In [1]:
```
#import the warnings.
#import warnings
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df1 = pd.read_csv("TelcomCustomer-Churn_1.csv",skipinitialspace=True)
df1.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
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

In [3]:
```
help(pd.read_csv)
```

```
        the separator, but the Python parsing engine can, meaning the latter will
        be used and automatically detect the separator by Python's builtin sniffer
        tool, ``csv.Sniffer``. In addition, separators longer than 1 character and
        different from ``'\s+'`` will be interpreted as regular expressions and
        will also force the use of the Python parsing engine. Note that regex
        delimiters are prone to ignoring quoted data. Regex example: ``'\r\t'``.
    delimiter : str, default ``None``
        Alias for sep.
    header : int, list of int, None, default 'infer'
        Row number(s) to use as the column names, and the start of the
        data.  Default behavior is to infer the column names: if no names
        are passed the behavior is identical to ``header=0`` and column
        names are inferred from the first line of the file, if column
        names are passed explicitly then the behavior is identical to
        ``header=None``. Explicitly pass ``header=0`` to be able to
        replace existing names. The header can be a list of integers that
        specify row locations for a multi-index on the columns
        e.g. [0,1,3]. Intervening rows that are not specified will be
        skipped (e.g. 2 in this example is skipped). Note that this
        parameter ignores commented lines and empty lines if
```

In [4]:
```
df1.shape
```

Out[4]: (7043, 10)

In [5]:
```
df2 = pd.read_csv("TelcomCustomer-Churn_2.csv",skipinitialspace=True)
df2.head()
```

Out[5]:

	customerID	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	Tota
0	7590-VHVEG	Yes	No	No	No	No	Month-to-month	Yes	Electronic check	29.85	
1	5575-GNVDE	No	Yes	No	No	No	One year	No	Mailed check	56.95	
2	3668-QPYBK	Yes	No	No	No	No	Month-to-month	Yes	Mailed check	53.85	
3	7795-CFOCW	No	Yes	Yes	No	No	One year	No	Bank transfer (automatic)	42.30	
4	9237-HQITU	No	No	No	No	No	Month-to-month	Yes	Electronic check	70.70	

In [6]:
```
df2.shape
```

Out[6]: (7043, 12)

In [7]:
```
df_concat=pd.concat([df1,df2],axis=1)
df_concat.head()
```

Out[7]:

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

In [8]:
```
df_concat.shape
```

Out[8]:
```
(7043, 22)
```

In [9]:
```
df_concat.columns
```

Out[9]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'customerID', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges',
       'Churn'],
      dtype='object')
```

In [10]:
```
df=pd.merge(df1, df2, on="customerID")
df.head()
```

Out[10]:

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

In [11]:
```
help(pd.merge)
```

```
Help on function merge in module pandas.core.reshape.merge:

merge(left: 'DataFrame | Series', right: 'DataFrame | Series', how: 'str' = 'inner', on: 'IndexLabel | None' = None, left_on:
'IndexLabel | None' = None, right_on: 'IndexLabel | None' = None, left_index: 'bool' = False, right_index: 'bool' = False, sor
t: 'bool' = False, suffixes: 'Suffixes' = ('_x', '_y'), copy: 'bool' = True, indicator: 'bool' = False, validate: 'str | None'
= None) -> 'DataFrame'
    Merge DataFrame or named Series objects with a database-style join.

    A named Series object is treated as a DataFrame with a single named column.

    The join is done on columns or indexes. If joining columns on
    columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes
    on indexes or indexes on a column or columns, the index will be passed on.
    When performing a cross merge, no column specifications to merge on are
    allowed.

    .. warning::

        If both key columns contain rows where the key is a null value, those
        rows will be matched against each other. This is different from usual SQL
        join behaviour and can lead to unexpected results.

    Parameters
    ----------
    left : DataFrame
    right : DataFrame or named Series
        Object to merge with.
    how : {'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'
        Type of merge to be performed.

        * left: use only keys from left frame, similar to a SQL left outer join;
          preserve key order.
        * right: use only keys from right frame, similar to a SQL right outer join;
          preserve key order.
        * outer: use union of keys from both frames, similar to a SQL full outer
          join; sort keys lexicographically.
        * inner: use intersection of keys from both frames, similar to a SQL inner
          join; preserve the order of the left keys.
        * cross: creates the cartesian product from both frames, preserves the order
          of the left keys.

          .. versionadded:: 1.2.0

    on : label or list
        Column or index level names to join on. These must be found in both
        DataFrames. If `on` is None and not merging on indexes then this defaults
        to the intersection of the columns in both DataFrames.
    left_on : label or list, or array-like
        Column or index level names to join on in the left DataFrame. Can also
        be an array or list of arrays of the length of the left DataFrame.
        These arrays are treated as if they are columns.
    right_on : label or list, or array-like
        Column or index level names to join on in the right DataFrame. Can also
        be an array or list of arrays of the length of the right DataFrame.
        These arrays are treated as if they are columns.
    left_index : bool, default False
        Use the index from the left DataFrame as the join key(s). If it is a
        MultiIndex, the number of keys in the other DataFrame (either the index
        or a number of columns) must match the number of levels.
    right_index : bool, default False
        Use the index from the right DataFrame as the join key. Same caveats as
        left_index.
    sort : bool, default False
        Sort the join keys lexicographically in the result DataFrame. If False,
        the order of the join keys depends on the join type (how keyword).
    suffixes : list-like, default is ("_x", "_y")
        A length-2 sequence where each element is optionally a string
        indicating the suffix to add to overlapping column names in
        `left` and `right` respectively. Pass a value of `None` instead
        of a string to indicate that the column name from `left` or
        `right` should be left as-is, with no suffix. At least one of the
        values must not be None.
    copy : bool, default True
        If False, avoid copy if possible.
    indicator : bool or str, default False
        If True, adds a column to the output DataFrame called "_merge" with
        information on the source of each row. The column can be given a different
        name by providing a string argument. The column will have a Categorical
        type with the value of "left_only" for observations whose merge key only
        appears in the left DataFrame, "right_only" for observations
        whose merge key only appears in the right DataFrame, and "both"
        if the observation's merge key is found in both DataFrames.

    validate : str, optional
        If specified, checks if merge is of specified type.
```

```
    * "one_to_one" or "1:1": check if merge keys are unique in both
      left and right datasets.
    * "one_to_many" or "1:m": check if merge keys are unique in left
      dataset.
    * "many_to_one" or "m:1": check if merge keys are unique in right
      dataset.
    * "many_to_many" or "m:m": allowed, but does not result in checks.

Returns
-------
DataFrame
    A DataFrame of the two merged objects.

See Also
--------
merge_ordered : Merge with optional filling/interpolation.
merge_asof : Merge on nearest keys.
DataFrame.join : Similar method using indices.

Notes
-----
Support for specifying index levels as the `on`, `left_on`, and
`right_on` parameters was added in version 0.23.0
Support for merging named Series objects was added in version 0.24.0

Examples
--------
>>> df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
...                     'value': [1, 2, 3, 5]})
>>> df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
...                     'value': [5, 6, 7, 8]})
>>> df1
    lkey value
0   foo      1
1   bar      2
2   baz      3
3   foo      5
>>> df2
    rkey value
0   foo      5
1   bar      6
2   baz      7
3   foo      8

Merge df1 and df2 on the lkey and rkey columns. The value columns have
the default suffixes, _x and _y, appended.

>>> df1.merge(df2, left_on='lkey', right_on='rkey')
  lkey  value_x rkey  value_y
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7

Merge DataFrames df1 and df2 with specified left and right suffixes
appended to any overlapping columns.

>>> df1.merge(df2, left_on='lkey', right_on='rkey',
...           suffixes=('_left', '_right'))
  lkey  value_left rkey  value_right
0  foo           1  foo            5
1  foo           1  foo            8
2  foo           5  foo            5
3  foo           5  foo            8
4  bar           2  bar            6
5  baz           3  baz            7

Merge DataFrames df1 and df2, but raise an exception if the DataFrames have
any overlapping columns.

>>> df1.merge(df2, left_on='lkey', right_on='rkey', suffixes=(False, False))
Traceback (most recent call last):
...
ValueError: columns overlap but no suffix specified:
    Index(['value'], dtype='object')

>>> df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
>>> df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})
>>> df1
      a  b
0   foo  1
1   bar  2
>>> df2
      a  c
```

```
0   foo  3
1   baz  4

>>> df1.merge(df2, how='inner', on='a')
     a  b  c
0   foo  1  3

>>> df1.merge(df2, how='left', on='a')
     a  b  c
0   foo  1  3.0
1   bar  2  NaN

>>> df1 = pd.DataFrame({'left': ['foo', 'bar']})
>>> df2 = pd.DataFrame({'right': [7, 8]})
>>> df1
    left
0   foo
1   bar
>>> df2
    right
0   7
1   8

>>> df1.merge(df2, how='cross')
   left  right
0   foo      7
1   foo      8
2   bar      7
3   bar      8
```

In [12]:
```
df.shape
```

Out[12]:
```
(7043, 21)
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.2+ MB
```

Data Cleansing

In [14]:
```
df.duplicated().sum()
```

Out[14]:
```
0
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
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       11
Churn               0
dtype: int64
```

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

Out[16]:
```
11
```

In [17]:
```
null_percentage= (df.isnull().sum()/7043)*100
null_percentage
```

Out[17]:
```
customerID          0.000000
gender              0.000000
SeniorCitizen       0.000000
Partner             0.000000
Dependents          0.000000
tenure              0.000000
PhoneService        0.000000
MultipleLines       0.000000
InternetService     0.000000
OnlineSecurity      0.000000
OnlineBackup        0.000000
DeviceProtection    0.000000
TechSupport         0.000000
StreamingTV         0.000000
StreamingMovies     0.000000
Contract            0.000000
PaperlessBilling    0.000000
PaymentMethod       0.000000
MonthlyCharges      0.000000
TotalCharges        0.156183
Churn               0.000000
dtype: float64
```

In [18]:
```
null_percentage= (df.isnull().sum()/len(df.index))*100
null_percentage
```

Out[18]:
```
customerID          0.000000
gender              0.000000
SeniorCitizen       0.000000
Partner             0.000000
Dependents          0.000000
tenure              0.000000
PhoneService        0.000000
MultipleLines       0.000000
InternetService     0.000000
OnlineSecurity      0.000000
OnlineBackup        0.000000
DeviceProtection    0.000000
TechSupport         0.000000
StreamingTV         0.000000
StreamingMovies     0.000000
Contract            0.000000
PaperlessBilling    0.000000
PaymentMethod       0.000000
MonthlyCharges      0.000000
TotalCharges        0.156183
Churn               0.000000
dtype: float64
```

In [19]:
```
null_percentage= df.isnull().sum()/len(df.index)
null_percentage = round(100*(null_percentage),2)
null_percentage
```

Out[19]:
```
customerID          0.00
gender              0.00
SeniorCitizen       0.00
Partner             0.00
Dependents          0.00
tenure              0.00
PhoneService        0.00
MultipleLines       0.00
InternetService     0.00
OnlineSecurity      0.00
OnlineBackup        0.00
DeviceProtection    0.00
TechSupport         0.00
StreamingTV         0.00
StreamingMovies     0.00
Contract            0.00
PaperlessBilling    0.00
PaymentMethod       0.00
MonthlyCharges      0.00
TotalCharges        0.16
Churn               0.00
dtype: float64
```

In [20]:
```
df.dropna(inplace=True)  #to drop the null rows
```

In [21]:
```
df.isnull().sum().sum()
```

Out[21]:
```
0
```

In [22]:
```
df.shape
```

Out[22]:
```
(7032, 21)
```

In [23]:
```
df.head()
```

Out[23]:

OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	No	No	No	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
No	Yes	No	No	No	One year	No	Mailed check	56.95	1889.50	No
Yes	No	No	No	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
No	Yes	Yes	No	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
No	No	No	No	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

In [24]:
```
df.InternetService.value_counts()
```

Out[24]:
```
Fiber optic    3096
DSL            2416
No             1520
Name: InternetService, dtype: int64
```

In [25]:
```
df.MultipleLines.value_counts()
```

Out[25]:
```
No                  3385
Yes                 2967
No phone service     680
Name: MultipleLines, dtype: int64
```

In [26]:
```
df.PhoneService.value_counts()
```

Out[26]:
```
Yes    6352
No      680
Name: PhoneService, dtype: int64
```

In [27]:
```
df.Dependents.value_counts()
```

Out[27]:
```
No     4933
Yes    2099
Name: Dependents, dtype: int64
```

In [28]:
```
df.Partner.value_counts()
```

Out[28]:
```
No     3639
Yes    3393
Name: Partner, dtype: int64
```

In [29]:
```
df.Churn.value_counts()
```

Out[29]:
```
No     5163
Yes    1869
Name: Churn, dtype: int64
```

In [30]:
```
df_val_counts = pd.DataFrame(df[['Partner', 'Dependents', 'PhoneService',  'PaperlessBilling']].value_counts())
df_val_counts
```

Out[30]:

				0
Partner	**Dependents**	**PhoneService**	**PaperlessBilling**	
No	**No**	**Yes**	**Yes**	1797
			No	1156
Yes	**No**	**Yes**	**Yes**	1006
	Yes	**Yes**	**Yes**	817
			No	763
	No	**Yes**	**No**	498
No	**No**	**No**	**Yes**	204
	Yes	**Yes**	**Yes**	162
			No	153
	No	**No**	**No**	123
Yes	**No**	**No**	**Yes**	91
	Yes	**No**	**No**	87
			Yes	73
	No	**No**	**No**	58
No	**Yes**	**No**	**No**	26
			Yes	18

In [31]:
```
ineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport','StreamingTV', 'StreamingMovies','MultipleLines']].value_counts())
```

		Yes	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	114
		No	**Yes**	**No**	**Yes**	**Yes**	**Yes**	114
	Yes	**No**	**No**	**No**	**No**	**No**	**No**	97
	No	**No**	**No**	**No**	**Yes**	**No**	**No**	93
		Yes	**No**	**No**	**No**	**No**	**No**	93
							Yes	85
		No	**No**	**No**	**No**	**Yes**	**Yes**	84
							No	83
					Yes	**Yes**	**No**	82
					No	**No**	**No phone service**	80
		Yes	**No**	**No**	**Yes**	**Yes**	**Yes**	77
		No	**Yes**	**No**	**No**	**No**	**No**	69

In [32]:
```
df.OnlineSecurity.value_counts()
```

Out[32]:
```
No                     3497
Yes                    2015
No internet service    1520
Name: OnlineSecurity, dtype: int64
```

In [ ]:

In [34]:
```
varlist =  ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']


def binary_map(x):
    return x.map({'Yes': 1, "No": 0})

#df[['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']]= df[['PhoneService', 'PaperlessBilling', 'Churn', 'Pa

df[varlist] = df[varlist].apply(binary_map)

#df["churn"].map({'Yes': 1, "No": 0})
#df[varlist].replace(['Yes', 'No'],[1, 0], inplace=True)
```

In [35]:
```
df.head()
```

Out[35]:

OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	No	No	No	No	Month-to-month	1	Electronic check	29.85	29.85	0
No	Yes	No	No	No	One year	0	Mailed check	56.95	1889.50	0
Yes	No	No	No	No	Month-to-month	1	Mailed check	53.85	108.15	1
No	Yes	Yes	No	No	One year	0	Bank transfer (automatic)	42.30	1840.75	0
No	No	No	No	No	Month-to-month	1	Electronic check	70.70	151.65	1

In [36]:
```
df.Churn.value_counts()
```

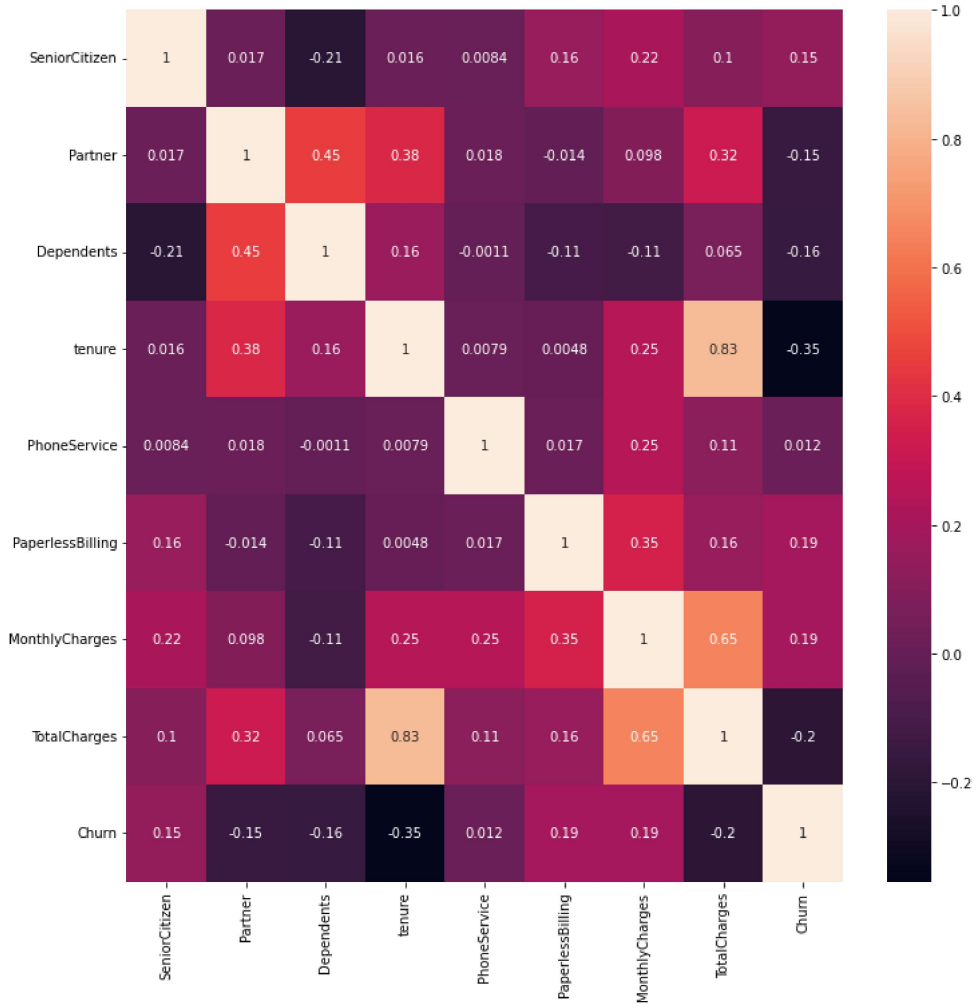
Out[36]:
```
0    5163
1    1869
Name: Churn, dtype: int64
```

In [37]:
```
df.corr()
```

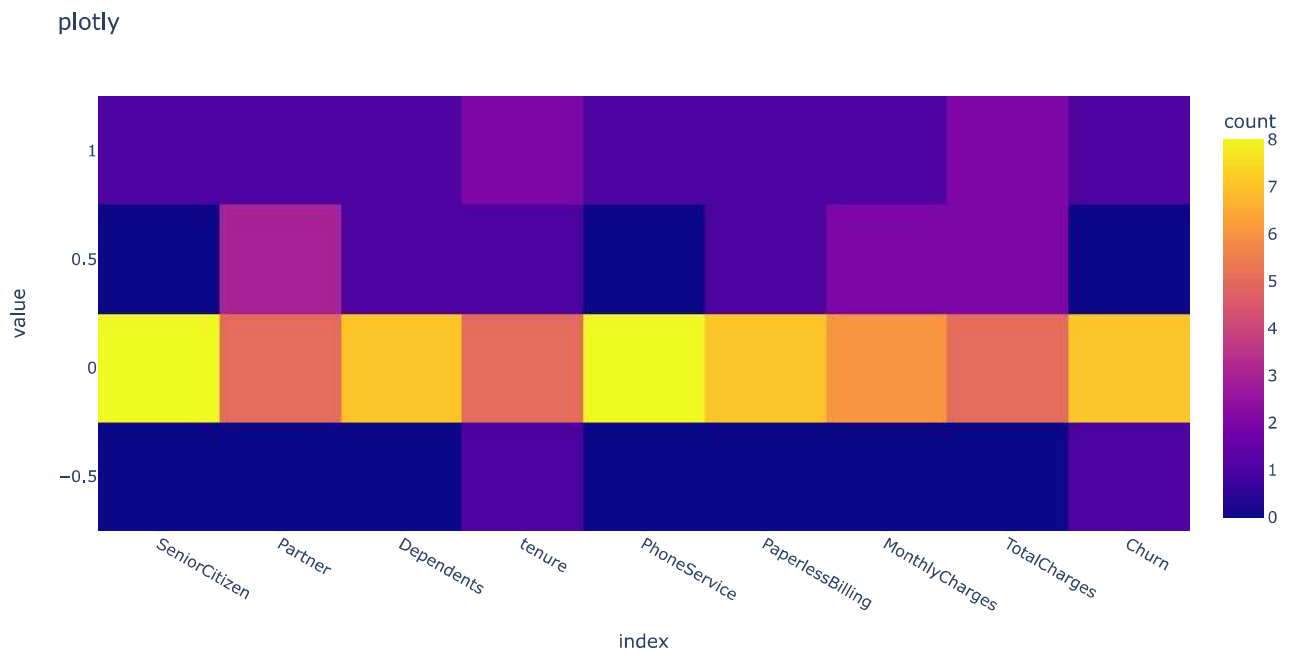
Out[37]:

	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn
SeniorCitizen	1.000000	0.016957	-0.210550	0.015683	0.008392	0.156258	0.219874	0.102411	0.150541
Partner	0.016957	1.000000	0.452269	0.381912	0.018397	-0.013957	0.097825	0.319072	-0.149982
Dependents	-0.210550	0.452269	1.000000	0.163386	-0.001078	-0.110131	-0.112343	0.064653	-0.163128
tenure	0.015683	0.381912	0.163386	1.000000	0.007877	0.004823	0.246862	0.825880	-0.354049
PhoneService	0.008392	0.018397	-0.001078	0.007877	1.000000	0.016696	0.248033	0.113008	0.011691
PaperlessBilling	0.156258	-0.013957	-0.110131	0.004823	0.016696	1.000000	0.351930	0.157830	0.191454
MonthlyCharges	0.219874	0.097825	-0.112343	0.246862	0.248033	0.351930	1.000000	0.651065	0.192858
TotalCharges	0.102411	0.319072	0.064653	0.825880	0.113008	0.157830	0.651065	1.000000	-0.199484
Churn	0.150541	-0.149982	-0.163128	-0.354049	0.011691	0.191454	0.192858	-0.199484	1.000000

In [40]:
```
plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(),annot = True)
plt.show()
```



In [41]:
```
import plotly.express as px
fig = px.density_heatmap(df.corr(), title="plotly")
fig.show()
```



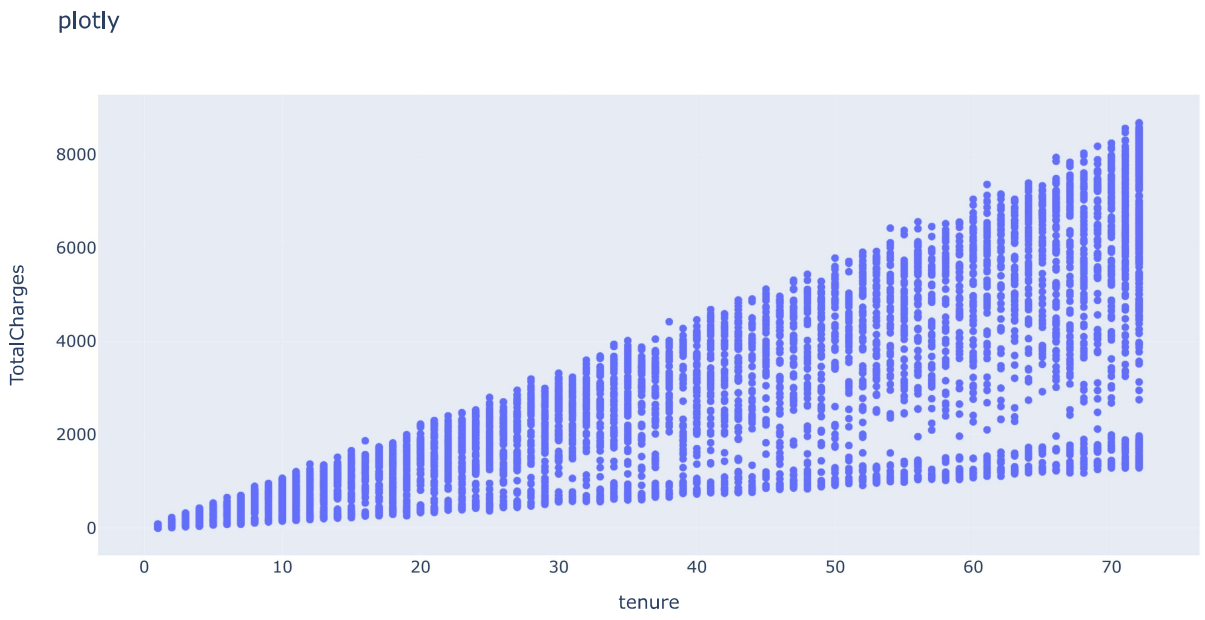
In [42]:
```
df.columns
```

Out[42]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

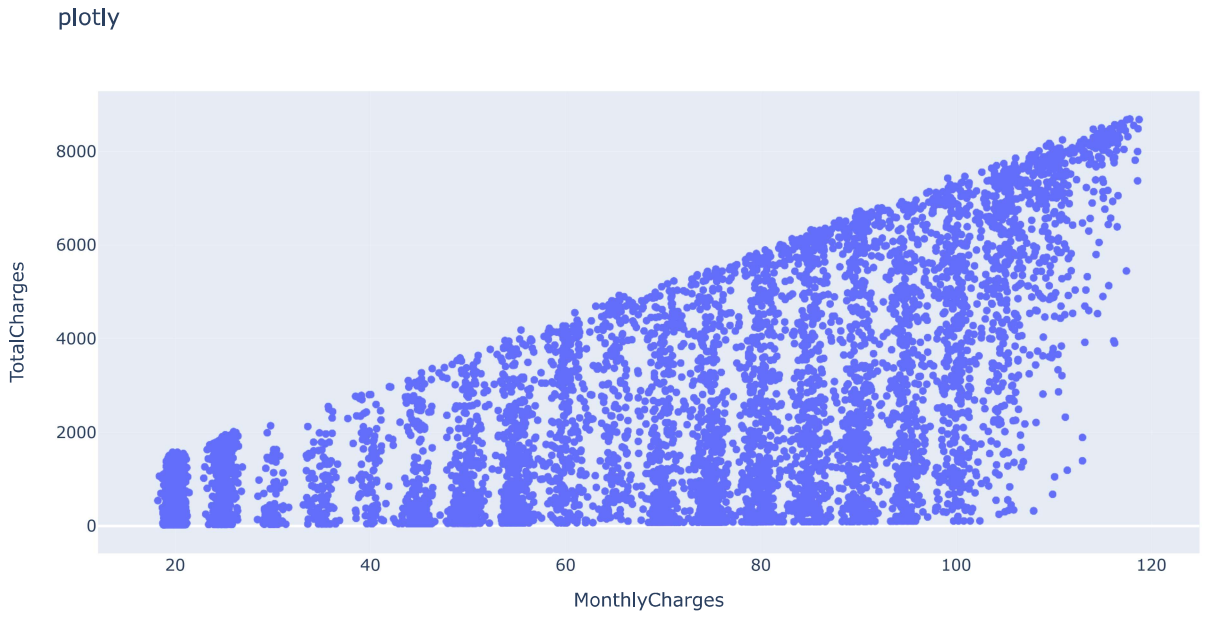
In [43]:
```
help(px)
```

```
Help on package plotly.express in plotly:

NAME
    plotly.express

DESCRIPTION
    `plotly.express` is a terse, consistent, high-level wrapper around `plotly.graph_objects`
    for rapid data exploration and figure generation. Learn more at https://plotly.express/ (https://plotly.express/)

PACKAGE CONTENTS
    _chart_types
    _core
    _doc
    _imshow
    _special_inputs
    colors (package)
    data (package)
    imshow_utils
    trendline_functions (package)
```

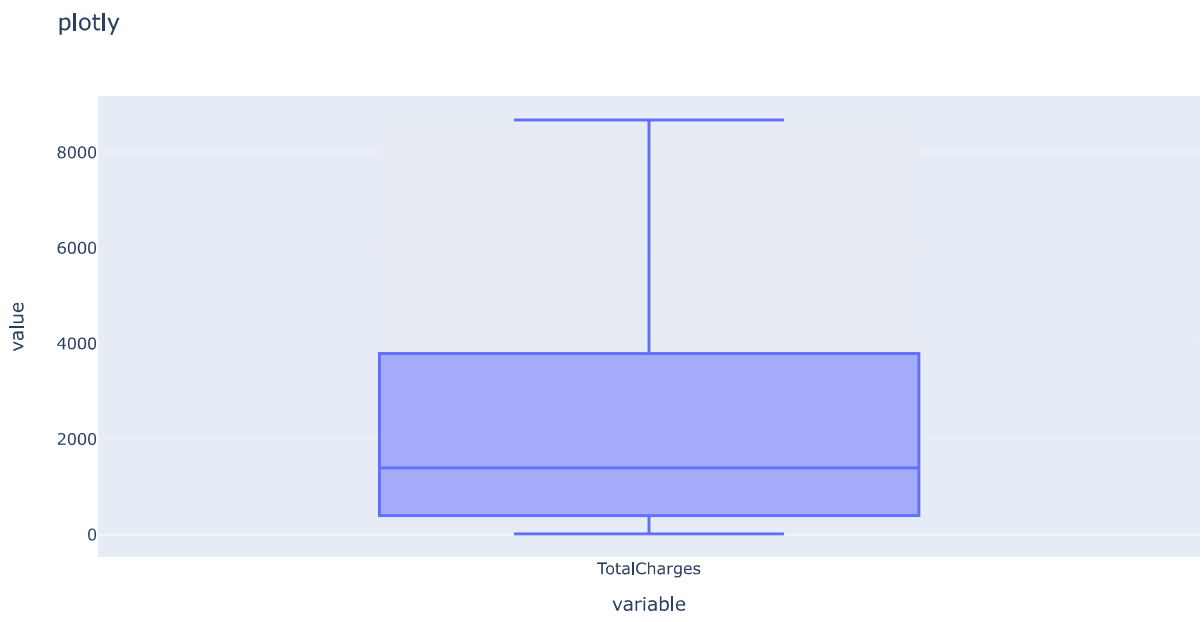
In [44]:
```
fig = px.scatter(df,x="tenure",y="TotalCharges", title="plotly")
fig.show()
```



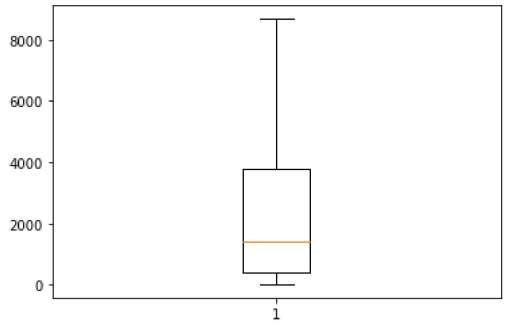
In [45]:
```
fig = px.scatter(df,x="MonthlyCharges",y="TotalCharges", title="plotly")
fig.show()
```



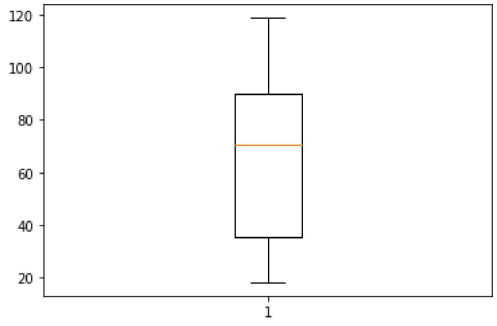
In [46]:
```
fig = px.box(df["TotalCharges"], title="plotly")
fig.show()
```



In [47]:
```
plt.boxplot(df["TotalCharges"])
plt.show()
```



In [48]:
```
plt.boxplot(df["MonthlyCharges"])
plt.show()
```



In [ ]:

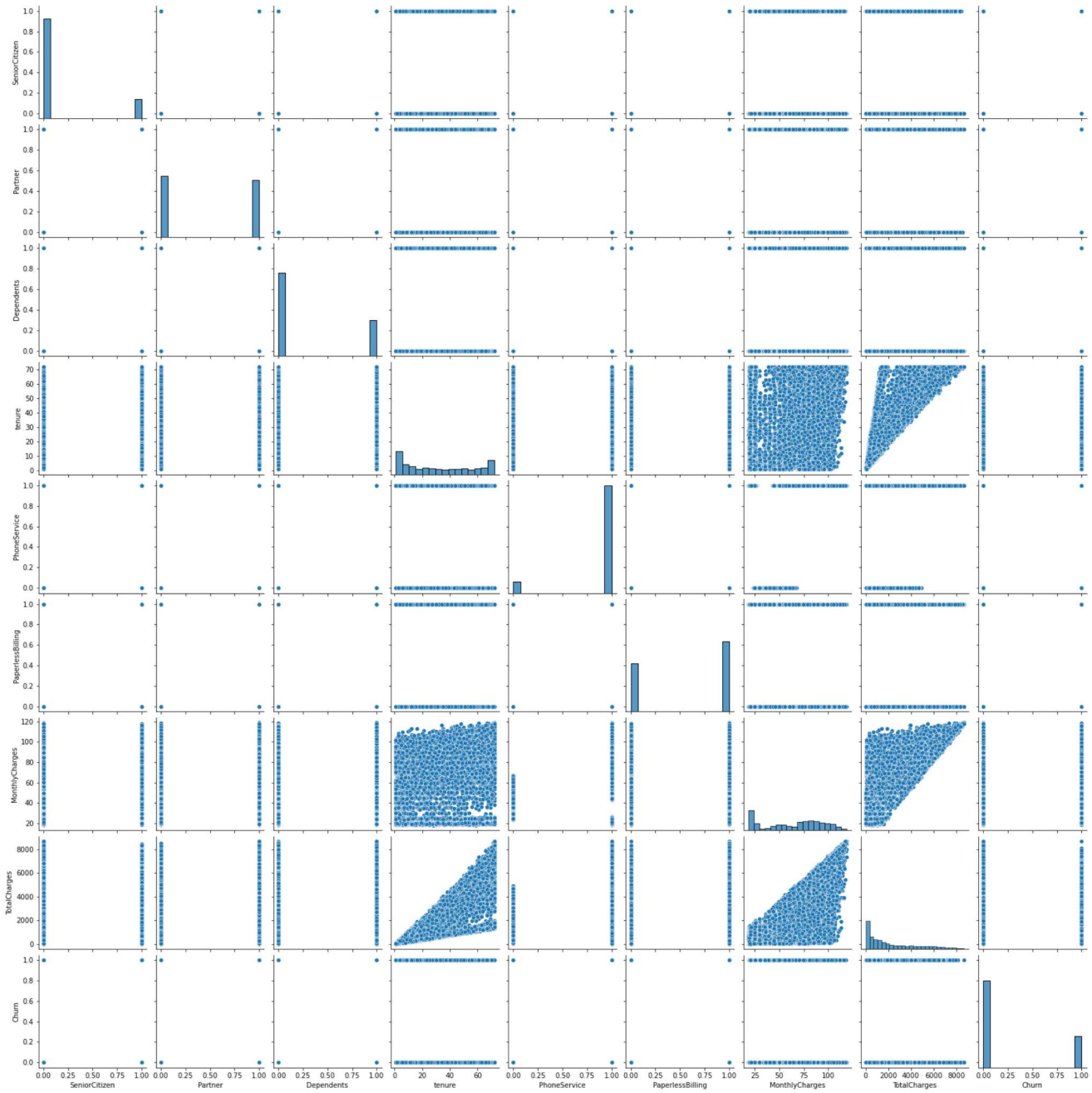
From above heatmap we can see that Tenure and Toral Charges have high Correlation. We can discard one of these but for now I am not discarding it.

Visualising Numeric Variables for better understanding

In [49]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7032 non-null   object 
 1   gender            7032 non-null   object 
 2   SeniorCitizen     7032 non-null   int64  
 3   Partner           7032 non-null   int64  
 4   Dependents        7032 non-null   int64  
 5   tenure            7032 non-null   int64  
 6   PhoneService      7032 non-null   int64  
 7   MultipleLines     7032 non-null   object 
 8   InternetService   7032 non-null   object 
 9   OnlineSecurity    7032 non-null   object 
 10  OnlineBackup      7032 non-null   object 
 11  DeviceProtection  7032 non-null   object 
 12  TechSupport       7032 non-null   object 
 13  StreamingTV       7032 non-null   object 
 14  StreamingMovies   7032 non-null   object 
 15  Contract          7032 non-null   object 
 16  PaperlessBilling  7032 non-null   int64  
 17  PaymentMethod     7032 non-null   object 
 18  MonthlyCharges    7032 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7032 non-null   int64  
dtypes: float64(2), int64(7), object(12)
memory usage: 1.2+ MB
```

In [50]:
```
sns.pairplot(df)
plt.show()
```



Kind of same is the inferences that we got from the heatmap

Visualising Categorical Variables for their better intrepretation

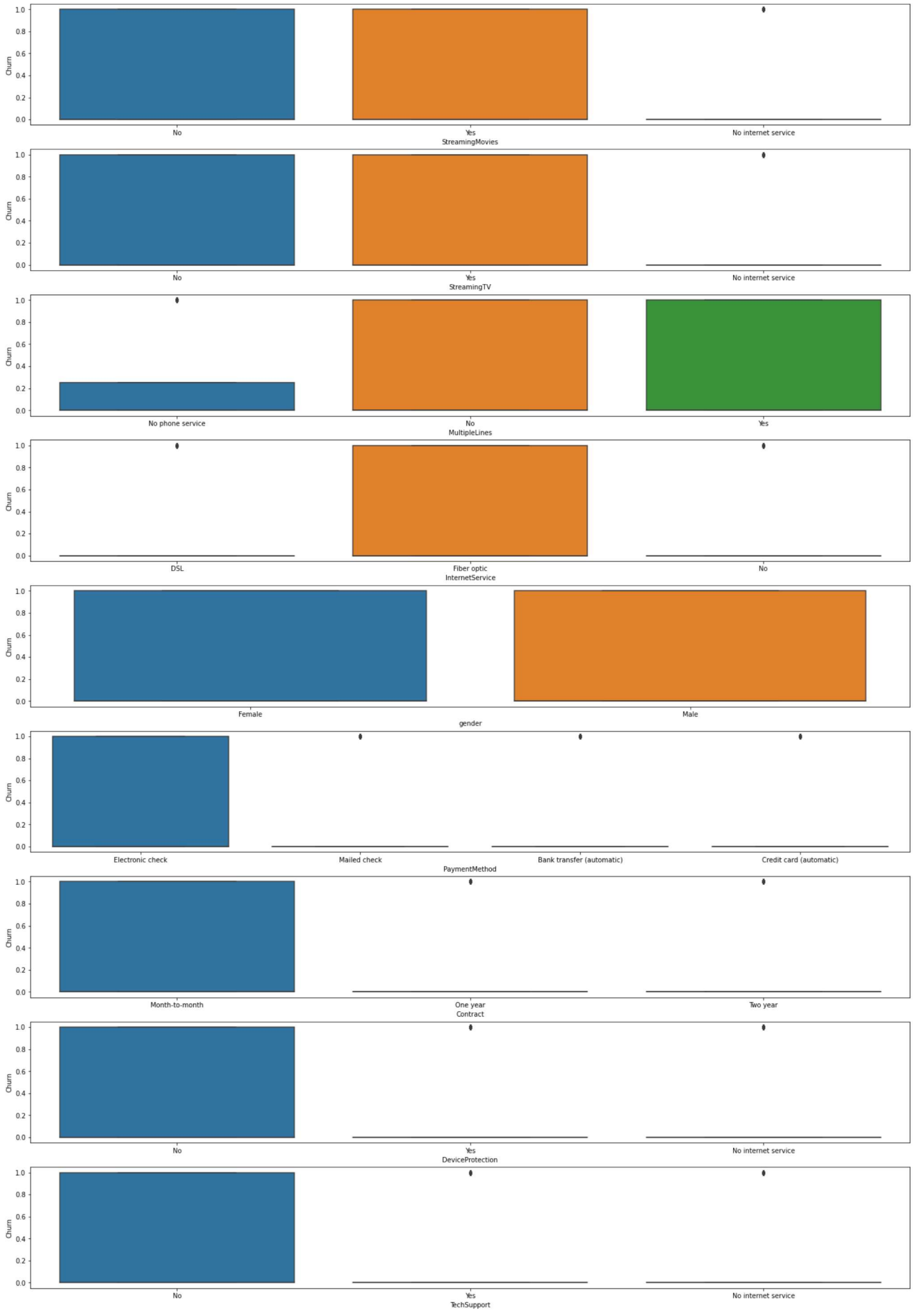
In [51]:
```
df.columns
```

Out[51]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

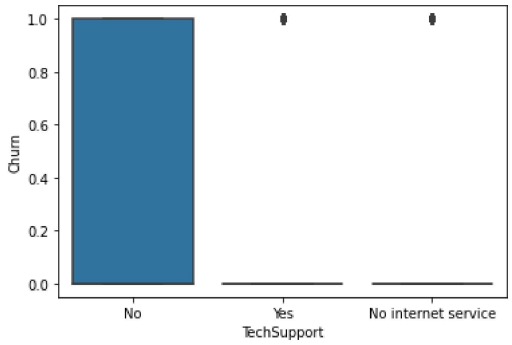
In [52]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7032 non-null   object 
 1   gender            7032 non-null   object 
 2   SeniorCitizen     7032 non-null   int64  
 3   Partner           7032 non-null   int64  
 4   Dependents        7032 non-null   int64  
 5   tenure            7032 non-null   int64  
 6   PhoneService      7032 non-null   int64  
 7   MultipleLines     7032 non-null   object 
 8   InternetService   7032 non-null   object 
 9   OnlineSecurity    7032 non-null   object 
 10  OnlineBackup      7032 non-null   object 
 11  DeviceProtection  7032 non-null   object 
 12  TechSupport       7032 non-null   object 
 13  StreamingTV       7032 non-null   object 
 14  StreamingMovies   7032 non-null   object 
 15  Contract          7032 non-null   object 
 16  PaperlessBilling  7032 non-null   int64  
 17  PaymentMethod     7032 non-null   object 
 18  MonthlyCharges    7032 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7032 non-null   int64  
dtypes: float64(2), int64(7), object(12)
memory usage: 1.4+ MB
```

In [53]:
```
plt.figure(figsize=(24, 36))
plt.subplot(9,1,1)
sns.boxplot(x = 'StreamingMovies', y = 'Churn', data = df)
plt.subplot(9,1,2)
sns.boxplot(x = 'StreamingTV', y = 'Churn', data = df)
plt.subplot(9,1,3)
sns.boxplot(x = 'MultipleLines', y = 'Churn', data = df)
plt.subplot(9,1,4)
sns.boxplot(x = 'InternetService', y = 'Churn', data = df)
plt.subplot(9,1,5)
sns.boxplot(x = 'gender', y = 'Churn', data = df)
plt.subplot(9,1,6)
sns.boxplot(x = 'PaymentMethod', y = 'Churn', data = df)
plt.subplot(9,1,7)
sns.boxplot(x = 'Contract', y = 'Churn', data = df)
plt.subplot(9,1,8)
sns.boxplot(x = 'DeviceProtection', y = 'Churn', data = df)
plt.subplot(9,1,9)
sns.boxplot(x = 'TechSupport', y = 'Churn', data = df)
plt.show()
```


Churn
No
Yes
No internet service
StreamingMovies
StreamingTV
No phone service
MultipleLines
DSL
Fiber optic
InternetService
Female
Male
gender
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod
Month-to-month
One year
Two year
Contract
DeviceProtection
TechSupport

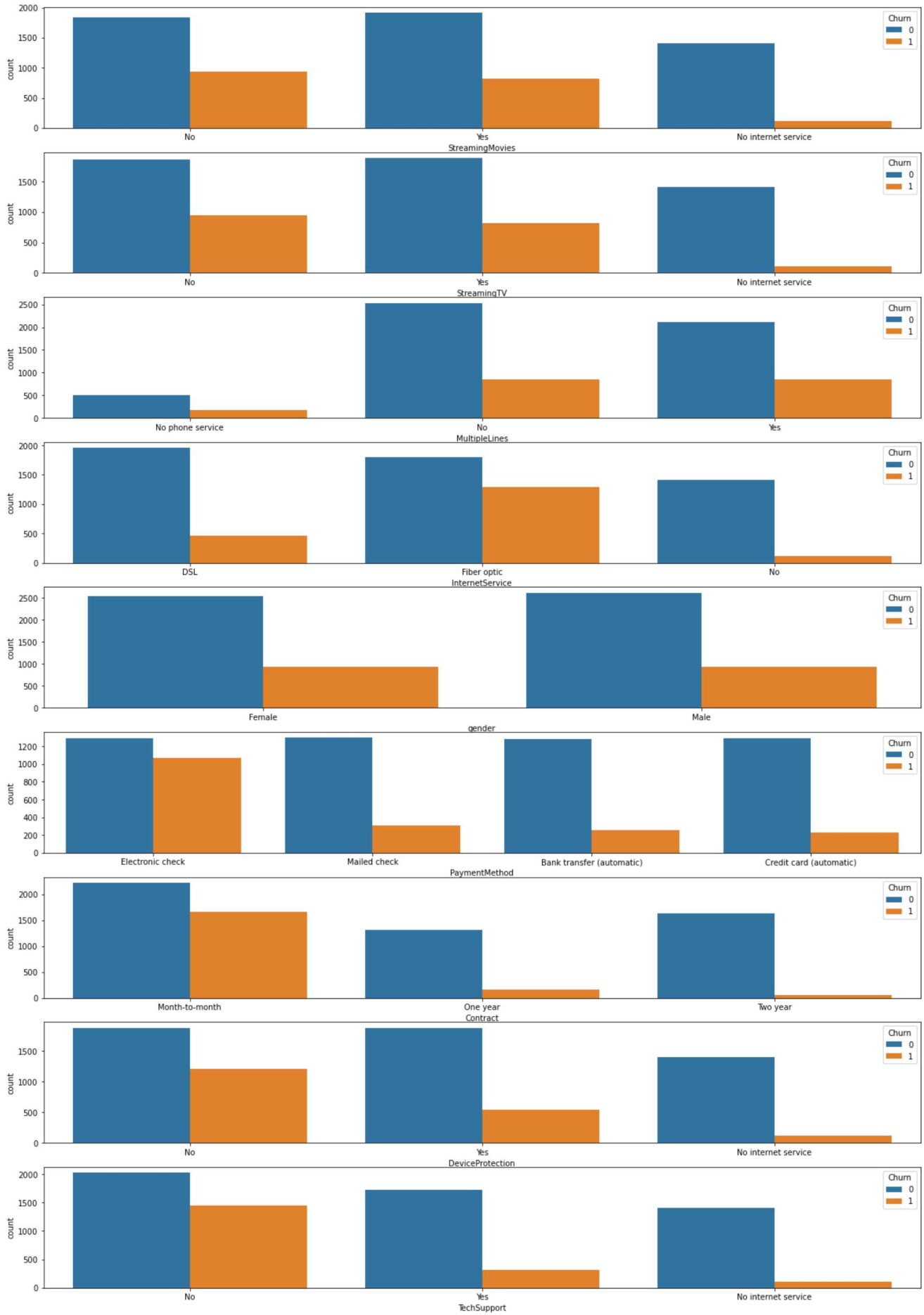
In [54]:
```
sns.boxplot(x = 'TechSupport', y = 'Churn', data = df)
plt.show()
```



In [ ]:
```
help(sns.boxplot)
```

That gave us much better understanding w.r.t churns

In [55]:
```
plt.figure(figsize=(20, 30))
plt.subplot(9,1,1)
sns.countplot(x = 'StreamingMovies', hue = 'Churn', data = df)
plt.subplot(9,1,2)
sns.countplot(x = 'StreamingTV', hue = 'Churn', data = df)
plt.subplot(9,1,3)
sns.countplot(x = 'MultipleLines', hue = 'Churn', data = df)
plt.subplot(9,1,4)
sns.countplot(x = 'InternetService', hue = 'Churn', data = df)
plt.subplot(9,1,5)
sns.countplot(x = 'gender', hue = 'Churn', data = df)
plt.subplot(9,1,6)
sns.countplot(x = 'PaymentMethod', hue = 'Churn', data = df)
plt.subplot(9,1,7)
sns.countplot(x = 'Contract', hue = 'Churn', data = df)
plt.subplot(9,1,8)
sns.countplot(x = 'DeviceProtection', hue = 'Churn', data = df)
plt.subplot(9,1,9)
sns.countplot(x = 'TechSupport', hue = 'Churn', data = df)
plt.show()
```


count
Churn
0
1
No
Yes
No internet service
StreamingMovies
StreamingTV
No phone service
MultipleLines
DSL
Fiber optic
InternetService
Female
Male
gender
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod
Month-to-month
One year
Two year
Contract
DeviceProtection
TechSupport

In [ ]:

```
```

In [56]:

```
# Creating a dummy variable for some of the categorical variables and dropping the first one.
dummy1 = pd.get_dummies(df[['Contract', 'PaymentMethod', 'gender', 'InternetService']], drop_first=True)

# Adding the results to the master dataframe
df = pd.concat([df, dummy1], axis=1)
```

In [57]:

```
df.head()
```

Out[57]:

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

In [58]:
```
# Creating dummy variables for the remaining categorical variables and dropping the level with big names.

ml = pd.get_dummies(df['MultipleLines'], prefix='MultipleLines')

ml1 = ml.drop(['MultipleLines_No phone service'], 1)

df = pd.concat([df,ml1], axis=1)


os = pd.get_dummies(df['OnlineSecurity'], prefix='OnlineSecurity')
os1 = os.drop(['OnlineSecurity_No internet service'], 1)

df = pd.concat([df,os1], axis=1)


ob = pd.get_dummies(df['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)


df = pd.concat([df,ob1], axis=1)


dp = pd.get_dummies(df['DeviceProtection'], prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)

df = pd.concat([df,dp1], axis=1)


ts = pd.get_dummies(df['TechSupport'], prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'], 1)

telecom = pd.concat([df,ts1], axis=1)


st =pd.get_dummies(df['StreamingTV'], prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'], 1)

telecom = pd.concat([df,st1], axis=1)


sm = pd.get_dummies(df['StreamingMovies'], prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'], 1)

df = pd.concat([df,sm1], axis=1)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:5: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:11: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:17: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:24: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:30: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:36: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\135075843.py:42: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
```

In [59]:
```
df.head()
```

Out[59]:

	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn	Contract_One year	Contract_Two year	F
	No	No	No	Month-to-month	1	Electronic check	29.85	29.85	0	0	0	
	No	No	No	One year	0	Mailed check	56.95	1889.50	0	1	0	
	No	No	No	Month-to-month	1	Mailed check	53.85	108.15	1	0	0	
	Yes	No	No	One year	0	Bank transfer (automatic)	42.30	1840.75	0	1	0	
	No	No	No	Month-to-month	1	Electronic check	70.70	151.65	1	0	0	

In [60]:
```
df.shape
```

Out[60]: (7032, 39)

In [61]:
```
# since we have created Dummy variables so deleting the repeated variables
df = df.drop(['Contract','PaymentMethod','gender','MultipleLines','InternetService', 'OnlineSecurity', 'OnlineBackup', 'DevicePro
       'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_7196\1106133567.py:2: FutureWarning:

In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
```

In [62]:
```
df.head(499)
```

	VHVEG										
1	5575-GNVDE	0	0	0	34	1	0	56.95	1889.50	0	1
2	3668-QPYBK	0	0	0	2	1	1	53.85	108.15	1	0
3	7795-CFOCW	0	0	0	45	0	0	42.30	1840.75	0	1
4	9237-HQITU	0	0	0	2	1	1	70.70	151.65	1	0
5	9305-CDSKC	0	0	0	8	1	1	99.65	820.50	1	0
6	1452-KIOVK	0	0	1	22	1	1	89.10	1949.40	0	0
7	6713-OKOMC	0	0	0	10	0	0	29.75	301.90	0	0
	7892-										

In [63]:
```
df.shape
```

Out[63]: (7032, 28)

In [64]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 28 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   customerID                             7032 non-null   object
 1   SeniorCitizen                          7032 non-null   int64
 2   Partner                                7032 non-null   int64
 3   Dependents                             7032 non-null   int64
 4   tenure                                 7032 non-null   int64
 5   PhoneService                           7032 non-null   int64
 6   PaperlessBilling                       7032 non-null   int64
 7   MonthlyCharges                         7032 non-null   float64
 8   TotalCharges                           7032 non-null   float64
 9   Churn                                  7032 non-null   int64
 10  Contract_One year                      7032 non-null   uint8
 11  Contract_Two year                      7032 non-null   uint8
 12  PaymentMethod_Credit card (automatic)  7032 non-null   uint8
 13  PaymentMethod_Electronic check         7032 non-null   uint8
 14  PaymentMethod_Mailed check             7032 non-null   uint8
 15  gender_Male                            7032 non-null   uint8
 16  InternetService_Fiber optic            7032 non-null   uint8
 17  InternetService_No                     7032 non-null   uint8
 18  MultipleLines_No                       7032 non-null   uint8
 19  MultipleLines_Yes                      7032 non-null   uint8
 20  OnlineSecurity_No                      7032 non-null   uint8
 21  OnlineSecurity_Yes                     7032 non-null   uint8
 22  OnlineBackup_No                        7032 non-null   uint8
 23  OnlineBackup_Yes                       7032 non-null   uint8
 24  DeviceProtection_No                    7032 non-null   uint8
 25  DeviceProtection_Yes                   7032 non-null   uint8
 26  StreamingMovies_No                     7032 non-null   uint8
 27  StreamingMovies_Yes                    7032 non-null   uint8
dtypes: float64(2), int64(7), object(1), uint8(18)
memory usage: 986.0+ KB
```

In [67]:
```
df.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[67]:

s	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMethod_
)	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7(
3	32.421786	0.903299	0.592719	64.798208	2283.300441	0.265785	0.209329	0.239619	0.216297	
9	24.545260	0.295571	0.491363	30.085974	2266.771362	0.441782	0.406858	0.426881	0.411748	
)	1.000000	0.000000	0.000000	18.250000	18.800000	0.000000	0.000000	0.000000	0.000000	
)	9.000000	1.000000	0.000000	35.587500	401.450000	0.000000	0.000000	0.000000	0.000000	
)	29.000000	1.000000	1.000000	70.350000	1397.475000	0.000000	0.000000	0.000000	0.000000	
)	55.000000	1.000000	1.000000	89.862500	3794.737500	1.000000	0.000000	0.000000	0.000000	
)	69.000000	1.000000	1.000000	102.645000	5976.640000	1.000000	1.000000	1.000000	1.000000	
)	72.000000	1.000000	1.000000	107.422500	6923.590000	1.000000	1.000000	1.000000	1.000000	
)	72.000000	1.000000	1.000000	114.734500	8039.883000	1.000000	1.000000	1.000000	1.000000	
)	72.000000	1.000000	1.000000	118.750000	8684.800000	1.000000	1.000000	1.000000	1.000000	

In [68]:
```
from sklearn.model_selection import train_test_split
```

In [79]:
```
help(train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._split:

train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
    Split arrays or matrices into random train and test subsets.

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and application to input data
    into a single call for splitting (and optionally subsampling) data in a
    oneliner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the proportion
        of the dataset to include in the test split. If int, represents the
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it will
        be set to 0.25.

    train_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int, RandomState instance or None, default=None
        Controls the shuffling applied to the data before applying the split.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.

    shuffle : bool, default=True
        Whether or not to shuffle the data before splitting. If shuffle=False
        then stratify must be None.

    stratify : array-like, default=None
        If not None, data is split in a stratified fashion, using this as
        the class labels.
        Read more in the :ref:`User Guide <stratification>`.

    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same as the
            input type.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_test_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
```

```
[[0, 1, 2], [3, 4]]
```

In [69]:
```
X = df.drop(['Churn','customerID'], axis=1)

X.head()
```

Out[69]:

honeService	PaperlessBilling	MonthlyCharges	TotalCharges	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMethod_Electronic check	PaymentMethod_Mai ch
0	1	29.85	29.85	0	0	0	1	
1	0	56.95	1889.50	1	0	0	0	
1	1	53.85	108.15	0	0	0	0	
0	0	42.30	1840.75	1	0	0	0	
1	1	70.70	151.65	0	0	0	1	

In [70]:
```
y = df['Churn']

y.head()
```

Out[70]:
```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

Data Preparation for model building

In [71]:
```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)
```

In [72]:
```
X_train.shape
```

Out[72]:
```
(4922, 26)
```

In [73]:
```
X_train.head()
```

Out[73]:

e	PaperlessBilling	MonthlyCharges	TotalCharges	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMethod_Electronic check	PaymentMethod_Mailed check	gende
1	1	54.65	1665.20	0	0	0	0	0	
0	1	50.85	2036.55	0	0	0	1	0	
1	1	82.30	82.30	0	0	0	1	0	
1	1	110.10	1043.30	0	0	0	1	0	
1	1	98.10	396.30	0	0	0	0	0	

In [74]:
```
X_test.shape
```

Out[74]:
```
(2110, 26)
```

In [75]:
```
from sklearn.preprocessing import StandardScaler
```

In [76]:
```
help(StandardScaler)
```

```
    The standard score of a sample `x` is calculated as:

        z = (x - u) / s

    where `u` is the mean of the training samples or zero if `with_mean=False`,
    and `s` is the standard deviation of the training samples or one if
    `with_std=False`.

    Centering and scaling happen independently on each feature by computing
    the relevant statistics on the samples in the training set. Mean and
    standard deviation are then stored to be used on later data using
    :meth:`transform`.

    Standardization of a dataset is a common requirement for many
    machine learning estimators: they might behave badly if the
    individual features do not more or less look like standard normally
    distributed data (e.g. Gaussian with 0 mean and unit variance).

    For instance many elements used in the objective function of
    a learning algorithm (such as the RBF kernel of Support Vector
```

In [80]:
```
scaler = StandardScaler()

X_train[['tenure','MonthlyCharges','TotalCharges']] = scaler.fit_transform(X_train[['tenure','MonthlyCharges','TotalCharges']])

X_train.head()
```

Out[80]:

honeService	PaperlessBilling	MonthlyCharges	TotalCharges	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMethod_Electronic check	PaymentMethod_Ma ch
1	1	-0.338074	-0.276449	0	0	0	0	
0	1	-0.464443	-0.112702	0	0	0	1	
1	1	0.581425	-0.974430	0	0	0	1	
1	1	1.505913	-0.550676	0	0	0	1	
1	1	1.106854	-0.835971	0	0	0	0	

In [82]:
```
help(scaler)
```

```
        X : array-like of shape (n_samples, n_features)
            Input samples.

        y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                default=None
            Target values (None for unsupervised transformations).

        **fit_params : dict
            Additional fit parameters.

        Returns
        -------
        X_new : ndarray array of shape (n_samples, n_features_new)
            Transformed array.

    ----------------------------------------------------------------------
    Methods inherited from sklearn.base.BaseEstimator:

    __getstate__(self)

    __repr__(self, N_CHAR_MAX=700)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: