Modelling Monthly Adjusted Paid Claims Weighting for Renewal Projections

June 29, 2022

[66]:
```
# Run this cell first!
from datascience import *
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Modelling Monthly Adjusted Paid Claims Weighting for Renewal Projections

The basis for this project is to create a model for the weights of each month's adjusted paid claims values (1-36) when calculating renewal rates for Huntington Medical Group. It is important to note that this model will generally be relaible only for this particular client. In a further analysis, we can look at whether this model can potentially be generalizable across other similar clients.

1.1 Background

For this project, the main focus will be on employer-sponsored health insurance renewals and using monthly Adjusted Paid Claims data to help us predict renewal rates. Our data will be the complete historical set of C&E reports for Huntington Hospital starting in 2015. We'll be using this to essentially create a model that determines how much we should weight each months' Adjusted Paid Claims values when predicting the yearly Adjusted Paid Claims value for the following year. Specifically, we've been tasked to find weights for the previous 36 months (e.g. 3 years).

1.1.1 Basic Methodology

Since our goal is to predict annual renewal rate increases, the granularity of the data will be yearly (i.e. each row will represent a **Renewal Year** and not a standard year). However, because Huntington Medical Group has data going back only to 2015, this doesn't give us many data points to use to train our model. To remedy this, we'll want to try and bootstrap the existing data to reconstruct the population from our sample and use the reconstructed population to train our model. *It is essential to note, however, that this method only works if the sample we are using to reconstruct the population is a representative sample.* Let's perform some EDA on the data to break this down.

1.2 Exploratory Data Analysis

1.2.1 Data

We have an initial dataset to use. Let's load it in and perform some EDA on it to determine whether our bootstrapping method is appropriate. This data was generated from Huntington Medical Group's historical Monthly C&E Reports.

[67]:
```
claims = pd.read_csv('Huntington Medical Paid Claims Data.csv')
claims
```

[67]:

	Month	Year	Renewal Year	Domestic Medical Claims	Non-Domestic Medical Claims	Total Hospital Medical Claims	Non-Hospital Medical Claims	Total Medical Claims	Rx Claims	Rx Rebates
0	1	2015	0	414313.73	324468.71	738782.44	385900.53	1124682.97	397849.95	0.00
1	2	2015	0	539505.55	307763.94	847269.49	507899.40	1355168.89	503271.97	67540.50
2	3	2015	0	582799.17	272127.91	854927.08	539969.58	1394896.66	406481.14	0.00
3	4	2015	0	894533.91	253610.74	1148144.65	593447.65	1741592.30	533681.67	0.00
4	5	2015	0	732374.57	424912.54	1157287.11	419625.09	1576912.20	451070.89	66067.50
..	…	…	…	…	…	…	…	…	…	…
84	1	2022	7	501463.78	474951.54	976415.32	619224.78	1595640.10	603637.45	0.00
85	2	2022	7	702513.46	541335.00	1243848.46	1140214.23	2384062.69	650981.28	370485.73
86	3	2022	7	760937.71	588458.73	1349396.44	943089.50	2292485.94	788584.54	0.00
87	4	2022	7	579591.91	513879.57	1093471.48	963612.60	2057084.08	659029.75	0.00
88	5	2022	7	462605.24	651661.71	1114266.95				

```
88                   1016179.46            2130446.41  627878.54        0.00

    Rx Performance Guarantee  Stop Loss Reimbursement  Adjusted Paid Claims  \
0                       0.00                      0.0            1522532.92
1                       0.00                      0.0            1790900.36
2                       0.00                      0.0            1801377.80
3                       0.00                      0.0            2275273.97
4                       0.00                      0.0            1961915.59
..                       …                        …                   …
84                      0.00                      0.0            2199277.55
85                      0.00                      0.0            2664558.24
86                  11435.00                      0.0            3069635.48
87                      0.00                      0.0            2716113.83
88                  11250.05                      0.0            2747074.90

    Member Count  EE Count
0           6146      2690
1           6111      2716
2           6133      2706
3           6113      2699
4           6114      2683
..           …        …
84          6622      3145
85          6612      3092
86          6643      3111
87          6588      3108
88          6618      3093

[89 rows x 15 columns]
```

We'll need to perform some transformations to get the data the way we need:

[68]:
```
claims['PMPM'] = claims['Adjusted Paid Claims'] / claims['Member Count']
claims['PEPM'] = claims['Adjusted Paid Claims'] / claims['EE Count']
claims
```

[68]:
```
    Month  Year  Renewal Year  Domestic Medical Claims  \
0       1  2015             0                414313.73
1       2  2015             0                539505.55
2       3  2015             0                582799.17
3       4  2015             0                894533.91
4       5  2015             0                732374.57
..    …   …           …                      …
84      1  2022             7                501463.78
85      2  2022             7                702513.46
86      3  2022             7                760937.71
87      4  2022             7                579591.91
88      5  2022             7                462605.24
```

	Non-Domestic Medical Claims	Total Hospital Medical Claims
0	324468.71	738782.44
1	307763.94	847269.49
2	272127.91	854927.08
3	253610.74	1148144.65
4	424912.54	1157287.11
..	…	…
84	474951.54	976415.32
85	541335.00	1243848.46
86	588458.73	1349396.44
87	513879.57	1093471.48
88	651661.71	1114266.95

	Non-Hospital Medical Claims	Total Medical Claims	Rx Claims	Rx Rebates
0	385900.53	1124682.97	397849.95	0.00
1	507899.40	1355168.89	503271.97	67540.50
2	539969.58	1394896.66	406481.14	0.00
3	593447.65	1741592.30	533681.67	0.00
4	419625.09	1576912.20	451070.89	66067.50
..	…	…	…	…
84	619224.78	1595640.10	603637.45	0.00
85	1140214.23	2384062.69	650981.28	370485.73
86	943089.50	2292485.94	788584.54	0.00
87	963612.60	2057084.08	659029.75	0.00
88	1016179.46	2130446.41	627878.54	0.00

	Rx Performance Guarantee	Stop Loss Reimbursement	Adjusted Paid Claims
0	0.00	0.0	1522532.92
1	0.00	0.0	1790900.36
2	0.00	0.0	1801377.80
3	0.00	0.0	2275273.97
4	0.00	0.0	1961915.59
..	…	…	…
84	0.00	0.0	2199277.55
85	0.00	0.0	2664558.24
86	11435.00	0.0	3069635.48
87	0.00	0.0	2716113.83
88	11250.05	0.0	2747074.90

	Member Count	EE Count	PMPM	PEPM
0	6146	2690	247.727452	565.997368
1	6111	2716	293.061751	659.388940
2	6133	2706	293.718865	665.697635
3	6113	2699	372.202514	843.006288
4	6114	2683	320.889040	731.239504
..	…	…	…	…

```
84          6622      3145  332.116815  699.293339
85          6612      3092  402.988240  861.758810
86          6643      3111  462.085726  986.703787
87          6588      3108  412.282002  873.910499
88          6618      3093  415.091402  888.158713

[89 rows x 17 columns]
```

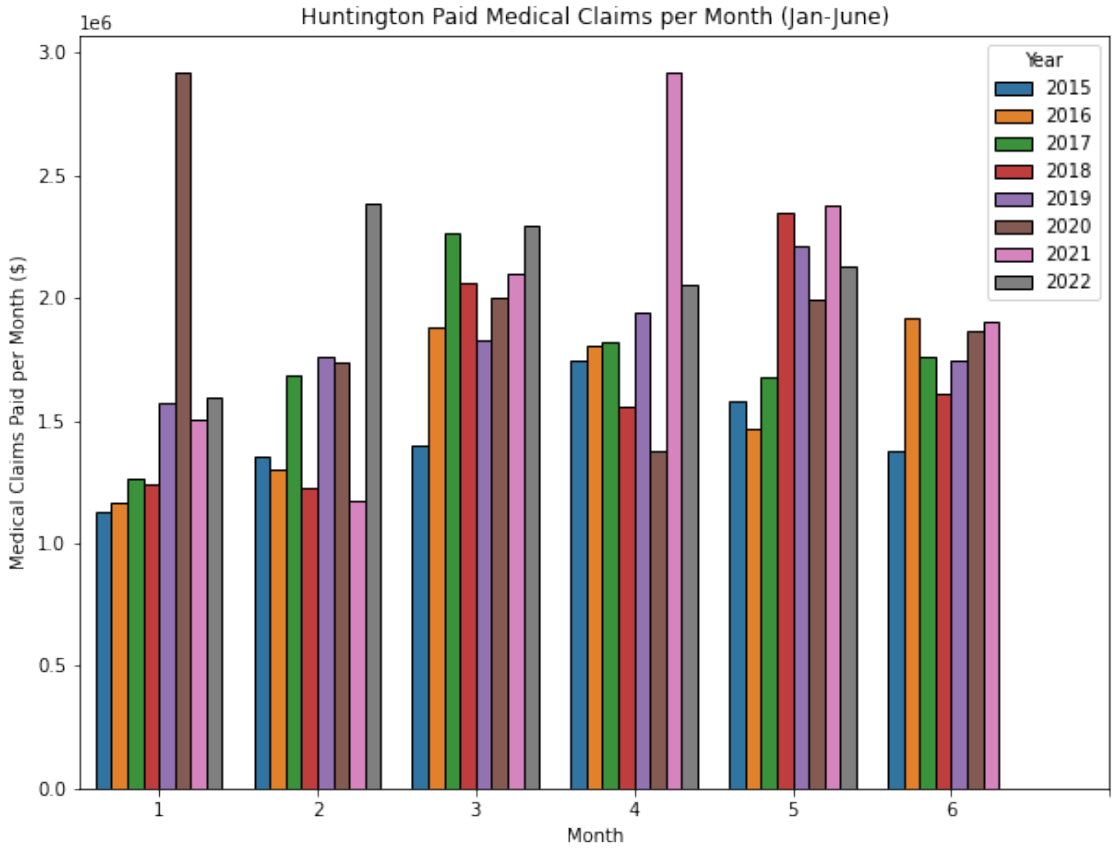
Features: 1. **Month** *(int)* - int value corresponding to the Month (i.e. January - 1, February - 2, March - 3, etc.) 2. **Year** *(int)* - int value designating the Year 3. **Renewal Year** *(int)* - int value designating the Renewal Year (*Note: Renewal Year runs from May of the previous year to June of the next*) 4. **Domestic Medical Claims** *(float)* - floating point value corresponding to the total Medical Claims paid that were incurred within Huntington Medical Group 5. **Non-Domestic Medical Claims** *(float)* - floating point value corresponding to the total Medical Claims paid that were incurred outside Huntington Medical Group 6. **Total Hospital Medical Claims** *(float)* - floating point value corresponding to the sum of **Domestic Medical Claims** and **Non-Domestic Medical Claims** 7. **Non-Hospital Medical Claims** *(float)* - floating point value corresponding to the total Medical Claims paid that were incurred at non-hospital facilities or for non-hospital services 8. **Total Medical Claims** *(float)* - floating point value corresponding to the total Medical Claims paid (i.e. sum of **Total Hospital Medical Claims** and **Non-Hospital Medical Claims**) 9. **Rx Claims** *(float)* - floating point value corresponding to the total Rx Claims paid 10. **Rx Rebates** *(float)* - floating point value corresponding to the total Rx Rebates 11. **Rx Performance Guarantee** *(float)* - floating point value corresponding to the total Rx Performance Guarantee 12. **Stop Loss Reimbursement** *(float)* - floating point value corresponding to the total Stop Loss Reimbursement issued 13. **Adjusted Paid Claims** *(float)* - floating point value corresponding to the total Adjusted Paid Claims (i.e. the difference between **Total Medical Claims** and the sum of **Rx Rebates**, **Rx Performance Guarantee**, and **Stop Loss Reimbusement**) **Note: Remember that the granularity of this dataset is monthly**

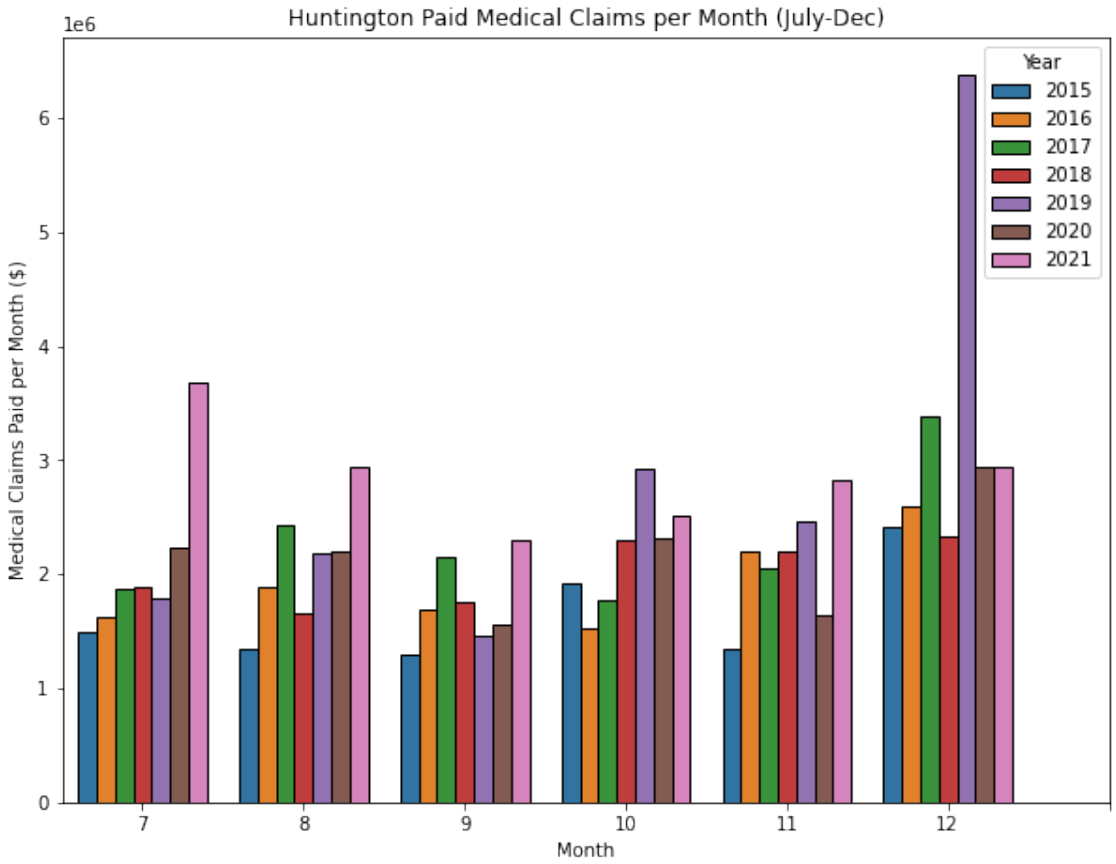
1.2.2 Data Visualization

Let's do a deeper dive into `claims` and see whether our assumption is fair to make

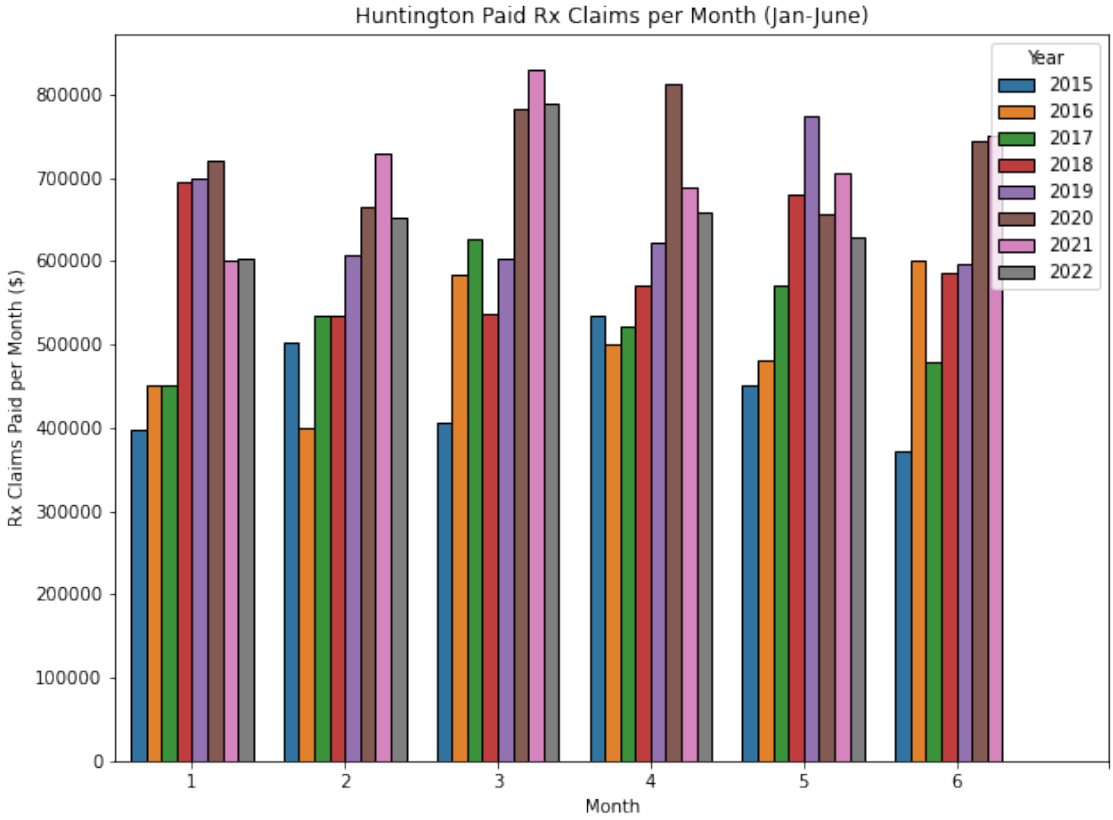
[69]:
```
# Visualizing Total Medical Paid Claims by Month
plt.figure(figsize=(10, 7.5))
sns.barplot(data=claims.query('Month <= 6'), x='Month', y='Total Medical␣
 ↪Claims', hue='Year')
plt.ylabel('Medical Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Huntington Paid Medical Claims per Month (Jan-June)')
plt.show()

plt.figure(figsize=(10, 7.5))
sns.barplot(data=claims.query('Month > 6'), x='Month', y='Total Medical␣
 ↪Claims', hue='Year')
plt.ylabel('Medical Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Huntington Paid Medical Claims per Month (July-Dec)')
```

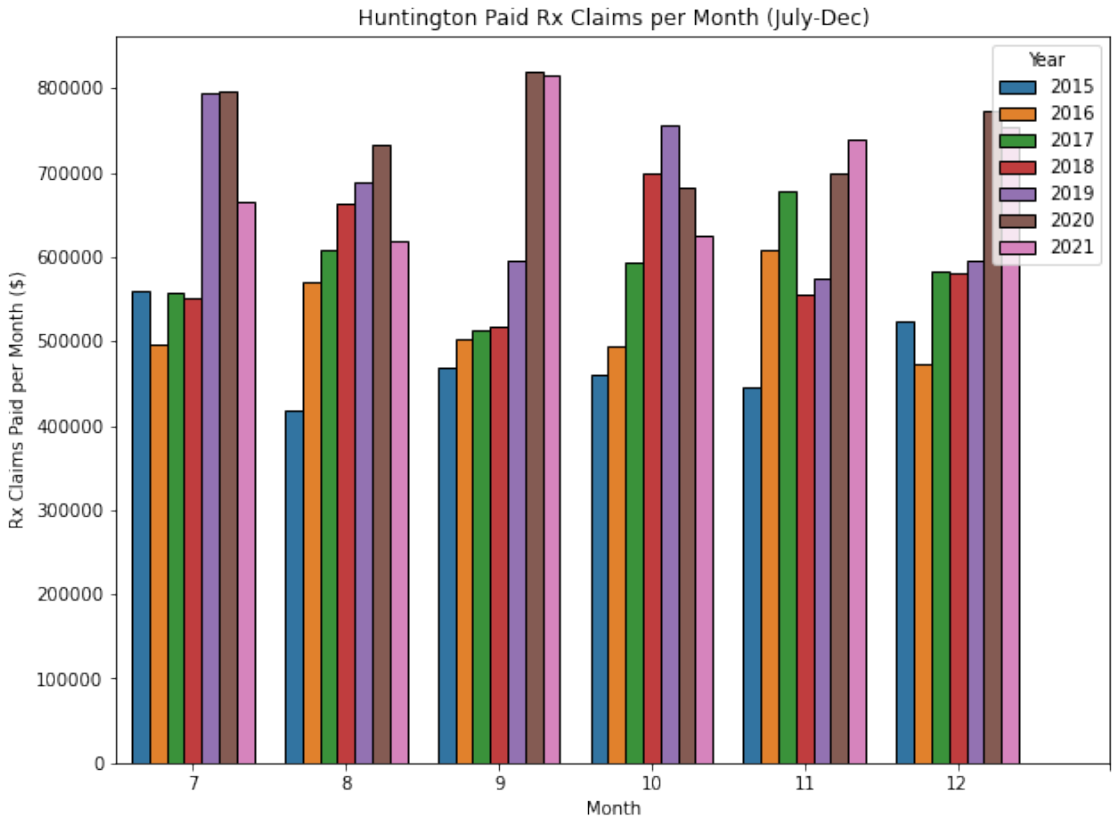
```
plt.show()
```


Huntington Paid Medical Claims per Month (Jan-June)
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Medical Claims Paid per Month ($)
1
2
3
4
5
6
Month
Year
2015
2016
2017
2018
2019
2020
2021
2022

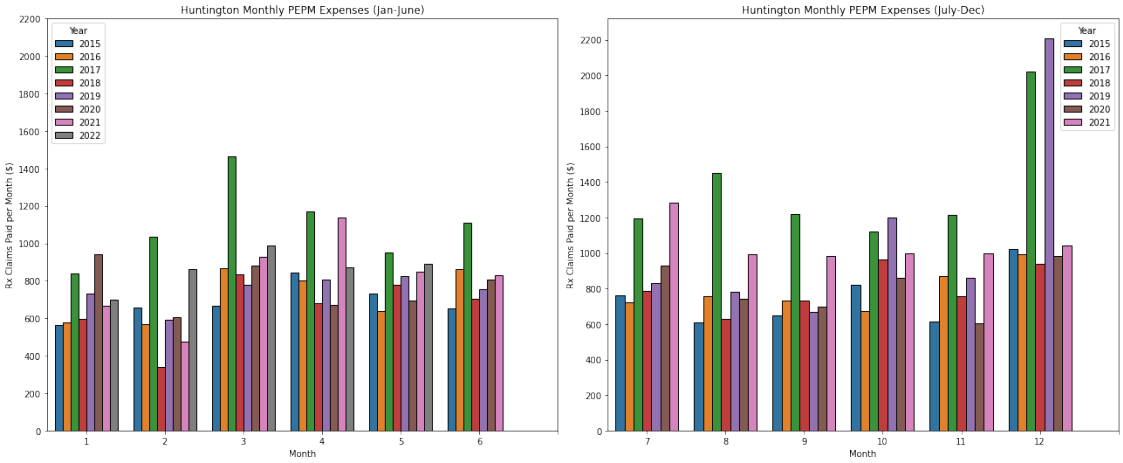


[70]:
```
# Visualizing Total Rx Claims
plt.figure(figsize=(10, 7.5))
sns.barplot(data=claims.query('Month <= 6'), x='Month', y='Rx Claims',␣
 ↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Huntington Paid Rx Claims per Month (Jan-June)')
plt.show()

plt.figure(figsize=(10, 7.5))
sns.barplot(data=claims.query('Month > 6'), x='Month', y='Rx Claims',␣
 ↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Huntington Paid Rx Claims per Month (July-Dec)')
plt.show()
```

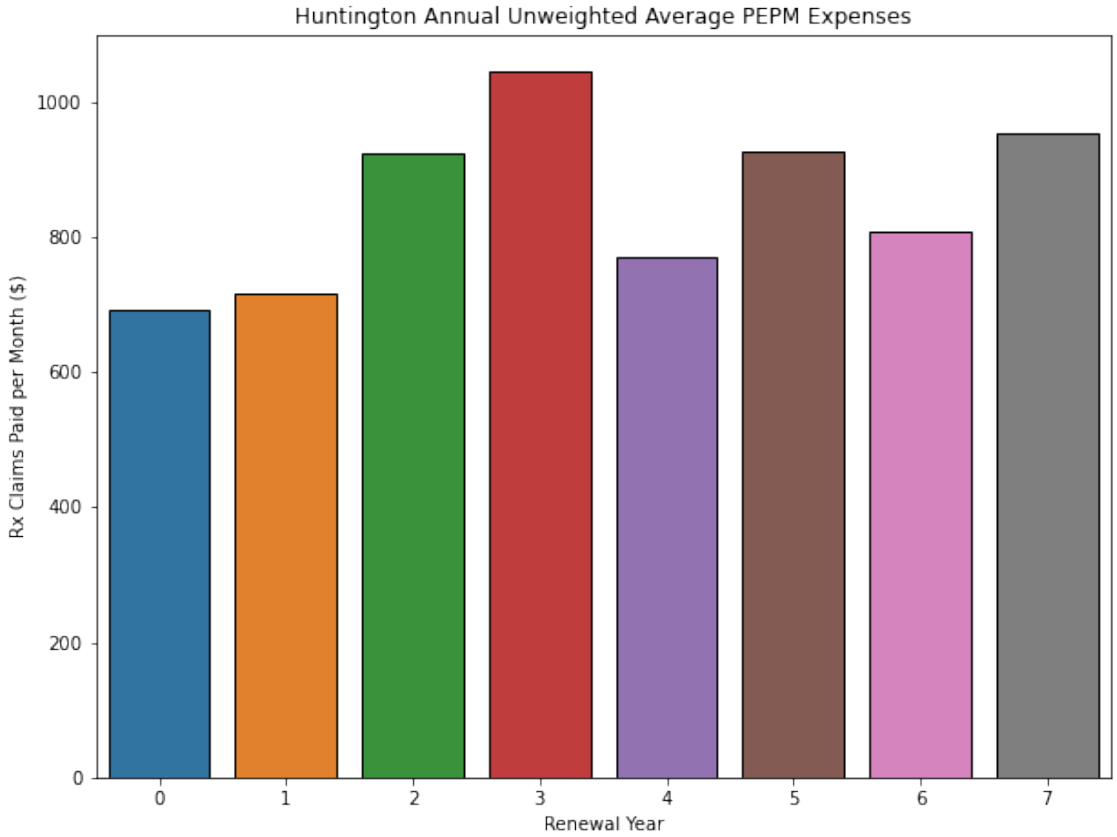

Huntington Paid Rx Claims per Month (Jan-June)
Year
2015
2016
2017
2018
2019
2020
2021
2022
Rx Claims Paid per Month ($)
0
100000
200000
300000
400000
500000
600000
700000
800000
1
2
3
4
5
6
Month



[71]:
```
# Visualizing MONTHLY PMPM expenses
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=claims.query('Month <= 6'), x='Month', y='PEPM', hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.yticks(np.arange(0, 2400, 200))
plt.title('Huntington Monthly PEPM Expenses (Jan-June)')

plt.subplot(1, 2, 2)
sns.barplot(data=claims.query('Month > 6'), x='Month', y='PEPM', hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.yticks(np.arange(0, 2400, 200))
plt.title('Huntington Monthly PEPM Expenses (July-Dec)')

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



[72]:
```
# Visualizing YEARLY PMPM expenses
plt.figure(figsize=(10, 7.5))
sns.barplot(data=claims.groupby('Renewal Year').mean().reset_index(),␣
 ↪x='Renewal Year', y='PEPM')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.title('Huntington Annual Unweighted Average PEPM Expenses')
plt.show()
```



Note: The values in the plot above are unweighted averages as they do not weight each month's PEPM expense by the proportion of Employee Months that are accrued. However, the EE Count remains relatively stable throughout and thus does not actually affect the overall shape of the plot.

Let's breakdown these plots. The first two barplots visualize monthly Medical Claims paid by Year. Looking at these, we see one major takeaway: * There appear to be some major outlying months (most obviously 1/20, 2/22, 4/21, 7/21, 12/19)

While this is interesting, it isn't really all that important since we really should be focusing on PMPM or PEPM values. The second two barplots are nearly identical to the first two except that they visualize **Rx Claims**. Unsurprisingly, we see some similar trends: * Values from more recent years seem abnormally high

Like before, this isn't really all that important. The next three plots are much more important as far as this project is concerned. The next two plots visualize Huntington Monthly PEPM Expenses by Year. Like the first four plots, there are certain months which are clearly outliers (most notably 2/17, 2/18, 3/17, 4/17, 4/18, 4/21, 5/17, 6/17, 7/17, 8/17, 9/17, 11/17, 12/17, and 12/19). Since we'll be wanting to Bootstrap using these values, we'll need to replace these outlier values. Below, we will be replacing these values with the average PEPM value between the PEPM expense for the same month of the previous year and the PEPM expense of the same month for the next year. **Should one of the outliers fall in the first or last year in the data, it will not be**

replaced. For now, we will manually remove the outliers on Excel and reload the dataset back in.

[73]:
```
stripped_claims = pd.read_csv('Huntington Medical Paid Claims Data Outlier␣
↪Removed.csv')
stripped_claims
```

[73]:

	Month	Year	Renewal Year	Domestic Medical Claims	\
0	1	2015	0	414313.73	
1	2	2015	0	539505.55	
2	3	2015	0	582799.17	
3	4	2015	0	894533.91	
4	5	2015	0	732374.57	
..	…	…	…	…	
84	1	2022	7	501463.78	
85	2	2022	7	702513.46	
86	3	2022	7	760937.71	
87	4	2022	7	579591.91	
88	5	2022	7	462605.24	

	Non-Domestic Medical Claims	Total Hospital Medical Claims	\
0	324468.71	738782.44	
1	307763.94	847269.49	
2	272127.91	854927.08	
3	253610.74	1148144.65	
4	424912.54	1157287.11	
..	…	…	
84	474951.54	976415.32	
85	541335.00	1243848.46	
86	588458.73	1349396.44	
87	513879.57	1093471.48	
88	651661.71	1114266.95	

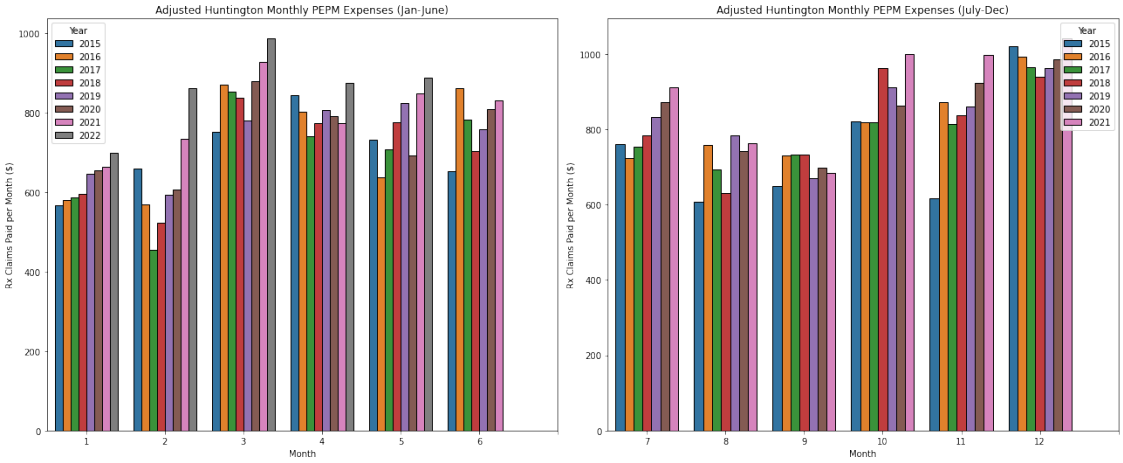
	Non-Hospital Medical Claims	Total Medical Claims	Rx Claims	Rx Rebates	\
0	385900.53	1124682.97	397849.95	0.00	
1	507899.40	1355168.89	503271.97	67540.50	
2	539969.58	1394896.66	406481.14	0.00	
3	593447.65	1741592.30	533681.67	0.00	
4	419625.09	1576912.20	451070.89	66067.50	
..	…	…	…	…	
84	619224.78	1595640.10	603637.45	0.00	
85	1140214.23	2384062.69	650981.28	370485.73	
86	943089.50	2292485.94	788584.54	0.00	
87	963612.60	2057084.08	659029.75	0.00	
88	1016179.46	2130446.41	627878.54	0.00	

	Rx Performance Guarantee	Stop Loss Reimbursement	Adjusted Paid Claims	\
0	0.00	0.0	1522532.92	

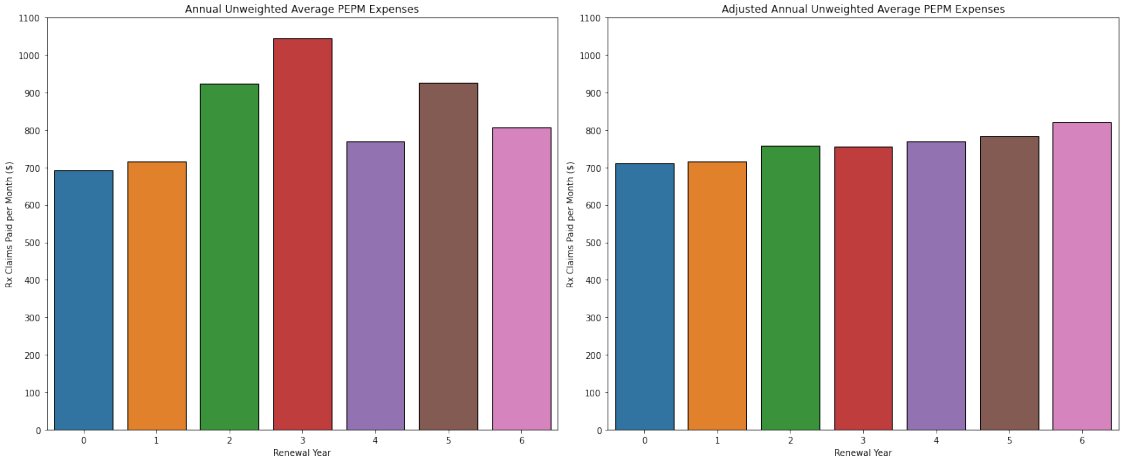
```
1                       0.00                      0.0            1790900.36
2                       0.00                      0.0            1801377.80
3                       0.00                      0.0            2275273.97
4                       0.00                      0.0            1961915.59
..                       …                        …                 …
84                      0.00                      0.0            2199277.55
85                      0.00                      0.0            2664558.24
86                  11435.00                      0.0            3069635.48
87                      0.00                      0.0            2716113.83
88                  11250.05                      0.0            2747074.90

    Member Count  EE Count        PEPM  Unnamed: 16
0           6146      2690  565.997368          NaN
1           6111      2716  659.388940          NaN
2           6133      2706  751.197614          NaN
3           6113      2699  843.006287          NaN
4           6114      2683  731.239504          NaN
..           …         …        …           …
84          6622      3145  699.293339          NaN
85          6612      3092  861.758810          NaN
86          6643      3111  986.703787          NaN
87          6588      3108  873.910499          NaN
88          6618      3093  888.158713          NaN

[89 rows x 17 columns]
```

Now let's recreate the visualizations from before:

[74]:
```
# Visualizing MONTHLY PMPM expenses
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=stripped_claims.query('Month <= 6'), x='Month', y='PEPM',␣
↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Adjusted Huntington Monthly PEPM Expenses (Jan-June)')

plt.subplot(1, 2, 2)
sns.barplot(data=stripped_claims.query('Month > 6'), x='Month', y='PEPM',␣
↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.title('Adjusted Huntington Monthly PEPM Expenses (July-Dec)')

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



[75]:
```
# Side-by-side comparison of actual and adjusted values
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=claims.query('`Renewal Year` != 7').groupby('Renewal Year').
 ↪mean().reset_index(), x='Renewal Year', y='PEPM')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.yticks(np.arange(0, 1200, 100))
plt.title('Annual Unweighted Average PEPM Expenses')

plt.subplot(1, 2, 2)
sns.barplot(data=stripped_claims.query('`Renewal Year` != 7').groupby('Renewal␣
 ↪Year').mean().reset_index(), x='Renewal Year', y='PEPM')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.yticks(np.arange(0, 1200, 100))
plt.title('Adjusted Annual Unweighted Average PEPM Expenses')
plt.tight_layout()
plt.show()
```



Note: The values in the plot above are unweighted averages as they do not weight each month's PEPM expense by the proportion of Employee Months that are accrued. However, the EE Count remains relatively stable throughout and thus does not actually affect the overall shape of the plot.

The plot to the right look much better, we'll go ahead and use this fixed up data (`stripped_claims`) to train our model.

1.3 Model Creation

Now that we've created a more representative dataset, we'll use it to train our model. Before we begin any actual programming, let's lay out a high level overview of the process we'll be following: 1. Generate 100 resamples per month 2. Strip resampled DataFrame to include just Month, Year, Renewal Year, and PEPM

[76]:
```
# Generating resamples
resampled_months = pd.DataFrame(columns=['Month',
                                         'Year'])
for num in np.arange(1, 13):
    for i in np.arange(1000):
        each_month = stripped_claims.query('Month == @num and Year <= 2021')
        month = [num, num, num, num, num, num, num]
        year = [2015, 2016, 2017, 2018, 2019, 2020, 2021]
        df = {'Month': month, 'Year': year}
        resampled = pd.DataFrame(data=df)
        resample = each_month[['EE Count', 'PEPM']].sample(frac=1,␣
 ↪replace=True, axis=0).reset_index().drop('index', axis=1)
        resampled['EE Count'] = resample['EE Count']
        resampled['PEPM'] = resample['PEPM']
        resampled_months = resampled_months.append(resampled)
resampled_months = resampled_months.reset_index().drop('index', axis=1).dropna()

# Further data transformations
def add_renewal_year(month, year):
    '''
    Function that returns a list of the corresponding Renewal Year for␣
 ↪Huntington Medical Group
    -----
    Inputs:

    month (Python list) - Month column in resampled_months DataFrame

    year (Python list) - Year column in resampled_months DataFrame
    -----
    Output:
```

```
    renewal_year (Python list) - Corresponding renewal year column to be added␣
 ↪to the
                                resampled_months DataFrame
    '''
    renewal_year = []
    for i in np.arange(0, len(month)):
        if month[i] <= 5 and year[i] == 2015:
            renewal_year.append(0)
        elif month[i] >= 6 and year[i] == 2015:
            renewal_year.append(1)
        elif month[i] <=5 and year[i] == 2016:
            renewal_year.append(1)
        elif month[i] >= 6 and year[i] == 2016:
            renewal_year.append(2)
        elif month[i] <= 5 and year[i] == 2017:
            renewal_year.append(2)
        elif month[i] >= 6 and year[i] == 2017:
            renewal_year.append(3)
        elif month[i] <= 5 and year[i] == 2018:
            renewal_year.append(3)
        elif month[i] >= 6 and year[i] == 2018:
            renewal_year.append(4)
        elif month[i] <= 5 and year[i] == 2019:
            renewal_year.append(4)
        elif month[i] >= 6 and year[i] == 2019:
            renewal_year.append(5)
        elif month[i] <= 5 and year[i] == 2020:
            renewal_year.append(5)
        elif month[i] >= 6 and year[i] == 2020:
            renewal_year.append(6)
        elif month[i] <= 5 and year[i] == 2021:
            renewal_year.append(6)
        elif month[i] >= 6 and year[i] == 2021:
            renewal_year.append(7)
        elif month[i] <= 5 and year[i] == 2022:
            renewal_year.append(7)
        else:
            renewal_year.append(10)
    return renewal_year
resampled_months['Renewal Year'] = add_renewal_year(resampled_months['Month'].
 ↪values, \
                                                    resampled_months['Year'].
 ↪values)
resampled_months
```

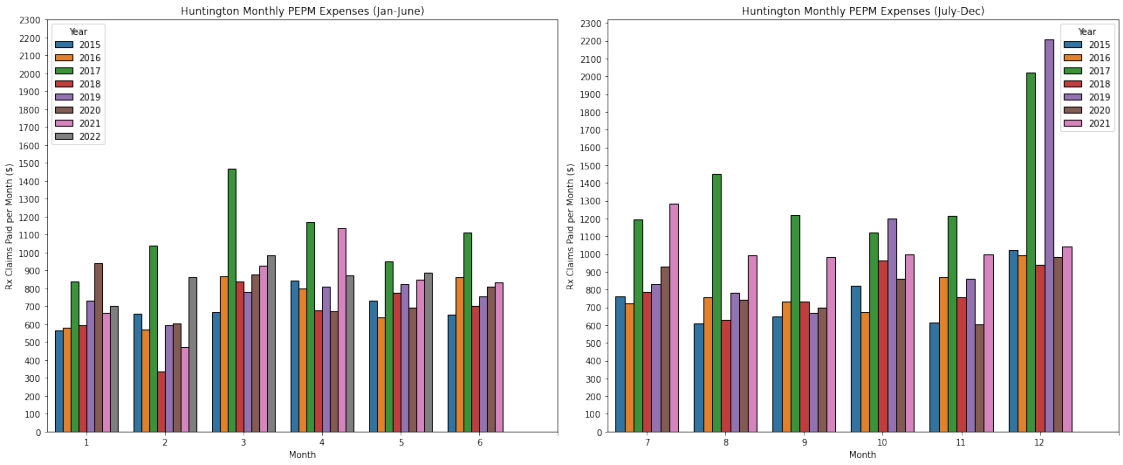
[76]:

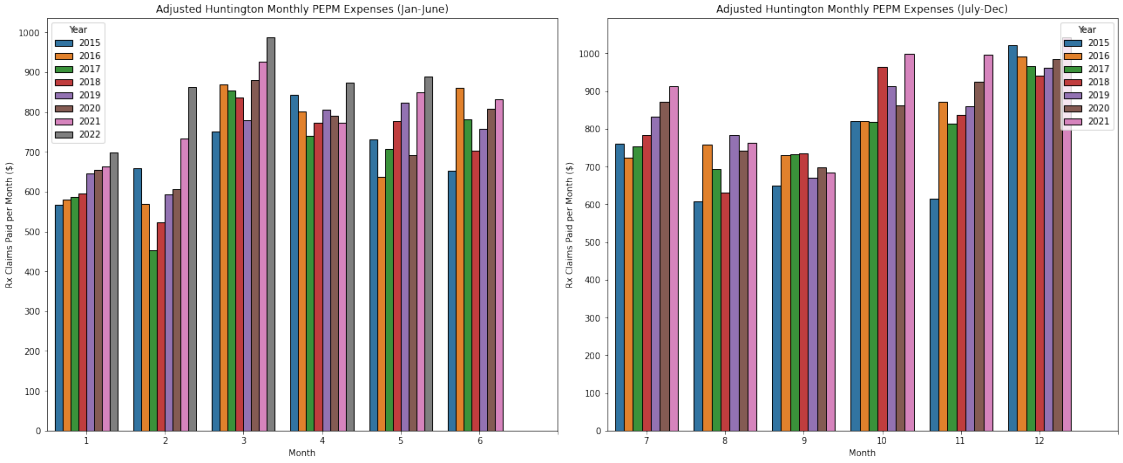
	Month	Year	EE Count	PEPM	Renewal Year
0	1	2015	2781.0	579.693833	0

```
1          1  2016    3164.0   655.335464             1
2          1  2017    2781.0   579.693833             2
3          1  2018    2781.0   579.693833             3
4          1  2019    2781.0   579.693833             4
…      …   …       …           …           …
83995     12  2017    3107.0   940.140283             3
83996     12  2018    3107.0   940.140283             4
83997     12  2019    3001.0   991.919943             5
83998     12  2020    2711.0  1021.661951             6
83999     12  2021    3128.0  1041.966809             7

[84000 rows x 5 columns]
```

[77]:
```
# Visualizing monthly PMPM expenses
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=claims.query('Month <= 6'), x='Month', y='PEPM', hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.yticks(np.arange(0, 2400, 100))
plt.title('Huntington Monthly PEPM Expenses (Jan-June)')

plt.subplot(1, 2, 2)
sns.barplot(data=claims.query('Month > 6'), x='Month', y='PEPM', hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.yticks(np.arange(0, 2400, 100))
plt.title('Huntington Monthly PEPM Expenses (July-Dec)')

plt.tight_layout()
plt.show()

# Plot side-by-side adjusted monthly PEPM
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=stripped_claims.query('Month <= 6'), x='Month', y='PEPM',␣
 ↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
plt.yticks(np.arange(0, 1100, 100))
plt.title('Adjusted Huntington Monthly PEPM Expenses (Jan-June)')

plt.subplot(1, 2, 2)
sns.barplot(data=stripped_claims.query('Month > 6'), x='Month', y='PEPM',␣
 ↪hue='Year')
plt.ylabel('Rx Claims Paid per Month ($)')
plt.xticks(np.arange(0, 7))
```

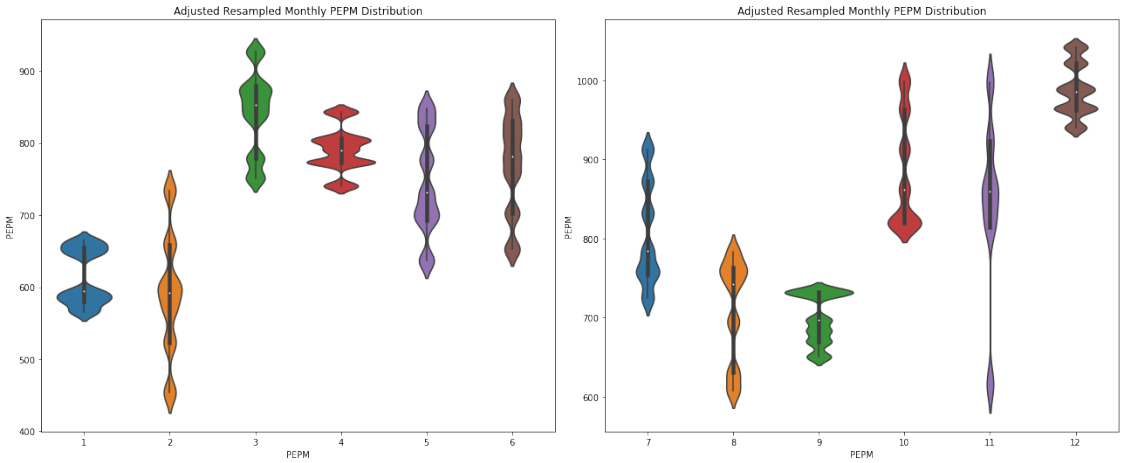
```
plt.yticks(np.arange(0, 1100, 100))
plt.title('Adjusted Huntington Monthly PEPM Expenses (July-Dec)')

plt.tight_layout(pad=1)
plt.show()
```



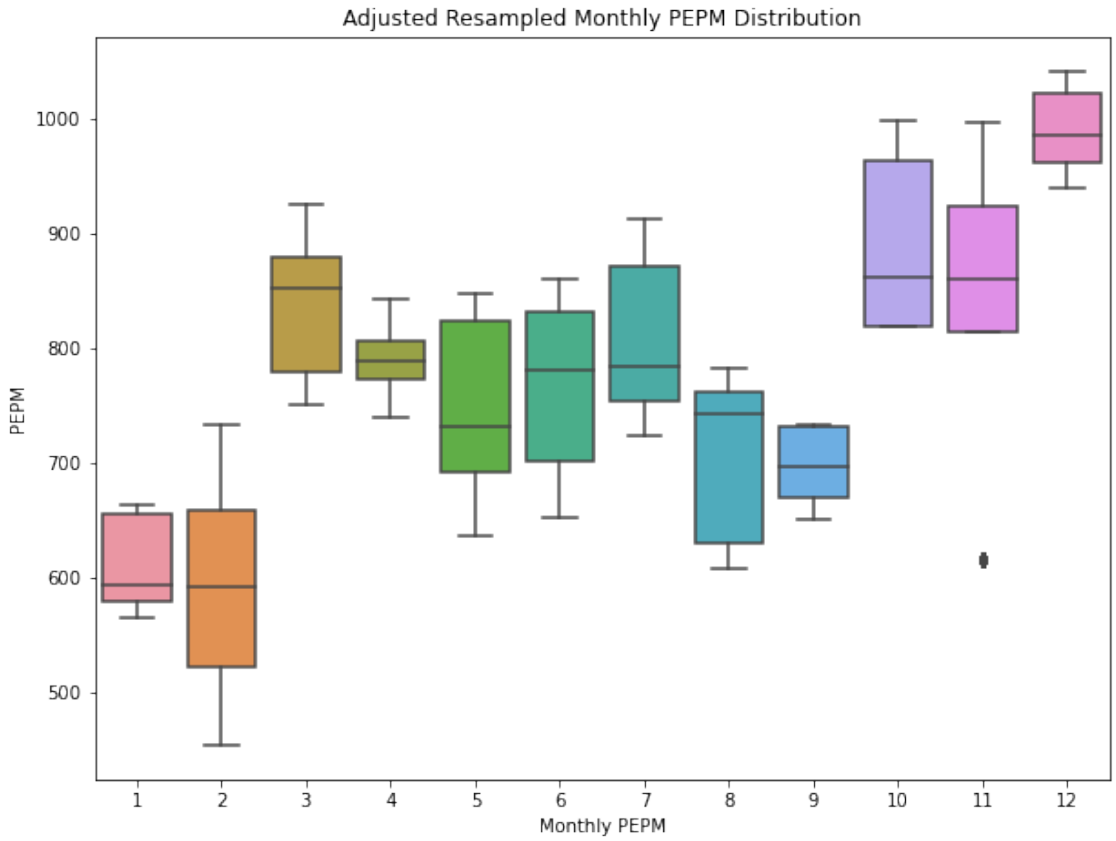


[78]:
```
# Violin plots for January-June
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.violinplot(data=resampled_months.query('Month <= 6'), x='Month', y='PEPM')
plt.xlabel('PEPM')
plt.title('Adjusted Resampled Monthly PEPM Distribution')

# Violin plots for July-December
plt.subplot(1, 2, 2)
```

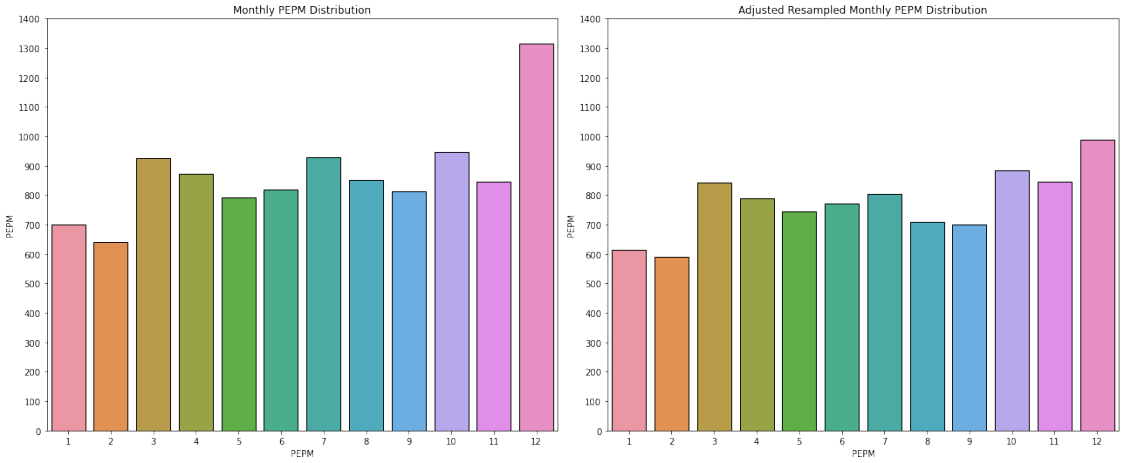
```
sns.violinplot(data=resampled_months.query('Month > 6'), x='Month', y='PEPM')
plt.xlabel('PEPM')
plt.title('Adjusted Resampled Monthly PEPM Distribution')

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



[79]:
```
# Boxplot version of the above
plt.figure(figsize=(10, 7.5))
sns.boxplot(data=resampled_months, x='Month', y='PEPM')
plt.xlabel('Monthly PEPM')
plt.title('Adjusted Resampled Monthly PEPM Distribution')
plt.show()
```



[80]:
```
# Side-by-side mean monthly PEPM
plt.figure(figsize=(18, 7.5))
plt.subplot(1, 2, 1)
sns.barplot(data=claims.groupby('Month').mean().reset_index(), x='Month',␣
↪y='PEPM')
plt.xlabel('PEPM')
plt.yticks(np.arange(0, 1500, 100))
plt.title('Monthly PEPM Distribution')

# Violin plots for July-December
plt.subplot(1, 2, 2)
sns.barplot(data=resampled_months.groupby('Month').mean().reset_index(),␣
↪x='Month', y='PEPM')
plt.xlabel('PEPM')
plt.yticks(np.arange(0, 1500, 100))
plt.title('Adjusted Resampled Monthly PEPM Distribution')

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



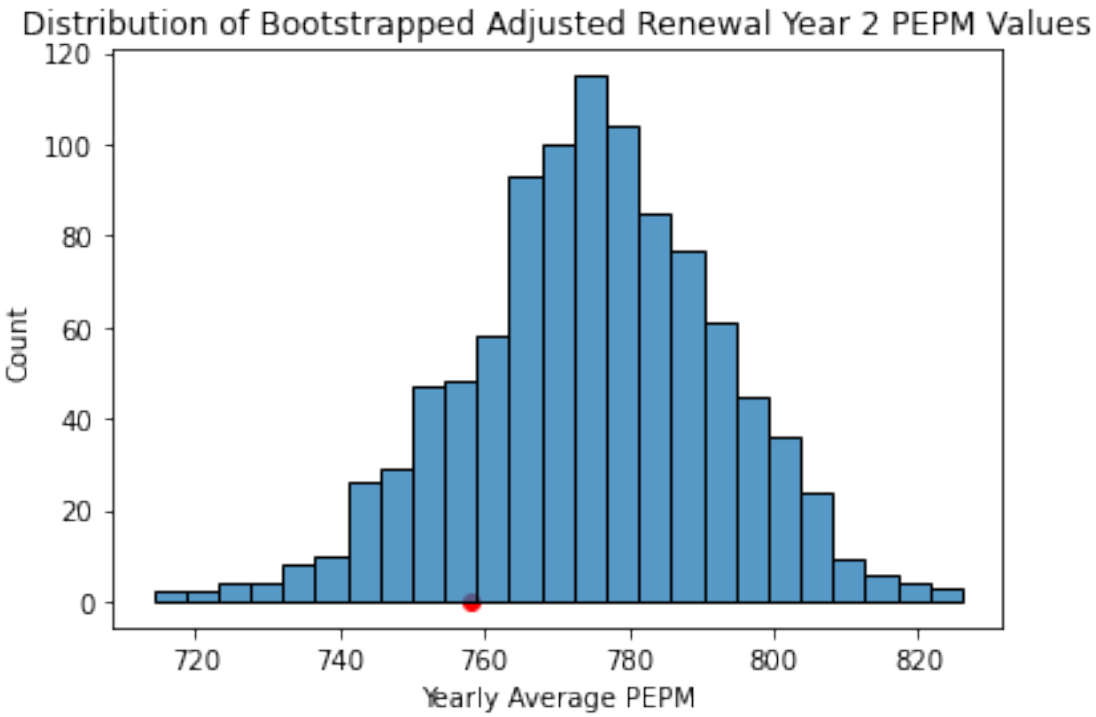
[81]:
```
def resampled_yearly_pepm(renewal_year, n):
    '''
    Extracts the average PEPM value from the resampled_months DataFrame
    for the specified year
    -----
    Input:

    renewal_year (int) - Int value specifying the renewal year to extract

    n (int) - Number of resamples
    -----
    Output:

    pepm_ryr (NumPy Array) - NumPy Array of length n (where n is the number of␣
↪resamples)
                             with the yearly mean PEPM values
    '''
    ryr = resampled_months.sort_values(['Year', 'Month']) \
                          .query('`Renewal Year` == @renewal_year')
    pepm_arr_ryr = ryr['PEPM'].values
    ee_arr_ryr = ryr['EE Count'].values
    pepm_arr_ryr_split = np.split(pepm_arr_ryr, 12)
    ee_arr_ryr_split = np.split(ee_arr_ryr, 12)
    ee_sums = [sum(item) for item in ee_arr_ryr_split]
    ee_sums = np.zeros(n)
    for i in range(len(ee_arr_ryr_split)):
        ee_sums += ee_arr_ryr_split[i]
    ee_props = [ee_arr_ryr_split[i] / ee_sums for i in␣
↪range(len(ee_arr_ryr_split))]
    weighted_totals = [np.multiply(ee_props[i], pepm_arr_ryr_split[i]) for i in␣
↪range(len(ee_props))]
```

```
    pepm_ryr = np.zeros(n)
    for i in range(len(weighted_totals)):
        pepm_ryr += weighted_totals[i]
    return pepm_ryr
```

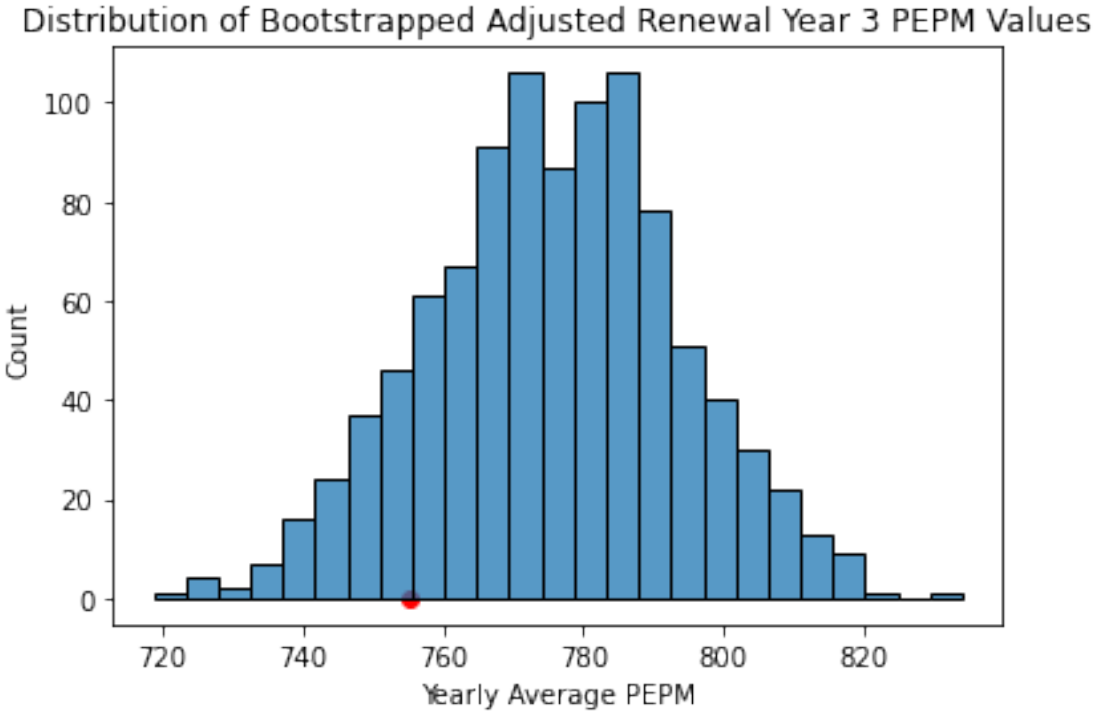
[82]:
```
pepm_ryr2 = resampled_yearly_pepm(2, 1000)
pepm_ryr3 = resampled_yearly_pepm(3, 1000)
pepm_ryr4 = resampled_yearly_pepm(4, 1000)
pepm_ryr5 = resampled_yearly_pepm(5, 1000)
```

[83]:
```
pepms = [pepm_ryr2, pepm_ryr3, pepm_ryr4, pepm_ryr5]
observed_yearly_pepms = stripped_claims.query('`Renewal Year` > 1 and `Renewal␣
↪Year` < 6') \
                                        .groupby('Renewal Year').mean()['PEPM'].
↪values
i = 0
start = 2
for pepm in pepms:
    observed_year_pepm = observed_yearly_pepms[i]
    sns.histplot(data=pepm)
    plt.xlabel('Yearly Average PEPM')
    plt.title(f'Distribution of Bootstrapped Adjusted Renewal Year {start} PEPM␣
↪Values')
    plt.scatter(observed_year_pepm, -0.01, color='red')
    print(f'p-value: {min(sum(pepms[i] >= observed_year_pepm) / len(pepms[i]),␣
↪sum(pepms[i] <= observed_year_pepm) / len(pepms[i]))}')
    plt.show()
    i += 1
    start += 1
```

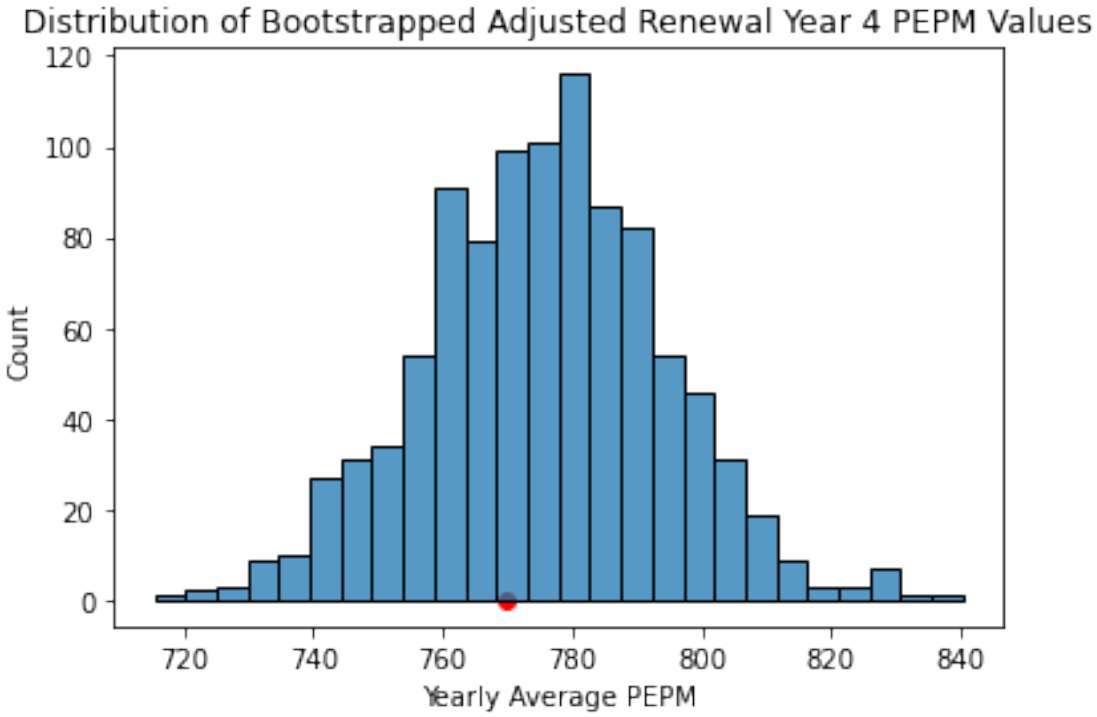
```
p-value: 0.172
```


Distribution of Bootstrapped Adjusted Renewal Year 2 PEPM Values
Count
Yearly Average PEPM
0
20
40
60
80
100
120
720
740
760
780
800
820

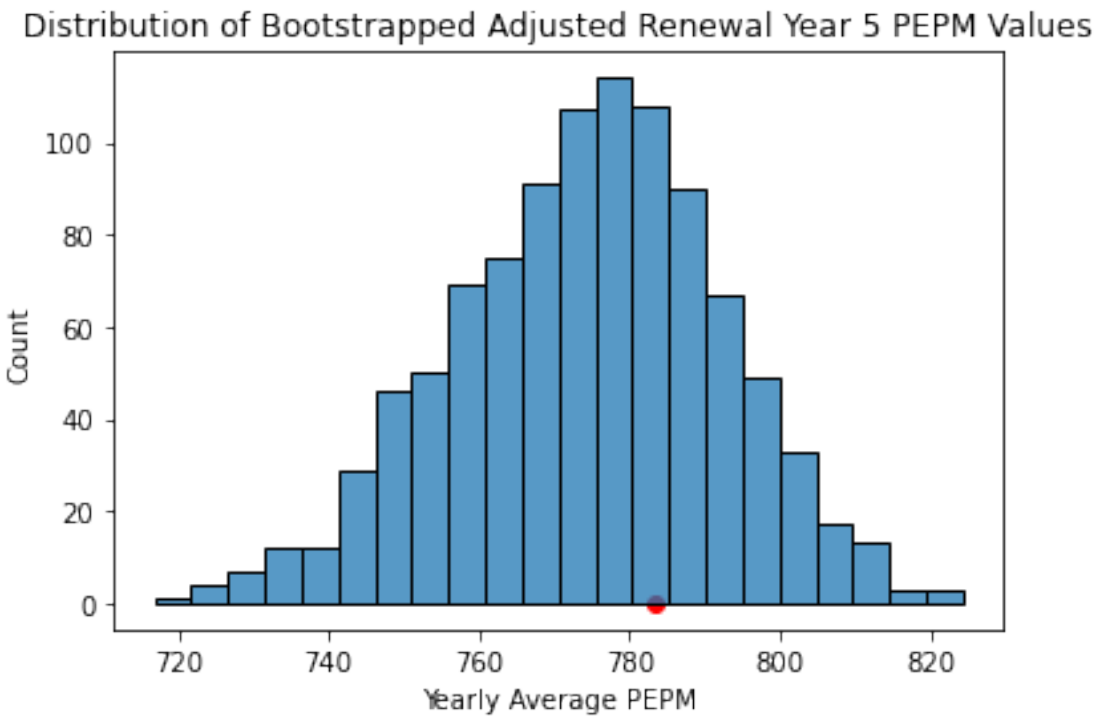
p-value: 0.131


Distribution of Bootstrapped Adjusted Renewal Year 3 PEPM Values
Count
Yearly Average PEPM
0
20
40
60
80
100
720
740
760
780
800
820

p-value: 0.382



p-value: 0.321



Looking at the plots above and their corresponding p-values, we see that the bootsrapping process has both built an approximately normal distribution of PEPM values for each Renewal Year (Years 2-5 in our case) and has approximately rebuilt the population. We can say this because the observed averages are well within the distribution (i.e. the chance of mistakenly characterizing the observed value as statistically sigificant is high). Now we'll need to perform a series of data transformations to form the dataset we'll be using to train our model.

[84]:
```
resampled = resampled_months.query('`Renewal Year` > 1 and `Renewal Year` < 6')
resampled
```

[84]:

	Month	Year	EE Count	PEPM	Renewal Year
2	1	2017	2781.0	579.693833	2
3	1	2018	2781.0	579.693833	3
4	1	2019	2781.0	579.693833	4
5	1	2020	3108.0	646.340514	5
9	1	2017	3123.0	594.689216	2
…	…	…	…	…	…
83990	12	2019	2711.0	1021.661951	5
83994	12	2016	3001.0	991.919943	2
83995	12	2017	3107.0	940.140283	3
83996	12	2018	3107.0	940.140283	4
83997	12	2019	3001.0	991.919943	5

[48000 rows x 5 columns]

[85]:
```
actual = resampled_months.query('`Renewal Year` == 5').sort_values(['Year',␣
↪'Month'])
pepm_arr_ryr = actual['PEPM'].values
ee_arr_ryr = actual['EE Count'].values
pepm_arr_ryr_split = np.split(pepm_arr_ryr, 12)
ee_arr_ryr_split = np.split(ee_arr_ryr, 12)
ee_sums = [sum(item) for item in ee_arr_ryr_split]
ee_sums = np.zeros(1000)
for i in range(len(ee_arr_ryr_split)):
    ee_sums += ee_arr_ryr_split[i]
ee_props = [ee_arr_ryr_split[i] / ee_sums for i in range(len(ee_arr_ryr_split))]
weighted_totals = [np.multiply(ee_props[i], pepm_arr_ryr_split[i]) for i in␣
↪range(len(ee_props))]
actual_pepm = np.zeros(1000)
for i in range(len(weighted_totals)):
    actual_pepm += weighted_totals[i]
```

[86]:
```
renewal_yr1 = np.split(resampled_months.query('`Renewal Year` == 2').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr2 = np.split(resampled_months.query('`Renewal Year` == 3').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr3 = np.split(resampled_months.query('`Renewal Year` == 4').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
design_matrix = pd.DataFrame(data={'Month 1': renewal_yr1[0], 'Month 2':␣
↪renewal_yr1[1],
                                   'Month 3': renewal_yr1[2], 'Month 4':␣
↪renewal_yr1[3],
                                   'Month 5': renewal_yr1[4], 'Month 6':␣
↪renewal_yr1[5],
                                   'Month 7': renewal_yr1[6], 'Month 8':␣
↪renewal_yr1[7],
                                   'Month 8': renewal_yr1[8], 'Month 9':␣
↪renewal_yr1[9],
                                   'Month 10': renewal_yr1[10], 'Month 11':␣
↪renewal_yr1[11],
                                   'Month 12': renewal_yr2[0], 'Month 13':␣
↪renewal_yr2[1],
                                   'Month 14': renewal_yr2[2], 'Month 15':␣
↪renewal_yr2[3],
                                   'Month 16': renewal_yr2[4], 'Month 17':␣
↪renewal_yr2[5],
                                   'Month 18': renewal_yr2[6], 'Month 19':␣
↪renewal_yr2[7],
                                   'Month 20': renewal_yr2[8], 'Month 21':␣
↪renewal_yr2[9],
```

```
                                    'Month 22': renewal_yr2[10], 'Month 23':
↪renewal_yr2[11],
                                    'Month 24': renewal_yr3[0], 'Month 25':␣
↪renewal_yr3[1],
                                    'Month 26': renewal_yr3[2], 'Month 27':␣
↪renewal_yr3[3],
                                    'Month 28': renewal_yr3[4], 'Month 29':␣
↪renewal_yr3[5],
                                    'Month 30': renewal_yr3[6], 'Month 31':␣
↪renewal_yr3[7],
                                    'Month 32': renewal_yr3[8], 'Month 33':␣
↪renewal_yr3[9],
                                    'Month 34': renewal_yr3[10], 'Month 35':␣
↪renewal_yr3[11],
                                    'Following Year Average PEPM': actual_pepm})
design_matrix
```

[86]:

	Month 1	Month 2	Month 3	Month 4	Month 5	Month 6	\
0	860.513329	872.348275	742.539715	697.144312	963.136584	860.013687	
1	757.194834	760.263674	630.378131	734.055902	963.136584	860.013687	
2	757.194834	912.692280	758.813987	669.447702	912.600062	923.611905	
3	860.513329	832.004270	742.539715	669.447702	999.040080	923.611905	
4	808.097704	912.692280	742.539715	731.902108	963.136584	813.685789	
..	…	…	…	…	…	…	
995	860.513329	872.348275	758.813987	734.055902	963.136584	813.685789	
996	757.194834	724.447765	630.378131	650.450107	819.746524	923.611905	
997	808.097704	832.004270	758.813987	669.447702	862.063540	997.168487	
998	652.563948	832.004270	694.596059	669.447702	963.136584	923.611905	
999	652.563948	760.263674	694.596059	669.447702	820.651995	836.849738	

	Month 7	Month 8	Month 9	Month 10	…	Month 27	\
0	1021.661951	592.667177	778.846267	801.607736	…	650.450107	
1	966.030113	592.667177	869.190770	773.141579	…	669.447702	
2	985.160269	453.848858	926.147536	789.862151	…	669.447702	
3	940.140283	453.848858	836.322759	773.495752	…	697.144312	
4	940.140283	592.667177	836.322759	773.141579	…	731.902108	
..	…	…	…	…	…	…	
995	966.030113	734.022439	751.197614	773.495752	…	734.055902	
996	1021.661951	592.667177	751.197614	740.408780	…	697.144312	
997	991.919943	659.388940	836.322759	789.862151	…	734.055902	
998	991.919943	453.848858	879.691395	801.607736	…	734.055902	
999	991.919943	523.258018	836.322759	740.408780	…	729.748314	

	Month 28	Month 29	Month 30	Month 31	Month 32	Month 33	\
0	819.746524	836.849738	991.919943	579.693833	569.691433	751.197614	
1	819.746524	860.013687	1021.661951	594.689216	569.691433	836.322759	

```
2    912.600062  860.013687  1041.966809  664.330413  659.388940  852.756765
3    912.600062  871.129612   940.140283  594.689216  592.667177  751.197614
4    818.841053  860.013687   966.030113  646.340514  453.848858  852.756765
..          …         …           …          …          …          …
995  862.063540  836.849738  1041.966809  594.689216  453.848858  852.756765
996  818.841053  813.685789   962.650276  655.335464  659.388940  869.190770
997  818.841053  615.914246  1041.966809  594.689216  453.848858  778.846267
998  999.040080  871.129612   985.160269  587.191524  453.848858  879.691395
999  862.063540  813.685789   940.140283  646.340514  659.388940  778.846267

       Month 34    Month 35  Following Year Average PEPM
0    806.582724  731.239504                   774.125438
1    740.408780  824.151838                   778.463429
2    801.607736  824.151838                   781.029321
3    789.862151  824.151838                   768.228209
4    806.582724  848.402127                   782.133768
..          …         …                          …
995  773.141579  731.239504                   753.567035
996  740.408780  706.539316                   794.162060
997  773.141579  776.423408                   771.294009
998  789.862151  776.423408                   768.473101
999  801.607736  776.423408                   793.967330

[1000 rows x 36 columns]
```

1.4 Training the Model

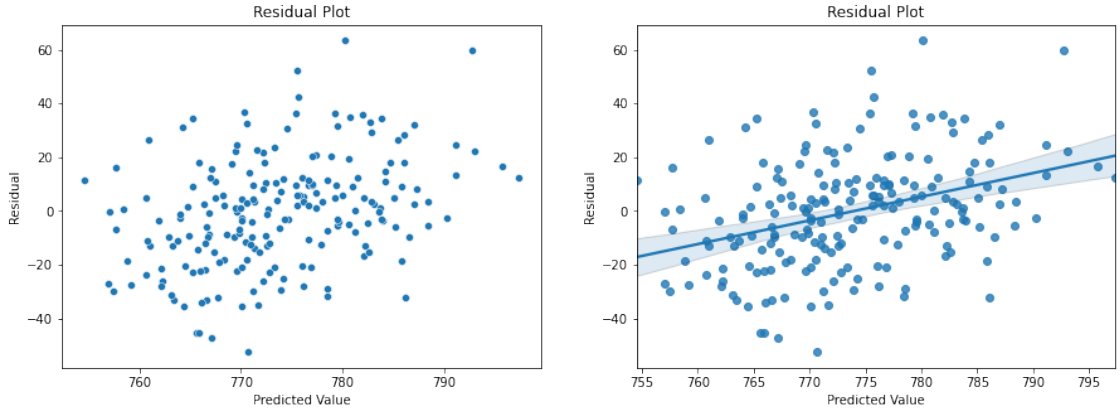
Now that our design matrix has been created we'll import in `sklearn` to be able to create and train a Linear Regression Model to predict average PEPM cost for the year based on the past 36 months of PEPM values.

[109]:
```
# Model Selection
from sklearn import model_selection, linear_model

# Split into train and test sets
X = design_matrix.drop('Following Year Average PEPM', axis=1)
Y = design_matrix[['Following Year Average PEPM']]
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,␣
 ↪test_size=0.2)

# Train the model
kf = model_selection.KFold(n_splits=5)
kf_lr_models = []
validation_errors = []

for train_idx, valid_idx in kf.split(X_train):
    # Split the data
```

```
    split_X_train = X_train.iloc[train_idx].iloc[:, 4:]
    split_X_valid = X_train.iloc[valid_idx].iloc[:, 4:]
    split_Y_train = Y_train.iloc[train_idx]
    split_Y_valid = Y_train.iloc[valid_idx]

    # Create and fit a Logistic Regression model on the training split
    model = linear_model.LinearRegression(fit_intercept=False)
    model.fit(split_X_train, split_Y_train)

    # Compute the Mean Cross Entropy Loss on the validation split
    thetas = model.coef_.flatten()
    mse = np.mean((model.predict(split_X_train) - split_Y_train.to_numpy()) **␣
 ↪2)

    kf_lr_models.append(model)
    validation_errors.append(mse)

lowest_validation_error = min(validation_errors)
index = 0
for error in validation_errors:
    if validation_errors[index] == lowest_validation_error:
        break
    else:
        index += 1
optimal_model = kf_lr_models[index]

# Model validation
optimal_model.fit(X_train, Y_train)
validation_predictions = optimal_model.predict(X_train)
test_predictions = optimal_model.predict(X_test)

model_validation_error = (np.mean((validation_predictions - Y_train) ** 2)) **␣
 ↪0.5
model_test_error = (np.mean((test_predictions - Y_test) ** 2)) ** 0.5
print(f'Validation MSE: {model_validation_error}')
print(f'Test MSE: {model_test_error}')
print(min(test_predictions.flatten()))
print(max(test_predictions.flatten()))

# Plotting the residuals
residuals = test_predictions.flatten() - Y_test.values.flatten()
residuals_df = pd.DataFrame(data={'Predicted Value': test_predictions.
 ↪flatten(), 'Residual': residuals})

plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(data=residuals_df, x='Predicted Value', y='Residual')
```

```
plt.title('Residual Plot')

plt.subplot(1, 2, 2)
sns.regplot(data=residuals_df, x='Predicted Value', y='Residual')
plt.title('Residual Plot')

plt.show()
print(f'Correlation: {residuals_df.corr()["Residual"].values[0]}')
```

```
Validation MSE: Following Year Average PEPM    19.339027
dtype: float64
Test MSE: Following Year Average PEPM    20.30204
dtype: float64
754.5384189904945
797.3741846865327
```




```
Correlation: 0.3666427895031009
```

[62]:
```
pd.DataFrame(data={'Test Predictions': test_predictions.flatten(), 'Actual␣
↪Values': Y_test.values.flatten()})
```

[62]:

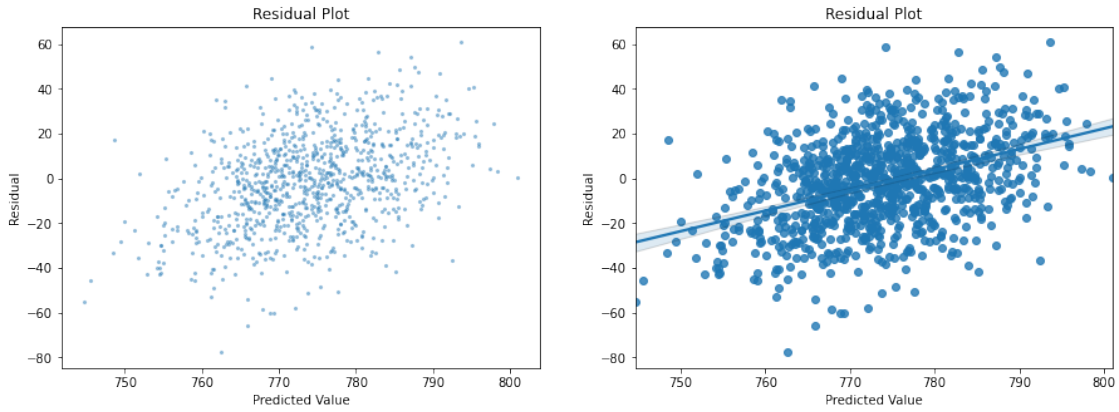
	Test Predictions	Actual Values
0	775.404471	769.784557
1	780.916314	757.775338
2	774.460958	800.384476
3	782.522118	782.090801
4	786.823199	751.132794
..	…	…
195	776.759810	817.090949
196	785.179420	775.404272
197	780.382125	767.667723
198	784.802541	763.605551
199	775.426920	773.163074

```
[200 rows x 2 columns]
```

[110]:
```
# Years 1, 2, and 3 predicting 4
pepm_arr_ryr = resampled_months.query('`Renewal Year` == 4')['PEPM'].values
ee_arr_ryr = resampled_months.query('`Renewal Year` == 4')['EE Count'].values
pepm_arr_ryr_split = np.split(pepm_arr_ryr, 12)
ee_arr_ryr_split = np.split(ee_arr_ryr, 12)
ee_sums = [sum(item) for item in ee_arr_ryr_split]
ee_sums = np.zeros(1000)
for i in range(len(ee_arr_ryr_split)):
    ee_sums += ee_arr_ryr_split[i]
ee_props = [ee_arr_ryr_split[i] / ee_sums for i in range(len(ee_arr_ryr_split))]
weighted_totals = [np.multiply(ee_props[i], pepm_arr_ryr_split[i]) for i in␣
 ↪range(len(ee_props))]
actual_pepm = np.zeros(1000)
for i in range(len(weighted_totals)):
    actual_pepm += weighted_totals[i]
Y_test2 = actual_pepm

renewal_yr1 = np.split(resampled_months.query('`Renewal Year` == 1').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr2 = np.split(resampled_months.query('`Renewal Year` == 2').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr3 = np.split(resampled_months.query('`Renewal Year` == 3').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
design_matrix2 = pd.DataFrame(data={'Month 1': renewal_yr1[0], 'Month 2':␣
 ↪renewal_yr1[1],
                                    'Month 3': renewal_yr1[2], 'Month 4':␣
 ↪renewal_yr1[3],
                                    'Month 5': renewal_yr1[4], 'Month 6':␣
 ↪renewal_yr1[5],
                                    'Month 7': renewal_yr1[6], 'Month 8':␣
 ↪renewal_yr1[7],
                                    'Month 8': renewal_yr1[8], 'Month 9':␣
 ↪renewal_yr1[9],
                                    'Month 10': renewal_yr1[10], 'Month 11':␣
 ↪renewal_yr1[11],
                                    'Month 12': renewal_yr2[0], 'Month 13':␣
 ↪renewal_yr2[1],
                                    'Month 14': renewal_yr2[2], 'Month 15':␣
 ↪renewal_yr2[3],
                                    'Month 16': renewal_yr2[4], 'Month 17':␣
 ↪renewal_yr2[5],
                                    'Month 18': renewal_yr2[6], 'Month 19':␣
 ↪renewal_yr2[7],
```

```
                                    'Month 20': renewal_yr2[8], 'Month 21':␣
 ↪renewal_yr2[9],
                                    'Month 22': renewal_yr2[10], 'Month 23':
 ↪renewal_yr2[11],
                                    'Month 24': renewal_yr3[0], 'Month 25':␣
 ↪renewal_yr3[1],
                                    'Month 26': renewal_yr3[2], 'Month 27':␣
 ↪renewal_yr3[3],
                                    'Month 28': renewal_yr3[4], 'Month 29':␣
 ↪renewal_yr3[5],
                                    'Month 30': renewal_yr3[6], 'Month 31':␣
 ↪renewal_yr3[7],
                                    'Month 32': renewal_yr3[8], 'Month 33':␣
 ↪renewal_yr3[9],
                                    'Month 34': renewal_yr3[10], 'Month 35':␣
 ↪renewal_yr3[11]})

test_predictions = optimal_model.predict(design_matrix2)
model_test_error = (np.mean((test_predictions - Y_test2) ** 2)) ** 0.5
print(f'Test RMSE: {model_test_error}')
print(f'Min: {min(test_predictions.flatten())}')
print(f'Max: {max(test_predictions.flatten())}')

residuals = test_predictions.flatten() - Y_test2.flatten()
residuals_df = pd.DataFrame(data={'Predicted Value': test_predictions.
 ↪flatten(), 'Residual': residuals})

plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(data=residuals_df, x='Predicted Value', y='Residual', s=10,␣
 ↪alpha=0.5)
plt.title('Residual Plot')

plt.subplot(1, 2, 2)
sns.regplot(data=residuals_df, x='Predicted Value', y='Residual')
plt.title('Residual Plot')

plt.show()
print(f'Correlation: {residuals_df.corr()["Residual"].values[0]}')
```

```
Test RMSE: 20.674039952735797
Min: 744.6434396722489
Max: 801.0820296005013
```



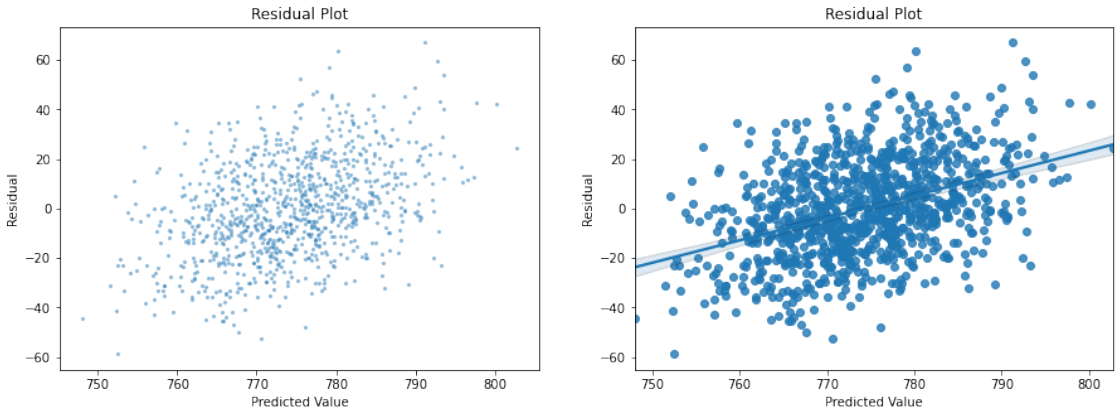

```
Correlation: 0.40940781178201735
```

[113]:
```
# Years 2, 3, and 4 predicting 5
pepm_arr_ryr = resampled_months.query('`Renewal Year` == 5')['PEPM'].values
ee_arr_ryr = resampled_months.query('`Renewal Year` == 5')['EE Count'].values
pepm_arr_ryr_split = np.split(pepm_arr_ryr, 12)
ee_arr_ryr_split = np.split(ee_arr_ryr, 12)
ee_sums = [sum(item) for item in ee_arr_ryr_split]
ee_sums = np.zeros(1000)
for i in range(len(ee_arr_ryr_split)):
    ee_sums += ee_arr_ryr_split[i]
ee_props = [ee_arr_ryr_split[i] / ee_sums for i in range(len(ee_arr_ryr_split))]
weighted_totals = [np.multiply(ee_props[i], pepm_arr_ryr_split[i]) for i in␣
 ↪range(len(ee_props))]
actual_pepm = np.zeros(1000)
for i in range(len(weighted_totals)):
    actual_pepm += weighted_totals[i]
Y_test2 = actual_pepm

renewal_yr1 = np.split(resampled_months.query('`Renewal Year` == 2').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr2 = np.split(resampled_months.query('`Renewal Year` == 3').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr3 = np.split(resampled_months.query('`Renewal Year` == 4').
 ↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
design_matrix2 = pd.DataFrame(data={'Month 1': renewal_yr1[0], 'Month 2':␣
 ↪renewal_yr1[1],
                                    'Month 3': renewal_yr1[2], 'Month 4':␣
 ↪renewal_yr1[3],
                                    'Month 5': renewal_yr1[4], 'Month 6':␣
 ↪renewal_yr1[5],
```

```
                                    'Month 7': renewal_yr1[6], 'Month 8':␣
↪renewal_yr1[7],
                                    'Month 8': renewal_yr1[8], 'Month 9':␣
↪renewal_yr1[9],
                                    'Month 10': renewal_yr1[10], 'Month 11':␣
↪renewal_yr1[11],
                                    'Month 12': renewal_yr2[0], 'Month 13':␣
↪renewal_yr2[1],
                                    'Month 14': renewal_yr2[2], 'Month 15':␣
↪renewal_yr2[3],
                                    'Month 16': renewal_yr2[4], 'Month 17':␣
↪renewal_yr2[5],
                                    'Month 18': renewal_yr2[6], 'Month 19':␣
↪renewal_yr2[7],
                                    'Month 20': renewal_yr2[8], 'Month 21':␣
↪renewal_yr2[9],
                                    'Month 22': renewal_yr2[10], 'Month 23':
↪renewal_yr2[11],
                                    'Month 24': renewal_yr3[0], 'Month 25':␣
↪renewal_yr3[1],
                                    'Month 26': renewal_yr3[2], 'Month 27':␣
↪renewal_yr3[3],
                                    'Month 28': renewal_yr3[4], 'Month 29':␣
↪renewal_yr3[5],
                                    'Month 30': renewal_yr3[6], 'Month 31':␣
↪renewal_yr3[7],
                                    'Month 32': renewal_yr3[8], 'Month 33':␣
↪renewal_yr3[9],
                                    'Month 34': renewal_yr3[10], 'Month 35':␣
↪renewal_yr3[11]})

test_predictions = optimal_model.predict(design_matrix2)
model_test_error = (np.mean((test_predictions - Y_test2) ** 2)) ** 0.5
print(f'Test RMSE: {model_test_error}')
print(f'Min: {min(test_predictions.flatten())}')
print(f'Max: {max(test_predictions.flatten())}')

residuals = test_predictions.flatten() - Y_test2.flatten()
residuals_df = pd.DataFrame(data={'Predicted Value': test_predictions.
↪flatten(), 'Residual': residuals})

plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(data=residuals_df, x='Predicted Value', y='Residual', s=10,␣
↪alpha=0.5)
plt.title('Residual Plot')
```

```
plt.subplot(1, 2, 2)
sns.regplot(data=residuals_df, x='Predicted Value', y='Residual')
plt.title('Residual Plot')

plt.show()
print(f'Correlation: {residuals_df.corr()["Residual"].values[0]}')
```

```
Test RMSE: 19.901301737340162
Min: 748.0031595445404
Max: 802.8207797274391
```



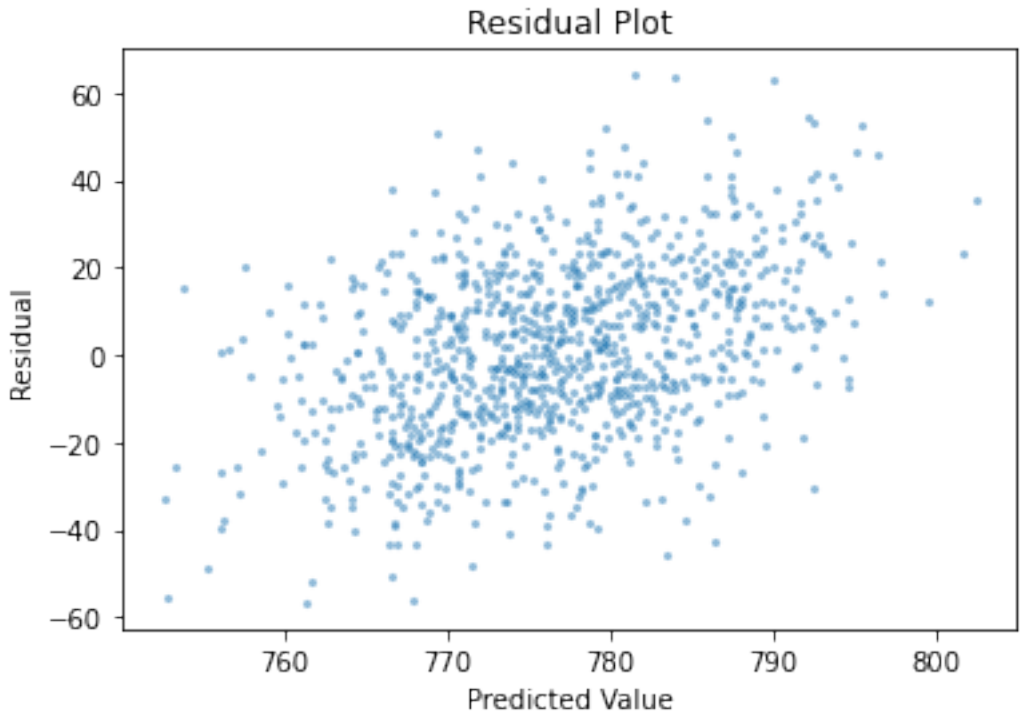

```
Correlation: 0.40407300445798916
```

[65]:
```
# Years 3, 4, and 5 predicting 6
three_plus = resampled_months.query('`Renewal Year` > 2 and `Renewal Year` < 7')
pepm_arr_ryr = three_plus.query('`Renewal Year` == 6')['PEPM'].values
ee_arr_ryr = three_plus.query('`Renewal Year` == 6')['EE Count'].values
pepm_arr_ryr_split = np.split(pepm_arr_ryr, 12)
ee_arr_ryr_split = np.split(ee_arr_ryr, 12)
ee_sums = [sum(item) for item in ee_arr_ryr_split]
ee_sums = np.zeros(1000)
for i in range(len(ee_arr_ryr_split)):
    ee_sums += ee_arr_ryr_split[i]
ee_props = [ee_arr_ryr_split[i] / ee_sums for i in range(len(ee_arr_ryr_split))]
weighted_totals = [np.multiply(ee_props[i], pepm_arr_ryr_split[i]) for i in␣
 ↪range(len(ee_props))]
actual_pepm = np.zeros(1000)
for i in range(len(weighted_totals)):
    actual_pepm += weighted_totals[i]
Y_test3 = actual_pepm
```

```
renewal_yr3 = np.split(resampled_months.query('`Renewal Year` == 3').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr4 = np.split(resampled_months.query('`Renewal Year` == 4').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
renewal_yr5 = np.split(resampled_months.query('`Renewal Year` == 5').
↪sort_values(['Year', 'Month'])['PEPM'].values, 12)
design_matrix3 = pd.DataFrame(data={'Month 1': renewal_yr3[0], 'Month 2':␣
↪renewal_yr3[1],
                                    'Month 3': renewal_yr3[2], 'Month 4':␣
↪renewal_yr3[3],
                                    'Month 5': renewal_yr3[4], 'Month 6':␣
↪renewal_yr3[5],
                                    'Month 7': renewal_yr3[6], 'Month 8':␣
↪renewal_yr3[7],
                                    'Month 8': renewal_yr3[8], 'Month 9':␣
↪renewal_yr3[9],
                                    'Month 10': renewal_yr3[10], 'Month 11':␣
↪renewal_yr3[11],
                                    'Month 12': renewal_yr4[0], 'Month 13':␣
↪renewal_yr4[1],
                                    'Month 14': renewal_yr4[2], 'Month 15':␣
↪renewal_yr4[3],
                                    'Month 16': renewal_yr4[4], 'Month 17':␣
↪renewal_yr4[5],
                                    'Month 18': renewal_yr4[6], 'Month 19':␣
↪renewal_yr4[7],
                                    'Month 20': renewal_yr4[8], 'Month 21':␣
↪renewal_yr4[9],
                                    'Month 22': renewal_yr4[10], 'Month 23':
↪renewal_yr4[11],
                                    'Month 24': renewal_yr5[0], 'Month 25':␣
↪renewal_yr5[1],
                                    'Month 26': renewal_yr5[2], 'Month 27':␣
↪renewal_yr5[3],
                                    'Month 28': renewal_yr5[4], 'Month 29':␣
↪renewal_yr5[5],
                                    'Month 30': renewal_yr5[6], 'Month 31':␣
↪renewal_yr5[7],
                                    'Month 32': renewal_yr5[8], 'Month 33':␣
↪renewal_yr5[9],
                                    'Month 34': renewal_yr5[10], 'Month 35':␣
↪renewal_yr5[11]})

test_predictions = optimal_model.predict(design_matrix3)
model_test_error = (np.mean((test_predictions - Y_test3) ** 2)) ** 0.5
```

```
print(f'Test RMSE: {model_test_error}')
print(f'Min: {min(test_predictions.flatten())}')
print(f'Max: {max(test_predictions.flatten())}')
pd.DataFrame(data={'Test Predictions': test_predictions.flatten(), 'Actual␣
 ↪Values':Y_test3.flatten()})

residuals = test_predictions.flatten() - Y_test3.flatten()
residuals_df = pd.DataFrame(data={'Predicted Value': test_predictions.
 ↪flatten(), 'Residual': residuals})
sns.scatterplot(data=residuals_df, x='Predicted Value', y='Residual', s=10,␣
 ↪alpha=0.5)
plt.title('Residual Plot')
plt.show()
print(f'Correlation: {residuals_df.corr()["Residual"].values[0]}')
```

```
Test RMSE: 19.783092063624593
Min: 752.5053323570154
Max: 802.4726402117981
```




```
Correlation: 0.4073851045139299
```

1.5 Unfinished – Ignore all below

Before we actually get started with the data wrangling/bootstrapping process, I think it is worth noting, again, that because we are training this model with data specifically from Huntington Hospital, we can really only say its predictions are valid for Huntington Hospital. That is not to say that its predictions aren't necessarily generalizable, but it take more rigorous testing to determine the extent to which this particular model will work for different clients.

[16]:
```
resampled = stripped_claims.query('Month == 1').sample(frac=1, replace=True,␣
↪axis=0)
resampled
```

[16]:

	Month	Year	Renewal Year	Domestic Medical Claims	Non-Domestic Medical Claims	Total Hospital Medical Claims	Non-Hospital Medical Claims	Total Medical Claims	Rx Claims	Rx Rebates	Rx Performance Guarantee	Stop Loss Reimbursement	Adjusted Paid Claims
12	1	2016	1	469762.84	336373.14	806135.98	356117.04	1162253.02	449875.53	0.0	0.0	0.0	1612128.55
0	1	2015	0	414313.73	324468.71	738782.44	385900.53	1124682.97	397849.95	0.0	0.0	0.0	1522532.92
24	1	2017	2	420330.71	275861.09	696191.80	567471.93	1263663.73	451547.90	0.0	0.0	0.0	2527327.46
24	1	2017	2	420330.71	275861.09	696191.80	567471.93	1263663.73	451547.90	0.0	0.0	0.0	2527327.46
48	1	2019	4	683149.70	333269.75	1016419.45	558604.05	1575023.50	698954.69	0.0	0.0	0.0	2273978.19
72	1	2021	6	308035.79	466745.22	774781.01	732462.14	1507243.15	601341.58	0.0	0.0	0.0	
0	1	2015	0	414313.73	324468.71	738782.44	385900.53	1124682.97	397849.95	0.0	0.0	0.0	
72	1	2021	6	308035.79	466745.22	774781.01	732462.14	1507243.15	601341.58	0.0	0.0	0.0	

```
72                      0.0                     0.0           2108584.73
0                       0.0                     0.0           1522532.92
72                      0.0                     0.0           2108584.73
```

```
    Member Count  EE Count        PEPM
12          6256      2781  579.693833
0           6146      2690  565.997368
24          6623      3020  587.191524
24          6623      3020  587.191524
48          6749      3108  731.653214
72          6758      3174  664.330413
0           6146      2690  565.997368
72          6758      3174  664.330413
```

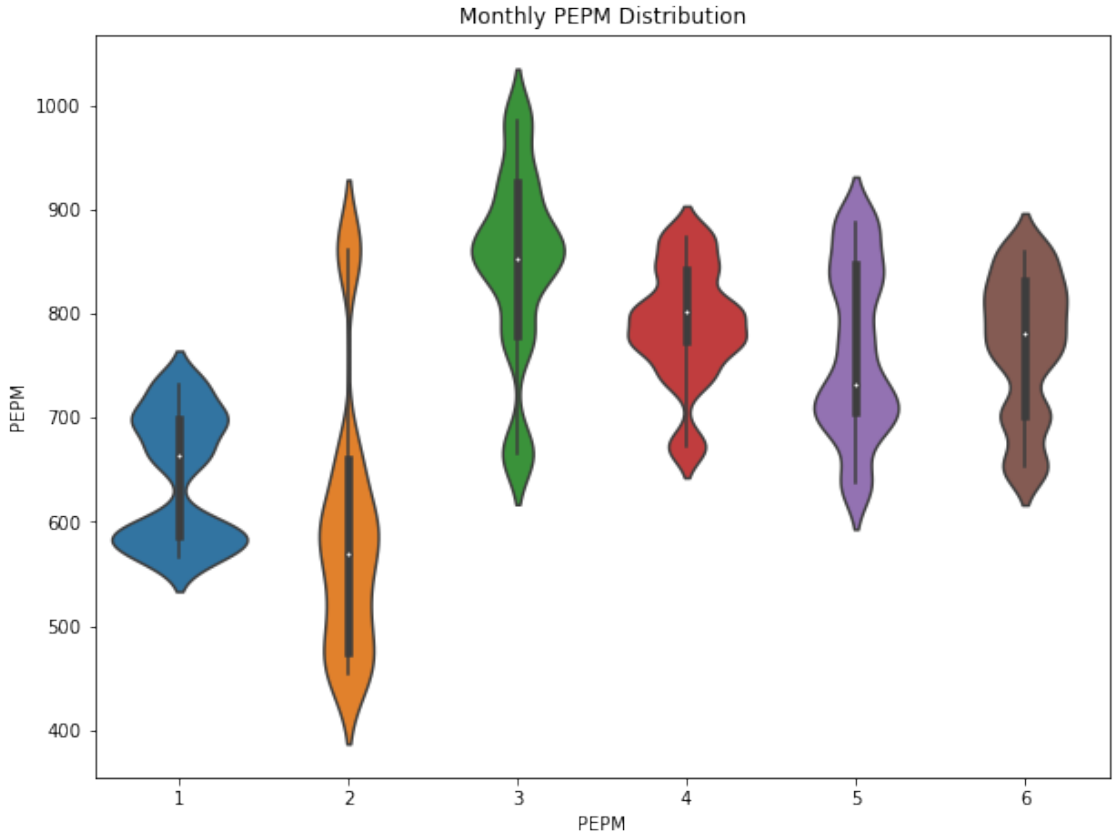
[13]:
```
# Bootstrapping time!
resampled_months = pd.DataFrame(columns=['Month',
                                         'Year',
                                         'Total Medical Claims',
                                         'Rx Claims',
                                         'Rx Rebates',
                                         'Rx Performance Guarantee',
                                         'Stop Loss Reimbursement',
                                         'Adjusted Paid Claims'])

for num in np.arange(1, 13):
    for i in np.arange(100):
        each_month = stripped_claims.query('Month == @num').loc[:, 'Year':'Stop␣
 ↪Loss Reimbursement']
        month = [num, num, num, num, num, num, num, num]
        year = [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]
        df = {'Month': month, 'Year': year}
        resampled = pd.DataFrame(data=df)
        for column in each_month.columns[2:]:
            resample = each_month.loc[:, 'Year':column].sample(frac=1,␣
 ↪replace=True, axis=0).reset_index().drop('index', axis=1)
            resampled[column] = resample[column]
        #claims_paid = resampled['Total Medical Claims'] \
        #            + resampled['Rx Claims']
        #reimbursements = resampled['Rx Rebates'] \
        #               + resampled['Rx Performance Guarantee'] \
        #               + resampled['Stop Loss Reimbursement']
        #resampled['Adjusted Paid Claims'] = claims_paid - reimbursements
        resampled_months = resampled_months.append(resampled)
resampled_months = resampled_months.reset_index().drop('index', axis=1).dropna()

# Further data transformations
def add_renewal_year(month, year):
```

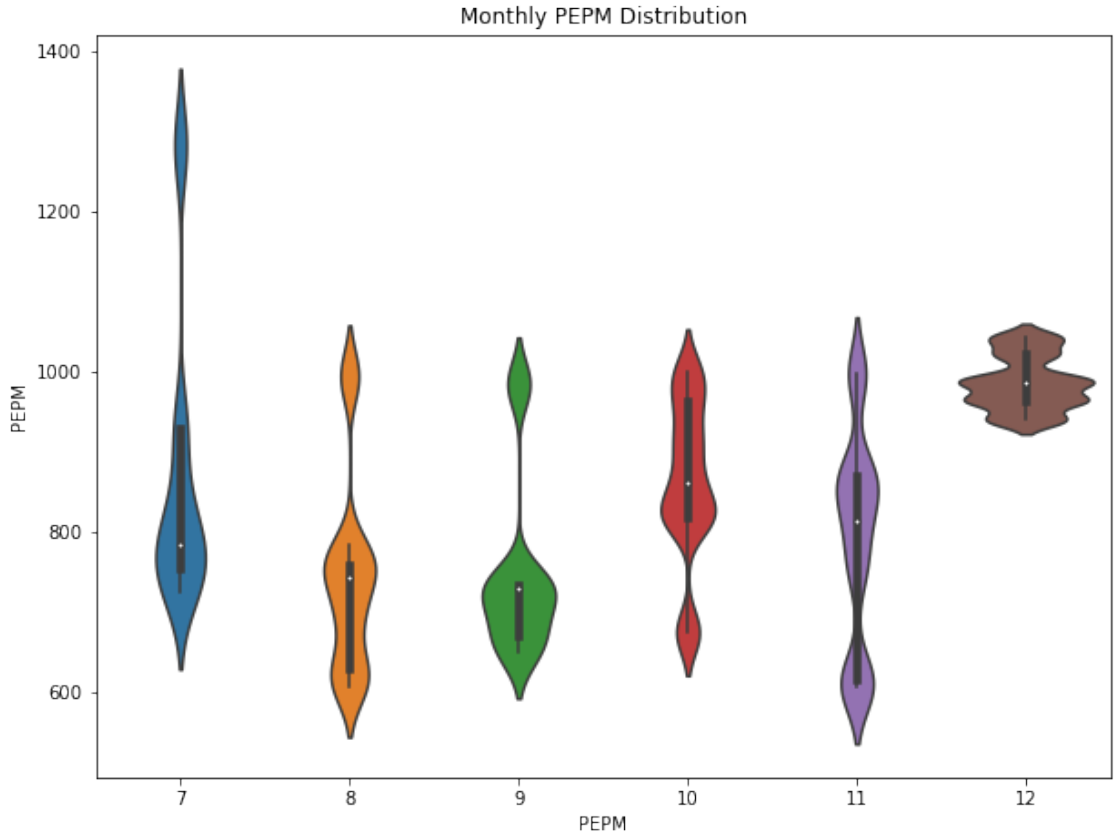
```
    '''
    Function that returns a list of the corresponding Renewal Year for␣
↪Huntington Medical Group
    -----
    Inputs:

    month (Python list) - Month column in resampled_months DataFrame

    year (Python list) - Year column in resampled_months DataFrame
    -----
    Output:

    renewal_year (Python list) - Corresponding renewal year column to be added␣
↪to the
                                 resampled_months DataFrame
    '''
    renewal_year = []
    for i in np.arange(0, len(month)):
        if month[i] <= 5 and year[i] == 2015:
            renewal_year.append(0)
        elif month[i] >= 6 and year[i] == 2015:
            renewal_year.append(1)
        elif month[i] <=5 and year[i] == 2016:
            renewal_year.append(1)
        elif month[i] >= 6 and year[i] == 2016:
            renewal_year.append(2)
        elif month[i] <= 5 and year[i] == 2017:
            renewal_year.append(2)
        elif month[i] >= 6 and year[i] == 2017:
            renewal_year.append(3)
        elif month[i] <= 5 and year[i] == 2018:
            renewal_year.append(3)
        elif month[i] >= 6 and year[i] == 2018:
            renewal_year.append(4)
        elif month[i] <= 5 and year[i] == 2019:
            renewal_year.append(4)
        elif month[i] >= 6 and year[i] == 2019:
            renewal_year.append(5)
        elif month[i] <= 5 and year[i] == 2020:
            renewal_year.append(5)
        elif month[i] >= 6 and year[i] == 2020:
            renewal_year.append(6)
        elif month[i] <= 5 and year[i] == 2021:
            renewal_year.append(6)
        elif month[i] >= 6 and year[i] == 2021:
            renewal_year.append(7)
        elif month[i] <= 5 and year[i] == 2022:
```

```
            renewal_year.append(7)
        else:
            renewal_year.append(10)
    return renewal_year
resampled_months['Renewal Year'] = add_renewal_year(resampled_months['Month'].
↪values, \
                                                    resampled_months['Year'].
↪values)
resampled_months
```

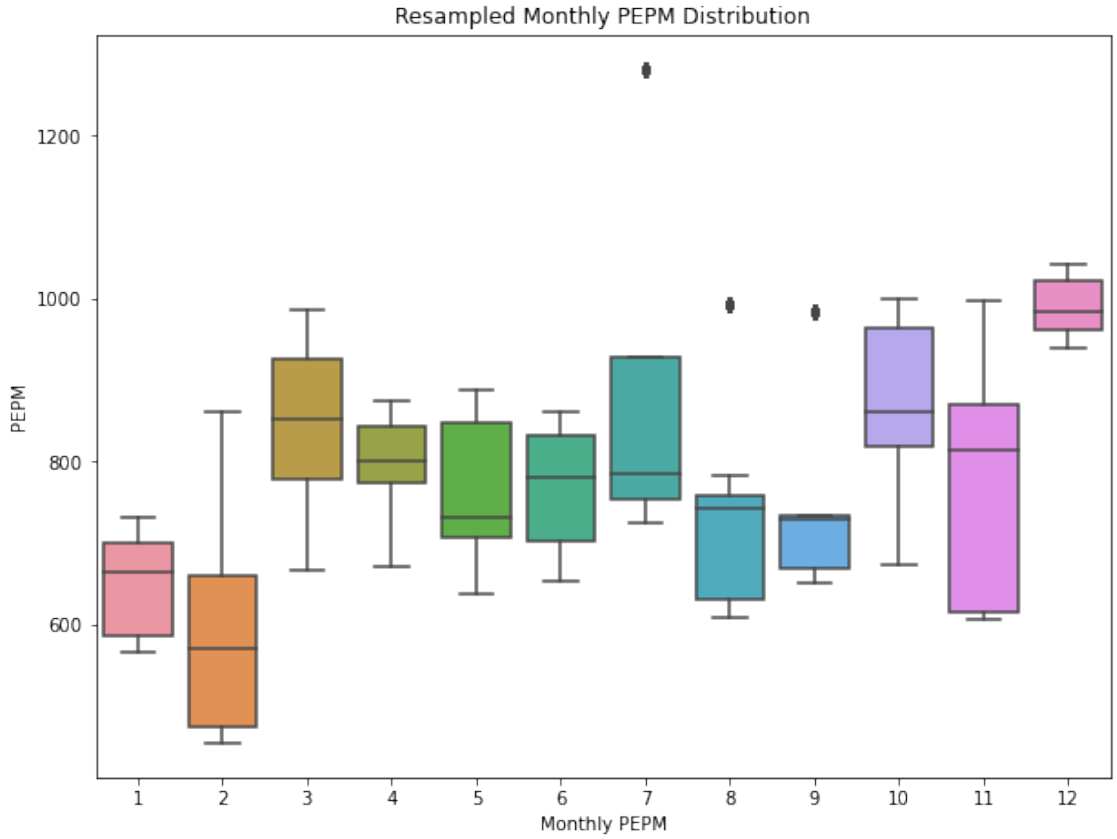
[13]:
```
Empty DataFrame
Columns: [Month, Year, Total Medical Claims, Rx Claims, Rx Rebates, Rx
Performance Guarantee, Stop Loss Reimbursement, Adjusted Paid Claims, Domestic
Medical Claims, Non-Domestic Medical Claims, Total Hospital Medical Claims, Non-
Hospital Medical Claims, Renewal Year]
Index: []
```

[41]:
```
# Violin plots for January-June
plt.figure(figsize=(10, 7.5))
sns.violinplot(data=resampled_months.query('Month <= 6'), x='Month', y='PEPM')
plt.xlabel('PEPM')
plt.title('Monthly PEPM Distribution')
plt.show()

# Violin plots for July-December
plt.figure(figsize=(10, 7.5))
sns.violinplot(data=resampled_months.query('Month > 6'), x='Month', y='PEPM')
plt.xlabel('PEPM')
plt.title('Monthly PEPM Distribution')
plt.show()
```


Monthly PEPM Distribution
PEPM
1000
900
800
700
600
500
400
1
2
3
4
5
6
PEPM



[42]:
```
# Boxplot version of the above
plt.figure(figsize=(10, 7.5))
sns.boxplot(data=resampled_months, x='Month', y='PEPM')
plt.xlabel('Monthly PEPM')
plt.title('Resampled Monthly PEPM Distribution')
plt.show()
```



[164]:
```
np.split(resampled_months.sort_values(['Month', 'Year', 'Renewal Year'],␣
↪ascending=True).query('`Renewal Year` == 1')['PEPM'].values, 12)[0]
```

[164]:
```
array([655.3354636, 655.3354636, 655.3354636, 579.6938332, 664.3304127,
       565.997368 , 646.3405145, 646.3405145, 646.3405145, 646.3405145,
       565.997368 , 664.3304127, 565.997368 , 699.2933386, 579.6938332,
       655.3354636, 646.3405145, 579.6938332, 594.6892155, 587.1915244,
       587.1915244, 579.6938332, 664.3304127, 664.3304127, 579.6938332,
       579.6938332, 664.3304127, 587.1915244, 565.997368 , 579.6938332,
       594.6892155, 587.1915244, 587.1915244, 646.3405145, 664.3304127,
       565.997368 , 655.3354636, 594.6892155, 655.3354636, 699.2933386,
       587.1915244, 594.6892155, 699.2933386, 594.6892155, 594.6892155,
       579.6938332, 579.6938332, 699.2933386, 587.1915244, 655.3354636,
       587.1915244, 664.3304127, 587.1915244, 664.3304127, 565.997368 ,
       579.6938332, 699.2933386, 594.6892155, 655.3354636, 594.6892155,
       594.6892155, 579.6938332, 594.6892155, 579.6938332, 699.2933386,
       664.3304127, 579.6938332, 587.1915244, 699.2933386, 655.3354636,
       587.1915244, 594.6892155, 664.3304127, 699.2933386, 594.6892155,
       587.1915244, 579.6938332, 664.3304127, 594.6892155, 699.2933386,
       565.997368 , 565.997368 , 594.6892155, 646.3405145, 565.997368 ,
```

```
       565.997368 , 646.3405145, 587.1915244, 594.6892155, 587.1915244,
       565.997368 , 664.3304127, 699.2933386, 699.2933386, 699.2933386,
       646.3405145, 664.3304127, 579.6938332, 579.6938332, 664.3304127])
```

[165]:

[171]:
```
resampled_months.query('`Renewal Year` == 4')
```

[171]:

	Month	Year	EE Count	PEPM	Renewal Year
2003	6	2018	3210.0	808.097704	4
2007	6	2018	3095.0	757.194834	4
2011	6	2018	3167.0	831.153148	4
2015	6	2018	3210.0	808.097704	4
2019	6	2018	2675.0	652.563948	4
…	…	…	…	…	…
4783	12	2018	3055.0	962.650276	4
4787	12	2018	3116.0	966.030113	4
4791	12	2018	3001.0	991.919943	4
4795	12	2018	3055.0	962.650276	4
4799	12	2018	3055.0	962.650276	4

[700 rows x 5 columns]

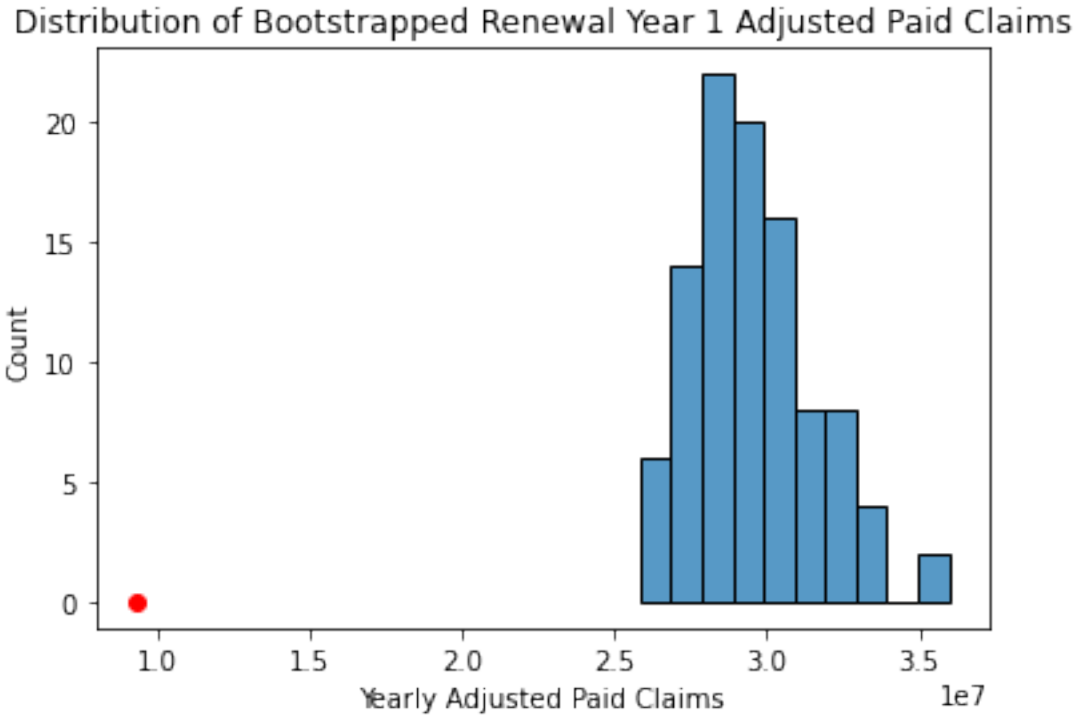
[168]:

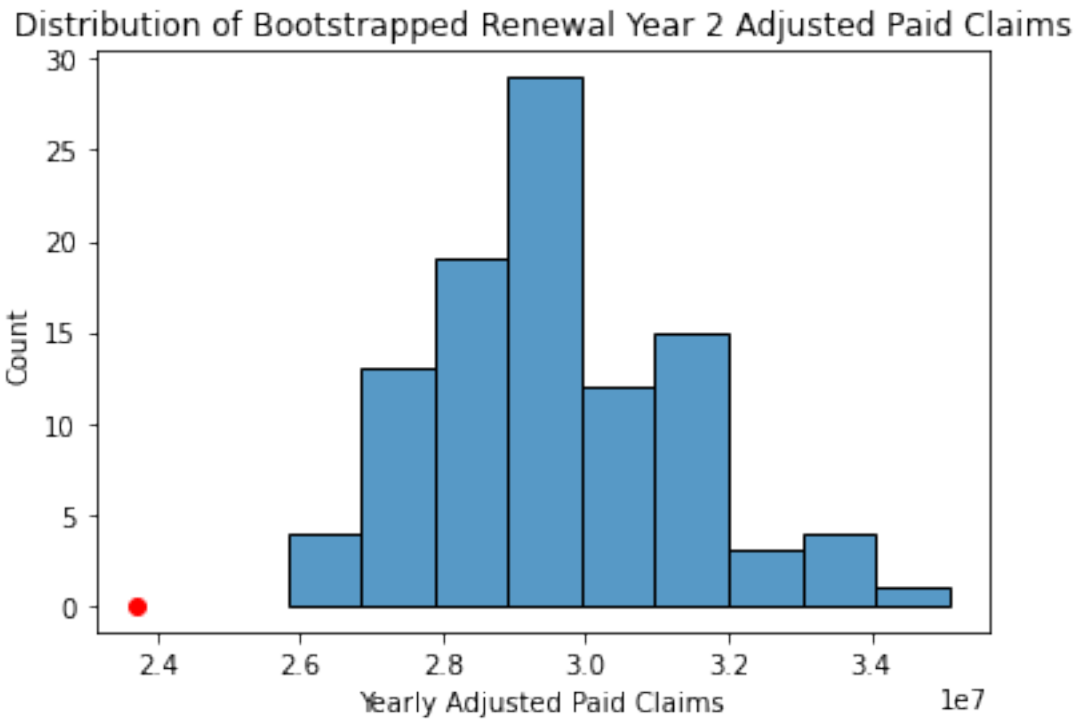
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-168-540a594a3f48> in <module>
----> 1 pepm_ryr1 = resampled_yearly_pepm(1, 100)
      2 pepm_ryr2 = resampled_yearly_pepm(2, 100)
      3 pepm_ryr3 = resampled_yearly_pepm(3, 100)
      4 pepm_ryr4 = resampled_yearly_pepm(4, 100)

NameError: name 'resampled_yearly_pepm' is not defined
```

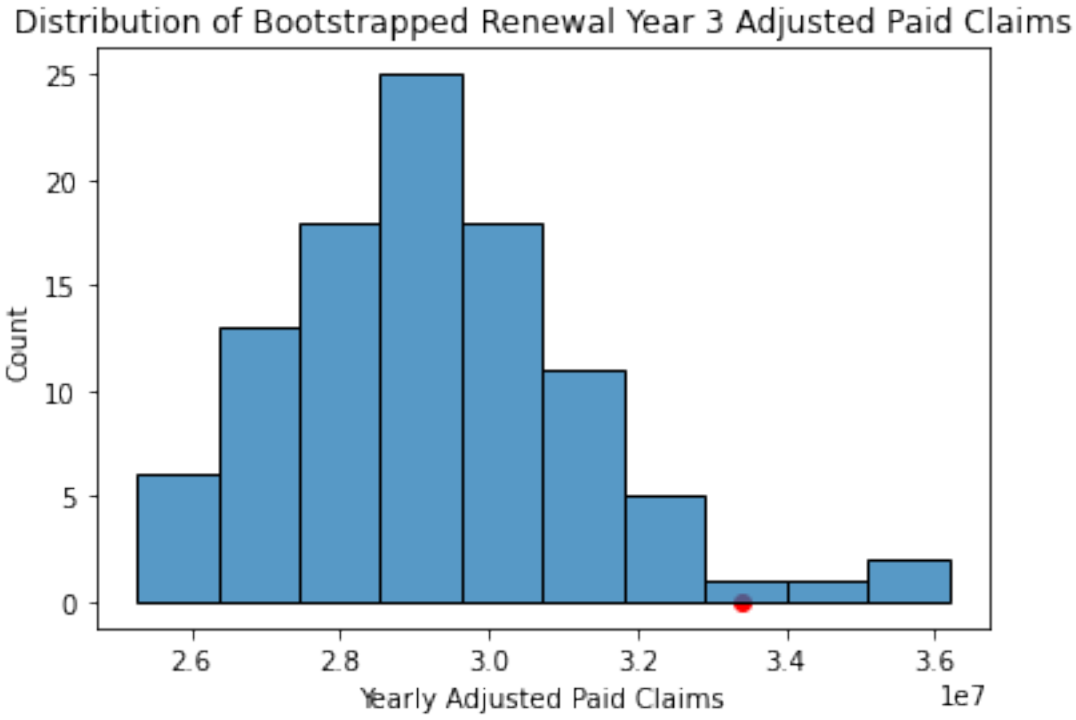
Now that we have a bootstrapped dataset, let's look at it and see if it seems to have successfully rebuilt the population.

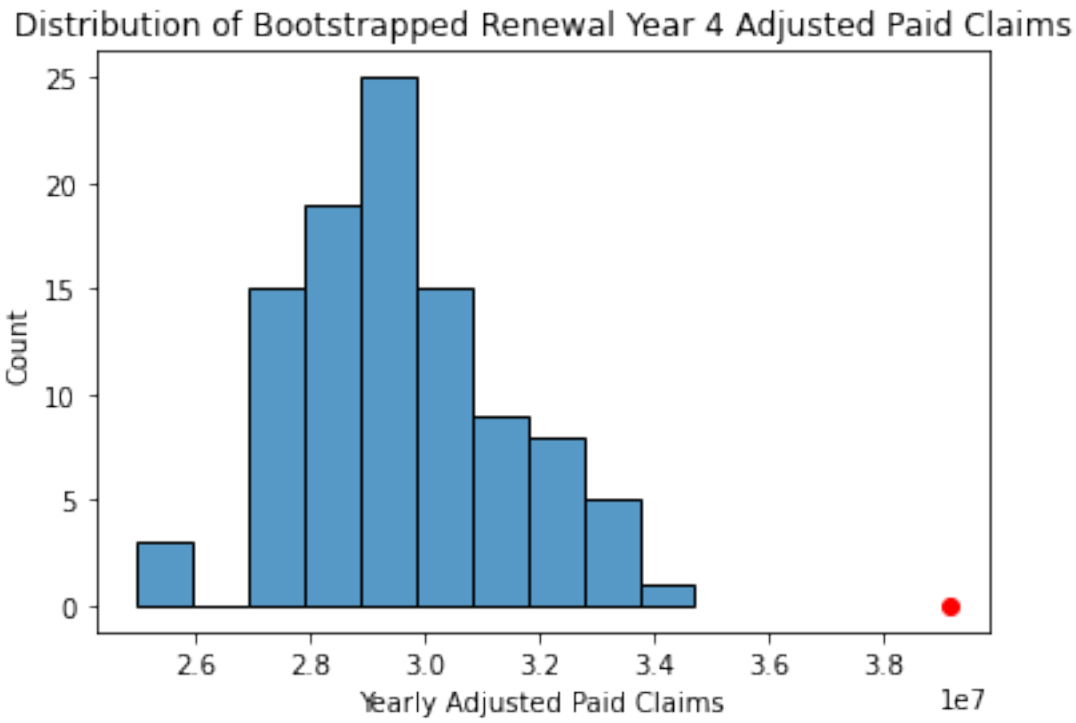
[33]:
```
apcs = [apc_ryr1, apc_ryr2, apc_ryr3, apc_ryr4, apc_ryr5, apc_ryr6]
observed_yearly_apcs = claims.groupby('Renewal Year').sum()['Adjusted Paid␣
↪Claims'].values
i = 0
for apc in apcs:
    observed_year_apc = observed_yearly_apcs[i]
    sns.histplot(data=apc)
    plt.xlabel('Yearly Adjusted Paid Claims')
```

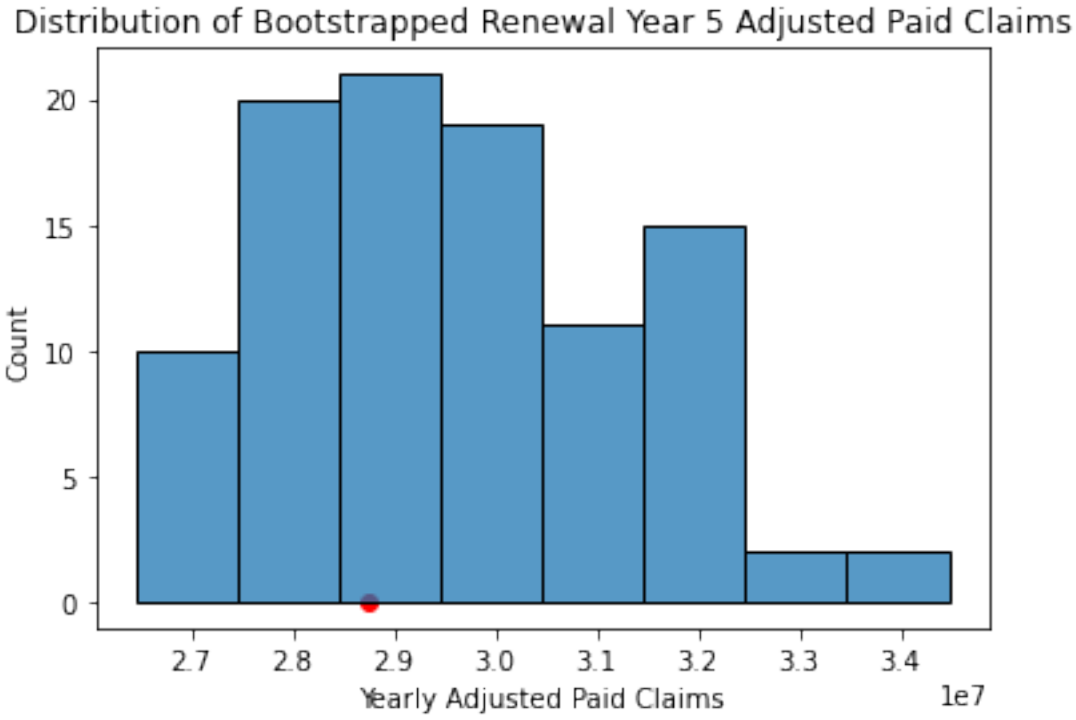
```
    plt.title(f'Distribution of Bootstrapped Renewal Year {i + 1} Adjusted Paid␣
↪Claims')
    plt.scatter(observed_year_apc, -0.01, color='red')
    plt.show()
    i += 1
```

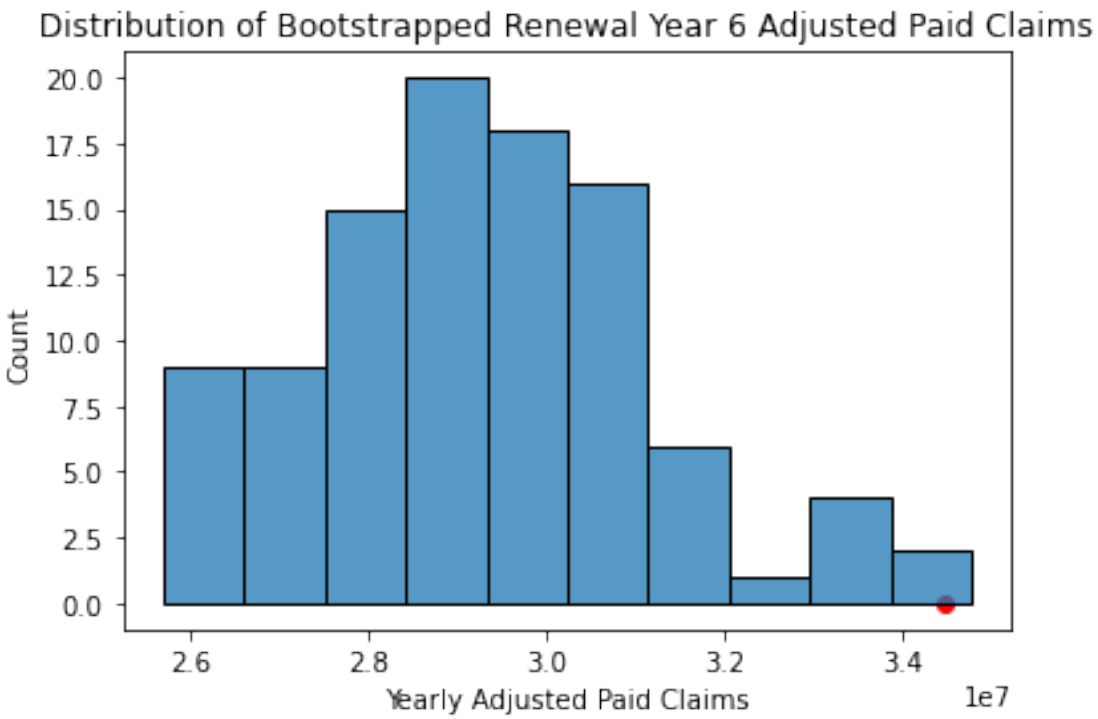



Distribution of Bootstrapped Renewal Year 2 Adjusted Paid Claims
Count
30
25
20
15
10
5
0
2.4
2.6
2.8
3.0
3.2
3.4
Yearly Adjusted Paid Claims
1e7


Distribution of Bootstrapped Renewal Year 3 Adjusted Paid Claims
Count
25
20
15
10
5
0
2.6
2.8
3.0
3.2
3.4
3.6
Yearly Adjusted Paid Claims
1e7

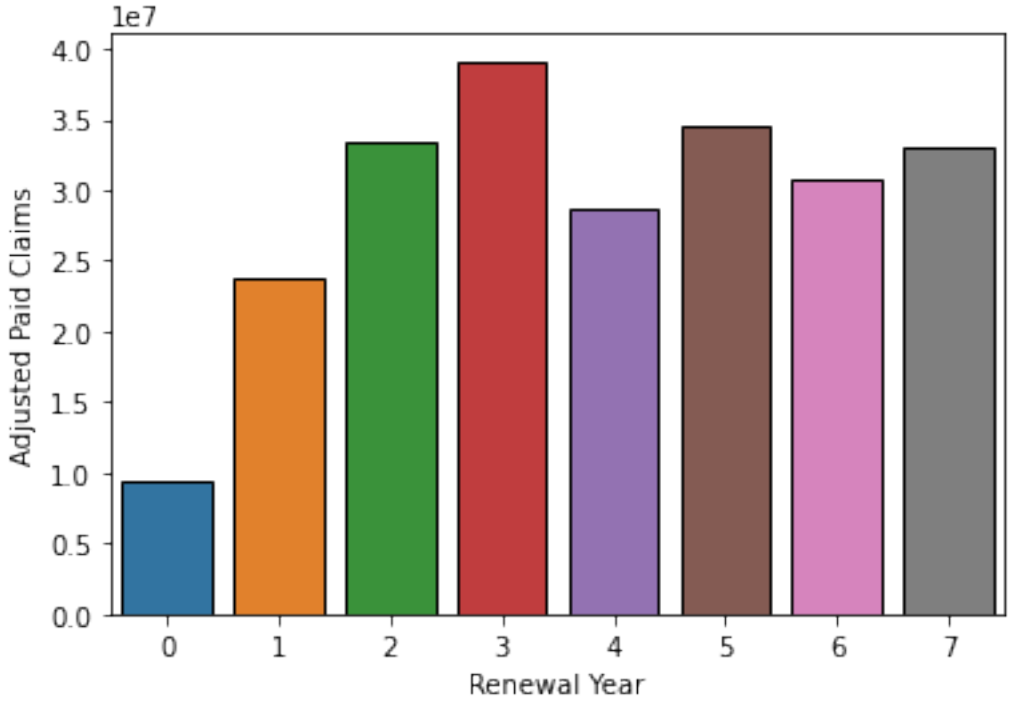

Distribution of Bootstrapped Renewal Year 4 Adjusted Paid Claims
Count
25
20
15
10
5
0
2.6
2.8
3.0
3.2
3.4
3.6
3.8
Yearly Adjusted Paid Claims
1e7


Distribution of Bootstrapped Renewal Year 5 Adjusted Paid Claims
Count
20
15
10
5
0
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
Yearly Adjusted Paid Claims
1e7



[319]:
```
sns.histplot()
```

[319]:
```
<AxesSubplot:xlabel='Renewal Year', ylabel='Adjusted Paid Claims'>
```



[357]:
```
def strip_month_year(month_number, year):
    '''
    Strips Adjusted Paid Claims values for a given month_number and year from␣
 ↪resampled_months
    and returns a NumPy array
    -----
    Inputs:

    month_number (int) - Integer corresponding to the month desired

    year (int) - Integer corresponding to the year desired
    -----
    Output:

    month_year (NumPy Array) - Adjusted Paid Claims values from␣
 ↪resampled_months corresponding to
                               the month and year specified
    '''
    return resampled_months.query('Month == @month_number and Year ==␣
 ↪@year')['Adjusted Paid Claims'].values

jan_2015, feb_2015, mar_2015 = [strip_month_year(month_num, 2015) for month_num␣
 ↪in np.arange(1, 4)]
apr_2015, may_2015, jun_2015 = [strip_month_year(month_num, 2015) for month_num␣
 ↪in np.arange(4, 7)]
jul_2015, aug_2015, sep_2015 = [strip_month_year(month_num, 2015) for month_num␣
 ↪in np.arange(7, 10)]
oct_2015, nov_2015, dec_2015 = [strip_month_year(month_num, 2015) for month_num␣
 ↪in np.arange(10, 13)]

jan_2016, feb_2016, mar_2016 = [strip_month_year(month_num, 2016) for month_num␣
 ↪in np.arange(1, 4)]
apr_2016, may_2016, jun_2016 = [strip_month_year(month_num, 2016) for month_num␣
 ↪in np.arange(4, 7)]
jul_2016, aug_2016, sep_2016 = [strip_month_year(month_num, 2016) for month_num␣
 ↪in np.arange(7, 10)]
oct_2016, nov_2016, dec_2016 = [strip_month_year(month_num, 2016) for month_num␣
 ↪in np.arange(10, 13)]

jan_2017, feb_2017, mar_2017 = [strip_month_year(month_num, 2017) for month_num␣
 ↪in np.arange(1, 4)]
apr_2017, may_2017, jun_2017 = [strip_month_year(month_num, 2017) for month_num␣
 ↪in np.arange(4, 7)]
```

```
jul_2017, aug_2017, sep_2017 = [strip_month_year(month_num, 2017) for month_num␣
 ↪in np.arange(7, 10)]
oct_2017, nov_2017, dec_2017 = [strip_month_year(month_num, 2017) for month_num␣
 ↪in np.arange(10, 13)]

jan_2018, feb_2018, mar_2018 = [strip_month_year(month_num, 2018) for month_num␣
 ↪in np.arange(1, 4)]
apr_2018, may_2018, jun_2018 = [strip_month_year(month_num, 2018) for month_num␣
 ↪in np.arange(4, 7)]
jul_2018, aug_2018, sep_2018 = [strip_month_year(month_num, 2018) for month_num␣
 ↪in np.arange(7, 10)]
oct_2018, nov_2018, dec_2018 = [strip_month_year(month_num, 2018) for month_num␣
 ↪in np.arange(10, 13)]

jan_2019, feb_2019, mar_2019 = [strip_month_year(month_num, 2019) for month_num␣
 ↪in np.arange(1, 4)]
apr_2019, may_2019, jun_2019 = [strip_month_year(month_num, 2019) for month_num␣
 ↪in np.arange(4, 7)]
jul_2019, aug_2019, sep_2019 = [strip_month_year(month_num, 2019) for month_num␣
 ↪in np.arange(7, 10)]
oct_2019, nov_2019, dec_2019 = [strip_month_year(month_num, 2019) for month_num␣
 ↪in np.arange(10, 13)]

jan_2020, feb_2020, mar_2020 = [strip_month_year(month_num, 2020) for month_num␣
 ↪in np.arange(1, 4)]
apr_2020, may_2020, jun_2020 = [strip_month_year(month_num, 2020) for month_num␣
 ↪in np.arange(4, 7)]
jul_2020, aug_2020, sep_2020 = [strip_month_year(month_num, 2020) for month_num␣
 ↪in np.arange(7, 10)]
oct_2020, nov_2020, dec_2020 = [strip_month_year(month_num, 2020) for month_num␣
 ↪in np.arange(10, 13)]

jan_2021, feb_2021, mar_2021 = [strip_month_year(month_num, 2021) for month_num␣
 ↪in np.arange(1, 4)]
apr_2021, may_2021, jun_2021 = [strip_month_year(month_num, 2021) for month_num␣
 ↪in np.arange(4, 7)]
jul_2021, aug_2021, sep_2021 = [strip_month_year(month_num, 2021) for month_num␣
 ↪in np.arange(7, 10)]
oct_2021, nov_2021, dec_2021 = [strip_month_year(month_num, 2021) for month_num␣
 ↪in np.arange(10, 13)]
```

[373]:
```
def aggregate_renewal_year(renewal_year):
    '''
    For the specified renewal_year, calculates the Total Adjusted Paid Claims␣
 ↪values for
    the 100 resamples
```

```
    -----
    Input:

    renewal_year (int) - The specified renewal year (starting with 1 from 6/
↪15-5/16)
    -----
    Outputs:

    renewal_year_totals (NumPy Array) - The Adjusted Paid Values for the 100␣
↪resamples
                                        corresponding to the specified␣
↪renewal_year

    renewal_label (NumPy Array) - NumPy Array consisting of the Renewal Year␣
↪for inputting
                                    into a DataFrame
    '''
    renewal_label = np.zeros(0)
    if renewal_year == 1:
        renewal_year_totals = jun_2015 + jul_2015 + aug_2015 + sep_2015 + \
                              oct_2015 + nov_2015 + dec_2015 + jan_2016 + \
                              feb_2016 + mar_2016 + apr_2016 + may_2016
    elif renewal_year == 2:
        renewal_year_totals = jun_2016 + jul_2016 + aug_2016 + sep_2016 + \
                              oct_2016 + nov_2016 + dec_2016 + jan_2017 + \
                              feb_2017 + mar_2017 + apr_2017 + may_2017
    elif renewal_year == 3:
        renewal_year_totals = jun_2017 + jul_2017 + aug_2017 + sep_2017 + \
                              oct_2017 + nov_2017 + dec_2017 + jan_2018 + \
                              feb_2018 + mar_2018 + apr_2018 + may_2018
    elif renewal_year == 4:
        renewal_year_totals = jun_2018 + jul_2018 + aug_2018 + sep_2018 + \
                              oct_2018 + nov_2018 + dec_2018 + jan_2019 + \
                              feb_2019 + mar_2019 + apr_2019 + may_2019
    elif renewal_year == 5:
        renewal_year_totals = jun_2019 + jul_2019 + aug_2019 + sep_2019 + \
                              oct_2019 + nov_2019 + dec_2019 + jan_2020 + \
                              feb_2020 + mar_2020 + apr_2020 + may_2020
    elif renewal_year == 6:
        renewal_year_totals = jun_2020 + jul_2020 + aug_2020 + sep_2020 + \
                              oct_2020 + nov_2020 + dec_2020 + jan_2021 + \
                              feb_2021 + mar_2021 + apr_2021 + may_2021
    for i in range(len(renewal_year_totals)):
        renewal_label = np.append(renewal_label, int(renewal_year))
    return renewal_year_totals, renewal_label
```

[372]:

[372]:
```
numpy.float64
```

[321]:
```
resampled_months['Year'].value_counts()
```

[321]:
```
2020    1200
2019    1200
2018    1200
2017    1200
2016    1200
2015    1200
2021     400
Name: Year, dtype: int64
```

[322]:
```
resampled_months['Month'].value_counts()
```

[322]:
```
4     700
3     700
2     700
1     700
12    600
11    600
10    600
9     600
8     600
7     600
6     600
5     600
Name: Month, dtype: int64
```

[ ]:
```

```