MSBA307 Final Project: Customer Churn Prediction

Student: Yichun Chen

In [1]:
```
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns
%matplotlib inline
```

1.Load Data

In [2]:
```
df = pd.read_csv("Telco-Customer-Churn.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Internet
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	Fib

5 rows × 21 columns

2.Data Exploration

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

Out[4]:
```
(7043, 21)
```

There are 7043 rows, and 21 columns.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
```

```
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

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

Out[6]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

In [7]:
```
len(df.drop_duplicates())
```

Out[7]:
```
7043
```

no duplicates

In [8]:
```
df.isna().sum()
```

Out[8]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
```

```
Churn               0
dtype: int64
```

no null values

In [9]:
```
# check the number of columns:
len(df.columns) #num of columns
```

Out[9]:
```
21
```

In [10]:
```
#check unique values for the categorical columns:
for col in df.columns:
    if df[col].dtypes == "object":
        print(col, len(df[col].unique()), df[col].unique())
```

```
customerID 7043 ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
gender 2 ['Female' 'Male']
Partner 2 ['Yes' 'No']
Dependents 2 ['No' 'Yes']
PhoneService 2 ['No' 'Yes']
MultipleLines 3 ['No phone service' 'No' 'Yes']
InternetService 3 ['DSL' 'Fiber optic' 'No']
OnlineSecurity 3 ['No' 'Yes' 'No internet service']
OnlineBackup 3 ['Yes' 'No' 'No internet service']
DeviceProtection 3 ['No' 'Yes' 'No internet service']
TechSupport 3 ['No' 'Yes' 'No internet service']
StreamingTV 3 ['No' 'Yes' 'No internet service']
StreamingMovies 3 ['No' 'Yes' 'No internet service']
Contract 3 ['Month-to-month' 'One year' 'Two year']
PaperlessBilling 2 ['Yes' 'No']
PaymentMethod 4 ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
TotalCharges 6531 ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
Churn 2 ['No' 'Yes']
```

Exploration findings:

- customerID should be dropped;
- SeniorCitizen should be categorical, but is shown as integer
- TotalCharges should be numeric, but is shown as object
- 16 categorical features;
- 3 numerical features
- 1 target: Churn

3. Data Wrangling:

Drop customerID

In [11]:
```
df = df.drop("customerID", axis=1)
```

In [12]:
```
df.head()
```

Out[12]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onli
0	Female	0	Yes	No	1	No	No phone service	DSL	
1	Male	0	No	No	34	Yes	No	DSL	

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
2	Male	0	No	No	2	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
4	Female	0	No	No	2	Yes	No	Fiber optic

SeniorCiziten

In [96]:
```
#binary yes->1, no->0
df.SeniorCitizen = df.SeniorCitizen.map(lambda x: "yes" if x== 1 else "No")
```

In [97]:
```
df.head()
```

Out[97]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onli
0	Female	No	Yes	No	1	No	No phone service	DSL	
1	Male	No	No	No	34	Yes	No	DSL	
2	Male	No	No	No	2	Yes	No	DSL	
3	Male	No	No	No	45	No	No phone service	DSL	
4	Female	No	No	No	2	Yes	No	Fiber optic	

TotalCharges

In [13]:
```
cnt = 0
for val in df.TotalCharges:
    if val == " ":
        cnt += 1
        continue
    float(val)
print(cnt)
```

```
11
```

There are 11 missing values in TotalCharges, will delete the 11 rows

In [14]:
```
df = df[df.TotalCharges != " "].reset_index(drop=True)
```

In [15]:
```
df.head()
```

Out[15]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onli
0	Female	0	Yes	No	1	No	No phone service	DSL	
1	Male	0	No	No	34	Yes	No	DSL	
2	Male	0	No	No	2	Yes	No	DSL	
3	Male	0	No	No	45	No	No phone service	DSL	
4	Female	0	No	No	2	Yes	No	Fiber optic	

In [16]:
```
len(df)
```
Out[16]:
```
7032
```

In [17]:
```
# change TotalCharges from "object" to "float"
df["TotalCharges"] = df["TotalCharges"].map(float)
```

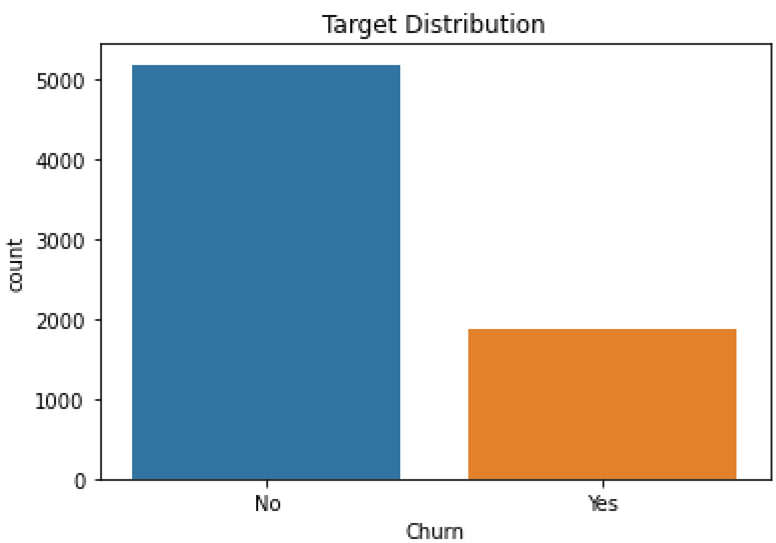
4. EDA

Target distribution:

The data is imbalanced, with non-churners being about 3 times the number of churners.

In [14]:
```
df.Churn.value_counts()
```
Out[14]:
```
No      5174
Yes     1869
Name: Churn, dtype: int64
```

In [68]:
```
sns.countplot(data=df, x="Churn")
plt.title("Target Distribution")
```
Out[68]:
```
Text(0.5, 1.0, 'Target Distribution')
```



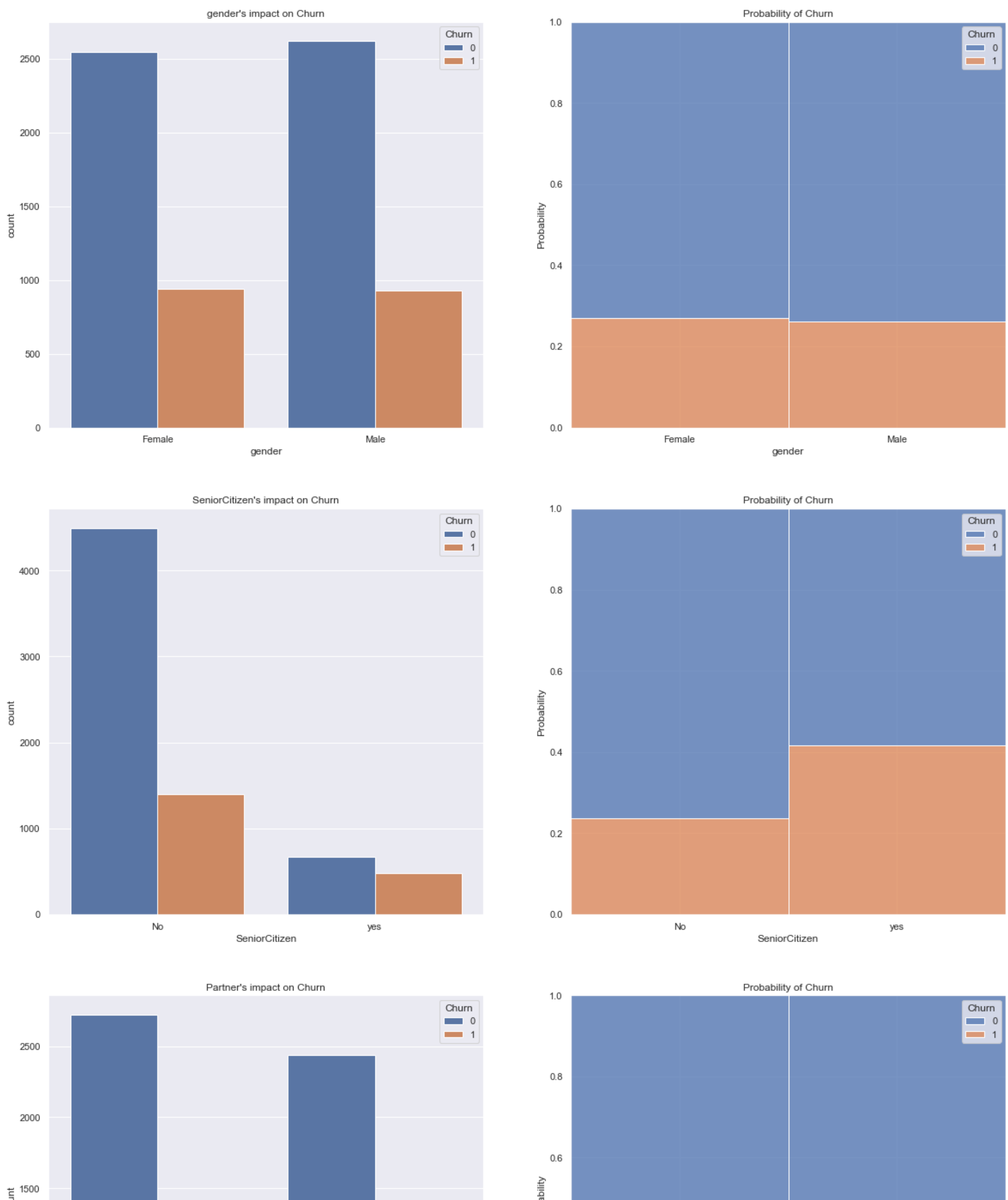
Impact of Demographic Features on Churn: 'gender', 'SeniorCitizen', 'Partner', 'Dependents',

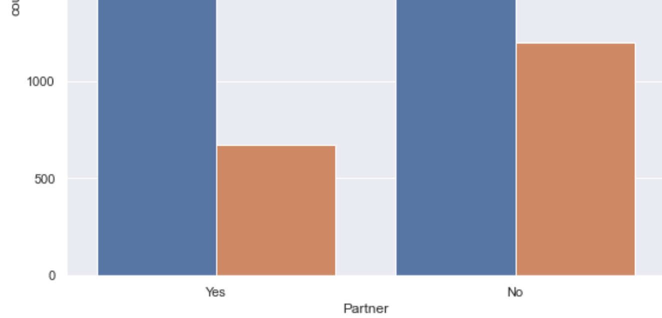
The following plots shows that all demographic features except gender have an impact on churn probability.

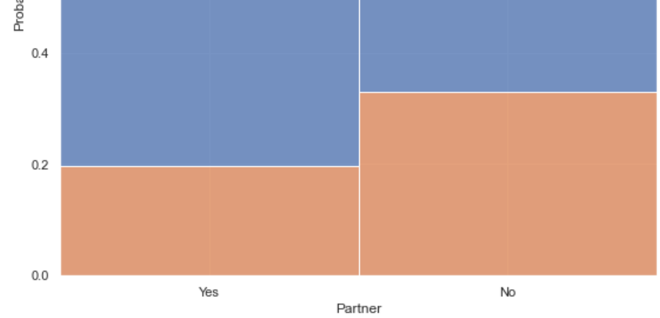
- No much difference between male and female
- Customoers are mostly non-senior citizens; but the probability plot showed that the older people are more likely to churn;
- customers with partner(s) are less likely to churn than those without;
- customers with dependent(s) are less likely to churn than those without

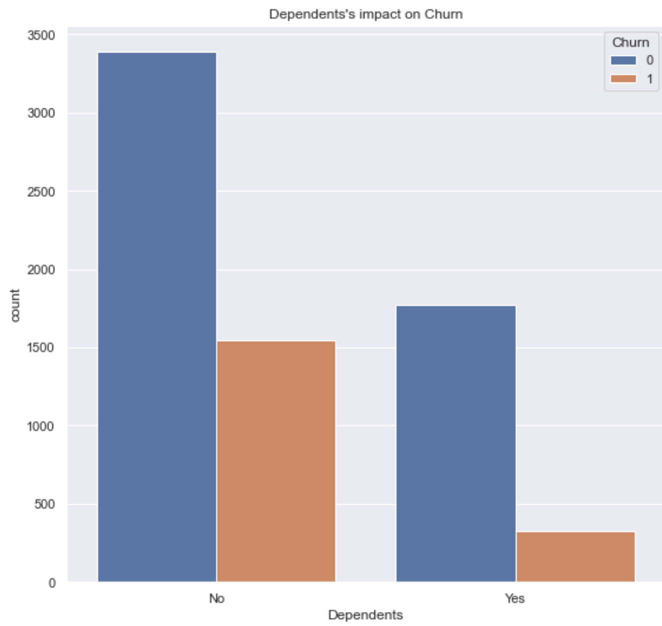
In [98]:
```
demographic_features = ['gender', 'SeniorCitizen', 'Partner', 'Dependents']
```

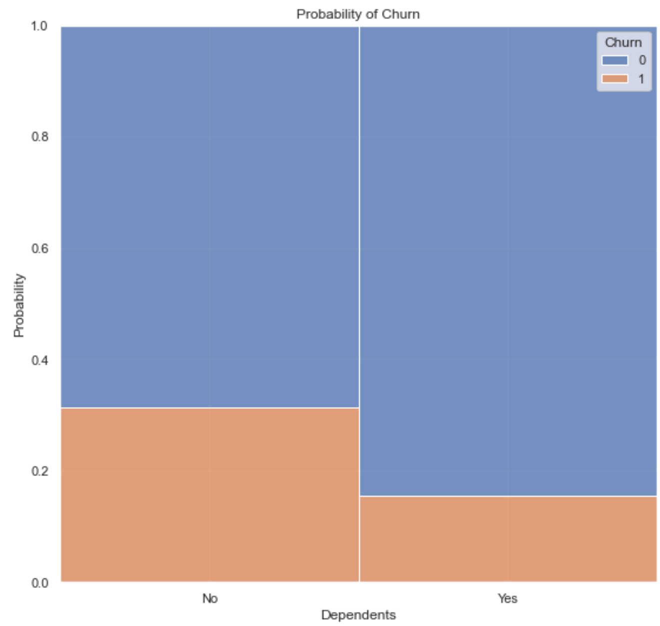
```python
# Plot config
sns.set_theme(style='darkgrid')


# Viz Foundation
fig, ax = plt.subplots(4,2,figsize=(20, 40))
n=0
for col in demographic_features:
    sns.countplot(data = df, x =col,  hue='Churn', ax = ax[n,0]),
    sns.histplot(data = df, x =col,  hue='Churn', multiple='fill', stat='probability', a
    ax[n,0].set_title(f'{col}\'s impact on Churn')
    ax[n,1].set_title('Probability of Churn')
    n= n+1
```











Impact of Service Usage Features on Churn:

'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV','StreamingMovies'

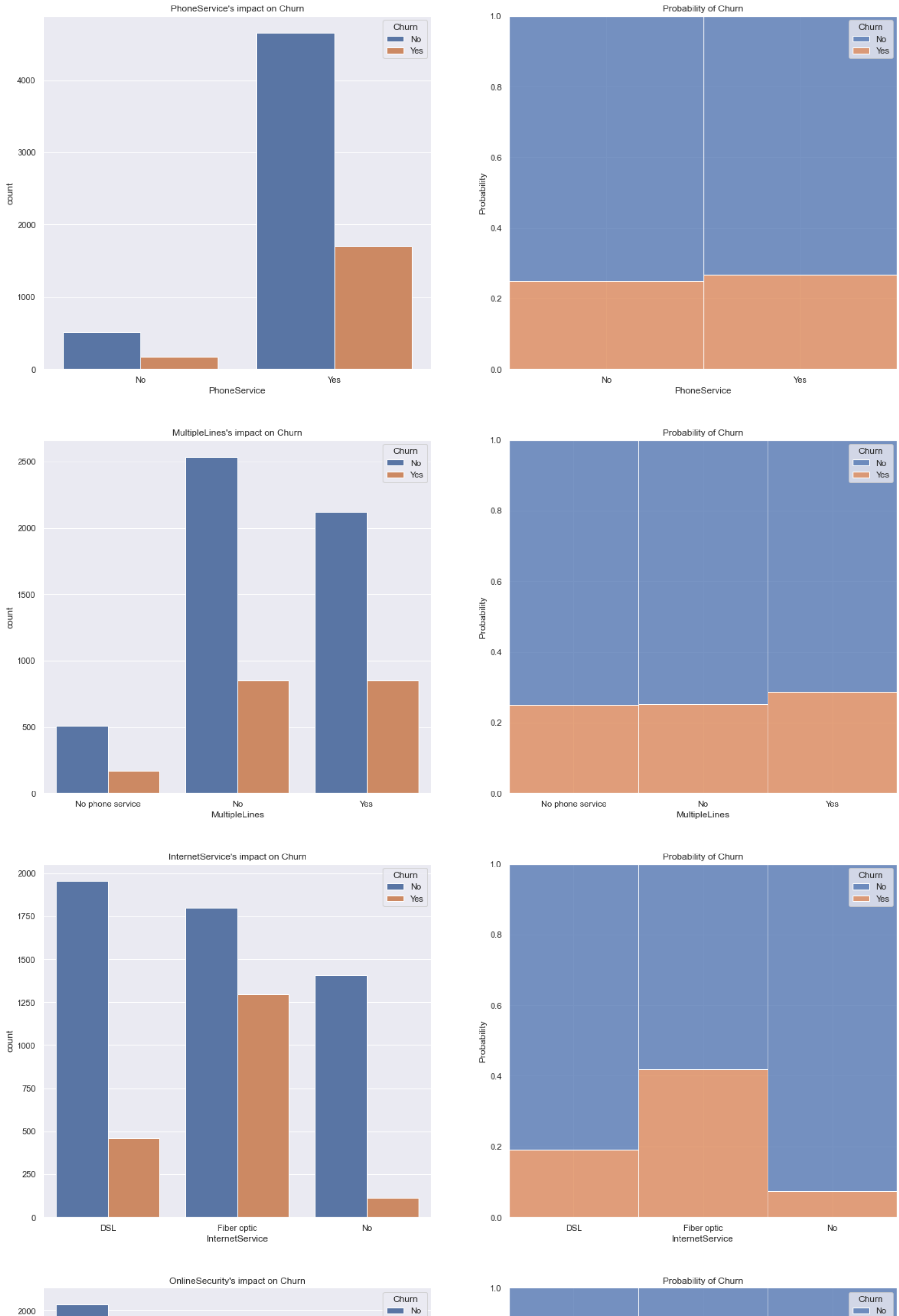
The following plots shows that:

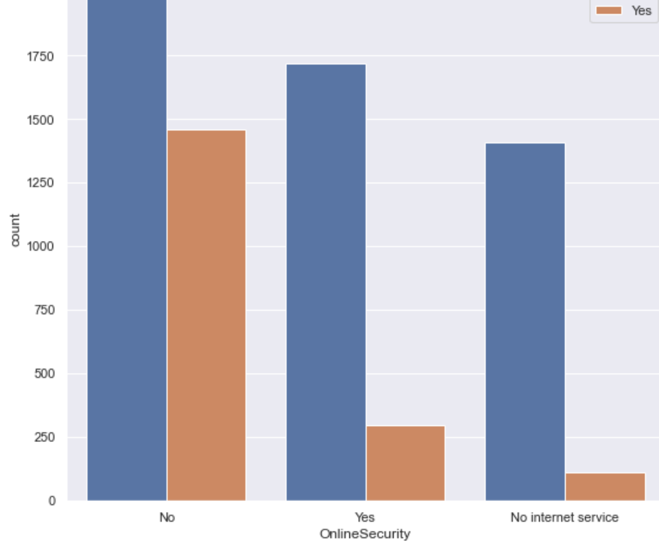
- customers who used fiber optic InternetService are twice likely to churn than those used DSL.
- customers who subscribed the following services are less likely to churn : 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport'
- Subscription of the following services has little impact on churn probability: 'PhoneService', 'MultipleLines', 'StreamingTV','StreamingMovies'

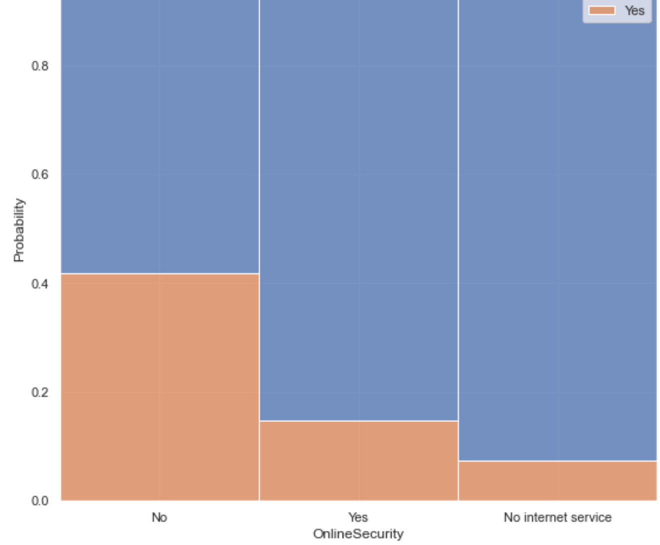
In [61]:

```
service_features = ['PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity'

# Plot config
sns.set_theme(style='darkgrid')


# Viz Foundation
fig, ax = plt.subplots(9,2,figsize=(20, 90))
n=0
for col in service_features:
    sns.countplot(data = df, x =col,  hue='Churn', ax = ax[n,0]),
    sns.histplot(data = df, x =col,  hue='Churn', multiple='fill', stat='probability', a
    ax[n,0].set_title(f'{col}\'s impact on Churn')
```

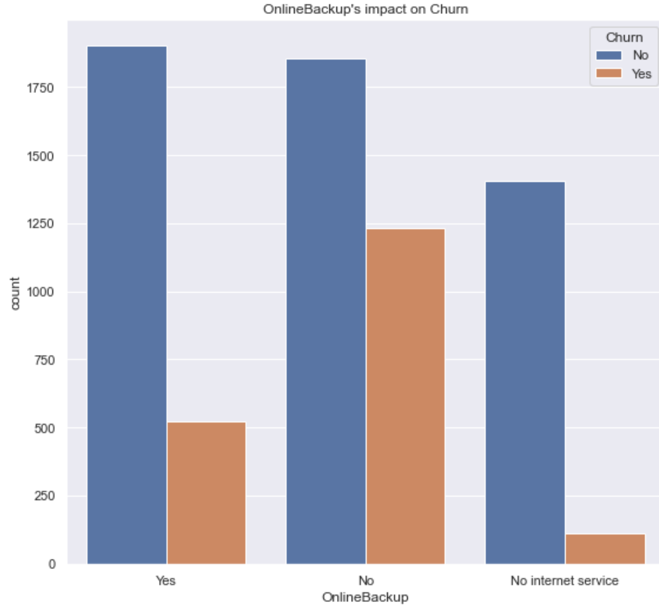
```
    ax[n,1].set_title('Probability of Churn')
    n= n+1
```



