```python
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import paramiko
```

```python
# connect to remote sftp server to retrieve file
# credentials removed for security

client = paramiko.SSHClient()
# automatically add keys without requiring human intervention
client.set_missing_host_key_policy( paramiko.AutoAddPolicy() )

client.connect(sftp_url, port=22, username=sftp_user, password=sftp_pass)

sftp = client.open_sftp()
files = sftp.listdir()

# lists all the folders in the directory after connecting through the sftp
print(files)
```

```
['.bash_logout', '.bash_profile', '.bashrc', 'hue-health']
```

```python
# read file to memory from the sftp server
with sftp.open('/home/jsinger/ /healthcare_claims_data.xlsx') as f:
    f.prefetch()
    # we have two sheets in the Excel file; save them as two variables
    claims_data_pre65 = pd.read_excel(f, sheet_name='Pre65')
    claims_data_post65 = pd.read_excel(f, sheet_name='Post65')
```

```python
# increase the number of columns displayed in this notebook to 100
pd.options.display.max_columns = 100
```

```python
claims_data_pre65.columns == claims_data_post65.columns
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True])
```

```python
# both sheets contain the same features, so we'll combine them into one dataframe
# first, let's add a column to each df to keep track of the original sheet
claims_data_pre65['sheet_name'] = 'Pre65'
claims_data_post65['sheet_name'] = 'Post65'

# next, we'll combine the two sheets of data into one df and reset the index
claims_data = pd.concat([claims_data_pre65,claims_data_post65], axis=0, ignore_index=
```

```python
claims_data.head()
```

	LINE_NO	CLAIM_ID	CLAIM_TYPE	PROV_NPI	PROV_TIN	PROV_NAME	PROV_ADDR1	PROV_ADDI
0	128	Removed for PHI						Na
1	129							Na
2	130							Na
3	131							Na
4	132							Na

```python
claims_data['sheet_name'].value_counts()
```

```
Pre65     855830
Post65    196320
Name: sheet_name, dtype: int64
```

```python
claims_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1052150 entries, 0 to 1052149
Data columns (total 32 columns):
 #   Column                              Non-Null Count    Dtype
---  ------                              --------------    -----
 0   LINE_NO                             1052150 non-null  int64
 1   CLAIM_ID                            1052150 non-null  object
 2   CLAIM_TYPE                          1052150 non-null  object
 3   PROV_NPI                            978282 non-null   float64
 4   PROV_TIN                            1052150 non-null  object
 5   PROV_NAME                           1052150 non-null  object
 6   PROV_ADDR1                          1052150 non-null  object
 7   PROV_ADDR2                          14981 non-null    object
 8   PROV_CITY                           1052150 non-null  object
 9   PROV_ST                             1052150 non-null  object
 10  PROV_ZIP                            1052150 non-null  int64
 11  MEMBER_ID                           1052150 non-null  int64
 12  MEMBER_ZIP                          1052150 non-null  int64
 13  PROV_SPECIALTY                      605031 non-null   object
 14  PLACE_OF_SERVICE                    1052150 non-null  int64
 15  MSC                                 1052150 non-null  object
 16  PROC_CD                             985980 non-null   object
 17  MODIFIER                            319491 non-null   object
 18  REVENUE_CD                          308913 non-null   float64
 19  SERVICE_COUNT                       1052150 non-null  int64
 20  UB_BILL_TYPE                        308913 non-null   object
 21  DRG_CODE                            72498 non-null    float64
 22  DIAG1_CD                            1052150 non-null  object
 23  DIAG2_CD                            696613 non-null   object
 24  DIAG3_CD                            488908 non-null   object
 25  Eligible Charges                    1052150 non-null  float64
 26  Re-Priced
Amount                      0 non-null        float64
 27  Provider In
PPO Network
(Yes / No)  0 non-null        float64
 28  Not Re-Priced
(Indicate w/ "x")     0 non-null        float64
 29  Capitated (Y/N)                     0 non-null        float64
 30  Capitation Description              0 non-null        float64
 31  sheet_name                          1052150 non-null  object
dtypes: float64(9), int64(6), object(17)
memory usage: 256.9+ MB
```

```python
claims_data.isnull().sum()
```

```
LINE_NO                                     0
CLAIM_ID                                    0
CLAIM_TYPE                                  0
PROV_NPI                                73868
PROV_TIN                                    0
PROV_NAME                                   0
PROV_ADDR1                                  0
PROV_ADDR2                            1037169
PROV_CITY                                   0
PROV_ST                                     0
PROV_ZIP                                    0
MEMBER_ID                                   0
MEMBER_ZIP                                  0
PROV_SPECIALTY                         447119
PLACE_OF_SERVICE                            0
MSC                                         0
PROC_CD                                 66170
MODIFIER                               732659
REVENUE_CD                             743237
SERVICE_COUNT                               0
UB_BILL_TYPE                           743237
DRG_CODE                               979652
DIAG1_CD                                    0
DIAG2_CD                               355537
DIAG3_CD                               563242
Eligible Charges                            0
Re-Priced\nAmount                     1052150
Provider In\nPPO Network\n(Yes / No)  1052150
Not Re-Priced\n(Indicate w/ "x")      1052150
Capitated (Y/N)                       1052150
Capitation Description                1052150
sheet_name                                  0
dtype: int64
```

```python
claims_data.dtypes
```

```
LINE_NO                                 int64
CLAIM_ID                               object
CLAIM_TYPE                             object
PROV_NPI                              float64
PROV_TIN                               object
PROV_NAME                              object
PROV_ADDR1                             object
PROV_ADDR2                             object
PROV_CITY                              object
PROV_ST                                object
PROV_ZIP                                int64
MEMBER_ID                               int64
MEMBER_ZIP                              int64
PROV_SPECIALTY                         object
PLACE_OF_SERVICE                        int64
MSC                                    object
PROC_CD                                object
MODIFIER                               object
REVENUE_CD                            float64
SERVICE_COUNT                           int64
UB_BILL_TYPE                           object
DRG_CODE                              float64
DIAG1_CD                               object
DIAG2_CD                               object
DIAG3_CD                               object
Eligible Charges                      float64
Re-Priced\nAmount                     float64
Provider In\nPPO Network\n(Yes / No)  float64
Not Re-Priced\n(Indicate w/ "x")      float64
Capitated (Y/N)                       float64
Capitation Description                float64
sheet_name                             object
dtype: object
```

```python
# converting LINE_NO and MEMBER_ID columns to object type
claims_data['LINE_NO'] = claims_data['LINE_NO'].astype('object')
claims_data['MEMBER_ID'] = claims_data['MEMBER_ID'].astype('object')
```

```python
claims_data.describe(include='all').loc[['unique'],['LINE_NO','CLAIM_ID','MEMBER_ID']]
```

	LINE_NO	CLAIM_ID	MEMBER_ID
unique	1052150.0	477389.0	34985.0

The 1,052,150 line items (services) account for 477,389 claims across 34,985 members.

Initial Data Cleaning & Data Quality Check

Let's drop the following columns that have all missing values:

- Re-Priced\nAmount
- Provider In\nPPO Network\n(Yes / No)
- Not Re-Priced\n(Indicate w/ "x")
- Capitated (Y/N)
- Capitation Description

Let's also drop the following columns that will not be useful for our analysis:

- PROV_TIN
- PROV_ADDR1
- PROV_ADDR2

```python
# drop columns
drop_columns = ['Re-Priced\nAmount', 'Provider In\nPPO Network\n(Yes / No)', 'Not Re-
                'Capitated (Y/N)', 'Capitation Description', 'PROV_TIN', 'PROV_ADDR1',

claims_data.drop(drop_columns, axis=1, inplace=True)
```

```python
claims_data.head()
```

	LINE_NO	CLAIM_ID	CLAIM_TYPE	PROV_NPI	PROV_NAME	PROV_CITY	PROV_ST	PROV_ZIP	ME
0	128	Removed for PHI							1
1	129								1
2	130								1
3	131								
4	132								

Let's do a quick data quality check on the following features:

- CLAIM_TYPE: should be either Professional (PR) or Institutional (IN)
- UB_BILL_TYPE: should be 3 digits/characters. See here for reference

```python
claims_data['CLAIM_TYPE'].value_counts()
```

```
PR    743237
IN    308913
Name: CLAIM_TYPE, dtype: int64
```

```python
bill_type_wrong_count = 0
for value in claims_data['UB_BILL_TYPE']:
    if np.nan:
        pass
    elif len(value) != 3:
        bill_type_wrong_count += 1

bill_type_wrong_count
```

```
0
```

```python
claims_data['UB_BILL_TYPE'].unique()
```

```
array([nan, 831, 131, 113, 141, 137, 857, 851, 133, 117, 111, 727, 837,
       147, 112, 321, 731, 132, 721, 134, 861, 114, 814, 711, 852, 323,
       233, 854, 211, 217, 324, 334, 223, 322, 214, 212, 824, 812, 853,
       222, 863, 862, 213, 762, 891, 841, 234, 135, 332, 140, 813, 181,
       182, 150, 184, 822, 817, 821, 897, 893, 763, 743, 333, 115, 744,
       771, '32G', 811, 341, 130, 221, 224, 713, 753, 823, 714, 232, 742,
       712, 231, '32H', 741, 121, 329, 752, '32I', 850, 183], dtype=object)
```

```python
claims_data.isnull().sum()
```

```
LINE_NO                  0
CLAIM_ID                 0
CLAIM_TYPE               0
PROV_NPI             73868
PROV_NAME                0
PROV_CITY                0
PROV_ST                  0
PROV_ZIP                 0
MEMBER_ID                0
MEMBER_ZIP               0
PROV_SPECIALTY      447119
PLACE_OF_SERVICE         0
MSC                      0
PROC_CD              66170
MODIFIER            732659
REVENUE_CD          743237
SERVICE_COUNT            0
UB_BILL_TYPE        743237
DRG_CODE            979652
DIAG1_CD                 0
DIAG2_CD            355537
DIAG3_CD            563242
Eligible Charges         0
sheet_name               0
dtype: int64
```

Patient Stratification

Since we'd like to stratify members by number of services provided and total charges, need to restructure data to format where each row represents one member.

```python
# per member, calculate the number of line items (services), unique claims, and total
member_group = claims_data.groupby('MEMBER_ID').aggregate({'LINE_NO': lambda x: len(x
                                                           'CLAIM_ID': lambda x: len(x.unique()),'El

member_group
```

MEMBER_ID	LINE_NO	CLAIM_ID	Eligible Charges
Removed for PHI	24	15	3079.60
	6	2	213.97
	40	16	3614.70
	10	8	6204.85
	168	52	13715.00
	...	...	...
	1	1	265.00
	1	1	42.13
	2	1	16.95
	5	2	392.00
	1	1	841.00

34985 rows × 3 columns

```python
member_group.corr()
```

	LINE_NO	CLAIM_ID	Eligible Charges
LINE_NO	1.000000	0.888160	0.717901
CLAIM_ID	0.888160	1.000000	0.599019
Eligible Charges	0.717901	0.599019	1.000000

```python
plt.scatter(member_group['LINE_NO'],member_group['CLAIM_ID'])
plt.title('Per member number of line items by number of claims')
plt.xlabel('Number of Line Items')
plt.ylabel('Number of Claims')
plt.show()
```

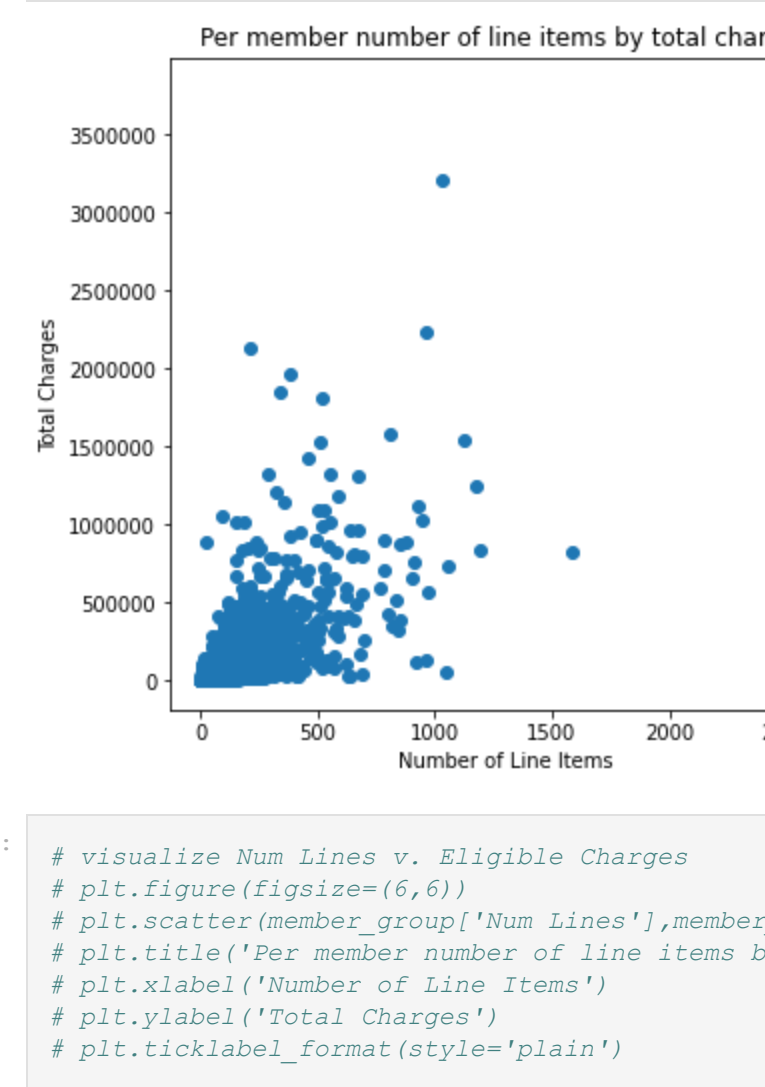
Since our data is at the line (service) level rather than the claim level, and the correlation between number of line items and charges is stronger than that between number of claims and charges, we'll use the LINE_NO feature for our model. Further, the per member number of line items correlates strongly with the per member number of claims, so we likely wouldn't see much difference in clusters if grouping at the claim level.

```python
member_group = claims_data.groupby('MEMBER_ID').aggregate({'LINE_NO': lambda x: len(x
                                                           'Eligible Charges': np.sum
member_group.rename(columns={'LINE_NO':'Num Lines'}, inplace=True)
member_group
```

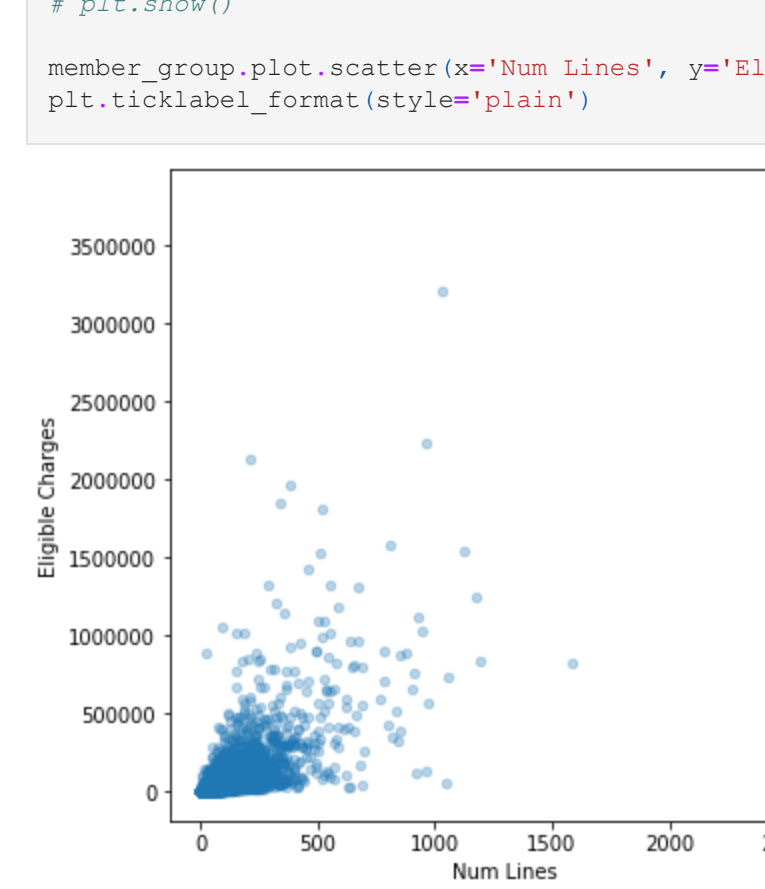
MEMBER_ID	Num Lines	Eligible Charges
Removed for PHI	24	3079.60
	6	213.97
	40	3614.70
	10	6204.85
	168	13715.00
	...	...
	1	265.00
	1	42.13
	2	16.95
	5	392.00
	1	841.00

34985 rows × 2 columns

```python
# visualize Num Lines v. Eligible Charges
plt.figure(figsize=(6,6))
plt.scatter(member_group['Num Lines'],member_group['Eligible Charges'])
plt.title('Per member number of line items by total charges')
plt.xlabel('Number of Line Items')
plt.ylabel('Total Charges')
plt.ticklabel_format(style='plain')

plt.show()
```




```python
# visualize Num Lines v. Eligible Charges
# plt.figure(figsize=(6,6))
# plt.scatter(member_group['Num Lines'],member_group['Eligible Charges'])
# plt.title('Per member number of line items by total charges')
# plt.xlabel('Number of Line Items')
# plt.ylabel('Total Charges')
# plt.ticklabel_format(style='plain')

# plt.show()

member_group.plot.scatter(x='Num Lines', y='Eligible Charges',alpha=0.3,figsize=(6,6)
plt.ticklabel_format(style='plain')
```




```python
member_group.describe(percentiles=[0.05,0.1,0.25,0.5,0.75,0.9,0.95])
```

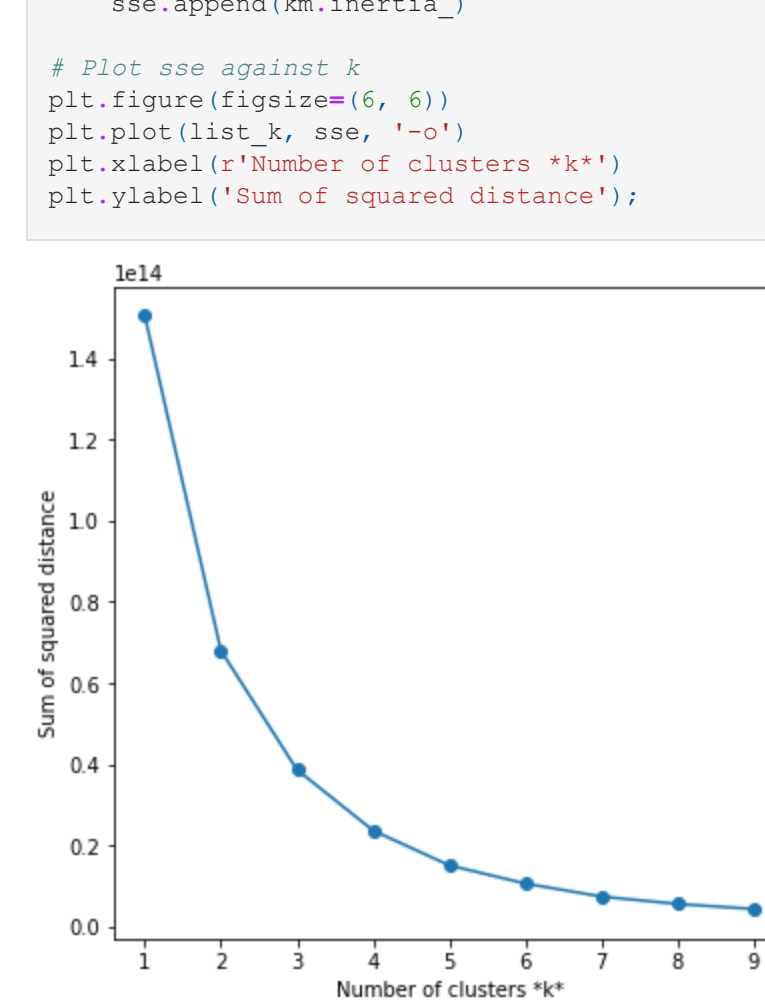
	Num Lines	Eligible Charges
count	34985.000000	3.498500e+04
mean	30.074318	1.414727e+04
std	58.175373	6.556536e+04
min	1.000000	4.150000e+00
5%	1.000000	1.400000e+02
10%	2.000000	2.320000e+02
25%	5.000000	5.450000e+02
50%	12.000000	1.710000e+03
75%	33.000000	7.107380e+03
90%	72.000000	2.628948e+04
95%	112.000000	5.634325e+04
max	2600.000000	3.805456e+06

K-Means Clustering

Let's use the Elbow Method to get an idea of what a good k number of clusters would be, by evaluating the sum of squared distance (SSE) between data points and their assigned clusters' centroids.

```python
from sklearn.cluster import KMeans
```

```python
# Run the Kmeans algorithm and get the index of data points clusters
sse = []
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(member_group[['Num Lines', 'Eligible Charges']])
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance');
```



Looks like k between 3 and 5 would be reasonable choices for number of clusters. Let's run and visualize kmeans models for k= 3, 4 and 5.

```python
# fit kmeans model
def kmeans_model(k):
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(member_group[['Num Lines', 'Eligible Charges']])
    return km
```

```python
km_3 = kmeans_model(3)
km_4 = kmeans_model(4)
km_5 = kmeans_model(5)
```

```python
# add columns with cluster labels to the member_group dataframe
member_group['cluster_3'] = km_3.labels_
member_group['cluster_4'] = km_4.labels_
member_group['cluster_5'] = km_5.labels_
```

```python
member_group.head()
```

MEMBER_ID	Num Lines	Eligible Charges	cluster_3	cluster_4	cluster_5
Removed for PHI	24	3079.60	0	0	0
	6	213.97	0	0	0
	40	3614.70	0	0	0
	10	6204.85	0	0	0
	168	13715.00	0	0	0

```python
member_group['cluster_3'].value_counts(normalize=True)*100
```

```
0    97.887666
1     1.937973
2     0.174360
Name: cluster_3, dtype: float64
```

```python
member_group['cluster_4'].value_counts(normalize=True)*100
```

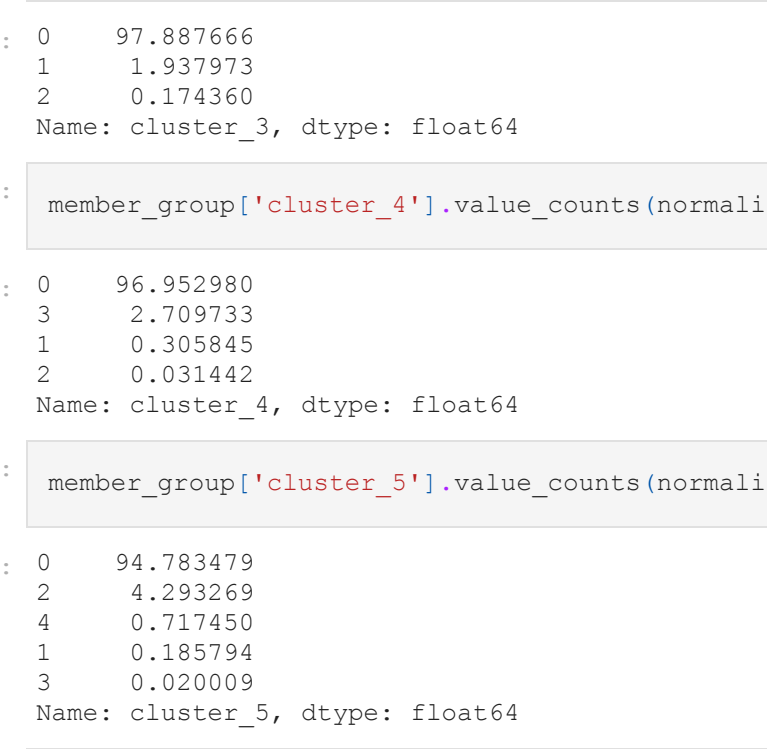
```
0    96.952980
3     2.709733
1     0.305845
2     0.031442
Name: cluster_4, dtype: float64
```

```python
member_group['cluster_5'].value_counts(normalize=True)*100
```

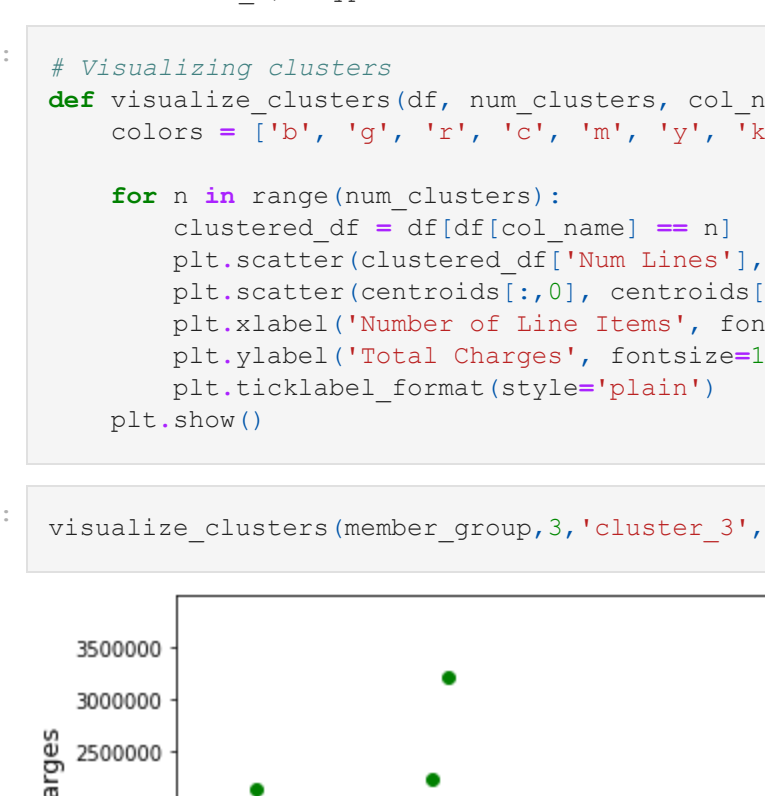
```
0    94.783479
2     4.293269
4     0.717450
1     0.185794
3     0.020009
Name: cluster_5, dtype: float64
```

```python
# Visualizing clusters
def visualize_clusters(df, num_clusters, col_name, centroids):
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

    for n in range(num_clusters):
        clustered_df = df[df[col_name] == n]
        plt.scatter(clustered_df['Num Lines'], clustered_df['Eligible Charges'], c=co
        plt.scatter(centroids[:,0], centroids[:,1], c='y', marker='*', s=100)
        plt.xlabel('Number of Line Items', fontsize=13)
        plt.ylabel('Total Charges', fontsize=13)
        plt.ticklabel_format(style='plain')
    plt.show()
```

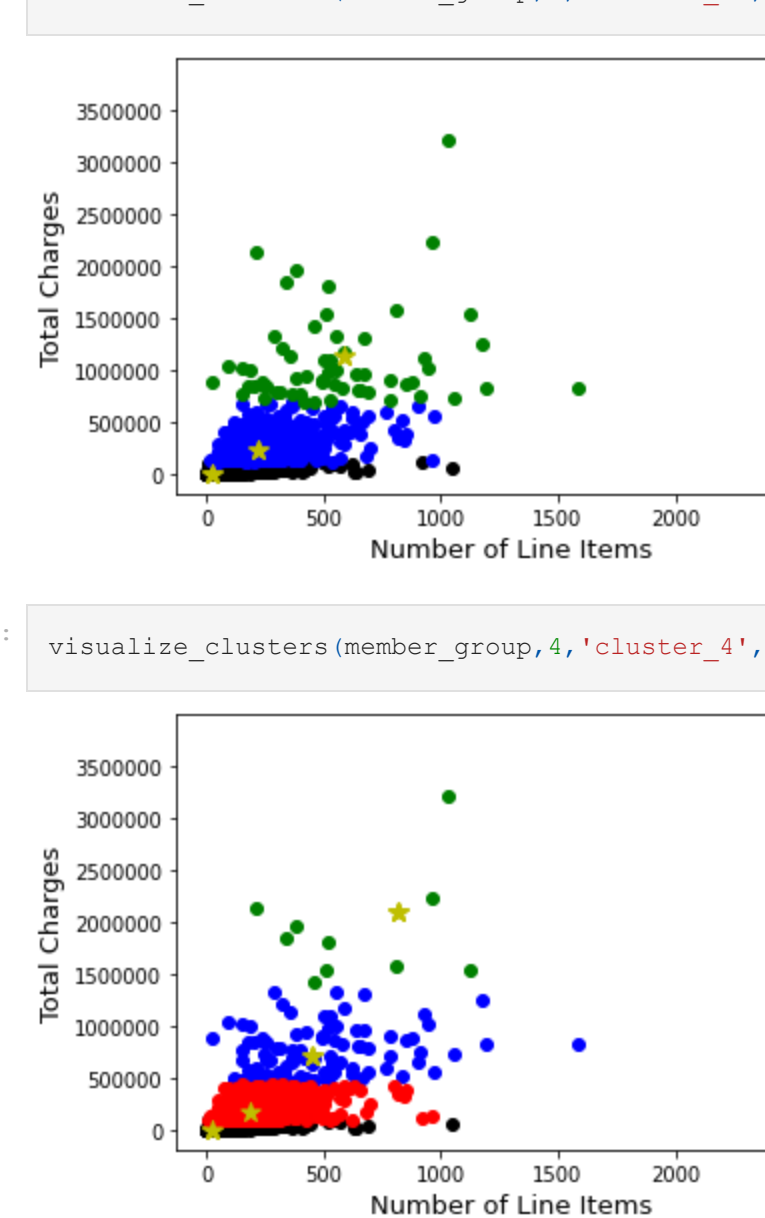
```python
visualize_clusters(member_group,3,'cluster_3',km_3.cluster_centers_)
```




```python
visualize_clusters(member_group,4,'cluster_4',km_4.cluster_centers_)
```




```python
visualize_clusters(member_group,5,'cluster_5',km_5.cluster_centers_)
```




```python
# centroids for each cluster for km_4
np.set_printoptions(suppress=True) # suppress scientific notation
km_4.cluster_centers_
```

```
array([[     24.1620283 ,     6799.96449381],
       [    449.44859813,   718575.37747664],
       [    813.09090909,  2097741.91454545],
       [    185.36325238,   173521.50734952]])
```

Since we'll use the kmeans model with 4 clusters, let's drop the cluster_3 and cluster_5 columns from our df, and rename the column to 'cluster'.

```python
member_group_cluster = member_group.drop(['cluster_3','cluster_5'],axis=1)
member_group_cluster.rename(columns={'cluster_4':'cluster'},inplace=True)
member_group_cluster
```

MEMBER_ID	Num Lines	Eligible Charges	cluster
Removed for PHI	24	3079.60	0
	6	213.97	0
	40	3614.70	0
	10	6204.85	0
	168	13715.00	0
	...	...	...
	1	265.00	0
	1	42.13	0
	2	16.95	0
	5	392.00	0
	1	841.00	0

34985 rows × 3 columns

```python
member_group_cluster['cluster'].value_counts(normalize=True)*100
```

```
0    96.952980
3     2.709733
1     0.305845
2     0.031442
Name: cluster, dtype: float64
```

The clusters generated by the km_4 model don't appear to align with our healthcare-specific expectations of the following groups of patients:

- Low cost, low utilizers
- Low cost, high utilizers
- High cost, low utilizers
- High cost, high utilizers

So, we'll manually stratify our patient population by setting bounds we think are more appropriate.

Manual Stratification

```python
member_group_manual = member_group_cluster.drop('cluster',axis=1)
member_group_manual.head()
```

MEMBER_ID	Num Lines	Eligible Charges
Removed for PHI	24	3079.60
	6	213.97
	40	3614.70
	10	6204.85
	168	13715.00

```python
member_group_manual.describe(percentiles=[.10,.25,.50,.75,.90])
```

	Num Lines	Eligible Charges
count	34985.000000	3.498500e+04
mean	30.074318	1.414727e+04
std	58.175373	6.556536e+04
min	1.000000	4.150000e+00
10%	2.000000	2.320000e+02
25%	5.000000	5.450000e+02
50%	12.000000	1.710000e+03
75%	33.000000	7.107380e+03
90%	72.000000	2.628948e+04
max	2600.000000	3.805456e+06

We'll consider low cost to be less than average ($1,710) and low utilization to be less than average (12)

```python
# low cost, low utilizers (lclu) - 25% for # services and charges
lclu = (member_group_manual['Num Lines']<=12) & (member_group_manual['Eligible Charge

# high cost, low utilizers (hclu) - >75% for charges, 25% for # services
hclu = (member_group_manual['Num Lines']<=12) & (member_group_manual['Eligible Charge

# low cost, high utilizers (lchu) - 25% for charges, 75% for # services
lchu = (member_group_manual['Num Lines']>12) & (member_group_manual['Eligible Charges

# high cost, high utilizers (hchu) - 75% for charges and # services
hchu = (member_group_manual['Num Lines']>12) & (member_group_manual['Eligible Charges
```

Let's check to make sure we capture all members

```python
member_group_manual.shape[0]
```

```
34985
```

```python
member_group_manual[lclu].shape[0] + member_group_manual[hclu].shape[0] + member_grou
```

```
34985
```

```python
member_group_manual.head()
```

MEMBER_ID	Num Lines	Eligible Charges
Removed for PHI	24	3079.60
	6	213.97
	40	3614.70
	10	6204.85
	168	13715.00

```python
# add columns with our group labels
member_group_manual_lclu = member_group_manual[lclu].copy()
member_group_manual_lclu['group'] = 'lclu'

member_group_manual_hclu = member_group_manual[hclu].copy()
member_group_manual_hclu['group'] = 'hclu'

member_group_manual_lchu = member_group_manual[lchu].copy()
```

```
member_group_manual_lchu['group'] = 'lchu'
member_group_manual_hchu = member_group_manual[hchu].copy()
member_group_manual_hchu['group'] = 'hchu'
```

In [51]:
```
# combine group dfs
member_group_manual = pd.concat([member_group_manual_lclu,member_group_manual_hclu,
                                 member_group_manual_lchu,member_group_manual_hchu])
```

In [52]:
```
member_group_manual.head()
```

Out[52]:

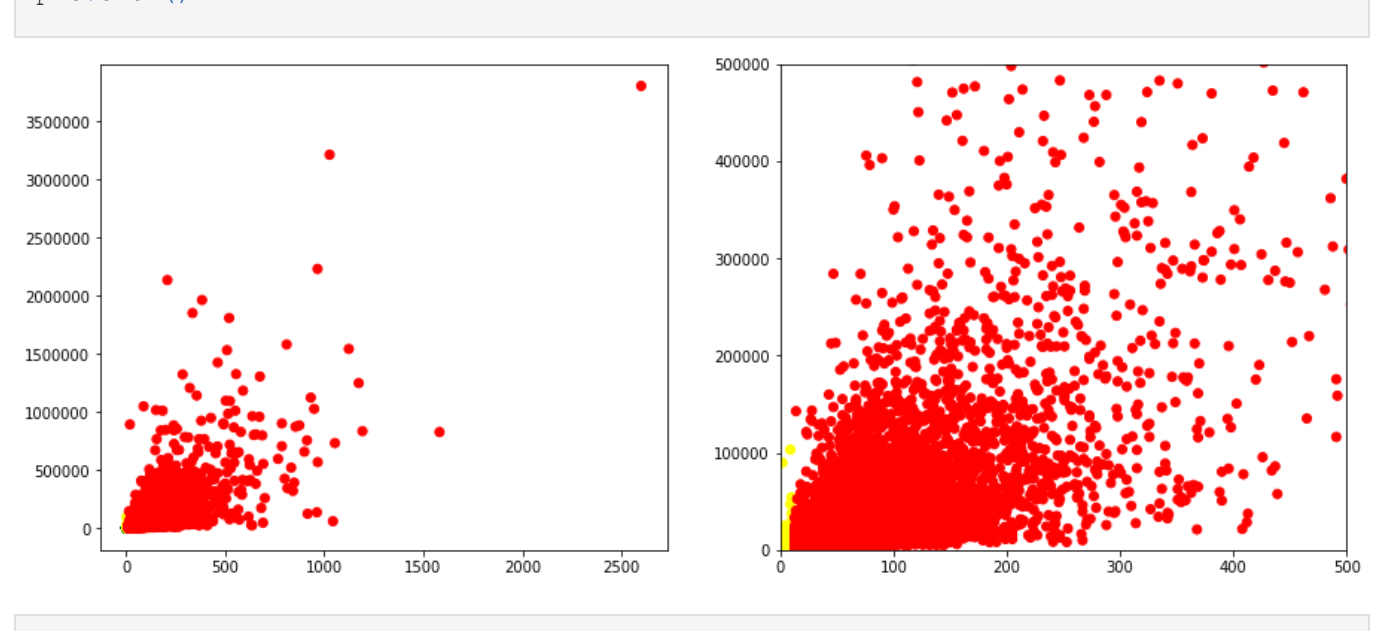
MEMBER_ID	Num Lines	Eligible Charges	group
Removed for PHI	6	213.97	lclu
	5	975.00	lclu
	6	460.00	lclu
	12	1541.96	lclu
	6	708.78	lclu

In [53]:
```
# convert group to category
member_group_manual['group'] = member_group_manual['group'].astype('category')
```

In [53]:
```
member_group_manual.dtypes
```

Out[53]:
```
Num Lines            int64
Eligible Charges   float64
group             category
dtype: object
```

In [54]:
```
# visualize our groups
# fig, ax = plt.subplots(figsize=(6,6))
colors = {'lclu':'green','hclu':'yellow','lchu':'blue','hchu':'red'}
# #member_group_manual.plot.scatter('Num Lines', 'Eligible Charges', c=member_group_ma
# ax.scatter(member_group_manual['Num Lines'], member_group_manual['Eligible Charges']
# ax.ticklabel_format(style='plain')
# plt.show()

fig = plt.figure(figsize=(15,6))

ax1 = fig.add_subplot(121)
ax1.scatter(member_group_manual['Num Lines'], member_group_manual['Eligible Charges'],
ax1.ticklabel_format(style='plain')

ax2 = fig.add_subplot(122)
ax1.scatter(member_group_manual['Num Lines'], member_group_manual['Eligible Charges'],
ax1.axis([0,500,0,500000])

plt.show()
```



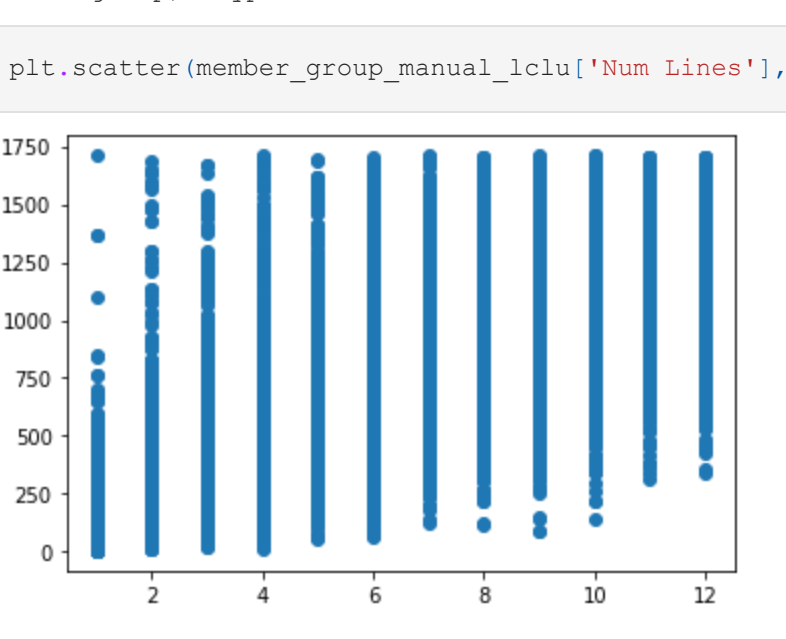
In [55]:
```
member_group_manual['group'].value_counts()
```

Out[55]:
```
lclu    15803
hchu    15617
hclu     1873
lchu     1692
Name: group, dtype: int64
```

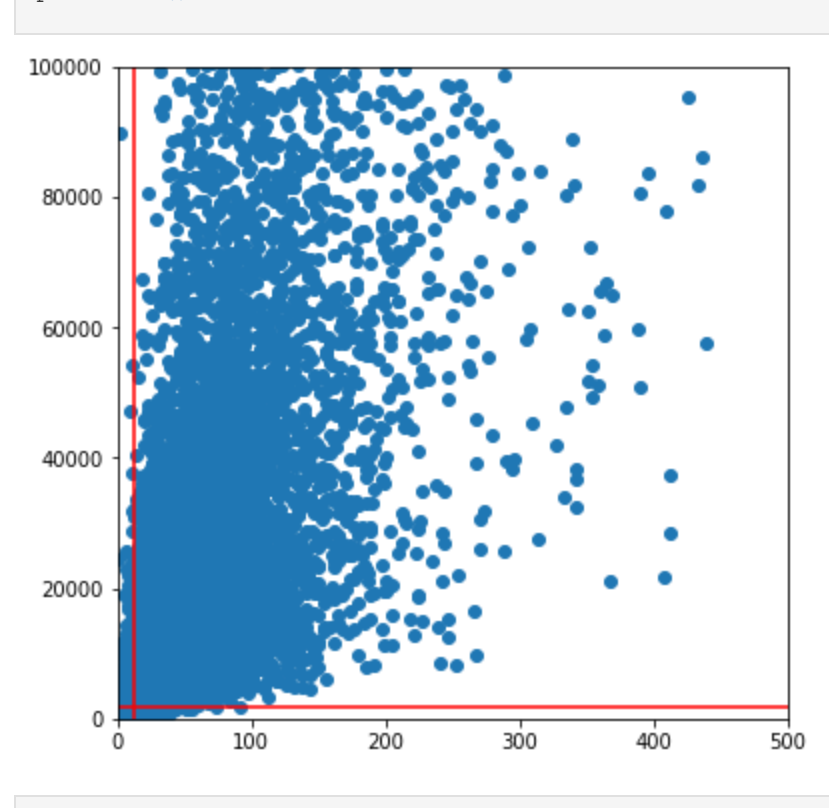
In [56]:
```
member_group_manual['group'].value_counts(normalize=True)*100
```

Out[56]:
```
lclu    45.170787
hchu    44.639131
hclu     5.353723
lchu     4.836358
Name: group, dtype: float64
```

In [57]:
```
plt.scatter(member_group_manual_lclu['Num Lines'], member_group_manual_lclu['Eligible
```



In [58]:
```
plt.figure(figsize=(6,6))
plt.scatter(member_group_manual['Num Lines'], member_group_manual['Eligible Charges'],
plt.axvline(12,c='r')
plt.axhline(1710,c='r')
plt.ticklabel_format(style='plain')
plt.axis([0,500,0,100000])
plt.show()
```



In [59]:
```
# % of members with more than 100 lines
(member_group_manual['Num Lines']>100).sum()/len(member_group_manual)
```

Out[59]:
```
0.06071173360011434
```

In [60]:
```
# % of members with more than 100 lines
(member_group_manual['Eligible Charges']>500000).sum()/len(member_group_manual)
```

Out[60]:
```
0.0027440331570673148
```

Analysis

In [61]:
```
member_group_manual['group'].value_counts()
```

Out[61]:
```
lclu    15803
hchu    15617
hclu     1873
lchu     1692
Name: group, dtype: int64
```

In [62]:
```
member_group_manual
```

Out[62]:

MEMBER_ID	Num Lines	Eligible Charges	group
Removed for PHI	6	213.97	lclu
	5	975.00	lclu
	6	460.00	lclu
	12	1541.96	lclu
	6	708.78	lclu
	...	...	...
	19	3294.72	hchu
	15	2939.92	hchu
	94	34269.96	hchu
	228	164853.91	hchu
	41	8535.04	hchu

34985 rows × 3 columns

In [63]:
```
# get list of members for each group
lclu_members = list(member_group_manual[lclu].index)
hclu_members = list(member_group_manual[hclu].index)
lchu_members = list(member_group_manual[lchu].index)
hchu_members = list(member_group_manual[hchu].index)
print(lclu_members[:10])
print(hclu_members[:10])
print(lchu_members[:10])
print(hchu_members[:10])
```

Removed for PHI

```
<ipython-input-63-68e34214afec>:2: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  lclu_members = list(member_group_manual[lclu].index)
<ipython-input-63-68e34214afec>:3: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  hclu_members = list(member_group_manual[hclu].index)
<ipython-input-63-68e34214afec>:4: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  lchu_members = list(member_group_manual[lchu].index)
<ipython-input-63-68e34214afec>:5: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  hchu_members = list(member_group_manual[hchu].index)
```

In [64]:
```
# create dataframes with line items for each member group
lclu_claims = claims_data[claims_data['MEMBER_ID'].isin(lclu_members)]
hclu_claims = claims_data[claims_data['MEMBER_ID'].isin(hclu_members)]
lchu_claims = claims_data[claims_data['MEMBER_ID'].isin(lchu_members)]
hchu_claims = claims_data[claims_data['MEMBER_ID'].isin(hchu_members)]
```

Find each group's % of total services and total cost

In [65]:
```
# calculate total # line items and charges for the entire patient population
print(claims_data.shape[0])
print(claims_data['Eligible Charges'].sum())
```

```
1052150
494942389.54
```

In [66]:
```
# calculate each groups # line items and charges
groups = [lclu_claims, hclu_claims, lchu_claims, hchu_claims]
group_items = {}
group_charges = {}
for i,group in enumerate(groups):
    group_items[i] = group.shape[0]
    group_charges[i] = group['Eligible Charges'].sum()
```

In [67]:
```
group_items
```

Out[67]:
```
{0: 77203, 1: 17089, 2: 28660, 3: 929198}
```

In [68]:
```
group_charges
```

Out[68]:
```
{0: 9179029.51, 1: 7690570.140000001, 2: 2168797.5, 3: 475903992.39}
```

In [69]:
```
# calculate % of total items
group_items_pct = {}
for key,value in group_items.items():
    group_items_pct[key] = value / claims_data.shape[0]
```

In [70]:
```
# calculate % of total charges
group_charge_pct = {}
for key,value in group_charges.items():
    group_charge_pct[key] = value / claims_data['Eligible Charges'].sum()
```

In [71]:
```
print(group_items_pct)
print(group_charge_pct)
```

```
{0: 0.07337641971201825, 1: 0.016241980706173074, 2: 0.027239462053889656, 3: 0.883142
137527919}
{0: 0.018545652391040904, 1: 0.015538313756369958, 2: 0.004381919079543142, 3: 0.96153
4114773046}
```

Quick dive into High cost, low utilizers - we'd expect charges to reflect either traumatic injuries or serious conditions that require infrequent, but expensive treatment. Much of these costs may be unpreventable.

In [72]:
```
hclu_charges = hclu_claims[['MEMBER_ID','CLAIM_ID','PROC_CD','DRG_CODE','DIAG1_CD','E
hclu_charges
```

Out[72]:

	MEMBER_ID	CLAIM_ID	PROC_CD	DRG_CODE	DIAG1_CD	Eligible Charges
101659	Removed for PHI		J2350	NaN	G35	100842.30
854362			A0435	NaN	S3289XA	60770.00
854361			A0430	NaN	S3289XA	29000.00
784539			J3357	NaN	K5000	24591.54
109711			67108	NaN	H33012	22638.00
...			...	...	...	...
883410			G8907	NaN	H2513	0.01
853852			91300	NaN	Z23	0.01
539825			99000	NaN	R509	0.01
46040			NaN	NaN	T22222A	0.01
761762			G8918	NaN	J3501	0.01

17089 rows × 6 columns

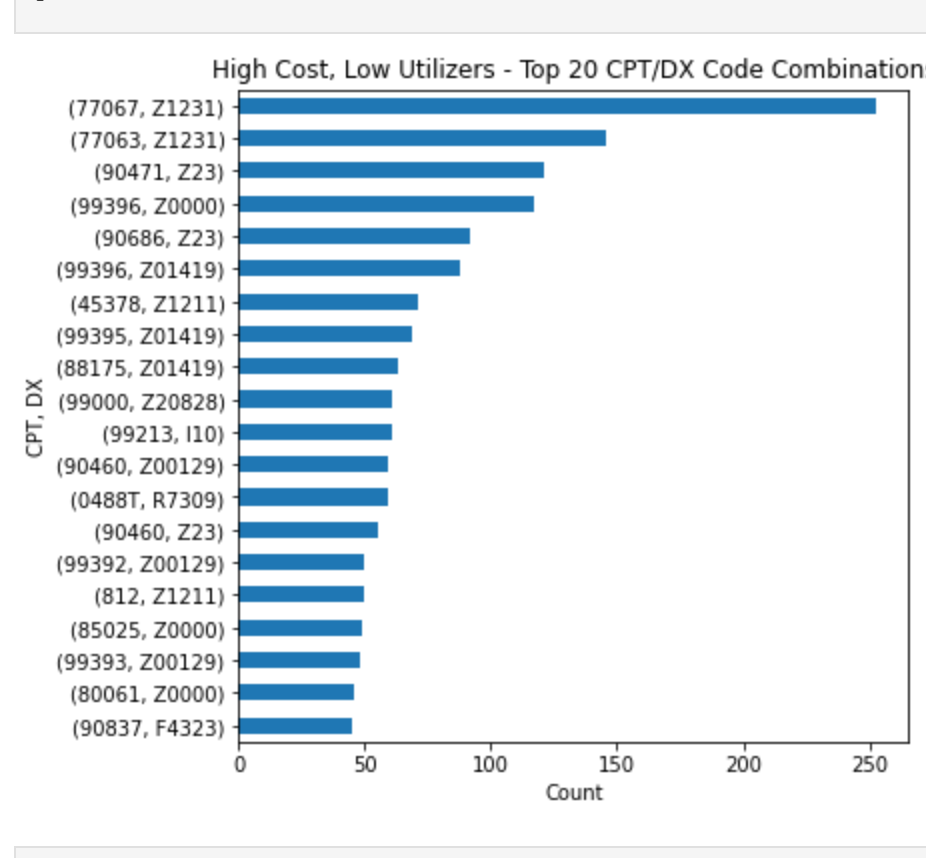
In [73]:
```
# filter for highest charges
hclu_charges.head(20)
```

Out[73]:

	MEMBER_ID	CLAIM_ID	PROC_CD	DRG_CODE	DIAG1_CD	Eligible Charges
101659	Removed for PHI		J2350	NaN	G35	100842.30
854362			A0435	NaN	S3289XA	60770.00
854361			A0430	NaN	S3289XA	29000.00
784539			J3357	NaN	K5000	24591.54
109711			67108	NaN	H33012	22638.00
642819			49587	NaN	K420	21331.00
9278			63030	NaN	M5127	19680.00
13065			58661	NaN	N83201	19468.00
486594			49585	NaN	K429	18283.00
526575			29881	NaN	S83232A	15460.00
702969			29881	NaN	S83232A	15460.00
420892			58562	NaN	T193XXA	15330.00
460557			49505	NaN	K4090	15237.00
545321			49505	NaN	K4090	15237.00
94749			29881	NaN	S83242A	14724.00
377762			49505	NaN	K4090	14511.00
315633			49505	NaN	K4090	14511.00
8434			49505	NaN	K4090	14511.00
44067			25111	NaN	M67432	14151.00
741965			19303	NaN	F640	12460.00

Find the most common CPT/Dx code combinations for Groups: HCLU, LCHU, HCHU

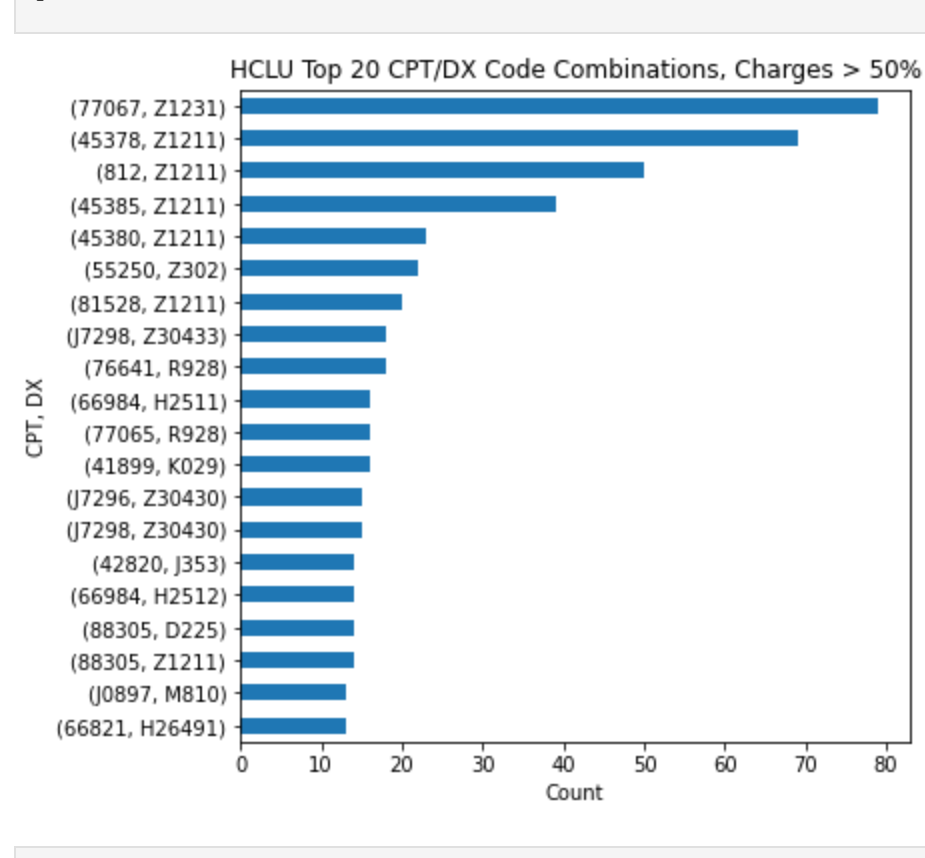
In [74]:
```
# group 2: HCLU
hclu_cpt_dx_combs = hclu_claims.groupby(['PROC_CD','DIAG1_CD']).size().sort_values(as
hclu_cpt_dx_combs.head(20).sort_values(ascending=True).plot.barh(figsize=(6,6))
plt.title('High Cost, Low Utilizers - Top 20 CPT/DX Code Combinations')
plt.xlabel('Count')
plt.ylabel('CPT, DX')
plt.show()
```



In [75]:
```
# for HCLU patients, we're more interested in the highest cost services, not just the
# so let's filter the top code combinations for those with associated average charges

# first, we'll find the average service charge in the hclu claims group
hclu_claims['Eligible Charges'].mean()
```

Out[75]:
```
450.03043712329395
```

In [76]:
```
# then, we'll filter our code combinations
hclu_claims_high_cost = hclu_claims[hclu_claims['Eligible Charges']>450]
hclu_cpt_dx_combs_high_cost = hclu_claims_high_cost.groupby(['PROC_CD','DIAG1_CD']).s
hclu_cpt_dx_combs_high_cost.head(20).sort_values(ascending=True).plot.barh(figsize=(6,
plt.title('HCLU Top 20 CPT/DX Code Combinations, Charges > 50%')
plt.xlabel('Count')
plt.ylabel('CPT, DX')
plt.show()
```



In [77]:
```
# looks like there's high variability with the 77067 charge
cpt_77067_lines = claims_data[claims_data['PROC_CD']=='77067']
cpt_77067_lines['Eligible Charges'].describe()
```

Out[77]:
```
count    5240.000000
mean      290.450855
std       154.283774
min        38.750000
25%       117.000000
50%       261.000000
75%       421.180000
max      1337.510000
Name: Eligible Charges, dtype: float64
```

Indeed, std = 154.28, and the charge can range from 38.75 to 1,337.51. Let's see if there's any correlation with cost of cpt 77067 and other variables.

In [78]:
```
cpt_77067_lines.corr()['Eligible Charges']
```

Out[78]:
```
PROV_NPI             0.061160
PROV_ZIP             0.038546
MEMBER_ZIP           0.008554
PLACE_OF_SERVICE    -0.569307
REVENUE_CD          -0.258780
SERVICE_COUNT        0.031925
DRG_CODE                  NaN
Eligible Charges     1.000000
Name: Eligible Charges, dtype: float64
```

Looks like there's a strong correlation with a few different variables: PROV_NPI, PROV_ZIP, MEMBER_ZIP, and SERVICE_COUNT

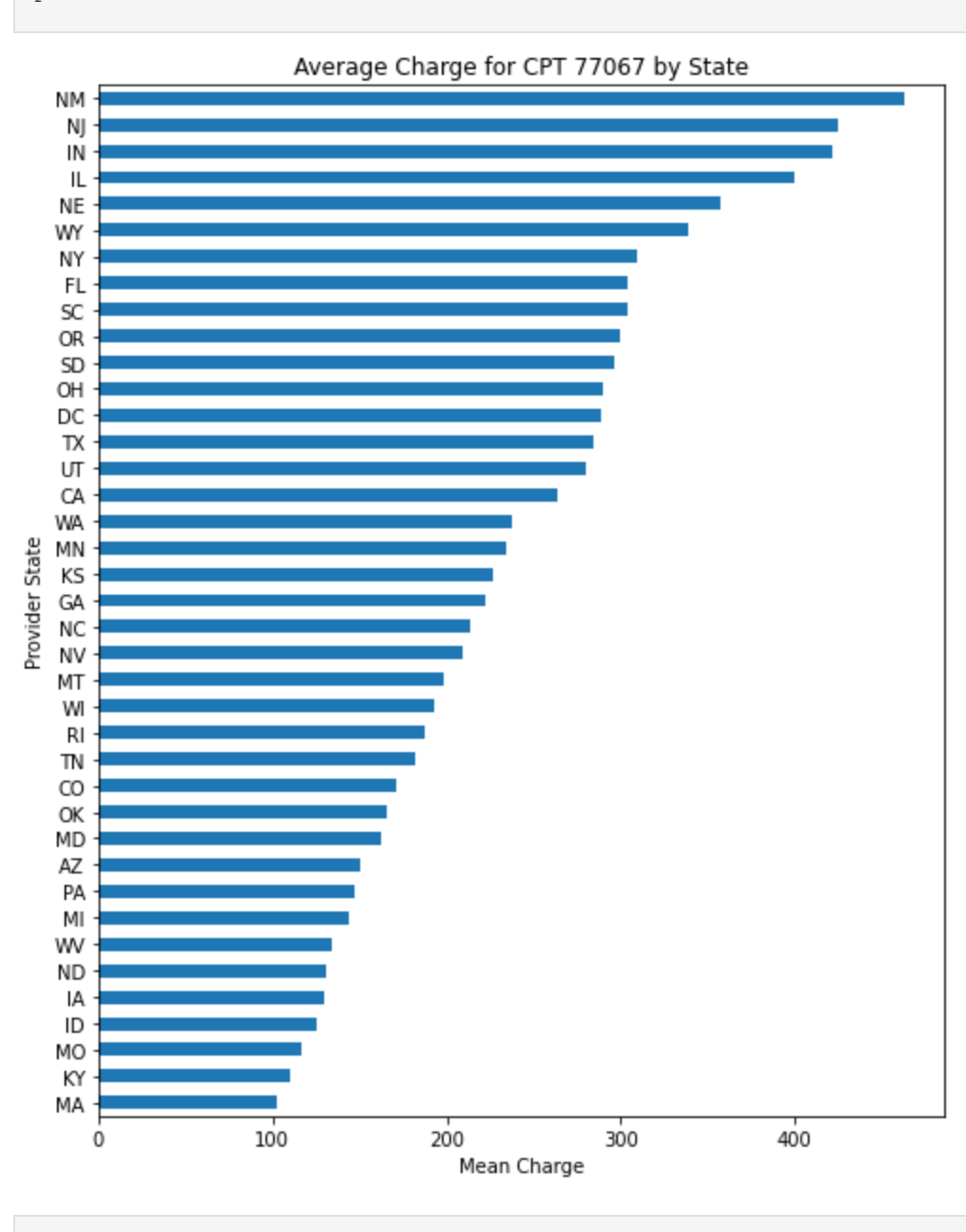
In [79]:
```
len(cpt_77067_lines['PROV_CITY'].unique())
```

Out[79]:
```
198
```

In [80]:
```
cpt_77067_lines.groupby('PROV_CITY')['Eligible Charges'].mean()
```

Out[80]:
```
PROV_CITY
AFTON        122.131579
AKRON        331.250000
ANACORTES    357.000000
ANTHEM       350.000000
ATLANTA      131.000000
                ...
WHEATLAND    239.000000
WINDSOR      390.000000
WORCESTER     94.670000
WORLAND      153.766667
YANKTON      369.500000
Name: Eligible Charges, Length: 198, dtype: float64
```

In [81]:
```
cpt_77067_lines.groupby('PROV_ST')['Eligible Charges'].mean().sort_values().plot.barh
plt.title('Average Charge for CPT 77067 by State')
plt.xlabel('Mean Charge')
plt.ylabel('Provider State')
plt.show()
```



In [87]:
```
cpt_77067_lines.groupby('PROV_NAME')['Eligible Charges'].mean().sort_values(ascending=
```

Out[87]:

PROV_NAME	
Removed for Sensitive Info	1257.610000
	1166.040000
	1132.960000
	1070.886667
	1060.900000
	...
	71.000000
	70.000000
	60.000000
	53.000000
	49.000000

Name: Eligible Charges, Length: 377, dtype: float64

In [122…
```
# filter for providers in CO and WY
cpt_77067_lines_WY_CO = cpt_77067_lines[(cpt_77067_lines['PROV_ST']=='WY') | (cpt_770
cpt_77067_lines_WY_CO.groupby('PROV_NAME')['Eligible Charges'].mean().sort_values(asc
```

Out[122…

PROV_NAME	
Removed for Sensitive Info	1257.610000
	1132.960000
	1070.886667
	1060.900000
	752.700000
	...
	96.000000
	94.425000
	78.500000
	70.000000
	53.000000

Name: Eligible Charges, Length: 131, dtype: float64

In [127…
```
cpt_77067_lines_WY_CO.corr()['Eligible Charges']
```

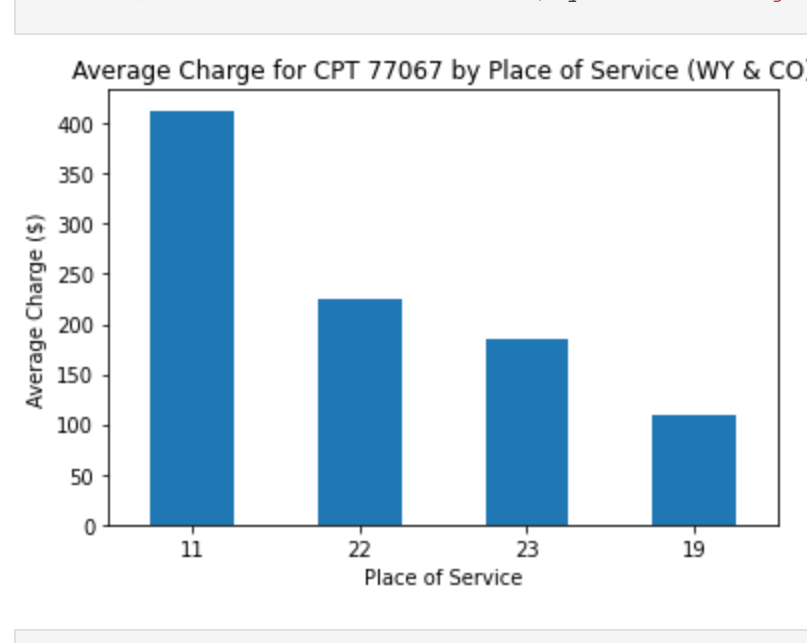
Out[127…
```
PROV_NPI             0.159068
PROV_ZIP             0.389693
MEMBER_ZIP          -0.000472
PLACE_OF_SERVICE    -0.608372
REVENUE_CD          -0.271815
SERVICE_COUNT        0.018542
DRG_CODE                  NaN
Eligible Charges     1.000000
Name: Eligible Charges, dtype: float64
```

In [128…
```
cpt_77067_lines_WY_CO.groupby('PLACE_OF_SERVICE').mean()
```

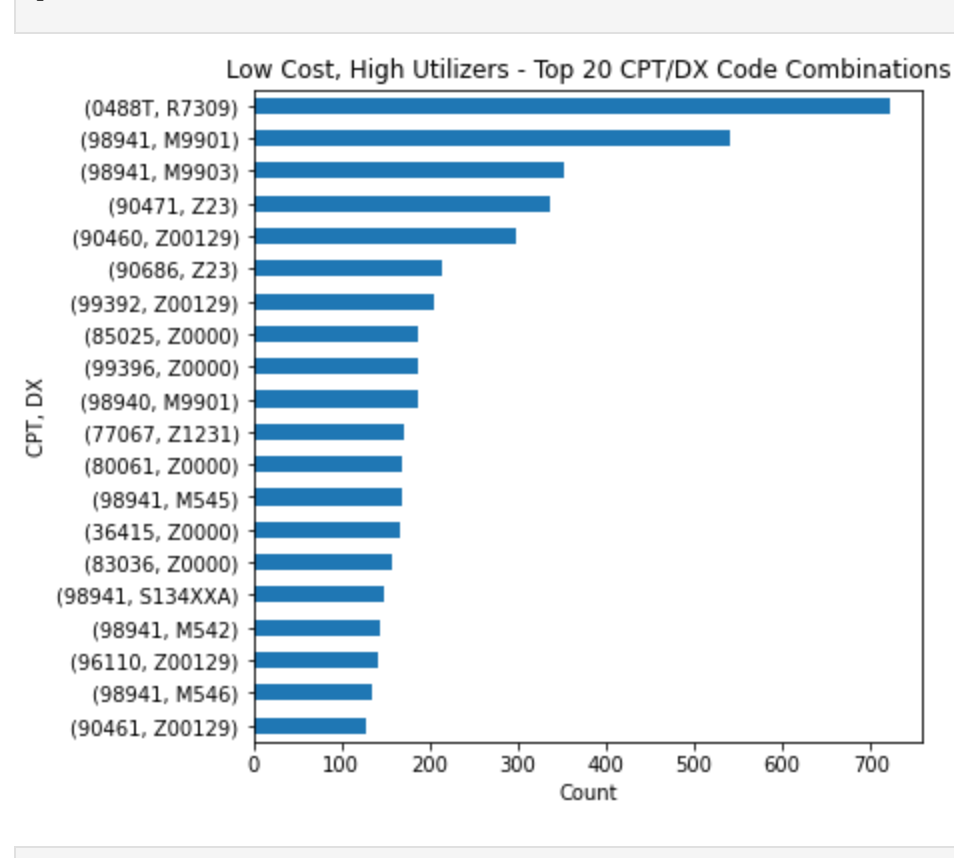
Out[128…

PLACE_OF_SERVICE	PROV_NPI	PROV_ZIP	MEMBER_ZIP	REVENUE_CD	SERVICE_COUNT	DRG_CODE
11	Removed for Sensitive Info			NaN	0.995402	NaN
19				NaN	0.985294	NaN
22				418.399596	0.995274	NaN
23				NaN	1.000000	NaN

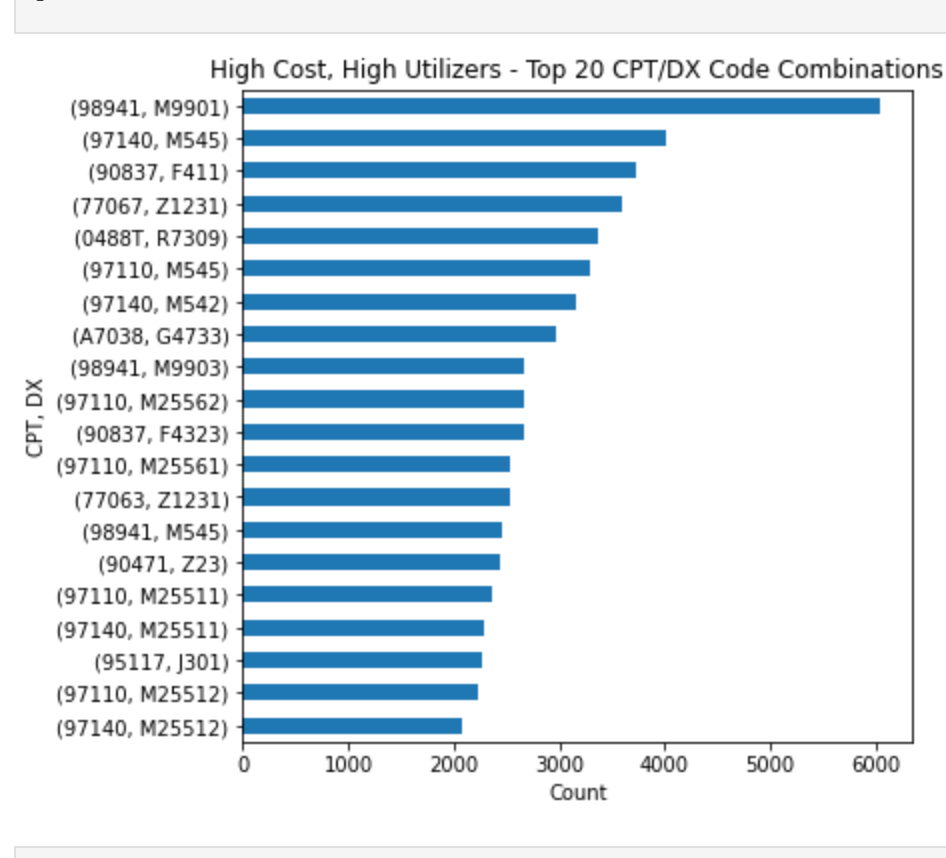
In [135…
```
ax = cpt_77067_lines_WY_CO.groupby('PLACE_OF_SERVICE').mean()['Eligible Charges'].sor
ax.set(xlabel='Place of Service', ylabel='Average Charge ($)', title='Average Charge
```



In [82]:
```
# group 3: LCHU
lchu_cpt_dx_combs = lchu_claims.groupby(['PROC_CD','DIAG1_CD']).size().sort_values(as
lchu_cpt_dx_combs.head(20).sort_values(ascending=True).plot.barh(figsize=(6,6))
plt.title('Low Cost, High Utilizers - Top 20 CPT/DX Code Combinations')
plt.xlabel('Count')
plt.ylabel('CPT, DX')
plt.show()
```



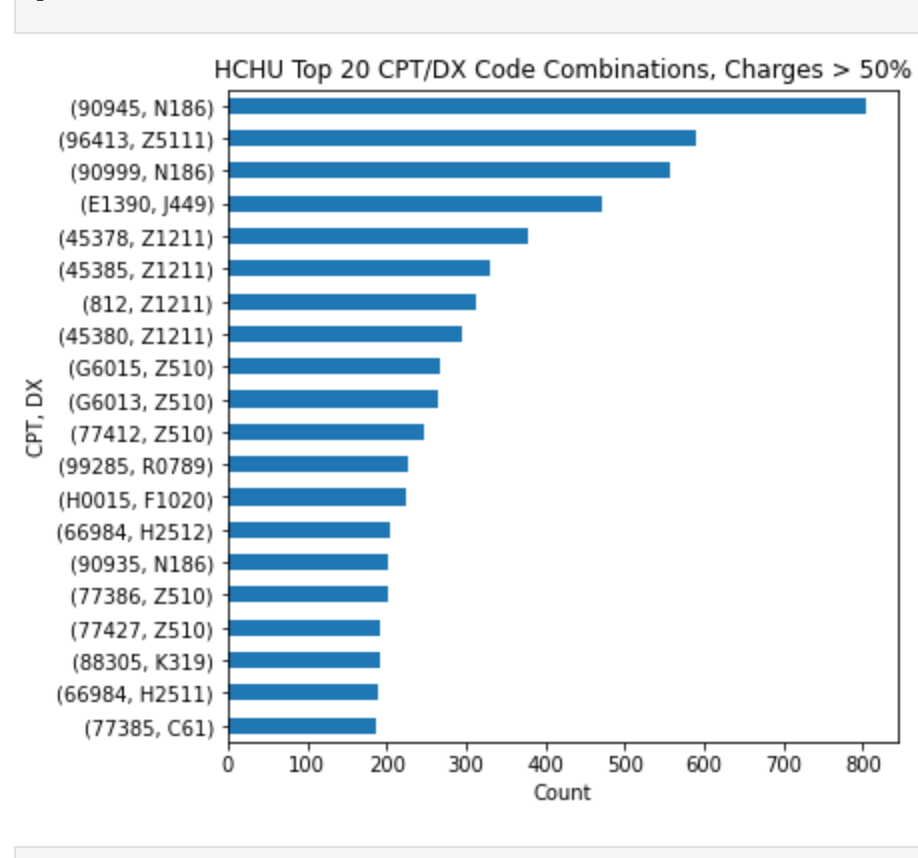
In [83]:
```
# group 3: HCHU
hchu_cpt_dx_combs = hchu_claims.groupby(['PROC_CD','DIAG1_CD']).size().sort_values(as
hchu_cpt_dx_combs.head(20).sort_values(ascending=True).plot.barh(figsize=(6,6))
plt.title('High Cost, High Utilizers - Top 20 CPT/DX Code Combinations')
plt.xlabel('Count')
plt.ylabel('CPT, DX')
plt.show()
```



In [84]:
```
hchu_claims['Eligible Charges'].mean()
```

Out[84]:
```
512.1663976784699
```

In [85]:
```
hchu_claims_high_cost = hchu_claims[hchu_claims['Eligible Charges']>512]
hchu_cpt_dx_combs_high_cost = hchu_claims_high_cost.groupby(['PROC_CD','DIAG1_CD']).s
hchu_cpt_dx_combs_high_cost.head(20).sort_values(ascending=True).plot.barh(figsize=(6,
plt.title('HCHU Top 20 CPT/DX Code Combinations, Charges > 50%')
plt.xlabel('Count')
plt.ylabel('CPT, DX')
plt.show()
```



In [90]:
```
cpt_98941_lines = claims_data[claims_data['PROC_CD']=='98941']
cpt_98941_lines['Eligible Charges'].describe()
```

Out[90]:
```
count    25829.000000
mean        58.718903
std         14.483718
min          5.000000
25%         50.000000
50%         55.000000
75%         68.000000
max        989.410000
Name: Eligible Charges, dtype: float64
```

In [94]:
```
cpt_98941_lines.corr()['Eligible Charges']
```

Out[94]:
```
PROV_NPI            -0.026720
PROV_ZIP             0.013966
MEMBER_ZIP           0.041024
PLACE_OF_SERVICE    -0.011998
REVENUE_CD                NaN
SERVICE_COUNT        0.016363
DRG_CODE             0.751820
Eligible Charges     1.000000
Name: Eligible Charges, dtype: float64
```

In [93]:
```
cpt_98941_lines.groupby('PROV_NAME')['Eligible Charges'].mean().sort_values(ascending=
```

Out[93]:

PROV_NAME	
Removed for Sensitive Info	149.664430
	126.390000
	126.390000
	95.833333
	89.290000
	...
	36.000000
	35.000000
	30.000000
	27.000000
	11.630000

Name: Eligible Charges, Length: 258, dtype: float64

In [100…
```
cpt_98941_lines.groupby('DRG_CODE')['Eligible Charges'].mean().sort_values(ascending=F
```

Out[100…
```
DRG_CODE
897.0    126.39
807.0     68.00
470.0     58.79
Name: Eligible Charges, dtype: float64
```

In [118…
```
ax = cpt_98941_lines.groupby('DRG_CODE')['Eligible Charges'].mean().sort_values(ascend
ax.set(xlabel='DRG Code', ylabel='Average Charge ($)', title='Average Charge for CPT 9
ax.text(-0.15,129,'126.39')
ax.text(0.85,70,'68.00')
ax.text(1.85,61,'58.79')
ax.set_ylim([0,140]);
```