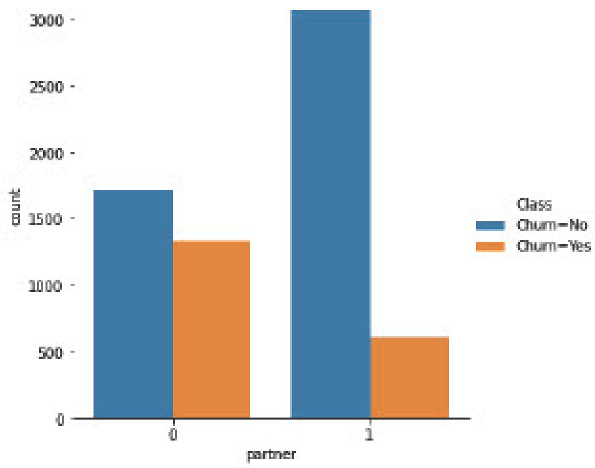
In [31]:
```
sns.catplot(x="partner", kind="count", data=df)
```

Out[31]: <seaborn.axisgrid.FacetGrid at 0x1eabbdfe430>



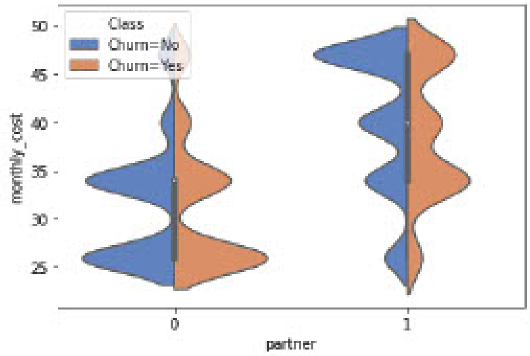
In [32]:
```
sns.catplot(x="partner", kind="count", hue="Class", data=df)
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x1eabbcd57f0>



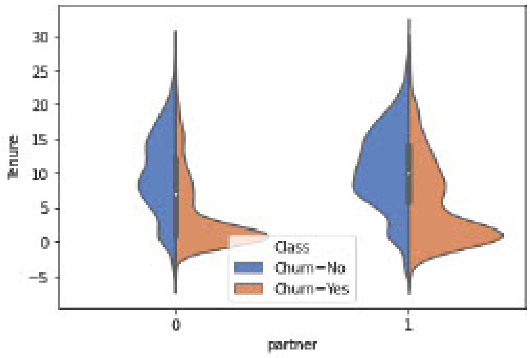
In [33]:
```
sns.violinplot(x="partner", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

Out[33]: <AxesSubplot:xlabel='partner', ylabel='monthly_cost'>



In [34]:
```
sns.violinplot(x="partner", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

Out[34]: <AxesSubplot:xlabel='partner', ylabel='Tenure'>



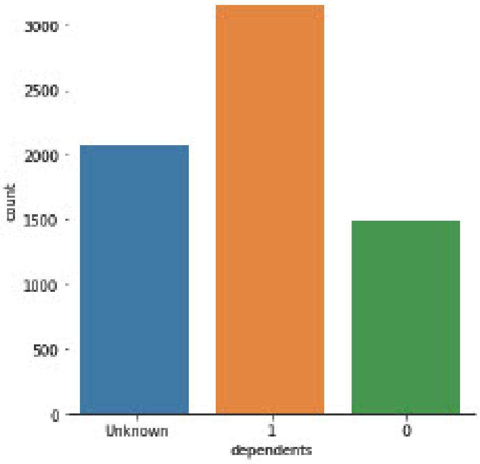
Dependents

In [35]:
```
print(df['dependents'].value_counts())
```

```
1          3151
Unknown    2067
0          1486
Name: dependents, dtype: int64
```

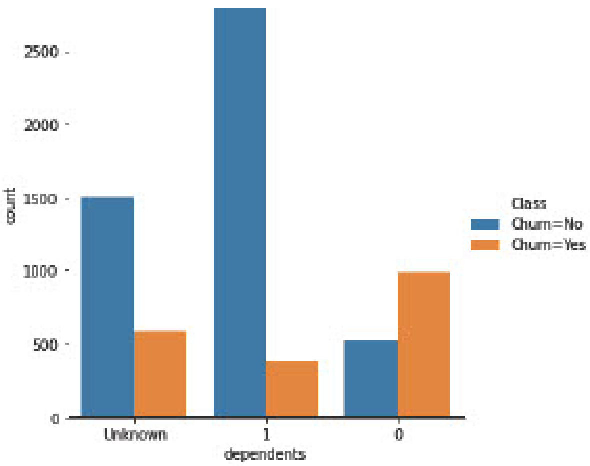
In [36]:
```
sns.catplot(x="dependents", kind="count", data=df)
```

Out[36]: <seaborn.axisgrid.FacetGrid at 0x1eabbde15e0>



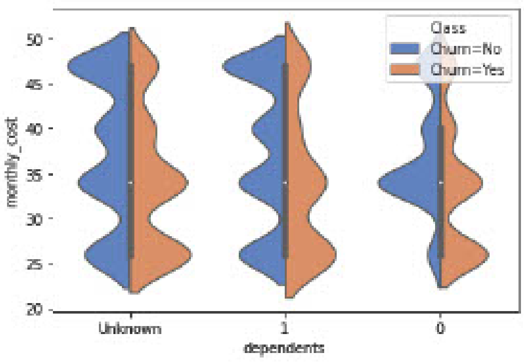
In [37]:
```
sns.catplot(x="dependents", kind="count", hue="Class", data=df)
```

Out[37]: <seaborn.axisgrid.FacetGrid at 0x1eabc085be0>



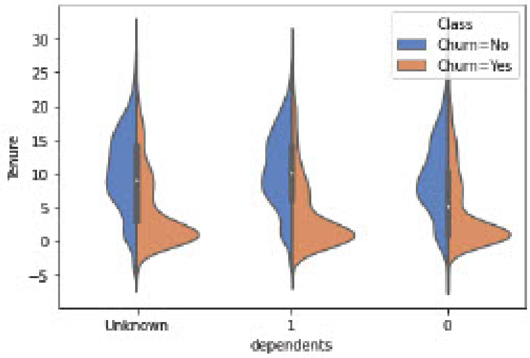
In [38]:
```
sns.violinplot(x="dependents", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

Out[38]: <AxesSubplot:xlabel='dependents', ylabel='monthly_cost'>



In [39]:
```
sns.violinplot(x="dependents", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

Out[39]: <AxesSubplot:xlabel='dependents', ylabel='Tenure'>



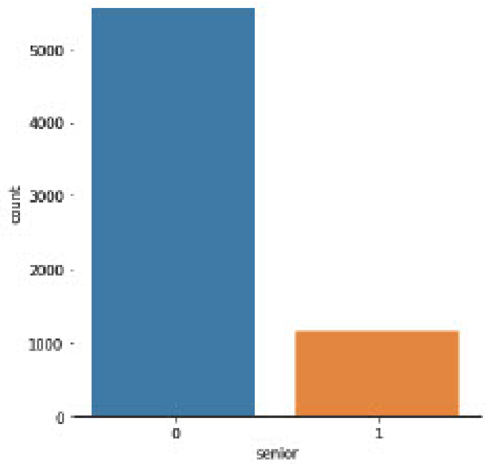
Seniors

In [40]:
```
print(df['senior'].value_counts())
```

```
0    5546
1    1158
Name: senior, dtype: int64
```

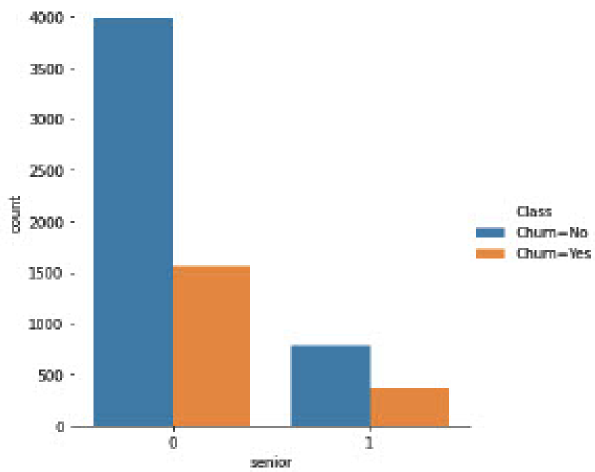
In [41]:
```
sns.catplot(x="senior", kind="count", data=df)
```

Out[41]: <seaborn.axisgrid.FacetGrid at 0x1eabd1703a0>



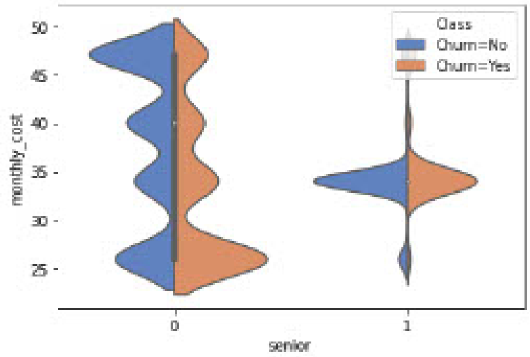
In [42]:
```
sns.catplot(x="senior", kind="count", hue="Class", data=df)
```

Out[42]: <seaborn.axisgrid.FacetGrid at 0x1eabd180820>



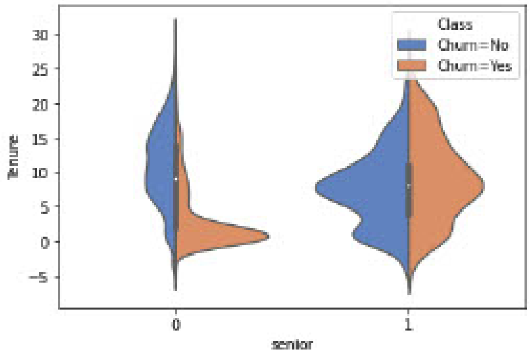
In [43]:
```
sns.violinplot(x="senior", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

Out[43]: <AxesSubplot:xlabel='senior', ylabel='monthly_cost'>



In [44]:
```
sns.violinplot(x="senior", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

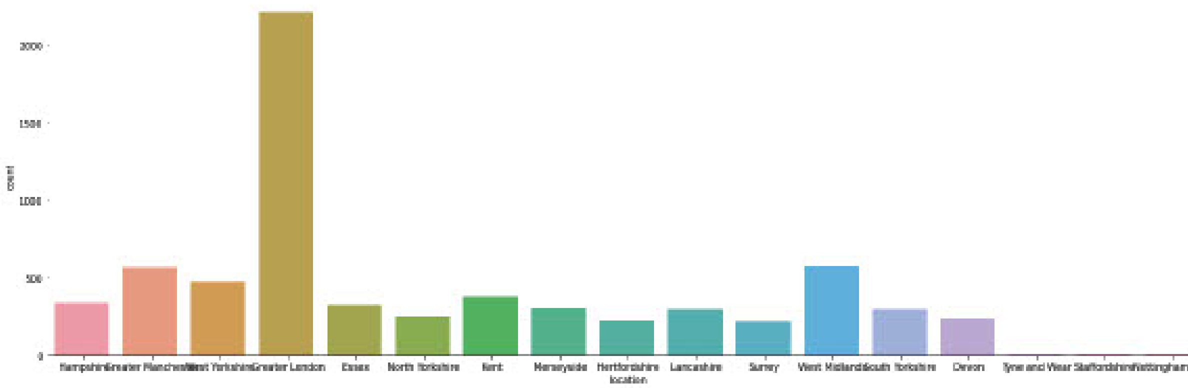
Out[44]: <AxesSubplot:xlabel='senior', ylabel='Tenure'>



Location

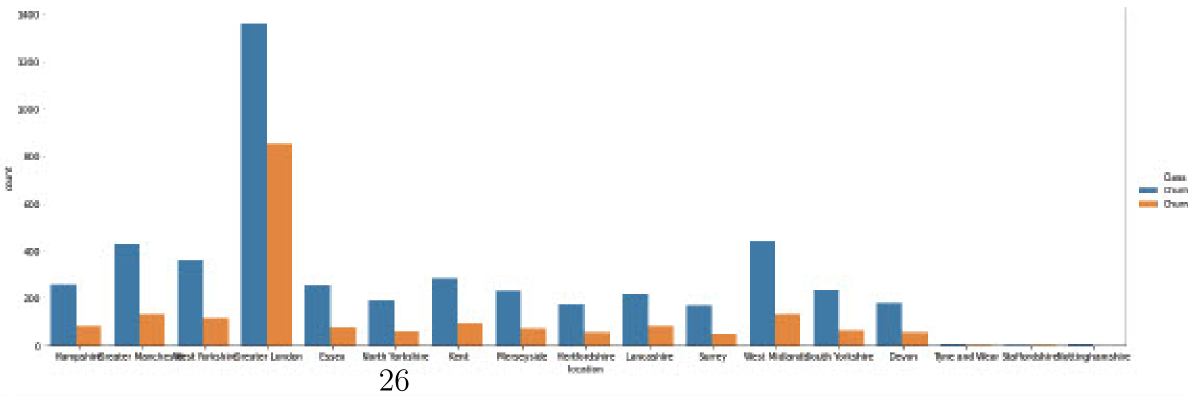
In [45]:
```
sns.catplot(x="location", kind="count", data=df,height=6, aspect=3)
```

Out[45]: <seaborn.axisgrid.FacetGrid at 0x1eabd3e8a90>



In [46]:
```
sns.catplot(x="location", kind="count", hue="Class", data=df,height=6, aspect=3)
```

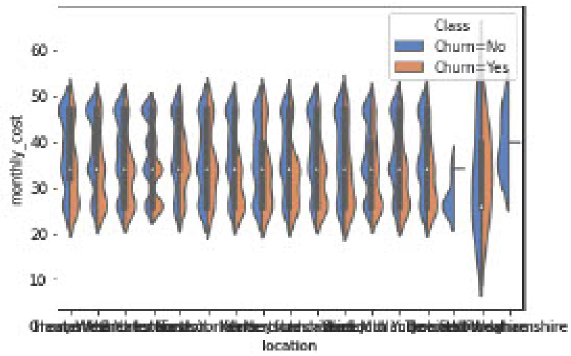
Out[46]: <seaborn.axisgrid.FacetGrid at 0x1eabd3e0190>



In [47]:

```
sns.violinplot(x="location", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

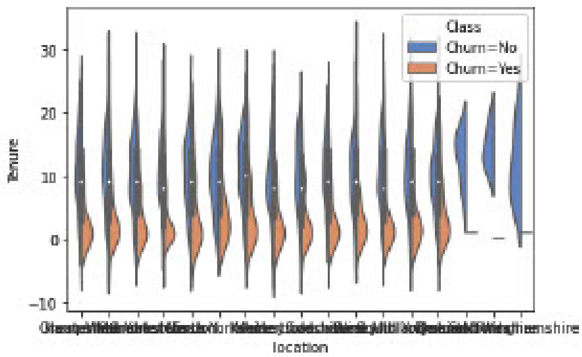
Out[47]: <AxesSubplot:xlabel='location', ylabel='monthly_cost'>



In [48]:

```
sns.violinplot(x="location", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

Out[48]: <AxesSubplot:xlabel='location', ylabel='Tenure'>



Survey

In [49]:

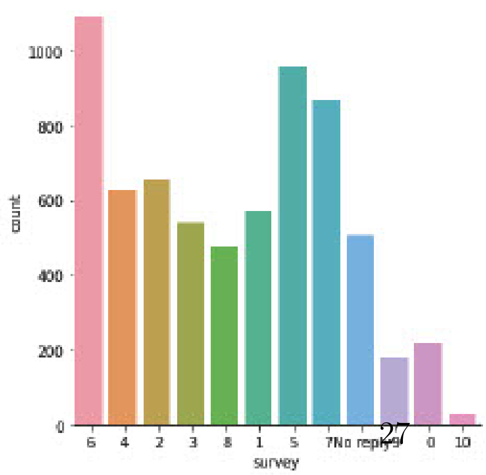
```
print(df['survey'].value_counts())
```

```
6           1090
5            956
7            865
2            654
4            625
1            568
3            539
No reply     507
8            476
0            218
9            177
10            29
Name: survey, dtype: int64
```

In [50]:

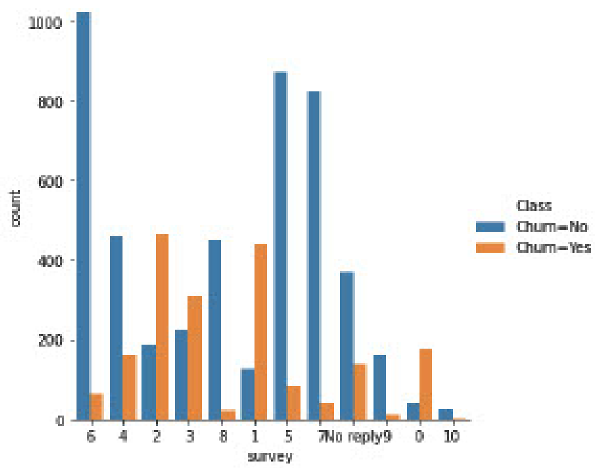
```
sns.catplot(x="survey", kind="count", data=df)
```

Out[50]: <seaborn.axisgrid.FacetGrid at 0x1eabdaa44c0>



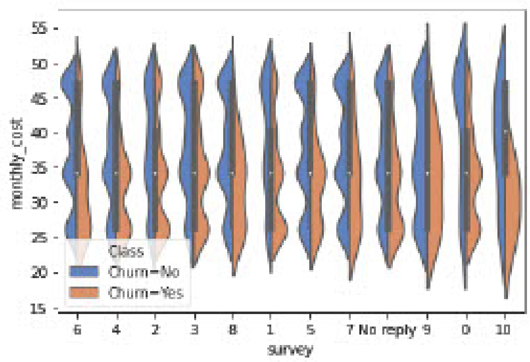
In [51]:
```
sns.catplot(x="survey", kind="count", hue="Class", data=df)
```

Out[51]: <seaborn.axisgrid.FacetGrid at 0x1eabdd56730>



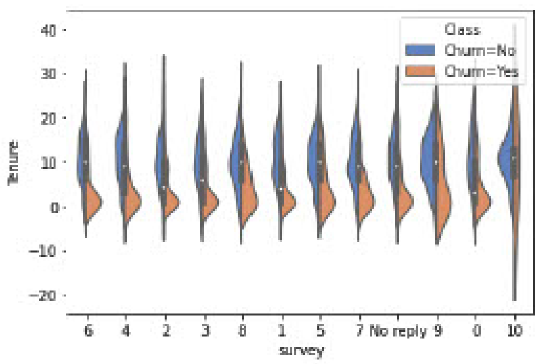
In [52]:
```
sns.violinplot(x="survey", y="monthly_cost",hue="Class", data=df,palette="muted", split=True)
```

Out[52]: <AxesSubplot:xlabel='survey', ylabel='monthly_cost'>



In [53]:
```
sns.violinplot(x="survey", y="Tenure",hue="Class", data=df,palette="muted", split=True)
```

Out[53]: <AxesSubplot:xlabel='survey', ylabel='Tenure'>



H0:people who dont have dependents are churning more.

Data pre processing

In [54]:
```
churn_numeric = {'Churn=Yes':1, 'Churn=No':0,'Y$e$s$$':1}
df.Class.replace(churn_numeric, inplace=True)
df.gender.replace({'Female':0,'Male':1},inplace=True)
```

In [55]:
```
df=df.replace(['West Midlands', 'Greater Manchester', 'West Yorkshire', 'Kent','Hampshire','Essex','Lancashire','Merse
df=pd.get_dummies(df,columns=['location'])
df
```

Out[55]:

customer_id	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area

	customer_id	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area
0	K3713	1	0	Unknown	0	12.0	26	1	6	0	0	1
1	D9048	1	1	1	0	21.0	47	4	6	0	0	1
2	K8227	0	0	Unknown	0	0.0	26	1	4	1	0	1
3	H3533	1	1	1	1	11.0	34	2	4	0	1	0
4	J4501	1	0	0	0	7.0	47	4	2	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
7344	I3187	1	1	Unknown	0	0.0	26	1	6	1	1	0
7345	H7244	0	0	0	0	1.0	26	1	1	1	1	0
7347	I5775	1	1	1	0	16.0	47	4	2	0	1	0
7348	E9984	0	0	1	0	17.0	26	1	4	1	1	0
7349	F2835	1	0	1	0	13.0	34	2	7	0	0	1

6704 rows × 12 columns

In [56]:

```
df.groupby('dependents')['Class'].value_counts()
```

Out[56]:

```
dependents  Class
0           1         973
            0         513
1           0        2780
            1         371
Unknown     0        1485
            1         582
Name: Class, dtype: int64
```

In [57]:

```
df.gender.replace({'Male':0,'Female':1}, inplace=True)
df.loc[df['Tenure'] < 0, 'Tenure'] = 0
```

In [58]:

```
df['Tenure']=df['Tenure'].astype(int)
# df['dependents']=df['dependents'].astype(int)
# df['survey']=df['survey'].astype(int)
df['Class']=df['Class'].astype(int)
df['monthly_cost']=df['monthly_cost'].astype(int)
df['location_Greater London']=df['location_Greater London'].astype(int)
df['location_Non london area']=df['location_Non london area'].astype(int)
```

In [59]:

```
df['Tenure'] = MinMaxScaler().fit_transform(np.array(df['Tenure']).reshape(-1,1))
df['monthly_cost'] = MinMaxScaler().fit_transform(np.array(df['monthly_cost']).reshape(-1,1))
# view normalized data
display(df)
```

	customer_id	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area
0	K3713	1	0	Unknown	0	0.400000	0.000000	1	6	0	0	1
1	D9048	1	1	1	0	0.700000	1.000000	4	6	0	0	1
2	K8227	0	0	Unknown	0	0.000000	0.000000	1	4	1	0	1
3	H3533	1	1	1	1	0.366667	0.380952	2	4	0	1	0
4	J4501	1	0	0	0	0.233333	1.000000	4	2	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
7344	I3187	1	1	Unknown	0	0.000000	0.000000	1	6	1	1	0
7345	H7244	0	0	0	0	0.033333	0.000000	1	1	1	1	0
7347	I5775	1	1	1	0	0.533333	1.000000	4	2	0	1	0
7348	E9984	0	0	1	0	0.566667	0.000000	1	4	1	1	0
7349	F2835	1	0	1	0	0.433333	0.380952	2	7	0	0	1

6704 rows × 12 columns

In [60]:

```
df1_test=df[df['dependents'] == "Unknown"]
df2_train=df[df['dependents'] != "Unknown"] #df2 main
df1_test.shape,df2_train.shape
```



```
((2067, 12), (4637, 12))
```

Out[60]:

In [61]:

```
df2_train['dependents']=df2_train['dependents'].astype(int)
classifier = LogisticRegression()
classifier.fit(df2_train[['gender','partner','senior','Tenure','Class']], df2_train['dependents'])
pred=classifier.predict(df1_test[['gender','partner','senior','Tenure','Class']])
pred
```

Out[61]:

```
array([1, 0, 1, ..., 1, 1, 0])
```

In [62]:

```
df1_test['dependents']=pred
df1_test
```

Out[62]:

	customer_id	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area
0	K3713	1	0	1	0	0.400000	0.000000	1	6	0	0	1
2	K8227	0	0	0	0	0.000000	0.000000	1	4	1	0	1
5	K3269	0	1	1	0	0.600000	0.000000	1	3	0	0	1
7	E1851	1	0	1	0	0.066667	0.000000	1	4	0	0	1
8	H3588	0	1	0	0	0.400000	0.666667	3	1	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
7334	D4511	1	1	1	0	0.566667	1.000000	4	7	0	1	0
7337	G4484	0	0	1	0	0.000000	0.000000	1	7	0	0	1
7341	C8964	0	1	1	0	0.766667	0.666667	3	4	0	1	0
7342	D1975	0	1	1	0	0.300000	0.380952	2	7	0	0	1
7344	I3187	1	1	0	0	0.000000	0.000000	1	6	1	1	0

2067 rows × 12 columns

In [63]:

```
df=pd.concat([df2_train,df1_test])
df=df.drop(['customer_id'], axis=1)
```

In [64]:

```
df = df.reset_index(drop=True)
df
```

Out[64]:

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area
0	1	1	1	0	0.700000	1.000000	4	6	0	0	1
1	1	1	1	1	0.366667	0.380952	2	4	0	1	0
2	1	0	0	0	0.233333	1.000000	4	2	1	1	0
3	0	1	0	0	0.533333	1.000000	4	8	0	0	1
4	1	0	1	0	0.633333	0.000000	1	8	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
6699	1	1	1	0	0.566667	1.000000	4	7	0	1	0
6700	0	0	1	0	0.000000	0.000000	1	7	0	0	1
6701	0	1	1	0	0.766667	0.666667	3	4	0	1	0
6702	0	1	1	0	0.300000	0.380952	2	7	0	0	1
6703	1	1	0	0	0.000000	0.000000	1	6	1	1	0

6704 rows × 11 columns

In [65]:

```
df['survey'] = df['survey'].replace(['No reply'],df['survey'].mode())
```

In [66]:

```
df['dependents']=df['dependents'].astype(int)
df['survey']=df['survey'].astype(int)
```

In [67]:

```
df
```


Out[67]:

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area

	gender	partner	dependents	senior	Tenure	monthly_cost	package	survey	Class	location_Greater London	location_Non london area
0	1	1	1	0	0.700000	1.000000	4	6	0	0	1
1	1	1	1	1	0.366667	0.380952	2	4	0	1	0
2	1	0	0	0	0.233333	1.000000	4	2	1	1	0
3	0	1	0	0	0.533333	1.000000	4	8	0	0	1
4	1	0	1	0	0.633333	0.000000	1	8	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
6699	1	1	1	0	0.566667	1.000000	4	7	0	1	0
6700	0	0	1	0	0.000000	0.000000	1	7	0	0	1
6701	0	1	1	0	0.766667	0.666667	3	4	0	1	0
6702	0	1	1	0	0.300000	0.380952	2	7	0	0	1
6703	1	1	0	0	0.000000	0.000000	1	6	1	1	0

6704 rows × 11 columns

In [68]:

```
import statsmodels.api as sm
x = df.drop(['Class'],axis=1)
y = df['Class']

# Statsmodels.OLS requires us to add a constant.
x = sm.add_constant(x)
model = sm.OLS(y,x)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Class   R-squared:                       0.625
Model:                            OLS   Adj. R-squared:                  0.625
Method:                 Least Squares   F-statistic:                     1241.
Date:                Mon, 08 Aug 2022   Prob (F-statistic):               0.00
Time:                        09:08:07   Log-Likelihood:                -907.01
No. Observations:                6704   AIC:                             1834.
Df Residuals:                    6694   BIC:                             1902.
Df Model:                           9
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                       0.7123      0.041     17.470      0.000       0.632       0.792
gender                      0.0041      0.007      0.609      0.543      -0.009       0.017
partner                    -0.0679      0.008     -8.244      0.000      -0.084      -0.052
dependents                 -0.4766      0.009    -55.949      0.000      -0.493      -0.460
senior                     -0.0825      0.011     -7.633      0.000      -0.104      -0.061
Tenure                     -0.3715      0.017    -21.650      0.000      -0.405      -0.338
monthly_cost               -0.1766      0.190     -0.931      0.352      -0.549       0.195
package                     0.0412      0.063      0.657      0.512      -0.082       0.164
survey                     -0.0620      0.002    -38.377      0.000      -0.065      -0.059
location_Greater London     0.3867      0.021     18.653      0.000       0.346       0.427
location_Non london area    0.3256      0.021     15.744      0.000       0.285       0.366
==============================================================================
Omnibus:                      355.252   Durbin-Watson:                   2.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              831.467
Skew:                           0.328   Prob(JB):                    2.81e-181
Kurtosis:                       4.596   Cond. No.                     4.15e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.41e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [70]:

```
df = df.drop(columns=['gender', 'package'])
```

In [73]:

```
df
```

Out[73]:

	partner	dependents	senior	Tenure	monthly_cost	survey	Class	location_Greater London	location_Non london area
0	1	1	0	0.700000	1.000000	6	0	0	1
1	1	1	1	0.366667	0.380952	4	0	1	0
2	0	0	0	0.233333	1.000000	2	1	1	0
3	1	0	0	0.533333	1.000000	8	0	0	1

	partner	dependents	senior	Tenure	monthly_cost	survey	Class	location_Greater London	location_Non london area
4	0	1	0	0.633333	0.000000	8	0	0	1
...	...	...	...	...	...	...	...	...	...
6699	1	1	0	0.566667	1.000000	7	0	1	0
6700	0	1	0	0.000000	0.000000	7	0	0	1
6701	1	1	0	0.766667	0.666667	4	0	1	0
6702	1	1	0	0.300000	0.380952	7	0	0	1
6703	1	0	0	0.000000	0.000000	6	1	1	0

6704 rows × 9 columns

In [74]:

```
df.dtypes
```

Out[74]:

```
partner                      int64
dependents                   int32
senior                       int64
Tenure                     float64
monthly_cost               float64
survey                       int32
Class                        int32
location_Greater London      int32
location_Non london area     int32
dtype: object
```

In [75]:

```
# df['survey'] = MinMaxScaler().fit_transform(np.array(df['survey']).reshape(-1,1))
# #df['monthly_cost'] = MinMaxScaler().fit_transform(np.array(df['monthly_cost']).reshape(-1,1))
# # view normalized data
# display(df)
```

In [76]:

```
df.dtypes
```

Out[76]:

```
partner                      int64
dependents                   int32
senior                       int64
Tenure                     float64
monthly_cost               float64
survey                       int32
Class                        int32
location_Greater London      int32
location_Non london area     int32
dtype: object
```

In [77]:

```
print(df['Class'].value_counts())
```

```
0    4778
1    1926
Name: Class, dtype: int64
```

In [78]:

```
from sklearn.utils import resample
df_major=df[df.Class==0]
df_minor=df[df.Class==1]

df_minority_upsampled = resample(df_minor, replace=True,n_samples=4778,random_state=30)

df_upsamp_ML=pd.concat([df_major, df_minority_upsampled])
print(df_upsamp_ML['Class'].value_counts())
```

```
0    4778
1    4778
Name: Class, dtype: int64
```

In [79]:

```
df_upsamp_ML
```

Out[79]:

	partner	dependents	senior	Tenure	monthly_cost	survey	Class	location_Greater London	location_Non london area
0	1	1	0	0.700000	1.000000	6	0	0	1
1	1	1	1	0.366667	0.380952	4	0	1	0
3	1	0	0	0.533333	1.000000	8	0	0	1
4	0	1	0	0.633333	0.000000	8	0	0	1
7	1	1	0	0.133333	1.000000	4	0	0	1
...	...	...	...	...	...	...	...	...	...

	partner	dependents	senior	Tenure	monthly_cost	survey	Class	location_Greater London	location_Non london area
2855	0	0	0	0.200000	0.000000	2	1	1	0
3987	0	0	0	0.033333	0.380952	2	1	1	0
1195	0	0	0	0.066667	1.000000	1	1	1	0
5941	0	0	0	0.066667	1.000000	5	1	1	0
3054	1	0	1	0.466667	0.380952	0	1	0	1

9556 rows × 9 columns

In [80]:
```
# x=df_upsamp_ML.drop(['Class','Location_Greater London','Location_Non London area'],axis=1)
x=df_upsamp_ML.drop(['Class'],axis=1)
y=df_upsamp_ML['Class']
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.20,random_state=30)
```

In [81]:
```
x.dtypes
```

Out[81]:
```
partner                      int64
dependents                   int32
senior                       int64
Tenure                     float64
monthly_cost               float64
survey                       int32
location_Greater London      int32
location_Non london area     int32
dtype: object
```

In [82]:
```
modelLogistic = LogisticRegression()
modelLogistic.fit(x_train,y_train)
```

Out[82]:
```
LogisticRegression()
```

In [83]:
```
modelLogistic.coef_
```

Out[83]:
```
array([[-0.84071217, -3.4135005 , -0.31291039, -4.269768  , -0.69463555,
        -0.63884516,  0.38282083, -0.38360583]])
```

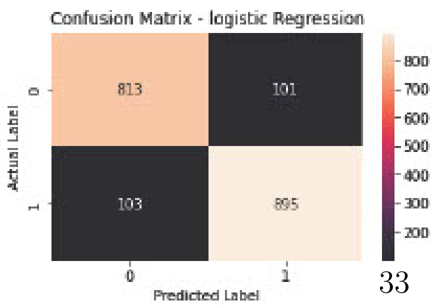
In [84]:
```
y_pred= modelLogistic.predict(x_test)
y_pred
```

Out[84]:
```
array([1, 0, 1, ..., 0, 0, 0])
```

In [85]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
print('Accuracy: %.2f%%' % (accuracy_score(y_test, y_pred) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, y_pred) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, y_pred) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, y_pred) * 100))
confusionmatrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - logistic Regression')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 89.33%
Precision: 89.86%
Recall: 89.68%
F1_Score: 89.77%
```

Out[85]:
```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



In [86]:
```
# import statsmodels.api as sm
# x_train = sm.add_constant(x_train)
# logit_model=sm.Logit(y_train,x_train)
# result=logit_model.fit()
# print(result.summary())
```

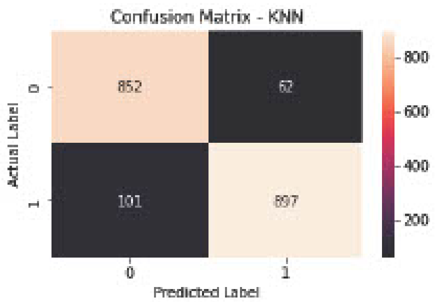
In [87]:
```
modelKNN = KNeighborsClassifier(n_neighbors = 20, metric = 'minkowski', p = 2)
modelKNN.fit(x_train,y_train)
pred_yknn=modelKNN.predict(x_test)
pred_yknn
```

Out[87]:
```
array([1, 0, 1, ..., 0, 0, 0])
```

In [107…
```
print('Accuracy: %.2f%%' % (accuracy_score(y_test, pred_yknn) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, pred_yknn) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, pred_yknn) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, pred_yknn) * 100))
confusionmatrix = confusion_matrix(y_test, pred_yknn)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - KNN')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 91.47%
Precision: 93.53%
Recall: 89.88%
F1_Score: 91.67%
```

Out[107…
```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



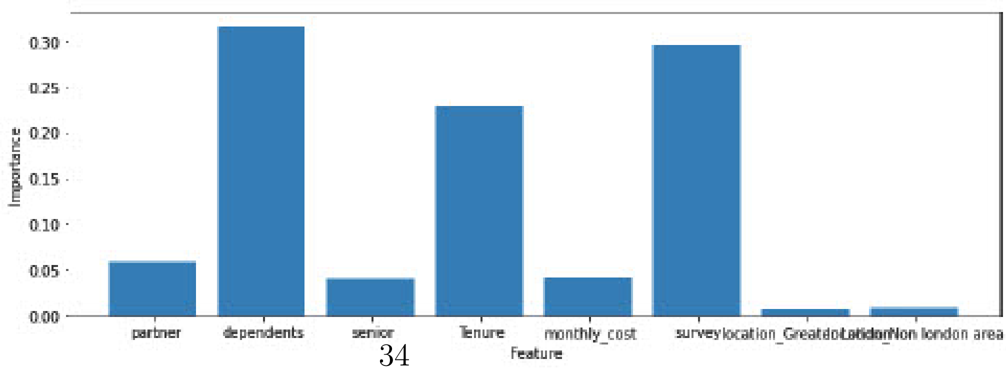
In [89]:
```
from sklearn.ensemble import RandomForestClassifier
modelrf = RandomForestClassifier()
modelrf.fit(x_train, y_train)
pred_Yrf = modelrf.predict(x_test)
```

In [108…
```
modelrf.feature_importances_
```

Out[108…
```
array([0.0593127 , 0.31572086, 0.0416468 , 0.22858003, 0.04276845,
       0.29541171, 0.00765033, 0.00890911])
```

In [113…
```
plt.figure(figsize=(12,4))
plt.bar(x.columns, modelrf.feature_importances_)
plt.xlabel("Feature")
plt.ylabel("Importance")
```

Out[113…
```
Text(0, 0.5, 'Importance')
```



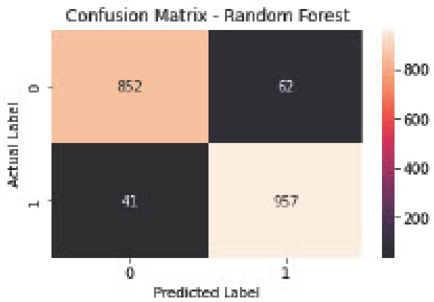
In [100…

```python
print('Accuracy: %.2f%%' % (accuracy_score(y_test, pred_Yrf) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, pred_Yrf) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, pred_Yrf) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, pred_Yrf) * 100))
confusionmatrix = confusion_matrix(y_test, pred_Yrf)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - Random Forest')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 94.61%
Precision: 93.92%
Recall: 95.89%
F1_Score: 94.89%
```

Out[100…

```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



In [91]:

```python
from sklearn.naive_bayes import GaussianNB
modelNB = GaussianNB()
modelNB.fit(x_train, y_train)
pred_YNB = modelNB.predict(x_test)
```

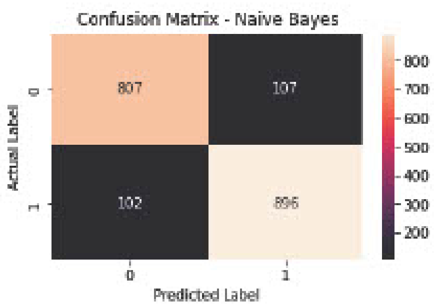
In [104…

```python
print('Accuracy: %.2f%%' % (accuracy_score(y_test, pred_YNB) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, pred_YNB) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, pred_YNB) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, pred_YNB) * 100))
confusionmatrix = confusion_matrix(y_test, pred_YNB)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - Naive Bayes')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 89.07%
Precision: 89.33%
Recall: 89.78%
F1_Score: 89.56%
```

Out[104…

```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



In [93]:

```python
from sklearn.svm import SVC
modelSVC = SVC()
modelSVC.fit(x_train, y_train)
pred_YSVC = modelSVC.predict(x_test)
```

In [105…

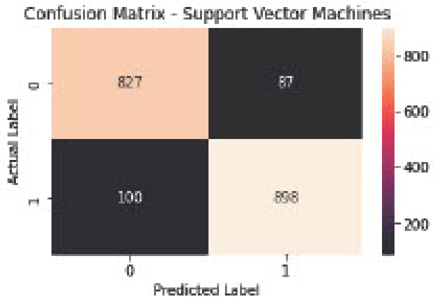
```python
print('Accuracy: %.2f%%' % (accuracy_score(y_test, pred_YSVC) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, pred_YSVC) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, pred_YSVC) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, pred_YSVC) * 100))
```

```
confusionmatrix = confusion_matrix(y_test, pred_YSVC)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - Support Vector Machines')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 90.22%
Precision: 91.17%
Recall: 89.98%
F1_Score: 90.57%
```

Out[105…

```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



In [95]:

```
from sklearn.ensemble import GradientBoostingClassifier
modelGB =GradientBoostingClassifier()
modelGB.fit(x_train, y_train)
pred_GB = modelGB.predict(x_test)
```

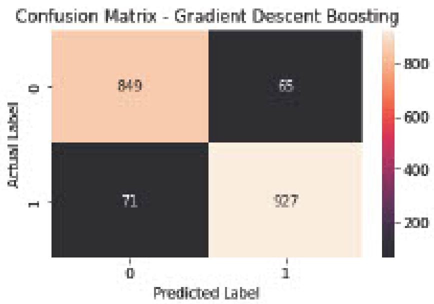
In [106…

```
print('Accuracy: %.2f%%' % (accuracy_score(y_test, pred_GB) * 100 ))
print('Precision: %.2f%%' % (precision_score(y_test, pred_GB) * 100))
print('Recall: %.2f%%' % (recall_score(y_test, pred_GB) * 100))
print('F1_Score: %.2f%%' % (f1_score(y_test, pred_GB) * 100))
confusionmatrix = confusion_matrix(y_test, pred_GB)
plt.figure(figsize=(5,3))
ax = plt.subplot()
sns.heatmap(confusionmatrix, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Label')
ax.set_ylabel('Actual Label')
ax.set_title('Confusion Matrix - Gradient Descent Boosting')
ax.xaxis.set_ticklabels(['0','1'])
ax.yaxis.set_ticklabels(['0','1'])
```

```
Accuracy: 92.89%
Precision: 93.45%
Recall: 92.89%
F1_Score: 93.17%
```

Out[106…

```
[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



In [ ]:

In [ ]: