Telecom Churn Case Study

Problem statement:-

1. To reduce customer churn, telecom companies need to predict which customers are at high risk of churn. In this project, we will analyse customer-level data of a leading telecom firm, build predictive models to identify customers at high risk of churn and identify the main indicators of churn.

2. Retaining high profitable customers is the main business goal here.

Steps

1. Reading, understanding and visualizing the data
2. Preparing the data for modelling
3. Building the model
4. Evaluate the model

Reading, understanding and visualizing the data

Reading and understanding the data

3]:
```
# Reading the dataset
df = pd.read_csv(r'C:\Users\vishnu.kamath\Desktop\ML\telecom_churn_data.csv')
df.head()
```

3]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_of_month_7	last_date_of_month_8	last_date_of
0	7000842753	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
1	7001865778	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
2	7001625959	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
3	7001204172	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
4	7000142493	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	

Handling missing values

Handling missing values ¶

```
df_missing_columns = (round(((df.isnull().sum()/len(df.index))*100),2).to_frame('null')).sort_values('null', ascending=False)
df_missing_columns
```

	null
arpu_3g_6	74.85
night_pck_user_6	74.85
total_rech_data_6	74.85
arpu_2g_6	74.85
max_rech_data_6	74.85
...	...
max_rech_amt_7	0.00
max_rech_amt_6	0.00
total_rech_amt_9	0.00
total_rech_amt_8	0.00
sep_vbc_3g	0.00

Handling missing values

```
[7]: # List the columns having more than 30% missing values
     col_list_missing_30 = list(df_missing_columns.index[df_missing_columns['null'] > 30])

[8]: # Delete the columns having more than 30% missing values
     df = df.drop(col_list_missing_30, axis=1)

[9]: # List the date columns
     date_cols = [k for k in df.columns.to_list() if 'date' in k]
     print(date_cols)

     ['last_date_of_month_6', 'last_date_of_month_7', 'last_date_of_month_8', 'last_date_of_month_9', 'date_of_last_rech_6', 'date_o
     f_last_rech_7', 'date_of_last_rech_8', 'date_of_last_rech_9']

10]: # Dropping date columns
     df = df.drop(date_cols, axis=1)

11]: # Drop circle_id column
     df = df.drop('circle_id', axis=1)
```

Filter high value customers

Filter high-value customers ¶

```
df['avg_rech_amt_6_7'] = (df['total_rech_amt_6'] + df['total_rech_amt_7'])/2
```

```
X = df['avg_rech_amt_6_7'].quantile(0.7)
X
```

```
368.5
```

```
df = df[df['avg_rech_amt_6_7'] >= X]
df.head()
```

	mobile_number	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_mou_8	onne
7	7000701601	0.0	0.0	0.0	1069.180	1349.850	3171.480	500.000	57.84	54.68	52.29	
8	7001524846	0.0	0.0	0.0	378.721	492.223	137.362	166.787	413.69	351.03	35.08	
13	7002191713	0.0	0.0	0.0	492.846	205.671	593.260	322.732	501.76	108.39	534.24	
16	7000875565	0.0	0.0	0.0	430.975	299.869	187.894	206.490	50.51	74.01	70.61	
17	7000187447	0.0	0.0	0.0	690.008	18.980	25.499	257.583	1185.91	9.28	7.79	

Filter high value customers

Filter high-value customers ¶

```
df['avg_rech_amt_6_7'] = (df['total_rech_amt_6'] + df['total_rech_amt_7'])/2
```

```
X = df['avg_rech_amt_6_7'].quantile(0.7)
X
```

```
368.5
```

```
df = df[df['avg_rech_amt_6_7'] >= X]
df.head()
```

	mobile_number	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_mou_8	onne
7	7000701601	0.0	0.0	0.0	1069.180	1349.850	3171.480	500.000	57.84	54.68	52.29	
8	7001524846	0.0	0.0	0.0	378.721	492.223	137.362	166.787	413.69	351.03	35.08	
13	7002191713	0.0	0.0	0.0	492.846	205.671	593.260	322.732	501.76	108.39	534.24	
16	7000875565	0.0	0.0	0.0	430.975	299.869	187.894	206.490	50.51	74.01	70.61	
17	7000187447	0.0	0.0	0.0	690.008	18.980	25.499	257.583	1185.91	9.28	7.79	

Handling missing values in rows

Handling missing values in rows

```
]: # Count the rows having more than 50% missing values
   df_missing_rows_50 = df[(df.isnull().sum(axis=1)) > (len(df.columns)//2)]
   df_missing_rows_50.shape
```

```
]: (114, 178)
```

```
]: # Deleting the rows having more than 50% missing values
   df = df.drop(df_missing_rows_50.index)
   df.shape
```

```
]: (29897, 178)
```

```
]: # Checking the missing values in columns again
   df_missing_columns = (round(((df.isnull().sum()/len(df.index))*100),2).to_frame('null')).sort_values('null', ascending=False)
   df_missing_columns
```

```
]:
```

null

Dataframe creation for MoU

```
# Listing the columns of MOU Sep(9)
print(((df_missing_columns[df_missing_columns['null'] == 5.32]).index).to_list())
```

```
['loc_ic_mou_9', 'og_others_9', 'loc_og_t2t_mou_9', 'loc_ic_t2t_mou_9', 'loc_og_t2m_mou_9', 'loc_og_t2f_mou_9', 'loc_og_t2c_mou
_9', 'std_ic_t2m_mou_9', 'loc_og_mou_9', 'std_og_t2t_mou_9', 'roam_og_mou_9', 'std_ic_t2o_mou_9', 'std_og_t2m_mou_9', 'std_og_t
2f_mou_9', 'spl_og_mou_9', 'std_og_t2c_mou_9', 'std_og_mou_9', 'isd_og_mou_9', 'std_ic_t2t_mou_9', 'std_ic_mou_9', 'onnet_mou_
9', 'spl_ic_mou_9', 'ic_others_9', 'isd_ic_mou_9', 'loc_ic_t2f_mou_9', 'offnet_mou_9', 'loc_ic_t2m_mou_9', 'std_ic_t2f_mou_9',
'roam_ic_mou_9']
```

```
# Creating a dataframe with the condition, in which MOU for Sep(9) are null
df_null_mou_9 = df[(df['loc_og_t2m_mou_9'].isnull()) & (df['loc_ic_t2f_mou_9'].isnull()) & (df['roam_og_mou_9'].isnull()) & (df['
  (df['loc_og_t2t_mou_9'].isnull()) & (df['std_ic_t2t_mou_9'].isnull()) & (df['loc_og_t2f_mou_9'].isnull()) & (df['loc_ic_mou_9']
  (df['loc_og_t2c_mou_9'].isnull()) & (df['loc_og_mou_9'].isnull()) & (df['std_og_t2t_mou_9'].isnull()) & (df['roam_ic_mou_9'].is
  (df['loc_ic_t2m_mou_9'].isnull()) & (df['std_og_t2m_mou_9'].isnull()) & (df['loc_ic_t2t_mou_9'].isnull()) & (df['std_og_t2f_mou
  (df['std_og_t2c_mou_9'].isnull()) & (df['og_others_9'].isnull()) & (df['std_og_mou_9'].isnull()) & (df['spl_og_mou_9'].isnull()
  (df['std_ic_t2f_mou_9'].isnull()) & (df['isd_og_mou_9'].isnull()) & (df['std_ic_mou_9'].isnull()) & (df['offnet_mou_9'].isnull(
  (df['isd_ic_mou_9'].isnull()) & (df['ic_others_9'].isnull()) & (df['std_ic_t2o_mou_9'].isnull()) & (df['onnet_mou_9'].isnull())
  (df['spl_ic_mou_9'].isnull())]

df_null_mou_9.head()
```

Dataframe creation for MoU

```
# Listing the columns of MOU Sep(9)
print(((df_missing_columns[df_missing_columns['null'] == 5.32]).index).to_list())
```

```
['loc_ic_mou_9', 'og_others_9', 'loc_og_t2t_mou_9', 'loc_ic_t2t_mou_9', 'loc_og_t2m_mou_9', 'loc_og_t2f_mou_9', 'loc_og_t2c_mou
_9', 'std_ic_t2m_mou_9', 'loc_og_mou_9', 'std_og_t2t_mou_9', 'roam_og_mou_9', 'std_ic_t2o_mou_9', 'std_og_t2m_mou_9', 'std_og_t
2f_mou_9', 'spl_og_mou_9', 'std_og_t2c_mou_9', 'std_og_mou_9', 'isd_og_mou_9', 'std_ic_t2t_mou_9', 'std_ic_mou_9', 'onnet_mou_
9', 'spl_ic_mou_9', 'ic_others_9', 'isd_ic_mou_9', 'loc_ic_t2f_mou_9', 'offnet_mou_9', 'loc_ic_t2m_mou_9', 'std_ic_t2f_mou_9',
'roam_ic_mou_9']
```

```
# Creating a dataframe with the condition, in which MOU for Sep(9) are null
df_null_mou_9 = df[(df['loc_og_t2m_mou_9'].isnull()) & (df['loc_ic_t2f_mou_9'].isnull()) & (df['roam_og_mou_9'].isnull()) & (df['
  (df['loc_og_t2t_mou_9'].isnull()) & (df['std_ic_t2t_mou_9'].isnull()) & (df['loc_og_t2f_mou_9'].isnull()) & (df['loc_ic_mou_9']
  (df['loc_og_t2c_mou_9'].isnull()) & (df['loc_og_mou_9'].isnull()) & (df['std_og_t2t_mou_9'].isnull()) & (df['roam_ic_mou_9'].is
  (df['loc_ic_t2m_mou_9'].isnull()) & (df['std_og_t2m_mou_9'].isnull()) & (df['loc_ic_t2t_mou_9'].isnull()) & (df['std_og_t2f_mou
  (df['std_og_t2c_mou_9'].isnull()) & (df['og_others_9'].isnull()) & (df['std_og_mou_9'].isnull()) & (df['spl_og_mou_9'].isnull()
  (df['std_ic_t2f_mou_9'].isnull()) & (df['isd_og_mou_9'].isnull()) & (df['std_ic_mou_9'].isnull()) & (df['offnet_mou_9'].isnull(
  (df['isd_ic_mou_9'].isnull()) & (df['ic_others_9'].isnull()) & (df['std_ic_t2o_mou_9'].isnull()) & (df['onnet_mou_9'].isnull()
  (df['spl_ic_mou_9'].isnull())]

df_null_mou_9.head()
```

Checking percentage for missing values

In [20]:
```
# Deleting the records for which MOU for Sep(9) are null
df = df.drop(df_null_mou_9.index)
```

In [21]:
```
# Again Cheking percent of missing values in columns
df_missing_columns = (round(((df.isnull().sum()/len(df.index))*100),2).to_frame('null')).sort_values('null', ascending=False)
df_missing_columns
```

Out[21]:

	null
isd_og_mou_8	0.55
roam_ic_mou_8	0.55
loc_og_mou_8	0.55
std_ic_t2o_mou_8	0.55
roam_og_mou_8	0.55
...	...
total_og_mou_9	0.00
total_og_mou_8	0.00
total_og_mou_7	0.00
total_og_mou_6	0.00
avg_rech_amt_6_7	0.00

178 rows × 1 columns

Tag churners

Tag churners

```
5]: df['churn'] = np.where((df['total_ic_mou_9']==0) & (df['total_og_mou_9']==0) & (df['vol_2g_mb_9']==0) & (df['vol_3g_mb_9']==0), 1
```

```
6]: # List the columns for churn month(9)
col_9 = [col for col in df.columns.to_list() if '_9' in col]
print(col_9)
```

```
['arpu_9', 'onnet_mou_9', 'offnet_mou_9', 'roam_ic_mou_9', 'roam_og_mou_9', 'loc_og_t2t_mou_9', 'loc_og_t2m_mou_9', 'loc_og_t2f
_mou_9', 'loc_og_t2c_mou_9', 'loc_og_mou_9', 'std_og_t2t_mou_9', 'std_og_t2m_mou_9', 'std_og_t2f_mou_9', 'std_og_t2c_mou_9', 's
td_og_mou_9', 'isd_og_mou_9', 'spl_og_mou_9', 'og_others_9', 'total_og_mou_9', 'loc_ic_t2t_mou_9', 'loc_ic_t2m_mou_9', 'loc_ic_
t2f_mou_9', 'loc_ic_mou_9', 'std_ic_t2t_mou_9', 'std_ic_t2m_mou_9', 'std_ic_t2f_mou_9', 'std_ic_t2o_mou_9', 'std_ic_mou_9', 'to
tal_ic_mou_9', 'spl_ic_mou_9', 'isd_ic_mou_9', 'ic_others_9', 'total_rech_num_9', 'total_rech_amt_9', 'max_rech_amt_9', 'last_d
ay_rch_amt_9', 'vol_2g_mb_9', 'vol_3g_mb_9', 'monthly_2g_9', 'sachet_2g_9', 'monthly_3g_9', 'sachet_3g_9']
```

```
7]: df = df.drop(col_9, axis=1)
```

```
8]: df = df.drop('sep_vbc_3g', axis=1)
```

```
9]: round(100*(df['churn'].mean()),2)
```

Checking Outliers

Outliers treatment ¶

```
df['mobile_number'] = df['mobile_number'].astype(object)
df['churn'] = df['churn'].astype(object)
```

```
# List only the numeric columns
numeric_cols = df.select_dtypes(exclude=['object']).columns
print(numeric_cols)
```

```
Index(['loc_og_t2o_mou', 'std_og_t2o_mou', 'loc_ic_t2o_mou', 'arpu_6',
       'arpu_7', 'arpu_8', 'onnet_mou_6', 'onnet_mou_7', 'onnet_mou_8',
       'offnet_mou_6',
       ...
       'monthly_3g_7', 'monthly_3g_8', 'sachet_3g_6', 'sachet_3g_7',
       'sachet_3g_8', 'aon', 'aug_vbc_3g', 'jul_vbc_3g', 'jun_vbc_3g',
       'avg_rech_amt_6_7'],
      dtype='object', length=134)
```

```
# Removing outliers below 10th and above 90th percentile
for col in numeric_cols:
    q1 = df[col].quantile(0.10)
    q3 = df[col].quantile(0.90)
    iqr = q3-q1
    range_low  = q1-1.5*iqr
    range_high = q3+1.5*iqr
    # Assigning the filtered dataset into data
    data = df.loc[(df[col] > range_low) & (df[col] < range_high)]

data.shape
```

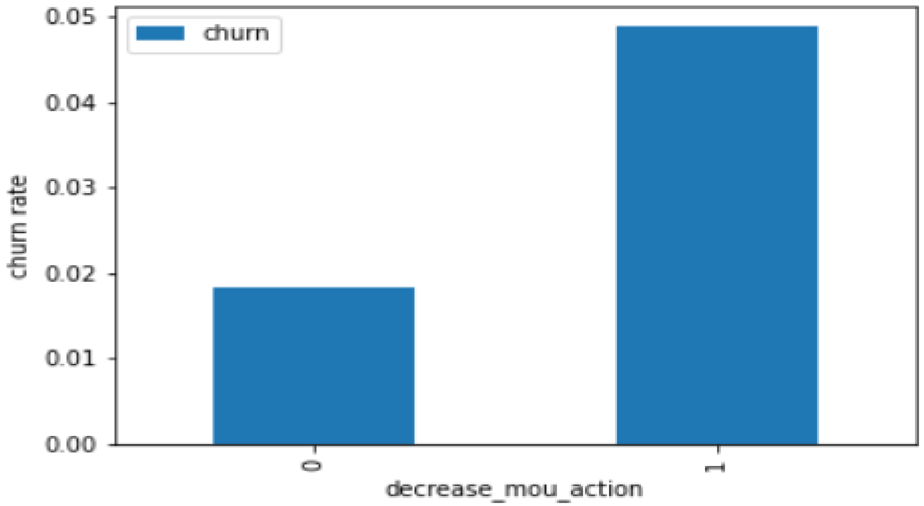
```
(27705, 136)
```

EDA

EDA

```
# Converting churn column to int in order to do aggfunc in the pivot table
data['churn'] = data['churn'].astype('int64')
```

```
data.pivot_table(values='churn', index='decrease_mou_action', aggfunc='mean').plot.bar()
plt.ylabel('churn rate')
plt.show()
```




```
data.pivot_table(values='churn', index='decrease_rech_num_action', aggfunc='mean').plot.bar()
plt.ylabel('churn rate')
plt.show()
```

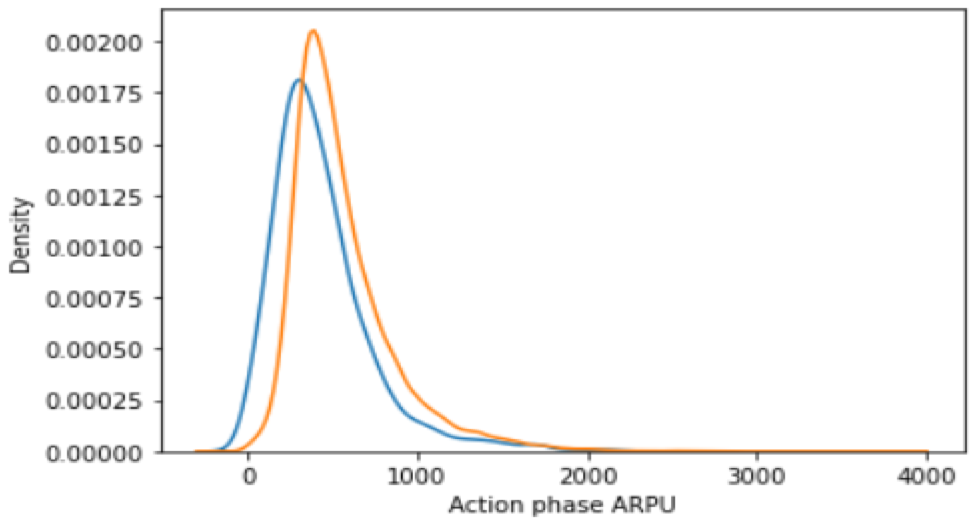
Analysis

Analysis

[68]:
```
# Creating churn dataframe
data_churn = data[data['churn'] == 1]
# Creating not churn dataframe
data_non_churn = data[data['churn'] == 0]
```

[69]:
```
# Distribution plot
ax = sns.distplot(data_churn['avg_arpu_action'],label='churn',hist=False)
ax = sns.distplot(data_non_churn['avg_arpu_action'],label='not churn',hist=False)
ax.set(xlabel='Action phase ARPU')
```

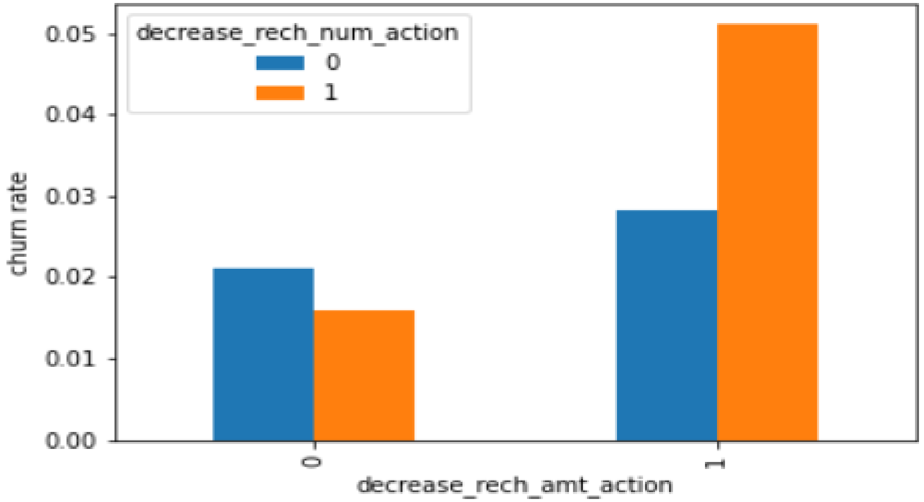
[69]: [Text(0.5, 0, 'Action phase ARPU')]



Bivariate Analysis

Bivariate analysis

71]:
```
data.pivot_table(values='churn', index='decrease_rech_amt_action', columns='decrease_rech_num_action', aggfunc='mean').plot.bar()
plt.ylabel('churn rate')
plt.show()
```



72]:
```
data.pivot_table(values='churn', index='decrease_rech_amt_action', columns='decrease_vbc_action', aggfunc='mean').plot.bar()
plt.ylabel('churn rate')
plt.show()
```

Train-Test

Train-Test Split

7]:

```
from sklearn.model_selection import train_test_split
# Putting feature variables into X
X = data.drop(['mobile_number','churn'], axis=1)
# Putting target variable to y
y = data['churn']
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)
```

Regression

Logistic regression with PCA ¶

```
# Importing scikit logistic regression module
from sklearn.linear_model import LogisticRegression
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

```
# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as recall as we are more focused on acheiving the higher sensitivity than the accuracy
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'recall',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_pca, y_train)
```

Model Accuracy

.00]:
```python
# Prediction on the test set
y_test_pred = log_pca_model.predict(X_test_pca)
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[5335   13]
 [ 175   18]]
```

.01]:
```python
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.9660711062985021
Sensitivity:- 0.09326424870466321
Specificity:- 0.9975691847419597
```

Model Accuracy

```
.00]: # Prediction on the test set
y_test_pred = log_pca_model.predict(X_test_pca)
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[5335   13]
 [ 175   18]]
```

```
.01]: TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.9660711062985021
Sensitivity:- 0.09326424870466321
Specificity:- 0.9975691847419597
```

Conclusion

After trying we can see that for acheiving the best sensitivity, which was our ultimate goal, the classic Logistic regression. For both the models the sensitivity was approx 81%. Also we have good accuracy of apporx 85%.