Workflow Steps

- In this project, we analyzed customer data for a telecommunications company.
- We started by merging three tables together for analysis.
- We checked for errors in each table and corrected them.
- Then we re-merged the tables to obtain one complete dataset.
- We analyzed each table separately and created visualizations.
- After that, we performed some combined analysis across the tables.
- We converted the textual data into numerical form for machine learning.
- We studied the relationships between columns and how they impact customer churn.
- We built a prediction model to identify customers who are likely to churn.
- Finally, we saved the model for use in another environment.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
color_pal=sns.color_palette()
import plotly.express as px

from scipy.stats import norm
from scipy.stats import wilcoxon
from scipy.stats import kruskal
import scipy.stats as stats
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')
```

Fix some rare problemes

In [2]:
```
path = r'C:\\Users\\youne\\OneDrive\\Bureau\\data analys\\training\\telecom custome
print(repr(path))
print([hex(ord(c)) for c in path[:5]])
```

```
'\u202aC:\\\\Users\\\\youne\\\\OneDrive\\\\Bureau\\\\data analys\\\\training\\\\tele
com customer churn\\\\churn_data.csv'
['0x202a', '0x43', '0x3a', '0x5c', '0x5c']
```

In [3]:
```
raw_path = '\u202aC:\\Users\\youne\\OneDrive\\Bureau\\data analys\\training\\teleco
print("raw_path repr:", repr(raw_path))

cleaned_path = raw_path.lstrip('\u202a')
print("cleaned_path repr:", repr(cleaned_path))
print("Unicode code points:", [hex(ord(c)) for c in cleaned_path[:3]])
```

```
raw_path repr: '\u202aC:\\Users\\youne\\OneDrive\\Bureau\\data analys\\training\\tel
ecom customer churn\\churn_data.csv'
cleaned_path repr: 'C:\\Users\\youne\\OneDrive\\Bureau\\data analys\\training\\telec
om customer churn\\churn_data.csv'
Unicode code points: ['0x43', '0x3a', '0x5c']
```

In [4]:
```
df1=pd.read_csv(r'C:\Users\youne\OneDrive\Bureau\data analys\training\telecom custo
df2=pd.read_csv(r'C:\Users\youne\OneDrive\Bureau\data analys\training\telecom custo
df3 = pd.read_csv(cleaned_path)
```

data explor----------

In [5]:
```
# Merging the three DataFrames
df12 = pd.merge(df1, df2, on='customerID', how='inner')
df = pd.merge(df12, df3, on='customerID', how='inner')
```

In [6]:
```
pd.set_option('display.max_columns', None)
df.head()
```

Out[6]:

	customerID	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtect
0	7590-VHVEG	No phone service	DSL	No	Yes	
1	5575-GNVDE	No	DSL	Yes	No	
2	3668-QPYBK	No	DSL	Yes	Yes	
3	7795-CFOCW	No phone service	DSL	Yes	No	
4	9237-HQITU	No	Fiber optic	No	No	

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7042 entries, 0 to 7041
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7042 non-null   object 
 1   MultipleLines     7042 non-null   object 
 2   InternetService   7042 non-null   object 
 3   OnlineSecurity    7042 non-null   object 
 4   OnlineBackup      7042 non-null   object 
 5   DeviceProtection  7042 non-null   object 
 6   TechSupport       7042 non-null   object 
 7   StreamingTV       7042 non-null   object 
 8   StreamingMovies   7042 non-null   object 
 9   gender            7042 non-null   object 
 10  SeniorCitizen     7042 non-null   int64  
 11  Partner           7042 non-null   object 
 12  Dependents        7042 non-null   object 
 13  tenure            7042 non-null   int64  
 14  PhoneService      7042 non-null   object 
 15  Contract          7042 non-null   object 
 16  PaperlessBilling  7042 non-null   object 
 17  PaymentMethod     7042 non-null   object 
 18  MonthlyCharges    7042 non-null   float64
 19  TotalCharges      7042 non-null   object 
 20  Churn             7042 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

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

Out[8]:
```
customerID          0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [9]:
```
df.duplicated().sum()
```

Out[9]: np.int64(0)

data celanning ----------

In [10]:
```
# Standardizing Column Names to Lowercase
df1.columns = df1.columns.str.lower()
df2.columns = df2.columns.str.lower()
df3.columns = df3.columns.str.lower()
```

In [11]:
```
print(df3.columns)
```

```
Index(['customerid', 'tenure', 'phoneservice', 'contract', 'paperlessbilling',
       'paymentmethod', 'monthlycharges', 'totalcharges', 'churn'],
      dtype='object')
```

In [ ]:

In [12]:
```
# transform  No phone service to 'No
df1['multiplelines'] = df1['multiplelines'].replace('No phone service', 'No')
```

In [13]:
```
target_ids = [
    '4472-LVYGI',
    '3115-CZMZD',
    '5709-LVOEQ',
    '4367-NUYAO',
    '1371-DWPAZ',
    '7644-OMVMY',
    '3213-VVOLG',
    '2520-SGTTA',
    '2923-ARZLG',
    '4075-WKNIU',
    '2775-SEFEE']

df3[df3['customerid'].isin(target_ids)]
```

Out[13]:

	customerid	tenure	phoneservice	contract	paperlessbilling	paymentmethod	mont
488	4472-LVYGI	0	No	Two year	Yes	Bank transfer (automatic)	
753	3115-CZMZD	0	Yes	Two year	No	Mailed check	
936	5709-LVOEQ	0	Yes	Two year	No	Mailed check	
1082	4367-NUYAO	0	Yes	Two year	No	Mailed check	
1340	1371-DWPAZ	0	No	Two year	No	Credit card (automatic)	
3331	7644-OMVMY	0	Yes	Two year	No	Mailed check	
3826	3213-VVOLG	0	Yes	Two year	No	Mailed check	
4380	2520-SGTTA	0	Yes	Two year	No	Mailed check	
5218	2923-ARZLG	0	Yes	One year	Yes	Mailed check	
6670	4075-WKNIU	0	Yes	Two year	No	Mailed check	
6754	2775-SEFEE	0	Yes	Two year	Yes	Bank transfer (automatic)	

In [14]:
```
# chan dtype for totalcharges from object to float64
converted = df3['totalcharges'] = pd.to_numeric(df3['totalcharges'], errors='coerce
```

In [15]:
```
# Filling missing values with the mean
df3['totalcharges'] = df3['totalcharges'].fillna(df3['totalcharges'].mean())
```

In [16]:
```
# add churn column in all dataframes
df1['churn'] = df1['customerid'].map(df3.set_index('customerid')['churn'])
df2['churn'] = df2['customerid'].map(df3.set_index('customerid')['churn'])
```

In [17]:
```
# Merging the three DataFrames again
df12 = pd.merge(df1, df2, on='customerid', how='inner')
df = pd.merge(df12, df3, on='customerid', how='inner')
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7042 entries, 0 to 7041
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerid        7042 non-null   object
 1   multiplelines     7042 non-null   object
 2   internetservice   7042 non-null   object
 3   onlinesecurity    7042 non-null   object
 4   onlinebackup      7042 non-null   object
 5   deviceprotection  7042 non-null   object
 6   techsupport       7042 non-null   object
 7   streamingtv       7042 non-null   object
 8   streamingmovies   7042 non-null   object
 9   churn_x           7042 non-null   object
 10  gender            7042 non-null   object
 11  seniorcitizen     7042 non-null   int64
 12  partner           7042 non-null   object
 13  dependents        7042 non-null   object
 14  churn_y           7042 non-null   object
 15  tenure            7042 non-null   int64
 16  phoneservice      7042 non-null   object
 17  contract          7042 non-null   object
 18  paperlessbilling  7042 non-null   object
 19  paymentmethod     7042 non-null   object
 20  monthlycharges    7042 non-null   float64
 21  totalcharges      7042 non-null   float64
 22  churn             7042 non-null   object
dtypes: float64(2), int64(2), object(19)
memory usage: 1.2+ MB
```

data analysis----------

internet data analysis

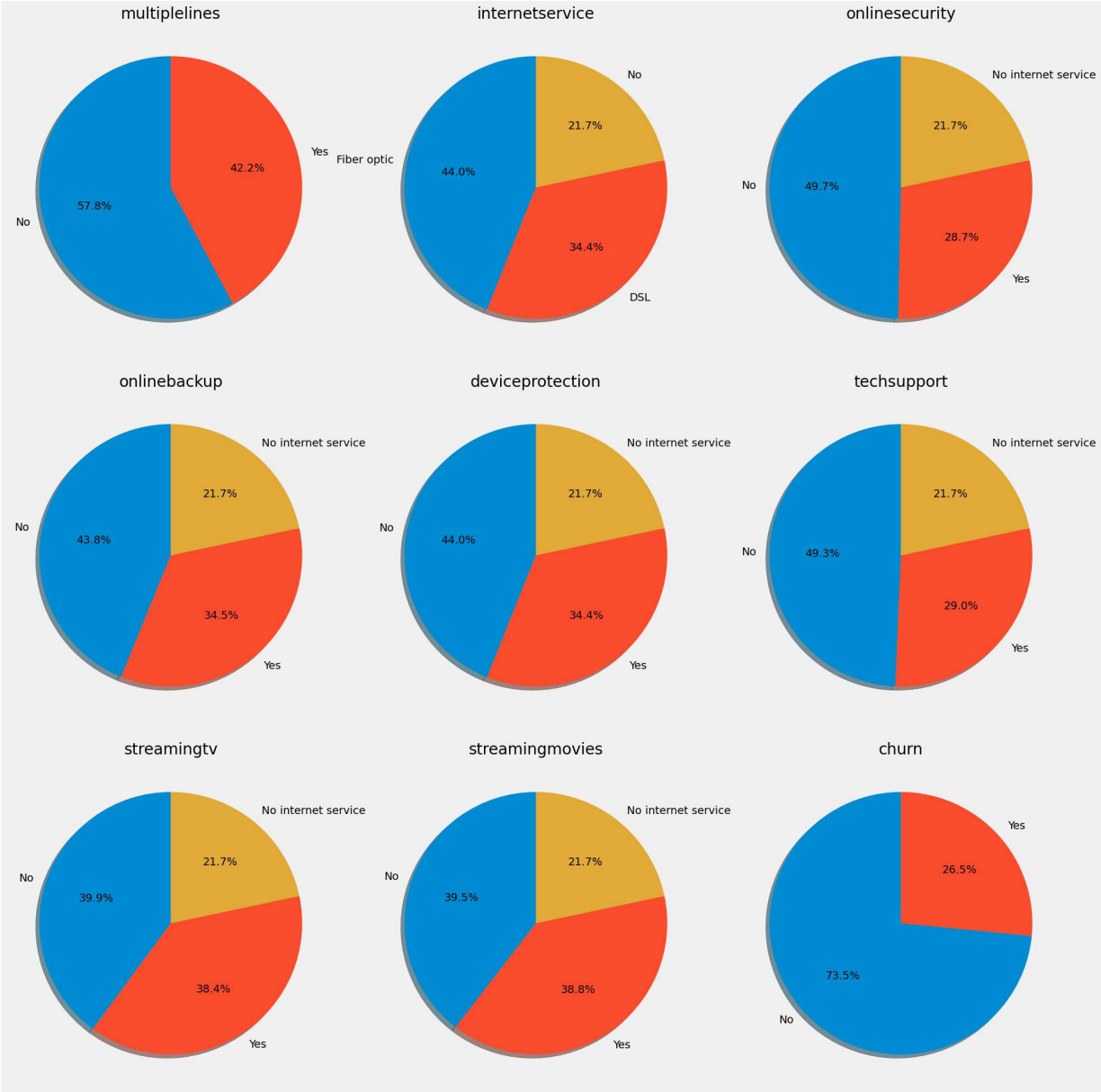
In [19]:
```
df1.head()
```

Out[19]:

	customerid	multiplelines	internetservice	onlinesecurity	onlinebackup	deviceprotectior
0	7590-VHVEG	No	DSL	No	Yes	No
1	5575-GNVDE	No	DSL	Yes	No	Yes
2	3668-QPYBK	No	DSL	Yes	Yes	No
3	7795-CFOCW	No	DSL	Yes	No	Yes
4	9237-HQITU	No	Fiber optic	No	No	No

In [20]:

```
cols = [ 'multiplelines', 'internetservice', 'onlinesecurity', 'onlinebackup', 'dev
        'streamingmovies', 'churn']

plt.figure(figsize=(20, 20))

for i, col in enumerate(cols, 1):
    plt.subplot(3, 3, i)
    df1[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, shadow=True)
    plt.title(col)
    plt.ylabel('')

plt.tight_layout()
plt.show()
```

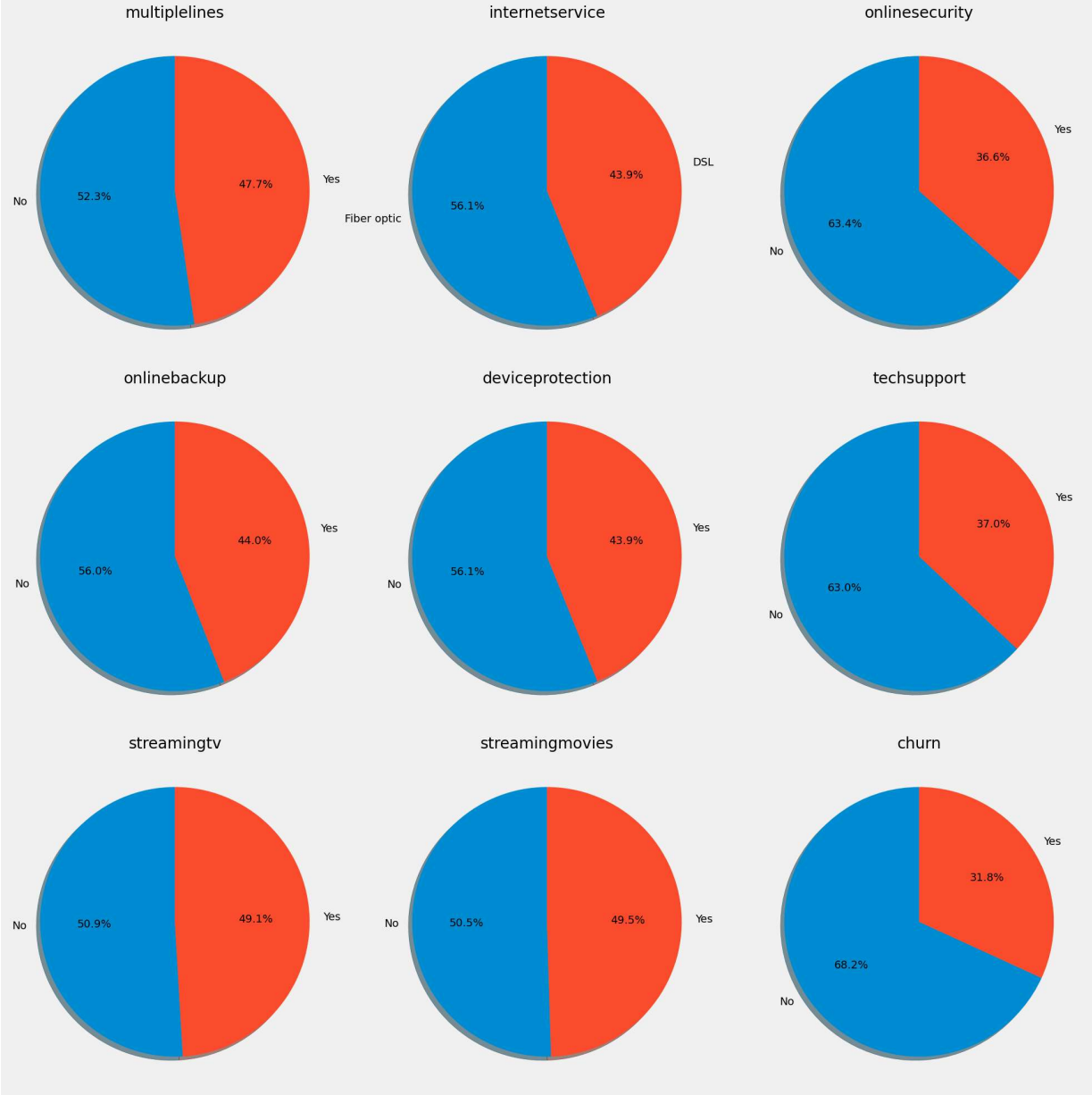


we have 21.7% from customer they don't use this service in the first place

In [21]:
```
cols_to_check = ['onlinesecurity','onlinebackup','deviceprotection','techsupport','
df1 = df1[~df1[cols_to_check].isin(['No internet service']).any(axis=1)]
df1 = df1[df1['internetservice'] != 'No']
```

In [22]:
```
cols = [ 'multiplelines', 'internetservice', 'onlinesecurity', 'onlinebackup', 'dev
       'streamingmovies', 'churn']

plt.figure(figsize=(20, 20))

for i, col in enumerate(cols, 1):
    plt.subplot(3, 3, i)
    df1[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, shadow=True)
    plt.title(col)
    plt.ylabel('')
```

```
plt.tight_layout()
plt.show()
```



The percentage of people who left while subscribed to the internet service has increased by 10%

Calculate the probability of churn with the remaining variables.

In [24]:
```
cols = ['multiplelines','onlinesecurity', 'onlinebackup', 'deviceprotection', 'tech

churn_df = df1[df1['churn'] == 'Yes']

for col in cols:
    total_churn = len(churn_df)
```

```
    count_yes = len(churn_df[churn_df[col] == 'Yes'])
    prob_yes = count_yes / total_churn if total_churn > 0 else 0

    count_no = len(churn_df[churn_df[col] == 'No'])
    prob_no = count_no / total_churn if total_churn > 0 else 0

    print(f"Column: {col}")
    print(f"  Probability of {col} = 'Yes' among churned customers: {prob_yes:.4f}")
    print(f"  Probability of {col} = 'No' among churned customers: {prob_no:.4f}\n")
```

```
Column: multiplelines
  Probability of multiplelines = 'Yes' among churned customers: 0.4784
  Probability of multiplelines = 'No' among churned customers: 0.5216

Column: onlinesecurity
  Probability of onlinesecurity = 'Yes' among churned customers: 0.1680
  Probability of onlinesecurity = 'No' among churned customers: 0.8320

Column: onlinebackup
  Probability of onlinebackup = 'Yes' among churned customers: 0.2978
  Probability of onlinebackup = 'No' among churned customers: 0.7022

Column: deviceprotection
  Probability of deviceprotection = 'Yes' among churned customers: 0.3104
  Probability of deviceprotection = 'No' among churned customers: 0.6896

Column: techsupport
  Probability of techsupport = 'Yes' among churned customers: 0.1765
  Probability of techsupport = 'No' among churned customers: 0.8235

Column: streamingtv
  Probability of streamingtv = 'Yes' among churned customers: 0.4636
  Probability of streamingtv = 'No' among churned customers: 0.5364

Column: streamingmovies
  Probability of streamingmovies = 'Yes' among churned customers: 0.4658
  Probability of streamingmovies = 'No' among churned customers: 0.5342
```

In [25]:
```
for service in df1['internetservice'].unique():
    total = len(df1[df1['internetservice'] == service])
    churn_yes = len(df1[(df1['internetservice'] == service) & (df1['churn'] == 'Yes
    pro = churn_yes / total if total > 0 else 0
    print(f"Probability of churn for customers with internet service '{service}': {
```

```
Probability of churn for customers with internet service 'DSL': 0.1896
Probability of churn for customers with internet service 'Fiber optic': 0.4191
```

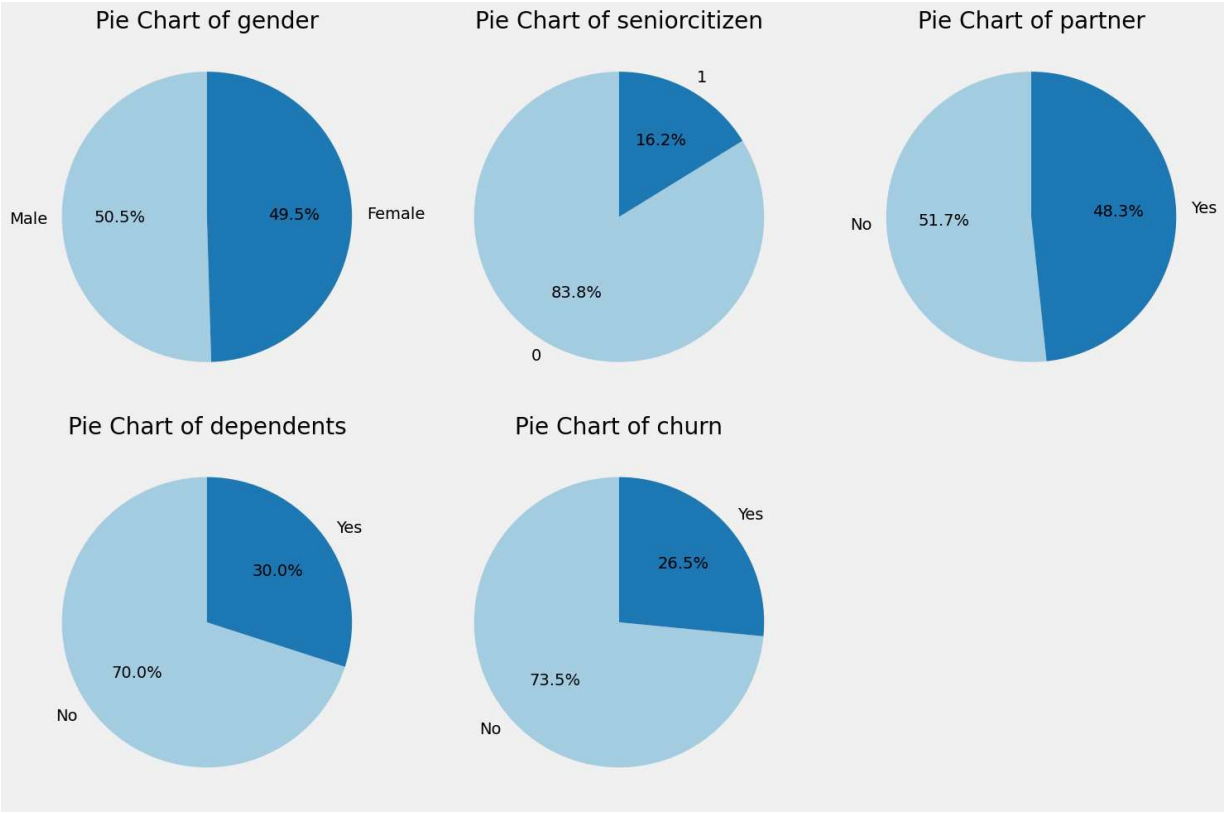
customer data analysis

In [27]:
```
df2.head()
```

Out[27]:

	customerid	gender	seniorcitizen	partner	dependents	churn
0	7590-VHVEG	Female	0	Yes	No	No
1	5575-GNVDE	Male	0	No	No	No
2	3668-QPYBK	Male	0	No	No	Yes
3	7795-CFOCW	Male	0	No	No	No
4	9237-HQITU	Female	0	No	No	Yes

In [28]:
```
columns = ['gender', 'seniorcitizen', 'partner', 'dependents', 'churn']

plt.figure(figsize=(15, 10))

for i, col in enumerate(columns, 1):
    plt.subplot(2, 3, i)
    df2[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=plt.c
    plt.title(f'Pie Chart of {col}')
    plt.ylabel('')

plt.tight_layout()
plt.show()
```



churn data analysis

In [31]:
```
df3.head()
```

Out[31]:

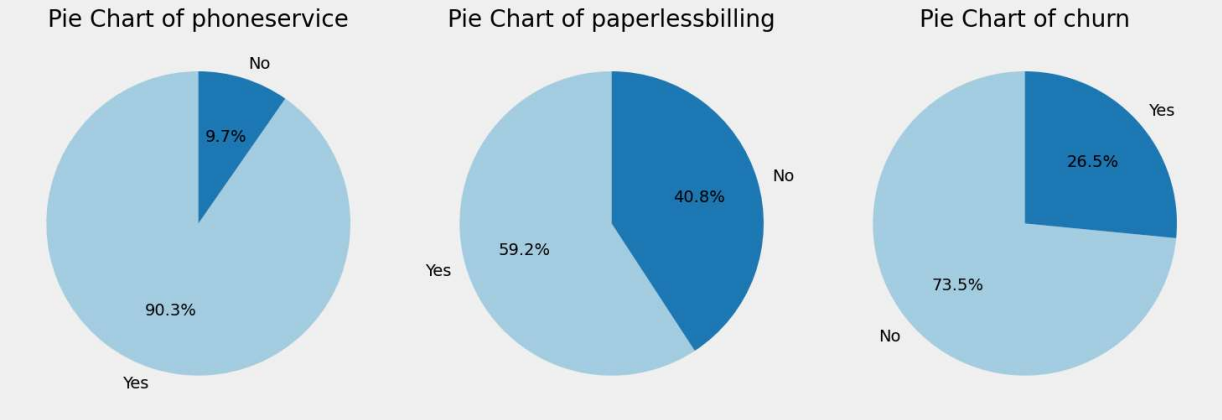
	customerid	tenure	phoneservice	contract	paperlessbilling	paymentmethod	monthlyc
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	
1	5575-GNVDE	34	Yes	One year	No	Mailed check	
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	

In [32]:
```
df3.describe()
```

Out[32]:

	tenure	monthlycharges	totalcharges
count	7042.000000	7042.000000	7042.000000
mean	32.366373	64.755886	2282.651714
std	24.557955	30.088238	2264.508688
min	0.000000	18.250000	18.800000
25%	9.000000	35.500000	402.087500
50%	29.000000	70.350000	1400.425000
75%	55.000000	89.850000	3783.600000
max	72.000000	118.750000	8684.800000

In [33]:
```
columns = ['phoneservice', 'paperlessbilling', 'churn']

plt.figure(figsize=(15, 10))

for i, col in enumerate(columns, 1):
    plt.subplot(2, 3, i)
    df3[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=plt.c
    plt.title(f'Pie Chart of {col}')
    plt.ylabel('')

plt.tight_layout()
plt.show()
```

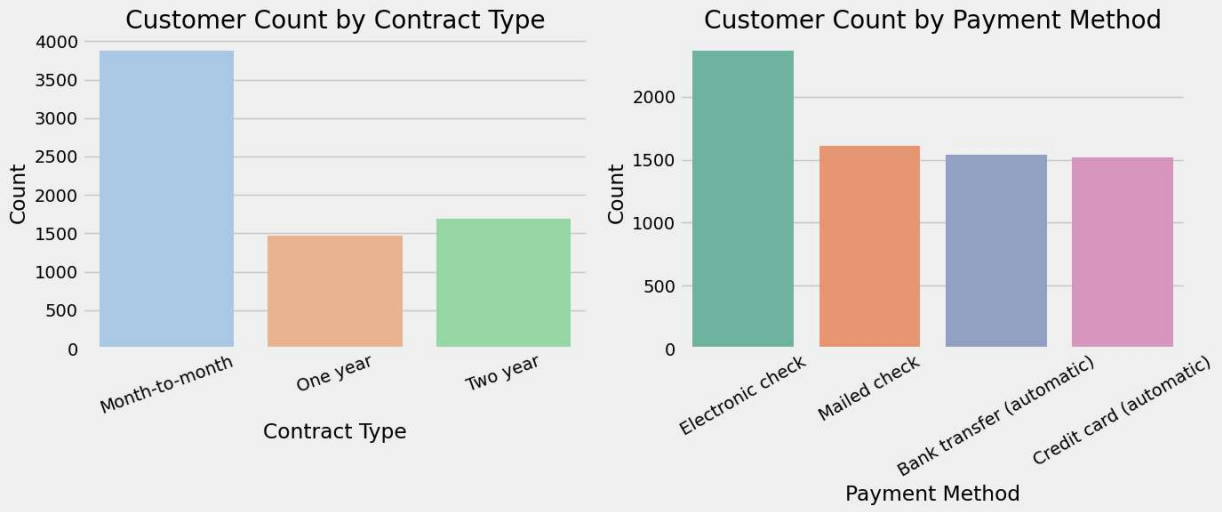


In [35]:
```
churn_df = df3[df3['churn'] == 'Yes']


cols = ['phoneservice', 'paperlessbilling']


for col in cols:
    print(f"Probabilities of {col} values among churned customers:")
    total_churn = len(churn_df)
    for val in churn_df[col].unique():
        count = len(churn_df[churn_df[col] == val])
        pro = count / total_churn if total_churn > 0 else 0
        print(f"  {val}: {pro:.4f}")
    print()
```

```
Probabilities of phoneservice values among churned customers:
  Yes: 0.9090
  No: 0.0910

Probabilities of paperlessbilling values among churned customers:
  Yes: 0.7491
  No: 0.2509
```

In [36]:
```
plt.figure(figsize=(14, 6))


plt.subplot(1, 2, 1)
sns.countplot(data=df3, x='contract', palette='pastel')
plt.title('Customer Count by Contract Type')
plt.xlabel('Contract Type')
plt.ylabel('Count')
plt.xticks(rotation=20)


plt.subplot(1, 2, 2)
sns.countplot(data=df3, x='paymentmethod', palette='Set2')
plt.title('Customer Count by Payment Method')
plt.xlabel('Payment Method')
plt.ylabel('Count')
plt.xticks(rotation=30)
```

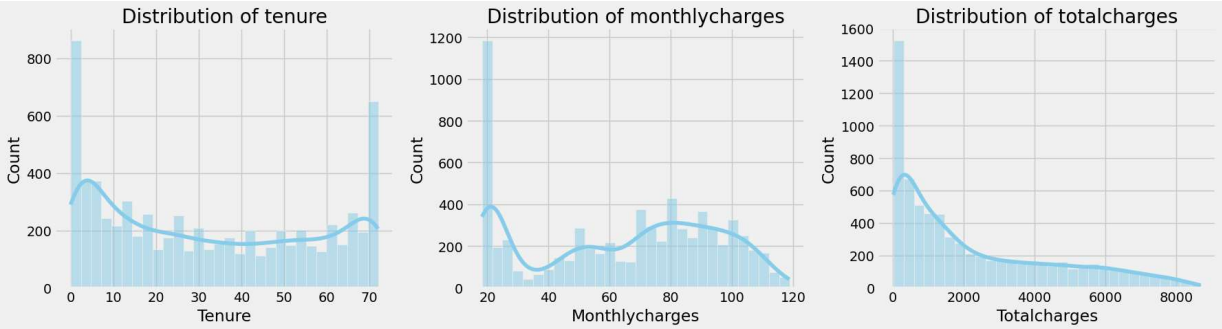
```
plt.tight_layout()
plt.show()
```



In [38]:
```
churn_df = df3[df3['churn'] == 'Yes']

for contract_type in df3['contract'].unique():
    count = len(churn_df[churn_df['contract'] == contract_type])
    total_churn = len(churn_df)
    prob = count / total_churn if total_churn > 0 else 0
    print(f"Among churned customers, probability of contract type '{contract_type}'
```

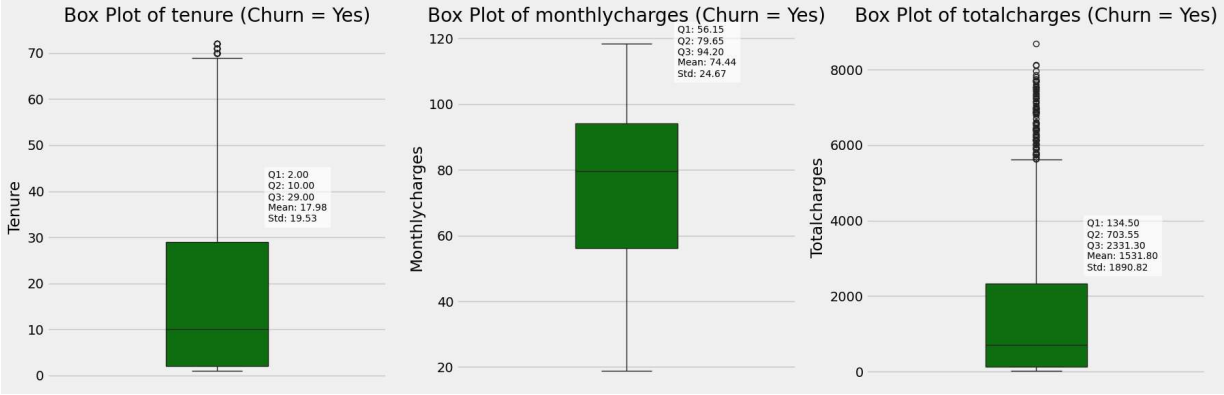
```
Among churned customers, probability of contract type 'Month-to-month': 0.8855
Among churned customers, probability of contract type 'One year': 0.0888
Among churned customers, probability of contract type 'Two year': 0.0257
```

In [39]:
```
numeric_cols = ['tenure', 'monthlycharges', 'totalcharges']

plt.figure(figsize=(18, 5))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(1, 3, i)
    sns.histplot(data=df3, x=col, kde=True, bins=30, color='skyblue')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```

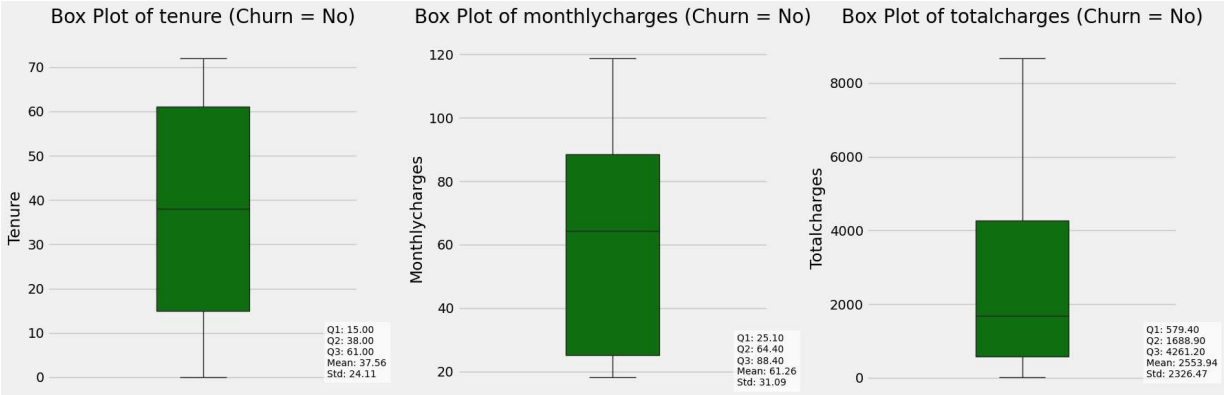


In [ ]:
```
numeric_cols = ['tenure', 'monthlycharges', 'totalcharges']
plt.figure(figsize=(18, 6))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(data=df3, y=col, color='green', width=0.3)


    q1 = df3[col].quantile(0.25)
    q2 = df3[col].quantile(0.50)
    q3 = df3[col].quantile(0.75)
    mean = df3[col].mean()
    std = df3[col].std()


    stats_text = f"Q1: {q1:.2f}\nQ2: {q2:.2f}\nQ3: {q3:.2f}\nMean: {mean:.2f}\nStd:
    plt.text(0.15, q3 + (q3*0.15), stats_text, fontsize=10, bbox=dict(facecolor='wh

    plt.title(f'Box Plot of {col}')
    plt.ylabel(col.capitalize())

plt.tight_layout()
plt.show()
```

In [41]:
```
numeric_cols = ['tenure', 'monthlycharges', 'totalcharges']


churned_df = df3[df3['churn'] == 'Yes']

plt.figure(figsize=(18, 6))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(data=churned_df, y=col, color='green', width=0.3)

    q1 = churned_df[col].quantile(0.25)
    q2 = churned_df[col].quantile(0.50)
    q3 = churned_df[col].quantile(0.75)
    mean = churned_df[col].mean()
    std = churned_df[col].std()

    stats_text = f"Q1: {q1:.2f}\nQ2: {q2:.2f}\nQ3: {q3:.2f}\nMean: {mean:.2f}\nStd:
    plt.text(0.15, q3 + (q3*0.15), stats_text, fontsize=10, bbox=dict(facecolor='wh

    plt.title(f'Box Plot of {col} (Churn = Yes)')
    plt.ylabel(col.capitalize())

plt.tight_layout()
plt.show()
```

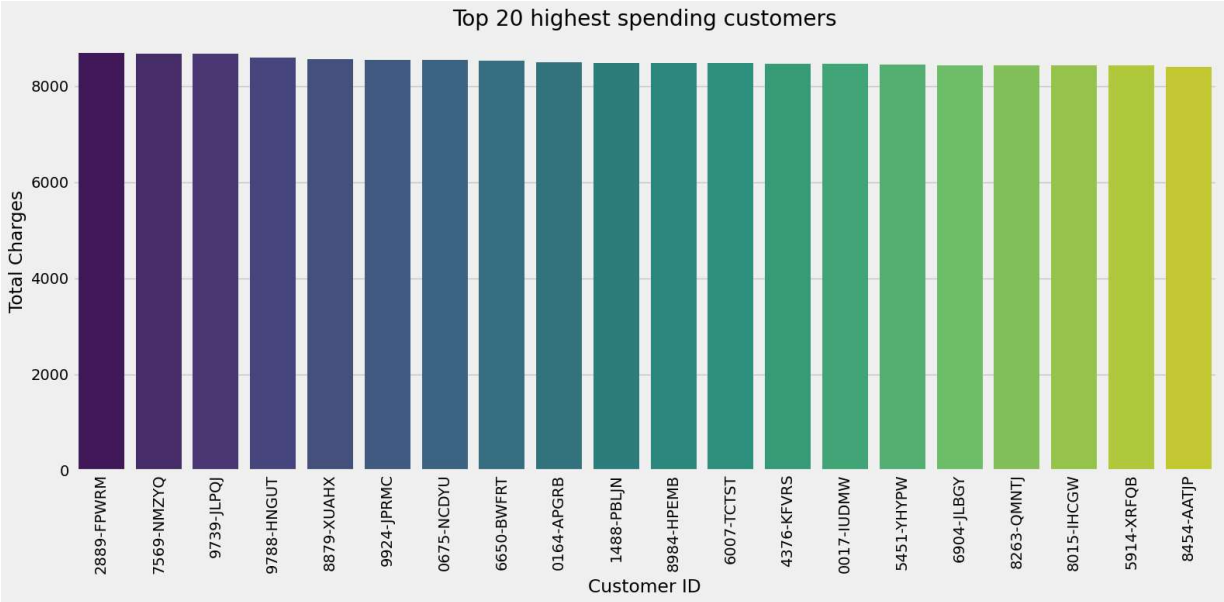


In [42]:
```
numeric_cols = ['tenure', 'monthlycharges', 'totalcharges']
churned_df = df3[df3['churn'] == 'No']

plt.figure(figsize=(18, 6))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(data=churned_df, y=col, color='green', width=0.3)

    q1 = churned_df[col].quantile(0.25)
    q2 = churned_df[col].quantile(0.50)
    q3 = churned_df[col].quantile(0.75)
    mean = churned_df[col].mean()
    std = churned_df[col].std()
    ymin = churned_df[col].min()

    stats_text = f"Q1: {q1:.2f}\nQ2: {q2:.2f}\nQ3: {q3:.2f}\nMean: {mean:.2f}\nStd:
    plt.text(0.4, ymin - (0.15 * abs(ymin)), stats_text, fontsize=10, bbox=dict(fac

    plt.title(f'Box Plot of {col} (Churn = No)', pad=20)
    plt.ylabel(col.capitalize())

plt.tight_layout()
plt.show()
```



In [43]:
```
top20 = df3.nlargest(20, 'totalcharges')

plt.figure(figsize=(16,8))
sns.barplot(x='customerid', y='totalcharges', data=top20, palette='viridis')
plt.title('Top 20 highest spending customers')
plt.xlabel('Customer ID')
```

```
plt.ylabel('Total Charges')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



encoding

In [44]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [45]:
```
print(df.columns)
```

```
Index(['customerid', 'multiplelines', 'internetservice', 'onlinesecurity',
       'onlinebackup', 'deviceprotection', 'techsupport', 'streamingtv',
       'streamingmovies', 'churn_x', 'gender', 'seniorcitizen', 'partner',
       'dependents', 'churn_y', 'tenure', 'phoneservice', 'contract',
       'paperlessbilling', 'paymentmethod', 'monthlycharges', 'totalcharges',
       'churn'],
      dtype='object')
```

In [46]:
```
print(df1.columns)
```

```
Index(['customerid', 'multiplelines', 'internetservice', 'onlinesecurity',
       'onlinebackup', 'deviceprotection', 'techsupport', 'streamingtv',
       'streamingmovies', 'churn'],
      dtype='object')
```

In [47]:
```
cols_to_clean = ['customerid', 'multiplelines', 'internetservice', 'onlinesecurity'
 'onlinebackup', 'deviceprotection', 'techsupport', 'streamingtv', 'streamingmovies
for col in cols_to_clean:
    df[col] = df[col].replace('No internet service', 'No')
```

In [48]:
```
for col in df.columns:
    if df[col].isin(['Yes', 'No']).all():
        df[col] = df[col].map({'Yes': 1, 'No': 0})
```

In [49]:
```
dummies = pd.get_dummies(df['internetservice'], drop_first=False).astype(int)
```

In [50]:
```
df = pd.concat([df.drop('internetservice', axis=1), dummies], axis=1)
```

In [51]:
```
df['gender'] = df['gender'].map({'Male': 1, 'Female': 0})
```

In [52]:
```
df['contract'] = le.fit_transform(df['contract'])
```

In [53]:
```
df = pd.get_dummies(df, columns=['paymentmethod'], prefix='paymentmethod', dtype=in
```

In [54]:
```
df.head()
```

Out[54]:

	customerid	multiplelines	onlinesecurity	onlinebackup	deviceprotection	techsupport
0	7590-VHVEG	0	0	1	0	0
1	5575-GNVDE	0	1	0	1	0
2	3668-QPYBK	0	1	1	0	0
3	7795-CFOCW	0	1	0	1	1
4	9237-HQITU	0	0	0	0	0

In [ ]:
```

```

In [55]:
```
df.isnull().sum()
```

Out[55]:
```
customerid                                0
multiplelines                             0
onlinesecurity                            0
onlinebackup                              0
deviceprotection                          0
techsupport                               0
streamingtv                               0
streamingmovies                           0
churn_x                                   0
gender                                    0
seniorcitizen                             0
partner                                   0
dependents                                0
churn_y                                   0
tenure                                    0
phoneservice                              0
contract                                  0
paperlessbilling                          0
monthlycharges                            0
totalcharges                              0
churn                                     0
DSL                                       0
Fiber optic                               0
No                                        0
paymentmethod_Bank transfer (automatic)   0
paymentmethod_Credit card (automatic)     0
paymentmethod_Electronic check            0
paymentmethod_Mailed check                0
dtype: int64
```

global analysis

In [56]:
```
df = df.drop(['customerid', 'churn_x', 'churn_y'], axis=1, errors='ignore')
```

In [57]:
```
df.head()
```

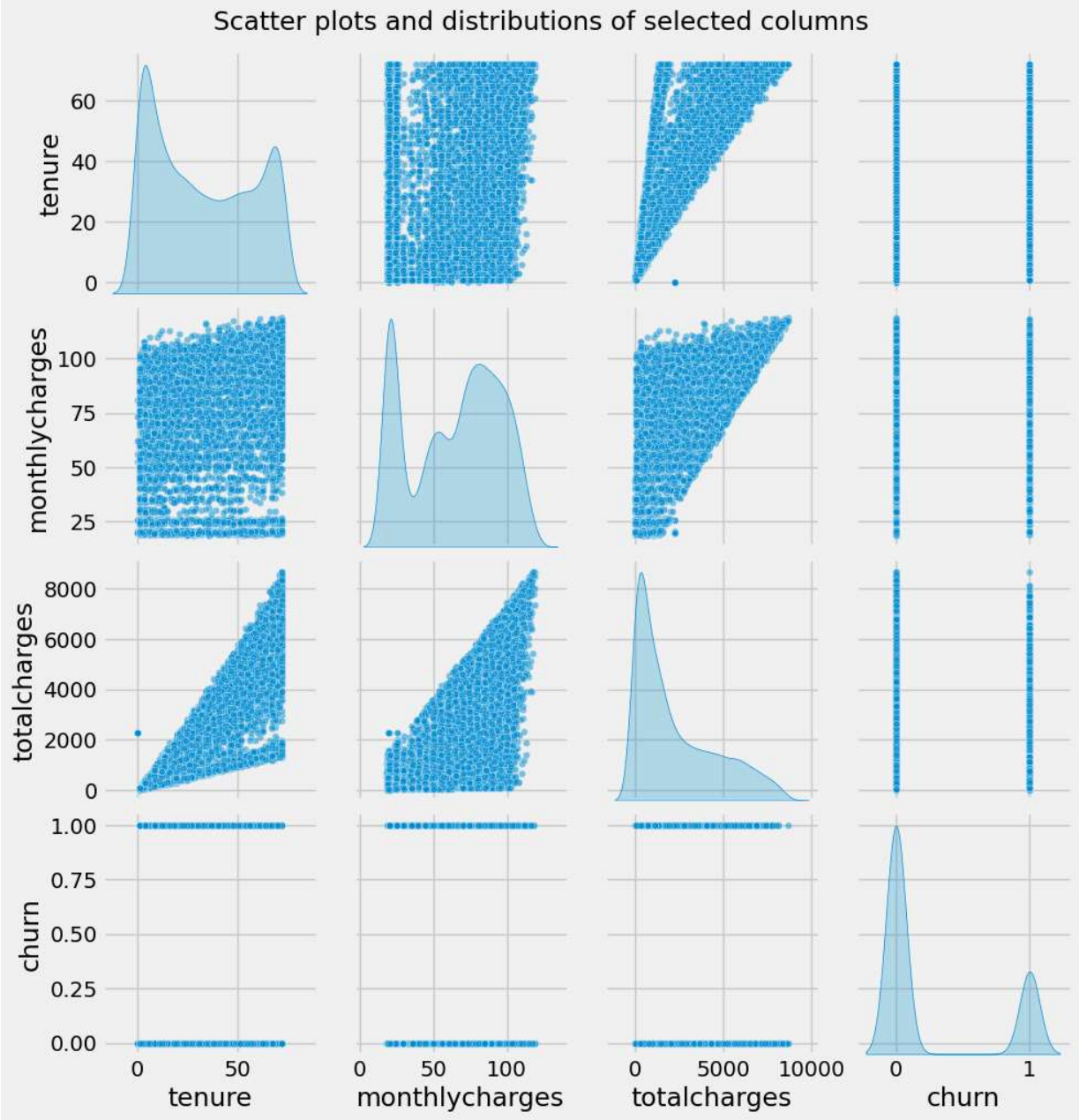
Out[57]:

	multiplelines	onlinesecurity	onlinebackup	deviceprotection	techsupport	streamingtv
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	0	1	1	0	0	0
3	0	1	0	1	1	0
4	0	0	0	0	0	0

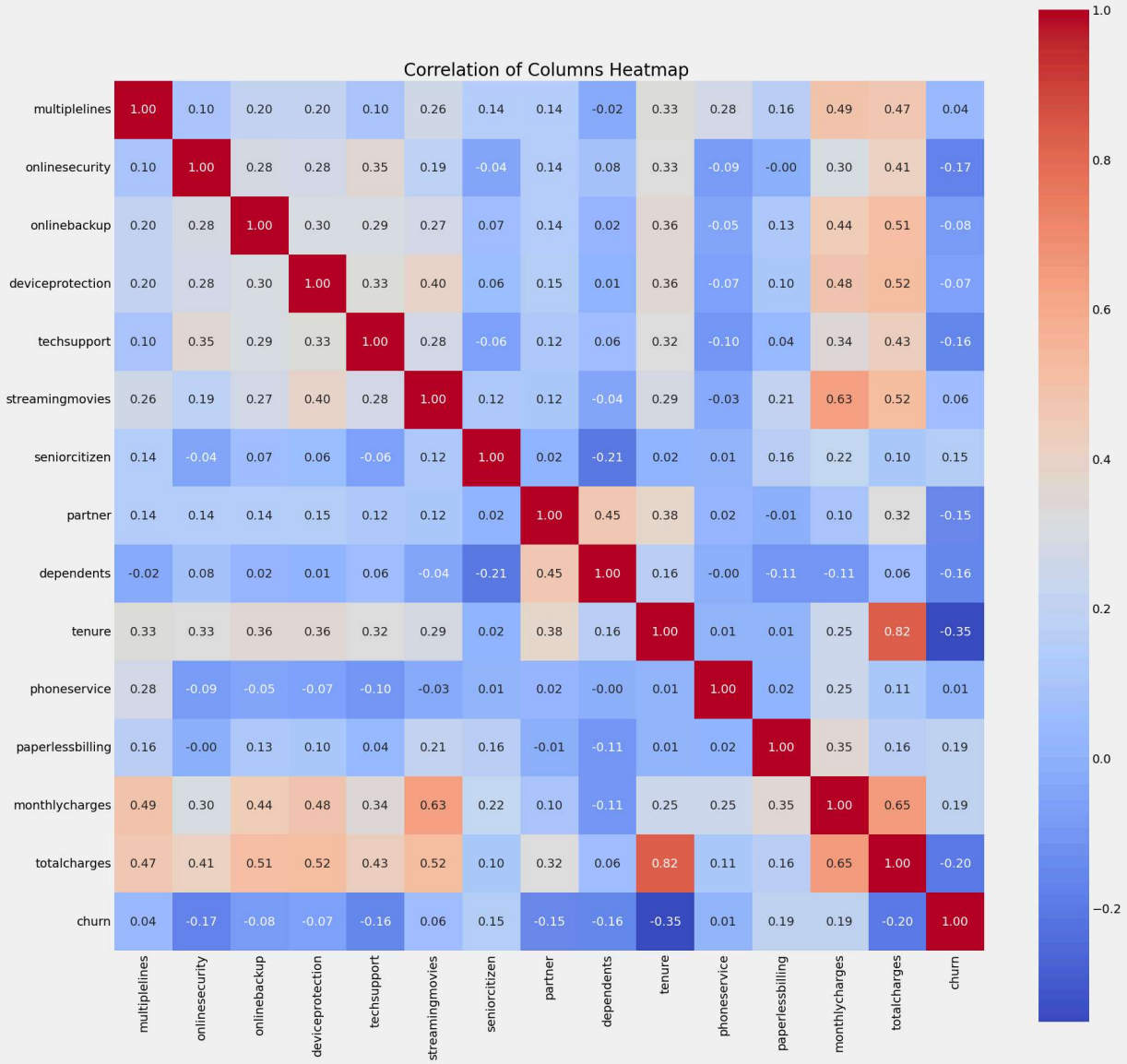
In [58]:
```
print(df.columns)
```

```
Index(['multiplelines', 'onlinesecurity', 'onlinebackup', 'deviceprotection',
       'techsupport', 'streamingtv', 'streamingmovies', 'gender',
       'seniorcitizen', 'partner', 'dependents', 'tenure', 'phoneservice',
       'contract', 'paperlessbilling', 'monthlycharges', 'totalcharges',
       'churn', 'DSL', 'Fiber optic', 'No',
       'paymentmethod_Bank transfer (automatic)',
       'paymentmethod_Credit card (automatic)',
       'paymentmethod_Electronic check', 'paymentmethod_Mailed check'],
      dtype='object')
```

In [59]:
```
cols = [ 'tenure', 'monthlycharges', 'totalcharges', 'churn', ]

sns.pairplot(df[cols], diag_kind='kde', plot_kws={'alpha':0.5, 's':20})

plt.suptitle('Scatter plots and distributions of selected columns', y=1.02)
plt.show()
```



In [60]:
```
corr = df[['multiplelines', 'onlinesecurity',
       'onlinebackup', 'deviceprotection', 'techsupport',
       'streamingmovies','seniorcitizen', 'partner', 'dependents',
       'tenure', 'phoneservice', 'paperlessbilling',
       'monthlycharges', 'totalcharges', 'churn',]].corr()

plt.figure(figsize=(20, 20))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar=True)

plt.title('Correlation of Columns Heatmap ')
plt.show()
```



In [61]:
```
print(df.columns)
```

```
Index(['multiplelines', 'onlinesecurity', 'onlinebackup', 'deviceprotection',
       'techsupport', 'streamingtv', 'streamingmovies', 'gender',
       'seniorcitizen', 'partner', 'dependents', 'tenure', 'phoneservice',
       'contract', 'paperlessbilling', 'monthlycharges', 'totalcharges',
       'churn', 'DSL', 'Fiber optic', 'No',
       'paymentmethod_Bank transfer (automatic)',
       'paymentmethod_Credit card (automatic)',
       'paymentmethod_Electronic check', 'paymentmethod_Mailed check'],
      dtype='object')
```

In [62]:
```
import statsmodels.api as sm
df['intercept']=1
lm=sm.OLS(df['churn'],df[['intercept', 'multiplelines', 'onlinesecurity', 'onlineba
       'techsupport', 'streamingtv', 'streamingmovies', 'gender',
       'seniorcitizen', 'partner', 'dependents', 'tenure', 'phoneservice',
       'contract', 'paperlessbilling', 'monthlycharges', 'totalcharges',
       'DSL', 'Fiber optic', 'No',
       'paymentmethod_Bank transfer (automatic)',
       'paymentmethod_Credit card (automatic)',
       'paymentmethod_Electronic check', 'paymentmethod_Mailed check']])
result=lm.fit()
result.summary()
```

Out[62]:

OLS Regression Results

Dep. Variable:	churn	**R-squared:**	0.280
Model:	OLS	**Adj. R-squared:**	0.278
Method:	Least Squares	**F-statistic:**	124.3
Date:	Sat, 21 Jun 2025	**Prob (F-statistic):**	0.00
Time:	14:01:40	**Log-Likelihood:**	-3077.0
No. Observations:	7042	**AIC:**	6200.
Df Residuals:	7019	**BIC:**	6358.
Df Model:	22		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
intercept	0.2729	0.070	3.879	0.000	0.135	0.411
multiplelines	0.0633	0.024	2.594	0.010	0.015	0.111
onlinesecurity	-0.0427	0.025	-1.717	0.086	-0.092	0.006
onlinebackup	-0.0114	0.024	-0.467	0.640	-0.059	0.037
deviceprotection	0.0035	0.025	0.143	0.886	-0.045	0.052
techsupport	-0.0428	0.025	-1.708	0.088	-0.092	0.006
streamingtv	0.0644	0.045	1.428	0.153	-0.024	0.153
streamingmovies	0.0667	0.045	1.480	0.139	-0.022	0.155
gender	-0.0036	0.009	-0.403	0.687	-0.021	0.014
seniorcitizen	0.0455	0.013	3.490	0.000	0.020	0.071
partner	-0.0008	0.011	-0.073	0.942	-0.022	0.020
dependents	-0.0214	0.011	-1.862	0.063	-0.044	0.001
tenure	-0.0019	0.000	-3.948	0.000	-0.003	-0.001
phoneservice	-0.0035	0.089	-0.039	0.969	-0.178	0.171
contract	-0.0371	0.008	-4.374	0.000	-0.054	-0.020
paperlessbilling	0.0461	0.010	4.605	0.000	0.026	0.066
monthlycharges	-0.0015	0.004	-0.337	0.736	-0.010	0.007
totalcharges	-4.691e-05	6.4e-06	-7.328	0.000	-5.95e-05	-3.44e-05
DSL	0.0803	0.025	3.243	0.001	0.032	0.129

Fiber optic	0.2993	0.132	2.261	0.024	0.040	0.559
No	-0.1067	0.087	-1.225	0.221	-0.277	0.064
paymentmethod_Bank transfer (automatic)	0.0541	0.020	2.748	0.006	0.016	0.093
paymentmethod_Credit card (automatic)	0.0479	0.020	2.432	0.015	0.009	0.087
paymentmethod_Electronic check	0.1234	0.019	6.398	0.000	0.086	0.161
paymentmethod_Mailed check	0.0475	0.019	2.441	0.015	0.009	0.086

Omnibus:	384.087	**Durbin-Watson:**	2.004
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	404.357
Skew:	0.554	**Prob(JB):**	1.57e-88
Kurtosis:	2.615	**Cond. No.**	8.69e+18

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.64e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Columns that have a high p-value do not have a direct effect on the churn column.

learn machine

In [63]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBRegressor


from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error
```

```
from sklearn.metrics import accuracy_score , recall_score , f1_score , precision_sc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.metrics import classification_report, accuracy_score, f1_score, recall
```

In [64]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [65]:
```
df['totalcharges'] = scaler.fit_transform(df[['totalcharges']])
df['monthlycharges'] = scaler.fit_transform(df[['monthlycharges']])
```

In [66]:
```
x= df.drop(['churn','onlinesecurity','deviceprotection','techsupport','partner','ph
y= df['churn']
```

In [67]:
```
smote = SMOTE(random_state=42)
```

In [68]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=21)
```

In [69]:
```
x_train, y_train = smote.fit_resample(x_train, y_train)
```

In [70]:
```
#call the models
#models = {
    #"LogisticRegression": (LogisticRegression(max_iter=1000), {
        #'C': [0.01, 0.1, 1, 10],
        #'solver': ['liblinear', 'lbfgs']
    #}),

   # "RandomForest": (RandomForestClassifier(), {
     #   'n_estimators': [50, 100],
     #   'max_depth': [None, 5, 10]
#    }),

 #   "GradientBoosting": (GradientBoostingClassifier(), {
  #      'n_estimators': [50, 100],
   #     'learning_rate': [0.05, 0.1],
    #    'max_depth': [3, 5]
#    }),

 #   "DecisionTree": (DecisionTreeClassifier(), {
  #      'max_depth': [None, 5, 10],
   #     'min_samples_split': [2, 5]
    #}),

 #   "KNeighbors": (KNeighborsClassifier(), {
  #      'n_neighbors': [3, 5, 7],
   #     'weights': ['uniform', 'distance']
    #}),

 #   "NaiveBayes": (GaussianNB(), {
```

```
  #
   # }),

 #   "SVC": (SVC(), {
  #       'C': [0.1, 1, 10],
   #      'kernel': ['linear', 'rbf']
   # }),

#}
```

```
  Cell In[70], line 6
    }),
    ^
IndentationError: unexpected indent
```

In [ ]:
```
# find the best parameters for models
#best_models = {}

#for name, (model, params) in models.items():
    #print(f"\n Running GridSearch for {name}")
    #grid = GridSearchCV(model, params, cv=5, scoring='accuracy', n_jobs=-1)
    #grid.fit(x_train, y_train)

    #print(f"Best Params: {grid.best_params_}")
    #print(f"Best CV Score: {grid.best_score_:.4f}")

    #best_model = grid.best_estimator_
    #best_models[name] = best_model

    #y_pred = best_model.predict(x_test)
    #acc = accuracy_score(y_test, y_pred)
    #print(f"Test Accuracy: {acc:.4f}")
```

In [71]:
```
log = LogisticRegression(C=0.1, solver='lbfgs', class_weight='balanced', max_iter=1

rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    class_weight='balanced',
    random_state=42)


svc = SVC(
    C=0.1,
    kernel='linear',
    class_weight='balanced',
    probability=True,
    random_state=42
)
```

In [72]:
```
def pred(model):
    model.fit(x_train, y_train)
    predictions = model.predict(x_test)

    print("=== Classification Report ===")
    print(classification_report(y_test, predictions))
```

```
    print("=== Accuracy Score ===")
    print(f"{accuracy_score(y_test, predictions):.4f}")

    print("=== F1 Score ===")
    print(f"{f1_score(y_test, predictions):.4f}")

    print("=== Recall Score ===")
    print(f"{recall_score(y_test, predictions):.4f}")

    print("=== Mean Absolute Error ===")
    print(f"{mean_absolute_error(y_test, predictions):.4f}")

    print("=== Mean Squared Error ===")
    print(f"{mean_squared_error(y_test, predictions):.4f}")

    print("=== Confusion Matrix ===")
    print(confusion_matrix(y_test, predictions))
```

In [73]:
```
pred(log)
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.88      0.76      0.82      1035
           1       0.53      0.72      0.61       374

    accuracy                           0.75      1409
   macro avg       0.70      0.74      0.71      1409
weighted avg       0.79      0.75      0.76      1409

=== Accuracy Score ===
0.7530
=== F1 Score ===
0.6090
=== Recall Score ===
0.7246
=== Mean Absolute Error ===
0.2470
=== Mean Squared Error ===
0.2470
=== Confusion Matrix ===
[[790 245]
 [103 271]]
```

In [74]:
```
pred(rf)
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.88      0.79      0.83      1035
           1       0.55      0.70      0.61       374

    accuracy                           0.77      1409
   macro avg       0.71      0.74      0.72      1409
weighted avg       0.79      0.77      0.77      1409

=== Accuracy Score ===
0.7658
=== F1 Score ===
0.6118
=== Recall Score ===
0.6952
=== Mean Absolute Error ===
0.2342
=== Mean Squared Error ===
0.2342
=== Confusion Matrix ===
[[819 216]
 [114 260]]
```

In [75]:
```
pred(svc)
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      1035
           1       0.51      0.74      0.61       374

    accuracy                           0.74      1409
   macro avg       0.70      0.74      0.71      1409
weighted avg       0.79      0.74      0.76      1409

=== Accuracy Score ===
0.7438
=== F1 Score ===
0.6063
=== Recall Score ===
0.7433
=== Mean Absolute Error ===
0.2562
=== Mean Squared Error ===
0.2562
=== Confusion Matrix ===
[[770 265]
 [ 96 278]]
```

In [76]:
```
probs = rf.predict_proba(x_test)[:, 1]


thresholds = np.arange(0.0, 1.01, 0.01)

f1_scores = []
```

```
precisions = []
recalls = []

for thresh in thresholds:
    preds = (probs >= thresh).astype(int)
    f1_scores.append(f1_score(y_test, preds))
    precisions.append(precision_score(y_test, preds))
    recalls.append(recall_score(y_test, preds))

best_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_idx]

print(f"Best threshold by F1 score: {best_threshold:.2f}")
print(f"F1 score at best threshold: {f1_scores[best_idx]:.4f}")
print(f"Precision at best threshold: {precisions[best_idx]:.4f}")
print(f"Recall at best threshold: {recalls[best_idx]:.4f}")
```

```
Best threshold by F1 score: 0.46
F1 score at best threshold: 0.6169
Precision at best threshold: 0.5342
Recall at best threshold: 0.7299
```

In [77]:

```
preds = (rf.predict_proba(x_test)[:, 1] >= 0.6).astype(int)

print("=== Classification Report ===")
print(classification_report(y_test, preds))

print("=== Accuracy Score ===")
print(f"{accuracy_score(y_test, preds):.4f}")

print("=== F1 Score ===")
print(f"{f1_score(y_test, preds):.4f}")

print("=== Recall Score ===")
print(f"{recall_score(y_test, preds):.4f}")

print("=== Mean Absolute Error ===")
print(f"{mean_absolute_error(y_test, preds):.4f}")

print("=== Mean Squared Error ===")
print(f"{mean_squared_error(y_test, preds):.4f}")

print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, preds))
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.84      0.83      0.84      1035
           1       0.55      0.57      0.56       374

    accuracy                           0.76      1409
   macro avg       0.70      0.70      0.70      1409
weighted avg       0.76      0.76      0.76      1409

=== Accuracy Score ===
0.7608
=== F1 Score ===
0.5595
=== Recall Score ===
0.5722
=== Mean Absolute Error ===
0.2392
=== Mean Squared Error ===
0.2392
=== Confusion Matrix ===
[[858 177]
 [160 214]]
```

Training the model in a different way(use k-fold split)

In [78]:
```
x= df.drop(['churn','onlinesecurity','deviceprotection','techsupport','partner','ph
y= df['churn']
```

In [79]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=21)
```

In [93]:
```
def call(model, x, y, n_splits=10, threshold=0.6, random_state=42):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_stat
    y_true_all = []
    y_prob_all = []

    for train_idx, test_idx in skf.split(x, y):
        x_train, x_test = x.iloc[train_idx], x.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        # ✅ SMOTE فقط على بيانات التدريب
        smote = SMOTE(random_state=random_state)
        x_train_res, y_train_res = smote.fit_resample(x_train, y_train)

        model.fit(x_train_res, y_train_res)
        y_prob = model.predict_proba(x_test)[:, 1]

        y_true_all.extend(y_test)
        y_prob_all.extend(y_prob)

    # تحويل الاحتمالات إلى تنبؤات حسب العتبة
    y_true_all = np.array(y_true_all)
    y_pred_all = (np.array(y_prob_all) >= threshold).astype(int)
```

```
    # ✅ التقارير الصحيحة على كامل البيانات وليس فقط آخر طيّة
    print(f"=== Classification Report (threshold={threshold}) ===")
    print(classification_report(y_true_all, y_pred_all))

    print("=== Accuracy Score ===")
    print(f"{accuracy_score(y_true_all, y_pred_all):.4f}")

    print("=== F1 Score ===")
    print(f"{f1_score(y_true_all, y_pred_all):.4f}")

    print("=== Recall Score ===")
    print(f"{recall_score(y_true_all, y_pred_all):.4f}")

    print("=== Mean Absolute Error ===")
    print(f"{mean_absolute_error(y_true_all, y_pred_all):.4f}")

    print("=== Mean Squared Error ===")
    print(f"{mean_squared_error(y_true_all, y_pred_all):.4f}")

    print("=== Confusion Matrix ===")
    print(confusion_matrix(y_true_all, y_pred_all))
```

In [86]:
```
call(rf, x, y, n_splits=20, threshold=0.6, random_state=42)
```

```
=== Classification Report (threshold=0.6) ===
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      5173
           1       0.59      0.66      0.62      1869

    accuracy                           0.79      7042
   macro avg       0.73      0.74      0.73      7042
weighted avg       0.79      0.79      0.79      7042

=== Accuracy Score ===
0.7856
=== F1 Score ===
0.6189
=== Recall Score ===
0.6560
=== Mean Absolute Error ===
0.2144
=== Mean Squared Error ===
0.2144
=== Confusion Matrix ===
[[4306  867]
 [ 643 1226]]
```

In [82]:
```
call(log, x, y, n_splits=10, threshold=0.6, random_state=42)
```

```
=== Classification Report (threshold=0.6) ===
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      5173
           1       0.57      0.65      0.61      1869

    accuracy                           0.78      7042
   macro avg       0.72      0.74      0.73      7042
weighted avg       0.79      0.78      0.78      7042

=== Accuracy Score ===
0.7788
=== F1 Score ===
0.6076
=== Recall Score ===
0.6453
=== Mean Absolute Error ===
0.2212
=== Mean Squared Error ===
0.2212
=== Confusion Matrix ===
[[4278  895]
 [ 663 1206]]
```

In [83]:
```
call(svc, x, y, n_splits=10, threshold=0.6, random_state=42)
```

```
=== Classification Report (threshold=0.6) ===
              precision    recall  f1-score   support

           0       0.87      0.81      0.84      5173
           1       0.56      0.68      0.61      1869

    accuracy                           0.77      7042
   macro avg       0.72      0.74      0.73      7042
weighted avg       0.79      0.77      0.78      7042

=== Accuracy Score ===
0.7749
=== F1 Score ===
0.6144
=== Recall Score ===
0.6758
=== Mean Absolute Error ===
0.2251
=== Mean Squared Error ===
0.2251
=== Confusion Matrix ===
[[4194  979]
 [ 606 1263]]
```

In [87]:
```
svc = SVC(kernel='rbf', probability=True, C=1.0, gamma='scale', random_state=42)
```

In [98]:
```
call(svc, x, y, n_splits=10, threshold=0.6, random_state=42)
```

```
=== Classification Report (threshold=0.6) ===
              precision    recall  f1-score   support

           0       0.89      0.79      0.83      5173
           1       0.55      0.72      0.62      1869

    accuracy                           0.77      7042
   macro avg       0.72      0.75      0.73      7042
weighted avg       0.80      0.77      0.78      7042

=== Accuracy Score ===
0.7688
=== F1 Score ===
0.6242
=== Recall Score ===
0.7234
=== Mean Absolute Error ===
0.2312
=== Mean Squared Error ===
0.2312
=== Confusion Matrix ===
[[4062 1111]
 [ 517 1352]]
```

save the best model to use it later

In [95]:
```
import joblib
```

In [97]:
```
joblib.dump(svc, 'svc_model.pkl')
```

Out[97]:
```
['svc_model.pkl']
```

summary

The model is very good at identifying customers at risk of churning (Recall = 72%), which is excellent because the goal is early intervention.

However, it does produce some false alarms (Precision = 55%), meaning some customers predicted to churn will actually stay.

This is acceptable in churn scenarios, as companies often prefer to send extra offers even if the customer doesn't churn, rather than lose them unexpectedly.

In [ ]:
```

```