Multivariate Anomaly Detection with PYOD

Author:
Hussain Abbas, MSc

Part I:

Data Preparation and Preprocessing

Github Code Link:

https://github.com/StatsAI/Anomaly-Detection/blob/main/Part%20I%20-%20Data%20Preparation%20%26%20Preprocessing.ipynb

1. Use **python** to load DEMO_D.csv, BPX_D.csv, and TCHOL_D.csv as **pandas** data frames. Merge the 3 datasets but keep only the records that appear in all 3 datasets.

2. The existing features RIDAGEMN and RIDAGEEX contain the age of the respondent at screening and examination, respectively; however, both contain some missing values.

    a. Create a new feature AGE_AT_SCREENING with no missing values that contains the most precise estimate of each respondent's age (in months) at the time of screening. Choose an appropriate strategy to estimate the missing values **and** explain your reasoning.

    b. Create a second feature AGE_AT_EXAM with no missing values that contains the most precise estimate of each respondent's age (in months) at the time of the follow-up examination. Choose an appropriate strategy to estimate the missing values and explain your reasoning.

3. The existing features DMDEDUC3 and DMDEDUC2 contain categorical responses for each respondent's highest level of education completed: however, the categories are somewhat overlapping and complicated.

    a. Create a new categorical feature HIGHEST_EDUCATION with the following categories: ELEMENTARY (did not graduate HS, or currently in grades K-12), HIGHSCHOOL (graduated or GED), and COLLEGE (4 year graduates only). This feature should reflect the highest level of education **completed** for each respondent, from among the 3 options. Choose an appropriate strategy to fill in any missing values and explain your reasoning.

4. The file DEMO_RETIRED.CSV contains a single feature named RETIRED which is a binary flag indicating whether the respondent is retired (1) or not (0). This feature has some missing values.

    a. Suggest an appropriate strategy to fill in the missing values. Justify your approach using graphs or statistics.

In [300…

```python
# Author: Hussain Abbas, MSc
# © 2021 Stats AI LLC
# All Rights Reserved

import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer
```

In [301…

```python
# read in the files into seperate dataframes and inspect

df1 = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/BPX_D.csv')
df2 = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/DEMO_D.csv')
df3 = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/TCHOL_D.csv')
df4 = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/DEMO_RETIRED.csv')

display(df1.head(), df2.head(), df3.head())
```

	SEQN	PEASCST1	PEASCTM1	PEASCCT1	BPXCHR	BPQ150A
0	31127.0	1.0	41.0	NaN	100.0	NaN
1	31128.0	1.0	401.0	NaN	NaN	2.0
2	31129.0	1.0	664.0	NaN	NaN	1.0
3	31130.0	2.0	NaN	4.0	NaN	2.0
4	31131.0	1.0	827.0	NaN	NaN	2.0

	SEQN	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEMN	RIDAGEEX	RIDAGEYR	RIDRETH1	DMDEDUC3	DMDEDUC2	DMDSCHOL
0	31127.0	4.0	2.0	1.0	11.0	12.0	5.397605e-79	3.0	NaN	NaN	NaN
1	31128.0	4.0	2.0	2.0	132.0	132.0	1.100000e+01	4.0	4.0	NaN	1.0
2	31129.0	4.0	2.0	1.0	189.0	190.0	1.500000e+01	4.0	10.0	NaN	1.0
3	31130.0	4.0	2.0	2.0	NaN	NaN	8.500000e+01	3.0	NaN	4.0	NaN
4	31131.0	4.0	2.0	2.0	535.0	536.0	4.400000e+01	4.0	NaN	4.0	NaN

	SEQN	LBXTC	LBDTCSI
0	31128.0	129.0	3.34
1	31129.0	170.0	4.40
2	31130.0	NaN	NaN
3	31131.0	105.0	2.72
4	31132.0	147.0	3.80

In [302…

```python
# merge the three dataframes into one dataframe

df = df1.merge(df2, left_on = 'SEQN', right_on = 'SEQN', how='inner')
df = df.merge(df3, left_on = 'SEQN', right_on = 'SEQN', how='inner')
df_names = df.columns

# convert categorical variables to categorical type

df['SEQN'] = df['SEQN'].astype("category")
df['PEASCST1'] = df['PEASCST1'].astype("category")
df['PEASCCT1'] = df['PEASCCT1'].astype("category")
df['BPQ150A'] = df['BPQ150A'].astype("category")
df['SDDSRVYR'] = df['SDDSRVYR'].astype("category")
df['RIDSTATR'] = df['RIDSTATR'].astype("category")
df['RIAGENDR'] = df['RIAGENDR'].astype("category")
df['RIDAGEYR'] = df['RIDAGEYR'].astype("category")
df['RIDRETH1'] = df['RIDRETH1'].astype("category")
df['DMDEDUC3'] = df['DMDEDUC3'].astype("category")
df['DMDEDUC2'] = df['DMDEDUC2'].astype("category")
df['DMDSCHOL'] = df['DMDSCHOL'].astype("category")

df
```

Out[302…

	SEQN	PEASCST1	PEASCTM1	PEASCCT1	BPXCHR	BPQ150A	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEMN	RIDAGEEX	RIDAGEYR
0	31128.0	1.0	401.0	NaN	NaN	2.0	4.0	2.0	2.0	132.0	132.0	11.0
1	31129.0	1.0	664.0	NaN	NaN	1.0	4.0	2.0	1.0	189.0	190.0	15.0
2	31130.0	2.0	NaN	4.0	NaN	2.0	4.0	2.0	2.0	NaN	NaN	85.0
3	31131.0	1.0	827.0	NaN	NaN	2.0	4.0	2.0	2.0	535.0	536.0	44.0
4	31132.0	1.0	730.0	NaN	NaN	2.0	4.0	2.0	1.0	842.0	843.0	70.0
...	...	...	...	...	...	...	...	...	...	...	...	...
8081	41469.0	1.0	715.0	NaN	NaN	2.0	4.0	2.0	1.0	235.0	235.0	19.0
8082	41471.0	1.0	703.0	NaN	NaN	2.0	4.0	2.0	1.0	148.0	149.0	12.0
8083	41472.0	1.0	567.0	NaN	NaN	2.0	4.0	2.0	1.0	410.0	411.0	34.0
8084	41473.0	1.0	645.0	NaN	NaN	2.0	4.0	2.0	1.0	255.0	255.0	21.0
8085	41474.0	1.0	579.0	NaN	NaN	2.0	4.0	2.0	2.0	200.0	200.0	16.0

8086 rows × 18 columns

In [303…

```python
# We are going to use the KNN imputer with 3 nearest neighbors to impute missing values for all numeric columns
# This is a reasonable approach to filling in missing values since it will estimate the missing information fro
# the three closest data vectors. This mitgates the chance that we fill in missing data with irrelevant informa

df_numeric_columns = df.select_dtypes(include='number')

# drop the numeric columns that contain missing data from df
columns_to_drop = list(df_numeric_columns.columns)
df.drop(columns_to_drop, axis=1, inplace = True)

imputer = KNNImputer(n_neighbors=3)
df_imputed = pd.DataFrame(imputer.fit_transform(df_numeric_columns),
                          columns = df_numeric_columns.columns)

df_imputed.rename(columns = {'RIDAGEMN':'AGE_AT_SCREENING',
                             'RIDAGEEX':'AGE_AT_EXAM'}, inplace = True)

# diplay the numeric columns, before and after using KNN imputation
display(df_numeric_columns)
display(df_imputed)

# display the quantiles for the variables of interest
display(df_imputed.AGE_AT_SCREENING.quantile([0, 0.025, 0.5, 0.975, 1]))
display(df_imputed.AGE_AT_EXAM.quantile([0, 0.025, 0.5, 0.975, 1]))

# display the histograms for the variables of interest
display(sns.displot(df_imputed.AGE_AT_SCREENING))
display(sns.displot(df_imputed.AGE_AT_EXAM))

# update df with the numeric data columns that we filled in using KNN
df = df.merge(df_imputed, left_index=True, right_index=True, how = 'inner')
```

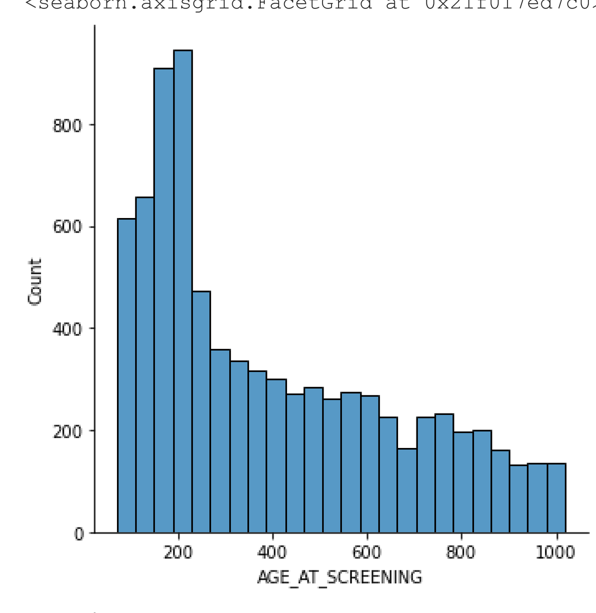
	PEASCTM1	BPXCHR	RIDAGEMN	RIDAGEEX	LBXTC	LBDTCSI
0	401.0	NaN	132.0	132.0	129.0	3.34
1	664.0	NaN	189.0	190.0	170.0	4.40
2	NaN	NaN	NaN	NaN	NaN	NaN
3	827.0	NaN	535.0	536.0	105.0	2.72
4	730.0	NaN	842.0	843.0	147.0	3.80
...	...	...	...	...	...	...
8081	715.0	NaN	235.0	235.0	193.0	4.99
8082	703.0	NaN	148.0	149.0	142.0	3.67
8083	567.0	NaN	410.0	411.0	140.0	3.62
8084	645.0	NaN	255.0	255.0	184.0	4.76
8085	579.0	NaN	200.0	200.0	101.0	2.61

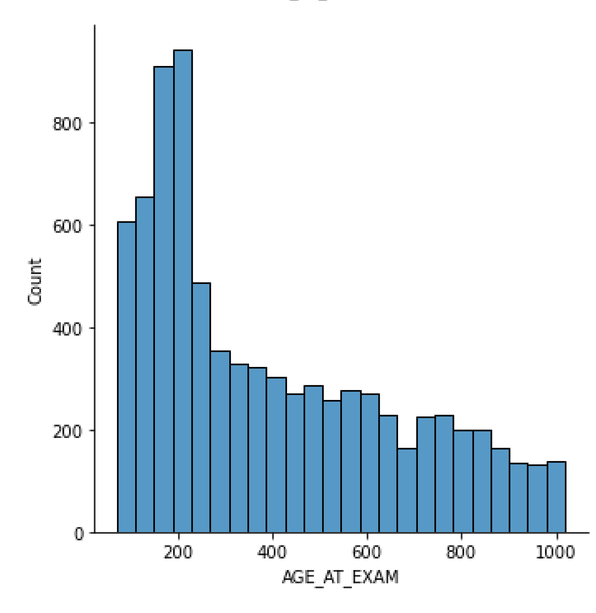
8086 rows × 6 columns

	PEASCTM1	BPXCHR	AGE_AT_SCREENING	AGE_AT_EXAM	LBXTC	LBDTCSI
0	401.000000	78.666667	132.000000	132.000000	129.00000	3.340000
1	664.000000	78.666667	189.000000	190.000000	170.00000	4.400000
2	612.427931	88.550562	400.234316	400.949476	184.47106	4.770417
3	827.000000	78.666667	535.000000	536.000000	105.00000	2.720000
4	730.000000	78.666667	842.000000	843.000000	147.00000	3.800000
...	...	...	...	...	...	...
8081	715.000000	78.666667	235.000000	235.000000	193.00000	4.990000
8082	703.000000	78.666667	148.000000	149.000000	142.00000	3.670000
8083	567.000000	78.666667	410.000000	411.000000	140.00000	3.620000
8084	645.000000	76.666667	255.000000	255.000000	184.00000	4.760000
8085	579.000000	78.666667	200.000000	200.000000	101.00000	2.610000

8086 rows × 6 columns

```
0.000      72.0
0.025      84.0
0.500     318.0
0.975     962.0
1.000    1019.0
Name: AGE_AT_SCREENING, dtype: float64
0.000      72.0
0.025      85.0
0.500     319.0
0.975     963.0
1.000    1019.0
Name: AGE_AT_EXAM, dtype: float64
<seaborn.axisgrid.FacetGrid at 0x21f046faa90>
<seaborn.axisgrid.FacetGrid at 0x21f017ed7c0>
```





In [305…

```python
# We use a dictionary to group the labels in DMDEDUC2 and DMDEDUC3 into logical categories
# Since users are in either one group or the other, data may be missing simply due to being a member of the oth
# These new columns will then be collapsed into a new education variable reflecting the person's HIGHEST_EDUCA
# The missing data that remains in HIGHEST EDUCATION will be true missing data which we fill in using the mode.

d = {1: 'ELEMENTARY', 2: 'ELEMENTARY',
     3: 'HIGH SCHOOL', 4: 'HIGH SCHOOL',
     5: 'COLLEGE',
     7: np.nan, 9: np.nan}

df["DMDEDUC2_NEW"] = df["DMDEDUC2"].map(d)

display(df['DMDEDUC2'].value_counts(dropna=False))
display(df['DMDEDUC2_NEW'].value_counts(dropna=False))
```

```
NaN    3313
4.0    1361
3.0    1136
5.0     939
2.0     733
1.0     597
7.0       4
9.0       3
Name: DMDEDUC2, dtype: int64
NaN            3320
HIGH SCHOOL    2497
ELEMENTARY     1330
COLLEGE         939
Name: DMDEDUC2_NEW, dtype: int64
```

In [306…

```python
# We use a dictionary to group the labels in DMDEDUC2 and DMDEDUC3 into logical categories
# Since users are in either one group or the other, data may be missing simply due to being a member of the oth
# These new columns will then be collapsed into a new education variable reflecting the person's HIGHEST_EDUCA
# The missing data that remains in HIGHEST EDUCATION will be true missing data which we fill in using the mode.

d = {1: 'ELEMENTARY', 2: 'ELEMENTARY', 3: 'ELEMENTARY', 4: 'ELEMENTARY', 5: 'ELEMENTARY', 6: 'ELEMENTARY',
     7: 'ELEMENTARY', 8: 'ELEMENTARY', 9: 'ELEMENTARY', 10: 'ELEMENTARY', 11: 'ELEMENTARY', 12: 'ELEMENTARY',
     13: 'HIGH SCHOOL', 14: 'HIGH SCHOOL', 15: 'HIGH SCHOOL',
     55: 'ELEMENTARY', 66:'ELEMENTARY'}

df["DMDEDUC3_NEW"] = df["DMDEDUC3"].map(d)

display(df['DMDEDUC3'].value_counts(dropna=False))
display(df['DMDEDUC3_NEW'].value_counts(dropna=False))
```

```
NaN                      4774
10.0                      300
8.0                       294
9.0                       282
6.0                       280
11.0                      264
7.0                       255
5.397605346934027e-79     232
13.0                      204
3.0                       204
5.0                       201
4.0                       197
1.0                       185
2.0                       182
15.0                      139
12.0                       46
66.0                       36
14.0                       10
99.0                        1
Name: DMDEDUC3, dtype: int64
NaN            5007
ELEMENTARY     2726
HIGH SCHOOL     353
Name: DMDEDUC3_NEW, dtype: int64
```

In [307…

```python
# observe the na's in each variable before combining them
display(df[['DMDEDUC2_NEW', 'DMDEDUC3_NEW']].head())

df['HIGHEST_EDUCATION'] = df['DMDEDUC3_NEW'].fillna('') + df['DMDEDUC2_NEW'].fillna('')

# observe how HIGHEST_EDUCATION combines the two columns
display(df[['DMDEDUC2_NEW', 'DMDEDUC3_NEW', 'HIGHEST_EDUCATION']].head())

# We drop the original and scaffolding columns since we no longer need them
df.drop(['DMDEDUC2', 'DMDEDUC3', 'DMDEDUC2_NEW', 'DMDEDUC3_NEW'], axis=1, inplace = True)

# Now we replace the remaining missing data with the mode
df.HIGHEST_EDUCATION[df.HIGHEST_EDUCATION == ''] = df.HIGHEST_EDUCATION.mode()[0]

# verify that the missing data in HIGHEST_EDUCATION has been replaced with the mode (which is ELEMENTARY)
display(df.HIGHEST_EDUCATION.value_counts(dropna=False))
```

	DMDEDUC2_NEW	DMDEDUC3_NEW
0	NaN	ELEMENTARY
1	NaN	ELEMENTARY
2	HIGH SCHOOL	NaN
3	HIGH SCHOOL	NaN
4	COLLEGE	NaN

	DMDEDUC2_NEW	DMDEDUC3_NEW	HIGHEST_EDUCATION
0	NaN	ELEMENTARY	ELEMENTARY
1	NaN	ELEMENTARY	ELEMENTARY
2	HIGH SCHOOL	NaN	HIGH SCHOOL
3	HIGH SCHOOL	NaN	HIGH SCHOOL
4	COLLEGE	NaN	COLLEGE

```
<ipython-input-307-e6ac9ff31878>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df.HIGHEST_EDUCATION[df.HIGHEST_EDUCATION == ''] = df.HIGHEST_EDUCATION.mode()[0]
ELEMENTARY     4297
HIGH SCHOOL    2850
COLLEGE         939
Name: HIGHEST_EDUCATION, dtype: int64
```

In [308…

```python
display(df.head())

# merge all four dataframes into one dataframe to see if we can find additional
# data to better estimate the missing values in the RETIREMENT field

df_new = df.copy()

df_new = df_new.merge(df4, left_on = 'SEQN', right_on = 'SEQN', how ='right')

df_new['RETIRED'] = df_new['RETIRED'].astype("category")

display(df_new.head())
```

	SEQN	PEASCST1	PEASCCT1	BPQ150A	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDRETH1	DMDSCHOL	PEASCTM1	BPXCHR
0	31128.0	1.0	NaN	2.0	4.0	2.0	2.0	11.0	4.0	1.0	401.000000	78.666667
1	31129.0	1.0	NaN	1.0	4.0	2.0	1.0	15.0	4.0	1.0	664.000000	78.666667
2	31130.0	2.0	4.0	2.0	4.0	2.0	2.0	85.0	3.0	NaN	612.427931	88.550562
3	31131.0	1.0	NaN	2.0	4.0	2.0	2.0	44.0	4.0	NaN	827.000000	78.666667
4	31132.0	1.0	NaN	2.0	4.0	2.0	1.0	70.0	3.0	NaN	730.000000	78.666667

	SEQN	PEASCST1	PEASCCT1	BPQ150A	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDRETH1	DMDSCHOL	PEASCTM1	BPXCHR
0	31127.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	31128.0	1.0	NaN	2.0	4.0	2.0	2.0	11.0	4.0	1.0	401.000000	78.666667
2	31129.0	1.0	NaN	1.0	4.0	2.0	1.0	15.0	4.0	1.0	664.000000	78.666667
3	31130.0	2.0	4.0	2.0	4.0	2.0	2.0	85.0	3.0	NaN	612.427931	88.550562
4	31131.0	1.0	NaN	2.0	4.0	2.0	2.0	44.0	4.0	NaN	827.000000	78.666667

In [309…

```python
# Non Retireds have a max AGE_AT_EXAM  of 852.
# Thus, we could say that anyone with an Age_AT_EXAM > 852 could be a retired person
display(df_new.groupby('RETIRED', dropna=False).quantile([0, 0.025, 0.5, 0.975, 1]))

# However, we don't seem to have any data at all on the SEQN's with missing data for Retirement
# Thus, it seems we are out of luck and must resort to using the mode to fill in the missing data for Retiremen
display(df_new[df_new['RETIRED'].isna()])

# Check value counts before replacing missing data
display(df4.RETIRED.value_counts(dropna=False))

# Now we replace the missing data with the mode
df4.RETIRED[df4.RETIRED.isna()] = df4.RETIRED.mode()[0]

# Verify that the missing data has been filled in with the mode
display(df4.RETIRED.value_counts(dropna=False))
```

		SEQN	PEASCTM1	BPXCHR	AGE_AT_SCREENING	AGE_AT_EXAM	LBXTC	LBDTCSI
RETIRED								
0.0	0.000	31128.0	5.0	58.000000	72.000000	72.0	78.0	2.020
	0.025	31379.6	22.0	76.000000	82.000000	83.0	118.0	3.050
	0.500	36282.0	626.0	78.666667	267.000000	268.0	177.0	4.580
	0.975	41202.2	945.2	94.000000	775.000000	776.2	279.0	7.210
	1.000	41474.0	2679.0	240.000000	851.000000	852.0	615.0	15.900
1.0	0.000	31130.0	9.0	73.333333	81.333333	82.0	84.0	2.170
	0.025	31385.3	33.3	76.666667	221.300000	224.1	120.0	3.100
	0.500	36569.0	668.0	78.666667	878.000000	878.0	190.0	4.910
	0.975	41211.6	1024.7	91.333333	1011.700000	1011.9	279.7	7.231
	1.000	41461.0	1733.0	104.000000	1019.000000	1019.0	345.0	8.920

	SEQN	PEASCST1	PEASCCT1	BPQ150A	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDRETH1	DMDSCHOL	PEASCTM1	BPXCHR
0	31127.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	31135.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	31136.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	31138.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
38	31165.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
10316	41443.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10328	41455.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10329	41456.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10336	41463.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10343	41470.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2262 rows × 18 columns

```
0.0    7073
NaN    2262
1.0    1013
Name: RETIRED, dtype: int64
0.0    9335
1.0    1013
Name: RETIRED, dtype: int64
```

In [ ]:

Part II:

Model Building

Github Code Link:

https://github.com/StatsAI/Anomaly-Detection/blob/main/Part%202%20-%20Cluster%20Analysis.ipynb

An auto-insurance company is revamping its pricing model. The analyst developing the new price model believes that the best approach is to develop several models based on groups of similar drivers. You have been contracted to develop a model to define groups of drivers based on driving behavior. The analyst has prepared a dataset (*driver_behavior.csv*) consisting of 10,000 customers and 5 engineered features which capture driving behavior. The data has already been preprocessed for you (i.e., no missing data, no outliers, data is scaled, and no correlated features).

Develop a model to group the drivers and comment on the quality of the generated groups using an appropriate metric or visualization. How many different groups of drivers did you create from the provided data?

In [30]:
```python
# Author: Hussain Abbas, MSc
# © 2021 Stats AI LLC
# All Rights Reserved

import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
```

In [31]:
```python
df = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/driver_behavior.csv')

df
```

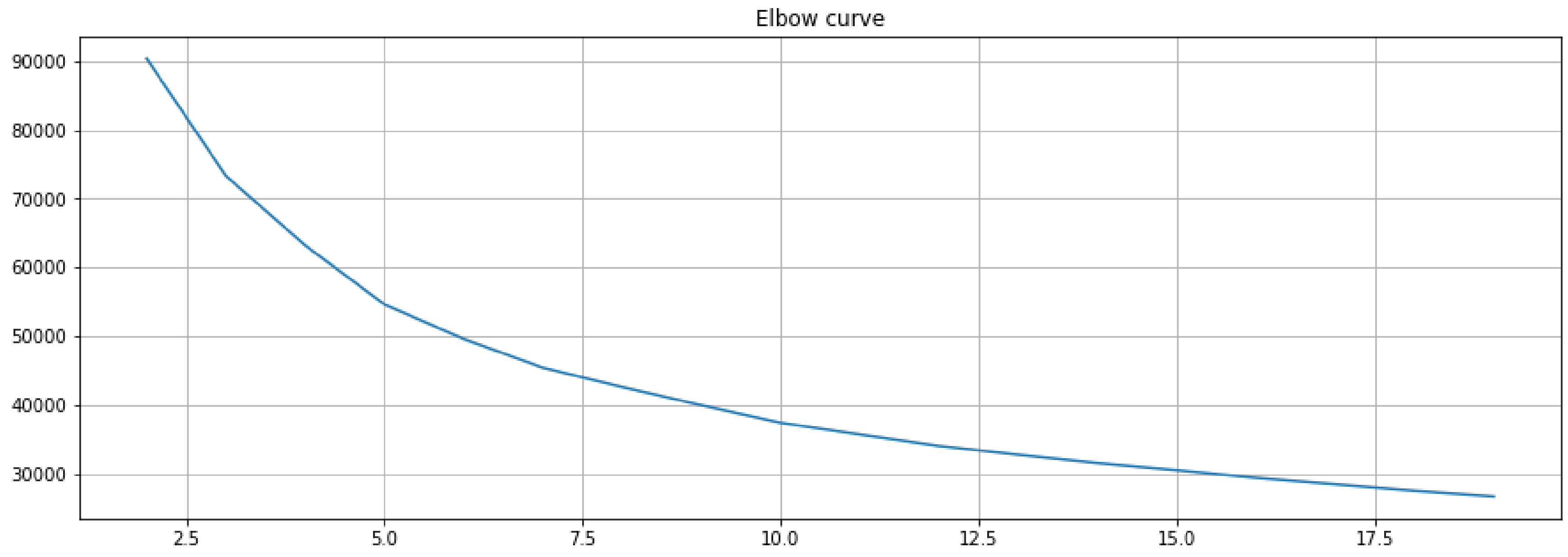
Out[31]:

	FAXEX	BCACE	DBBAB	AAXFE	BXCEC
0	-0.893210	-2.080530	1.581359	-0.739197	-0.115923
1	-0.279951	2.575102	0.329756	0.248520	0.324917
2	-3.898434	-0.024275	-1.728197	4.992603	2.395236
3	-3.120864	-3.031472	-0.021979	2.755907	1.216255
4	-1.049326	0.953137	-1.905779	-1.566512	2.589435
...	...	...	...	...	...
9995	-0.618424	-2.945030	2.429603	1.786126	1.497155
9996	-0.971600	1.545476	0.379079	-0.642436	0.262755
9997	0.045780	-2.214428	2.672059	3.002813	-0.052627
9998	-1.849218	-1.168485	-1.120131	2.010948	1.427708
9999	2.085431	-4.700154	2.802765	3.767104	1.717923

10000 rows × 5 columns

In [32]:
```python
# We will use the Kmeans method for clustering since it is a good starting point for unsupervised learning.
# Clustering is an iterative process: Generally we use some method to get a set of clusters
# We then take some action that depends upon the clusters, i.e., build models that use the cluster information
# We can refine our "targeting" via clusters by better modeling the data generating process
# Thus, if our clusters are not spherical, we could potentially refine our targeting with a method like DBSCAN

# The following loop enables us to compute the optimal number of clusters using the elbow method.

distorsions = []

for k in range(2, 20):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(df)
    distorsions.append(kmeans.inertia_)

fig = plt.figure(figsize=(15, 5))
plt.plot(range(2, 20), distorsions)
plt.grid(True)
plt.title('Elbow curve')

# From the elbow curve, it seems the optimal number of clusters is between 6 and 10.
# Kmeans with 6 clusters is a widely known heuristic.
# After 10 clusters, we seem to be getting dimishing returns, so we'll pick 10
# to be the optimal number of clusters.
```

Out[32]: Text(0.5, 1.0, 'Elbow curve')



In [33]:
```python
# create our new kmeans model with 10 clusters as identified using the elbow plot

kmeans = KMeans(n_clusters = 10)
kmeans.fit(df)

# add the cluster labels to the dataframe
df['cluster_labels'] = kmeans.labels_

# show the new dataframe
display(df)

# show the percentage of data corresponding to each cluster
df['cluster_labels'].value_counts() / len(df['cluster_labels'])
```

	FAXEX	BCACE	DBBAB	AAXFE	BXCEC	cluster_labels
0	-0.893210	-2.080530	1.581359	-0.739197	-0.115923	3
1	-0.279951	2.575102	0.329756	0.248520	0.324917	5
2	-3.898434	-0.024275	-1.728197	4.992603	2.395236	9
3	-3.120864	-3.031472	-0.021979	2.755907	1.216255	9
4	-1.049326	0.953137	-1.905779	-1.566512	2.589435	7
...	...	...	...	...	...	...
9995	-0.618424	-2.945030	2.429603	1.786126	1.497155	4
9996	-0.971600	1.545476	0.379079	-0.642436	0.262755	5
9997	0.045780	-2.214428	2.672059	3.002813	-0.052627	4
9998	-1.849218	-1.168485	-1.120131	2.010948	1.427708	9
9999	2.085431	-4.700154	2.802765	3.767104	1.717923	4

10000 rows × 6 columns

Out[33]:
```
7    0.1486
3    0.1217
9    0.1047
6    0.0993
1    0.0988
5    0.0930
0    0.0919
8    0.0827
2    0.0825
4    0.0768
Name: cluster_labels, dtype: float64
```

In [ ]:

Part III:

Model Evaluation

1. What is one way to determine the number of clusters in K-Means clustering? How would you estimate the efficacy, or quality, of the K-means clustering results?

① We can use the % of explained variation to set the number of clusters. Usualy a number around 6-10 is reasonble. After this, you usualy start to see diminishing returns. The efficacy of the cluster result can be measud via how useful knowledge of cluster membership is in improved decision making.

2. Your linear regression model is suffering from low bias and high variance. What steps can you take to improve your model?

② Variance can be reduced via methods like Lasso regression, which shrinks coeffients to zero. This has the effect of exchanging a slight increase in bias for a larger decrease in variance.

3. Below is a scenario for training error (TE) and validation error (VE) for several iterations of a machine learning model. Which model would you choose, and why?

Model	TE	VE
1	105	90
2	200	85
3	250	96
4	105	85
5	300	100

(3) We first find the models which minimize VE and then select the one that has the smallest TE.
Thus, Model 4 is the best model.

4. You have built a model for a binary classification problem. The trained model was applied to the validation dataset and produced the results documented in the following confusion matrix.

n = 263		Predicted N	Predicted Y
Actual	N	97	48
Actual	Y	6	112

a. Calculate Recall, Precision and F-1 score.

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{112}{112+6} = \frac{112}{118} = \boxed{0.95}$$

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{112}{112+48} = \frac{112}{160} = \boxed{0.7}$$

$$\text{F1-Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \boxed{0.81}$$

b. If your classifier model is attempting to predict cancer in patients. Which type of error should you focus on for this type of problem? Which evaluation metric would you choose and why?

4b For Cancer Prediction, we want to minimize False Negatives. Thus, we want maximize Recall. This is due to the fact that a False Negative Means we claim a person is healthy when they are really dying wheras a False Positive means we claim a person is sick when they are really healthy. This may cause an inconvenience with more testing, etc.

However, if the costs of FP's are higher (i.e., more invaisive tests are needed) then we should maximize both Recall and Precision, and thefore maximize F1-Score. However, recall should be weighted higher, so we would want to maximize F1-Beta Score with say Beta = 2.

c. If your classifier model is attempting to determine whether or not to recommend a YouTube video. Which type of error should you focus on for this type of problem? Which evaluation metric would you choose and why?

4c For Youtube recs, a False negative means we fail to recommend a video the user wolud have liked. A FP means we recommend a video that the user doesn't like. Thus, the consequnces of a FP are a lot worse than a FN, since the user has direct feedback in the case of a FP (the rec algo is bad) wheras the user has no knowlege of if the rec algo is failing to recommend them content they would have liked.

For this reason, Metrics like Precision @ k are typically used as a starting point for evaluating recommender systems.

Thus, we want maximize Precision @ k.

Part IV:

Anomaly Detection

Github Code Link:

https://github.com/StatsAI/Anomaly-Detection/blob/main/Part%203%20-%20Multivariate%20Anomaly%20Detection%20with%20PYOD.ipynb

In [2]:
```python
# Author: Hussain Abbas, MSc
# © 2021 Stats AI LLC
# All Rights Reserved

import pandas as pd
import seaborn as sns
from pyod.models.copod import COPOD
from datetime import timedelta
```

In [3]:
```python
df = pd.read_csv('C:/Users/deepl/Desktop/Anomaly_Detection Project/anomaly_detection.csv')

dti = pd.date_range("2016-01-01", periods=365, freq="D")

df.index = dti

X_train = df[df.index <= '2016-09-30']

X_test = df[df.index > '2016-09-30']

display(X_train, X_test)
```

	CEACC	CCDEF	FAXAE	FBFFD	EDDAB
2016-01-01	0.000000	0.000000	3.000000	3.000000	8.000000
2016-01-02	1.232260	-0.217305	3.193780	3.489992	7.792553
2016-01-03	2.540129	0.606709	3.180950	3.464811	7.817440
2016-01-04	5.520271	1.470812	3.330528	2.968375	7.409152
2016-01-05	5.914298	2.745060	3.061706	3.440980	7.153049
...	...	...	...	...	...
2016-09-26	14.133139	7.947718	1.800914	6.852511	9.812531
2016-09-27	13.673763	7.091657	1.611077	6.991557	9.378149
2016-09-28	12.138013	6.593396	1.228136	6.763265	9.575867
2016-09-29	11.702856	6.147422	0.825552	6.746459	9.643524
2016-09-30	10.555822	5.669821	0.454363	6.398245	9.293962

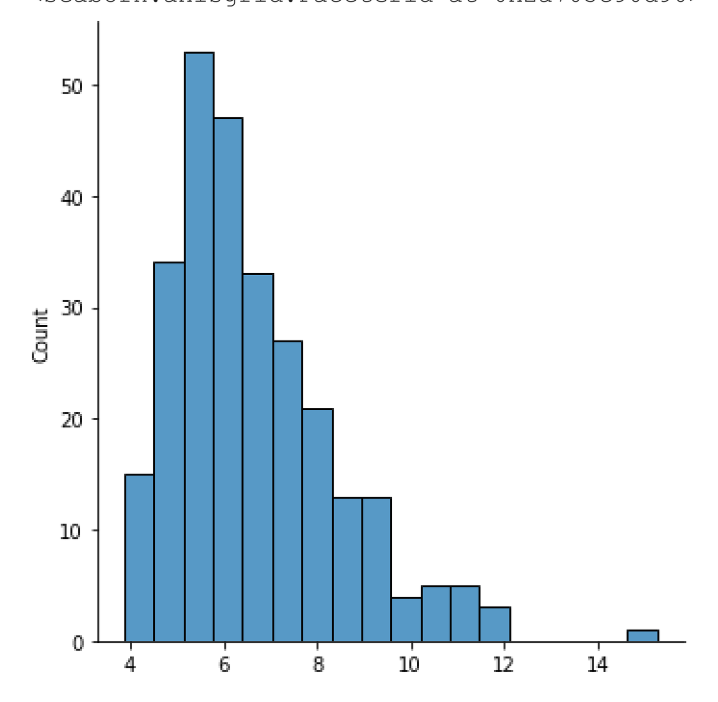
274 rows × 5 columns

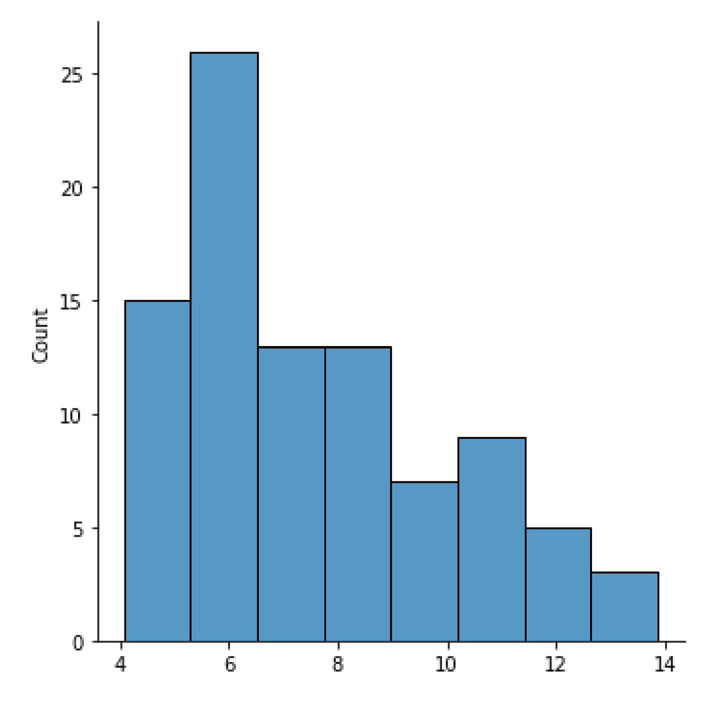
	CEACC	CCDEF	FAXAE	FBFFD	EDDAB
2016-10-01	8.046032	5.295155	0.490067	5.972950	9.221123
2016-10-02	8.769270	4.184700	0.815883	5.476941	8.813725
2016-10-03	8.311614	4.632987	0.679195	5.056123	8.552767
2016-10-04	7.850020	3.950758	0.800483	5.045620	8.481428
2016-10-05	8.176031	3.795881	0.791072	5.016191	8.465436
...	...	...	...	...	...
2016-12-26	11.364339	-0.925751	-3.259216	6.572904	11.452315
2016-12-27	11.316572	-0.081164	-3.746738	6.252706	11.859704
2016-12-28	10.841367	-0.317127	-3.680397	6.617491	11.644486
2016-12-29	11.585932	-0.710426	-4.046116	6.519041	11.756765
2016-12-30	11.991952	-0.399588	-4.436065	6.155139	12.052701

91 rows × 5 columns

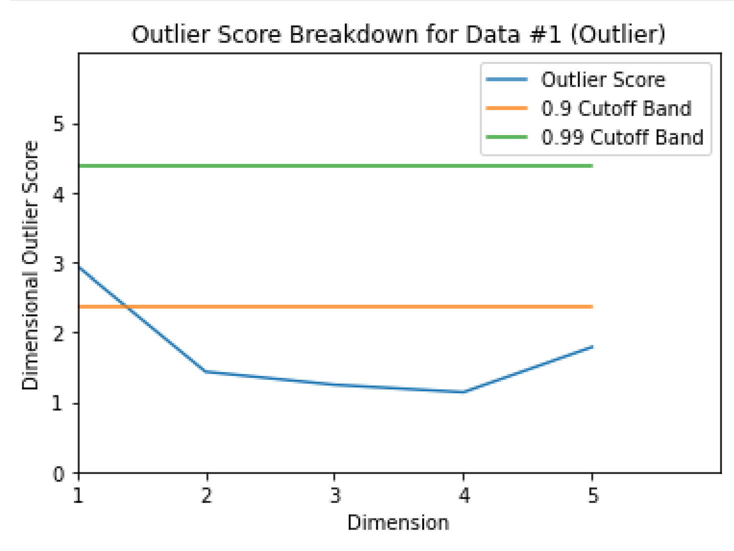
In [4]:
```python
# train the COPOD detector
clf = COPOD()
clf.fit(X_train)

# get raw outlier scores
y_train_scores = clf.decision_scores_
y_test_scores = clf.decision_function(X_test)

# get binary labels (0: inliers, 1: outliers)
y_train_pred = clf.labels_
y_test_pred = clf.predict(X_test)

# print outlier score distribution
display(sns.displot(y_train_scores))
display(sns.displot(y_test_scores))
```

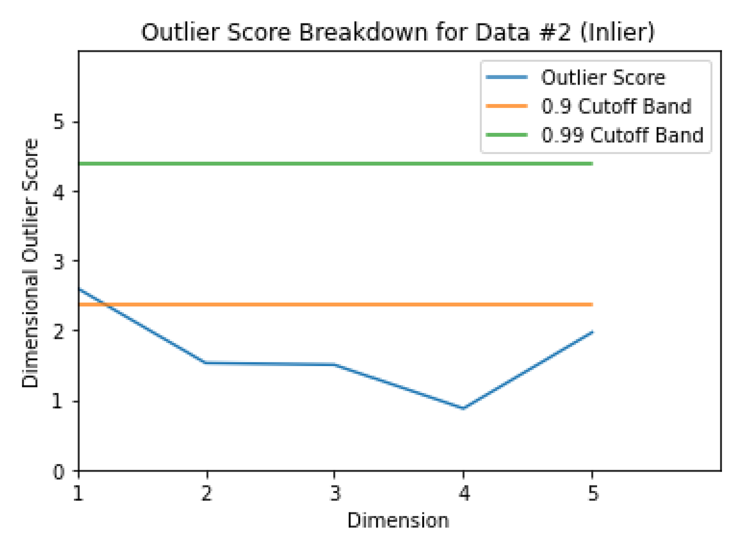
```
<seaborn.axisgrid.FacetGrid at 0x2a703c97640>
<seaborn.axisgrid.FacetGrid at 0x2a703c90d90>
```





In [5]:
```python
a, b, c = clf.explain_outlier(0)
a, b, c = clf.explain_outlier(1)
```





In [6]:
```python
X_train['outlier'] = y_train_pred
X_train['outlier_score'] = y_train_scores

X_test['outlier'] = y_test_pred
X_test['outlier_score'] = y_test_scores


display(X_train, X_test)
```

```
<ipython-input-6-b08fb2e615b8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  X_train['outlier'] = y_train_pred
<ipython-input-6-b08fb2e615b8>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  X_train['outlier_score'] = y_train_scores
<ipython-input-6-b08fb2e615b8>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  X_test['outlier'] = y_test_pred
<ipython-input-6-b08fb2e615b8>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  X_test['outlier_score'] = y_test_scores
```

	CEACC	CCDEF	FAXAE	FBFFD	EDDAB	outlier	outlier_score
2016-01-01	0.000000	0.000000	3.000000	3.000000	8.000000	1	9.449444
2016-01-02	1.232260	-0.217305	3.193780	3.489992	7.792553	0	8.915368
2016-01-03	2.540129	0.606709	3.180950	3.464811	7.817440	0	8.015674
2016-01-04	5.520271	1.470812	3.330528	2.968375	7.409152	0	8.268666
2016-01-05	5.914298	2.745060	3.061706	3.440980	7.153049	0	7.178465
...	...	...	...	...	...	...	...
2016-09-26	14.133139	7.947718	1.800914	6.852511	9.812531	0	7.209228
2016-09-27	13.673763	7.091657	1.611077	6.991557	9.378149	0	7.176601
2016-09-28	12.138013	6.593396	1.228136	6.763265	9.575867	0	6.276495
2016-09-29	11.702856	6.147422	0.825552	6.746459	9.643524	0	6.130867
2016-09-30	10.555822	5.669821	0.454363	6.398245	9.293962	0	5.735883

274 rows × 7 columns

	CEACC	CCDEF	FAXAE	FBFFD	EDDAB	outlier	outlier_score
2016-10-01	8.046032	5.295155	0.490067	5.972950	9.221123	0	5.285119
2016-10-02	8.769270	4.184700	0.815883	5.476941	8.813725	0	4.707704
2016-10-03	8.311614	4.632987	0.679195	5.056123	8.552767	0	4.635184
2016-10-04	7.850020	3.950758	0.800483	5.045620	8.481428	0	4.531716
2016-10-05	8.176031	3.795881	0.791072	5.016191	8.465436	0	4.524614
...	...	...	...	...	...	...	...
2016-12-26	11.364339	-0.925751	-3.259216	6.572904	11.452315	1	10.159554
2016-12-27	11.316572	-0.081164	-3.746738	6.252706	11.859704	1	11.025948
2016-12-28	10.841367	-0.317127	-3.680397	6.617491	11.644486	1	10.773763
2016-12-29	11.585932	-0.710426	-4.046116	6.519041	11.756765	1	12.158379
2016-12-30	11.991952	-0.399588	-4.436065	6.155139	12.052701	1	13.222450

91 rows × 7 columns

In [7]:
```python
# Compute number of outliers in each period
display(X_train['outlier'].value_counts())
display(X_test['outlier'].value_counts())

# Compute outlier score distribution breakdown by outlier label for each period
display(X_train.groupby(["outlier"])['outlier_score'].quantile([0, 0.025, 0.5, 0.975, 1]))
display(X_test.groupby(["outlier"])['outlier_score'].quantile([0, 0.025, 0.5, 0.975, 1]))
```

```
0    246
1     28
Name: outlier, dtype: int64
0    68
1    23
Name: outlier, dtype: int64
outlier
0        0.000     3.870575
         0.025     4.333878
         0.500     6.065151
         0.975     8.737817
         1.000     9.108401
1        0.000     9.147896
         0.025     9.161529
         0.500    10.232649
         0.975    13.099191
         1.000    15.274125
Name: outlier_score, dtype: float64
outlier
0        0.000     4.073165
         0.025     4.290297
         0.500     6.097016
         0.975     8.720498
         1.000     8.980789
1        0.000     9.462245
         0.025     9.531259
         0.500    11.025948
         0.975    13.507007
         1.000    13.854799
Name: outlier_score, dtype: float64
```

In [8]:
```python
# Create function to compute outliers according to outlier group definintions
# outlier_day_count between 2 and 14 are defined to be outliers

def compute_outliers(data):

    count = 0
    outliers = []

    for ind, values in data.iterrows():

        if values['outlier'] == 1:
            count = count + 1
            start_date = ind - timedelta(days = count - 1)
            outliers.append([start_date, ind, count])
        else:
            count = 0

    outliers = pd.DataFrame(outliers)
    outliers.columns = ['start_date', 'end_date', 'outlier_day_count']
    outliers = outliers.groupby('start_date')[['end_date', 'outlier_day_count']].max()
    outliers = outliers.reset_index()

    return outliers[(outliers.outlier_day_count > 2) & (outliers.outlier_day_count <= 14)]
```

In [9]:
```python
# compute outliers according to our new definition as defined above

display(compute_outliers(X_train))
display(compute_outliers(X_test))
```

	start_date	end_date	outlier_day_count
1	2016-01-08	2016-01-10	3
3	2016-02-13	2016-02-15	3
5	2016-02-20	2016-02-24	5
6	2016-08-08	2016-08-11	4
8	2016-09-06	2016-09-09	4
9	2016-09-11	2016-09-14	4

	start_date	end_date	outlier_day_count
0	2016-11-16	2016-11-20	5

In [ ]: