Import Neccessary Librabies

```
import pandas as pd
import numpy as np
import datetime as dt
import random

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('ggplot')
import matplotlib.style as style
style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (15, 8)


from sklearn import preprocessing
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')
```

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

```
df
```

	@timestamp	_id	ip_address
0	July 8th 2019, 14:43:03.000	XswJ0msBoTGddM7vxMDB	10.1.1.285
1	July 8th 2019, 14:43:01.000	dKQJ0msB7mP0GwVzvJjz	10.1.2.389
2	July 8th 2019, 14:42:59.000	CcwJ0msBoTGddM7vtb8y	10.1.1.415
3	July 8th 2019, 14:42:57.000	bKQJ0msB7mP0GwVzrZdT	10.1.1.79
4	July 8th 2019, 14:42:55.000	L6QJ0msB7mP0GwVzpZeI	10.1.1.60
...	...	...	...
721542	June 9th 2019, 00:00:25.000	dkWJOWsBoTGddM7vRUOR	10.1.2.66
721543	June 9th 2019, 00:00:19.000	TBuJOWsB7mP0GwVzLmol	10.1.1.249
721544	June 9th 2019, 00:00:15.000	vUWJOWsBoTGddM7vHkGD	10.1.1.200
721545	June 9th 2019, 00:00:09.000	aUWJOWsBoTGddM7vB0AR	10.1.2.432
721546	June 9th 2019, 00:00:05.000	AEWIOWsBoTGddM7v90Bw	10.1.2.156

721547 rows × 3 columns

Feature Extraction

```
import re
```

```
df['@timestamp'] = df['@timestamp'].apply(lambda x: re.sub(r'(\d+)
        (st|nd|rd|th)', r'\1', x))

# Now convert to datetime
df['@timestamp'] = pd.to_datetime(df['@timestamp'])
```

This changes the timestamp from original to modified form

```
df.sort_values(['ip_address', '@timestamp'], inplace=True)
```

This sorts the dataset wrt to ip address and timestamp

```
df['shift_time'] = df.groupby(['ip_address'])['@timestamp'].shift(1)
```

It creates shift_time column one shiftdown for timestamp coumn

```
df['time_diff'] = (df['@timestamp'] -df['shift_time']).dt.seconds//60
```

Finding the time differnce between timestamp and shift time

```
df['date'] = df['@timestamp'].dt.date
```

Extracting the date from timestamp and creating seperate column

```
df['weekday'] = df['@timestamp'].dt.weekday
```

Extracting the no of weekdays present in timestamp

```
df['hour'] = df['@timestamp'].dt.hour
```

Finding the no.of hours present in timestamp column

```
df['is_weekend'] = ((df['weekday']==5 |
        (df['weekday']==6)).astype(int))
```

Finding how many weekend are present in timestamp

```
df['hour_bucket'] = df['hour']//4
```

```
df
```

	@timestamp	_id	ip_address	shift_time	time_diff	date	week
721473	2019-06-09 00:06:09	DBuOOWsB7mP0GwVzhZ9U	10.1.1.1	NaT	NaN	2019-06-09	6
720483	2019-06-09 01:28:39	bB7aOWsB7mP0GwVzDY5G	10.1.1.1	2019-06-09 00:06:09	82.0	2019-06-09	6
719233	2019-06-09 03:12:49	R0w5OmsBoTGddM7vayZT	10.1.1.1	2019-06-09 01:28:39	104.0	2019-06-09	6
719222	2019-06-09 03:13:45	U0w6OmsBoTGddM7vRi8R	10.1.1.1	2019-06-09 03:12:49	0.0	2019-06-09	6
718875	2019-06-09 03:42:39	z01UOmsBoTGddM7vuzyC	10.1.1.1	2019-06-09 03:13:45	28.0	2019-06-09	6
...	...	...	...	...	...	...	...
3225	2019-07-08 11:49:00	3cBq0WsBoTGddM7va5TJ	10.1.2.99	2019-07-08 11:47:15	1.0	2019-07-08	0
2422	2019-07-08 12:22:29	QMKJ0WsBoTGddM7vE9N1	10.1.2.99	2019-07-08 11:49:00	33.0	2019-07-08	0
1704	2019-07-08 12:52:25	9pyk0WsB7mP0GwVze7sV	10.1.2.99	2019-07-08 12:22:29	29.0	2019-07-	0

	@timestamp	_id	ip_address	shift_time	time_diff	date	week
						08	
1138	2019-07-08 13:15:59	B8a60WsBoTGddM7vDnqQ	10.1.2.99	2019-07-08 12:52:25	23.0	2019-07-08	0
368	2019-07-08 13:47:19	M8jW0WsBoTGddM7vvplE	10.1.2.99	2019-07-08 13:15:59	31.0	2019-07-08	0

721547 rows × 10 columns

```
ip_addr ='ip_address'
```

```
ip_counts = df.groupby(ip_addr)['@timestamp'].count().reset_index()
```

Counting how many times the user in logging into the computer wrt to ip address

```
ip_counts
```

	ip_address	@timestamp
0	10.1.1.1	1446
1	10.1.1.100	2860
2	10.1.1.101	1465
3	10.1.1.106	1408
4	10.1.1.109	1459
...	...	...
381	10.1.2.86	4307
382	10.1.2.89	2826
383	10.1.2.90	2904
384	10.1.2.95	2868
385	10.1.2.99	1423

386 rows × 2 columns

```
ip_counts = ip_counts.rename(columns={'@timestamp':'total_count'})
```

```
daily_counts = df.groupby([ip_addr,'date'])
        ['@timestamp'].count().reset_index()
```

```
daily_counts
```

	ip_address	date	@timestamp
0	10.1.1.1	2019-06-09	36
1	10.1.1.1	2019-06-10	37
2	10.1.1.1	2019-06-11	70
3	10.1.1.1	2019-06-12	38
4	10.1.1.1	2019-06-13	32
...	...	...	...
11575	10.1.2.99	2019-07-04	79
11576	10.1.2.99	2019-07-05	61
11577	10.1.2.99	2019-07-06	89
11578	10.1.2.99	2019-07-07	47

	ip_address	date	@timestamp
11579	10.1.2.99	2019-07-08	41

11580 rows × 3 columns

```
daily_counts = daily_counts.rename(columns=
        {'@timestamp':'daily_counts'})

daily_counts_avg =
        daily_counts.groupby(ip_addr).daily_counts.median().reset_index()
```

Calculating the median of the counts for all the date having the same ip address

```
daily_counts_avg.head(5)
```

	ip_address	daily_counts
0	10.1.1.1	40.0
1	10.1.1.100	78.0
2	10.1.1.101	40.0
3	10.1.1.106	35.5
4	10.1.1.109	42.5

```
weekend_counts = df.groupby([ip_addr, 'is_weekend'])
        ['@timestamp'].count().reset_index()

weekend_counts
```

	ip_address	is_weekend	@timestamp
0	10.1.1.1	0	1245
1	10.1.1.1	1	201
2	10.1.1.100	0	2463
3	10.1.1.100	1	397
4	10.1.1.101	0	1260
...	...	...	...
767	10.1.2.90	1	395
768	10.1.2.95	0	2478
769	10.1.2.95	1	390
770	10.1.2.99	0	1198
771	10.1.2.99	1	225

772 rows × 3 columns

```
weekend_counts = weekend_counts.rename(columns=
        {'@timestamp':'weekend_counts'})

weekend_counts.head()
```

	ip_address	is_weekend	weekend_counts
0	10.1.1.1	0	1245
1	10.1.1.1	1	201
2	10.1.1.100	0	2463
3	10.1.1.100	1	397
4	10.1.1.101	0	1260

calucating no.of logins occured in the weekends and weekdays

```
weekend_counts_avg = weekend_counts.pivot_table(index=ip_addr,
        columns='is_weekend').reset_index([0])
```

```
weekend_counts_avg.head()
```

	ip_address	weekend_counts	
is_weekend		**0**	**1**
0	10.1.1.1	1245.0	201.0
1	10.1.1.100	2463.0	397.0
2	10.1.1.101	1260.0	205.0
3	10.1.1.106	1190.0	218.0
4	10.1.1.109	1244.0	215.0

```
weekend_counts_avg.columns = weekend_counts_avg.columns.droplevel()
```

```
weekend_counts_avg
```

is_weekend		0	1
0	10.1.1.1	1245.0	201.0
1	10.1.1.100	2463.0	397.0
2	10.1.1.101	1260.0	205.0
3	10.1.1.106	1190.0	218.0
4	10.1.1.109	1244.0	215.0
...	...	...	...
381	10.1.2.86	3699.0	608.0
382	10.1.2.89	2435.0	391.0
383	10.1.2.90	2509.0	395.0
384	10.1.2.95	2478.0	390.0
385	10.1.2.99	1198.0	225.0

386 rows × 3 columns

```
weekend_counts_avg.columns = [ip_addr, 'week_day', 'weekend']
```

```
weekend_counts_avg['is_weekend_ratio'] =
        weekend_counts_avg['week_day']/ weekend_counts_avg['weekend']
```

finding the ratio between weekday and weekend

```
weekend_counts_avg.head()
```

	ip_address	week_day	weekend	is_weekend_ratio
0	10.1.1.1	1245.0	201.0	6.194030
1	10.1.1.100	2463.0	397.0	6.204030
2	10.1.1.101	1260.0	205.0	6.146341
3	10.1.1.106	1190.0	218.0	5.458716
4	10.1.1.109	1244.0	215.0	5.786047

```
lean_weekend_counts_avg = weekend_counts_avg[[ip_addr,
        'is_weekend_ratio']]
```

```
lean_weekend_counts_avg.head()
```

	ip_address	is_weekend_ratio
0	10.1.1.1	6.194030
1	10.1.1.100	6.204030
2	10.1.1.101	6.146341
3	10.1.1.106	5.458716
4	10.1.1.109	5.786047

```
avg_time_data = df.groupby(ip_addr).agg({'time_diff':
        ['mean','max']}).reset_index()
```

```
avg_time_data.head()
```

	ip_address	time_diff	
		mean	**max**
0	10.1.1.1	28.999308	362.0
1	10.1.1.100	14.427072	185.0
2	10.1.1.101	28.520492	211.0
3	10.1.1.106	29.771144	319.0
4	10.1.1.109	28.711934	278.0

```
avg_time_data.columns = avg_time_data.columns.droplevel()
```

```
avg_time_data.columns = [ip_addr, 'td_mean', 'td_max']
```

calulating the mean and max for time_diff column and labelling as td_mean and td_max

```
avg_time_data.head()
```

	ip_address	td_mean	td_max
0	10.1.1.1	28.999308	362.0
1	10.1.1.100	14.427072	185.0
2	10.1.1.101	28.520492	211.0
3	10.1.1.106	29.771144	319.0
4	10.1.1.109	28.711934	278.0

Total Features Set

```
mer_1 = ip_counts.merge(daily_counts_avg, on=ip_addr, how='left')
```

```
mer_2 = mer_1.merge(lean_weekend_counts_avg, on=ip_addr, how='left')
```

```
final_dataset = mer_2.merge(avg_time_data, on=ip_addr, how='left')
```

```
final_dataset.head()
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max
0	10.1.1.1	1446	40.0	6.194030	28.999308	362.0
1	10.1.1.100	2860	78.0	6.204030	14.427072	185.0

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max
2	10.1.1.101	1465	40.0	6.146341	28.520492	211.0
3	10.1.1.106	1408	35.5	5.458716	29.771144	319.0
4	10.1.1.109	1459	42.5	5.786047	28.711934	278.0

```
ip_map = final_dataset[ip_addr].to_dict()
```

```
RANDOM_STATE = 30
```

KMeans

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

```
final_dataset.columns
```

```
Index(['ip_address', 'total_count', 'daily_counts',
'is_weekend_ratio',
       'td_mean', 'td_max'],
      dtype='object')
```

Displaying the columns present in the final_dataset

```
requi_feature_cols = ['total_count', 'daily_counts',
        'is_weekend_ratio', 'td_mean', 'td_max']
```

```
new_data = final_dataset[requi_feature_cols]
```

```
min_max_scaler = preprocessing.MinMaxScaler()
```

```
data_scaled = min_max_scaler.fit_transform(new_data)
new_data = pd.DataFrame(data_scaled,columns=requi_feature_cols)
```

```
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
```

```
sns.pairplot(final_dataset[requi_feature_cols])
```

```
<seaborn.axisgrid.PairGrid at 0x7c4342ffcc10>
```

```
sns.pairplot(new_data)
```

```
<seaborn.axisgrid.PairGrid at 0x7c43370efe20>
```

```
cluster_num = range(1,15)
```

```
kmeans = [KMeans(n_clusters=i,
        random_state=RANDOM_STATE).fit(new_data) for i in
        cluster_num]
```

```
kmeans
```

```
[KMeans(n_clusters=1, random_state=30),
 KMeans(n_clusters=2, random_state=30),
 KMeans(n_clusters=3, random_state=30),
 KMeans(n_clusters=4, random_state=30),
 KMeans(n_clusters=5, random_state=30),
 KMeans(n_clusters=6, random_state=30),
 KMeans(n_clusters=7, random_state=30),
 KMeans(random_state=30),
 KMeans(n_clusters=9, random_state=30),
 KMeans(n_clusters=10, random_state=30),
 KMeans(n_clusters=11, random_state=30),
```

```
 KMeans(n_clusters=12, random_state=30),
 KMeans(n_clusters=13, random_state=30),
 KMeans(n_clusters=14, random_state=30)]
```

```
scores = [kmeans[i].score(new_data) for i in range(len(kmeans))]
```

```
scores
```

```
[-105.61717382018342,
 -22.662497940589923,
 -18.425259145173772,
 -11.34116401117781,
 -9.624264656308904,
 -7.6834491165191485,
 -6.605626776063946,
 -5.773343216208603,
 -5.2669670163554665,
 -4.731542252194097,
 -4.483092839218669,
 -4.1109723094544375,
 -3.9656970956213393,
 -3.803909950255066]
```

```
fig, sd = plt.subplots()
sd.plot(cluster_num,scores)
plt.show()
```

Displays the graph between no of clusters taken (i.e., 15) and scores calculate using kmeans

```
kmeans_cluster_model=kmeans[5]
```

```
kmeans_cluster_model
```

```
▾ KMeans
KMeans(n_clusters=6, random_state=30)
```

```
final_dataset['cluster'] = kmeans_cluster_model.predict(new_data)
final_dataset['cluster'].value_counts()
```

	count
cluster	
4	99
2	94
3	73
0	60
5	50
1	10

dtype: int64

Clustering model with TSNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

```
from sklearn.manifold import TSNE
```

```
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300,
           random_state=RANDOM_STATE)

tsne_result = tsne.fit_transform(new_data)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 386 samples in 0.003s...
[t-SNE] Computed neighbors for 386 samples in 0.012s...
[t-SNE] Computed conditional probabilities for sample 386 / 386
[t-SNE] Mean sigma: 0.081652
[t-SNE] KL divergence after 250 iterations with early exaggeration:
48.038315
[t-SNE] KL divergence after 300 iterations: 0.375644
```

```
tsne_result
```

```
array([[  7.5516505 ,  -1.8475966 ],
       [-10.225705  ,   3.391964  ],
       [  1.2972114 ,  -0.71510386],
       [ -0.46138164,  -9.297569  ],
       [ -0.4155114 ,  -5.7562766 ],
       [ -0.5469004 ,  -3.0171854 ],
       [ -0.5501037 ,  -8.547561  ],
       [  7.765187  ,   6.006567  ],
       [ -0.15269239,  -1.8949823 ],
       [ -3.8881352 ,  -9.977213  ],
       [  3.4908392 ,  -5.704336  ],
       [ -1.0357306 ,  -6.9751925 ],
       [  1.1646099 ,   1.7552608 ],
       [  8.432264  ,   6.2101417 ],
       [ -1.4891664 ,  -9.2100115 ],
       [ -7.519641  ,  -1.3749399 ],
       [ -0.76689535,  -2.0927768 ],
       [  7.1565075 ,   5.0961294 ],
       [ -8.274764  ,   5.093319  ],
       [  0.87136555,  -2.0926366 ],
       [ -7.826874  ,   3.5390432 ],
       [  3.791072  ,   6.8412786 ],
       [ -9.575649  ,   3.1339493 ],
       [  8.120027  ,  -0.70197636],
       [  3.0600898 ,   2.448223  ],
       [ -8.662781  ,   5.5448275 ],
       [ -2.0453768 ,  -3.1752026 ],
       [ -9.21111   ,   2.8052282 ],
       [  4.239176  ,  -1.2265401 ],
       [ -8.504666  ,  -0.6268253 ],
       [ -0.24986817,  -0.21668836],
       [  8.872415  ,   6.5721226 ],
       [ -3.2113032 ,  -9.733126  ],
       [ -3.2729151 ,  -9.581918  ],
       [ 10.037203  ,   5.1175065 ],
       [ -3.9138107 ,  -7.388327  ],
       [  6.612384  ,   4.912858  ],
       [ -2.8380337 , -10.594885  ],
       [ -0.33648354,  -2.0505588 ],
       [ -9.085518  ,   2.0763085 ],
       [ -7.3435044 ,  -0.46982232],
       [ -4.4136896 ,  -9.014411  ],
       [  9.521205  ,  -1.4890804 ],
       [  0.80809855,  -8.774018  ],
       [-10.035588  ,   4.123908  ],
       [ -0.25679493,  -1.3036724 ],
       [ -0.70464915,  -2.4489253 ],
```

```
       [  0.67449737,  -6.130704  ],
       [  9.275403  ,  -1.6018873 ],
       [ -7.214809  ,   0.12313751],
       [ -7.5840807 ,   2.904727  ],
       [  4.3864813 ,   1.1620686 ],
       [ -2.3519268 ,  -2.3771853 ],
       [ -0.36104614,  -0.47613105],
       [  8.072875  ,   1.8572581 ],
       [  6.431285  ,   0.41142148],
       [  9.965297  ,   6.219332  ],
       [  9.586744  ,   3.4561617 ],
       [  1.7513157 ,  -7.278637  ],
       [  0.91844064,  -0.16697057],
       [ -2.642427  , -10.553289  ],
       [  1.4841914 ,   2.204627  ],
       [ -4.9721    ,   7.415359  ],
       [  8.068536  ,  -2.3025377 ],
       [ 10.16497   ,   1.929176  ],
       [  2.109656  ,   2.9511747 ],
       [ -9.566312  ,   5.146094  ],
       [-10.845969  ,   1.7490666 ],
       [ -3.046348  ,  -4.8552985 ],
       [  5.0444207 ,   1.3191721 ],
       [ -3.3538218 ,  -6.0042315 ],
       [  0.43712318,  -2.864621  ],
       [  7.615604  ,   3.1735542 ],
       [  1.3784151 ,  -0.17452662],
       [  5.341224  ,  -0.2307079 ],
       [  6.34334   ,   6.0535536 ],
       [  7.093119  ,  -0.3511597 ],
       [ -3.5887682 ,  -7.4006276 ],
       [  7.9050703 ,   6.5780907 ],
       [  7.7043557 ,   5.6815295 ],
       [  1.1050584 ,   1.5189962 ],
       [ -0.6201843 ,  -0.5302933 ],
       [-10.227171  ,   5.006499  ],
       [ -0.46834883,  -5.0612507 ],
       [  4.544411  ,   6.431243  ],
       [  9.202032  ,   3.3721266 ],
       [  1.7531061 ,  -5.38834   ],
       [-10.280642  ,   6.4500546 ],
       [  0.40690765,  -6.5543003 ],
       [ -7.863549  ,  -1.3731467 ],
       [ -1.3133913 ,  -2.5810833 ],
       [  1.9910551 ,  -6.0370464 ],
       [ 10.249829  ,   1.9259892 ],
       [ -3.69339   ,  -9.244521  ],
       [ -0.3367704 ,  -8.247842  ],
       [ -7.206224  ,   1.206936  ],
       [ -9.002671  ,  -1.4336945 ],
       [  3.3440201 ,   6.385072  ],
       [  3.315866  ,  -5.7613273 ],
       [ -7.311106  ,   0.4642485 ],
       [ -9.527311  ,   4.920621  ],
       [ -8.454741  ,   2.7023454 ],
       [  6.907701  ,   4.7917414 ],
       [ -1.3976176 ,  -2.8039713 ],
       [  5.835698  ,   6.1200414 ],
       [  4.03682   ,   0.12603366],
       [  4.9663386 ,   1.9493986 ],
       [ 10.842741  ,   3.5911164 ],
```

```
[ -0.5877799 ,  -2.3881931 ],
[  0.17206359,  -6.8011966 ],
[ -1.853055  ,  -2.2830646 ],
[  0.5560259 ,   1.731691  ],
[ -0.25288728,  -5.4363832 ],
[  2.5792649 ,   1.3703887 ],
[  0.52658135,  -2.611129  ],
[ -9.2440195 ,   5.2866893 ],
[ -0.23880327,  -2.3984556 ],
[  6.326616  ,  -4.2380624 ],
[ -5.0232887 ,   7.423323  ],
[  2.6420152 ,   0.2956374 ],
[  1.2277536 ,  -7.4989886 ],
[  5.4381614 ,  -0.01398319],
[ -3.7401092 ,  -7.947749  ],
[ -3.3155658 ,  -5.578449  ],
[ -0.15798189,  -8.171668  ],
[ -9.601808  ,   2.9162862 ],
[  5.1273026 ,  -6.417984  ],
[  2.9859195 ,   0.72215056],
[ -7.8021283 ,   4.0924783 ],
[ -8.201233  ,   0.25701195],
[ -8.007552  ,   2.317952  ],
[ -7.5932875 ,  -0.8512897 ],
[ -0.8032999 ,  -4.602177  ],
[ -1.1964248 ,  -4.7707667 ],
[ -7.1217957 ,  -1.2531309 ],
[ -8.810299  ,   5.2131615 ],
[  3.4220288 ,   5.821672  ],
[ -8.109658  ,   4.216654  ],
[  3.8678193 ,   4.251815  ],
[  4.4329    ,   3.1282263 ],
[  3.097558  ,   4.575771  ],
[  6.537287  ,   5.5245233 ],
[ -0.8822149 ,  -6.141899  ],
[  2.5658786 ,  -6.375063  ],
[ -8.436011  ,   2.788177  ],
[  9.654978  ,   5.267055  ],
[  4.7307844 ,   0.787298  ],
[  5.270579  ,  -5.541794  ],
[  1.871643  ,  -7.440982  ],
[  4.3594656 ,  -6.292139  ],
[ -8.137572  ,   1.621781  ],
[ -8.475302  ,  -0.52125907],
[  7.3244944 ,   0.8871866 ],
[  6.544547  ,  -1.1849209 ],
[  6.9725847 ,   3.2726576 ],
[ -9.84224   ,   5.982141  ],
[  4.1053405 ,   5.0714827 ],
[  8.86385   ,   6.8338666 ],
[ -0.31250063,  -7.7112803 ],
[  8.526107  ,   3.7651196 ],
[  7.5085917 ,  -2.902245  ],
[ -8.411455  ,   2.73082   ],
[  8.610803  ,   0.71551114],
[ 11.079265  ,   4.6273437 ],
[ -4.8988442 ,   7.399555  ],
[  4.675354  ,  -7.8408713 ],
[ -3.797157  ,  -7.165094  ],
[  9.992079  ,   0.576537  ],
[ -9.4438305 ,   4.4574184 ],
```

```
[ -3.9194436 ,  -7.852621  ],
[  3.7866926 ,   0.7845617 ],
[ -7.186013  ,  -1.1096123 ],
[ -9.086448  ,   1.19851   ],
[  4.823687  ,   4.3859944 ],
[ -9.487331  ,   4.857388  ],
[ -7.0874476 ,   0.6224468 ],
[  3.364738  ,  -7.9632673 ],
[ -4.9857235 ,   7.4169545 ],
[ -9.188779  ,   4.317134  ],
[  3.5353937 ,   4.307541  ],
[  4.388     ,  -7.63075   ],
[ -2.406084  ,  -3.1559694 ],
[  0.8602654 ,  -7.4981318 ],
[ -4.2980394 ,  -6.3929906 ],
[  2.5513408 ,  -7.0550423 ],
[-10.415072  ,   3.2952728 ],
[  5.178182  ,   2.1974082 ],
[  4.9382787 ,  -6.8248773 ],
[ -5.0056467 ,   7.42017   ],
[ -7.5232344 ,  -0.9926067 ],
[ -9.705006  ,   6.2904177 ],
[ -9.278574  ,   2.306893  ],
[ -2.1292305 ,  -9.06412   ],
[ -0.9430999 ,  -1.2244378 ],
[ -8.422344  ,  -0.943735  ],
[ -2.7546048 ,  -9.490207  ],
[  6.306572  ,   4.126214  ],
[  0.02314082,  -7.427666  ],
[  1.0587151 ,   0.83791167],
[ -0.69347787,  -6.353191  ],
[ -0.15043582,  -5.734606  ],
[  0.25352892,  -9.646122  ],
[ -4.1101327 ,  -6.143951  ],
[  2.7571301 ,   5.3297725 ],
[  9.790222  ,   3.5122783 ],
[  1.0009325 ,  -5.6560564 ],
[  8.050724  ,   7.6007414 ],
[  0.02448624,  -0.6766023 ],
[ -7.3892007 ,  -1.5381092 ],
[ -3.0116904 ,  -9.1967945 ],
[ -1.0414081 ,  -2.0300717 ],
[ -7.2622075 ,  -0.39397246],
[  8.322739  ,   5.9876657 ],
[ -7.294854  ,   0.11719458],
[ -0.4323299 ,  -6.322633  ],
[ -7.5532084 ,   0.46661198],
[ 10.912439  ,   4.255603  ],
[  0.951645  ,  -1.5597274 ],
[ -8.63143   ,   5.4017906 ],
[  7.8908863 ,   0.6722568 ],
[-11.033548  ,   5.2551756 ],
[  6.8722095 ,  -1.9401643 ],
[ -7.5752506 ,   2.297085  ],
[  3.680281  ,  -5.208433  ],
[  3.024354  ,   5.9182634 ],
[ 10.393672  ,   6.281297  ],
[  3.6114502 ,  -6.021234  ],
[  7.2847047 ,  -2.8780723 ],
[  2.313024  ,   1.3269377 ],
[  3.8739963 ,   0.21393618],
```

```
       [  8.782332  ,   6.8852186 ],
       [ -4.187333  ,  -6.2750416 ],
       [ -8.49623   ,  -1.6392134 ],
       [-10.741546  ,   6.2381315 ],
       [  8.571211  ,   5.853613  ],
       [  9.54919   ,   6.937699  ],
       [  7.0398626 ,   5.601891  ],
       [ -7.8039646 ,  -0.29211736],
       [  5.3021564 ,  -1.9037117 ],
       [  6.219912  ,   3.762932  ],
       [  2.527923  ,   3.0185199 ],
       [ -3.2136517 ,  -5.282903  ],
       [ -8.730516  ,   0.9586098 ],
       [ -9.529506  ,   5.193868  ],
       [  7.1998634 ,  -2.4381976 ],
       [  7.7580886 ,   1.3580742 ],
       [  9.69853   ,   2.7546942 ],
       [  3.345229  ,  -6.58084   ],
       [  7.262591  ,   5.6247582 ],
       [  9.160646  ,   1.5266347 ],
       [  6.1522856 ,   4.9449887 ],
       [  2.5619423 ,  -6.0043397 ],
       [  0.87731916,   1.7117002 ],
       [  8.738298  ,   1.2790021 ],
       [  5.3049564 ,  -0.24052036],
       [ -4.9492664 ,   7.4108257 ],
       [  6.1071415 ,   5.1790843 ],
       [  4.0289793 ,  -7.869498  ],
       [  3.8387558 ,   5.3000684 ],
       [-10.439198  ,   5.1277018 ],
       [-11.097665  ,   3.1170197 ],
       [  0.39373803,  -7.6296663 ],
       [ -2.8092585 ,  -8.272404  ],
       [ -0.26586327,  -6.2073565 ],
       [  4.49466   ,   3.155689  ],
       [ -1.3482525 ,  -3.5573733 ],
       [ -3.5214849 ,  -8.2582245 ],
       [  5.158204  ,   1.9199684 ],
       [ -1.5660052 ,  -5.909728  ],
       [  0.52627033,  -0.0300377 ],
       [  9.965371  ,   5.216836  ],
       [  5.3379364 ,   1.9800582 ],
       [ -4.0188093 ,  -8.074947  ],
       [  1.9550326 ,   6.0001497 ],
       [ -0.81441575,  -4.520684  ],
       [ -7.45731   ,   2.6286607 ],
       [ -3.8121068 ,  -9.090844  ],
       [ 10.286534  ,   4.654329  ],
       [ -0.93606526,   0.45939672],
       [ -4.6507792 ,  -9.14857   ],
       [ -8.695256  ,   6.030247  ],
       [  7.804068  ,   6.1362076 ],
       [ -7.9924383 ,  -2.050252  ],
       [  9.396147  ,   3.818502  ],
       [  0.04974277,  -6.077409  ],
       [  6.126538  ,   5.8030763 ],
       [ 10.370402  ,   0.16605313],
       [ -1.0850594 ,  -3.179241  ],
       [-11.199616  ,   4.407139  ],
       [ -9.728398  ,   5.902996  ],
       [  1.8749496 ,   3.8859546 ],
```

```
       [  2.1479826 ,   0.35189345],
       [ -7.436793  ,   1.9987918 ],
       [ -8.62981   ,   3.3067336 ],
       [ -9.459321  ,   3.6035857 ],
       [ -2.79324   , -10.108137  ],
       [  3.1166472 ,   1.1724608 ],
       [  4.689856  ,   1.0671614 ],
       [  3.6645362 ,   0.17381257],
       [  8.321787  ,   5.518148  ],
       [  6.776233  ,   6.485729  ],
       [  5.1911197 ,   3.723264  ],
       [  9.88609   ,  -0.7481167 ],
       [  9.043081  ,   5.084916  ],
       [ -1.6038747 ,  -1.0869946 ],
       [  9.171249  ,   2.1249588 ],
       [  6.6266427 ,  -0.18255654],
       [  5.039749  ,   5.200887  ],
       [ -8.276483  ,  -0.9401012 ],
       [  8.059337  ,  -1.366189  ],
       [ -1.1869653 ,  -2.8552525 ],
       [ -4.950544  ,   7.4102535 ],
       [  2.7333407 ,   6.4027762 ],
       [ 10.331442  ,   3.1637485 ],
       [  9.704704  ,   6.3120627 ],
       [ -0.6945104 ,  -2.1428742 ],
       [  7.901538  ,   7.957917  ],
       [  3.9481118 ,  -5.980286  ],
       [ -8.020684  ,  -0.270091  ],
       [  4.312795  ,   6.0028105 ],
       [  8.533733  ,   4.7656245 ],
       [-10.230547  ,   0.58226115],
       [-10.42106   ,   2.159379  ],
       [-10.326785  ,   4.2871704 ],
       [ -8.171837  ,   5.6394114 ],
       [  4.948659  ,   2.5407665 ],
       [  3.4604983 ,  -6.266952  ],
       [ -7.5860553 ,   3.4124322 ],
       [ -8.322621  ,  -1.1907896 ],
       [  4.1393948 ,  -5.4775953 ],
       [ -8.69376   ,   3.3114686 ],
       [  9.7203455 ,  -1.1430985 ],
       [  4.168974  ,  -1.2225254 ],
       [  6.2312474 ,   0.24203338],
       [  6.107955  ,  -0.82137555],
       [  0.6657279 ,  -2.344703  ],
       [  7.692807  ,   8.009152  ],
       [ -6.892473  ,   1.026458  ],
       [  3.2526484 ,   1.3005866 ],
       [  1.7111582 ,  -6.4716787 ],
       [ 10.620441  ,   2.8196354 ],
       [  8.534326  ,  -1.8689102 ],
       [  7.488677  ,  -1.8195282 ],
       [ -7.5705876 ,   2.033136  ],
       [ -9.709394  ,   1.364841  ],
       [  9.775258  ,   4.3876944 ],
       [  1.3832229 ,   1.9480975 ],
       [ -2.1925237 ,  -3.0229917 ],
       [ -9.50282   ,   2.7679112 ],
       [ -4.2546062 ,  -6.472413  ],
       [  3.9185512 ,   6.5085764 ],
       [  5.466359  ,  -1.9501779 ],
```

```
       [  0.10308456,  -0.6080009 ],
       [  0.8582513 ,  -0.40795267],
       [ -7.2561626 ,   2.281721  ],
       [ -2.0023286 ,  -7.0024652 ],
       [  3.3344615 ,   0.7163736 ],
       [ -5.0044384 ,   7.418109  ],
       [  8.498514  ,   7.351565  ],
       [  0.22413048,   0.4884707 ],
       [ -0.8560239 ,  -0.80459887],
       [ -9.013876  ,   1.8174455 ],
       [ -2.742616  ,  -7.886526  ],
       [ -0.7596878 ,  -5.8139873 ],
       [-10.273685  ,   2.293888  ],
       [ -9.669187  ,  -0.7673675 ],
       [ -3.7343407 ,  -7.5329447 ],
       [-10.817912  ,   1.8150467 ],
       [ -7.780097  ,   4.739589  ],
       [  6.0820384 ,  -0.8053435 ],
       [ -5.031483  ,   7.4263315 ],
       [ -7.01314   ,   0.7499768 ],
       [  7.9539533 ,   5.8679953 ],
       [  9.257565  ,   7.262332  ],
       [  3.3248353 ,   6.420118  ],
       [ -3.635407  ,  -9.083748  ],
       [  1.0136844 ,  -9.377589  ],
       [  1.329351  ,  -2.6755066 ],
       [ -8.456686  ,  -0.28654614],
       [  4.6833477 ,   2.5245333 ],
       [  3.1899688 ,   4.7476654 ],
       [ -4.9805217 ,   7.416666  ],
       [-11.072982  ,   3.176785  ],
       [ -9.226236  ,   4.7922134 ],
       [-10.478027  ,   4.4418173 ],
       [ -3.2145355 ,  -9.342764  ]], dtype=float32)
```

```
final_dataset['tsne-2d-one'] = tsne_result[:,0]
final_dataset['tsne-2d-two'] = tsne_result[:,1]

tsne_cluster = final_dataset.groupby('cluster').agg({'tsne-2d-
        one':'mean','tsne-2d-two':'mean'}).reset_index()

plt.figure(figsize=(16,10))


sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="cluster",
    palette=sns.color_palette("hls", 6),
    data=final_dataset,
    legend="full",
    alpha=1
)

plt.scatter(x="tsne-2d-one", y="tsne-2d-two", data=tsne_cluster,
        s=100, c='b')


plt.show()
```

```
centers = kmeans_cluster_model.cluster_centers_

pts = np.asarray(new_data)

pts
```

```
array([[0.03813281, 0.05263158, 0.51209855, 0.88495446, 0.71662125],
       [0.50295858, 0.49707602, 0.51626613, 0.23136629, 0.23433243],
       [0.0443787 , 0.05263158, 0.49222476, 0.86347879, 0.30517711],
       ...,
       [0.51742275, 0.47368421, 0.57788924, 0.22188316, 0.16348774],
       [0.50558843, 0.49122807, 0.57870081, 0.2304837 , 0.24250681],
       [0.03057199, 0.01754386, 0.14970523, 0.90324786, 0.45776567]])
```

```
total_dist = pd.Series()
```

```
import numpy as np
import pandas as pd

def get_sum_square_distance(data, kmeans_cluster_model,
        requi_feature_cols):
    # Get cluster centers from the k-means model
    centers = kmeans_cluster_model.cluster_centers_

    # Convert the data to a numpy array of the required feature
        columns
    pts = np.asarray(data[requi_feature_cols])

    # Initialize total_dist as an empty Series of the same length as
        pts
    total_dist = pd.Series(np.zeros(len(pts)))

    # Loop over each point and compute its sum of squared distances to
        each cluster center
    for i in range(len(pts)):
        dist = 0
        for j in range(len(centers)):
            a = np.linalg.norm(pts[i] - centers[j])  # Euclidean
        distance
            dist += a**2  # Square the distance and add it to the
        total
        total_dist.iloc[i] = dist  # Use iloc to set the value at
        index i

    return total_dist

# Example assuming `new_data`, `kmeans_cluster_model`, and
        `requi_feature_cols` are defined
sum_square_distances = get_sum_square_distance(new_data,
        kmeans_cluster_model, requi_feature_cols)

# Check that the length of the result matches the number of rows in
        `final_dataset`
if len(sum_square_distances) == len(final_dataset):
    final_dataset['sum_squa_dist'] = sum_square_distances
else:
    print("Error: The length of the result does not match the number
        of rows in final_dataset.")
```

```
plt.hist(final_dataset['sum_squa_dist'], bins=100)
```

```
(array([ 6.,  7., 18., 23., 25., 44., 26., 23., 17., 20., 11., 21.,
17.,
        25., 14., 15., 14., 11.,  7., 10.,  4.,  4.,  4.,  2.,  2.,
1.,
         1.,  0.,  0.,  0.,  0.,  3.,  0.,  0.,  0.,  0.,  0.,  0.,
1.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,
         0.,  0.,  0.,  0.,  0.,  0.,  2.,  0.,  0.,  0.,  1.,  2.,
```

```
0.,
         1.,  0.,  1.,  0.,  1.,  1.,  0.,  0.,  1.]),
 array([ 3.39616405,  3.48347384,  3.57078364,  3.65809343,
3.74540322,
         3.83271302,  3.92002281,  4.0073326 ,  4.09464239,
4.18195219,
         4.26926198,  4.35657177,  4.44388157,  4.53119136,
4.61850115,
         4.70581095,  4.79312074,  4.88043053,  4.96774032,
5.05505012,
         5.14235991,  5.2296697 ,  5.3169795 ,  5.40428929,
5.49159908,
         5.57890887,  5.66621867,  5.75352846,  5.84083825,
5.92814805,
         6.01545784,  6.10276763,  6.19007742,  6.27738722,
6.36469701,
         6.4520068 ,  6.5393166 ,  6.62662639,  6.71393618,
6.80124597,
         6.88855577,  6.97586556,  7.06317535,  7.15048515,
7.23779494,
         7.32510473,  7.41241452,  7.49972432,  7.58703411,  7.6743439
,
         7.7616537 ,  7.84896349,  7.93627328,  8.02358307,
8.11089287,
         8.19820266,  8.28551245,  8.37282225,  8.46013204,
8.54744183,
         8.63475162,  8.72206142,  8.80937121,  8.896681  ,  8.9839908
,
         9.07130059,  9.15861038,  9.24592018,  9.33322997,
9.42053976,
         9.50784955,  9.59515935,  9.68246914,  9.76977893,
9.85708873,
         9.94439852, 10.03170831, 10.1190181 , 10.2063279 ,
10.29363769,
        10.38094748, 10.46825728, 10.55556707, 10.64287686,
10.73018665,
        10.81749645, 10.90480624, 10.99211603, 11.07942583,
11.16673562,
        11.25404541, 11.3413552 , 11.428665  , 11.51597479,
11.60328458,
        11.69059438, 11.77790417, 11.86521396, 11.95252375,
12.03983355,
        12.12714334]),
 <BarContainer object of 100 artists>)
```

```
cutoff = 6

final_dataset['anomaly_kmeans'] = (final_dataset['sum_squa_dist'] >=
        cutoff).astype(int)

sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="anomaly_kmeans",
    data=final_dataset,
    legend="full",
    alpha=1
)
```

```
<Axes: xlabel='tsne-2d-one', ylabel='tsne-2d-two'>
```

```
final_dataset.loc[final_dataset['anomaly_kmeans']==1]
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsn on
42	10.1.1.199	1365	40.5	6.260638	30.801320	455.0	5	9.5
62	10.1.1.249	4301	116.5	6.004886	9.459535	101.0	1	-4.9
118	10.1.1.386	4300	118.5	6.388316	9.453361	104.0	1	-5.0
163	10.1.1.483	1330	39.0	7.364780	31.564334	316.0	0	11.0
164	10.1.1.486	4317	117.0	5.611026	9.417285	108.0	1	-4.8
177	10.1.1.63	4339	112.0	6.136513	9.368142	101.0	1	-4.9
188	10.1.1.86	4293	113.0	6.276271	9.456897	110.0	1	-5.0
255	10.1.2.249	4353	112.0	5.876777	9.332721	102.0	1	-4.9
286	10.1.2.323	1408	43.5	7.045714	29.719261	458.0	5	10.3
311	10.1.2.386	4326	108.0	5.910543	9.392370	110.0	1	-4.9
331	10.1.2.432	1437	40.5	6.445596	29.176880	466.0	5	9.7
357	10.1.2.486	4251	114.0	6.316695	9.571059	99.0	1	-5.0
370	10.1.2.63	4372	121.0	6.435374	9.268588	118.0	1	-5.0
381	10.1.2.86	4307	111.0	6.083882	9.441013	122.0	1	-4.9

Isolated Forest

```
from sklearn.ensemble import IsolationForest
```

```
outlier_frac = 0.028
```

```
final_dataset.columns
```

```
Index(['ip_address', 'total_count', 'daily_counts',
'is_weekend_ratio',
       'td_mean', 'td_max', 'cluster', 'tsne-2d-one', 'tsne-2d-two',
       'sum_square_dist', 'sum_squa_dist', 'anomaly_kmeans'],
      dtype='object')
```

```
new_data.head()
```

	total_count	daily_counts	is_weekend_ratio	td_mean	td_max
0	0.038133	0.052632	0.512099	0.884954	0.716621
1	0.502959	0.497076	0.516266	0.231366	0.234332
2	0.044379	0.052632	0.492225	0.863479	0.305177
3	0.025641	0.000000	0.205662	0.919573	0.599455
4	0.042406	0.081871	0.342074	0.872065	0.487738

```
iso_model =  IsolationForest(n_jobs=-1, n_estimators=200,
        max_features=3, random_state=RANDOM_STATE,
        contamination=outlier_frac)
```

```
iso_model.fit(new_data)
```

```
▾ IsolationForest
IsolationForest(contamination=0.028, max_features=3, n_estimators=200,
                n_jobs=-1, random_state=30)
```

```
final_dataset['anomaly_isolated'] =
        pd.Series(iso_model.predict(new_data))

final_dataset['anomaly_isolated'] =
        final_dataset['anomaly_isolated'].map( {1: 0, -1: 1} )

final_dataset['anomaly_isolated'].value_counts()
```

	count
anomaly_isolated	
0	375
1	11

dtype: int64

```
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="anomaly_isolated",
    data=final_dataset,
    legend="full",
    alpha=1
)
```

```
<Axes: xlabel='tsne-2d-one', ylabel='tsne-2d-two'>
```

```
final_dataset.loc[final_dataset['anomaly_isolated']==1]
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsn one
62	10.1.1.249	4301	116.5	6.004886	9.459535	101.0	1	-4.9
118	10.1.1.386	4300	118.5	6.388316	9.453361	104.0	1	-5.0
163	10.1.1.483	1330	39.0	7.364780	31.564334	316.0	0	11.0
164	10.1.1.486	4317	117.0	5.611026	9.417285	108.0	1	-4.8
177	10.1.1.63	4339	112.0	6.136513	9.368142	101.0	1	-4.9
188	10.1.1.86	4293	113.0	6.276271	9.456897	110.0	1	-5.0
255	10.1.2.249	4353	112.0	5.876777	9.332721	102.0	1	-4.9
311	10.1.2.386	4326	108.0	5.910543	9.392370	110.0	1	-4.9
357	10.1.2.486	4251	114.0	6.316695	9.571059	99.0	1	-5.0
370	10.1.2.63	4372	121.0	6.435374	9.268588	118.0	1	-5.0
381	10.1.2.86	4307	111.0	6.083882	9.441013	122.0	1	-4.9

One Class SVM

```
from sklearn.svm import OneClassSVM

svm_model =  OneClassSVM(nu=outlier_frac, degree=2, kernel='rbf')

new_data_clean = new_data.loc[new_data.total_count
        <=new_data.total_count.quantile(1-outlier_frac)]

svm_model.fit(new_data_clean)
```

```
▾ OneClassSVM
OneClassSVM(degree=2, nu=0.028)
```

```
final_dataset['anomaly_svm'] = pd.Series(svm_model.predict(new_data))

final_dataset['anomaly_svm'] = final_dataset['anomaly_svm'].map( {1:
        0, -1: 1} )

final_dataset['anomaly_svm'].value_counts()
```

	count
anomaly_svm	
0	367
1	19

dtype: int64

```
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="anomaly_svm",
    data=final_dataset,
    legend="full",
    alpha=1
)
```

```
<Axes: xlabel='tsne-2d-one', ylabel='tsne-2d-two'>
```

```
final_dataset.loc[final_dataset['anomaly_svm']==1]
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsn one
9	10.1.1.12	1392	40.0	4.974249	30.152408	250.0	3	-3.8
37	10.1.1.186	1372	37.0	4.965217	30.557987	308.0	3	-2.8
62	10.1.1.249	4301	116.5	6.004886	9.459535	101.0	1	-4.9
118	10.1.1.386	4300	118.5	6.388316	9.453361	104.0	1	-5.0
164	10.1.1.486	4317	117.0	5.611026	9.417285	108.0	1	-4.8
177	10.1.1.63	4339	112.0	6.136513	9.368142	101.0	1	-4.9
183	10.1.1.73	1484	40.5	5.480349	28.191504	195.0	3	-4.2
188	10.1.1.86	4293	113.0	6.276271	9.456897	110.0	1	-5.0
225	10.1.2.178	1400	39.0	7.284024	29.964975	258.0	0	10.3
233	10.1.2.195	2859	76.5	6.919668	14.448915	206.0	2	-10
255	10.1.2.249	4353	112.0	5.876777	9.332721	102.0	1	-4.9
282	10.1.2.314	2891	75.0	5.312227	14.274740	131.0	2	-7.9
286	10.1.2.323	1408	43.5	7.045714	29.719261	458.0	5	10.3
311	10.1.2.386	4326	108.0	5.910543	9.392370	110.0	1	-4.9
331	10.1.2.432	1437	40.5	6.445596	29.176880	466.0	5	9.7
357	10.1.2.486	4251	114.0	6.316695	9.571059	99.0	1	-5.0
370	10.1.2.63	4372	121.0	6.435374	9.268588	118.0	1	-5.0
373	10.1.2.67	1364	40.0	6.976608	30.748349	229.0	0	9.2
381	10.1.2.86	4307	111.0	6.083882	9.441013	122.0	1	-4.9

Performance of the Algorithm

We must manually categorise the data and examine the performance of the individual algos because it is unsupervised.

```
total_counts_condition = (final_dataset['total_count'] >=
        final_dataset['total_count'].quantile(0.90))
daily_counts_condition =  (final_dataset['daily_counts'] >=
        final_dataset['daily_counts'].quantile(0.90))

final_dataset['anomaly_manual'] = (total_counts_condition |
        daily_counts_condition).astype(int)

final_dataset.head()
```

	ip_address	total_count	daily_counts	is_weekend_ratio	td_mean	td_max	cluster	tsne-2 one
0	10.1.1.1	1446	40.0	6.194030	28.999308	362.0	5	7.5516
1	10.1.1.100	2860	78.0	6.204030	14.427072	185.0	2	-10.22
2	10.1.1.101	1465	40.0	6.146341	28.520492	211.0	4	1.2972
3	10.1.1.106	1408	35.5	5.458716	29.771144	319.0	3	-0.461
4	10.1.1.109	1459	42.5	5.786047	28.711934	278.0	3	-0.415

```
from sklearn.metrics import f1_score, roc_auc_score, accuracy_score,
        confusion_matrix

def get_sensitivity_specificity(y_true, y_pred):
    cf = confusion_matrix(y_true, y_pred)
    sensitivity = cf[0,0]/(cf[:,0].sum())
    specificity = cf[1,1]/(cf[:,1].sum())
    return sensitivity, specificity

f1_iso =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
acc_iso =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
roc_iso =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
sen_iso, spec_iso =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_isolated'])
met_iso = {
           'f1_score': f1_iso,
           'accuracy': acc_iso,
           'roc_score': roc_iso,
           'sensitivity': sen_iso,
           'specificity': spec_iso
          }

f1_kmeans =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
acc_kmeans =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
roc_kmeans =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
sen_kmeans, spec_kmeans =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_kmeans'])
met_kmeans = {
           'f1_score': f1_kmeans,
           'accuracy': acc_kmeans,
           'roc_score': roc_kmeans,
           'sensitivity': sen_kmeans,
           'specificity': spec_kmeans
          }

f1_svm =
        f1_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
acc_svm =
        accuracy_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
roc_svm =
        roc_auc_score(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
```

```
sen_svm, spec_svm =
        get_sensitivity_specificity(final_dataset['anomaly_manual'],final_dataset['anomaly_svm'])
met_svm = {
           'f1_score': f1_svm,
           'accuracy': acc_svm,
           'roc_score': roc_svm,
           'sensitivity': sen_svm,
           'specificity': spec_svm
          }

metrics = {'isolated_forest': met_iso,
           'kmeans': met_kmeans,
           'svm': met_svm
          }

all_metrics = pd.DataFrame.from_dict(metrics)

all_metrics
```

	isolated_forest	kmeans	svm
f1_score	0.303030	0.289855	0.270270
accuracy	0.880829	0.873057	0.860104
roc_score	0.589399	0.584867	0.577314
sensitivity	0.880000	0.879032	0.877384
specificity	0.909091	0.714286	0.526316

Ensemble models for Classification

```
from sklearn.preprocessing import LabelEncoder
final_dataset['ip_address'] =
        LabelEncoder().fit_transform(final_dataset.iloc[:, -1])

features = ['ip_address', 'total_count', 'daily_counts',
        'is_weekend_ratio', 'td_mean', 'td_max']
X = final_dataset[features]
y = final_dataset['anomaly_isolated']

X, y = final_dataset[features], final_dataset['anomaly_isolated']
X.shape, y.shape
```

```
((386, 6), (386,))
```

Baseline Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

rf = OneVsRestClassifier(estimator = GaussianNB())

# Create RepeatedStratifiedKFold cross-validator with 10 folds, 3
        repeats and a seed of 1.
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
        random_state=20)

# Calculating accuracy using `cross_val_score()` with model
        instantiated, data to fit,
# target variable, 'accuracy' scoring, cross validator, n_jobs=-1, and
        error_score set to 'raise'
n_scores = cross_val_score(rf, X, y, scoring='f1_weighted', cv=cv,
        error_score='raise')


# Print mean and standard deviation of n_scores:
print(n_scores)
print('Baseline scores \n mean f1 weighted: %.3f with a %.3f standard
        deviation in scores ' % (np.mean(n_scores),
        np.std(n_scores)))
```

```
[0.91306815 0.92877493 0.86485671 0.94457323 0.96084546 0.95655271
 0.96070175 0.91064618 0.91064618 0.87780396 0.92877493 0.83100233
 0.94457323 0.91306815 0.92877493 0.95655271 0.94306894 0.87780396
 0.94306894 0.96070175 0.96084546 0.91306815 0.91306815 0.89725275
 0.91306815 0.86824618 0.89434985 1.         0.94306894 0.96070175]
Baseline scores
 mean f1 weighted: 0.924 with a 0.036 standard deviation in scores
```

Stacking Classifier

Stacking classifier takes:

1. estimators: List of baseline classifiers
2. final_estimator: Defined meta classifier
3. cv: Number of cross validations to perform

```
def create_stacking_models():
    base_models = list()
    base_models.append(('KNNC', KNeighborsClassifier(n_neighbors =
        len(np.unique(y))
                                                         , weights =
        'distance')
                  ))
    base_models.append(('SVC', SVC(kernel = 'linear'
                                   , class_weight = 'balanced'
                                  , break_ties = True)
                  ))
    base_models.append(('GNB', GaussianNB()))
    base_models.append(('RF', RandomForestClassifier(n_estimators=
        200,
                                                       oob_score = True,
                                                       class_weight =
        "balanced",
                                                       random_state = 20,
                                                       ccp_alpha = 0.1)
                  ))



    meta_model = LogisticRegression()
    final_model = StackingClassifier(estimators = base_models, ##Base
        estimators which will be stacked together
                                     final_estimator = meta_model,
                                     cv = 5
                                    )
    return final_model


def models_all():
    all_models = dict()
    all_models['KNNC']= KNeighborsClassifier(n_neighbors =
        len(np.unique(y))
```

```
                                                , weights = 'distance')
    all_models['SVC']= SVC(kernel = 'linear'
                           , class_weight = 'balanced'
                          , break_ties = True
                          )
    all_models['RF']= RandomForestClassifier(n_estimators= 200,
                                                   oob_score = True,
                                                   class_weight =
        "balanced",
                                                   random_state = 20,
                                                   ccp_alpha = 0.15)
    all_models['GNB'] = GaussianNB()
    all_models['Stacking'] = create_stacking_models()
    return all_models

def evaluate_model(model):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
        random_state=42)
    scores = cross_val_score(model, X, y, scoring='f1_weighted',
        cv=cv, error_score='raise')
    return scores

model_results = list()
models = models_all()
names = list()

# Creating a for loop that iterates over each name, model in models
        dictionary
for name, model in models.items():
    scores = evaluate_model(model)
    model_results.append(scores)
    names.append(name)
#   print(model_results)
    print('>%s %.3f (%.3f) \n' % (name, np.mean(scores),
        np.std(scores)))
```

```
>KNNC 0.994 (0.012)

>SVC 0.996 (0.011)

>RF 0.996 (0.011)

>GNB 0.921 (0.035)

>Stacking 0.996 (0.011)
```

```
# Flatten the data for the boxplot
flat_names = np.repeat(names, [len(r) for r in model_results])
flat_results = np.concatenate(model_results)

# Create the boxplot with reshaped data
plt.figure(figsize=(15,5))
sns.boxplot(x=flat_names, y=flat_results, showmeans=True)
plt.title("Model Comparisons")
plt.show()
```

Conclusion

- With a stacking model, we were able to get an F1 score of 96.7%, which was greater than the target of 81.1%