problem statement:-

We are using a customer churn dataset from Kaggle with 7,043 rows and 21 columns. Our goal is to explore the data using Exploratory Data Analysis (EDA) to understand why customers leave, identify key patterns, and determine whether a customer has churned or not.

What do you mean by Churn?

Churn refers to when a customer stops using a product or service over a given period. It's also known as customer attrition.For example:-

1. A Netflix user cancels their subscription → Churned customer.
2. A bank customer closes their account → Churned customer.
3. A telecom user switches to another provider → Churned customer.

In Exploratory Data Analysis (EDA), we typically perform the following steps:-

1. Importing Libraries – Load essential Python libraries such as pandas, numpy, seaborn, and matplotlib.
2. Loading the Dataset – Read the dataset into a dataframe for analysis.
3. Handling Missing Values – Identify and manage null or missing values.
4. Exploring Numerical Variables – Analyze statistical properties, distributions, and outliers.
5. Exploring Categorical Variables – Examine unique values, frequencies, and distributions.
6. Feature Relationships & Insights – Use correlation, pair plots, and visualizations to understand feature dependencies.

Importing Some Important Library

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading and reading the dataset

In [4]:
```
df=pd.read_csv('Customer Churn.csv')
df
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Devic
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	
...	...	...	...	...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	

7043 rows × 21 columns

Top five rows from the start.

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

Out[5]:

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

5 rows × 21 columns

Top five rows from the bottom/end

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

Out[6]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Devic
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	

5 rows × 21 columns

Check how many rows and columns are present in this dataset.

In [7]:
```
print("The Number of rows",df.shape[0])
```

```
print("The Number of columns",df.shape[1])
```

```
The Number of rows 7043
The Number of columns 21
```

Observation:-

1. There are 7043 rows and 21 columns

Check Information of the Dataset

In [8]:
```
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
```

Check the numbers of null value present in this dataset

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

Out[9]:
```
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
```

Observation:-

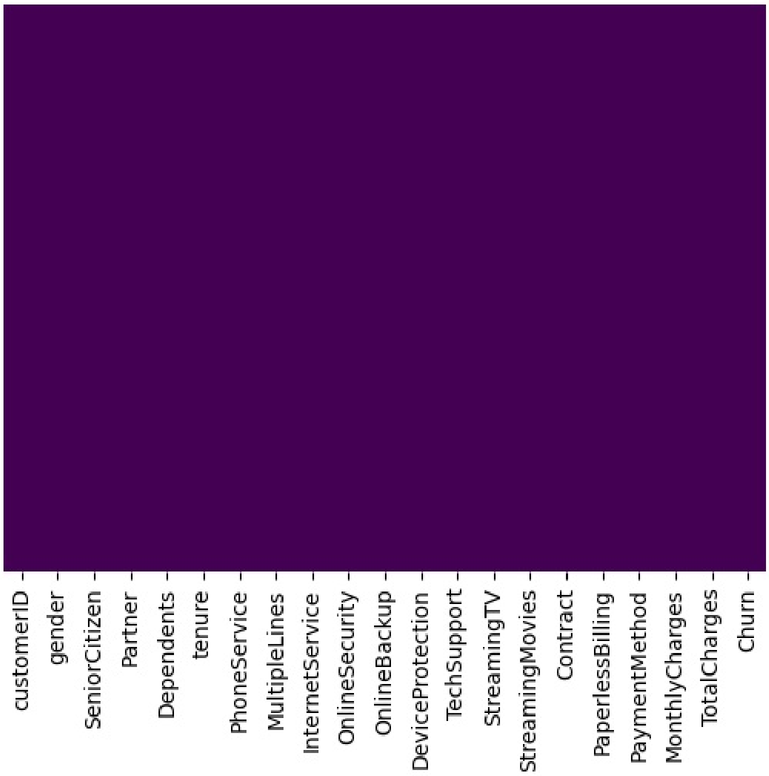
1. There are no any null-value present of this datasets

visualize these null value by using the heatmap

In [10]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[10]:

```
<Axes: >
```



Perform Statistical analysis

In [11]:

```
jac=df.describe()
jac
```

Out[11]:

	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

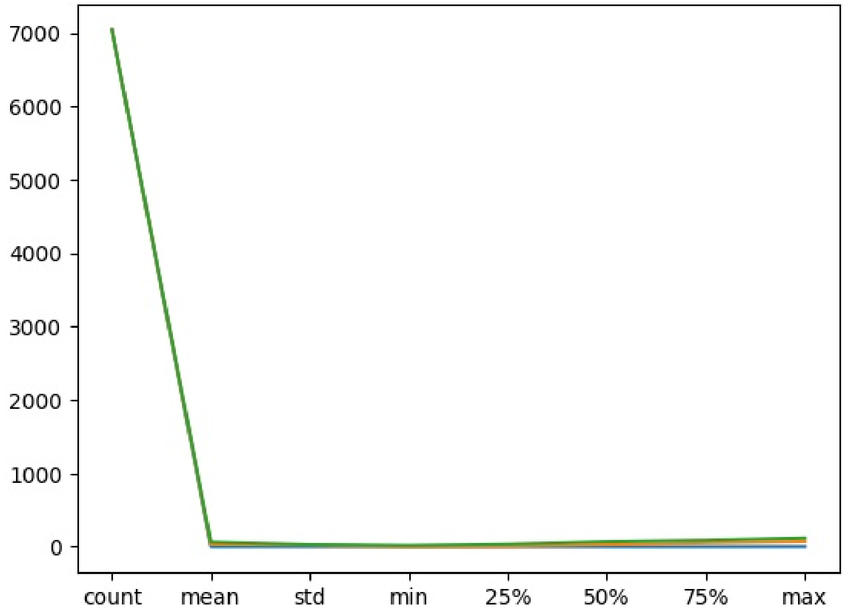
Observations:

1. SeniorCitizen:- Most customers are not senior citizens, as only about 16.2% fall into this category.
2. Tenure:- On average, customers stay for about 32 months, but the tenure varies widely, ranging from 0 to 72 months.
3. MonthlyCharges:- The monthly charges vary significantly, with some customers paying as low as 18.25 and others payingup to 118.75.

Visualize Statistical analysis

In [12]:

```
plt.plot(jac)
plt.show()
```



Check the number of unique values in each column

In [13]:
```
vil=df.nunique()
vil
```

Out[13]:
```
customerID          7043
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

Categorical feature

In [15]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtype=='O']
categorical_features
```

```
Out[15]: ['customerID',
 'gender',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'TotalCharges',
 'Churn']
```

Number of Categorical feature

```
In [16]: print(len([feature for feature in df.columns if df[feature].dtype=='O']))
```

```
18
```

Observation:-

1. There are 18 categorical feature present of this datasets

Numerical columns

```
In [17]: numerical_features = [feature for feature in df.columns if df[feature].dtype!='O']
numerical_features
```

```
Out[17]: ['SeniorCitizen', 'tenure', 'MonthlyCharges']
```

Number of Numerical feature

```
In [18]: print(len([feature for feature in df.columns if df[feature].dtype!='O']))
```

```
3
```

Observation:-

1. There are 3 numerical feature present of this datasets

Count the number of males and females present in the gender column of this dataset.

```
In [19]: df['gender'].value_counts()
```

```
Out[19]: gender
Male      3555
Female    3488
Name: count, dtype: int64
```

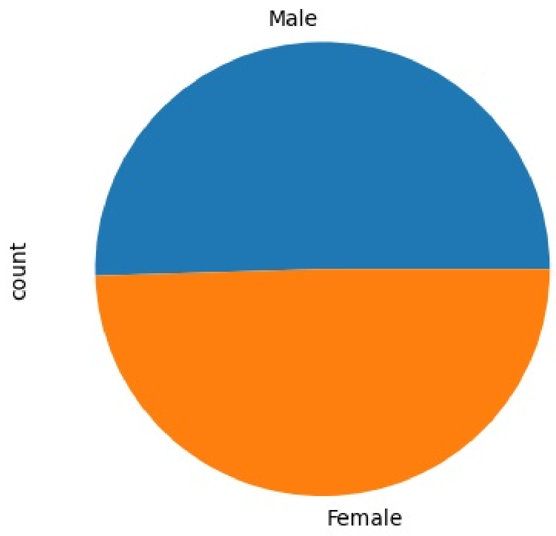
Observation

1. There are 3,555 males and 3,588 females.

Visualization of the Gender Column Using a Pie Chart

```
In [20]: df['gender'].value_counts().plot(kind='pie')
```

```
Out[20]: <Axes: ylabel='count'>
```



Check the Number of Duplicate Records in the CustomerID Column of This Dataset.

In [21]:
```
print(df['customerID'].duplicated().sum())
```

```
0
```

Observation:-

1. There are zero duplicate records present in the CustomerID Column of This Dataset.

We are trying to split the numerical and categorical parts of the CustomerID column in this dataset.

In [22]:
```
df['customerID']
```

Out[22]:
```
0       7590-VHVEG
1       5575-GNVDE
2       3668-QPYBK
3       7795-CFOCW
4       9237-HQITU
           ...
7038    6840-RESVB
7039    2234-XADUH
7040    4801-JZAZL
7041    8361-LTMKD
7042    3186-AJIEK
Name: customerID, Length: 7043, dtype: object
```

In [23]:
```
numerical_part = []
categorical_part = []

for i in df['customerID']:
    num, cat = i.split('-')  # Split into two parts based on '-'
    numerical_part.append(num)  # Append numerical part
    categorical_part.append(cat)  # Append categorical part

# Convert lists into new DataFrame columns
df['customerID_numerical'] = numerical_part
df['customerID_categorical'] = categorical_part

# Display the first few rows to verify
df[['customerID', 'customerID_numerical', 'customerID_categorical']].head()
```

Out[23]:

	customerID	customerID_numerical	customerID_categorical
0	7590-VHVEG	7590	VHVEG
1	5575-GNVDE	5575	GNVDE
2	3668-QPYBK	3668	QPYBK
3	7795-CFOCW	7795	CFOCW
4	9237-HQITU	9237	HQITU

Replacing blanks with 0 as tenure is 0 and no total charges are recorded

In [24]:
```
df["TotalCharges"]
```

Out[24]:
```
0         29.85
1        1889.5
2        108.15
3       1840.75
4        151.65
         ...
7038     1990.5
7039     7362.9
7040     346.45
7041      306.6
7042     6844.5
Name: TotalCharges, Length: 7043, dtype: object
```

In [25]:
```
df["TotalCharges"].dtypes
```

Out[25]:
```
dtype('O')
```

In [26]:
```
df["TotalCharges"] = df["TotalCharges"].replace(" ","0")
df["TotalCharges"] = df["TotalCharges"].astype("float")
```

In [27]:
```
df["TotalCharges"].dtypes
```

Out[27]:
```
dtype('float64')
```

Transforming SeniorCitizen Column from Numeric (0/1) to Categorical (No/Yes).

In [28]:
```
df['SeniorCitizen']
```

Out[28]:
```
0       0
1       0
2       0
3       0
4       0
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: SeniorCitizen, Length: 7043, dtype: int64
```

In [29]:
```
def conv(value):
    if value == 1:
        return "yes"
    else:
        return "no"

df['SeniorCitizen'] = df["SeniorCitizen"].apply(conv)
```

In [30]:
```
df['SeniorCitizen']
```

Out[30]:
```
0        no
1        no
2        no
3        no
4        no
       ...
7038     no
7039     no
7040     no
7041    yes
7042     no
Name: SeniorCitizen, Length: 7043, dtype: object
```

Observation:-

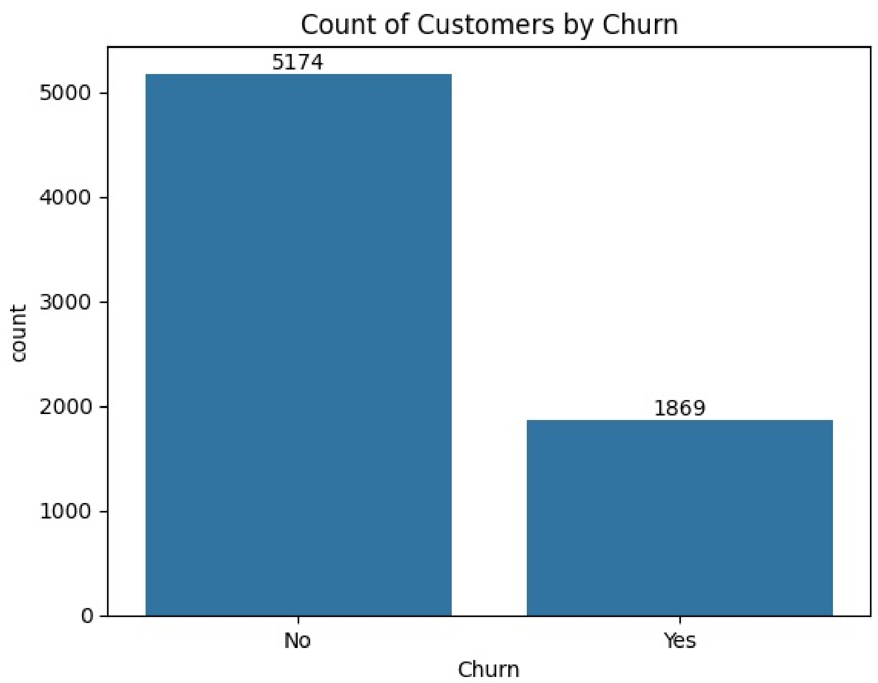
1. The SeniorCitizen column originally had numeric values (0 and 1), where 0 meant "No" and 1 meant "Yes".

2. After applying the conv function, the column now has categorical values ("No" and "Yes"), making it easier to understand.

3. Most of the values in the column are "No", which means the majority of customers are not senior citizens.

Converted 0 and 1 values of senior citizen to yes/no to make it easier to understand

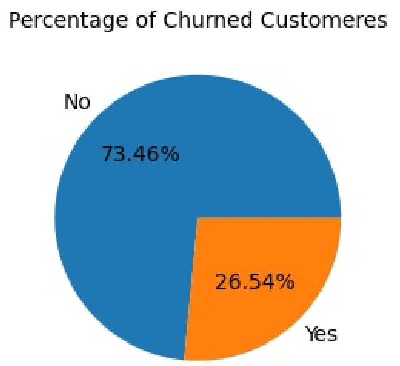
In [31]:
```
ax = sns.countplot(x = 'Churn', data = df)

ax.bar_label(ax.containers[0])
plt.title("Count of Customers by Churn")
plt.show()
```



Observations:-

1. Most customers stay loyal:
2. 73.5% (5,174) haven't churned ("No").
3. Only 26.5% (1,869) left the service ("Yes").

In [32]:
```
plt.figure(figsize = (3,4))
gb = df.groupby("Churn").agg({'Churn':"count"})
plt.pie(gb['Churn'], labels = gb.index, autopct = "%1.2f%%")
plt.title("Percentage of Churned Customeres", fontsize = 10)
plt.show()
```

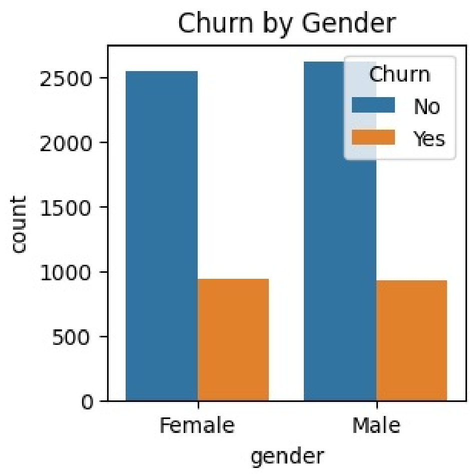


From the given pie chart we can conclude that 26.54% of our

customers have churned out.

Not let's explore the reason behind it

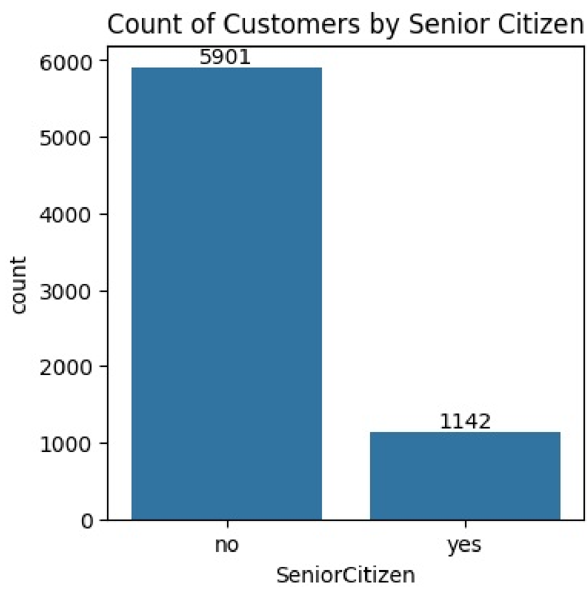
In [33]:
```
plt.figure(figsize = (3,3))
sns.countplot(x = "gender", data = df, hue = "Churn")
plt.title("Churn by Gender")
plt.show()
```



Observation:-

1. Churn rate is similar for both genders.
2. The number of customers who stay (No Churn) and leave (Churn) is almost equal for males and females.
3. Gender does not seem to be a significant factor in customer churn.

In [34]:
```
plt.figure(figsize = (4,4))
ax = sns.countplot(x = "SeniorCitizen", data = df)
ax.bar_label(ax.containers[0])
plt.title("Count of Customers by Senior Citizen")
plt.show()
```

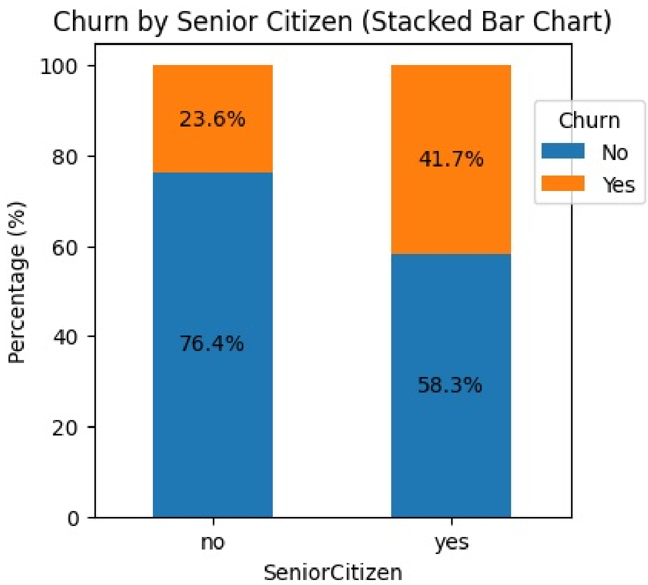


Observations:-

1. Most Customers Are Not Senior Citizens – 5,901 customers (majority) are non-senior citizens.
2. Fewer Senior Citizen Customers – Only 1,142 customers are senior citizens.
3. Customer Base is Mostly Younger – The majority of users are young or middle-aged.

In [35]:
```
total_counts = df.groupby('SeniorCitizen')['Churn'].value_counts(normalize=True).unstack() * 100

# Plot
fig, ax = plt.subplots(figsize=(4, 4))  # Adjust figsize for better visualization
```

```
# Plot the bars
total_counts.plot(kind='bar', stacked=True, ax=ax, color=['#1f77b4', '#ff7f0e'])  # Customize colors if desired

# Add percentage labels on the bars
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.text(x + width / 2, y + height / 2, f'{height:.1f}%', ha='center', va='center')

plt.title('Churn by Senior Citizen (Stacked Bar Chart)')
plt.xlabel('SeniorCitizen')
plt.ylabel('Percentage (%)')
plt.xticks(rotation=0)
plt.legend(title='Churn', bbox_to_anchor = (0.9,0.9))  # Customize legend location

plt.show()
```

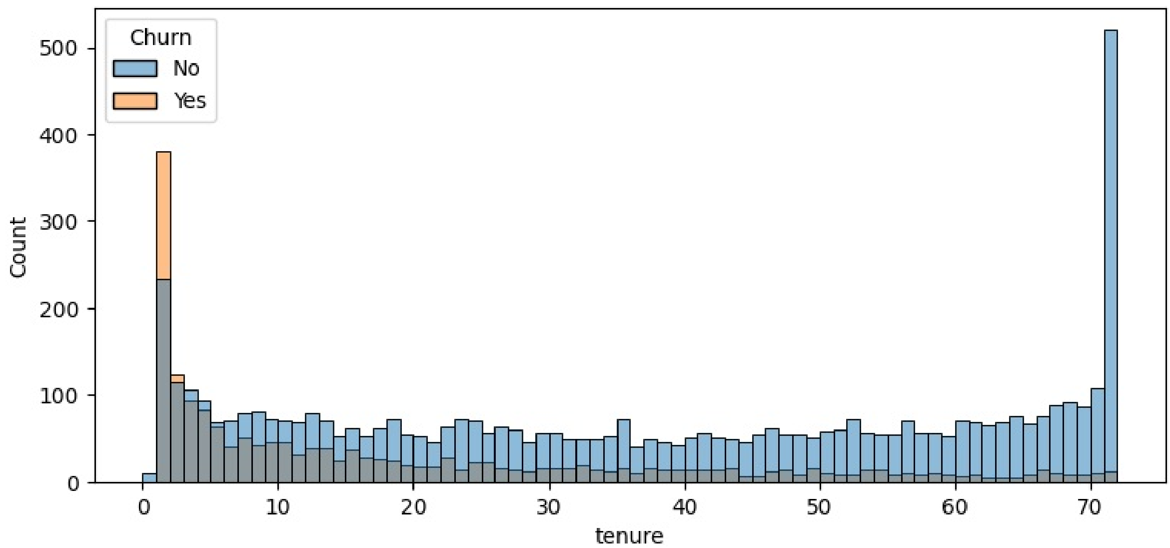


Observations:-

1. Higher Churn Among Senior Citizens:- 41.7% of senior citizens have churned, compared to 23.6% of non-senior customers.
2. Lower Retention for Seniors:- Only 58.3% of senior citizens remain, while 76.4% of non-seniors continue using the service.
3. Key Insight:- Senior citizens are twice as likely to churn, suggesting a need for better support, customized plans, or incentives.

comparative a greater pecentage of people in senior citizen category have churned.

In [36]:
```
plt.figure(figsize = (9,4))
sns.histplot(x = "tenure", data = df, bins = 72, hue = "Churn")
plt.show()
```

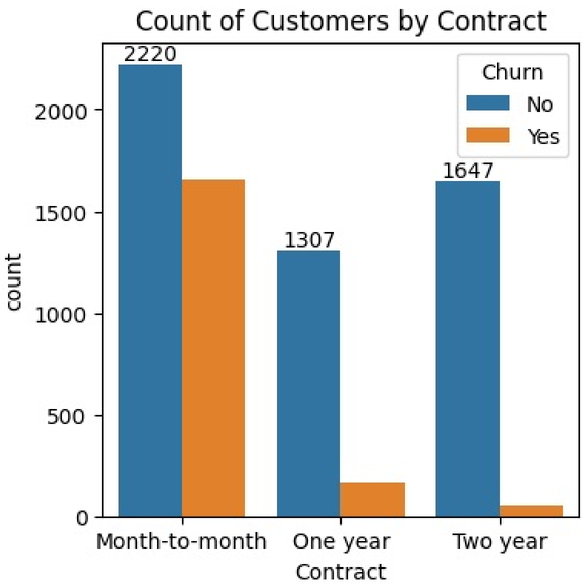


people who have used our services for a long time have stayed

and people who have used our sevices.

1 or 2 months have churned

In [37]:
```
plt.figure(figsize = (4,4))
ax = sns.countplot(x = "Contract", data = df, hue = "Churn")
ax.bar_label(ax.containers[0])
plt.title("Count of Customers by Contract")
plt.show()
```



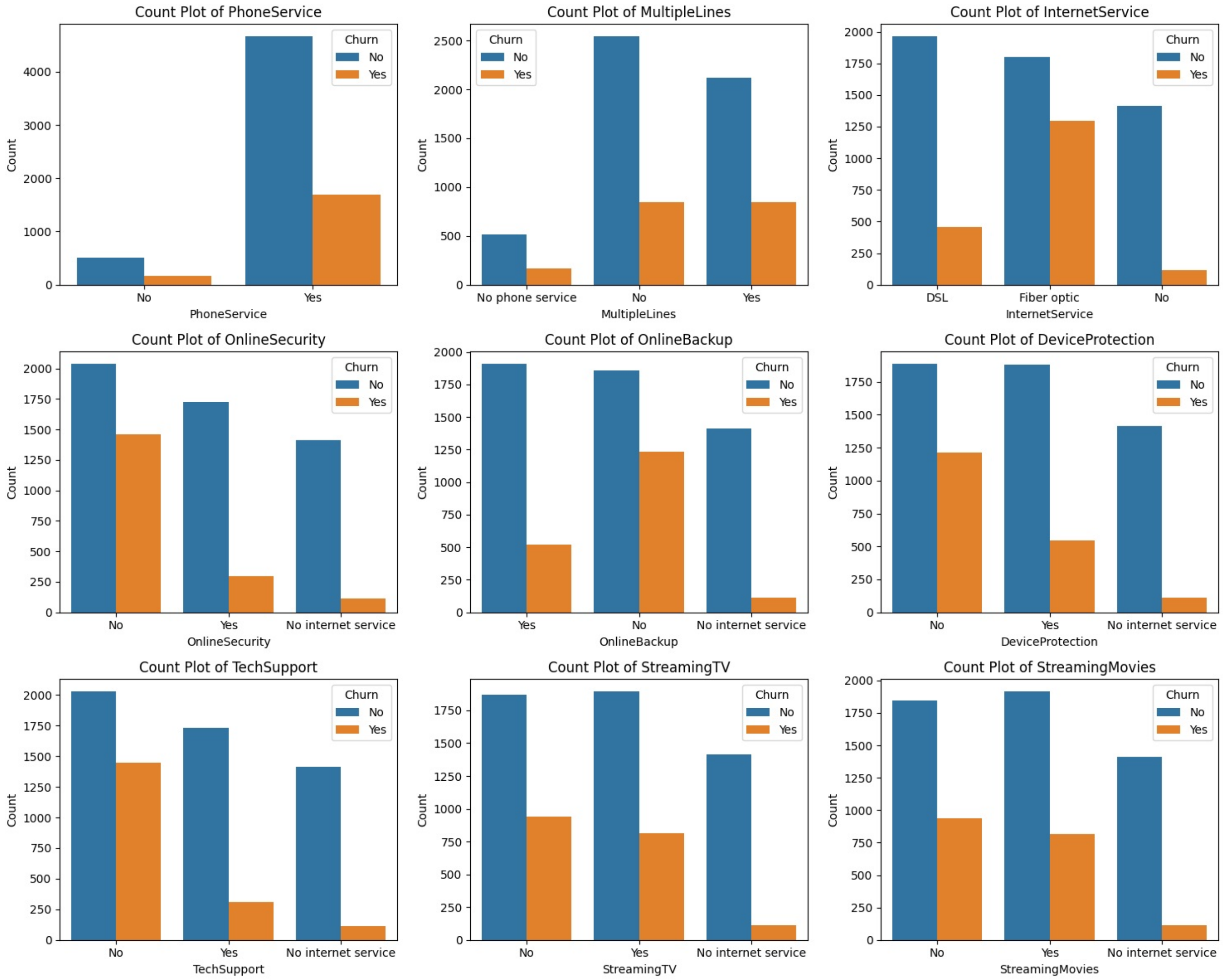
Observations:-

1. Month-to-Month Contracts Have High Churn:- These customers are more likely to leave than those with 1-year or 2-year contracts.
2. Longer Contracts Have Lower Churn:- 1-year and 2-year contract customers show higher retention.
3. Key Takeaway:- Encouraging long-term contracts with discounts or perks can reduce churn.

In [38]:
```
df.columns.values
```

Out[38]:
```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn', 'customerID_numerical',
       'customerID_categorical'], dtype=object)
```

In [39]:
```
columns = ['PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
           'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

# Number of columns for the subplot grid (you can change this)
n_cols = 3
n_rows = (len(columns) + n_cols - 1) // n_cols  # Calculate number of rows needed

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 4))  # Adjust figsize as needed

# Flatten the axes array for easy iteration (handles both 1D and 2D arrays)
axes = axes.flatten()

# Iterate over columns and plot count plots
for i, col in enumerate(columns):
    sns.countplot(x=col, data=df, ax=axes[i], hue = df["Churn"])
    axes[i].set_title(f'Count Plot of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

# Remove empty subplots (if any)
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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

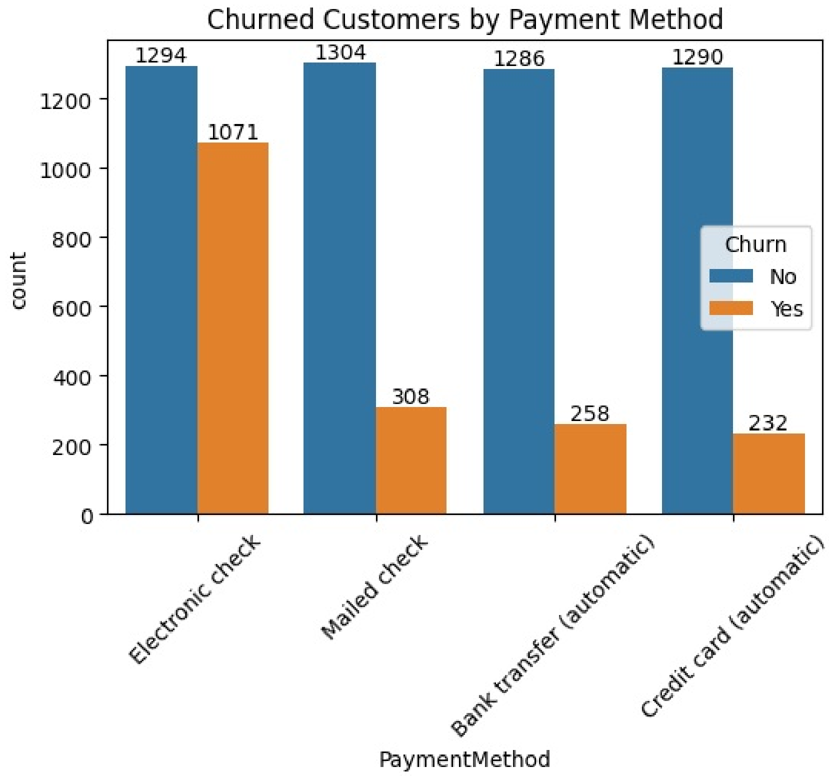


Observations:-

1. Lower churn is seen among customers with PhoneService, DSL Internet, and OnlineSecurity.
2. Higher churn occurs when OnlineBackup, TechSupport, and StreamingTV are not used or unavailable.
3. This suggests that customers with additional services tend to stay longer, possibly due to increased engagement or value from these services.

In [40]:

```
plt.figure(figsize = (6,4))
ax = sns.countplot(x = "PaymentMethod", data = df, hue = "Churn")
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title("Churned Customers by Payment Method")
plt.xticks(rotation = 45)
plt.show()
```



Observation:-

Customers using electronic checks have the highest churn rate compared to other payment methods. Automatic payments (bank transfer, credit card) have the lowest churn, suggesting they are more stable payment options.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js