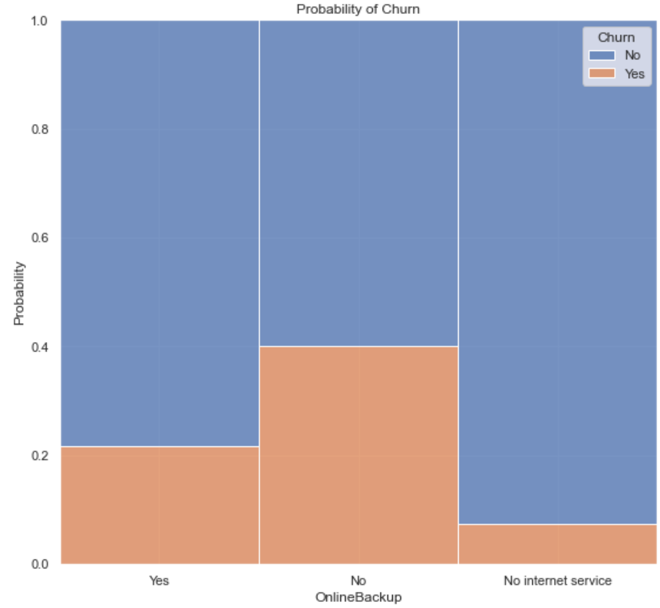




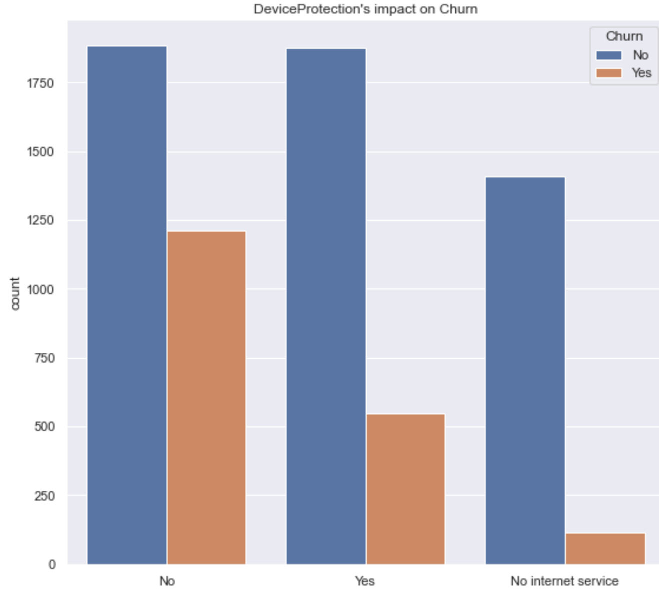
OnlineBackup's impact on Churn



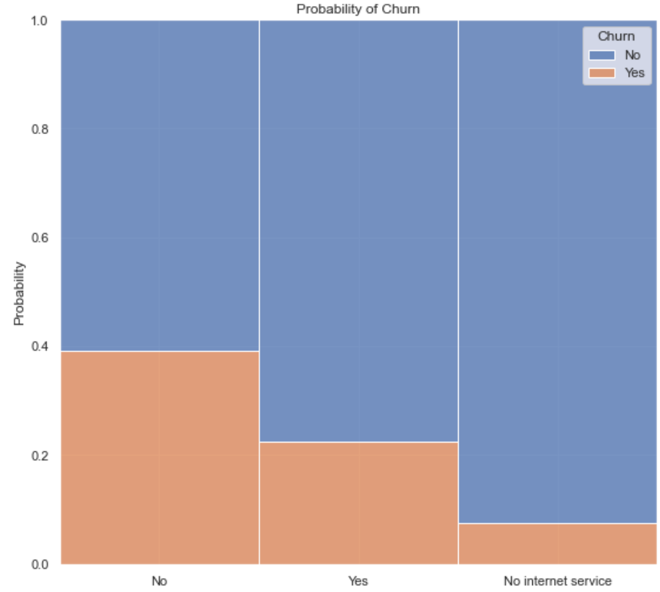
Probability of Churn



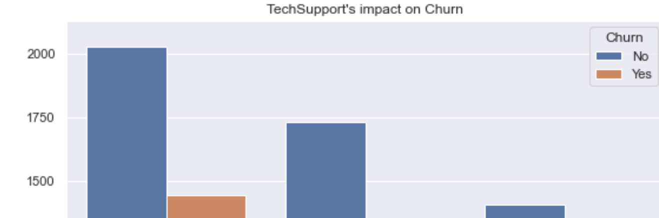
DeviceProtection's impact on Churn



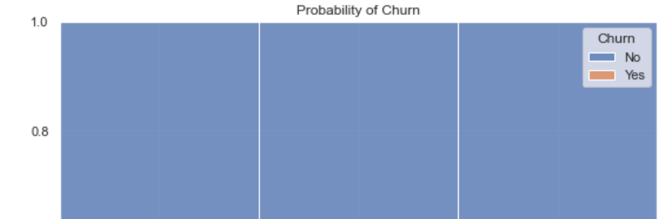
Probability of Churn

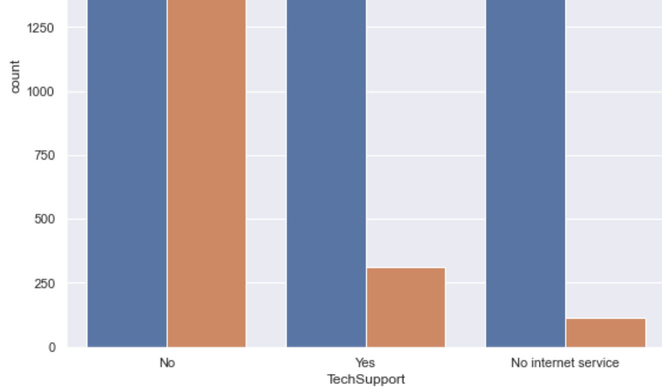


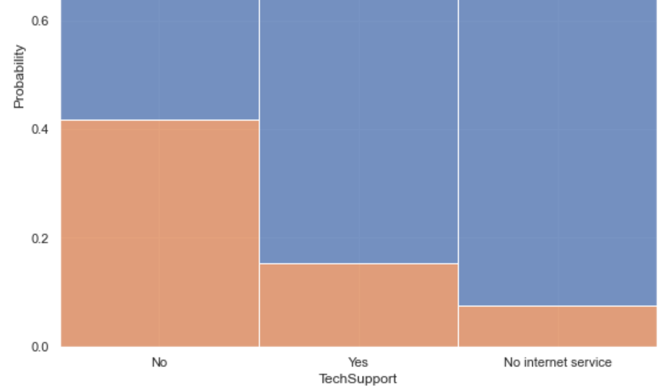
TechSupport's impact on Churn

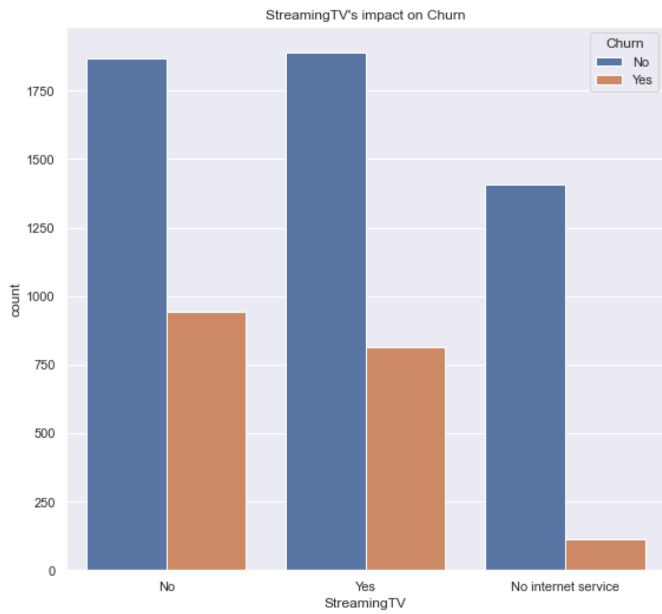


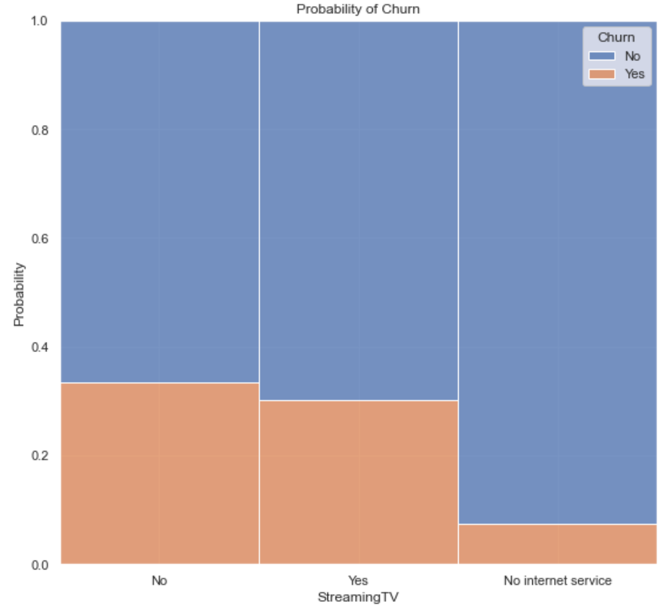
Probability of Churn

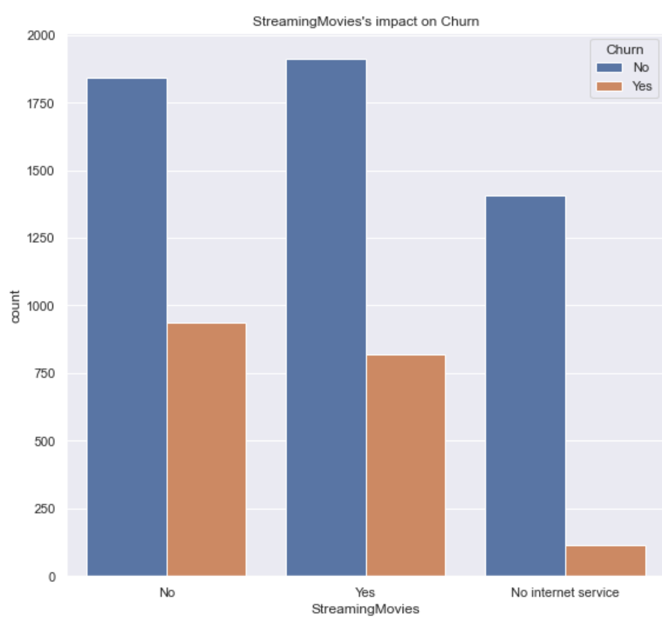


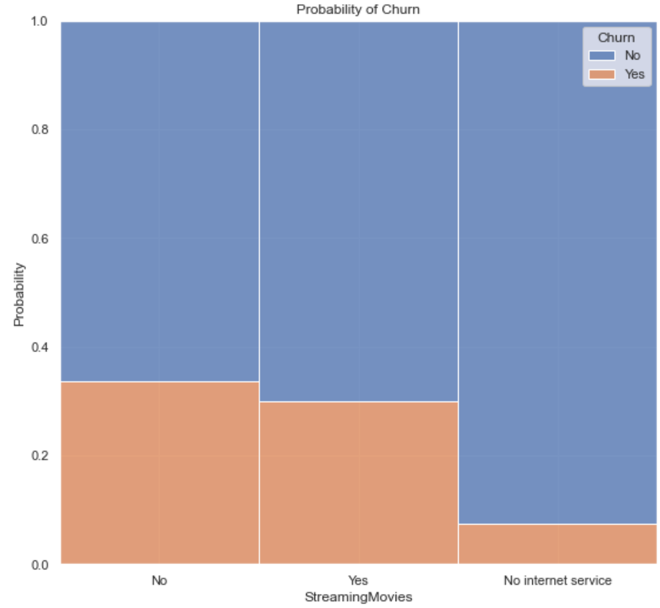












Impact of Account Information on Churn:

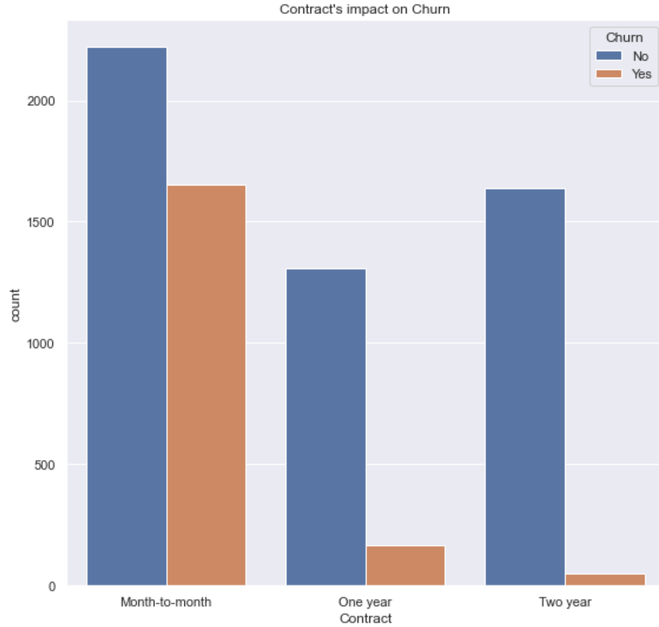
Account data can be diviced into categorical features and numeric features

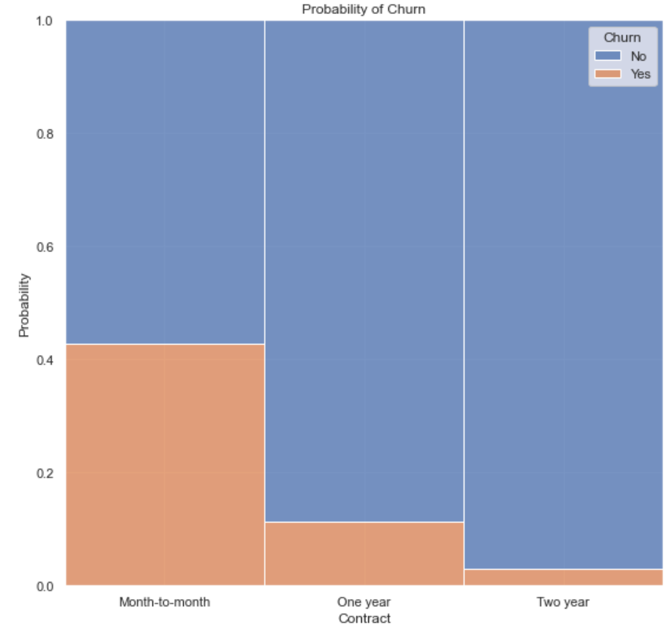
- categorical featuares : ['Contract', 'PaperlessBilling', 'PaymentMethod']
- numeric features: ['tenure', 'MonthlyCharges', 'TotalCharges']

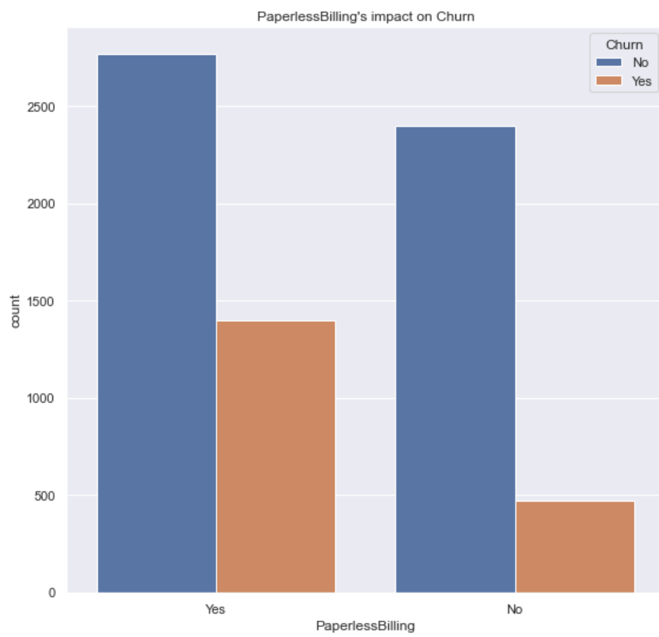
The plots showed that:

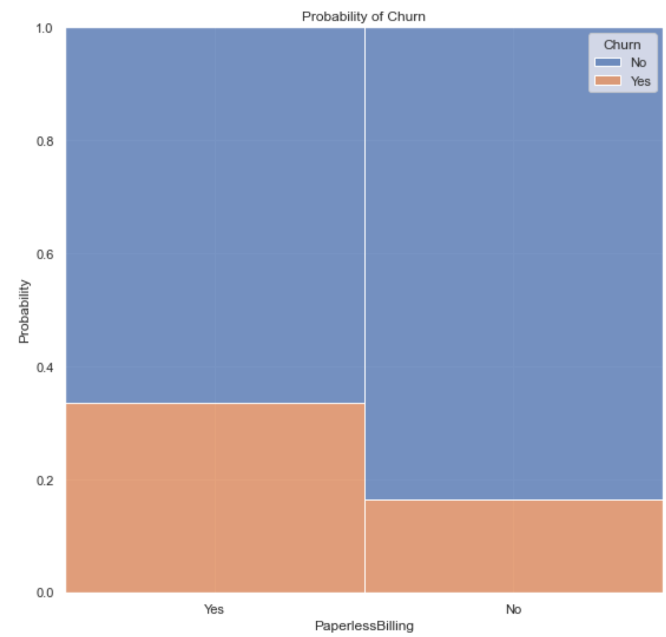
- customers in month-to-month contract are far likely to churn than customers in 1-year or 2-year contract.
- customers who signed up for paperless billing has a higher probability to churn
- customers who make payments by electronic check are about twice likely to churn than people who use other payments such as mailed check, bank transfer and credit card.
- tenure seems to be negatively correlated with churn, which is logical as customers who have stayed with the company for a longer time are usually less likely to churn
- monthly charges: no clear pattern except that customers who paid very little or very much are less likely to churn
- total charges: the plot show one unexpected insight that total charges is negatively correlated with churn. It may be because loyal customers subscribe a lot more services and thus have higer charges.

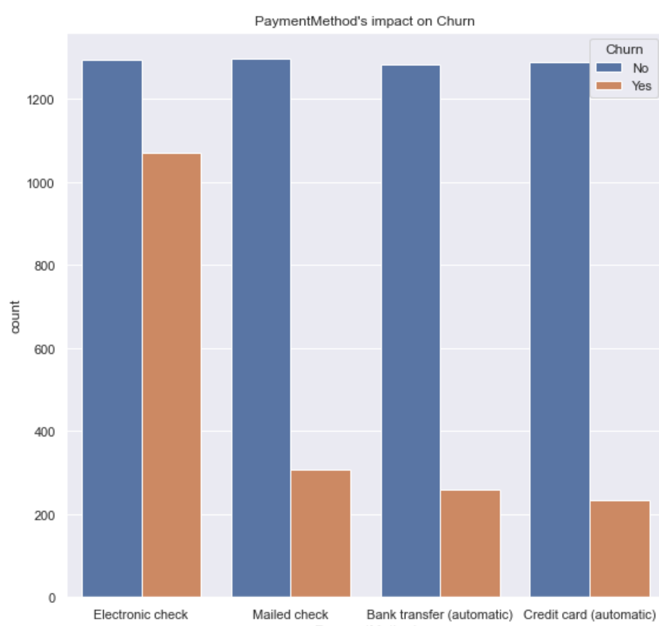
In [67]:
```
account_features = ['Contract', 'PaperlessBilling', 'PaymentMethod']

# Plot config
sns.set_theme(style='darkgrid')

# Viz Foundation
fig, ax = plt.subplots(3,2,figsize=(20, 30))
n=0
for col in account_features:
    sns.countplot(data = df, x =col,  hue='Churn', ax = ax[n,0]),
    sns.histplot(data = df, x =col,  hue='Churn', multiple='fill', stat='probability', a
    ax[n,0].set_title(f'{col}\'s impact on Churn')
    ax[n,1].set_title('Probability of Churn')
    n= n+1
```

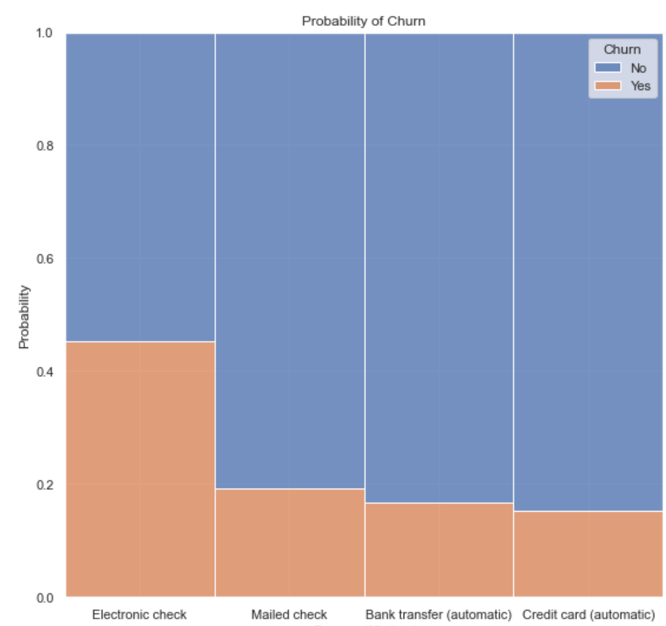








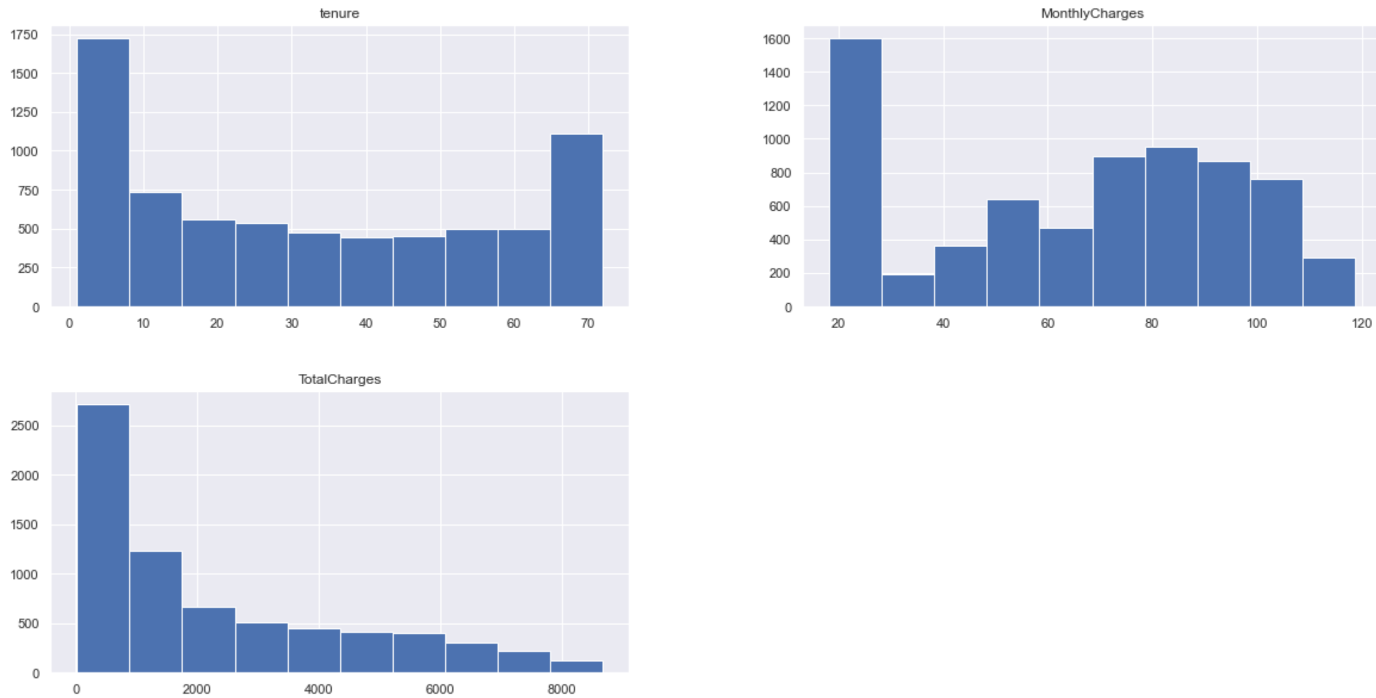




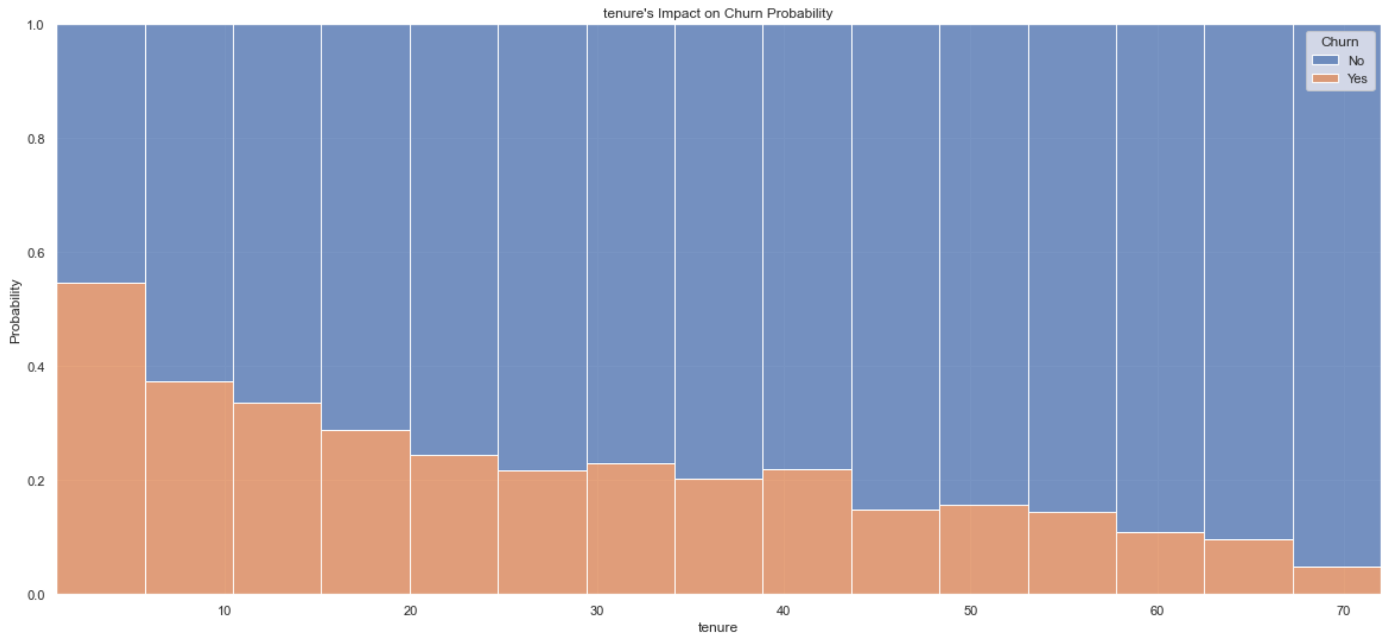
numerical features distribution:

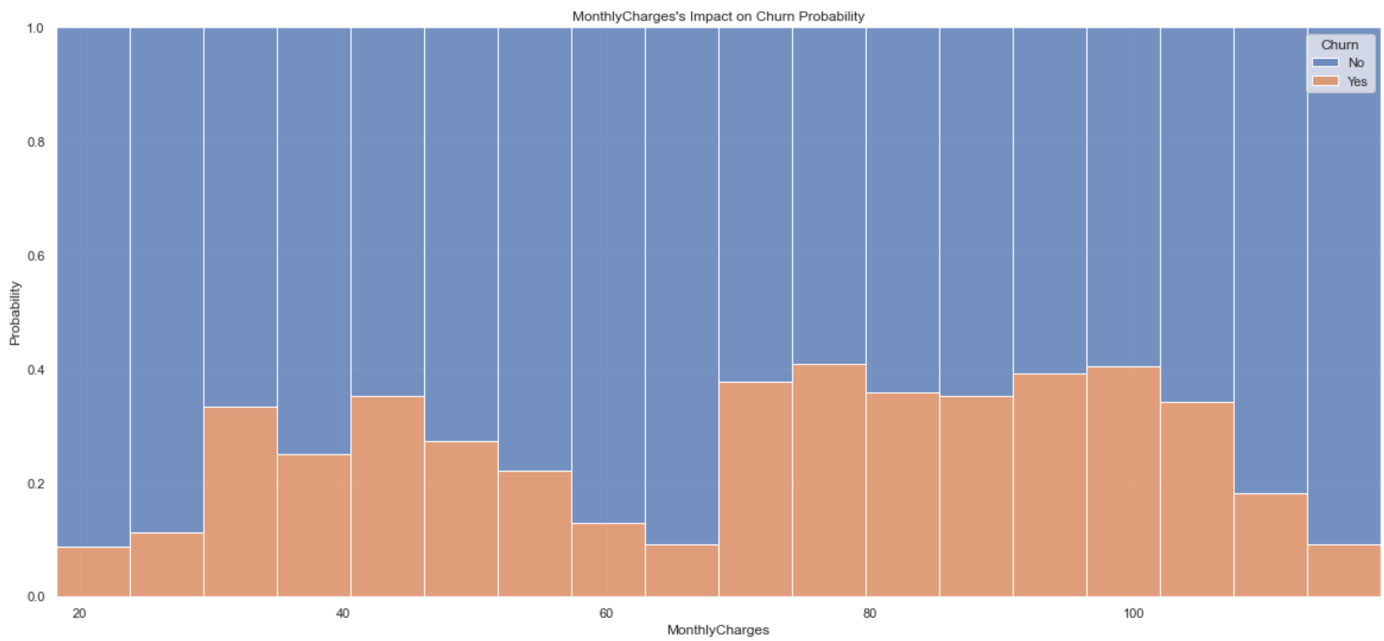
In [74]:
```
df.hist(column= numeric_features, figsize=(20,10))
```

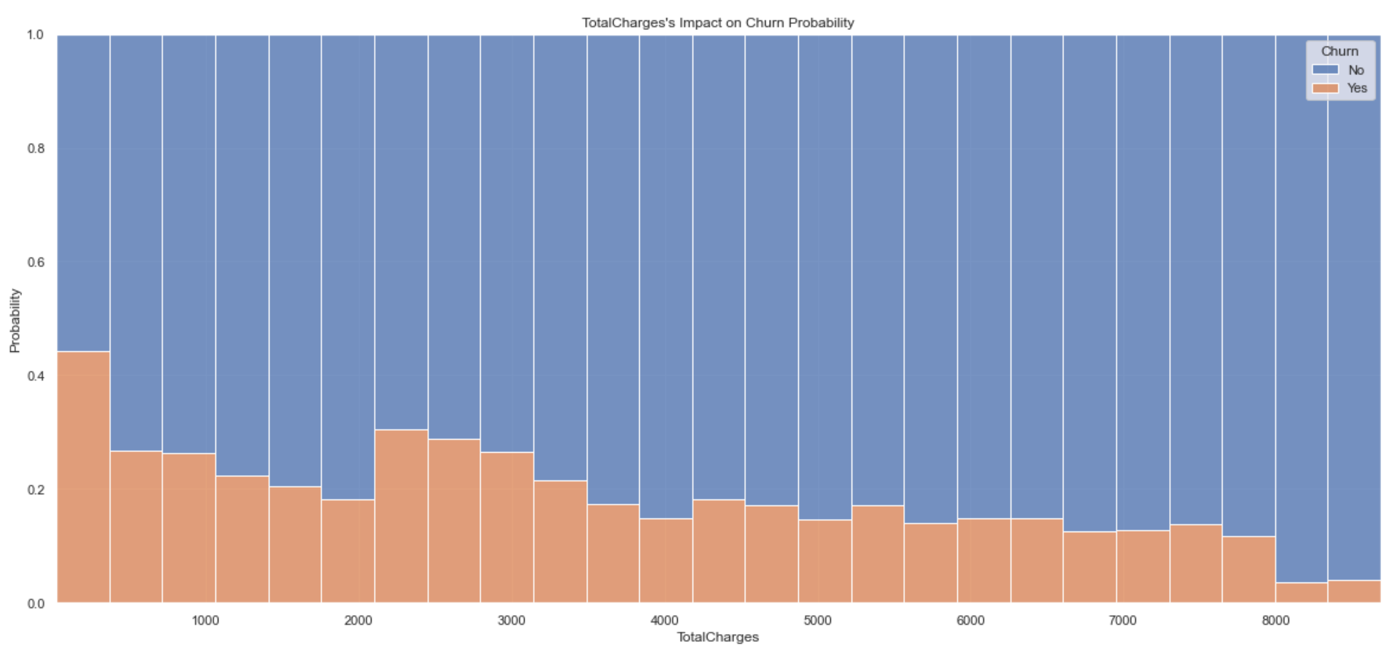
Out[74]:
```
array([[<AxesSubplot:title={'center':'tenure'}>,
        <AxesSubplot:title={'center':'MonthlyCharges'}>],
       [<AxesSubplot:title={'center':'TotalCharges'}>, <AxesSubplot:>]],
      dtype=object)
```



In [86]:
```
numeric_features = ['tenure', 'MonthlyCharges', 'TotalCharges']

# Plot config
sns.set_theme(style='darkgrid')

# Viz Foundation
fig, ax = plt.subplots(3,1,figsize=(20, 30))
n=0
for col in numeric_features:
    sns.histplot(data = df, x =col,  hue='Churn', multiple='fill', stat='probability', a
    ax[n].set_title(f'{col}\'s Impact on Churn Probability')
    n= n+1
```


tenure's Impact on Churn Probability
Churn
No
Yes
Probability
1.0
0.8
0.6
0.4
0.2
0.0
10
20
30
40
50
60
70
tenure


MonthlyCharges's Impact on Churn Probability
Churn
No
Yes
Probability
1.0
0.8
0.6
0.4
0.2
0.0
20
40
60
80
100
MonthlyCharges


TotalCharges's Impact on Churn Probability
Churn
No
Yes
Probability
1.0
0.8
0.6
0.4
0.2
0.0
1000
2000
3000
4000
5000
6000
7000
8000
TotalCharges

5. Data Preparation for modeling

Target

In [89]:
```
#binary yes->1, no->0
df.Churn = df.Churn.map(lambda x: 1 if x=="Yes" else 0)
```

In [90]:
```
df.head()
```

Out[90]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onli
0	Female	0	Yes	No	1	No	No phone service	DSL	
1	Male	0	No	No	34	Yes	No	DSL	
2	Male	0	No	No	2	Yes	No	DSL	
3	Male	0	No	No	45	No	No phone service	DSL	
4	Female	0	No	No	2	Yes	No	Fiber optic	

column transformation:

- One Hot Encode all the categorical features
- Standardize all the numeric features

In [117…
```
# import libraries:
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
```

In [118…
```

# select all categorical features
categorical_features = df.select_dtypes(include=['object']).columns

# create OneHotEncoder step for categorical features
categorical_transformer = Pipeline(steps=[
                                ('onehot', OneHotEncoder(handle_unknown='ignore'))
                                        ])

# select all numeric features(exclude target "Churn")
numeric_features = df.select_dtypes(include=['number']). drop(columns=['Churn']).columns

# normalize all numeric features
numeric_transformer = Pipeline(steps=[
                                ('scaler', StandardScaler())
                                    ])

## create pre-processor container containing numeric and categorical transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
                ])
```

6. Modeling

In [110…
```
## Split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=20
```

We will build two prediciton models (random forest and logistic regression) , and then select the one with higher performance

Model 1 - Random Forest

In [219…
```
from sklearn.ensemble import RandomForestClassifier
```

In [220…
```
# instantiate container with pipeline steps
rf = Pipeline(steps=[
                    ('preprocessor', preprocessor),
                    ('classifier', RandomForestClassifier(n_estimators=100, random_state
                    ])
```

In [221…
```
# Fit model with training data
rf.fit(X_train, y_train);
```

Model 1 Performance

In [222…
```
from sklearn.metrics import classification_report, roc_auc_score, accuracy_score
```

classification report (model 1 - Random Forest):

In [223…
```
y_pred_rf = rf.predict(X_test)
```

In [224…
```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1549
           1       0.63      0.45      0.53       561

    accuracy                           0.78      2110
   macro avg       0.73      0.68      0.69      2110
weighted avg       0.77      0.78      0.77      2110
```

ROC AUC(model 1 - Random Forest):

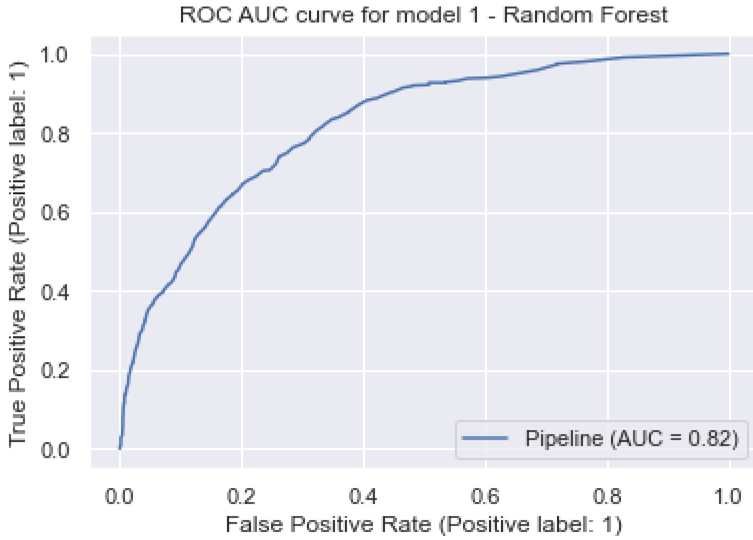
In [225…
```
y_prob_rf = rf.predict_proba(X_test)[:,1]
```

In [226…
```
print("model 1(Random Forest) ROC AUC score is:", roc_auc_score(y_test, y_prob_rf))
```

```
model 1(Random Forest) ROC AUC score is: 0.8195333888000883
```

In [231…
```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_estimator(rf, X_test, y_test)
plt.title("ROC AUC curve for model 1 - Random Forest")
```

Out[231]:
```
Text(0.5, 1.0, 'ROC AUC curve for model 1 - Random Forest')
```



Accuracy score and confusion matrix (model 1 - Random Forest):

In [228…

```
print("model 1(Random Forest) accuracy score is:", accuracy_score(y_test, y_pred_rf))
```

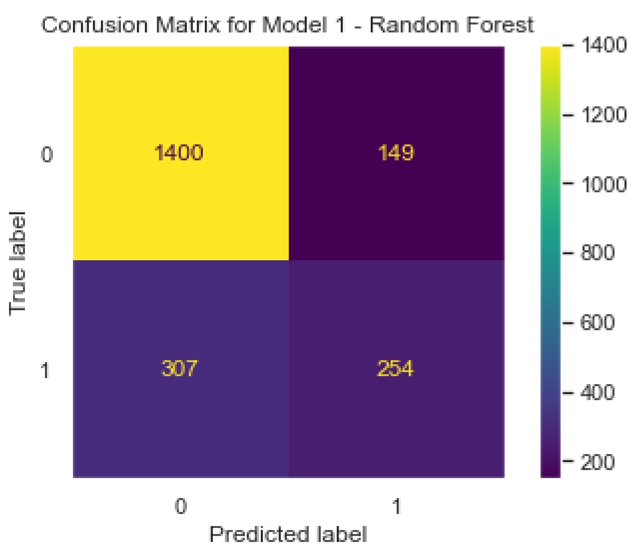
```
model 1(Random Forest) accuracy score is: 0.7838862559241706
```

In [232…

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test)
plt.grid(None)
plt.title("Confusion Matrix for Model 1 - Random Forest")
```

Out[232]:

```
Text(0.5, 1.0, 'Confusion Matrix for Model 1 - Random Forest')
```



Model 2 - Logistic Regression

In [140…

```
from sklearn.linear_model import LogisticRegression
```

In [142…

```
# instantiate container with pipeline steps
LR = Pipeline(steps=[
                    ('preprocessor', preprocessor),
                    ('classifier', LogisticRegression(class_weight="balanced"))
                    ])
```

In [143…

```
# Fit model with training data
LR.fit(X_train, y_train);
```

```
/Users/yichunchen/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Model 2 Performance

classification report (model 2 - Logistic Regression):

In [144…
```
y_pred_LR = LR.predict(X_test)
```

In [145…
```
print(classification_report(y_test, y_pred_LR))
```

```
              precision    recall  f1-score   support

           0       0.91      0.72      0.80      1549
           1       0.51      0.81      0.63       561

    accuracy                           0.74      2110
   macro avg       0.71      0.76      0.72      2110
weighted avg       0.81      0.74      0.76      2110
```

ROC AUC (model 2 - Logistic Regression):

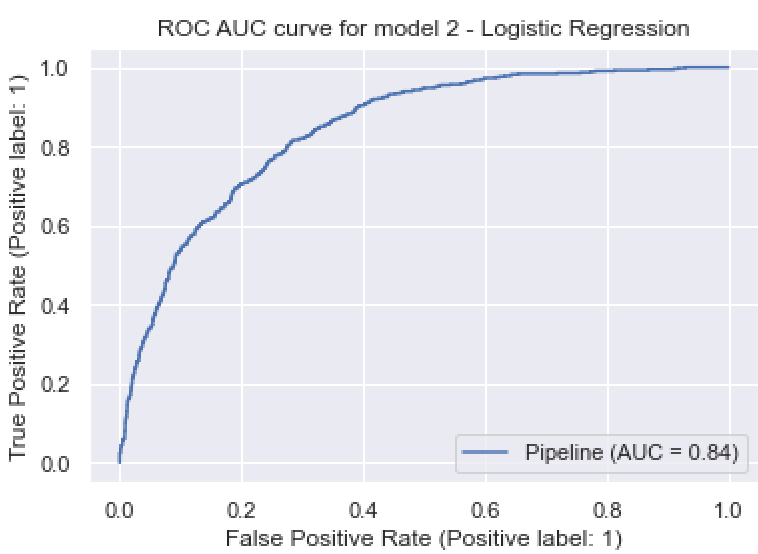
In [146…
```
y_prob_LR = LR.predict_proba(X_test)[:,1]
```

In [147…
```
print("model 2(Logistic Regression) ROC AUC score is:", roc_auc_score(y_test, y_prob_LR)
```

```
model 2(Logistic Regression) ROC AUC score is: 0.8424007668681651
```

In [233…
```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_estimator(LR, X_test, y_test)
plt.title("ROC AUC curve for model 2 - Logistic Regression")
```

Out[233]:
```
Text(0.5, 1.0, 'ROC AUC curve for model 2 - Logistic Regression')
```



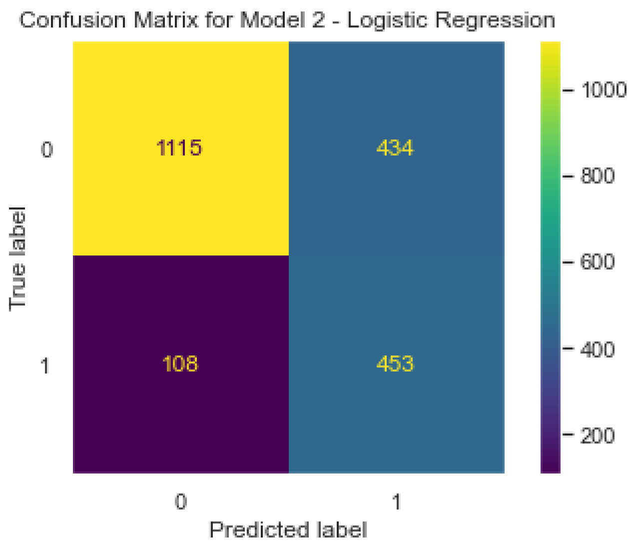
Accuracy score and confusion matrix (model 2 - Logistic Regression):

In [157…
```
print("model 2(Logistic Regression) accuracy score is:", accuracy_score(y_test, y_pred_L
```

```
model 2(Logistic Regression) accuracy score is: 0.7431279620853081
```

In [234…
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(LR, X_test, y_test)
plt.grid(None)
plt.title("Confusion Matrix for Model 2 - Logistic Regression")
```

Out[234]:
```
Text(0.5, 1.0, 'Confusion Matrix for Model 2 - Logistic Regression')
```



7. Feature Importance

Model 1 - Random Forest:

In [165…
```
features = df.drop(columns=["Churn"]).columns
```

In [41]:
```
weights = list(rf.feature_importances_)
```

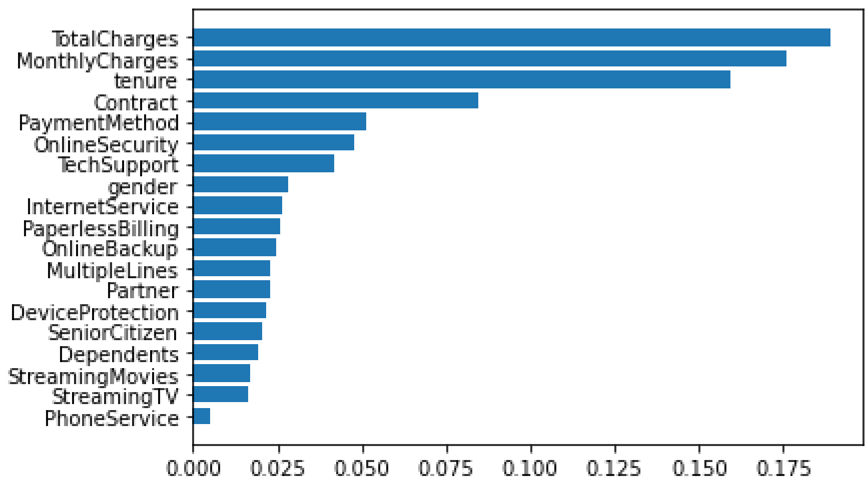
In [42]:
```
feat_imp = sorted(list(zip(features, weights)), key=lambda x:x[1])
```

In [43]:
```
yticks, heights = zip(*feat_imp)
```

In [44]:
```
plt.barh(range(len(heights)), heights)
plt.yticks(range(len(heights)), yticks)
```

Out[44]:
```
([<matplotlib.axis.YTick at 0x1251bf040>,
  <matplotlib.axis.YTick at 0x1251a49a0>,
  <matplotlib.axis.YTick at 0x125198ac0>,
  <matplotlib.axis.YTick at 0x1211622b0>,
  <matplotlib.axis.YTick at 0x121162a00>,
  <matplotlib.axis.YTick at 0x12116e190>,
  <matplotlib.axis.YTick at 0x121162af0>,
  <matplotlib.axis.YTick at 0x12116e7c0>,
  <matplotlib.axis.YTick at 0x121176070>,
  <matplotlib.axis.YTick at 0x1211766a0>,
  <matplotlib.axis.YTick at 0x121176df0>,
  <matplotlib.axis.YTick at 0x12117b580>,
  <matplotlib.axis.YTick at 0x121176e50>,
  <matplotlib.axis.YTick at 0x12116eca0>,
  <matplotlib.axis.YTick at 0x12117b850>,
  <matplotlib.axis.YTick at 0x1211832b0>,
  <matplotlib.axis.YTick at 0x121183a00>,
```

```
 <matplotlib.axis.YTick at 0x121189190>,
 <matplotlib.axis.YTick at 0x1211898e0>],
[Text(0, 0, 'PhoneService'),
 Text(0, 1, 'StreamingTV'),
 Text(0, 2, 'StreamingMovies'),
 Text(0, 3, 'Dependents'),
 Text(0, 4, 'SeniorCitizen'),
 Text(0, 5, 'DeviceProtection'),
 Text(0, 6, 'Partner'),
 Text(0, 7, 'MultipleLines'),
 Text(0, 8, 'OnlineBackup'),
 Text(0, 9, 'PaperlessBilling'),
 Text(0, 10, 'InternetService'),
 Text(0, 11, 'gender'),
 Text(0, 12, 'TechSupport'),
 Text(0, 13, 'OnlineSecurity'),
 Text(0, 14, 'PaymentMethod'),
 Text(0, 15, 'Contract'),
 Text(0, 16, 'tenure'),
 Text(0, 17, 'MonthlyCharges'),
 Text(0, 18, 'TotalCharges')])
```



The top 3 important features in the random forest model are: TotalCharges, MonthlyCharges and tenure.

Model 2 - Logistic Regression:

In [177…
```
features1 = df.drop(columns=["Churn"]).columns
```

In [194…
```
weights1 = list(LR.named_steps['classifier'].coef_[0])
```

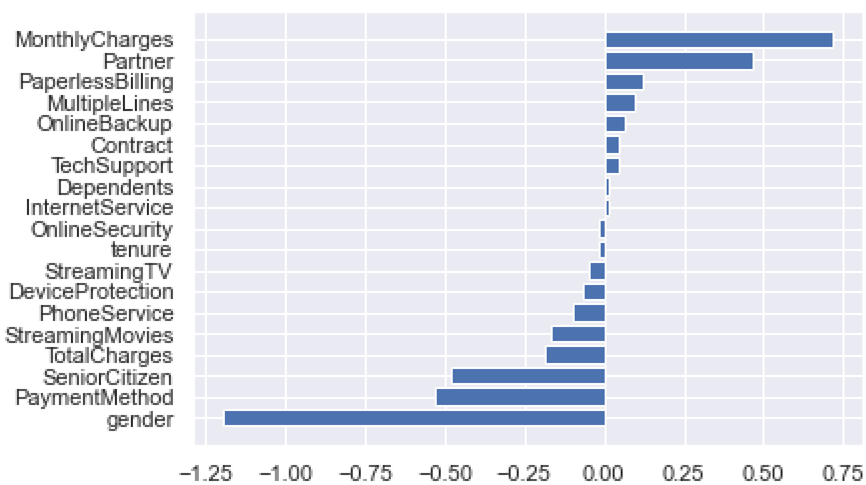
In [195…
```
feat_imp1 = sorted(list(zip(features1, weights1)), key=lambda x:x[1])
```

In [196…
```
yticks, heights = zip(*feat_imp1)
```

In [197…
```
plt.barh(range(len(heights)), heights)
plt.yticks(range(len(heights)), yticks)
```

Out[197]:
```
([<matplotlib.axis.YTick at 0x1327ea850>,
  <matplotlib.axis.YTick at 0x1327ea0d0>,
  <matplotlib.axis.YTick at 0x1327f37f0>,
  <matplotlib.axis.YTick at 0x132dcf790>,
  <matplotlib.axis.YTick at 0x132dcfc10>,
  <matplotlib.axis.YTick at 0x132dd53a0>,
  <matplotlib.axis.YTick at 0x132dd5af0>,
  <matplotlib.axis.YTick at 0x132dda280>,
```

```
  <matplotlib.axis.YTick at 0x132dda9d0>,
  <matplotlib.axis.YTick at 0x132ddf160>,
  <matplotlib.axis.YTick at 0x132ddac70>,
  <matplotlib.axis.YTick at 0x132dd5880>,
  <matplotlib.axis.YTick at 0x132ddf790>,
  <matplotlib.axis.YTick at 0x132de5040>,
  <matplotlib.axis.YTick at 0x132de5670>,
  <matplotlib.axis.YTick at 0x132de5dc0>,
  <matplotlib.axis.YTick at 0x132deb550>,
  <matplotlib.axis.YTick at 0x132de5910>,
  <matplotlib.axis.YTick at 0x132dd5100>],
 [Text(0, 0, 'gender'),
  Text(0, 1, 'PaymentMethod'),
  Text(0, 2, 'SeniorCitizen'),
  Text(0, 3, 'TotalCharges'),
  Text(0, 4, 'StreamingMovies'),
  Text(0, 5, 'PhoneService'),
  Text(0, 6, 'DeviceProtection'),
  Text(0, 7, 'StreamingTV'),
  Text(0, 8, 'tenure'),
  Text(0, 9, 'OnlineSecurity'),
  Text(0, 10, 'InternetService'),
  Text(0, 11, 'Dependents'),
  Text(0, 12, 'TechSupport'),
  Text(0, 13, 'Contract'),
  Text(0, 14, 'OnlineBackup'),
  Text(0, 15, 'MultipleLines'),
  Text(0, 16, 'PaperlessBilling'),
  Text(0, 17, 'Partner'),
  Text(0, 18, 'MonthlyCharges')])
```



The top 3 important features in the Logistic Regression model are: MonthlyCharges, Partner, and PaperlessBilling.