Importing Libraries

In [9]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the Data for ML Project

In [10]:

```
df=pd.read_csv('TelcoChurn.csv')
```

In [11]:

```
df.shape
```

Out[11]:

```
(7043, 21)
```

In [12]:

```
df.head()
```

Out[12]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	In
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

In [13]:

```
df.dtypes
```

Out[13]:

```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
```

```
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                object
dtype: object
```

Setting Display options to ensure feature name visibility

In [14]:
```
pd.set_option('display.max_columns',None)
```

Warning Suppression

In [15]:
```
import warnings
warnings.filterwarnings('ignore')
```

How many rows have missing ID ?

In [16]:
```
df['customerID'].isnull().sum()
```

Out[16]:
```
0
```

Drop ID Feature from the dataset

In [17]:
```
df=df.drop(['customerID'],axis=1)
```

Label the Churn feature to 1/0

In [18]:
```
df['Churn'].value_counts()
```

Out[18]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [19]:
```
df['target']=np.where(df['Churn']=="Yes",1,0)
```

Drop the Churn feature to retain only Target

In [20]:
```
df=df.drop(['Churn'],axis=1)
```

Defining Target and Independent Features

In [21]:
```
Y=df[['target']]
X=df.drop(['target'],axis=1)
```

Get the Churn Rate

In [22]:
```
Y.mean()
```

Out[22]:
```
target    0.26537
dtype: float64
```

Split features into Numerical and Categorical

In [23]:
```
num=X.select_dtypes(include="number")
char=X.select_dtypes(include="object")
```

In [24]:
```
num.head()
```

Out[24]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
0	0	1	29.85	29.85
1	0	34	56.95	1889.50
2	0	2	53.85	108.15
3	0	45	42.30	1840.75
4	0	2	70.70	151.65

In [25]:
```
#Check whether SeniorCitizon feaure is an indicator
num.SeniorCitizen.value_counts()
```

Out[25]:
```
0    5901
1    1142
Name: SeniorCitizen, dtype: int64
```

In [26]:
```
char.head()
```

Out[26]:

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	Onlin
0	Female	Yes	No	No	No phone service	DSL	No	
1	Male	No	No	Yes	No	DSL	Yes	
2	Male	No	No	Yes	No	DSL	Yes	
3	Male	No	No	No	No phone	DSL	Yes	

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	Onlin
					service			
4	Female	No	No	Yes	No	Fiber optic	No	

Dropping the indicator features from num to build a separate DF

In [27]:
```
ind=num[['SeniorCitizen']]
num=num.drop(['SeniorCitizen'],axis=1)
```

Outlier Analysis of Numerical Features

In [28]:
```
num.describe(percentiles=[0.01,0.05,0.10,0.25,0.50,0.75,0.85,0.9,0.99])
```

Out[28]:

	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	2279.798992
std	24.559481	30.090047	2266.730170
min	0.000000	18.250000	18.800000
1%	1.000000	19.200000	19.871000
5%	1.000000	19.650000	49.070000
10%	2.000000	20.050000	83.470000
25%	9.000000	35.500000	398.550000
50%	29.000000	70.350000	1394.550000
75%	55.000000	89.850000	3786.600000
85%	65.000000	98.550000	5195.485000
90%	69.000000	102.600000	5973.690000
99%	72.000000	114.729000	8039.256000
max	72.000000	118.750000	8684.800000

Capping and Flooring of outliers

In [29]:
```
def outlier_cap(x):
    x=x.clip(lower=x.quantile(0.01))
    x=x.clip(upper=x.quantile(0.99))
    return(x)
```

In [30]:
```
num=num.apply(lambda x : outlier_cap(x))
```

In [31]:
```
num.describe(percentiles=[0.01,0.05,0.10,0.25,0.50,0.75,0.85,0.9,0.99])
```

Out[31]:

	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000
mean	32.372710	64.749689	2277.243407
std	24.557454	30.062810	2260.002318
min	1.000000	19.200000	19.871000
1%	1.000000	19.200000	19.883180
5%	1.000000	19.650000	49.070000
10%	2.000000	20.050000	83.470000
25%	9.000000	35.500000	398.550000
50%	29.000000	70.350000	1394.550000
75%	55.000000	89.850000	3786.600000
85%	65.000000	98.550000	5195.485000
90%	69.000000	102.600000	5973.690000
99%	72.000000	114.716820	8037.867480
max	72.000000	114.729000	8039.256000

Missing Value Analysis

In [32]:
```
num.isnull().mean()
```

Out[32]:
```
tenure            0.0
MonthlyCharges    0.0
TotalCharges      0.0
dtype: float64
```

In [33]:
```
# Since the data does not contain any missing values Imputation Processes are not re
```

Feature Selection - Numerical Features

Part 1 : Remove Features with 0 Variance

In [34]:
```
from sklearn.feature_selection import VarianceThreshold

varselector= VarianceThreshold(threshold=0)
varselector.fit_transform(num)
# Get columns to keep and create new dataframe with those only
cols = varselector.get_support(indices=True)
num_1 = num.iloc[:,cols]
```

In [35]:
```
num_1.iloc[0]
```

Out[35]:
```
tenure             1.00
MonthlyCharges    29.85
TotalCharges      29.85
Name: 0, dtype: float64
```

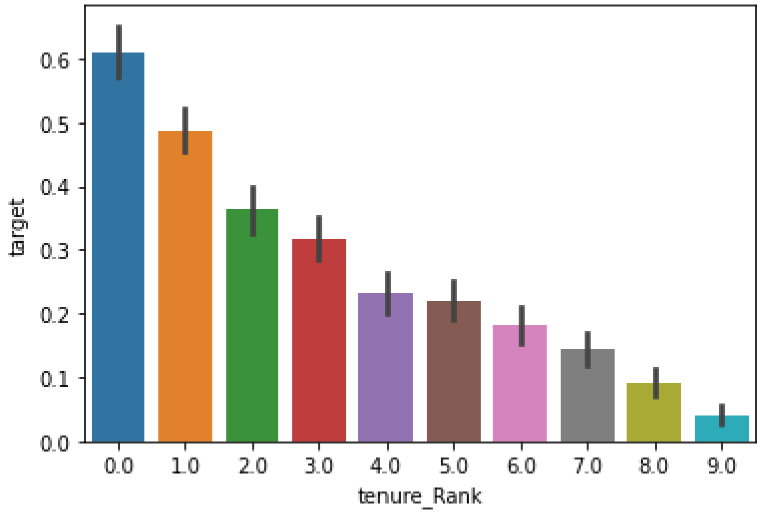
Part 2 - Bi Variate Analysis (Feature Discretization)

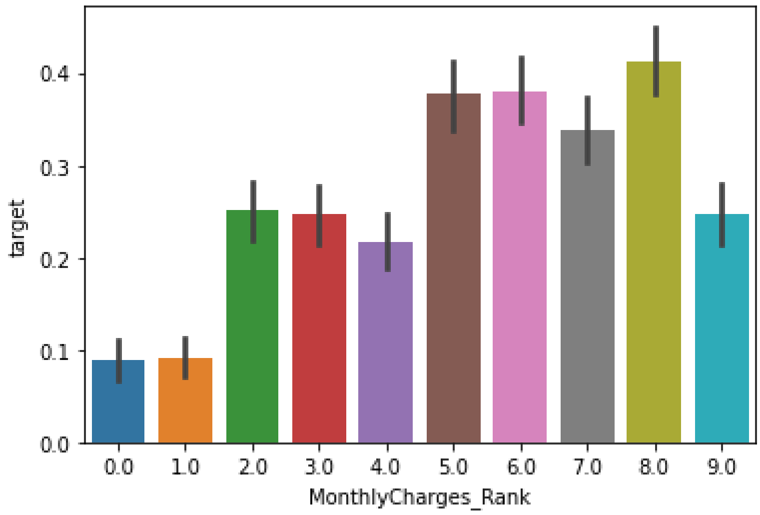
In [36]:
```
from sklearn.preprocessing import KBinsDiscretizer
discrete=KBinsDiscretizer(n_bins=10,encode='ordinal', strategy='quantile')
num_binned=pd.DataFrame(discrete.fit_transform(num_1),index=num_1.index, columns=num
num_binned.head()
```

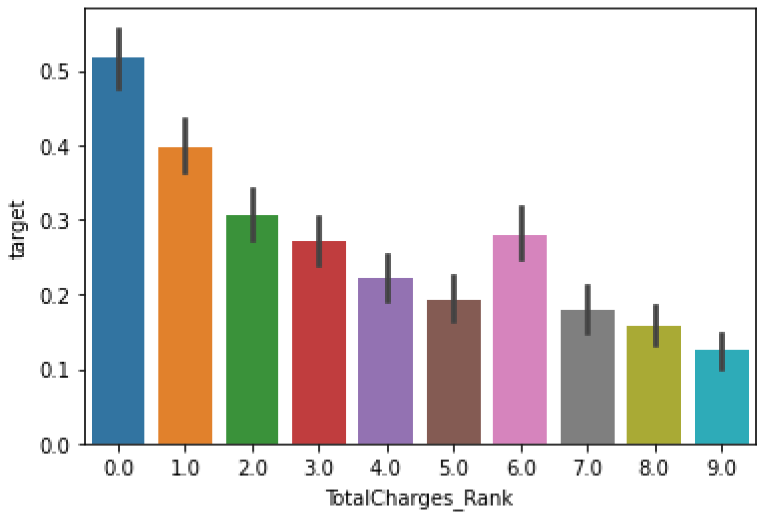
Out[36]:

	tenure_Rank	MonthlyCharges_Rank	TotalCharges_Rank
0	0.0	2.0	0.0
1	5.0	3.0	5.0
2	1.0	3.0	1.0
3	6.0	2.0	5.0
4	1.0	5.0	1.0

In [37]:
```
#Check if the features show a slope at all
#If they do, then do you see some deciles below the population average and some high
#If that is the case then the slope will be strong
#Conclusion: A strong slope is indicative of the features' ability to discriminate t
#            making it a good predictor

#percentage_income_goesinto_intallments=Insallment/annual_inc (Derived Variables/Fea

X_bin_combined=pd.concat([Y,num_binned],axis=1,join='inner')

from numpy import mean
for col in (num_binned.columns):
    plt.figure()
    sns.barplot(x=col, y="target",data=X_bin_combined, estimator=mean )
plt.show()
```







In [38]:

```
# All features from num_2 will get selected due to good discrimination
select_features_df_num=num_1
```

In [39]:

```
num_1.shape
```

Out[39]: (7043, 3)

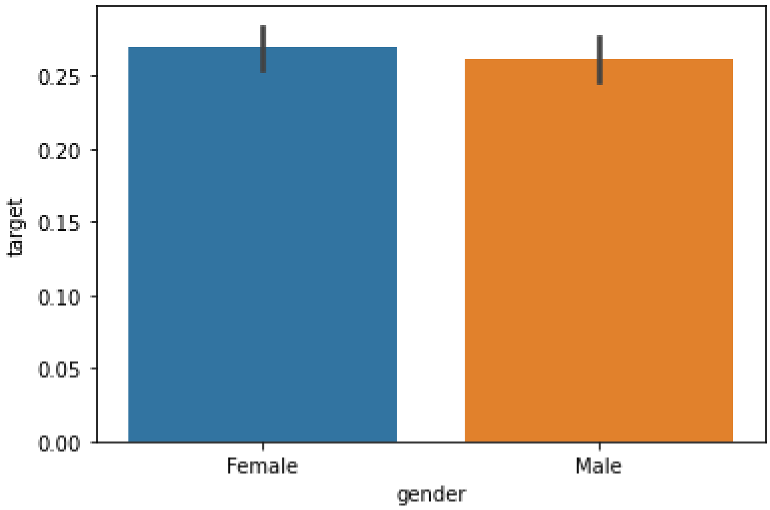
Feature Selection - Categorical Features

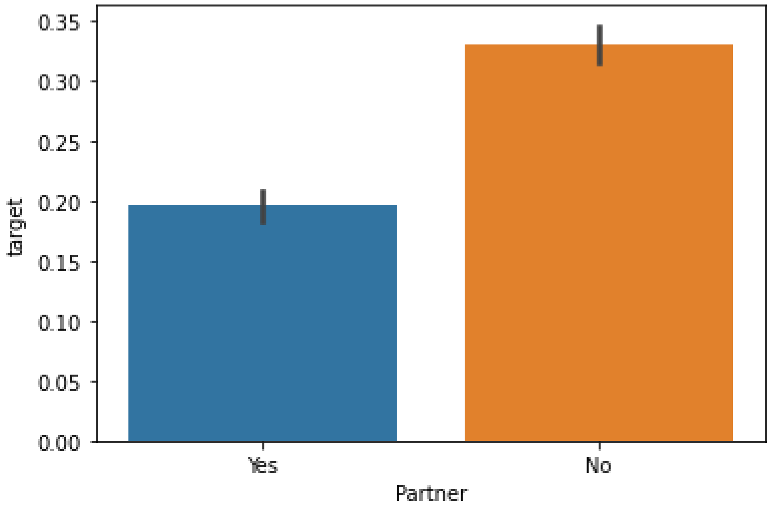
In [40]:
```
char.dtypes
```

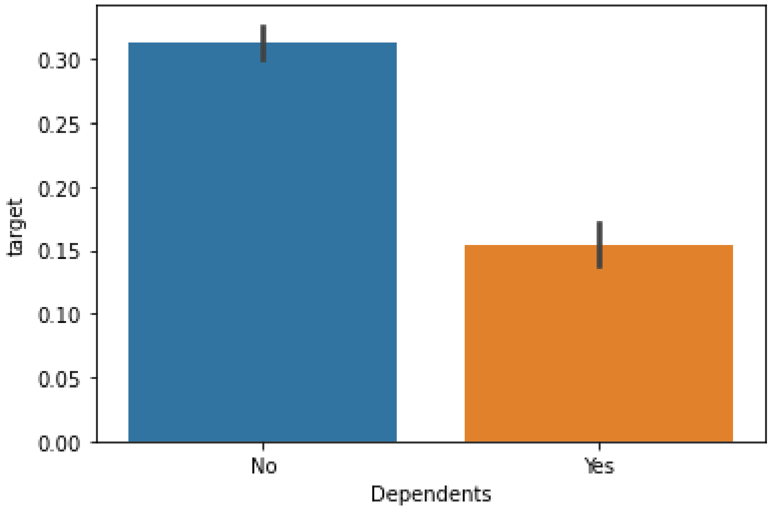
Out[40]:
```
gender              object
Partner             object
Dependents          object
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
dtype: object
```

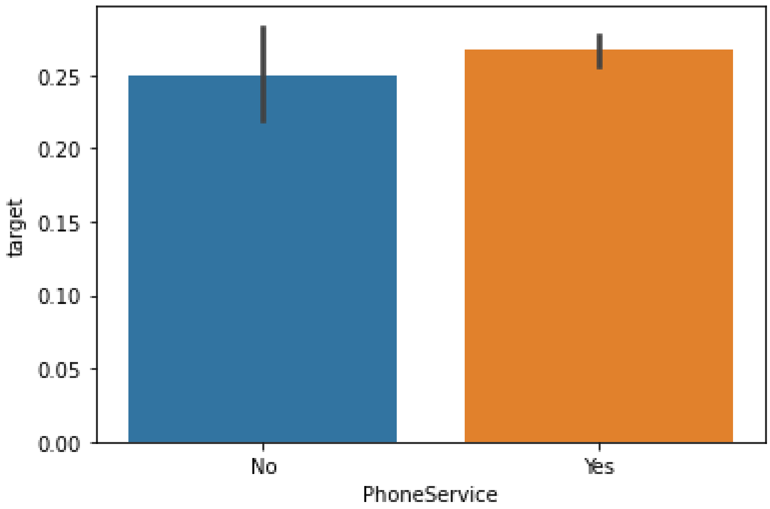
Part 1 - Bi Variate Analysis

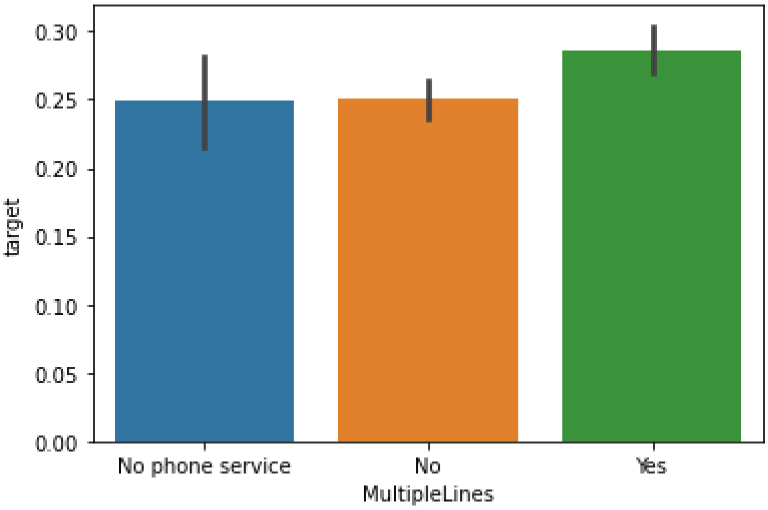
In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns
X_char_merged=pd.concat([Y,char],axis=1,join='inner')

from numpy import mean
for col in (char.columns):
    plt.figure()
    sns.barplot(x=col, y="target",data=X_char_merged, estimator=mean )
plt.show()
```



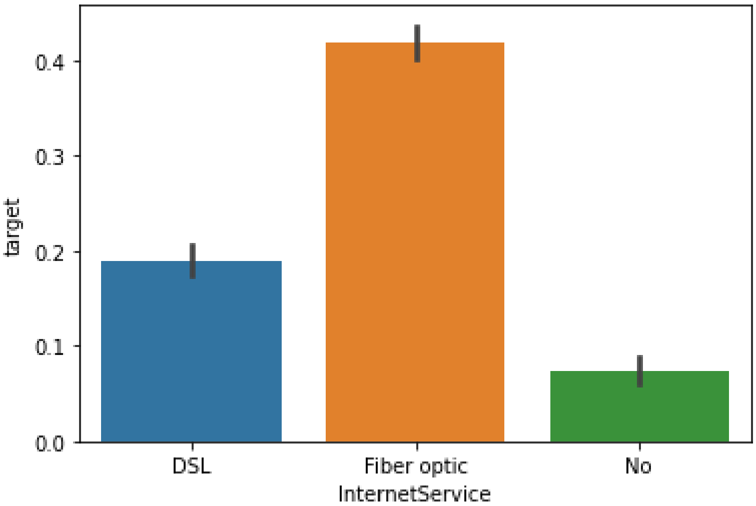

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
target
Yes
No
Partner


0.30
0.25
0.20
0.15
0.10
0.05
0.00
target
No
Yes
Dependents

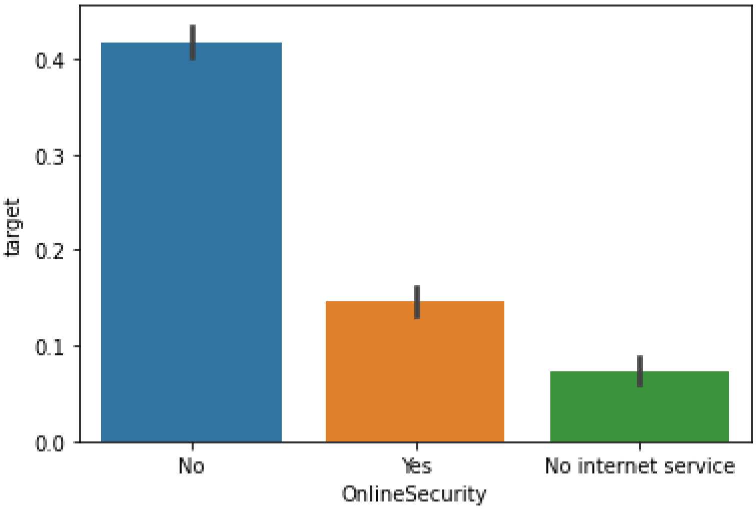

0.25
0.20
0.15
0.10
0.05
0.00
target
No
Yes
PhoneService



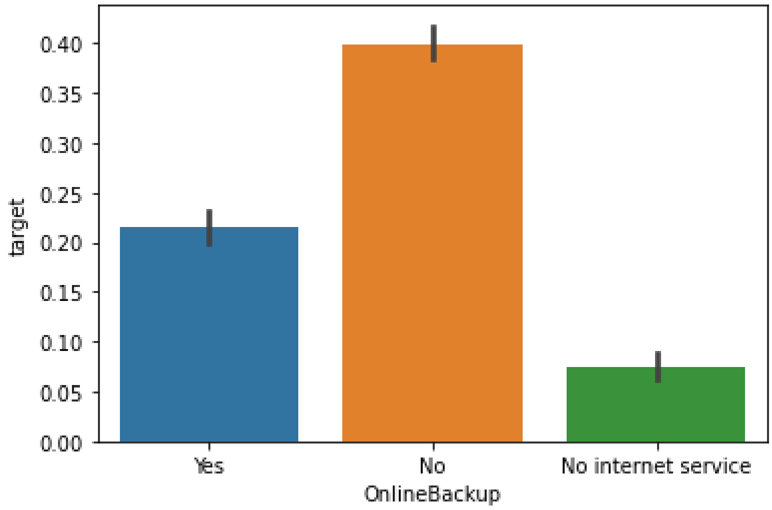
0.30
0.25
0.20
0.15
0.10
0.05
0.00
target
No phone service
No
Yes
MultipleLines

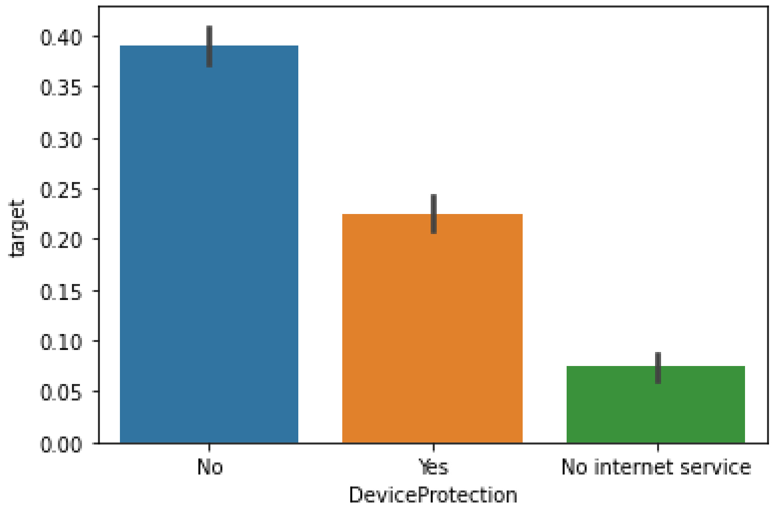


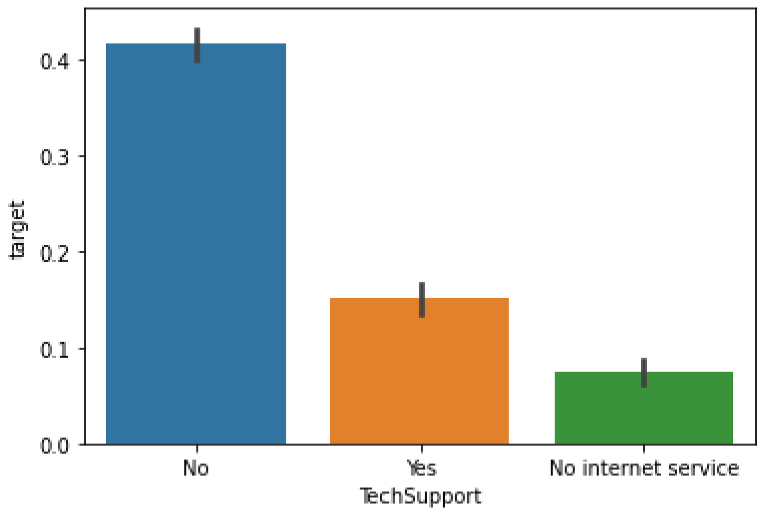
0.4
0.3
0.2
0.1
0.0
target
DSL
Fiber optic
No
InternetService

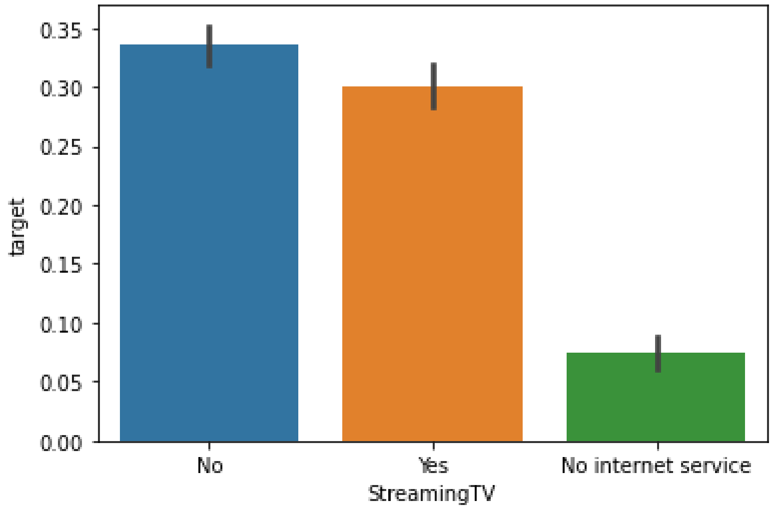


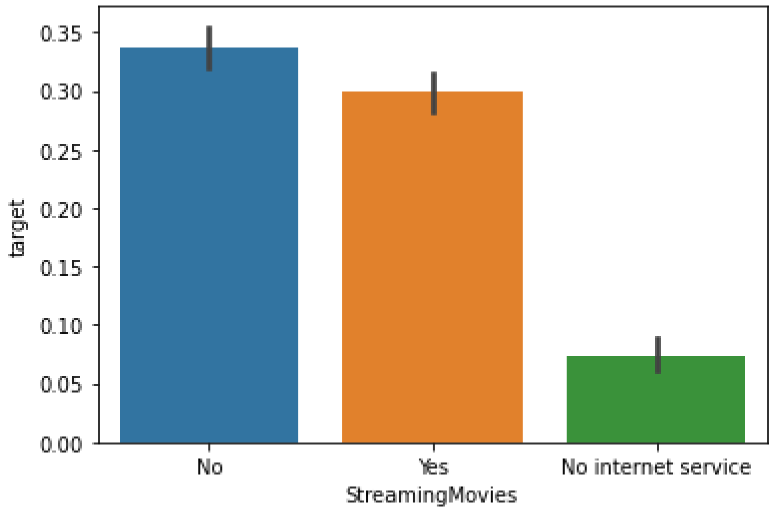
0.4
0.3
0.2
0.1
0.0
target
No
Yes
No internet service
OnlineSecurity

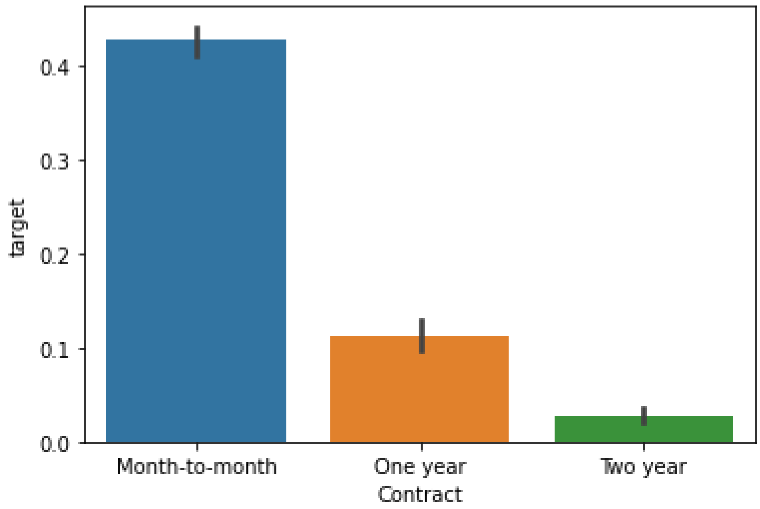

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
target
Yes
No
No internet service
OnlineBackup

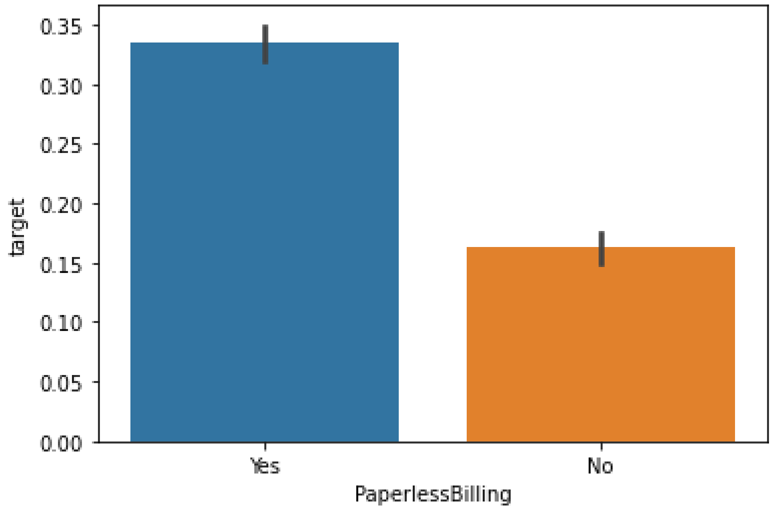

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
target
No
Yes
No internet service
DeviceProtection

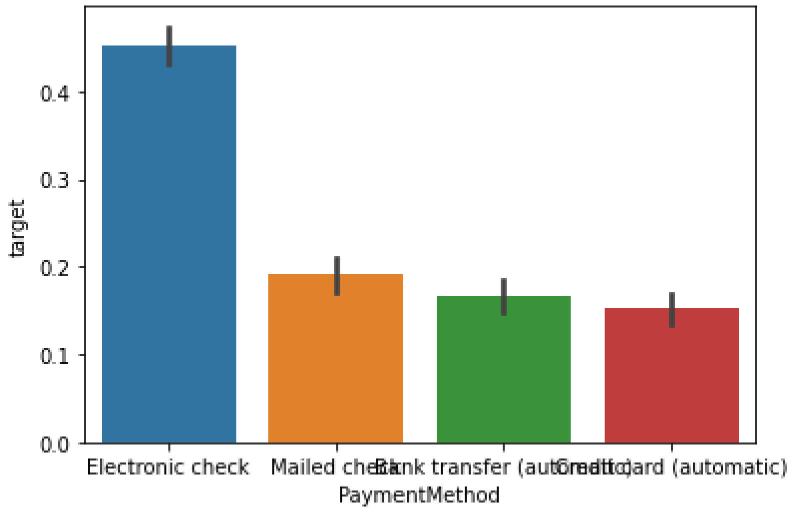

0.4
0.3
0.2
0.1
0.0
target
No
Yes
No internet service
TechSupport


target
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
No
Yes
No internet service
StreamingTV


target
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
No
Yes
No internet service
StreamingMovies


target
0.4
0.3
0.2
0.1
0.0
Month-to-month
One year
Two year
Contract





In [42]:

```
char=char.drop(['gender','PhoneService','MultipleLines'],axis=1)
```

In [43]:

```
# Create dummy features with n-1 levels
X_char_dum = pd.get_dummies(char, drop_first = True)
X_char_dum.shape
```

Out[43]:

```
(7043, 22)
```

Part 2 - Select K Best

In [44]:

```
# Select K Best for Categorical Features
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(chi2, k=20)
selector.fit_transform(X_char_dum, Y)
# Get columns to keep and create new dataframe with those only
cols = selector.get_support(indices=True)
select_features_df_char = X_char_dum.iloc[:,cols]
```

In [45]:

```
select_features_df_char.iloc[0]
```

Out[45]:

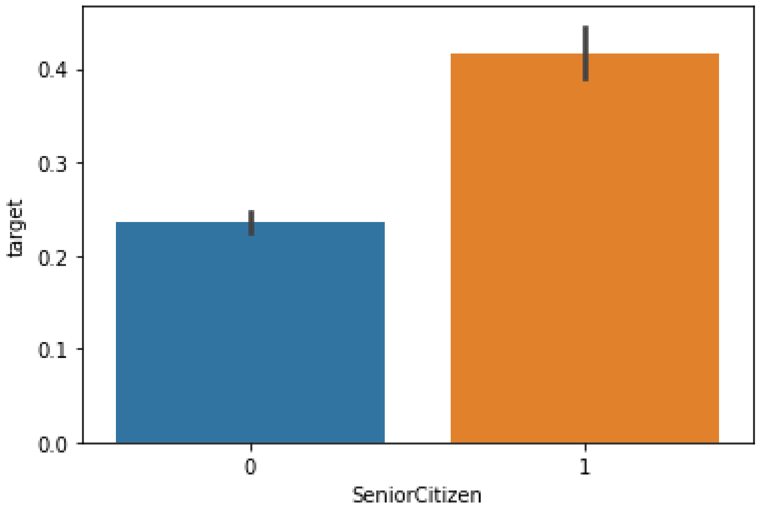
```
Partner_Yes                          1
Dependents_Yes                       0
InternetService_Fiber optic          0
InternetService_No                   0
```

```
OnlineSecurity_No internet service       0
OnlineSecurity_Yes                       0
OnlineBackup_No internet service         0
OnlineBackup_Yes                         1
DeviceProtection_No internet service     0
DeviceProtection_Yes                     0
TechSupport_No internet service          0
TechSupport_Yes                          0
StreamingTV_No internet service          0
StreamingMovies_No internet service      0
Contract_One year                        0
Contract_Two year                        0
PaperlessBilling_Yes                     1
PaymentMethod_Credit card (automatic)    0
PaymentMethod_Electronic check           1
PaymentMethod_Mailed check               0
Name: 0, dtype: uint8
```

Feature Selection - Numerical Indicator Features

In [46]:

```
X_ind_merged=pd.concat([Y,ind],axis=1,join='inner')
from numpy import mean
for col in (ind.columns):
    plt.figure()
    sns.barplot(x=col, y="target",data=X_ind_merged, estimator=mean )
plt.show()
```



In [47]:

```
select_features_df_ind=ind
```

Creating the Master Feature Set for Model Development

In [48]:

```
X_all=pd.concat([select_features_df_char,select_features_df_num,select_features_df_i
```

In [49]:

```
Y['target'].value_counts()
```

Out[49]:
```
0    5174
1    1869
Name: target, dtype: int64
```

Train Test Split

In [50]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X_all, Y, test_size=0.3, random_st
```

In [51]:
```
print("Shape of Training Data",X_train.shape)
print("Shape of Testing Data",X_test.shape)
print("Response Rate in Training Data",y_train.mean())
print("Response Rate in Testing Data",y_test.mean())
```

```
Shape of Training Data (4930, 24)
Shape of Testing Data (2113, 24)
Response Rate in Training Data target    0.266126
dtype: float64
Response Rate in Testing Data target    0.263606
dtype: float64
```

In [52]:
```
# Non Linearity in feature relationships are observed which makes tree methods a goo
# There are few options to consider among tree methods
# White Box (Completely Explainable Set of Rules) - Decision Tree
# Ensemble Methods - Random Forest (With Bagging)
# Ensemble Methods - GBM/XGBoost (Boosting)
```

In [53]:
```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression(random_state=0)
logreg.fit(X_train,y_train)
```

Out[53]:
```
LogisticRegression
LogisticRegression(random_state=0)
```

In [54]:
```
coeff_df=pd.DataFrame(X_all.columns)
coeff_df.columns=['features']
coeff_df["Coefficient Estimate"] = pd.Series(logreg.coef_[0])
coeff_df
```

Out[54]:

	features	Coefficient Estimate
0	Partner_Yes	0.010674
1	Dependents_Yes	-0.095012
2	InternetService_Fiber optic	0.598124
3	InternetService_No	-0.173241
4	OnlineSecurity_No internet service	-0.173241
5	OnlineSecurity_Yes	-0.616837
6	OnlineBackup_No internet service	-0.173241
7	OnlineBackup_Yes	-0.299833

	features	Coefficient Estimate
8	DeviceProtection_No internet service	-0.173241
9	DeviceProtection_Yes	-0.019946
10	TechSupport_No internet service	-0.173241
11	TechSupport_Yes	-0.518968
12	StreamingTV_No internet service	-0.173241
13	StreamingMovies_No internet service	-0.173241
14	Contract_One year	-0.458669
15	Contract_Two year	-0.639223
16	PaperlessBilling_Yes	0.489748
17	PaymentMethod_Credit card (automatic)	-0.152683
18	PaymentMethod_Electronic check	0.247514
19	PaymentMethod_Mailed check	-0.215688
20	tenure	-0.055983
21	MonthlyCharges	0.003205
22	TotalCharges	0.000290
23	SeniorCitizen	0.218403

In [55]:

```
# Building a Decision Tree Model
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier(criterion='gini',random_state=0)
```

In [56]:

```
np.random.seed(44)
from sklearn.model_selection import GridSearchCV
param_dist = {'max_depth': [3, 5, 6, 7], 'min_samples_split': [50, 100, 150, 200, 25
tree_grid = GridSearchCV(dtree, cv = 10, param_grid=param_dist,n_jobs = 3)
tree_grid.fit(X_train,y_train)
print('Best Parameters using grid search: \n', tree_grid.best_params_)
```

```
Best Parameters using grid search:
 {'max_depth': 6, 'min_samples_split': 50}
```

In [57]:

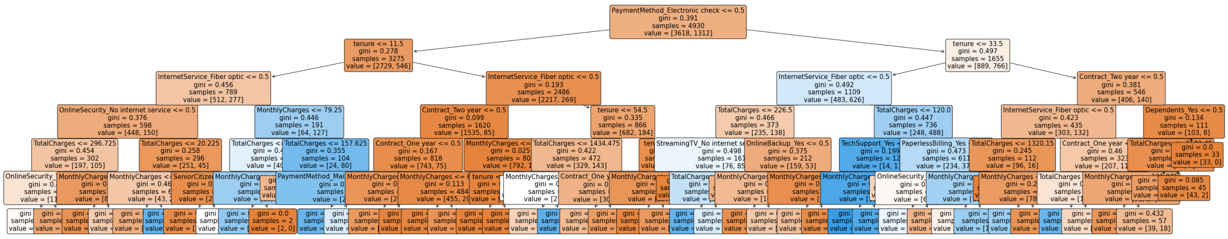
```
dtree=DecisionTreeClassifier(criterion='gini',random_state=0,max_depth=6,min_samples
dtree.fit(X_train,y_train)
```

Out[57]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, min_samples_split=50, random_state=0)
```

In [58]:

```
from sklearn import tree
import pydotplus
import matplotlib.pyplot as plt
plt.figure(figsize=[50,10])
tree.plot_tree(dtree,filled=True,fontsize=15,rounded=True,feature_names=X_all.column
plt.show()
```



In [59]:

```
# Building a Random Forest Model
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='gini',random_state=0,max_depth=6,min_samples_sp
rf.fit(X_train,y_train)
```

Out[59]:

```
RandomForestClassifier
RandomForestClassifier(max_depth=6, min_samples_split=50, random_state=0)
```

In [60]:

```
import pandas as pd
feature_importances=pd.DataFrame(rf.feature_importances_,
                                 index=X_train.columns,
                                 columns=['importance']).sort_values('importance',as
feature_importances
```

Out[60]:

	importance
tenure	0.200408
TotalCharges	0.149898
InternetService_Fiber optic	0.115766
PaymentMethod_Electronic check	0.086082
MonthlyCharges	0.084691
Contract_Two year	0.076559
OnlineSecurity_Yes	0.038781
Contract_One year	0.035319
InternetService_No	0.028913
TechSupport_Yes	0.027326
TechSupport_No internet service	0.024787
OnlineSecurity_No internet service	0.018912
StreamingMovies_No internet service	0.016281
StreamingTV_No internet service	0.013703
PaperlessBilling_Yes	0.012808
OnlineBackup_No internet service	0.012688
OnlineBackup_Yes	0.011998
DeviceProtection_No internet service	0.011318
SeniorCitizen	0.006703
PaymentMethod_Mailed check	0.006276
Partner_Yes	0.005536

	importance
Dependents_Yes	0.005457
DeviceProtection_Yes	0.004954
PaymentMethod_Credit card (automatic)	0.004836

In [61]:

```
# Building a Gradient Boosting Model
from sklearn.ensemble import GradientBoostingClassifier
gbm=GradientBoostingClassifier(criterion='mse',random_state=0,max_depth=6,min_sample
gbm.fit(X_train,y_train)
```

Out[61]:

```
GradientBoostingClassifier
GradientBoostingClassifier(criterion='mse', max_depth=6, min_samples_split=
50,
                           random_state=0)
```

In [62]:

```
import pandas as pd
feature_importances=pd.DataFrame(gbm.feature_importances_,
                                 index=X_train.columns,
                                 columns=['importance']).sort_values('importance',as
feature_importances
```

Out[62]:

	importance
tenure	0.207958
TotalCharges	0.186127
MonthlyCharges	0.166350
PaymentMethod_Electronic check	0.137275
InternetService_Fiber optic	0.133854
Contract_Two year	0.036155
Contract_One year	0.033409
PaperlessBilling_Yes	0.019680
OnlineSecurity_Yes	0.014681
TechSupport_Yes	0.013139
SeniorCitizen	0.011829
OnlineBackup_Yes	0.007532
PaymentMethod_Credit card (automatic)	0.005456
Partner_Yes	0.005341
OnlineBackup_No internet service	0.003851
PaymentMethod_Mailed check	0.002851
Dependents_Yes	0.002808
DeviceProtection_Yes	0.002678
OnlineSecurity_No internet service	0.002631

	importance
TechSupport_No internet service	0.002463
DeviceProtection_No internet service	0.001938
StreamingTV_No internet service	0.001536
InternetService_No	0.000367
StreamingMovies_No internet service	0.000092

In [63]:

```
base_learners = [
                        ('rf', RandomForestClassifier(criterion='gini',random_state=
                        ('gbm', GradientBoostingClassifier(criterion='mse',random_st
                       ]
```

In [64]:

```
from sklearn.ensemble import StackingClassifier
clf = StackingClassifier(estimators=base_learners, final_estimator=LogisticRegressio
```

In [65]:

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

Out[65]:

```
StackingClassifier
rf                          gbm
RandomForestClassifier      GradientBoostingClassifier
final_estimator
LogisticRegression
```

In [66]:

```
# Model Evaluation
y_pred_logreg=logreg.predict(X_test)
y_pred_tree=dtree.predict(X_test)
y_pred_rf=rf.predict(X_test)
y_pred_gbm=gbm.predict(X_test)
y_pred_stacking=clf.predict(X_test)
```

In [67]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

In [68]:

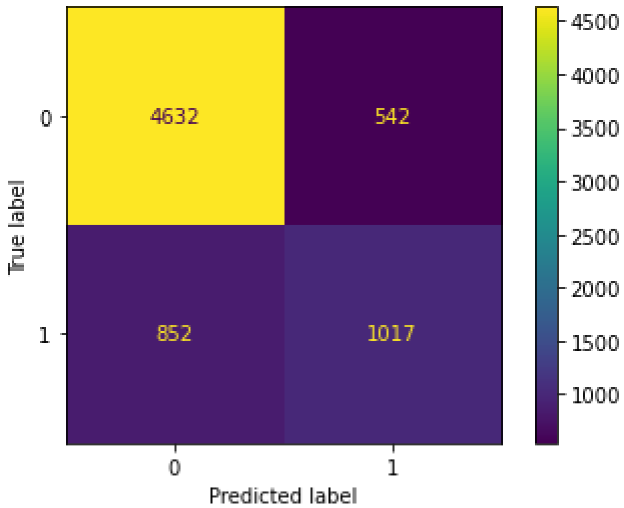
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_logreg))
print("Precision",metrics.precision_score(y_test,y_pred_logreg))
print("Recall",metrics.recall_score(y_test,y_pred_logreg))
print("f1_score",metrics.f1_score(y_test,y_pred_logreg))
```

```
Accuracy: 0.79649787032655
Precision 0.6365591397849463
Recall 0.5314183123877917
f1_score 0.5792563600782779
```

In [69]:

```
metrics.plot_confusion_matrix(logreg,X_all,Y)
```

Out[69]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23caaadcb50>



In [70]:

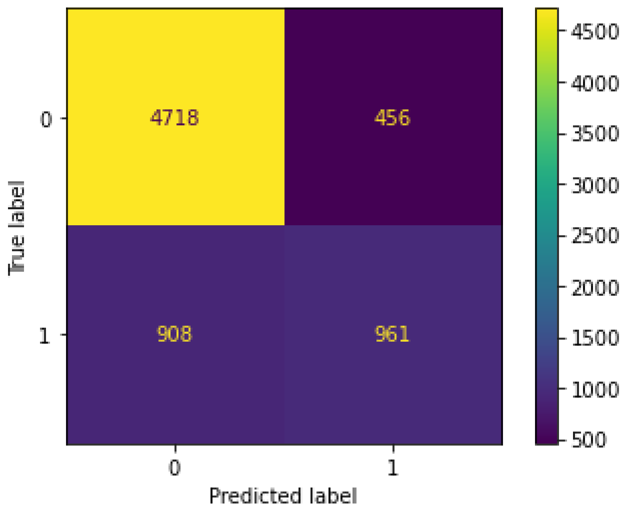
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_tree))
print("Precision",metrics.precision_score(y_test,y_pred_tree))
print("Recall",metrics.recall_score(y_test,y_pred_tree))
print("f1_score",metrics.f1_score(y_test,y_pred_tree))
```

```
Accuracy: 0.7950780880265026
Precision 0.6455399061032864
Recall 0.49371633752244165
f1_score 0.5595116988809766
```

In [71]:

```
metrics.plot_confusion_matrix(dtree,X_all,Y)
```

Out[71]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23caa15df10>



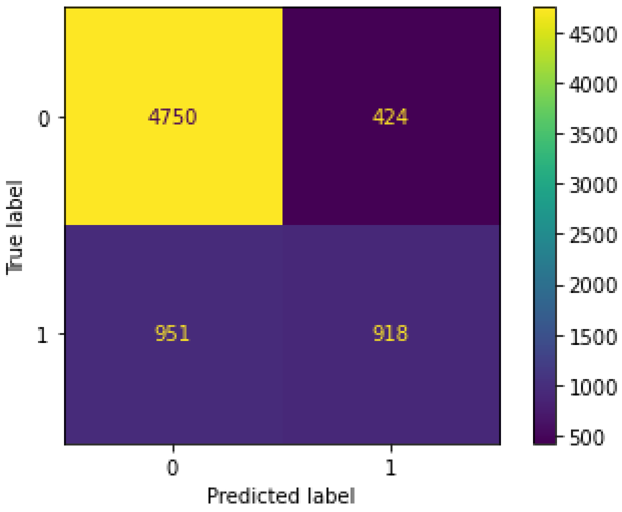
In [72]:

```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_rf))
print("Precision",metrics.precision_score(y_test,y_pred_rf))
print("Recall",metrics.recall_score(y_test,y_pred_rf))
print("f1_score",metrics.f1_score(y_test,y_pred_rf))
```

```
Accuracy: 0.7983909133932797
Precision 0.6641604010025063
Recall 0.4757630161579892
f1_score 0.5543933054393305
```

In [73]:
```
metrics.plot_confusion_matrix(rf,X_all,Y)
```

Out[73]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23caac3a700>

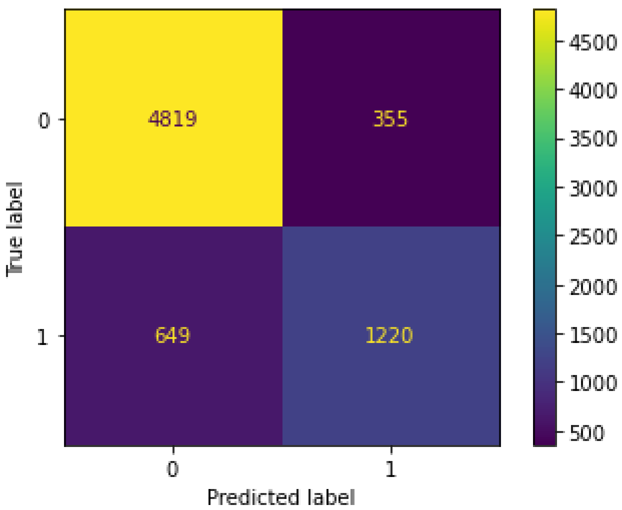


In [74]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_gbm))
print("Precision",metrics.precision_score(y_test,y_pred_gbm))
print("Recall",metrics.recall_score(y_test,y_pred_gbm))
print("f1_score",metrics.f1_score(y_test,y_pred_gbm))
```

```
Accuracy: 0.7974443918599148
Precision 0.6423841059602649
Recall 0.5224416517055656
f1_score 0.5762376237623762
```

In [75]:
```
metrics.plot_confusion_matrix(gbm,X_all,Y)
```

Out[75]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23caad4d9d0>



In [76]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_stacking))
print("Precision",metrics.precision_score(y_test,y_pred_stacking))
print("Recall",metrics.recall_score(y_test,y_pred_stacking))
print("f1_score",metrics.f1_score(y_test,y_pred_stacking))
```

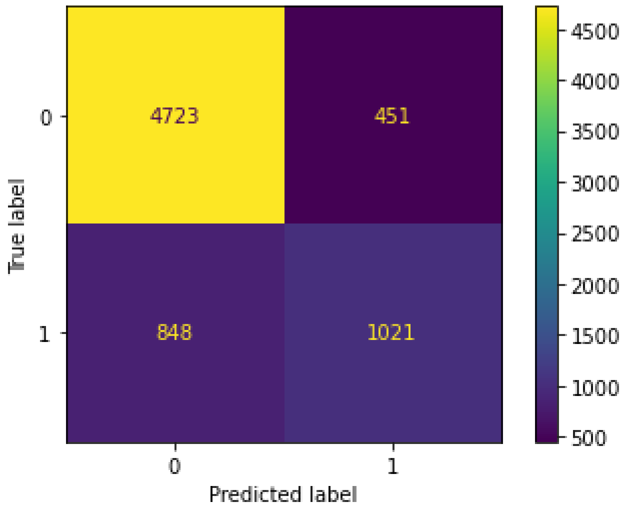
```
Accuracy: 0.8007572172266919
Precision 0.6552511415525114
Recall 0.5152603231597845
f1_score 0.5768844221105527
```

In [77]:

```
metrics.plot_confusion_matrix(clf,X_all,Y)
```

Out[77]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23caac3a5e0>
```



In [78]:

```
# Lorenz Curve
```

In [79]:

```
# Decsion Tree Lorenz Curve
```

In [80]:

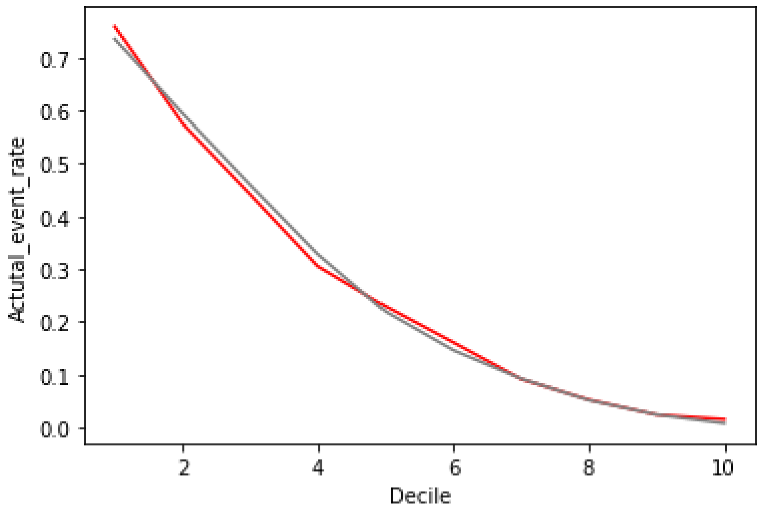
```
y_pred_prob = logreg.predict_proba(X_all)[:, 1]
df['pred_prob_logreg']=pd.DataFrame(y_pred_prob)
df['P_Rank_logreg']=pd.qcut(df['pred_prob_logreg'].rank(method='first').values,10,du
rank_df_actuals=df.groupby('P_Rank_logreg')['target'].agg(['count','mean'])
rank_df_predicted=df.groupby('P_Rank_logreg')['pred_prob_logreg'].agg(['mean'])
rank_df_actuals=pd.DataFrame(rank_df_actuals)

rank_df_actuals.rename(columns={'mean':'Actutal_event_rate'},inplace=True)
rank_df_predicted=pd.DataFrame(rank_df_predicted)
rank_df_predicted.rename(columns={'mean':'Predicted_event_rate'},inplace=True)
rank_df=pd.concat([rank_df_actuals,rank_df_predicted],axis=1,join="inner")

sorted_rank_df=rank_df.sort_values(by='P_Rank_logreg',ascending=False)
sorted_rank_df['N_events']=rank_df['count']*rank_df['Actutal_event_rate']
sorted_rank_df['cum_events']=sorted_rank_df['N_events'].cumsum()
sorted_rank_df['event_cap']=sorted_rank_df['N_events']/max(sorted_rank_df['N_events'
sorted_rank_df['cum_event_cap']=sorted_rank_df['event_cap'].cumsum()

sorted_rank_df['N_non_events']=sorted_rank_df['count']-sorted_rank_df['N_events']
sorted_rank_df['cum_non_events']=sorted_rank_df['N_non_events'].cumsum()
sorted_rank_df['non_event_cap']=sorted_rank_df['N_non_events']/max(sorted_rank_df['N
sorted_rank_df['cum_non_event_cap']=sorted_rank_df['non_event_cap'].cumsum()

sorted_rank_df['KS']=round((sorted_rank_df['cum_event_cap']-sorted_rank_df['cum_non_

sorted_rank_df['random_cap']=sorted_rank_df['count']/max(sorted_rank_df['count'].cum
sorted_rank_df['cum_random_cap']=sorted_rank_df['random_cap'].cumsum()
sorted_reindexed=sorted_rank_df.reset_index()
sorted_reindexed['Decile']=sorted_reindexed.index+1
sorted_reindexed
```

Out[80]:

	P_Rank_logreg	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_cap
0	10	705	0.758865	0.734978	535.0	535.0	0.286249
1	9	704	0.575284	0.594199	405.0	940.0	0.216693
2	8	704	0.441761	0.459874	311.0	1251.0	0.166399
3	7	704	0.305398	0.328381	215.0	1466.0	0.115035
4	6	704	0.228693	0.218736	161.0	1627.0	0.086142
5	5	705	0.160284	0.145551	113.0	1740.0	0.060460
6	4	704	0.090909	0.091718	64.0	1804.0	0.034243
7	3	704	0.052557	0.051707	37.0	1841.0	0.019797
8	2	704	0.024148	0.024314	17.0	1858.0	0.009096
9	1	705	0.015603	0.007843	11.0	1869.0	0.005886

In [81]:

```
ax = sns.lineplot( x="Decile", y="Actutal_event_rate", data=sorted_reindexed,color='
ax = sns.lineplot( x="Decile", y="Predicted_event_rate", data=sorted_reindexed,color
```



In [82]:

```
y_pred_prob = dtree.predict_proba(X_all)[:, 1]
df['pred_prob_dtree']=pd.DataFrame(y_pred_prob)
df['P_Rank_tree']=pd.qcut(df['pred_prob_dtree'].rank(method='first').values,10,dupli
rank_df_actuals=df.groupby('P_Rank_tree')['target'].agg(['count','mean'])
rank_df_predicted=df.groupby('P_Rank_tree')['pred_prob_dtree'].agg(['mean'])
rank_df_actuals=pd.DataFrame(rank_df_actuals)

rank_df_actuals.rename(columns={'mean':'Actutal_event_rate'},inplace=True)
rank_df_predicted=pd.DataFrame(rank_df_predicted)
rank_df_predicted.rename(columns={'mean':'Predicted_event_rate'},inplace=True)
rank_df=pd.concat([rank_df_actuals,rank_df_predicted],axis=1,join="inner")

sorted_rank_df=rank_df.sort_values(by='P_Rank_tree',ascending=False)
sorted_rank_df['N_events']=rank_df['count']*rank_df['Actutal_event_rate']
sorted_rank_df['cum_events']=sorted_rank_df['N_events'].cumsum()
sorted_rank_df['event_cap']=sorted_rank_df['N_events']/max(sorted_rank_df['N_events'
sorted_rank_df['cum_event_cap']=sorted_rank_df['event_cap'].cumsum()

sorted_rank_df['N_non_events']=sorted_rank_df['count']-sorted_rank_df['N_events']
sorted_rank_df['cum_non_events']=sorted_rank_df['N_non_events'].cumsum()
```

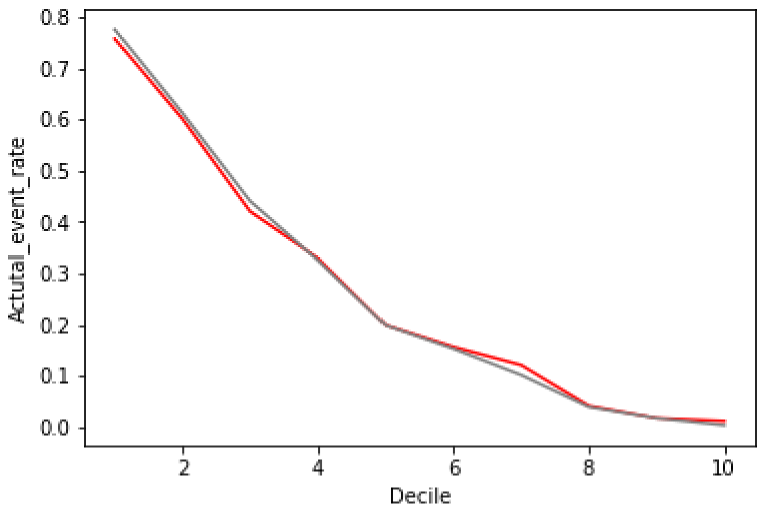
```
sorted_rank_df['non_event_cap']=sorted_rank_df['N_non_events']/max(sorted_rank_df['N
sorted_rank_df['cum_non_event_cap']=sorted_rank_df['non_event_cap'].cumsum()

sorted_rank_df['KS']=round((sorted_rank_df['cum_event_cap']-sorted_rank_df['cum_non_

sorted_rank_df['random_cap']=sorted_rank_df['count']/max(sorted_rank_df['count'].cum
sorted_rank_df['cum_random_cap']=sorted_rank_df['random_cap'].cumsum()
sorted_reindexed=sorted_rank_df.reset_index()
sorted_reindexed['Decile']=sorted_reindexed.index+1
sorted_reindexed
```

Out[82]:

	P_Rank_tree	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_cap	cu
0	10	705	0.757447	0.775697	534.0	534.0	0.285714	
1	9	704	0.600852	0.612564	423.0	957.0	0.226324	
2	8	704	0.420455	0.440659	296.0	1253.0	0.158373	
3	7	704	0.329545	0.325437	232.0	1485.0	0.124131	
4	6	704	0.198864	0.198429	140.0	1625.0	0.074906	
5	5	705	0.156028	0.152310	110.0	1735.0	0.058855	
6	4	704	0.120739	0.101395	85.0	1820.0	0.045479	
7	3	704	0.041193	0.039016	29.0	1849.0	0.015516	
8	2	704	0.017045	0.016893	12.0	1861.0	0.006421	
9	1	705	0.011348	0.003096	8.0	1869.0	0.004280	

In [83]:

```
ax = sns.lineplot( x="Decile", y="Actutal_event_rate", data=sorted_reindexed,color='
ax = sns.lineplot( x="Decile", y="Predicted_event_rate", data=sorted_reindexed,color
```



In [84]:

```
# Random Forest Lorenz Curve
```

In [85]:

```
y_pred_prob = rf.predict_proba(X_all)[:, 1]
df['pred_prob_rf']=pd.DataFrame(y_pred_prob)
df['P_Rank_rf']=pd.qcut(df['pred_prob_rf'].rank(method='first').values,10,duplicates
rank_df_actuals=df.groupby('P_Rank_rf')['target'].agg(['count','mean'])
rank_df_predicted=df.groupby('P_Rank_rf')['pred_prob_rf'].agg(['mean'])
rank_df_actuals=pd.DataFrame(rank_df_actuals)
```

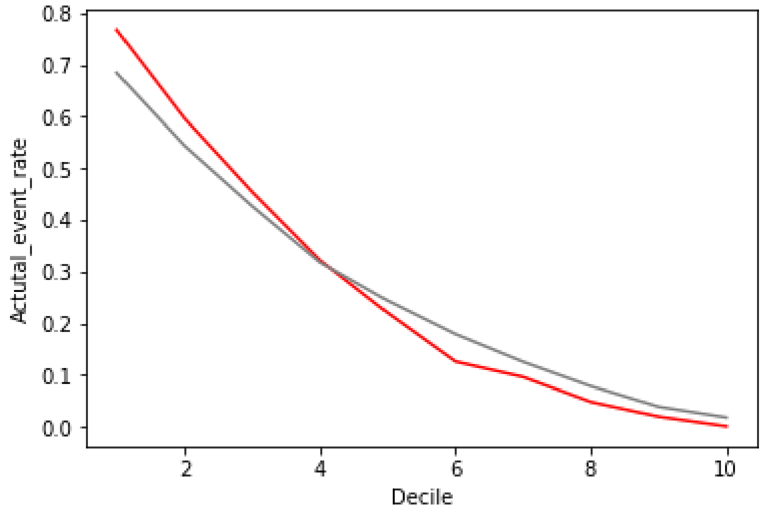
```
rank_df_actuals.rename(columns={'mean':'Actutal_event_rate'},inplace=True)
rank_df_predicted=pd.DataFrame(rank_df_predicted)
rank_df_predicted.rename(columns={'mean':'Predicted_event_rate'},inplace=True)
rank_df=pd.concat([rank_df_actuals,rank_df_predicted],axis=1,join="inner")

sorted_rank_df=rank_df.sort_values(by='P_Rank_rf',ascending=False)
sorted_rank_df['N_events']=rank_df['count']*rank_df['Actutal_event_rate']
sorted_rank_df['cum_events']=sorted_rank_df['N_events'].cumsum()
sorted_rank_df['event_cap']=sorted_rank_df['N_events']/max(sorted_rank_df['N_events'
sorted_rank_df['cum_event_cap']=sorted_rank_df['event_cap'].cumsum()

sorted_rank_df['N_non_events']=sorted_rank_df['count']-sorted_rank_df['N_events']
sorted_rank_df['cum_non_events']=sorted_rank_df['N_non_events'].cumsum()
sorted_rank_df['non_event_cap']=sorted_rank_df['N_non_events']/max(sorted_rank_df['N
sorted_rank_df['cum_non_event_cap']=sorted_rank_df['non_event_cap'].cumsum()

sorted_rank_df['KS']=round((sorted_rank_df['cum_event_cap']-sorted_rank_df['cum_non_

sorted_rank_df['random_cap']=sorted_rank_df['count']/max(sorted_rank_df['count'].cum
sorted_rank_df['cum_random_cap']=sorted_rank_df['random_cap'].cumsum()
sorted_reindexed=sorted_rank_df.reset_index()
sorted_reindexed['Decile']=sorted_reindexed.index+1
sorted_reindexed
```

Out[85]:

	P_Rank_rf	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_cap	cum
0	10	705	0.767376	0.684186	541.0	541.0	0.289460	
1	9	704	0.596591	0.543542	420.0	961.0	0.224719	
2	8	704	0.454545	0.426888	320.0	1281.0	0.171215	
3	7	704	0.322443	0.318193	227.0	1508.0	0.121455	
4	6	704	0.220170	0.243803	155.0	1663.0	0.082932	
5	5	705	0.126241	0.179210	89.0	1752.0	0.047619	
6	4	704	0.096591	0.125750	68.0	1820.0	0.036383	
7	3	704	0.048295	0.078972	34.0	1854.0	0.018192	
8	2	704	0.019886	0.038930	14.0	1868.0	0.007491	
9	1	705	0.001418	0.018429	1.0	1869.0	0.000535	

In [86]:

```
ax = sns.lineplot( x="Decile", y="Actutal_event_rate", data=sorted_reindexed,color='
ax = sns.lineplot( x="Decile", y="Predicted_event_rate", data=sorted_reindexed,color
```



In [87]:

```
y_pred_prob = gbm.predict_proba(X_all)[:, 1]
df['pred_prob_gbm']=pd.DataFrame(y_pred_prob)
df['P_Rank_GBM']=pd.qcut(df['pred_prob_gbm'].rank(method='first').values,10,duplicat
rank_df_actuals=df.groupby('P_Rank_GBM')['target'].agg(['count','mean'])
rank_df_predicted=df.groupby('P_Rank_GBM')['pred_prob_gbm'].agg(['mean'])
rank_df_actuals=pd.DataFrame(rank_df_actuals)

rank_df_actuals.rename(columns={'mean':'Actutal_event_rate'},inplace=True)
rank_df_predicted=pd.DataFrame(rank_df_predicted)
rank_df_predicted.rename(columns={'mean':'Predicted_event_rate'},inplace=True)
rank_df=pd.concat([rank_df_actuals,rank_df_predicted],axis=1,join="inner")

sorted_rank_df=rank_df.sort_values(by='P_Rank_GBM',ascending=False)
sorted_rank_df['N_events']=rank_df['count']*rank_df['Actutal_event_rate']
sorted_rank_df['cum_events']=sorted_rank_df['N_events'].cumsum()
sorted_rank_df['event_cap']=sorted_rank_df['N_events']/max(sorted_rank_df['N_events'
sorted_rank_df['cum_event_cap']=sorted_rank_df['event_cap'].cumsum()

sorted_rank_df['N_non_events']=sorted_rank_df['count']-sorted_rank_df['N_events']
sorted_rank_df['cum_non_events']=sorted_rank_df['N_non_events'].cumsum()
sorted_rank_df['non_event_cap']=sorted_rank_df['N_non_events']/max(sorted_rank_df['N
sorted_rank_df['cum_non_event_cap']=sorted_rank_df['non_event_cap'].cumsum()

sorted_rank_df['KS']=round((sorted_rank_df['cum_event_cap']-sorted_rank_df['cum_non_

sorted_rank_df['random_cap']=sorted_rank_df['count']/max(sorted_rank_df['count'].cum
sorted_rank_df['cum_random_cap']=sorted_rank_df['random_cap'].cumsum()
sorted_reindexed=sorted_rank_df.reset_index()
sorted_reindexed['Decile']=sorted_reindexed.index+1
sorted_reindexed
```

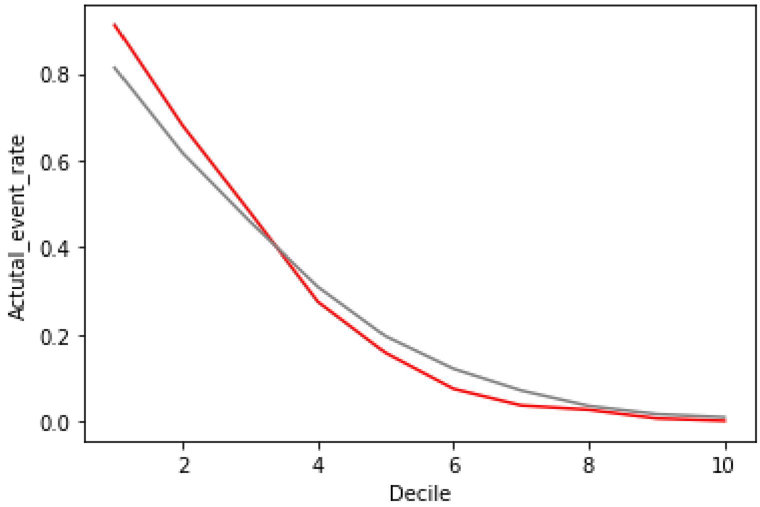
Out[87]:

	P_Rank_GBM	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_cap
0	10	705	0.912057	0.813446	643.0	643.0	0.344034
1	9	704	0.680398	0.617223	479.0	1122.0	0.256287
2	8	704	0.480114	0.458661	338.0	1460.0	0.180845
3	7	704	0.275568	0.310075	194.0	1654.0	0.103799
4	6	704	0.157670	0.196081	111.0	1765.0	0.059390
5	5	705	0.075177	0.121366	53.0	1818.0	0.028357
6	4	704	0.036932	0.071378	26.0	1844.0	0.013911

	P_Rank_GBM	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_cap
7	3	704	0.026989	0.035787	19.0	1863.0	0.010166
8	2	704	0.007102	0.016955	5.0	1868.0	0.002675
9	1	705	0.001418	0.009773	1.0	1869.0	0.000535

In [88]:

```
ax = sns.lineplot( x="Decile", y="Actutal_event_rate", data=sorted_reindexed,color='
ax = sns.lineplot( x="Decile", y="Predicted_event_rate", data=sorted_reindexed,color
```



In [89]:

```
y_pred_prob = clf.predict_proba(X_all)[:, 1]
df['pred_prob_stacking']=pd.DataFrame(y_pred_prob)
df['P_Rank_stacking']=pd.qcut(df['pred_prob_stacking'].rank(method='first').values,1
rank_df_actuals=df.groupby('P_Rank_stacking')['target'].agg(['count','mean'])
rank_df_predicted=df.groupby('P_Rank_stacking')['pred_prob_stacking'].agg(['mean'])
rank_df_actuals=pd.DataFrame(rank_df_actuals)

rank_df_actuals.rename(columns={'mean':'Actutal_event_rate'},inplace=True)
rank_df_predicted=pd.DataFrame(rank_df_predicted)
rank_df_predicted.rename(columns={'mean':'Predicted_event_rate'},inplace=True)
rank_df=pd.concat([rank_df_actuals,rank_df_predicted],axis=1,join="inner")

sorted_rank_df=rank_df.sort_values(by='P_Rank_stacking',ascending=False)
sorted_rank_df['N_events']=rank_df['count']*rank_df['Actutal_event_rate']
sorted_rank_df['cum_events']=sorted_rank_df['N_events'].cumsum()
sorted_rank_df['event_cap']=sorted_rank_df['N_events']/max(sorted_rank_df['N_events'
sorted_rank_df['cum_event_cap']=sorted_rank_df['event_cap'].cumsum()

sorted_rank_df['N_non_events']=sorted_rank_df['count']-sorted_rank_df['N_events']
sorted_rank_df['cum_non_events']=sorted_rank_df['N_non_events'].cumsum()
sorted_rank_df['non_event_cap']=sorted_rank_df['N_non_events']/max(sorted_rank_df['N
sorted_rank_df['cum_non_event_cap']=sorted_rank_df['non_event_cap'].cumsum()

sorted_rank_df['KS']=round((sorted_rank_df['cum_event_cap']-sorted_rank_df['cum_non_

sorted_rank_df['random_cap']=sorted_rank_df['count']/max(sorted_rank_df['count'].cum
sorted_rank_df['cum_random_cap']=sorted_rank_df['random_cap'].cumsum()
sorted_reindexed=sorted_rank_df.reset_index()
sorted_reindexed['Decile']=sorted_reindexed.index+1
sorted_reindexed
```

Out[89]:

	P_Rank_stacking	count	Actutal_event_rate	Predicted_event_rate	N_events	cum_events	event_ca
0	10	705	0.801418	0.778338	565.0	565.0	0.30230
1	9	704	0.603693	0.598471	425.0	990.0	0.22739
2	8	704	0.471591	0.418600	332.0	1322.0	0.17763
3	7	704	0.316761	0.267492	223.0	1545.0	0.11931
4	6	704	0.205966	0.182756	145.0	1690.0	0.07758
5	5	705	0.124823	0.129712	88.0	1778.0	0.04708
6	4	704	0.078125	0.097591	55.0	1833.0	0.02942
7	3	704	0.036932	0.075475	26.0	1859.0	0.01391
8	2	704	0.012784	0.060534	9.0	1868.0	0.00481
9	1	705	0.001418	0.054276	1.0	1869.0	0.00053

In [90]:

```
# Project Conclusion :-
# The GBM Model has performed the best and will be used for Customer targeting with
# Since Monthly Income and Existing EMI are the most important features for the GBM
# We will build a Business Value Metric based on Existing EMI/Monthly Income
# Low Values of this ratio will indicate valueable customers
# Within the High Value group, we can leverage the model to identify the best target
```

In [91]:

```
df['Tenure_Rank']=pd.qcut(df['tenure'].rank(method='first').values,10,duplicates='dr
```

In [92]:

```
df.groupby('Tenure_Rank')['tenure'].agg(['min','max','mean'])
```

Out[92]:

Tenure_Rank	min	max	mean
1	0	2	1.099291
2	2	6	3.566761
3	6	12	8.779830
4	12	20	15.531250
5	20	29	24.153191
6	29	40	34.004261
7	40	50	45.014205
8	50	60	55.370739
9	60	69	65.001420
10	69	72	71.191489

In [93]:

```
df['tenure'].mean()
```

Out[93]:

```
32.37114865824223
```

In [94]:
```
df['Tenure_Segment']=np.where(df['Tenure_Rank']<=6,"Low Tenure","High Tenure")
```

In [95]:
```
df['MonthlyCharges_Rank']=pd.qcut(df['MonthlyCharges'].rank(method='first').values,1
```

In [96]:
```
df.groupby('MonthlyCharges_Rank')['MonthlyCharges'].agg(['min','max','mean'])
```

Out[96]:

MonthlyCharges_Rank	min	max	mean
1	18.25	20.05	19.622482
2	20.05	25.05	21.732599
3	25.05	45.85	35.514773
4	45.85	58.75	52.532244
5	58.85	70.35	65.314965
6	70.35	79.10	74.623864
7	79.10	85.50	82.140057
8	85.50	94.25	89.840199
9	94.25	102.60	98.036364
10	102.60	118.75	108.260922

In [97]:
```
df['MonthlyCharges'].mean()
```

Out[97]:
```
64.76169246059922
```

In [98]:
```
df['Monthly_Charge_Segment']=np.where(df['MonthlyCharges_Rank']<=5,"Low Charges","Hi
```

In [99]:
```
df['Predicted_Churn_Rank']=np.where(df['P_Rank_GBM']>=8,"Top 3","Bottom 7")
```

Slice the data with respect to Top 4 and Bottom 6 Probability Ranks from the GBM Model

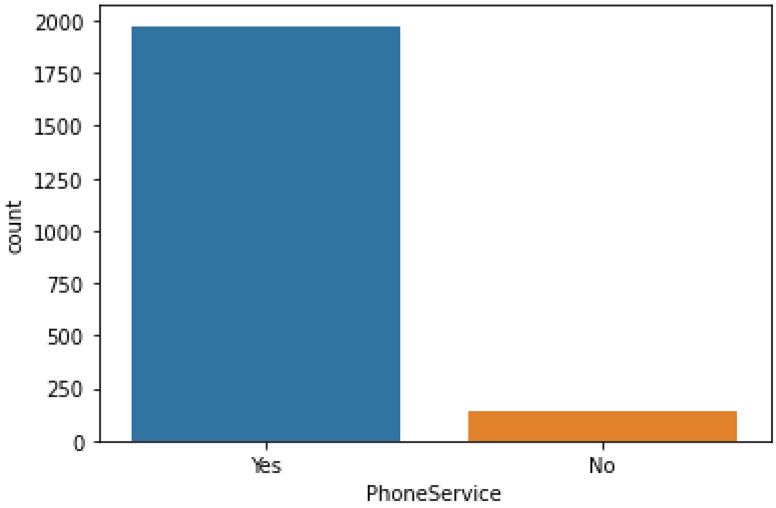
In [100…
```
df_top3=df.loc[df['Predicted_Churn_Rank']=='Top 3',:]
```

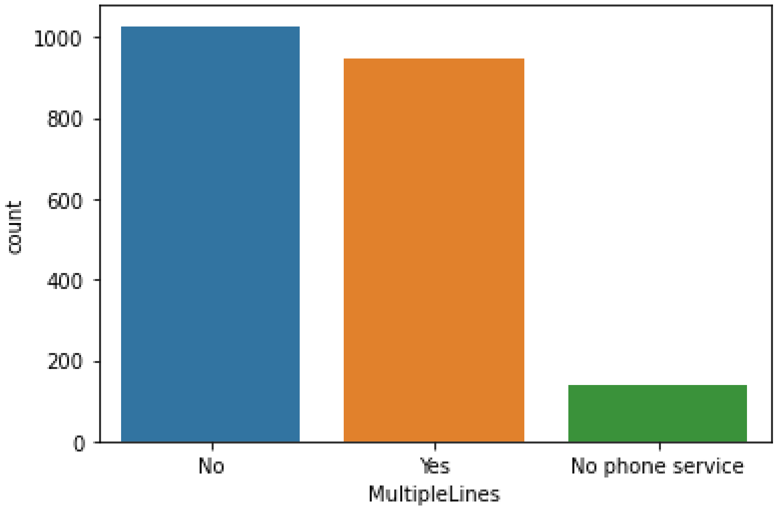
In [101…
```
service_list=['PhoneService','MultipleLines','InternetService','OnlineSecurity','Onl
              ,'StreamingTV','StreamingMovies','Contract','PaperlessBilling']
target=['target']

total=service_list+target
```

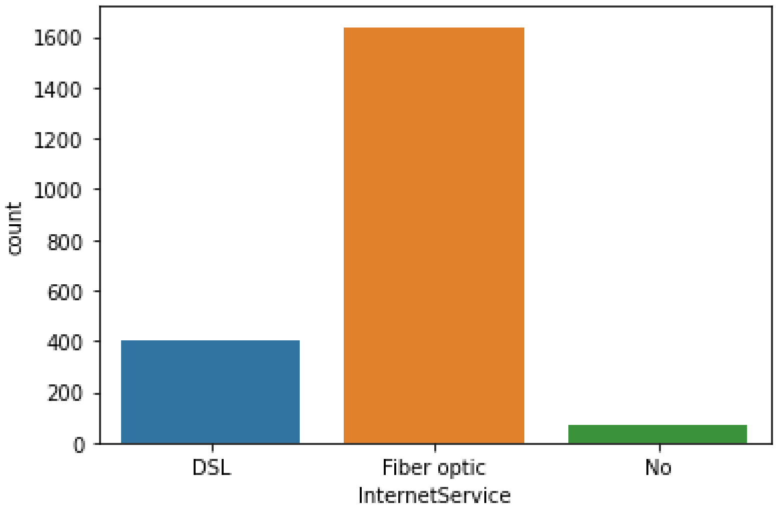
In [102…
```
df_top3_services=df_top3[service_list]
```

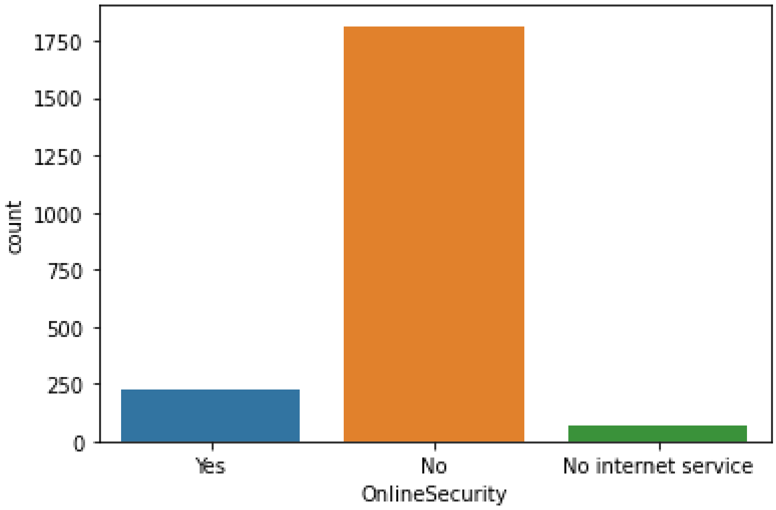
In [103…

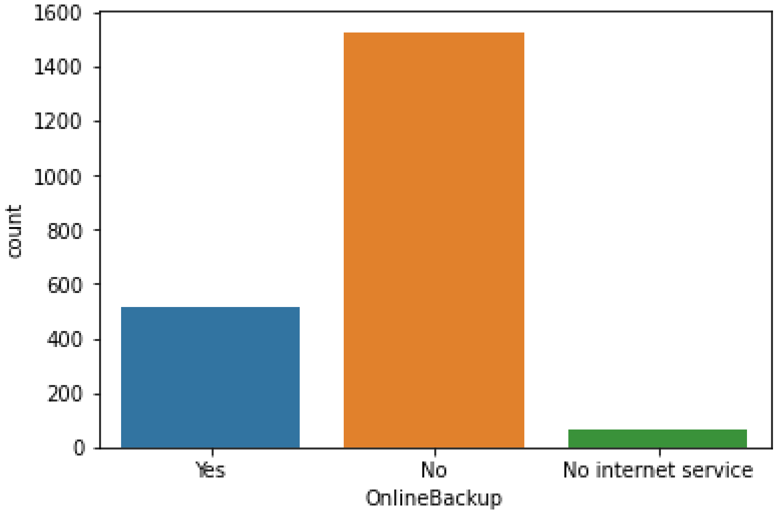
```
for col in (df_top3_services.columns):
    plt.figure()
    sns.countplot(x=col,data=df_top3_services)
plt.show()
```

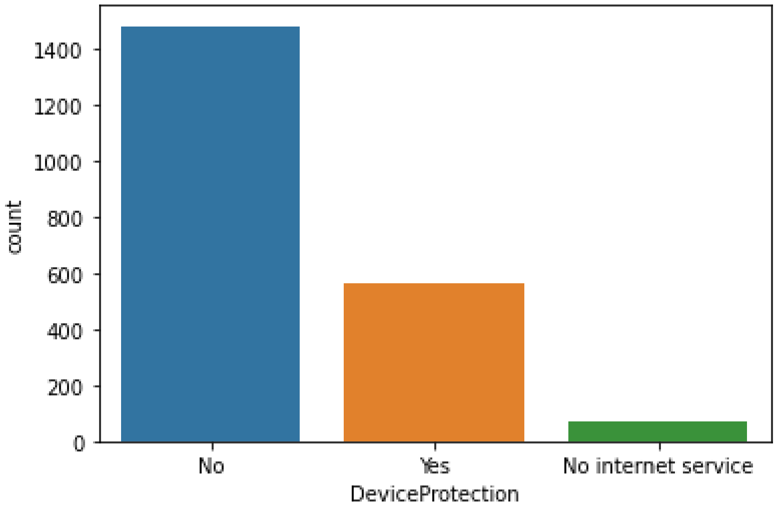


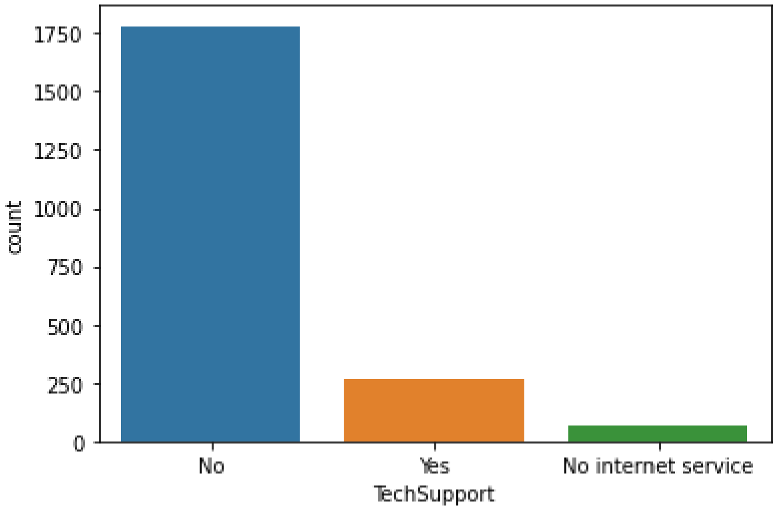


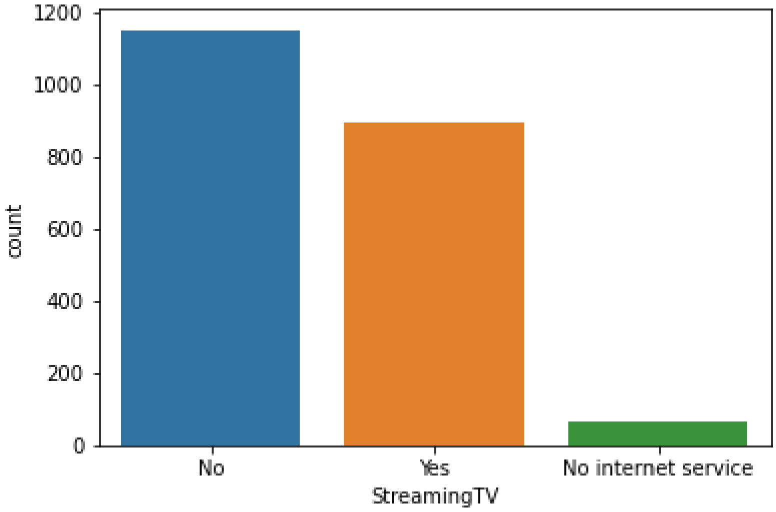


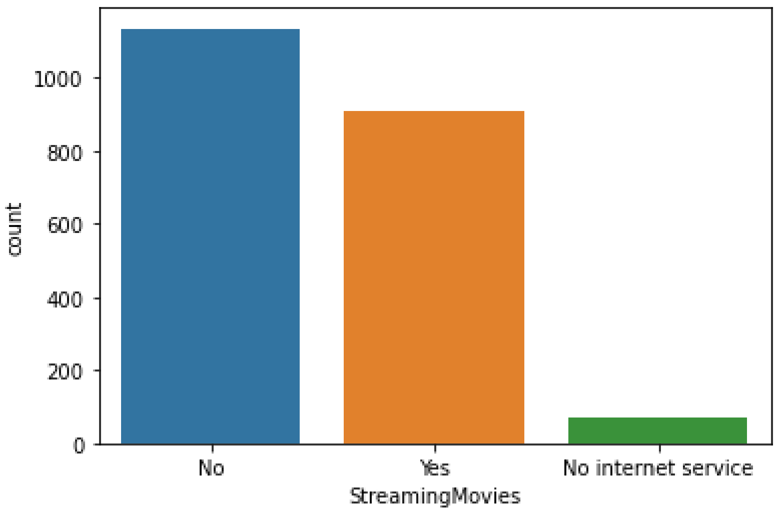

1750
1500
1250
1000
750
500
250
0
count
Yes
No
No internet service
OnlineSecurity

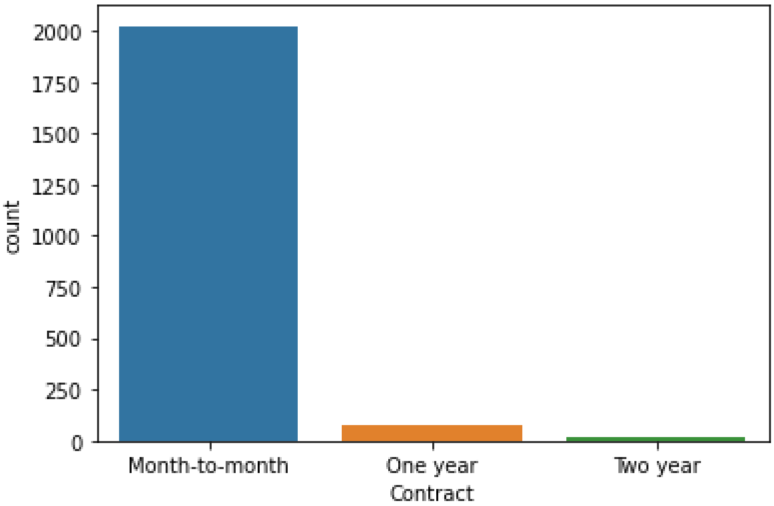

1600
1400
1200
1000
800
600
400
200
0
count
Yes
No
No internet service
OnlineBackup

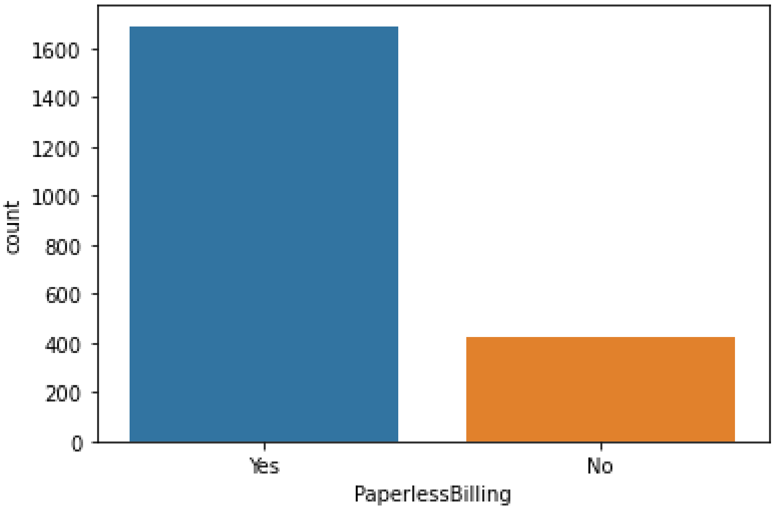

1400
1200
1000
800
600
400
200
0
count
No
Yes
No internet service
DeviceProtection


count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
TechSupport


count
1200
1000
800
600
400
200
0
No
Yes
No internet service
StreamingTV


count
1000
800
600
400
200
0
No
Yes
No internet service
StreamingMovies





In [104…

```
pd.crosstab(index=df_top3['Monthly_Charge_Segment'], columns=df_top3['Tenure_Segment
```

Out[104…

Tenure_Segment	High Tenure	Low Tenure
Monthly_Charge_Segment		
High Charges	99.889737	86.625057
Low Charges	44.785714	48.050422

In [105…

```
pd.crosstab(index=df_top3['Monthly_Charge_Segment'], columns=df_top3['Tenure_Segment
```

Out[105…

Tenure_Segment	High Tenure	Low Tenure
Monthly_Charge_Segment		
High Charges	190	1317
Low Charges	14	592

In [106…

```
# Recommendations
# Device Protection with Online Services
# Convert customer to DSL if they are facing challenges with Fiber Optics
# Offer discounts on Yearly contracts
```

In [ ]: