practiceQuestions2(Answers)

October 15, 2023

[1]:

```
import pandas as pd
import numpy as np
```

[2]:

```
#Reading the CSV file
telco = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

[4]:

```
#Displaying column names to identify target variable
columnNames = telco.columns.tolist()
print("Columns: ", columnNames)
```

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

[5]:

```
#1.
#Identifying the target variable
targetVariable = 'Churn'
```

[6]:

```
#Counting the number of customers who have churned
telco["Churn"].value_counts()
```

[6]:

```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

2. The problem we are facing is a binary classification problem in which the datapoint are limited to only two classes(such as yes or no)

[8]:

```
#3.
telco.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
```

[9]:

```
#4.
#Checking for missing values
missingValues = telco.isnull().sum()
print("Missing values:\n", missingValues)
```

```
Missing values:
 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
```

4. There are no missing values

[10]:

```
#5.
#Checking for duplicate rows based on all columns
telcoDuplicate = telco.duplicated(keep=False)

#Displaying duplicate rows
print(telco[telcoDuplicate])
```

```
Empty DataFrame
Columns: [customerID, gender, SeniorCitizen, Partner, Dependents, tenure,
PhoneService, MultipleLines, InternetService, OnlineSecurity, OnlineBackup,
DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract,
PaperlessBilling, PaymentMethod, MonthlyCharges, TotalCharges, Churn]
Index: []

[0 rows x 21 columns]
```

5. After executing the above code, it is clear that there are no duplicate rows in the dataset.

[11]:

```
#6. Identifying categorical features
telco.dtypes
```

[11]:

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
```

```
dtype: object
```

[12]:
```
#6. I transformed the data using label encoding that assigns a unique numerical␣
 ↪value to each category. The reason why this
#code cannot execute is because I have already encoded it previously. The␣
 ↪verification is among the codes.
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['customerID'] = labelEncoder.fit_transform(telco['customerID'])
```

[13]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['gender'] = labelEncoder.fit_transform(telco['gender'])
```

[14]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['Partner'] = labelEncoder.fit_transform(telco['Partner'])
```

[15]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['Dependents'] = labelEncoder.fit_transform(telco['Dependents'])
```

[16]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['PhoneService'] = labelEncoder.fit_transform(telco['PhoneService'])
```

[17]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['MultipleLines'] = labelEncoder.fit_transform(telco['MultipleLines'])
```

[18]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['InternetService'] = labelEncoder.fit_transform(telco['InternetService'])
```

[19]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['OnlineSecurity'] = labelEncoder.fit_transform(telco['OnlineSecurity'])
```

[20]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['OnlineBackup'] = labelEncoder.fit_transform(telco['OnlineBackup'])
```

[21]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['DeviceProtection'] = labelEncoder.
 ↪fit_transform(telco['DeviceProtection'])
```

[22]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['TechSupport'] = labelEncoder.fit_transform(telco['TechSupport'])
```

[23]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['StreamingTV'] = labelEncoder.fit_transform(telco['StreamingTV'])
```

[24]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['StreamingMovies'] = labelEncoder.fit_transform(telco['StreamingMovies'])
```

[25]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['Contract'] = labelEncoder.fit_transform(telco['Contract'])
```

[26]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['PaperlessBilling'] = labelEncoder.
 ↪fit_transform(telco['PaperlessBilling'])
```

[27]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['PaymentMethod'] = labelEncoder.fit_transform(telco['PaymentMethod'])
```

[28]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['TotalCharges'] = labelEncoder.fit_transform(telco['TotalCharges'])
```

[29]:

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
telco['Churn'] = labelEncoder.fit_transform(telco['Churn'])
```

[30]:

```
telco.dtypes
```

[30]:

```
customerID           int32
gender               int32
SeniorCitizen        int64
Partner              int32
Dependents           int32
tenure               int64
PhoneService         int32
MultipleLines        int32
InternetService      int32
OnlineSecurity       int32
OnlineBackup         int32
DeviceProtection     int32
```

```
 TechSupport           int32
 StreamingTV           int32
 StreamingMovies       int32
 Contract              int32
 PaperlessBilling      int32
 PaymentMethod         int32
 MonthlyCharges      float64
 TotalCharges          int32
 Churn                 int32
 dtype: object
```

[31]:
```python
telco.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   int32
 1   gender            7043 non-null   int32
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   int32
 4   Dependents        7043 non-null   int32
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   int32
 7   MultipleLines     7043 non-null   int32
 8   InternetService   7043 non-null   int32
 9   OnlineSecurity    7043 non-null   int32
 10  OnlineBackup      7043 non-null   int32
 11  DeviceProtection  7043 non-null   int32
 12  TechSupport       7043 non-null   int32
 13  StreamingTV       7043 non-null   int32
 14  StreamingMovies   7043 non-null   int32
 15  Contract          7043 non-null   int32
 16  PaperlessBilling  7043 non-null   int32
 17  PaymentMethod     7043 non-null   int32
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   int32
 20  Churn             7043 non-null   int32
dtypes: float64(1), int32(18), int64(2)
memory usage: 660.4 KB
```

[33]:
```python
#7.
churnDistributions = telco['Churn'].value_counts()
print(churnDistributions)
```

```
Churn
0    5174
```

```
1    1869
Name: count, dtype: int64
```

8. An outlier is a datapoint that is separated from other datapoints in a dataset. For instance, if a test was conducted, and majority of the students got good grades like 10,10,10,9,9,10,8,… and so forth but someone got 1, it means that 1 is an outlier because it deviates from the other scores.

[39]:
```
#9.
#Checking for outliers
#Q1 = telco['Churn'].quantile(0.25)
#Q3 = telco['Churn'].quantile(0.25)
Q1 = telco.quantile(0.25)
Q3 = telco.quantile(0.75)
IQR = Q3 - Q1

lowerBound = Q1 - 1.5 * IQR
upperBound = Q3 + 1.5 * IQR

#outliers = telco[(telco['Churn'] < lowerBound) | (telco['Churn'] > upperBound)]
#identifying outliers using boolean indexing
outliers = (telco < lowerBound) | (telco > upperBound)
#print(outliers)

#Displaying outliers for each column
outlierRows = telco[outliers.any(axis=1)]
print(outlierRows)
```

```
      customerID  gender  SeniorCitizen  Partner  Dependents  tenure  \
0           5375       0              0        1           0       1
3           5535       1              0        0           0      45
7           4770       0              0        0           0      10
20          6207       1              1        0           0       1
27          6119       1              0        1           1       1
…            …       …            …       …          …      …
7031        2521       1              1        1           0      55
7032        4893       1              1        0           0       1
7036        5504       0              0        0           0      12
7040        3367       0              0        1           1      11
7041        5934       1              1        1           0       4

      PhoneService  MultipleLines  InternetService  OnlineSecurity  …  \
0                0              1                0               0  …
3                0              1                0               2  …
7                0              1                0               2  …
20               0              1                0               0  …
27               0              1                0               0  …
…              …              …                …               …  …
7031             1              2                0               2  …
```

```
7032             1              2                1               0  …
7036             0              1                0               0  …
7040             0              1                0               2  …
7041             1              2                1               0  …

      DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  \
0                    0            0            0                0         0
3                    2            2            0                0         1
7                    0            0            0                0         0
20                   2            0            0                2         0
27                   0            0            0                0         0
…                …          …          …              …       …
7031                 0            0            0                0         1
7032                 0            0            0                0         0
7036                 2            2            2                2         1
7040                 0            0            0                0         0
7041                 0            0            0                0         0

      PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn
0                    1              2           29.85          2505      0
3                    0              0           42.30          1400      0
7                    0              3           29.75          2609      0
20                   1              2           39.65          3340      1
27                   0              2           30.20          2592      1
…                …            …             …           …    …
7031                 0              1           60.00          2880      0
7032                 1              2           75.75          5776      1
7036                 0              2           60.65          5741      0
7040                 1              2           29.60          2994      0
7041                 1              3           74.40          2660      1

[1720 rows x 21 columns]
```

[57]:

```
from sklearn.preprocessing import QuantileTransformer
#9
#Transforming outliers using quantile transformation
quantileTransformer = QuantileTransformer(output_distribution='normal')
telco['ChurnTransformed'] = quantileTransformer.fit_transform(telco[['Churn']])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[57], line 5
      2 #9
      3 #Transforming outliers using quantile transformation
      4 quantileTransformer = QuantileTransformer(output_distribution='normal')
----> 5 telco['ChurnTransformed'] = quantileTransformer.
 ↪fit_transform(telco[['Churn']])
```

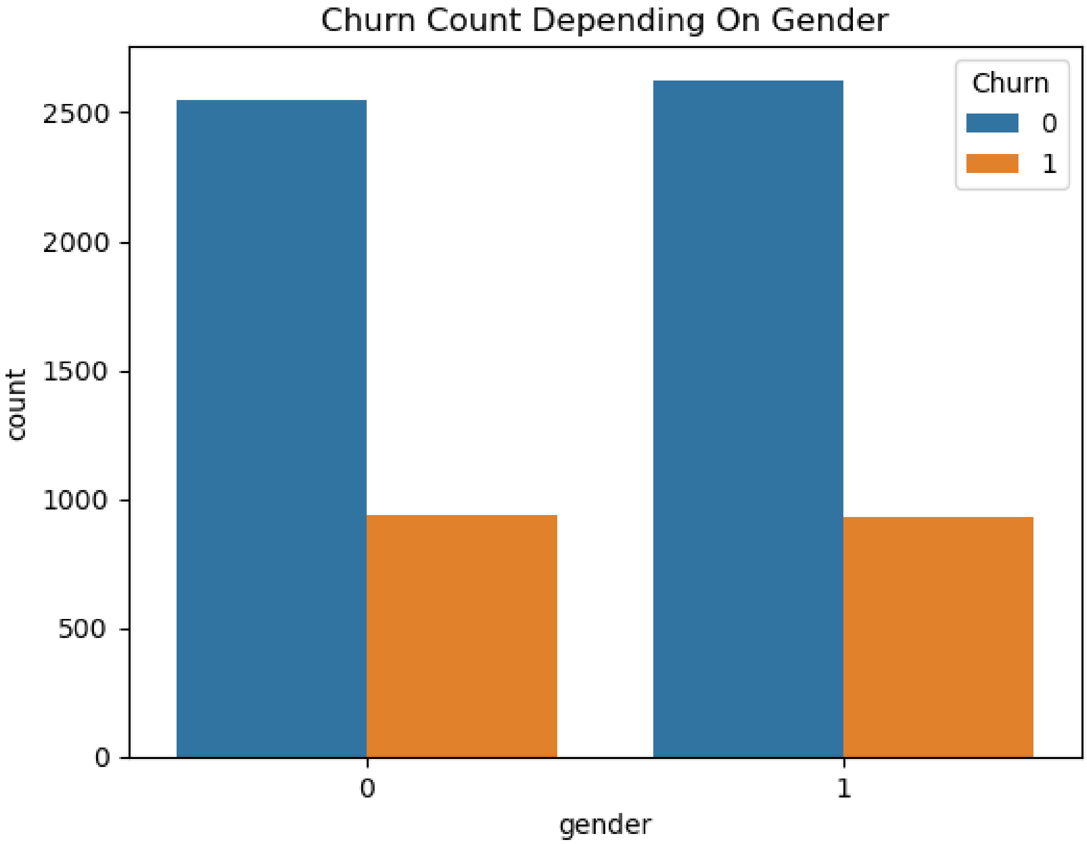
```
File ~\anaconda3\Lib\site-packages\sklearn\utils\_set_output.py:140, in _wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    138 @wraps(f)
    139 def wrapped(self, X, *args, **kwargs):
--> 140     data_to_wrap = f(self, X, *args, **kwargs)
    141     if isinstance(data_to_wrap, tuple):
    142         # only wrap the first output for cross decomposition
    143         return_tuple = (
    144             _wrap_data_with_container(method, data_to_wrap[0], X, self)
    145             *data_to_wrap[1:],
    146         )

File ~\anaconda3\Lib\site-packages\sklearn\base.py:915, in TransformerMixin.fit_transform(self, X, y, **fit_params)
    911 # non-optimized default implementation; override when a better
    912 # method is possible for a given clustering algorithm
    913 if y is None:
    914     # fit method of arity 1 (unsupervised transformation)
--> 915     return self.fit(X, **fit_params).transform(X)
    916 else:
    917     # fit method of arity 2 (supervised transformation)
    918     return self.fit(X, y, **fit_params).transform(X)

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1151, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\Lib\site-packages\sklearn\preprocessing\_data.py:2663, in QuantileTransformer.fit(self, X, y)
   2656 if self.n_quantiles > self.subsample:
   2657     raise ValueError(
   2658         "The number of quantiles cannot be greater than"
   2659         " the number of samples used. Got {} quantiles"
   2660         " and {} samples.".format(self.n_quantiles, self.subsample)
   2661     )
-> 2663 X = self._check_inputs(X, in_fit=True, copy=False)
   2664 n_samples = X.shape[0]
   2666 if self.n_quantiles > n_samples:
```

```
File ~\anaconda3\Lib\site-packages\sklearn\preprocessing\_data.py:2752, in QuantileTransformer._check_inputs(self, X, in_fit, accept_sparse_negative, copy)
  2750 def _check_inputs(self, X, in_fit, accept_sparse_negative=False, copy=False):
  2751     """Check inputs before fit and transform."""
-> 2752     X = self._validate_data(
  2753         X,
  2754         reset=in_fit,
  2755         accept_sparse="csc",
  2756         copy=copy,
  2757         dtype=FLOAT_DTYPES,
  2758         force_all_finite="allow-nan",
  2759     )
  2760     # we only accept positive sparse matrix when ignore_implicit_zeros  s
  2761     # false and that we call fit or transform.
  2762     with np.errstate(invalid="ignore"):  # hide NaN comparison warnings

File ~\anaconda3\Lib\site-packages\sklearn\base.py:604, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **check_params)
    602     out = X, y
    603 elif not no_val_X and no_val_y:
--> 604     out = check_array(X, input_name="X", **check_params)
    605 elif no_val_X and not no_val_y:
    606     out = _check_y(y, **check_params)

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:959, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    953         raise ValueError(
    954             "Found array with dim %d. %s expected <= 2."
    955             % (array.ndim, estimator_name)
    956         )
    958     if force_all_finite:
--> 959         _assert_all_finite(
    960             array,
    961             input_name=input_name,
    962             estimator_name=estimator_name,
    963             allow_nan=force_all_finite == "allow-nan",
    964         )
    966 if ensure_min_samples > 0:
    967     n_samples = _num_samples(array)

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:124, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    121 if first_pass_isfinite:
    122     return
```

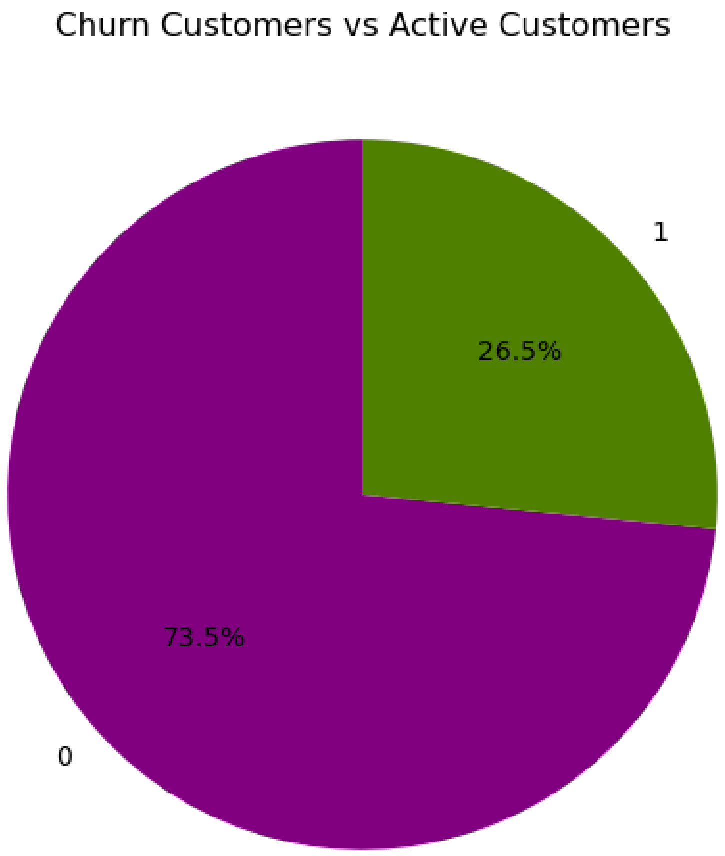
```
--> 124 _assert_all_finite_element_wise(
    125     X,
    126     xp=xp,
    127     allow_nan=allow_nan,
    128     msg_dtype=msg_dtype,
    129     estimator_name=estimator_name,
    130     input_name=input_name,
    131 )

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:173, in␣
 ↪_assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name,␣
 ↪input_name)
    156 if estimator_name and input_name == "X" and has_nan_error:
    157     # Improve the error message on how to handle missing values in
    158     # scikit-learn.
    159     msg_err += (
    160         f"\n{estimator_name} does not accept missing values"
    161         " encoded as NaN natively. For supervised learning, you might␣
 ↪want"
   (…)
    171         "#estimators-that-handle-nan-values"
    172     )
--> 173 raise ValueError(msg_err)

ValueError: Input X contains infinity or a value too large for dtype('float64')
```

[40]:

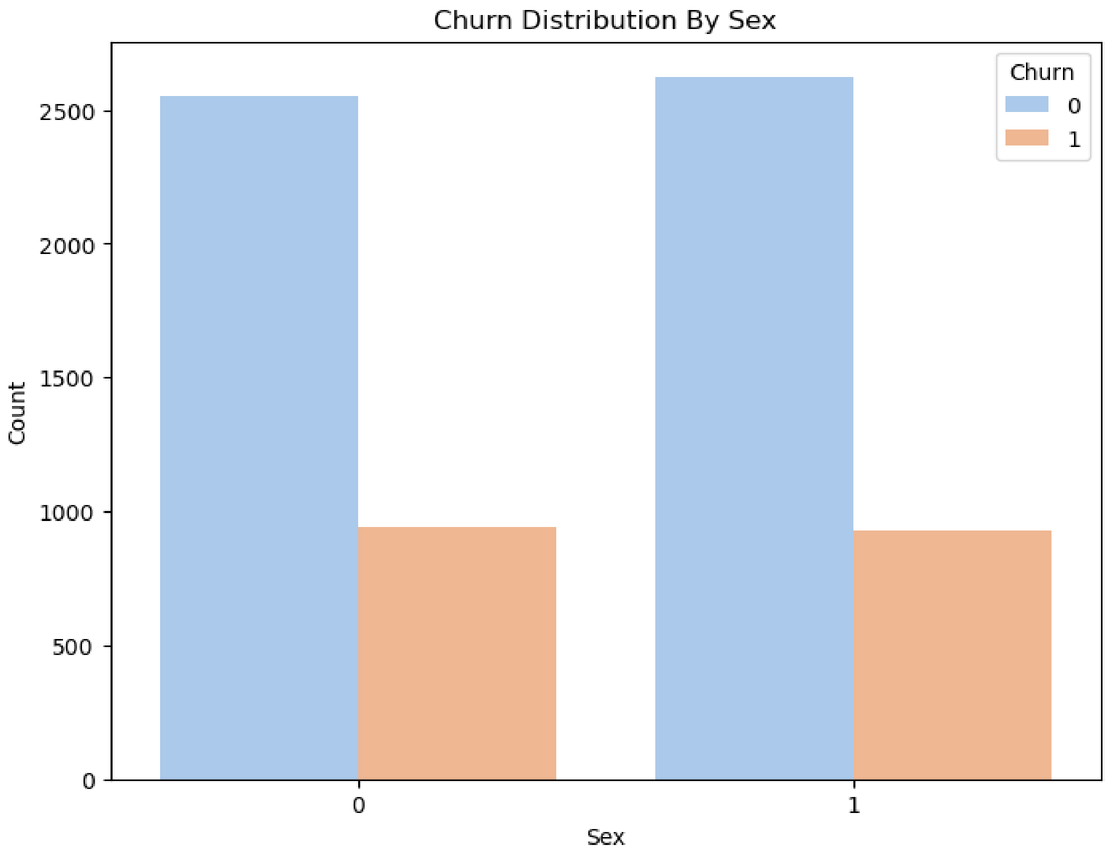
```
#10.
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='gender', hue='Churn', data=telco)
plt.title('Churn Count Depending On Gender')
plt.show()
```



[48]:
```
#11.
#Computing percentage of churn customers and active customers
percentageChurn = telco['Churn'].value_counts(normalize=True) * 100
#Using pie chart to visualize
plt.figure(figsize=(6, 6))
plt.pie(percentageChurn, labels=percentageChurn.index, autopct='%1.1f%%',␣
 ↪startangle=90, colors=['#800080', '#508000'])
plt.title('Churn Customers vs Active Customers')
plt.show()
```



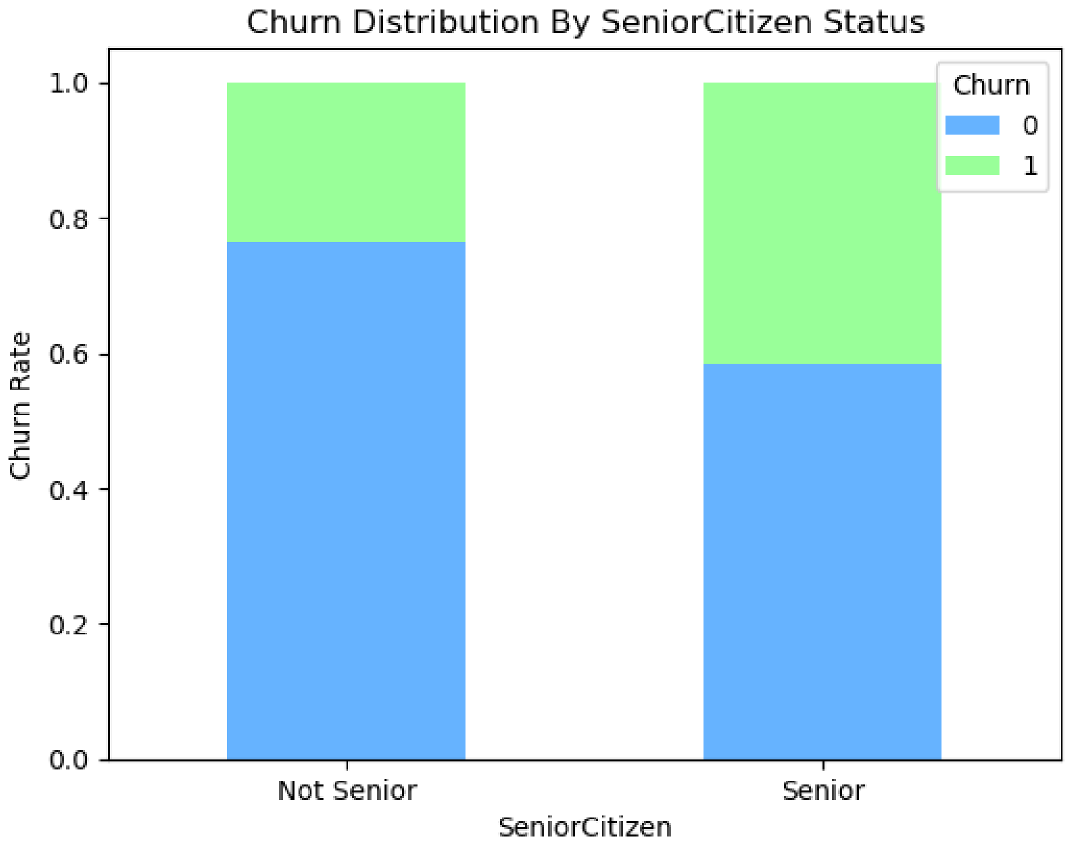
[49]:
```
#12. 1st visualization
#Plotting churn distribution based on sex
plt.figure(figsize=(8, 6))
sns.countplot(x='gender', hue='Churn', data=telco, palette='pastel')
plt.title('Churn Distribution By Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
```



[51]:

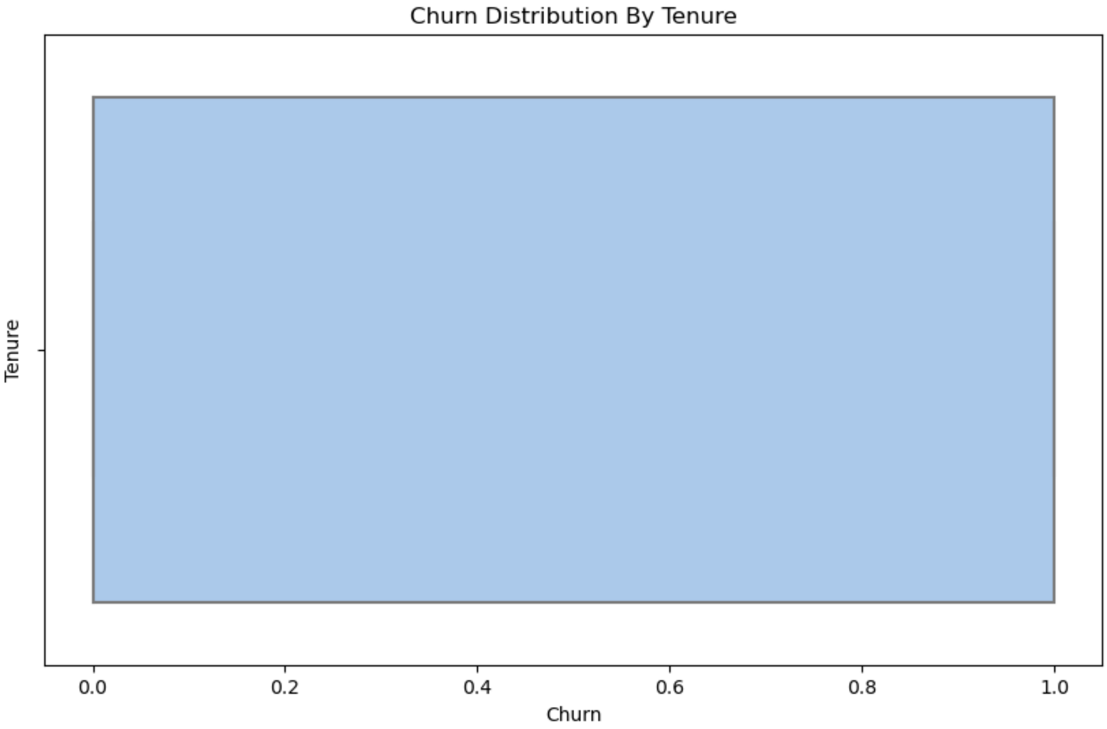
```
#12. 2nd visualization
#Plotting churn rate based on senior citizen status
SeniorChurn = telco.groupby('SeniorCitizen')['Churn'].
 ↪value_counts(normalize=True).unstack()

plt.figure(figsize=(8, 6))
SeniorChurn.plot(kind='bar', stacked=True, color=['#66b3ff','#99ff99'])
plt.title('Churn Distribution By SeniorCitizen Status')
plt.xlabel('SeniorCitizen')
plt.ylabel('Churn Rate')
plt.xticks(ticks=[0, 1], labels=['Not Senior', 'Senior'], rotation=0)
plt.legend(title='Churn', loc='upper right')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



[52]:

```
#12. 3rd visualization
#Visualizing churn rate based on tenure
plt.figure(figsize=(10, 6))
sns.boxplot(x='Churn', hue='tenure', data=telco, palette='pastel')
plt.title('Churn Distribution By Tenure')
plt.xlabel('Churn')
plt.ylabel('Tenure')
plt.show()
```


Churn Distribution By Tenure
Tenure
0.0
0.2
0.4
0.6
0.8
1.0
Churn