Load Everything Here

[ ] ↳ *3 cells hidden*

Read Data

```
df = pd.read_csv("Edgelist2019_2.csv")
```

```
df
```

	Source	Target	Weight
0	f419	l03115	0.0212
1	f419	l03116	0.0224
2	f419	e039	0.1444
3	f419	e11649	0.0124
4	f419	e8342	0.0645
...	...	...	...
28920	c6921	c6922	0.7194
28921	t23342a	t23341a	0.6923
28922	t23342a	t23351a	0.5488
28923	t23342a	t23362a	0.7402
28924	t23341a	t23351a	0.7546

28925 rows × 3 columns

Convert to Graph and Visualize

graph conversion & info

```
%%time
graph=nx.convert_matrix.from_pandas_edgelist(df,source='Source', target='Target', edge_attr=N
graph.name = "Covid DisNet for Edgelist2019_2"
print(nx.info(graph))
print("-----------------------------------")
```

```
Name: Covid DisNet for Edgelist2019_2
Type: Graph
Number of nodes: 2075
Number of edges: 28925
Average degree:  27.8795
-----------------------------------
CPU times: user 70.4 ms, sys: 0 ns, total: 70.4 ms
Wall time: 71.9 ms
```

```
degree_centrality = nx.algorithms.centrality.degree_centrality(graph)
first10pairs = {k: degree_centrality[k] for k in sorted(degree_centrality.keys())[:10]}
first10pairs
```

```
{'a0472': 0.029411764705882353,
 'a0839': 0.00048216007714561236,
 'a084': 0.00048216007714561236,
 'a403': 0.00048216007714561236,
 'a4101': 0.0009643201542912247,
 'a4102': 0.0009643201542912247,
 'a4151': 0.00819672131147541,
 'a4159': 0.0014464802314368371,
 'a4181': 0.00048216007714561236,
 'a419': 0.13211186113789777}
```

```
eigenvector_centrality = nx.algorithms.centrality.eigenvector_centrality_numpy(graph)
first10pairs = {k: eigenvector_centrality[k] for k in sorted(eigenvector_centrality.keys())[:
first10pairs
```

```
{'a0472': 0.02891100865845509,
 'a0839': -5.5534461071812384e-18,
 'a084': 0.0005382745559326981,
 'a403': -4.0986084256558263e-19,
 'a4101': 0.0003591042805007767,
 'a4102': 0.0003591042805007741,
 'a4151': 0.005965411861105286,
 'a4159': 0.0007979795021039903,
 'a4181': 5.059927347811221e-07,
 'a419': 0.08380720406444628}
```

```
katz_centrality = nx.algorithms.centrality.katz_centrality_numpy(graph)
first10pairs = {k: katz_centrality[k] for k in sorted(katz_centrality.keys())[:10]}
first10pairs
```

```
{'a0472': 0.01463210426939267,
 'a0839': 0.00850252717177153,
 'a084': 0.009158457312284665,
 'a403': 0.00850252717177153,
 'a4101': 0.011119140974806818,
 'a4102': 0.011119140974806813,
 'a4151': 0.008266137920715882,
 'a4159': 0.013698653581779189,
```

```
 'a4181': 0.00863492411871393,
 'a419': 0.02062314791167209}
```

```
number_of_triangles = sum(nx.triangles(graph).values()) / 3
number_of_triangles
```

```
791705.0
```

```
nx.algorithms.cluster.transitivity(graph)
```

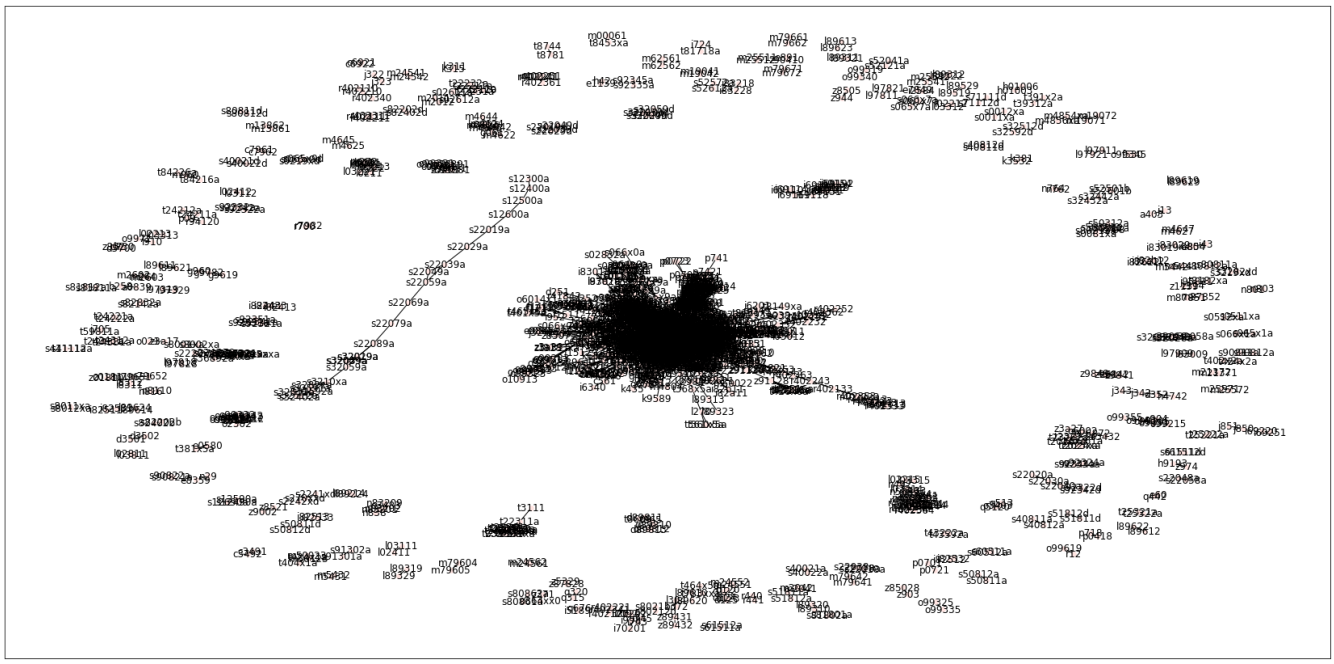
```
0.49345707598767147
```

```
print(nx.average_clustering(graph))
```

```
0.4677968262043043
```

whole graph plot

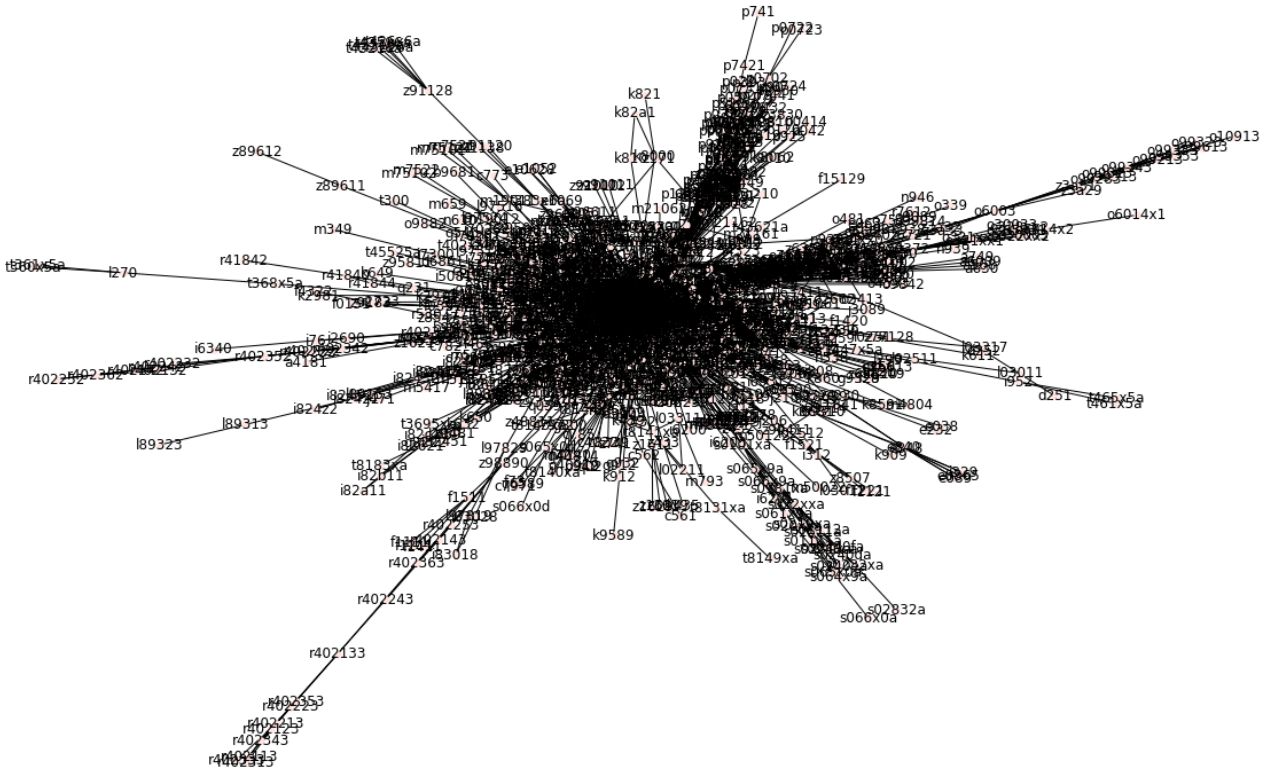
```
%%time
nx.draw_networkx(graph,
                 #pos,
                 with_labels=True,
                 node_size=30,
                 node_color="mistyrose",
                 #edgelist=edges,
                 #edge_color=weights,
                 edge_cmap=plt.cm.Accent,
                 style="solid",
                 width=1)
nx.draw_networkx(graph.subgraph('z20828'),  font_size=16,node_size=120, node_color='red')
plt.subplots_adjust(left=1, bottom=3.2, right=4.8, top=6)
plt.show()

print("----------------------------------------")
print("Density:",nx.classes.function.density(graph))
print("----------------------------------------")
```



```
----------------------------------------
Density: 0.013442390584517433
----------------------------------------
CPU times: user 28.7 s, sys: 579 ms, total: 29.3 s
Wall time: 29 s
```

partial graph plot

```
%%time
plt.figure(figsize=(16, 10))
gcc = max(nx.connected_components(graph), key=lambda x: len(x))
H = graph.subgraph(gcc)
nx.draw(H, node_size=30, node_color='mistyrose',with_labels=True,edge_cmap=plt.cm.Accent,styl
plt.subplots_adjust(left=1, bottom=3.2, right=4.8, top=6)
plt.show()
print("Density:",nx.classes.function.density(H))
print("----------------------------------------")
```




```
Density: 0.024968072360538753
----------------------------------------
CPU times: user 19.1 s, sys: 431 ms, total: 19.5 s
Wall time: 19.6 s
```

plot for z20828's neighbors

```
%%time
plt.figure(figsize=(16, 10))
Sub = nx.classes.function.induced_subgraph(graph,set(graph.neighbors(n="z20828")))
nx.draw_networkx(Sub,  font_size=16,node_size=120, node_color='red')
print("----------------------------------------")
print("Density:",nx.classes.function.density(Sub))
print("----------------------------------------")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/networkx/classes/graph.py in neighbors(self, n)
   1237         try:
-> 1238             return iter(self._adj[n])
   1239         except KeyError as e:

KeyError: 'z20828'

The above exception was the direct cause of the following exception:

NetworkXError                             Traceback (most recent call last)
```

4 frames

```
<decorator-gen-53> in time(self, line, cell, local_ns)

<timed exec> in <module>()

-> 1240             raise NetworkXError(f"The node {n} is not in the graph.") from e
```

Fit node2vec

```
vector_size = round(df.shape[0]**0.25)
vector_size
```

```
13
```

```
%%time
setup = Node2Vec(graph,dimensions=vector_size, walk_length=5, num_walks=5)
model = setup.fit(window=10, min_count=1)
print("-------------------------------------")
```

```
Computing transition probabilities: 100%  2075/2075 [00:44<00:00, 46.52it/s]
Generating walks (CPU: 1):   0%|          | 0/5 [00:00<?, ?it/s]
Generating walks (CPU: 1): 100%|██████████| 5/5 [00:02<00:00,  1.91it/s]
-------------------------------------
CPU times: user 48 s, sys: 484 ms, total: 48.5 s
Wall time: 48.9 s
```

```
%%time
#vocab, vectors = model.wv.key_to_index, model.wv.get_normed_vectors()
vocab, vectors = model.wv.vocab, model.wv.vectors

# get node name and embedding vector index.
name_index = np.array([(v[0], v[1].index) for v in vocab.items()]) #.index

# init dataframe using embedding vectors and set index as node name
node2vec_output = pd.DataFrame(vectors[name_index[:,1].astype(int)])
node2vec_output.index = name_index[:,0]
```

```
CPU times: user 5.17 ms, sys: 971 µs, total: 6.14 ms
Wall time: 5.74 ms
```

```
node2vec_output.shape
```

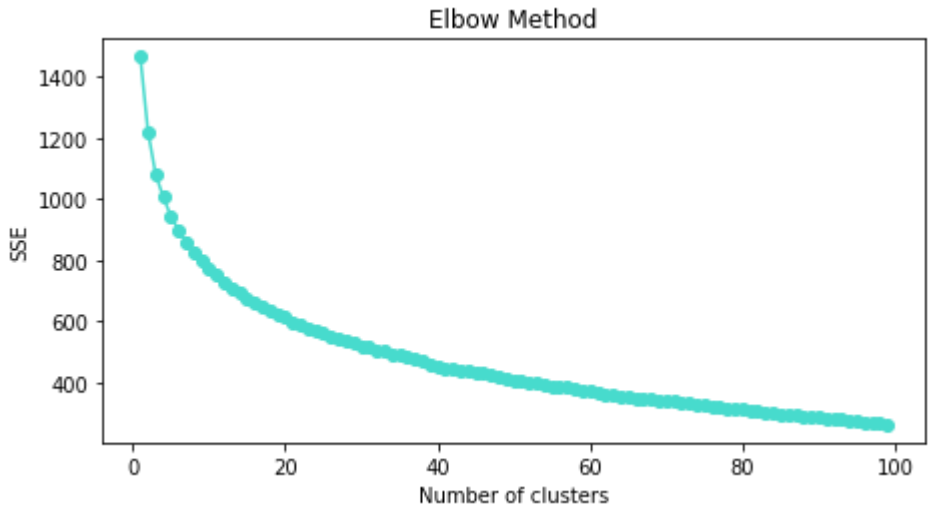
```
(2075, 13)
```

```
model.wv.most_similar("z20828",topn=10)
```

K-means

Find k

```
%%time
SSE = []
for i in range(1,100):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=100, n_init=50, random_state=42)
    kmeans.fit(node2vec_output)
    SSE.append(kmeans.inertia_)
plt.plot(range(1,100), SSE,"o-",color="#47DBCD")
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.subplots_adjust(left=0.25, bottom=0.8, right=1.2, top=1.5)
plt.show()
```




```
CPU times: user 7min 39s, sys: 5min 51s, total: 13min 30s
Wall time: 7min 8s
```

plot k-means clustering

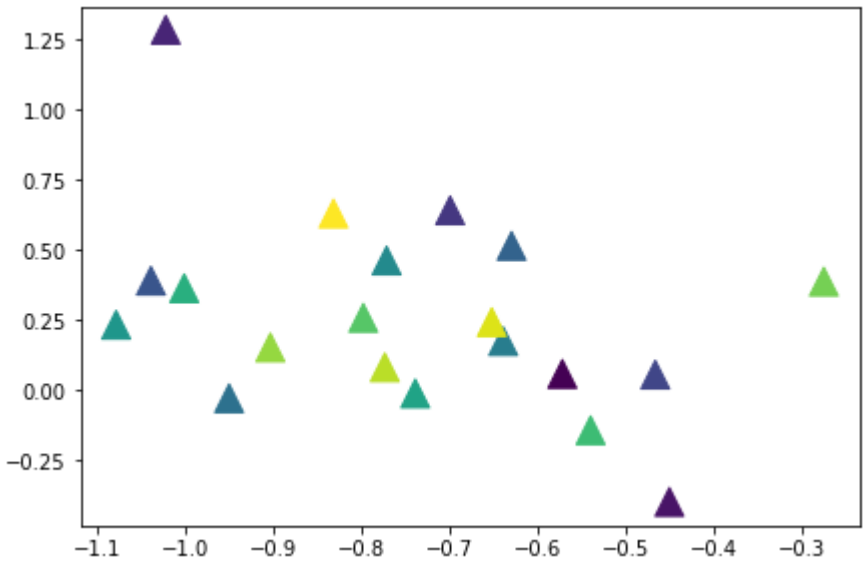
```
n clusters=kmeans.n iter
```

```
kmeans = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=1000, n_init=50, random_sta
```

```
kmeans.fit(node2vec_output)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,
       n_clusters=20, n_init=50, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

```
t = np.arange(n_clusters)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=200, c=t,marker="^'
plt.subplots_adjust(left=0.1, bottom=0.1, right=1, top=1)
```




```
subsample=[]
for i in range(kmeans.n_clusters):
  temp = []
  temp=node2vec_output.iloc[kmeans.labels_==i,:]
  subsample.append(temp)
```

```
for list in range(len(subsample)):
  print("Group",list+1)
  print(subsample[list])
  print("----------------------------------------------------------------")
```

```
[269 rows x 13 columns]
----------------------------------------------------------------
Group 20
                0         1         2   ...        10        11        12
l89310  -0.993056  0.734035 -0.343363  ... -0.365639  0.012149  0.749330
l89320  -1.015382  0.740550 -0.348918  ... -0.330585  0.026440  0.759516
s32059a -1.191844  0.629160 -0.037903  ... -0.271478  0.420447  0.703685
t391x2a -0.742426  0.455953 -0.467846  ... -0.174823  0.495758  0.810865
t39312a -0.691739  0.492456 -0.419451  ... -0.195004  0.468967  0.736829
s32029a -1.055378  0.560430 -0.100037  ... -0.273258  0.442441  0.591646
```

```
s50812d -0.577900  0.797166 -0.047643  ... -0.311124  0.413424  0.663857
s50811d -0.565609  0.818055 -0.178556  ... -0.359861  0.379919  0.584093
s32049a -0.969280  0.455477 -0.049946  ... -0.293215  0.240875  0.495105
s32039a -0.989405  0.452811 -0.115008  ... -0.333014  0.362007  0.544861
s32019a -1.066514  0.608291 -0.185056  ... -0.304432  0.406221  0.556716
s22089a -0.931121  0.472039 -0.241528  ... -0.252095  0.286668  0.464168
s92322a -1.155321  0.710388 -0.406186  ... -0.201485  0.041104  0.026009
s92342a -1.202957  0.748560 -0.452954  ... -0.146870  0.119967  0.017911
s92332a -1.179161  0.736252 -0.464957  ... -0.155866  0.115902  0.023813
s50311a -0.628116  1.004994 -0.240101  ... -0.042853  0.346823  0.361430
s80212a -0.547678  0.744888 -0.193784  ... -0.023248  0.243988  0.441745
s22039a -0.765272  0.391081 -0.363202  ... -0.153255 -0.145579  0.388967
s22059a -0.763003  0.468817 -0.396569  ... -0.161155 -0.140995  0.361825
s22069a -0.818684  0.345560 -0.231986  ... -0.134685  0.094852  0.415746
s22079a -0.758011  0.408257 -0.255678  ... -0.135622  0.176681  0.477926
c7962   -0.465184  0.314882 -0.283938  ... -0.235773  0.285863  0.640216
c7961   -0.426870  0.311334 -0.234318  ... -0.302480  0.332442  0.670176
h4742   -0.948616  0.726801 -0.271734  ... -0.460077  0.239947  0.303764
d352    -1.352407  1.220397 -0.595024  ... -0.714845  0.402737  0.286710
j342    -1.350485  1.205491 -0.542173  ... -0.708781  0.458047  0.247367
j343    -0.925696  0.713223 -0.297658  ... -0.500147  0.232816  0.267582
s22049a -0.764399  0.358083 -0.271678  ... -0.148444 -0.089374  0.385005
c7b8    -0.441223  0.718800 -0.332277  ...  0.045817 -0.002821  0.772620
c7a8    -0.454384  0.675361 -0.292155  ...  0.040029  0.099672  0.765976
b372    -0.765097  0.343687 -0.824926  ... -0.311394  0.277953  0.591798
l304    -0.793705  0.339783 -0.800079  ... -0.196104  0.305985  0.488225
l89529  -1.048253 -0.017454 -0.629358  ... -0.199440  0.564745 -0.084565
t59811a -1.185262  0.675923 -0.470203  ... -0.702325  0.010078  0.223276
j705    -1.204738  0.665935 -0.501996  ... -0.761729  0.039683  0.302483
s82202b -0.589536  0.739573 -0.216320  ... -0.019602  0.646297  0.411887
s82402b -0.602419  0.744510 -0.157587  ...  0.009142  0.659131  0.475003
s80211a -0.574273  1.004498 -0.267113  ... -0.014799  0.369407  0.415824
s0081xa -0.562279  0.599095 -0.152111  ... -0.075801  0.189345  0.441407
m19072  -1.036905  0.910544 -0.037598  ...  0.282082  0.247060  0.582505

m19071  -0.968061  0.908698 -0.112139  ...  0.287499  0.301793  0.659054
s12490a -0.811155  0.600155 -0.532949  ... -0.411080  0.435647 -0.019122
s12590a -0.939699  0.849238 -0.825214  ... -0.574542  0.634803 -0.264070
s12690a -0.804431  0.603982 -0.499284  ... -0.414211  0.414069 -0.023284
n803    -0.941909  0.453790 -0.759537  ... -0.058291  0.471336  0.539394
n801    -0.982363  0.421095 -0.755633  ...  0.008651  0.458387  0.633568
s22030a -0.517664  0.980204 -0.998884  ... -0.607953  0.304184  0.172500
s22020a -0.495499  0.556402 -0.541966  ... -0.447460  0.183472  0.301029
s22040a -0.536504  0.684664 -0.770988  ... -0.501595  0.219759  0.262707
r402111 -0.739676  0.570173 -0.630862  ... -0.097161  0.072572 -0.125812
r402311 -0.669979  0.547992 -0.582326  ... -0.163980  0.110357 -0.064199
r402211 -0.694935  0.546233 -0.570987  ... -0.131386  0.136321 -0.085218

[52 rows x 13 columns]
------------------------------------------------------------
```

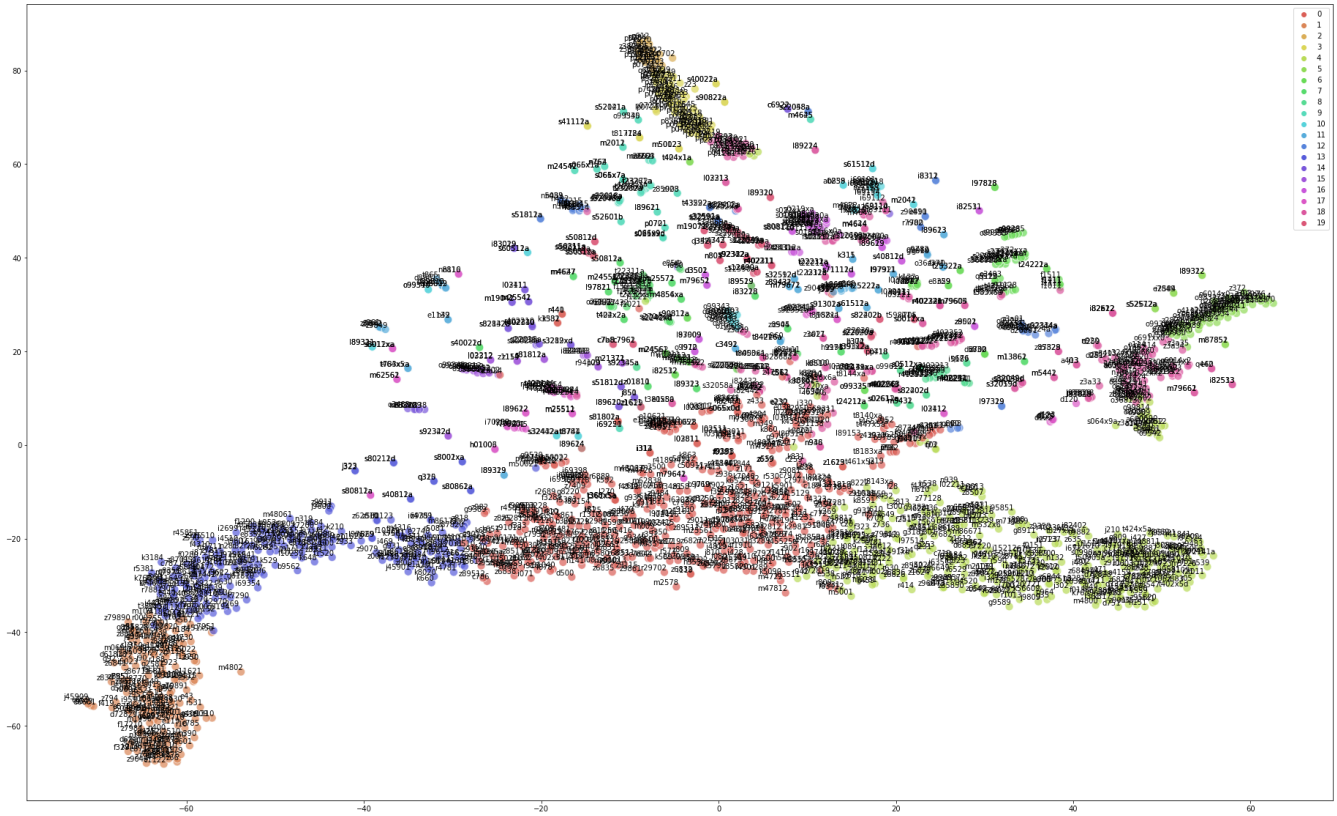
T-SNE

```
def tsne_plot(model):
```

```
    "Creates and TSNE model and plots it"
    labels = []
    tokens = []

    for word in model.wv.vocab:
        tokens.append(model[word])
        labels.append(word)

    tsne_model = TSNE(perplexity=30, n_components=2, learning_rate=10, init='random', n_iter=
    new_values = tsne_model.fit_transform(tokens)

    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    plt.figure(figsize=(32, 20))
    sns.scatterplot(
        x=x, y=y,
        hue= kmeans.labels_,
        palette=sns.color_palette("hls", len(set(kmeans.labels_))),
        legend="full",
        alpha=0.7,
        s=120
        )
    for i in range(len(x)):

      plt.annotate(labels[i],
                   xy=(x[i], y[i]),
                   xytext=(3, 1),
                   textcoords='offset points',
                   ha='right',
                   va='bottom')


    plt.show()
```

```
%%time
tsne_plot(model)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: DeprecationWarning: Cal
  import sys
```



```
CPU times: user 2min 40s, sys: 1.94 s, total: 2min 42s
Wall time: 1min 27s
```

✓ 1s completed at 12:07 AM