Clustering Project - Offboarding

Introduction and Problem Statement

I am currently working as the one of the team leads in UBS' offboarding team and presently the team are having difficulty meeting the agreed Service Level Agreements (SLA) for a number of offboarding requests. Offboarding requests are assigned a priority from 0 to 4 based on the following client attributes:

1. Anti-Money Laundering (AML) risk rating
    - This is from Higher to Low
        A. Know Your Customer (KYC) review date
        - In date, expired, and expired over 90 days

As the offboarding team are sent numerous requests at different levels of the UBS data structure, I believe clustering can be used to help identify where we are facing issues. The result of this analysis can help direct where we deploy resources to attend to this population. This allocation of resources will enable the pipeline to be tackled in a more efficient manner and reduce the time offboarding requests take. This time save will improve client satisfaction in the process rather than cost efficiencies .

In this project I will cluster the different types of offboarding inputs and their priorities to determine:

- Which priority of offboard requests are going over their SLAs?
- Which types of offboarding have been taking the longest historically?

If the above proves successful I will try and visualise which priorities are going over the agreed Service Line Agreement (SLA).

**Testable Hypothesis **
Offboarding requests can be clustered by priority to determine which what inputs are breaking their SLA.

Overview

I am going to attempt to cluster the different type of offboarding requests and the time each request takes to get through the full front to back offboarding process. I can the use my subject matter expert (SME) knowledge to provide reasons why this might be the case.
I have completed this project on my own and not as part of a team.

Tasks

In order for this analysis to be performed to a useful level I will have to:

1. **Import and clean the data.**
    - **The data I am using is taken from a quasi-structured database that has been manually maintained over a number of years by a number of different team members. I predict there will be a number of data quality issues that I will need to clean up.**

1. **Check the data types.**
    - **This will involve making sure numbers are classed as integers to make sure the analysis can be performed correctly.**

1. **Data Visualisation.**
    - **Visualising the data in different ways should to give some insight into potential populations that I should cluster. It is a good way to highlight any potential outliers in the population alongside picking up any data quality issues that might have been missed in the original data clean-up.**
    - **Boxplots and scatterplots immediately strike me as the most useful for this analysis.**

1. **Clustering first attempt**
    - **I am going to use KMeans to cluster my data. It is one of the standard clustering algorithms available to use. This initial clustering attempt will be able to provide insight into the population which I can improve on.**

1. **Elbow test**
    - **This will help me identify if I have used the most appropriate amount of clusters to create the best outputs for analysis.**

1. **Potential clustering second attempt**
    - **Based on the elbow to see if I can improve on the first attempt and create better results.**

1. **Conclusion**
    - **Has my clustering been successful, what can this analysis tell me, and how can I use it to improve the process I manage?**

Sourcing the data

This data has been sourced from the CMO Offboarding team's database of requests.

This database has existed since the CMO offboarding team was created in 2016. This tracker is stored on a SharePoint site and has gone through numerous iterations as the process has been improved and enhanced. It contains all the details required for an offboard request. This includes

- **Parent Level (PL)**
- **Consolidated Entity (CE)**
- **RXM (Credit-aggregation identifier)**
- **GL (Also known as a cConsol, which is a client account number)**
- **Xref (client sub-account identifier for different products and regions).**

All requests in this tracker have been submitted by different areas of the business. One of the key risk controls the offboarding team adheres to is that we do not generate our own pipeline. This is important as each line in the tacker is related to a case and is important to someone within the company, as such it needs to be actioned appropriately.

There are two stages to offboarding:

- **Triage**
- **Execution**

Triage
The first stage of offboarding is the analytics and triage stage. This is the risk assessment stage and I am the supervisor for this team. The team checks a large number of systems and areas of the bank to see if there is an active client relationship or another reason as to why offboarding a client might not be safe to offboard, such as an open position. Once all checks have been completed the case is handed over to the Execution team

Execution
Once the Execution team have been handed over the client data it is their responsibility to make sure the client is unable to trade in any 'duel-maintained' systems. These are the manually maintained systems that do feed directly from the master systems and if left open are a unacceptable trading risk. Only once the client data has been made inactive in these systems can final master deactivation take place.

The data I have exported is from a simplified summary view of the offboarding tracker which summarises the completed request by type and priority.

There is no CID (Client Identifying Data) in this data set which means it can be seen external to UBS.

Exploring the data

In [1]:

```
# import libaries required for data manipulation
import numpy as np
import pandas as pd

# import required visualiation packages
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

# import KMeans algorithm to do the clustering
from sklearn.cluster import KMeans

# import scale from sklearn to allows graphs to be displayed in notebook
from sklearn.preprocessing import scale
from sklearn.preprocessing import MinMaxScaler

# inline from matplotlib allows graphs to be shown in the Juypter notebook
%matplotlib inline
```

In [2]:

```
# Importing Offboarding data required for this task
data = pd.read_csv("offboarding_data_new.csv", encoding='ANSI')
```

In [3]:

```
# view the column headers to see whats in the data
data.head()
```

Out[3]:

	ID	Offboarding_Reference	Input Key - 1	Date of Request	Priority_Calc	Triage Completed/Rejected date	U Mast Deactivat d
0	42	IBSIMP_001	CE	20/01/2017	P2	NaN	03/02/2(
1	43	IBSIMP_002	CE	20/01/2017	P2	NaN	03/02/2(
2	44	IBSIMP_003	CE	20/01/2017	P1	NaN	03/02/2(
3	45	IBSIMP_004	CE	20/01/2017	P4	NaN	03/02/2(
4	46	IBSIMP_005	CE	20/01/2017	P2	NaN	03/02/2(

I am dropping the 'ID' and 'Offboarding_Reference' columns as they are not needed. I can use Python's zero index more effectively.

In [4]:

```
data.drop(['ID','Offboarding_Reference'],axis = 1, inplace = True)
```

In [5]:

```
#checking the data to make sure the above has worked properly
data.head(2)
```

Out[5]:

	Input Key - 1	Date of Request	Priority_Calc	Triage Completed/Rejected date	UBS Masters Deactivation date	No. Days to Triage	No. Days to Execute	I Days Offbo
0	CE	20/01/2017	P2	NaN	03/02/2017	-42755	42769	
1	CE	20/01/2017	P2	NaN	03/02/2017	-42755	42769	

I can see that I am working with a limited data set. This has advantages and disadvantages. This small data set will make analysis easier however it is unlikely I can extrapolate the results due to the limited scope. The amount of data is minimal as it only contains case priority, offboarding stage dates and input keys.

Based on the above I will need to amend the column headers to make the data more malleable.

In [6]:

```
# shortening column names to make my code easier to read
data.columns = data.columns.str.strip()
data.columns = ['input' ,'request_date','priority','triage_date','execution_date','tria
ge_days','execution_days','offboard_days']
```

In [7]:

```
# checking to confirm colmn names have been shorterned correctly
data.columns
```

Out[7]:

```
Index(['input', 'request_date', 'priority', 'triage_date', 'execution_dat
e',
       'triage_days', 'execution_days', 'offboard_days'],
      dtype='object')
```

I am going to look into the contents of this data set to see what I will be working with. I will be checking the types of data provided make any adjustments required.

In [8]:

```
#Overview of data in order to assess what the data is made up of
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46675 entries, 0 to 46674
Data columns (total 8 columns):
input             46675 non-null object
request_date      46675 non-null object
priority          46675 non-null object
triage_date       33697 non-null object
execution_date    46675 non-null object
triage_days       46675 non-null int64
execution_days    46675 non-null int64
offboard_days     46675 non-null int64
dtypes: int64(3), object(5)
memory usage: 2.8+ MB
```

The above confirms the required columns are correctly defined as integers.

In [9]:

```
# describe the data to see statistical breakdown
data.describe()
```

Out[9]:

	triage_days	execution_days	offboard_days
count	46675.000000	46675.000000	46675.000000
mean	-11838.678050	12016.180482	177.502432
std	19265.445335	19251.112791	167.800242
min	-43607.000000	-286.000000	0.000000
25%	-42705.000000	43.000000	83.000000
50%	32.000000	70.000000	102.000000
75%	105.000000	42849.000000	229.000000
max	949.000000	43676.000000	998.000000

The above shows some very interesting information.

- **The longest an offboarding request has taken is 998 days!**
  - **This is just under three years.**
- **The mean figure is 186 days.**
  - **This is just over six months.**
  - **Our longest SLA is 90 days. On average requests are taking twice as long as they should be.**
  - **I can confirm this is due to the huge backlog of requests currently in the pipeline. I can argue it is taking a long time to start offboarding requests rather than the time spent working on them.**

In [10]:

```
#checking the number of different priorites
data.priority.value_counts()
```

Out[10]:

```
P2    25419
P4    10219
P1     9459
P3      969
P0      609
Name: priority, dtype: int64
```

I can see 'priority' is an object and all inputs begin with P.

I'm going to remove this first character and turn it into a integer so it can be visualised easier.

In [11]:

```
#Remove the 'P' from the priority coloumn so they be made into intergers later
data['priority'] = data['priority'].replace('P' , '' , regex = True)
```

In [12]:

```
#Checking the above has functioned correctly
data['priority'].head()
```

Out[12]:

```
0    2
1    2
2    1
3    4
4    2
Name: priority, dtype: object
```

The above has worked but it is still an 'object' so I need to convert this to a numerical format.

In [13]:

```
#convert the 'priority' column into float data types
data['priority'] = data['priority'].astype(int)
data['priority'].dtype
```

Out[13]:

```
dtype('int32')
```

In [14]:

```
#group by priority to see the breakdowm
data.groupby('priority').count()
```

Out[14]:

priority	input	request_date	triage_date	execution_date	triage_days	execution_days	offbo
0	609	609	407	609	609	609	
1	9459	9459	6226	9459	9459	9459	
2	25419	25419	20852	25419	25419	25419	
3	969	969	416	969	969	969	
4	10219	10219	5796	10219	10219	10219	

As I are using a limited dataset the above makes sense due to the weighting of the priority matrix in favour of 2s and 4s.

In [15]:

```
# check for any NaNs in the data
data.isnull().any()
```

Out[15]:

```
input             False
request_date      False
priority          False
triage_date        True
execution_date    False
triage_days       False
execution_days    False
offboard_days     False
dtype: bool
```

I'm going to drop any column with the Triage date as 'null' as this has a knock on effect with the rest of the data. It will make the amount of Triage days column inaccurate. I will inform my team that this missing data needs to be completed so next time this analysis is run the enriched data can provide more accurate results.

In [16]:

```
data.dropna(subset = ['triage_date'], inplace = True)
```

In [17]:

```
# check for any NaNs in the data
data.isnull().any()
```

Out[17]:

```
input             False
request_date      False
priority          False
triage_date       False
execution_date    False
triage_days       False
execution_days    False
offboard_days     False
dtype: bool
```

In [18]:

```
#checking the number of different priorites
data.input.value_counts()
```

Out[18]:

```
Xref                              15355
CE                                10860
RXM                                2728
GL                                 2529
XRef                               1231
CMIS Parent                         966
Trading Name / CRM / Shortname       24
AuthProd                              4
Name: input, dtype: int64
```

I need to make sure the column inputs are uniform. As python is case-sensitive, I need to amend 'XRef' to 'Xref' and shorten the rest.

In [19]:

```
data["input"] = data["input"].replace({'XRef':'Xref'}, regex=True)
data["input"] = data["input"].replace({'CMIS Parent':'CMIS'}, regex=True)
data["input"] = data["input"].replace({'Trading Name / CRM / Shortname':'TNs'}, regex=T
rue)
```

In [20]:

```
#checking the data again
data.input.value_counts()
```

Out[20]:

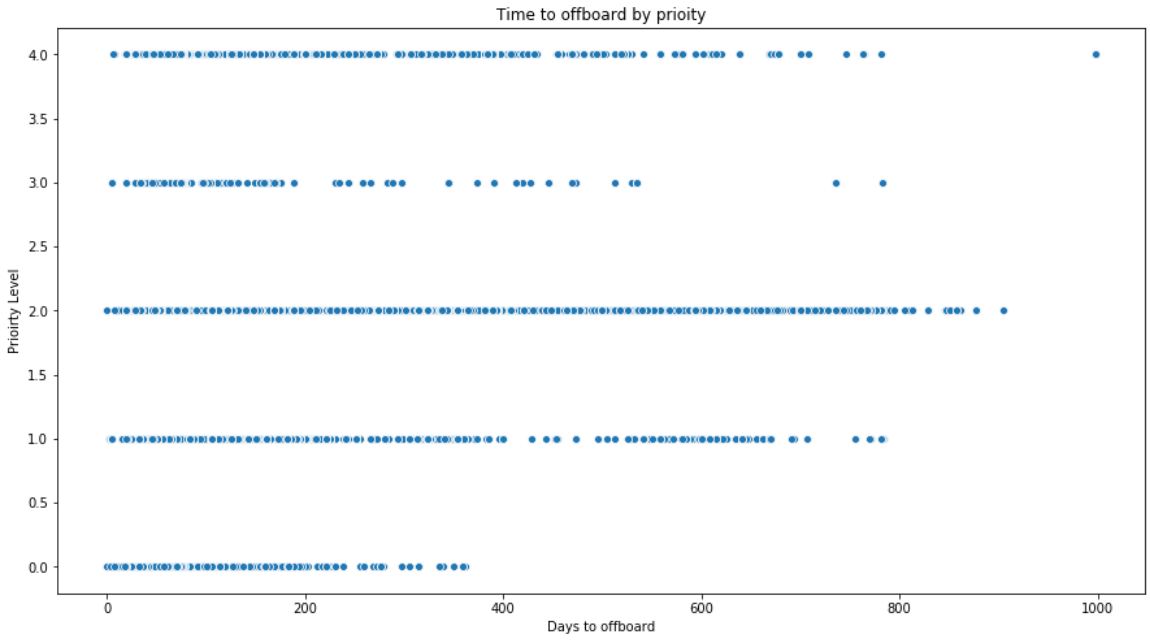
```
Xref        16586
CE          10860
RXM          2728
GL           2529
CMIS          966
TNs            24
AuthProd        4
Name: input, dtype: int64
```

This shows me the most popular requests are at the Xref level. This is the lowest level of the UBS data structure so I expect to see these take the shortest amount of time to offboard.

Visualisation of data

In [21]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'offboard_days',y = 'priority', data = data)
plt.title("Time to offboard by prioity")
plt.xlabel("Days to offboard")
plt.ylabel("Prioirty Level")
plt.savefig('PrioirtyandDays.pdf')
```



The above tells me that that offboarding have number of cases that have breached their SLAs. When I rerun this code with updated data from the latest quarter I can compare the plots so show progress. In order to do this I need to save a version of this as PDF to include in a presentation. The code above does this.

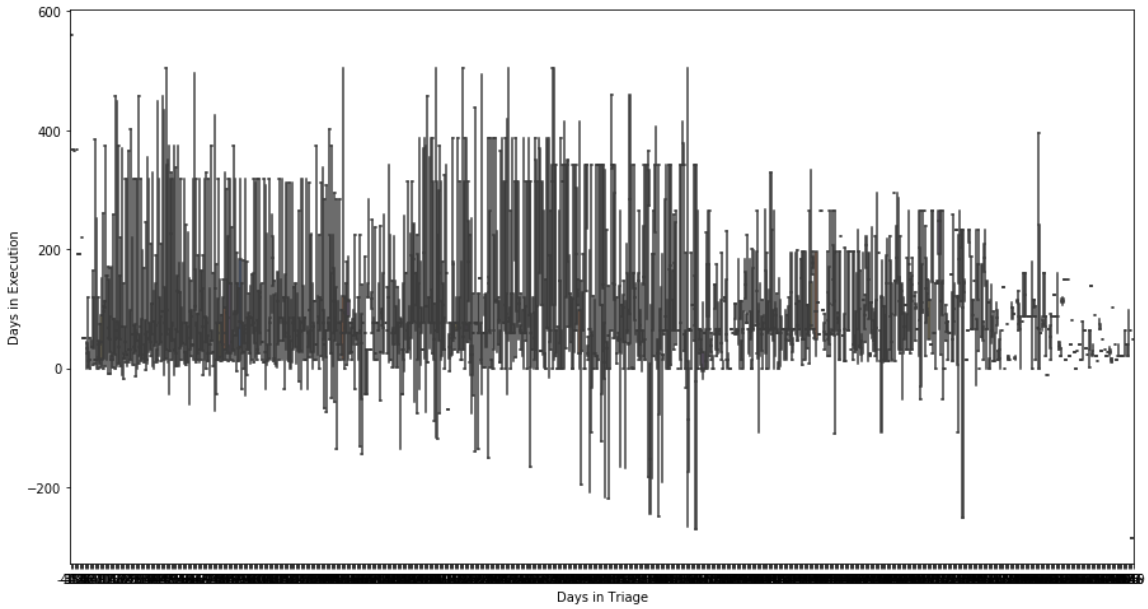
Now I'm going onto the K-Means clustering in order to provide the data I can use to tackle the issue. I will continue to visualise the data before doing this.

In [22]:

```
plt.figure(figsize=(15,8))
sns.boxplot(x="triage_days", y="execution_days", data=data,whis="range", palette="mute
d")
plt.xlabel("Days in Triage")
plt.ylabel("Days in Execution")
plt.legend
```

Out[22]:

```
<function matplotlib.pyplot.legend(*args, **kwargs)>
```



While this boxplot looks messy it is telling me something very important. There are negative figures in the 'Days in Triage and 'Days in Execution' columns.

I can work off the assumption that the date format has been inputted incorrectly. The team have been using a mix of the American date format (MM-DD-YYYY) and the British format (DD-MM-YYYY). This can cause issues with dates such as 03/02/2019 as it can either be March 2nd or 3rd February. This means I have more data cleaning work to do.

I am going to suggest to the global offboarding lead that the team changes the standard date format to the universal "YYYYMMDD" format in future. This will avoid confusion and provide better data quality going forward.

In [23]:

```
# drop all rows with a triage time of less than 0 days
data.drop(data[data.triage_days < 0].index, inplace=True)
```

In [24]:

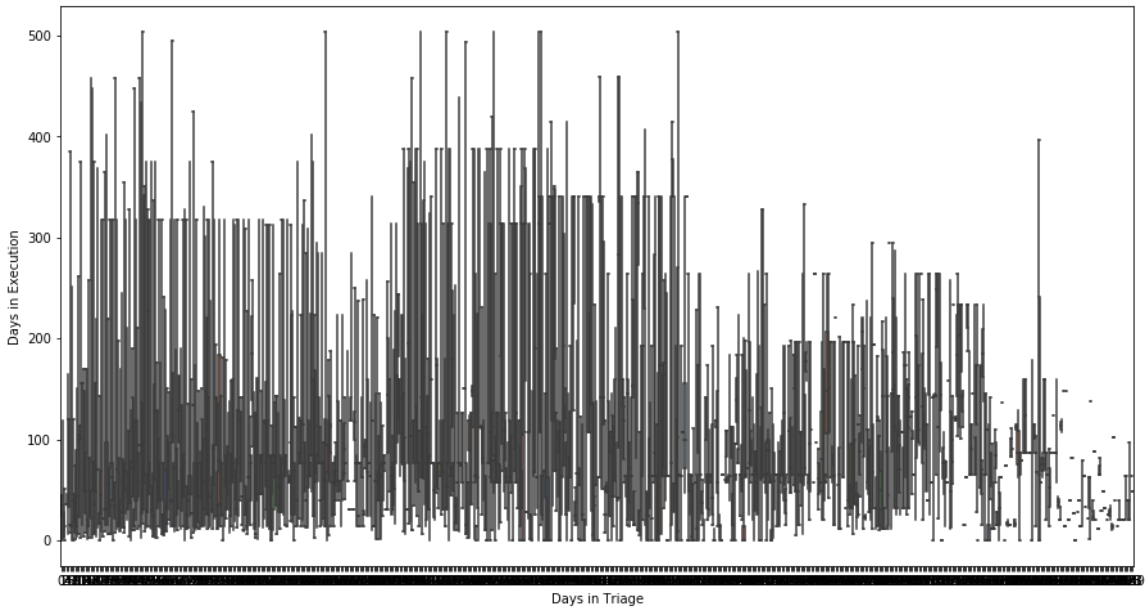
```
# drop all rows with a execution time of less than 0 days
data.drop(data[data.execution_days < 0].index, inplace=True)
```

In [25]:

```
plt.figure(figsize=(15,8))
sns.boxplot(x="triage_days", y="execution_days", data=data,whis="range", palette="mute
d")
plt.xlabel("Days in Triage")
plt.ylabel("Days in Execution")
plt.legend
```

Out[25]:

```
<function matplotlib.pyplot.legend(*args, **kwargs)>
```



This box plot shows that the data clean-up has been successful, there are no negative numbers left and how vast the offboarding request pipeline is. I need to check data again to see how much data I have left to cluster.

In [26]:

```
data.groupby('priority').count()
```

Out[26]:

priority	input	request_date	triage_date	execution_date	triage_days	execution_days	offbo
0	393	393	393	393	393	393	
1	6184	6184	6184	6184	6184	6184	
2	20757	20757	20757	20757	20757	20757	
3	411	411	411	411	411	411	
4	5695	5695	5695	5695	5695	5695	

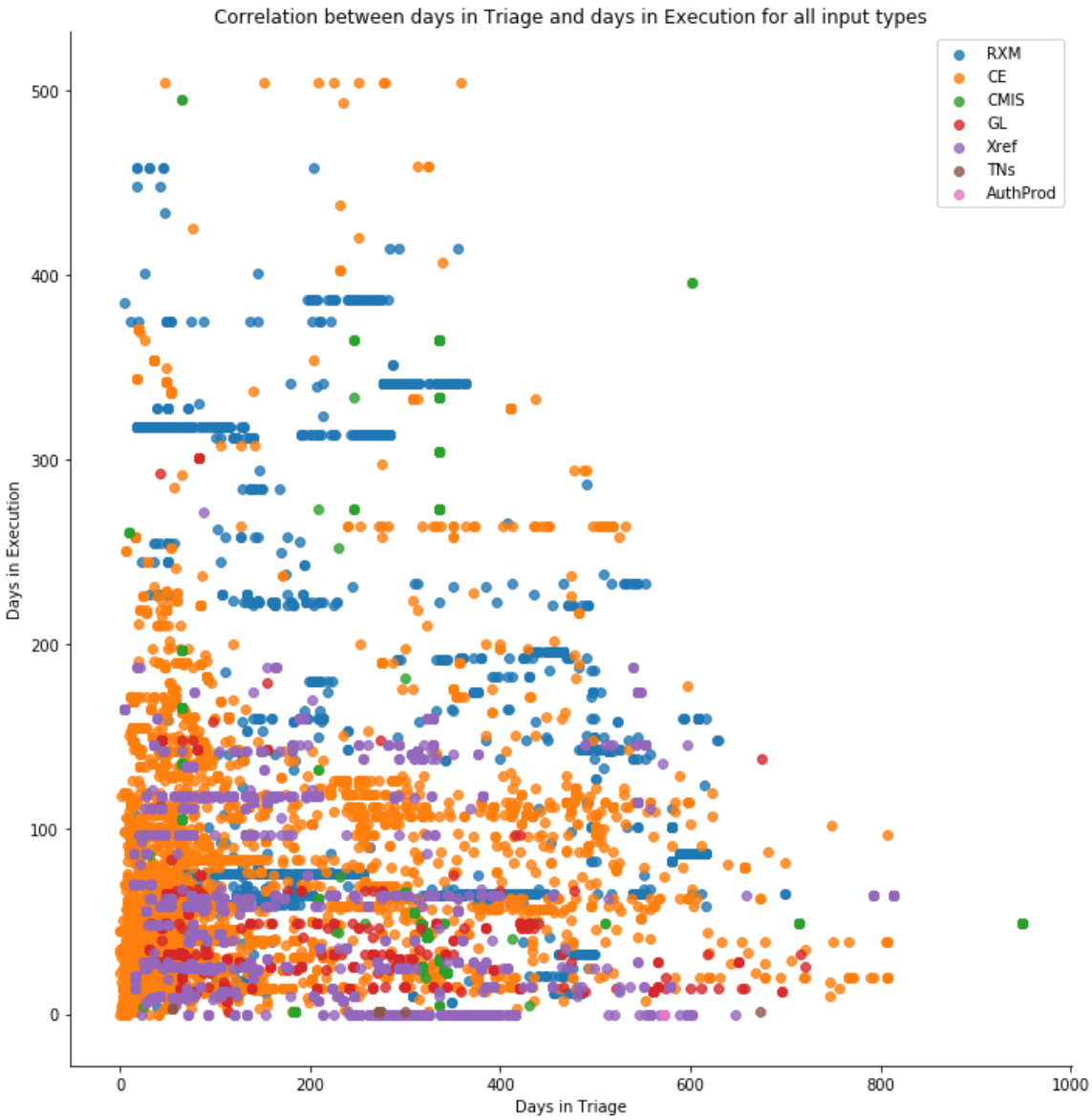
I have dropped a large number of incomplete/ inaccurate data points.

In [27]:

```
sns.lmplot(x="triage_days", y="execution_days", data = data, fit_reg=False, hue='input'
, legend=False, height=10)
plt.title("Correlation between days in Triage and days in Execution for all input type
s")
plt.xlabel("Days in Triage")
plt.ylabel("Days in Execution")
plt.legend(loc='best')
```

Out[27]:

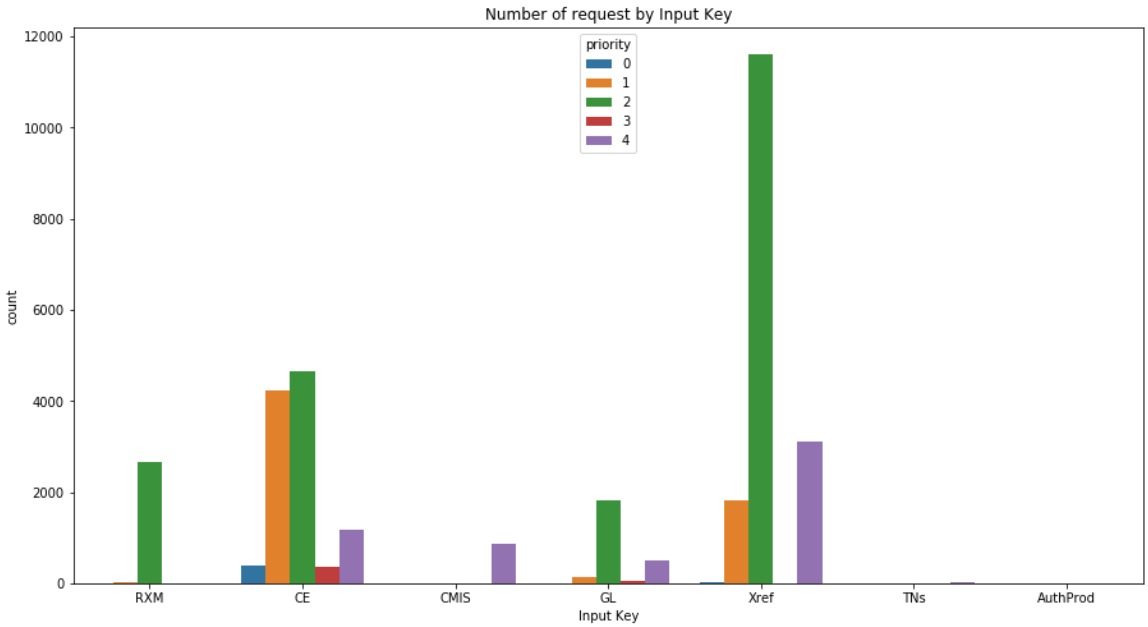
```
<matplotlib.legend.Legend at 0x1a226208>
```



This visualisation shows that the vast amount of requests that have not met the agreed SLAs. This visualisation will be used to report the extent of the problem to senior management. There might be an argument that the amount of requests offboarding have in the pipeline means the SLAs are not possible to meet. There are some interesting outliers (e.g. the CMIS parent that has spent 900 days in Triage) that I will report back to my team in our next throughput meeting.

In [28]:

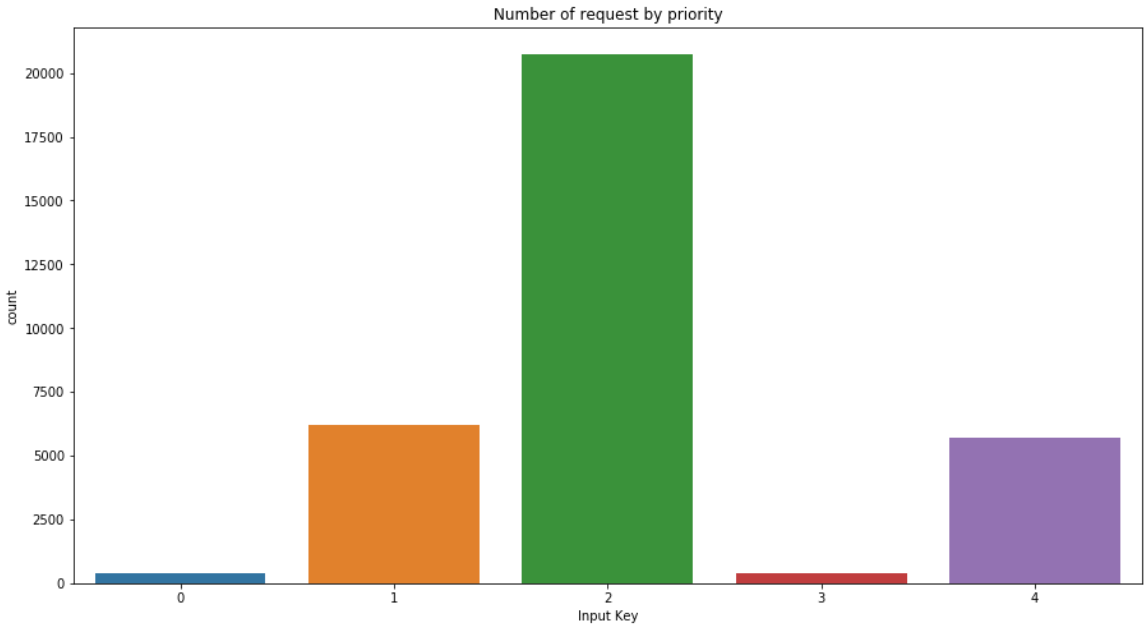
```
plt.figure(figsize=(15,8))
sns.countplot(x = 'input', hue = 'priority', data = data)
plt.title("Number of request by Input Key")
plt.xlabel('Input Key')
plt.show()
```



The above show that there are a lot more CE requests given the P1 priority compared to than all other inputs. This due to the priority for cases that have a regulatory requirement (e.g. MIFID II, Swiss Stay etc...).

In [29]:

```
plt.figure(figsize=(15,8))
sns.countplot(x = 'priority', data = data)
plt.title("Number of request by priority")
plt.xlabel('Input Key')
plt.show()
```



Our prioritisation matrix is weighted to P2 requests due to the RXM input having a larger backlog that the input types.

I am going to create a new variable with priority zero requests only. As these are highest priority type I can use this opportunity to gather some quality management information (MI) on this population for senior management.

In [30]:

```
zero = data.drop(data[data.priority != 0].index)
```

In [31]:

```
zero.head()
```

Out[31]:

	input	request_date	priority	triage_date	execution_date	triage_days	execution_days
1232	CE	24/02/2017	0	25/09/2017	18/12/2017	213	84
1589	CE	13/02/2017	0	04/04/2017	22/05/2017	50	48
1622	CE	13/02/2017	0	06/03/2017	12/05/2017	21	67
1667	CE	13/02/2017	0	28/03/2017	12/05/2017	43	45
5398	CE	31/03/2017	0	25/09/2017	09/02/2018	178	137

The variable creation has been successful. The P0 SLA is 30 days so seeing a case take 213 to complete triage is alarming.

In [32]:

```
zero.describe()
```

Out[32]:

	priority	triage_days	execution_days	offboard_days
count	393.0	393.000000	393.000000	393.000000
mean	0.0	40.246819	38.396947	78.643766
std	0.0	50.835897	41.958923	73.878384
min	0.0	0.000000	0.000000	0.000000
25%	0.0	9.000000	13.000000	24.000000
50%	0.0	21.000000	21.000000	55.000000
75%	0.0	53.000000	50.000000	106.000000
max	0.0	315.000000	293.000000	362.000000

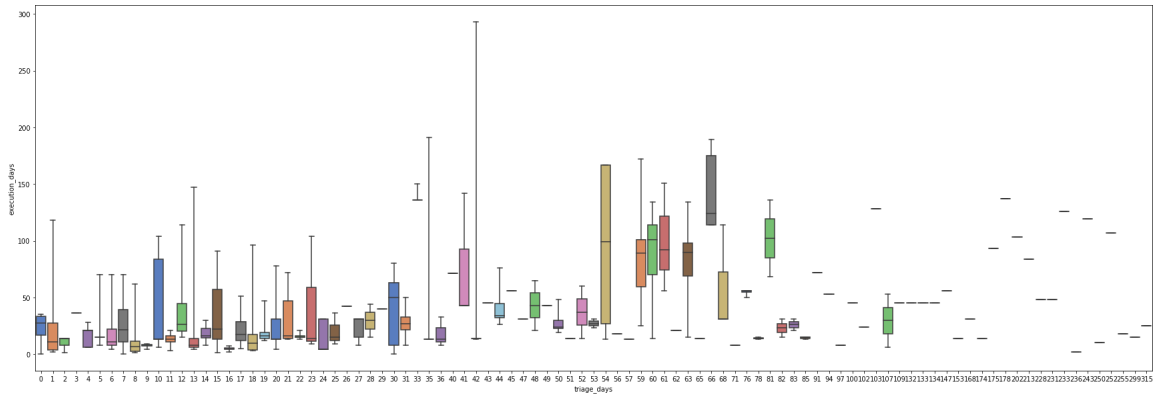
The mean for a P0 offboard if current 78 days. Reporting this to senior management is misleading as it contains request from 2017 when then process was in its infancy. A future approach would be splitting the data by year request received to see the improvement in this metric.

In [33]:

```
plt.figure(figsize=(30,10))
sns.boxplot(x = 'triage_days', y = 'execution_days', data = zero ,whis="range", palette
="muted")
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b2ec978>
```



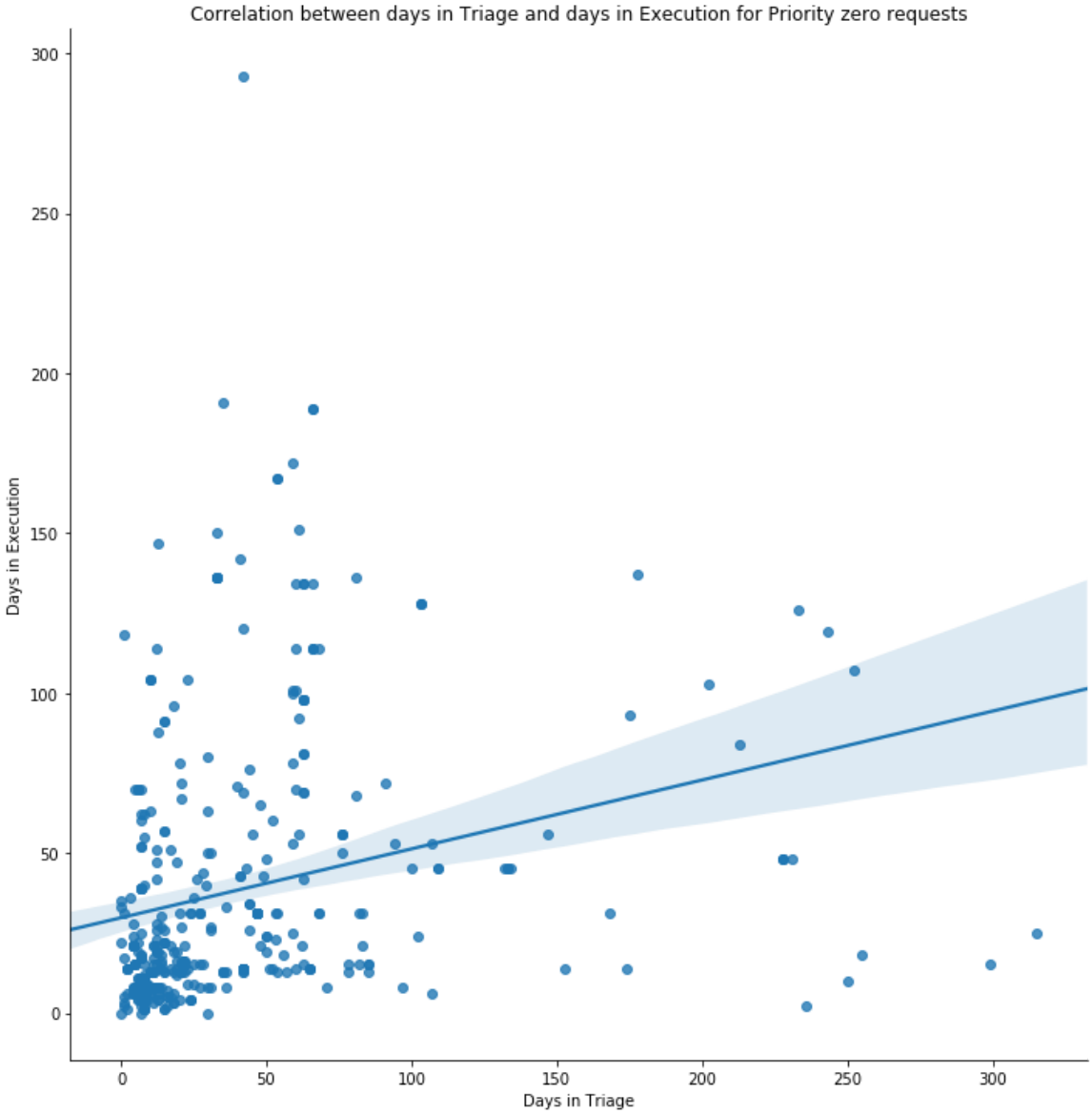
The range and mean for offboarding cases based on timings vary greatly based on these boxplots. There does not appear to be a pattern in the data however this is likely due to the outlying data points being included. I think looking in the correlation between the data points is a good next step.

In [34]:

```
sns.lmplot(x = 'triage_days', y = 'execution_days', data = zero, height = 10)
plt.title("Correlation between days in Triage and days in Execution for Priority zero r
equests")
plt.xlabel("Days in Triage")
plt.ylabel("Days in Execution")
```

Out[34]:

```
Text(3.674999999999997, 0.5, 'Days in Execution')
```

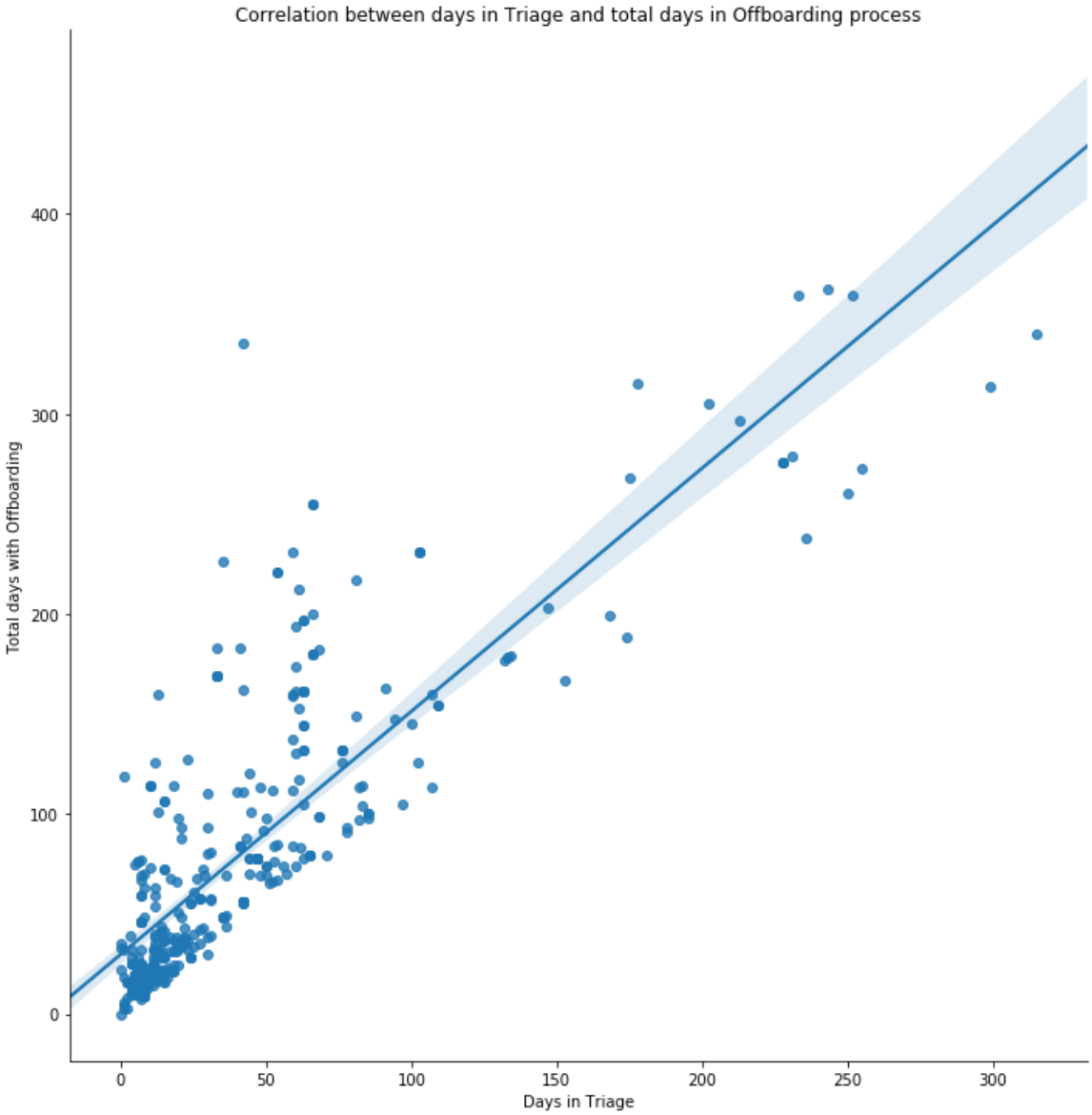


In [35]:

```
sns.lmplot(x = 'triage_days', y = 'offboard_days', data = zero, height = 10)
plt.title("Correlation between days in Triage and total days in Offboarding process")
plt.xlabel("Days in Triage")
plt.ylabel("Total days with Offboarding")
```

Out[35]:

```
Text(3.799999999999997, 0.5, 'Total days with Offboarding')
```

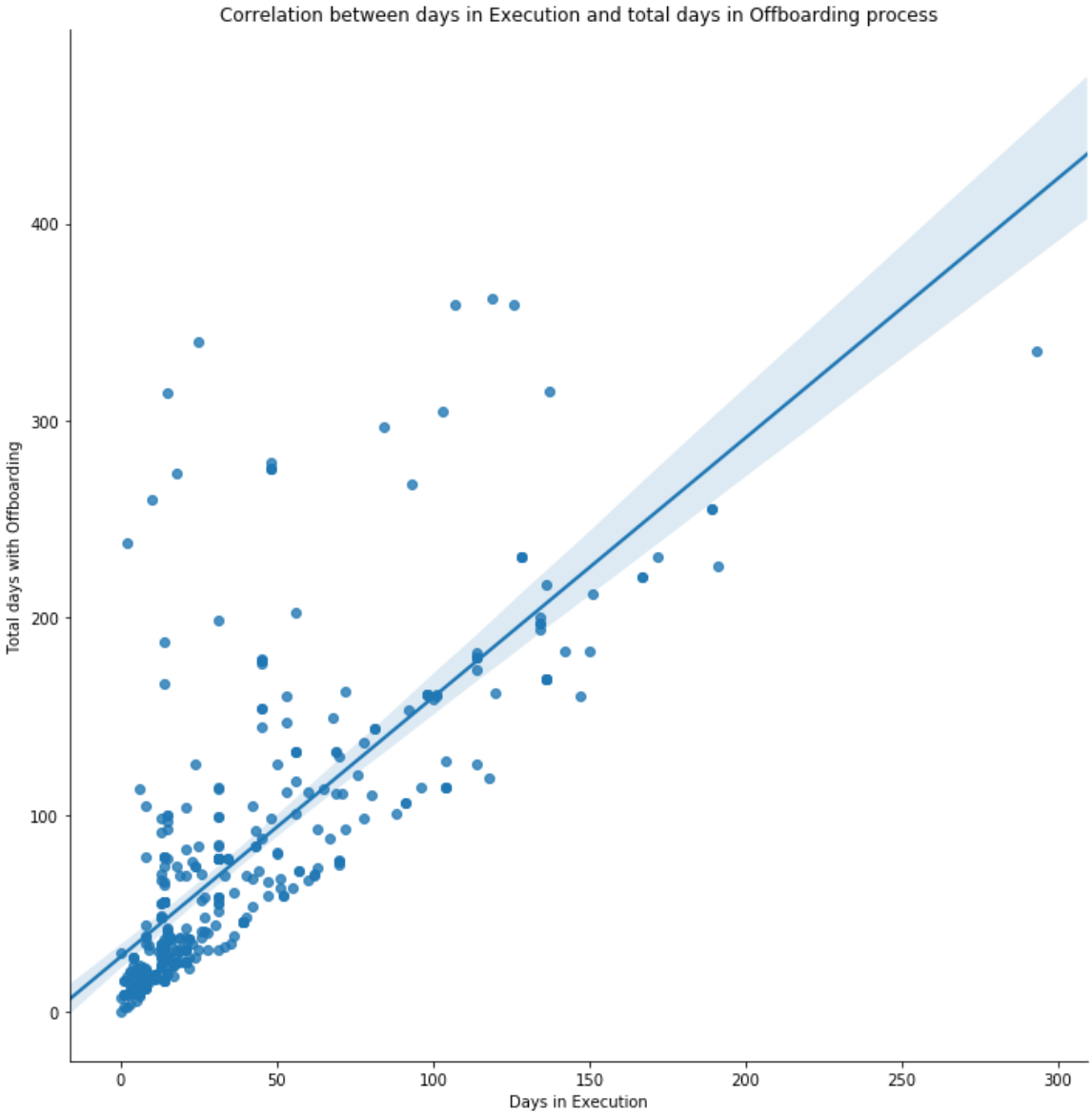


In [36]:

```
sns.lmplot(x = 'execution_days', y = 'offboard_days', data = zero, height = 10)
plt.title("Correlation between days in Execution and total days in Offboarding process"
)
plt.xlabel("Days in Execution")
plt.ylabel("Total days with Offboarding")
```

Out[36]:

```
Text(3.799999999999997, 0.5, 'Total days with Offboarding')
```



As I expected there is a positive correlation between the 'days in triage' and 'days in execution'. This is due to the more complicated client setups taking longer to triage and thus taking longer execute. I will look into the outliers as these could be due to human error in the process. If this is the case it can used to justify the need for more automation within offboarding.

KMeans Model

One of the standard clustering models I can use is the KMeans. I'm going to employ this and perform analysis on the outcome before coming onto different models I could use.

In [37]:

```
#drop the ward coloumn
dataKM = data.drop(['input','request_date','triage_date','execution_date','triage_days'
,'execution_days'], axis=1)
```

I have dropped the input column because all input types have the same SLA agreements therefore it is not required.

In [38]:

```
dataKM_scaled = scale(dataKM)
```

In [39]:

```
dataKM_scaled
```

Out[39]:

```
array([[-0.15189653,  3.8911574 ],
       [-0.15189653,  3.41020207],
       [-0.15189653,  3.41020207],
       ...,
       [-0.15189653, -1.18193306],
       [ 1.95094754, -1.18852149],
       [-2.25474059, -1.18193306]])
```

In [40]:

```
#running K-Means with 3 clusters
model = KMeans(n_clusters=3)
model.fit(scale(dataKM))
```

Out[40]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [41]:

```
#adding cluster label to data
dataKM['cluster'] = model.labels_.astype(int)
dataKM.head()
```

Out[41]:

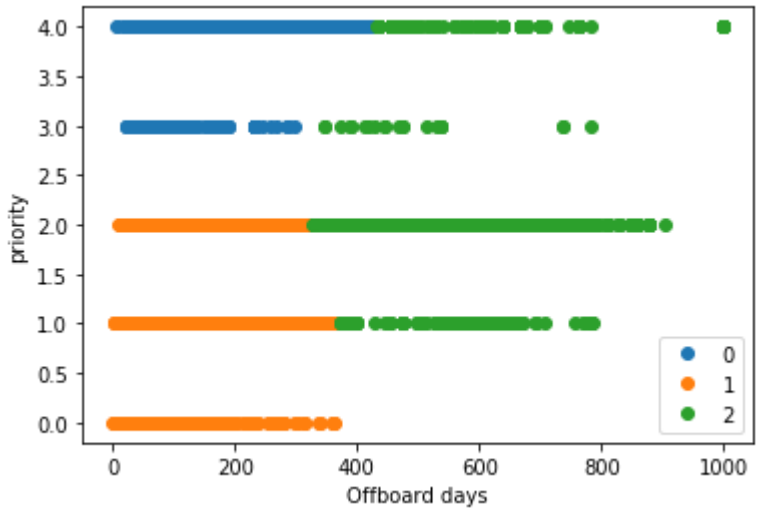
	priority	offboard_days	cluster
543	2	777	2
544	2	704	2
545	2	704	2
546	2	704	2
547	2	704	2

In [42]:

```
groups = dataKM.groupby('cluster')
```

In [43]:

```
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.offboard_days, group.priority, marker='o', linestyle='', label = name
)

plt.xlabel('Offboard days')
plt.ylabel('priority')
ax.legend()
plt.show()
```

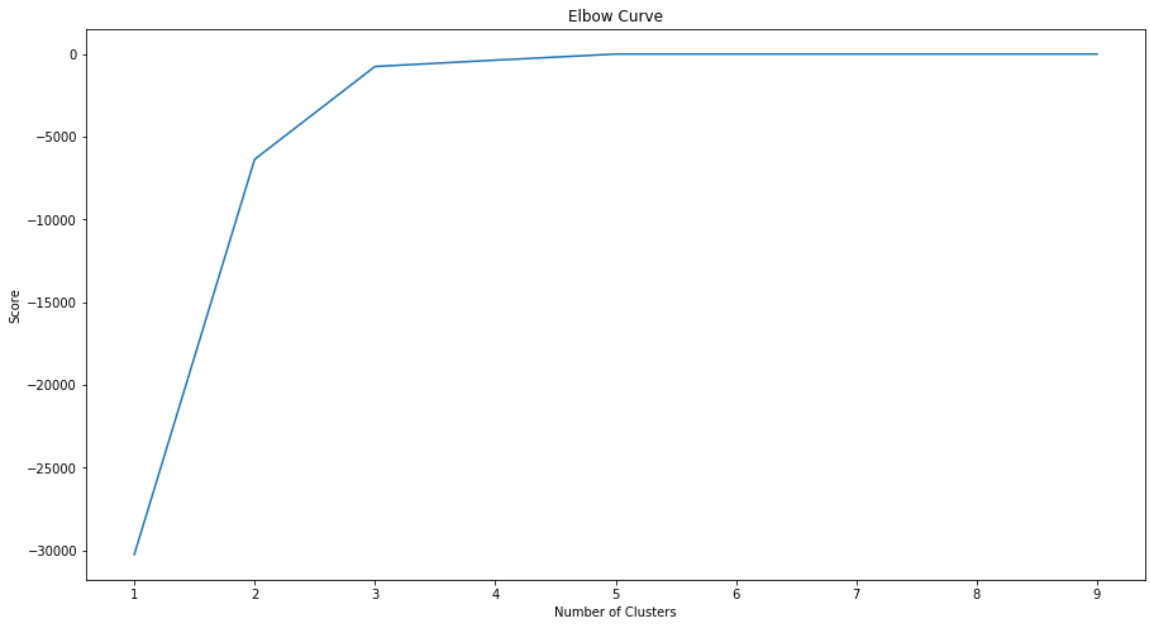


Three clusters appears to group large parts of the data together. I can see that there is potential for smaller clusters o be created. An elbow test should be performed next to see if I need to increase the amount of clusters in my model.

In [44]:

```
# Elbow Test
X = dataKM[['offboard_days']]
Y = dataKM[['priority']]

Nc = range(1, 10)
kmeans = [KMeans(n_clusters=i) for i in Nc]
score = [kmeans[i].fit(Y).score(Y) for i in range(len(kmeans))]
plt.figure(figsize=(15,8))
plt.plot(Nc,score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```

```
\\ubsprod.msad.ubs.net\UserData\BEARCROD\Home\Miniconda3_64bit\envs\pyt367
\lib\site-packages\ipykernel_launcher.py:7: ConvergenceWarning:

Number of distinct clusters (5) found smaller than n_clusters (6). Possibl
y due to duplicate points in X.

\\ubsprod.msad.ubs.net\UserData\BEARCROD\Home\Miniconda3_64bit\envs\pyt367
\lib\site-packages\ipykernel_launcher.py:7: ConvergenceWarning:

Number of distinct clusters (5) found smaller than n_clusters (7). Possibl
y due to duplicate points in X.

\\ubsprod.msad.ubs.net\UserData\BEARCROD\Home\Miniconda3_64bit\envs\pyt367
\lib\site-packages\ipykernel_launcher.py:7: ConvergenceWarning:

Number of distinct clusters (5) found smaller than n_clusters (8). Possibl
y due to duplicate points in X.

\\ubsprod.msad.ubs.net\UserData\BEARCROD\Home\Miniconda3_64bit\envs\pyt367
\lib\site-packages\ipykernel_launcher.py:7: ConvergenceWarning:

Number of distinct clusters (5) found smaller than n_clusters (9). Possibl
y due to duplicate points in X.
```



This elbow chart shows me that in the data set the optimum amount of clusters is 3. This is was I originally ran so I am to rerun the K-Means code above with 4 clusters and compare the two sets of data just to see the difference.

In [45]:

```
#running K-Means with 4 clusters
model = KMeans(n_clusters = 4)
model.fit(scale(dataKM))
```

Out[45]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [46]:

```
#adding cluster label to data
dataKM['cluster'] = model.labels_.astype(int)
dataKM.head()
```

Out[46]:

	priority	offboard_days	cluster
543	2	777	2
544	2	704	2
545	2	704	2
546	2	704	2
547	2	704	2

In [47]:

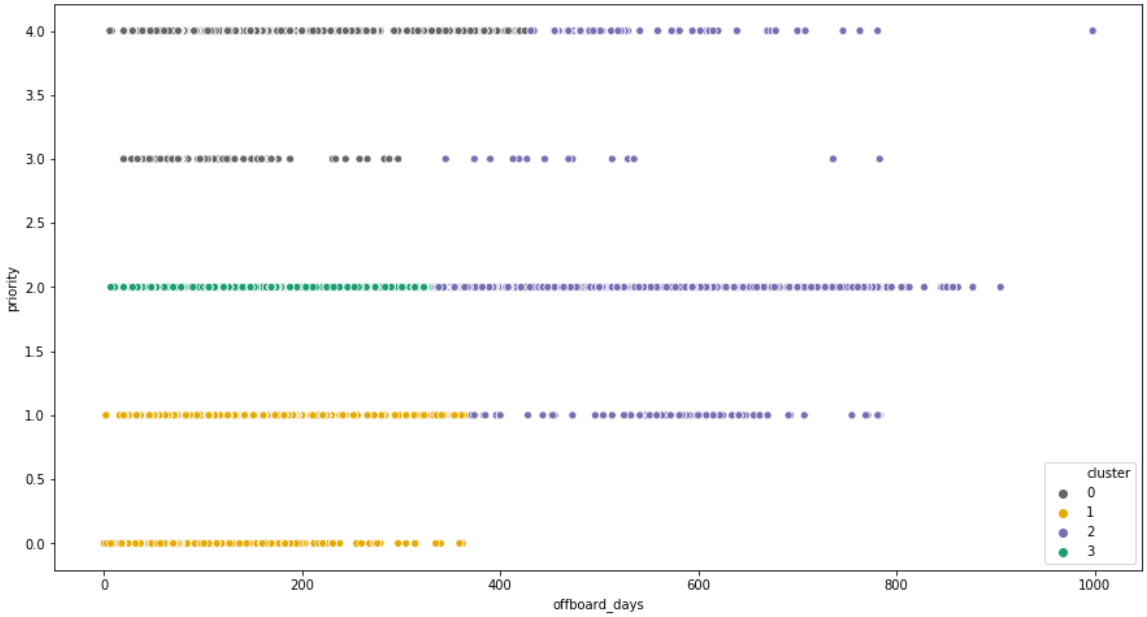
```
groups = dataKM.groupby('cluster')
```

In [48]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(x = dataKM['offboard_days'], y = data['priority'],
                hue = dataKM['cluster'],
               palette= 'Dark2_r')
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efa048>
```



I changed the plot type to seaborn as it is clearer to see the data points as separate plots rather than a continuous line.

I can see that these four clusters make logical sense:

- **P0 and P1s**
- **Vastly aged P1, P2, P3, and P4**
- **Lesser aged P2 (the split as P2 is the largest population)**
- **Lesser aged P3 and P4s**

Conclusion

I can interpret a number of things from the above:

- **All priority types are breaching their SLAs regularly.**
- **P2 and P3 are worked in similar time frames and within the SLA.**
- **P0 and P1 are similarly worked in similar time frames however they are breeching the SLA agreement in a lot of cases.**
- **KMeans has clustered together cases from all priorities that are over 300 days.**
- **The P2s have their own cluster. This is most likely due to the large amount of P2 requests offboarding receive. This shows that the prioritisation matrix is weighted towards this level of requests so should be readdressed.**

There are limits to this data and data model. The limited scope really has an impact on the usefulness of the observations I can make. Once the data has been cleaned up a source I could get a lot more useful and accurate data. I know the mean and medium offboard times stated above is inaccurate and the one action I am going to take away from this project is data validation rules within the tracker to prevent this from happening in the future.

From this I can take clearly see offboarding need to pay more attention to the timeliness of P0 and P1 requests. These are the most high-risk offboards. These clients require their KYC review to done on a yearly bases. If offboarding make sure to complete these requests quickly it will reduce the burden on the KYC review team as well as reducing the overall risk appetite of the bank. These are all in cluster 1 and this cluster will allow me to come up with a plan how offboarding can clear the backlog of P0 and P1s in six weeks. If the team can meet the SLA for new P1 requests the aged population can be completed as a clean-up population. If the team manage to get the P0 and P1 populations reduced it can be easily managed by on the analysis as part of a weekly control report.

The next population to clear will be cluster 2. The aged backlog for the lower priorities should be cleared if our throughput remains higher than the amount of requests offboarding are getting in. Once done the team can tackle cluster 3 before finally cluster 0. As we clear the aged population and clear the data up for open cases this analysis can be rerun to create updated clusters of requests to work on.

I will present this plan to senior management and the team as a logical way to reduce our aged pipeline.

Additional notes

There are other clustering algorithms that might be useful for future iterations of this analysis. KMedians or DBSCAN are two examples.

- **K Medians uses the medians in its calculations as opposed to the mean, which KMeans uses.**
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise) can be used when there are more data points. This runs by picking an arbitrary point in the data and applying rules to cluster them based on 'location' near the point. As soon as there is no more data to add to the cluster based on these rules it picks another arbitrary point to work from until it reaches the amount of points you have specified.**