Regression Project

Problem Statement

Does the time it take to complete the triage on an offboarding request have an impact on the time it takes to execute the same offboarding request?

Overview

I expect that the input key and the time it takes to triage an offboarding request will have an impact on the length of time it takes to execute. One of the request types the offboarding team receives are at a client account level. These tend to be very straightforward (i.e. close the account therefore everything under it must be made inactive). The team also gets much more complicated requests and a good example of this involves removing a business division from a client entity/ consolidated entity (CE). This involves looking at the accounts relating to the CE and identifying what products the account is entitled to, and what business division the product belongs to. After this initial process the team needs to make sure to only mark the relevant components of the accounts and the CE to be offboarded/ made inactive. The offboarding team need to make sure that that the wrong data has not been marked up as this could negatively impact other business divisions in UBS. Since UBS has a shared data model, this step can be very complicated. The more shared attributes offboarded incorrectly the more likely there could be a negative impact on different business divisions. This is relevant is easier the request level the quicker it should be to pass through the initial triage offboarding stage.

The offboarding process has two different teams:

- Triage
- Execution

The triage team performs all the risk assessments and checks to make sure clients can be offboarded safety. Once all checks have been passed to a satisfactory level the triage process can be regarded as complete so the client data is passed over the execution team. The execution team are then responsible for making sure the client is turned off in all trading systems before finally making the sure the accounts are made inactive in MasterFiles and finally the client legal entity is made inactive in Entity Master, (legal entity master system).

This analysis will be relevant to the offboarding team as I can propose more realistic Service Line Agreements (SLA) based on this analysis. This will help performance and throughput to be measured more accurately. Currently all input keys have the same SLA, and this SLA is determined by priority only. Stating that all input key types take the same amount of time no matter the complexity of the request highlights the immaturity of the process.

I will be completing this analysis soley by myself

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 in order to perform a regression analysis properly.

1. Checking and cleaning the data
    - This will involve making sure all data tapes are correctly identified in order to make sure analysis can be performed accurately. For example making sure appropriate numbers are classed as integers or floats.

1. Data Visualisation.
    - Visualising the data in different ways should provide insight into potential relationships between the data I can investigate. There maybe potential relationships related to my problem statement that can be identified. Visualisation is can also be used 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 can be used for this. Some simple regplots in seaborn can provide insight what data my regression model can potentially be built with

1. Build the regression Model
    - In order to build the regression model I will need to carry out the following steps:
      a. split the data
      b. train the model
      c. test the model
      d. analyse the model
      e. evaluate the model

1. Conclusion
    - Has my problem statement been address and how can I use this analysis to improve the offboarding process?

Sourcing the data

The data for this project has come from the offboarding pipeline databases.

There will be Client Identifying Data (CID) in this population so I will be masking or removing this data when submitting for inclusion in my portfolio.

There are a couple of potential issues around sourcing the data due to the source being the CMO Offboarding SharePoint (SP) site. The SP site is an database which has been since the creation of the CMO offboarding team in 2016. There are numerous data quality issues in this tracker which will need to be addressed as part of the data clean-up. An example of this is the old input keys that are no longer processed any more (e.g. Remove Business Account). There have been a number of iterations made to the database and various clean-up exercises undertaken but it will never be a perfectly clean dataset. There is no way to set rows as read-only once the request has been completed therefore it is possible to change completion dates. This and other input errors will impact the results of the analysis. There is no recordable audit trail for updates to the site so I unable identify when the amendment will have been made. There are also a number of redundant columns that are no longer used however must be kept for audit purposes. There are also multiple offboarding requests for the same clients as a previous request might have been rejected however six months later the business relationship might have ended. There are also requests on different levels for the same client. For example if an entire client cannot be offboarded, it does not mean the FRC (Fixed Income, Rates and Credit) products cannot be removed to only leave the active Equities relationship active.

Exploring the data

In [1]:

```
# import numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# import seaborn and matplotlib for visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# import sklean to split, train and test the data model
from sklearn import datasets, linear_model, preprocessing
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# import statsmodels to use as part of testing the model
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.graphics.api import abline_plot

# importing warning to repress the warnings regarding future versions of this software
import warnings
warnings.simplefilter(action = 'ignore', category = FutureWarning)
```

In [2]:

```
# importing the data set
data = pd.read_csv('OffboardingData.csv', encoding = 'ANSI')
```

In [3]:

```
# show data headings
data.head(5)
```

Out[3]:

	ID	Offboard Type	Input Key - 1	CE Number	CE Name	GL	Attribute_Id	Attribute Value	Date of Request	Request Age	I
0	594	Remove Credit Risk	RXM	NaN	NaN	NaN	NaN	CG0753	13/02/2017	859	
1	595	Remove Credit Risk	RXM	NaN	NaN	NaN	NaN	C31534	13/02/2017	859	
2	596	Remove Credit Risk	RXM	NaN	NaN	NaN	NaN	CD9455	13/02/2017	859	
3	597	Remove Credit Risk	RXM	NaN	NaN	NaN	NaN	CI2041	13/02/2017	859	
4	598	Remove Credit Risk	RXM	NaN	NaN	NaN	NaN	CD9454	13/02/2017	859	

The data headings show me that there are a number of columns with null data (NaN).
I can drop the 'ID' column as Python has zero indexed the data frame already and this 'ID' is only used for recording purposes.

In [4]:

```
# check the datatypes for all columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30005 entries, 0 to 30004
Data columns (total 17 columns):
ID                                  30005 non-null int64
Offboard Type                       30005 non-null object
Input Key - 1                       30005 non-null object
CE Number                           13170 non-null float64
CE Name                             13226 non-null object
GL                                  18026 non-null float64
Attribute_Id                        15674 non-null float64
Attribute Value                     21853 non-null object
Date of Request                     30005 non-null object
Request Age                         30005 non-null int64
KYC Review Date                     11295 non-null object
AML Risk Classification             11268 non-null object
Priority_Calc                       30005 non-null object
CMIS Parent ID                      1147 non-null object
CMIS Parent Name                    1146 non-null object
Triage Completed/Rejected date      30005 non-null object
UBS Masters Deactivation date       30005 non-null object
dtypes: float64(3), int64(2), object(12)
memory usage: 3.9+ MB
```

- The data currently contains three different data types.
- The 'ID' and 'Request Age' columns are the only integer fields.
- The 'CE Number', 'GL' and 'Attribute Value' columns are floats.
  - I know these numbers will not need decimal places so I will convert them to integers later on.
- 'Triage Completed/ Rejected date' and 'UBS Masters Deactivation date' are objects. I will convert these to 'datetimes'.

In [5]:

```
# description of the data
data.describe()
```

Out[5]:

	ID	CE Number	GL	Attribute_Id	Request Age
count	30005.000000	1.317000e+04	1.802600e+04	15674.000000	30005.000000
mean	56662.202266	6.556627e+06	6.597973e+06	1082.127089	400.260123
std	18953.466155	3.082104e+06	2.627511e+06	478.488779	162.294555
min	594.000000	1.230394e+06	0.000000e+00	5.000000	2.000000
25%	42819.000000	3.272560e+06	3.795140e+06	1039.000000	300.000000
50%	61034.000000	7.527050e+06	7.444114e+06	1371.000000	328.000000
75%	72917.000000	9.218268e+06	8.680635e+06	1371.000000	505.000000
max	89939.000000	9.993248e+06	7.520680e+07	1461.000000	1121.000000

The numbers are too large to be displaced properly. It appears they are displaying in the scientific formant.

In [6]:

```
# check for null data in the all if the columns
data.isnull().any()
```

Out[6]:

```
ID                                False
Offboard Type                     False
Input Key - 1                     False
CE Number                          True
CE Name                            True
GL                                 True
Attribute_Id                       True
Attribute Value                    True
Date of Request                   False
Request Age                       False
KYC Review Date                    True
AML Risk Classification            True
Priority_Calc                     False
CMIS Parent ID                     True
CMIS Parent Name                   True
Triage Completed/Rejected date    False
UBS Masters Deactivation date     False
dtype: bool
```

There are a number of columns with null data. In the offboarding tracker not all fields are required to be completed for a request to be valid. If you want a client offboarded you do not need to include all GL numbers as client entities can be linked to multiple GLs.

There are data fields which I will not need for my regression analysis as they are related to data points which related to a client (e.g. account numbers). I will drop these as part of my data clean-up exercises.

In [7]:

```
# checking what values are in the priority calc coluns
data["Priority_Calc"].unique()
```

Out[7]:

```
array(['P2', 'P1', 'P3', 'P4', 'P0'], dtype=object)
```

In [8]:

```
# count of priority type to see which is the most common
data.Priority_Calc.value_counts()
```

Out[8]:

```
P2    19704
P1     5136
P4     4574
P0      361
P3      230
Name: Priority_Calc, dtype: int64
```

- I am going to remove the "P" in this column as this will allow me more flexibility when it comes to visualising the data by treating them as integers.

Cleaning the data

- I am going to drop any CID data from the data frame in order to protect myself against any potential data leak issues as per the UBS data handling guidance.
- This data is not required as I can use the attributes numbers to lookup client details if required.
- This will also help remove errors due to names being wrong or formatted differently.

In [9]:

```
# dropping the columns which contain CID
data.drop(['CE Name','CMIS Parent Name'], axis = 1, inplace = True)
```

In [10]:

```
# checking the columns have been successfully dropped
data.head(2)
```

Out[10]:

	ID	Offboard Type	Input Key - 1	CE Number	GL	Attribute_Id	Attribute Value	Date of Request	Request Age	KYC Review Date
0	594	Remove Credit Risk	RXM	NaN	NaN	NaN	CG0753	13/02/2017	859	NaN
1	595	Remove Credit Risk	RXM	NaN	NaN	NaN	C31534	13/02/2017	859	NaN

- All CID data has now been removed from the data.

In [11]:

```
# renaming the column names to make the coding easier
data.rename(columns={'Offboard Type':'offboard_type',
                     'Input Key - 1':'input_key',
                     'Date of Request':'date_of_request',
                     'Triage Completed/Rejected date':'triage_completed',
                     'Priority_Calc':'priority',
                     'Request Age':'age',
                     'UBS Masters Deactivation date':'execution_date'}, inplace=True
)
```

- I have renamed the columns that I will be using the most will make the coding easier.
- I have also removed the white space and replacing it with an underscore in order to make the code more efficient to create.
- The column headers I have not amended will be dropped later on.

In [12]:

```
#checking the changes have been successful
data.head(5)
```

Out[12]:

	ID	offboard_type	input_key	CE Number	GL	Attribute_Id	Attribute Value	date_of_request	age
0	594	Remove Credit Risk	RXM	NaN	NaN	NaN	CG0753	13/02/2017	859
1	595	Remove Credit Risk	RXM	NaN	NaN	NaN	C31534	13/02/2017	859
2	596	Remove Credit Risk	RXM	NaN	NaN	NaN	CD9455	13/02/2017	859
3	597	Remove Credit Risk	RXM	NaN	NaN	NaN	CI2041	13/02/2017	859
4	598	Remove Credit Risk	RXM	NaN	NaN	NaN	CD9454	13/02/2017	859

In [13]:

```
# removng the 'P' character found at the start of every priority
data["priority"] = data["priority"].replace({'P':''}, regex = True)
```

I have used regular expression to remove the "P" from the priority field. This "P" repeats at the same place in every column. Since removing one character which repeats is straightforward I can do this without impacting the data and then convert the remaining character into an integer.

In [14]:

```
#check the change has been successful
data["priority"].unique()
```

Out[14]:

```
array(['2', '1', '3', '4', '0'], dtype=object)
```

- The "P" has been removed and the data type has changed to an object
- I am going to convert the above priorities to integers as they are already whole numbers.
- Setting the data type to integers will make the visualisations easier and clearer.

In [15]:

```
# converting the prioity column in an interger
data["priority"] = data["priority"].astype(int)
```

In [16]:

```
#checking the conversion has been successful
data["priority"].unique()
```

Out[16]:

```
array([2, 1, 3, 4, 0], dtype=int64)
```

The conversion from an object into integers has been successful.

In [17]:

```
# checking all the input keys in the data
data["input_key"].unique()
```

Out[17]:

```
array(['RXM', 'CE', 'CMIS Parent', 'GL', 'XRef', 'Xref',
       'Trading Name / CRM / Shortname', 'AuthProd'], dtype=object)
```

In [18]:

```
# showing input keys in frequency
data.input_key.value_counts()
```

Out[18]:

```
Xref                             14483
CE                                8398
RXM                               2682
GL                                2387
XRef                              1189
CMIS Parent                        838
Trading Name / CRM / Shortname      24
AuthProd                             4
Name: input_key, dtype: int64
```

- Some of the input keys are quite long and repeated due to Python being case sensitive.
- I am going amend the required input keys to:
    1. Merge the data point that are the same
    2. Make the longer names easier to code with

In [19]:

```
# amending the input keys so they are easier to read
data["input_key"] = data["input_key"].replace({'XRef':'Xref'}, regex = True)
data["input_key"] = data["input_key"].replace({'Trading Name / CRM / Shortname':'TN'},
regex = True)
data["input_key"] = data["input_key"].replace({'CMIS Parent':'CMIS'}, regex = True)
```

In [20]:

```
# checking the amendments have worked
data.input_key.value_counts()
```

Out[20]:

```
Xref        15672
CE           8398
RXM          2682
GL           2387
CMIS          838
TN             24
AuthProd        4
Name: input_key, dtype: int64
```

The change has worked successfully as the 'Xref' value now contains the population that were originally classed as 'XRef'.

In [21]:

```
# grouping the data by priority to show the commonality.
data.groupby(['priority']).count()
```

Out[21]:

priority	ID	offboard_type	input_key	CE Number	GL	Attribute_Id	Attribute Value	date_of_req
0	361	361	361	361	14	12	88	
1	5136	5136	5136	3486	1940	1817	2495	
2	19704	19704	19704	7887	13006	11215	15387	1
3	230	230	230	230	63	6	18	
4	4574	4574	4574	1206	3003	2624	3865	

- Grouping requests by priority will give me a good view of what the most common types of request are.
- I will expand on this by grouping with other data points

In [22]:

```
# grouping by priority and input key to see if I can identify any further details
data.groupby(['priority','input_key']).count()
```

Out[22]:

		ID	offboard_type	CE Number	GL	Attribute_Id	Attribute Value	date_of_requ
priority	**input_key**							
0	**CE**	347	347	347	0	0	76	
	GL	2	2	2	2	0	0	
	Xref	12	12	12	12	12	12	
1	**CE**	3181	3181	3181	2	0	714	3
	GL	121	121	115	121	0	0	
	RXM	17	17	0	0	0	17	
	Xref	1817	1817	190	1817	1817	1764	1
2	**AuthProd**	4	4	4	4	4	0	
	CE	4014	4014	4014	1	0	764	4
	CMIS	1	1	0	0	0	0	
	GL	1809	1809	1809	1809	0	797	1
	RXM	2664	2664	4	3	0	2664	2
	TN	1	1	0	0	0	1	
	Xref	11211	11211	2056	11189	11211	11161	11
3	**CE**	165	165	165	0	0	11	
	CMIS	2	2	2	0	0	1	
	GL	57	57	57	57	0	0	
	Xref	6	6	6	6	6	6	
4	**CE**	691	691	691	0	0	365	
	CMIS	835	835	0	0	0	834	
	GL	398	398	374	398	0	16	
	RXM	1	1	0	0	0	1	
	TN	23	23	1	0	0	23	
	Xref	2626	2626	140	2605	2624	2626	2

- The largest request type offboarding process are Xref closures at the P2 level.
- Xref closures are the simplest cases so I expect to see that these have been completed fairly quickly. In the past Xref requests have been a lower priority for the business than CE offboards which may have caused them to have age. This is due to the risk related to these request is a lot lower than requests at the CE level. This might impact the final analysis output.
- Offboarding are bringing in some automated solutions into offboarding in December 2019 which should reduce the time for all offboard requests.

In [23]:

```
# group by input key
data.groupby(['input_key']).count()
```

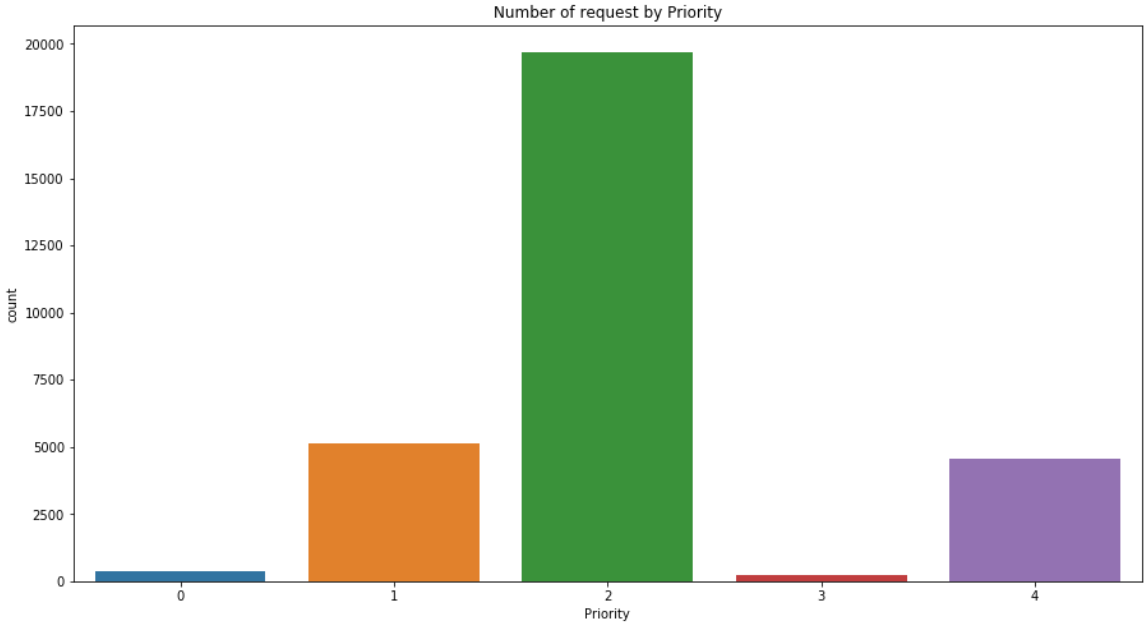
Out[23]:

input_key	ID	offboard_type	CE Number	GL	Attribute_Id	Attribute Value	date_of_request	a
AuthProd	4	4	4	4	4	0	4	
CE	8398	8398	8398	3	0	1930	8398	83
CMIS	838	838	2	0	0	835	838	8
GL	2387	2387	2357	2387	0	813	2387	23
RXM	2682	2682	4	3	0	2682	2682	26
TN	24	24	1	0	0	24	24	
Xref	15672	15672	2404	15629	15670	15569	15672	156

- This new 'group by' view shows me that the CE pipeline is the second most popular input after Xref.
- Due to the majority of regulatory requirements being performed at the client entity level, these CE level requests have always taken business priority.

In [24]:

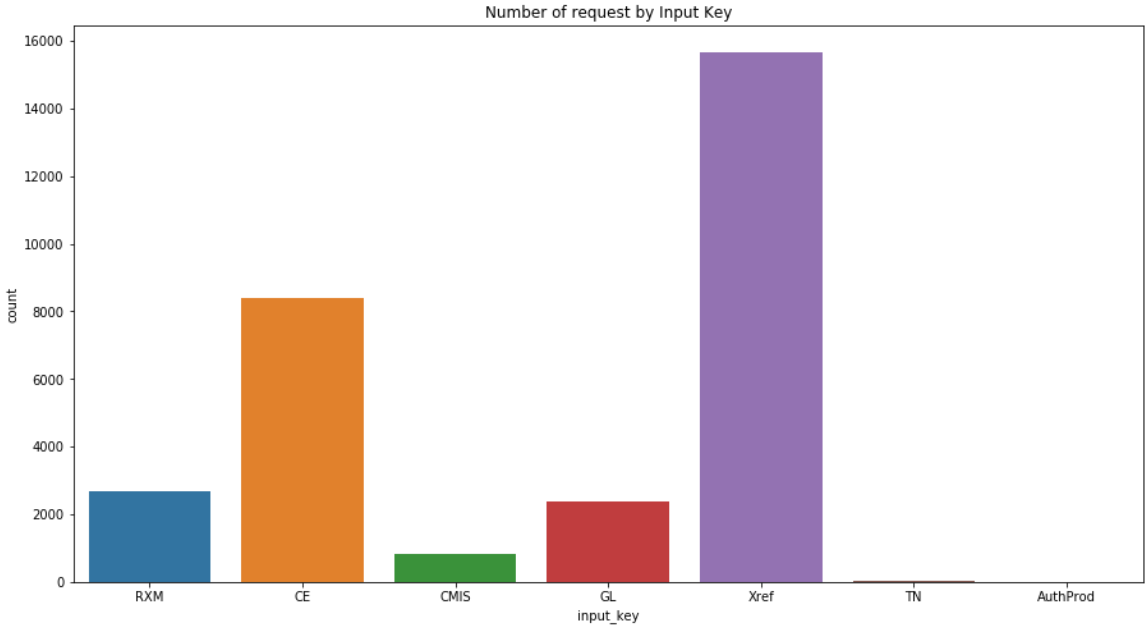
```
# visualing the data by priority
plt.figure(figsize=(15,8))
sns.countplot(x = 'priority', data = data)
plt.title("Number of request by Priority")
plt.xlabel('Priority')
plt.show()
```



- The above plot and table shows me that there is an unusually high amount of cases with the priority of 2.
- This could be an underlying problem with how the prioritisation is worked out.
  - There is possibility offboarding are just getting more request types that are level 2 priority based on the input they are submitted at.
- The previous table shows offboarding get a large amount of Xref requests with the priority level of 2.
- Based off this project data I will be bringing this up with management to reassess that priority matrix to make sure requests are more evenly spread between the priorities. This will reduce unnecessary pressure on the team.

In [25]:

```
#visualisation of requests by input key
plt.figure(figsize=(15,8))
sns.countplot(x = 'input_key', data = data)
plt.title("Number of request by Input Key")
plt.xlabel('input_key')
plt.show()
```



- CE input are the second most popular as there are four different types of offboard offered at this level.
- I'm going to create a new variable containing only the columns I need to perform the analysis.

In [26]:

```
# creating a new variable with just the inputs I will need for the regression analysis
regdata = data[["offboard_type","input_key","date_of_request","age","priority","triage_
completed","execution_date"]].copy()
```

In [27]:

```
# checking the new variable has worked
regdata.head(5)
```

Out[27]:

	offboard_type	input_key	date_of_request	age	priority	triage_completed	execution_date
0	Remove Credit Risk	RXM	13/02/2017	859	2	23/10/2018	01/04/2019
1	Remove Credit Risk	RXM	13/02/2017	859	2	11/02/2018	18/01/2019
2	Remove Credit Risk	RXM	13/02/2017	859	2	23/10/2018	18/01/2019
3	Remove Credit Risk	RXM	13/02/2017	859	2	23/10/2018	18/01/2019
4	Remove Credit Risk	RXM	13/02/2017	859	2	11/02/2018	18/01/2019

In [28]:

```
# groupby input to make sure all data is there
regdata.groupby(['input_key']).count()
```

Out[28]:

input_key	offboard_type	date_of_request	age	priority	triage_completed	execution_date
AuthProd	4	4	4	4	4	4
CE	8398	8398	8398	8398	8398	8398
CMIS	838	838	838	838	838	838
GL	2387	2387	2387	2387	2387	2387
RXM	2682	2682	2682	2682	2682	2682
TN	24	24	24	24	24	24
Xref	15672	15672	15672	15672	15672	15672

- The new variable was created successfully.
- I now need to convert the following columns into datatime so they can be used effectively:
  - date_of_request
  - triage_completed
  - execution_date

In [29]:

```
# changing the date inputs to datetime so they can be used for analysis
regdata['date_of_request'] = pd.to_datetime(regdata['date_of_request'])
regdata['triage_completed'] = pd.to_datetime(regdata['triage_completed'])
regdata['execution_date'] = pd.to_datetime(regdata['execution_date'])
```

In [30]:

```
# checking the dataset to see if the changes have been performed correctly
regdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30005 entries, 0 to 30004
Data columns (total 7 columns):
offboard_type       30005 non-null object
input_key           30005 non-null object
date_of_request     30005 non-null datetime64[ns]
age                 30005 non-null int64
priority            30005 non-null int32
triage_completed    30005 non-null datetime64[ns]
execution_date      30005 non-null datetime64[ns]
dtypes: datetime64[ns](3), int32(1), int64(1), object(2)
memory usage: 1.5+ MB
```

- Now I have converted and cleaned the data I believe there are a few data points I don't currently have but can derive from the data which will be essential for good analysis.

In [31]:

```
# check new columns with the date information
regdata['days_in_triage'] = (regdata['triage_completed'] - regdata['date_of_request']).
dt.days
regdata['days_in_execution'] = (regdata['execution_date'] - regdata['triage_completed'
]).dt.days
regdata['total_days_to_complete'] = (regdata['execution_date'] - regdata['date_of_reque
st']).dt.days
```

- I have created a new column for total_days_to_complete as the 'age' column can change on the SharePoint if a member of the team updates a closed request. This new column has accurate request age data.

In [32]:

```
# checking the columns have been created successfully
regdata.head()
```

Out[32]:

	offboard_type	input_key	date_of_request	age	priority	triage_completed	execution_date
0	Remove Credit Risk	RXM	2017-02-13	859	2	2018-10-23	2019-01-04
1	Remove Credit Risk	RXM	2017-02-13	859	2	2018-11-02	2019-01-18
2	Remove Credit Risk	RXM	2017-02-13	859	2	2018-10-23	2019-01-18
3	Remove Credit Risk	RXM	2017-02-13	859	2	2018-10-23	2019-01-18
4	Remove Credit Risk	RXM	2017-02-13	859	2	2018-11-02	2019-01-18

In [33]:

```
# checking what the new column types are
regdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30005 entries, 0 to 30004
Data columns (total 10 columns):
offboard_type             30005 non-null object
input_key                 30005 non-null object
date_of_request           30005 non-null datetime64[ns]
age                       30005 non-null int64
priority                  30005 non-null int32
triage_completed          30005 non-null datetime64[ns]
execution_date            30005 non-null datetime64[ns]
days_in_triage            30005 non-null int64
days_in_execution         30005 non-null int64
total_days_to_complete    30005 non-null int64
dtypes: datetime64[ns](3), int32(1), int64(4), object(2)
memory usage: 2.2+ MB
```

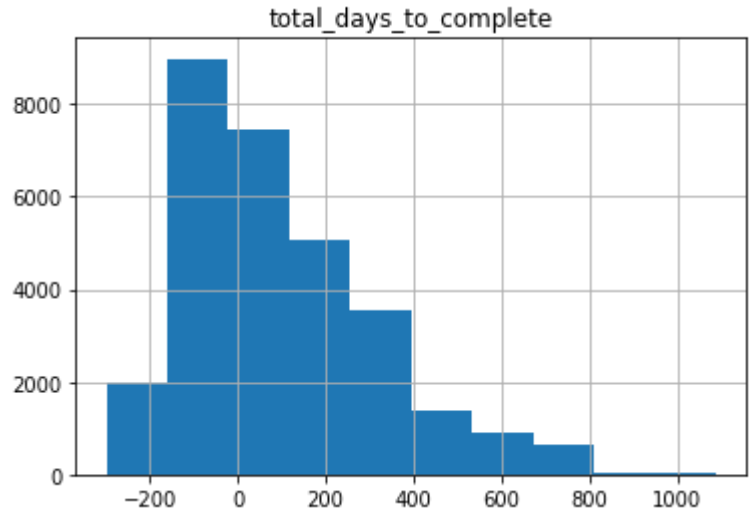
The new columns are integers so I can continue to use these for more detailed analysis and visualisation.

In [34]:

```
#check if quality is continious
regdata.hist(column=['total_days_to_complete'])
```

Out[34]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000000127E8AC
8>]],
      dtype=object)
```



The above plot shows me two important points:

- The data is continuous which means I can use this to perform regression analysis.
- There are negative values on the x-axis I will need to investigate.

In [35]:

```
# statistical analysis on the total days to complete
regdata['total_days_to_complete'].describe()
```

Out[35]:

```
count    30005.000000
mean        86.593534
std        228.975032
min       -300.000000
25%       -134.000000
50%         84.000000
75%        218.000000
max       1088.000000
Name: total_days_to_complete, dtype: float64
```

- Three days to offboard a client happens when it is an urgent request and there are no accounts in the MasterFiles client accounts system.
- The mean of 264 days to offboard a client would not apply for cases received in 2019. This is old process data distorting the result
- The max being 1088 days (Just under 3 years!) is due to data quality issues for RXMs.
    - I expect this to be closed soon as the team has just solved the underlying data quality issues with these cases so they can be processed.

In [36]:

```
# statistical analysis on the age of requests is different to total days to complete
regdata['age'].describe()
```

Out[36]:

```
count    30005.000000
mean       400.260123
std        162.294555
min          2.000000
25%        300.000000
50%        328.000000
75%        505.000000
max       1121.000000
Name: age, dtype: float64
```

This highlights issue of using the 'age' column from the data. It is inaccurate as rows can be edited after the case has closed. This explains why the mean is 400 days and the max has move to 1121 days.

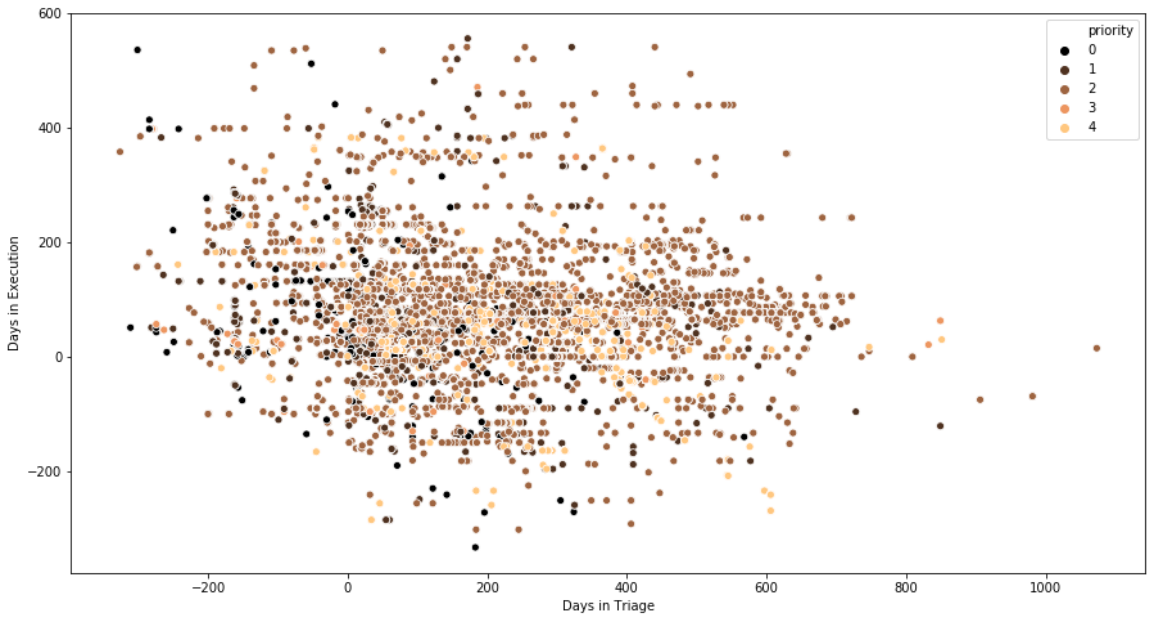
Visualising the data

In [37]:

```
#scatterplot of days in execution and days in triage
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'days_in_triage',y = 'days_in_execution', data = regdata, palette = 'copper', hue='priority', legend = 'full')
plt.title('')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[37]:

```
Text(0, 0.5, 'Days in Execution')
```



- The above plot shows negative time. This is a problem with the underlying data.
- Different date formats are likely the cause of this (dd-mm-yyy or mm-dd-yyyy) however I have no way to prove this conclusively.
- I am going to drop this negative data as it is a problem at the data source.
- I am going to take an action from this to ask the respective teams to audit their data as it's currently inaccurate. This will be brought up at the weekly team governance meeting I chair.

In [38]:

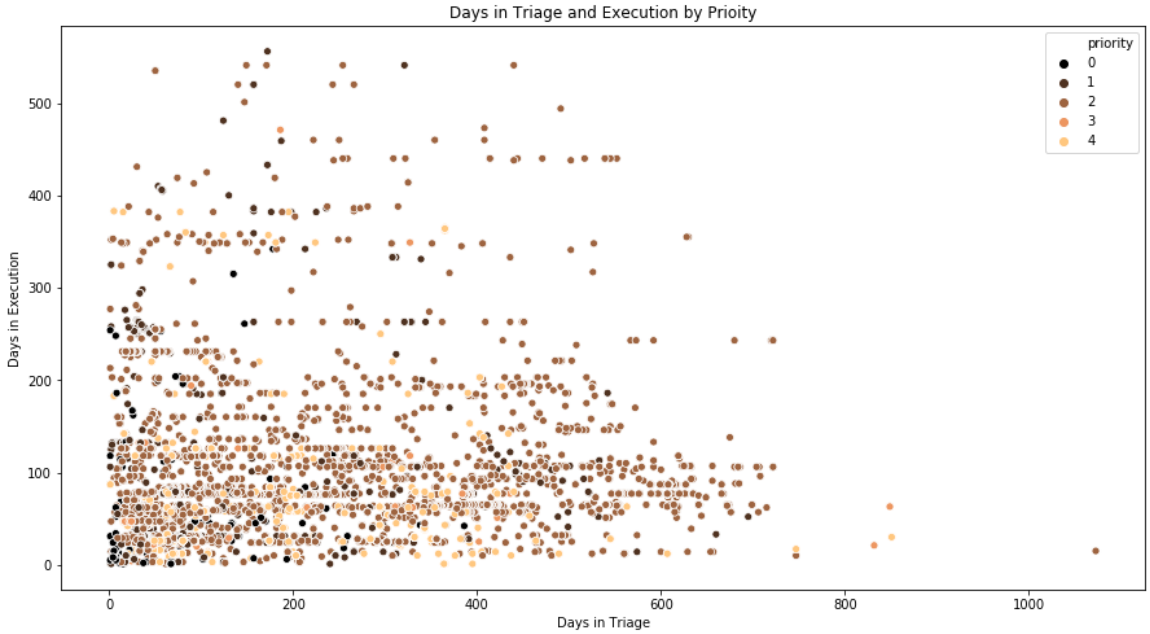
```
# removing the days which come back negitive due to data quality issues
regdata = regdata[(regdata['days_in_triage']>0) & (regdata['days_in_execution']>0) & (r
egdata['total_days_to_complete']>0)]
```

In [39]:

```
#replot the data to confirm the negative values removed, by priority
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'days_in_triage',y = 'days_in_execution', data = regdata, palette =
'copper',hue='priority', legend = "full")
plt.title('Days in Triage and Execution by Prioity')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[39]:

```
Text(0, 0.5, 'Days in Execution')
```



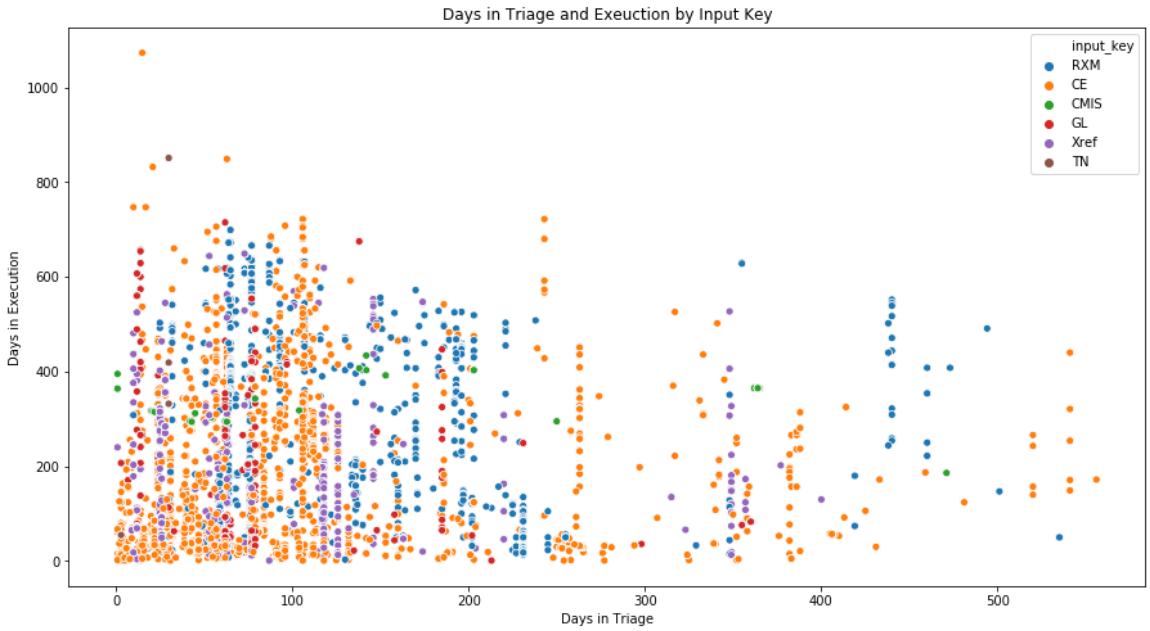
- The visualisation does not suggest any strong correlation between the priority level and time cases spend with the two offboarding teams.
- There are a lot of data points which seem inaccurate. There are cases that have been over 400 days in Triage and another 500 days with execution.
  - This could be due to the old process where populations were batched up and worked. This lead to a lot of delays in Triage
  - Delays in execution could be due to waiting for IT scripts to ran. Now the team have an self-service deactivation tool which stops this from happening.
- There is a promising cluster of P0 cases in the bottom corner. This is expected as the SLA for these cases is 30 days and this tends to hit consistently.

In [40]:

```
# plot the data by input key
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'days_in_execution',y = 'days_in_triage', data = regdata, hue='input_key', legend = "full")
plt.title('Days in Triage and Exeuction by Input Key')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[40]:

```
Text(0, 0.5, 'Days in Execution')
```



- There are few interesting patterns in the data. Columns of CE offboard requests are appearing in the visualisation.
- This could be due to the old execution process where batches of CEs were deactivated by IT script.
- Now there is a much more flexible process which explains the more fluctuating numbers you can see in the bottom left of the plot.
- The same applies to Triage as in April 2018 the process moved to a more flexible model where cases were handed over to the execution team when they were ready. Previously all CEs had to wait for all checks to be completed on a population before they could be passed over.

In [41]:

```
# groupby input to check the different in the numbers the removeal of negitive dates ma
de
regdata.groupby(['input_key']).count()
```

Out[41]:

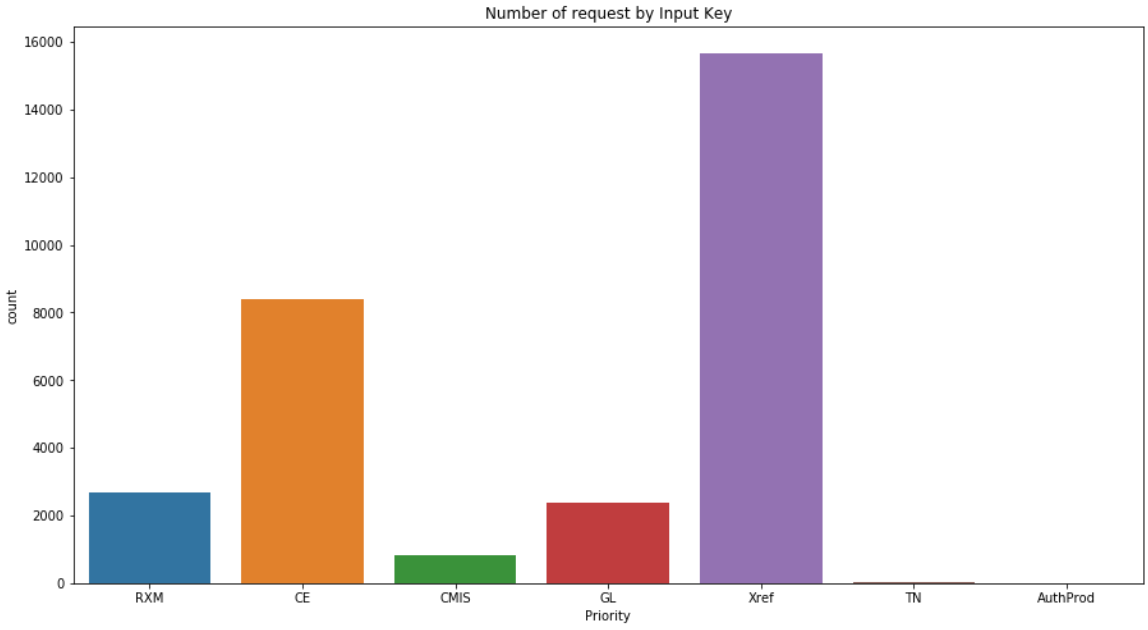
input_key	offboard_type	date_of_request	age	priority	triage_completed	execution_date	c
CE	5047	5047	5047	5047	5047	5047	
CMIS	388	388	388	388	388	388	
GL	1552	1552	1552	1552	1552	1552	
RXM	2446	2446	2446	2446	2446	2446	
TN	24	24	24	24	24	24	
Xref	2502	2502	2502	2502	2502	2502	

In [42]:

```
# reploting the input key with amended data
plt.figure(figsize=(15,8))
sns.countplot(x = 'input_key', data = data)
plt.title("Number of request by Input Key")
plt.xlabel('Priority')
```

Out[42]:

```
Text(0.5, 0, 'Priority')
```



This has had a larger than expected impact on the data

- RXM went from 2,682 to 2,446 (-9%)
- CE went from 8,398 to 5,047 (-40%)
- CMIS went from 838 to 388 (-54%)
- GL went from 2,387 to 1,552 (-35%)
- Xref went from 15,672 to 2,502 (-84%)

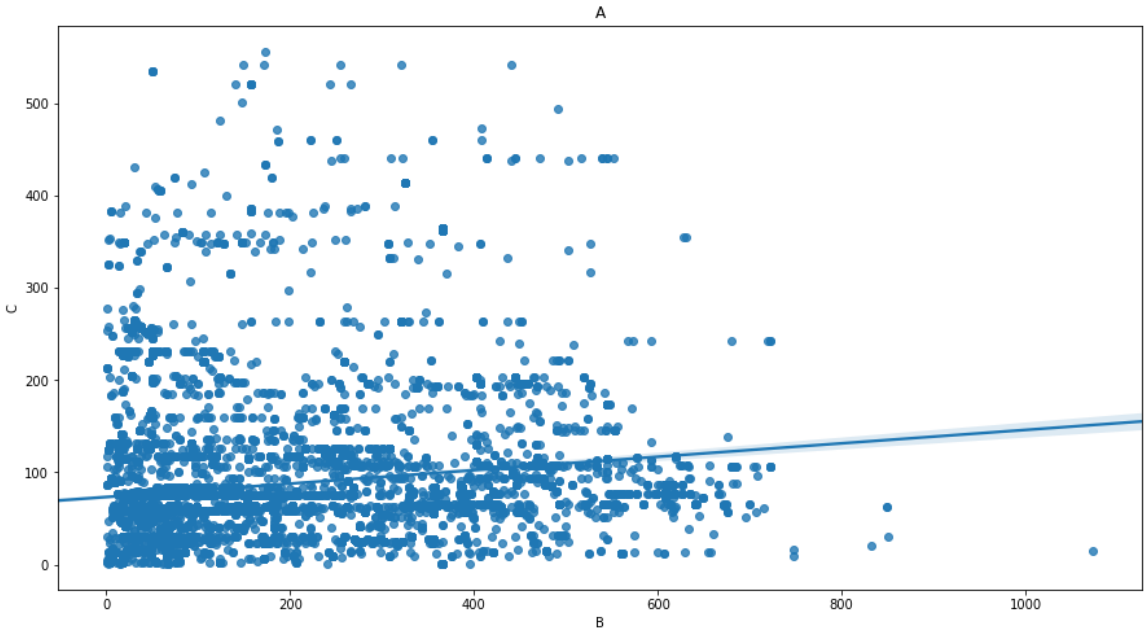
This is just over a 60% reduction in the usable population of data. It's clear that there are a large number of data quality issues affecting offboarding. The biggest impact is the Xref request. There are the most popular type and 84% of the current data is not usable. This will impact the model output negatively as the data used is not accurately representative of the throughput time.

In [43]:

```
# reploting the input key with amended data
plt.figure(figsize=(15,8))
sns.regplot(x = "days_in_triage", y = "days_in_execution", data = regdata)
plt.title("A")
plt.xlabel("B")
plt.ylabel("C")
```

Out[43]:

```
Text(0, 0.5, 'C')
```



- There is a minor correlation between days in Triage and days in Execution. I have mentioned previously I believe this down to the old process where cases were worked in batched.
- A good next step would be to look at the data post April 2018 when the process changed and then requests received from the start of 2020 once automation has gone live.

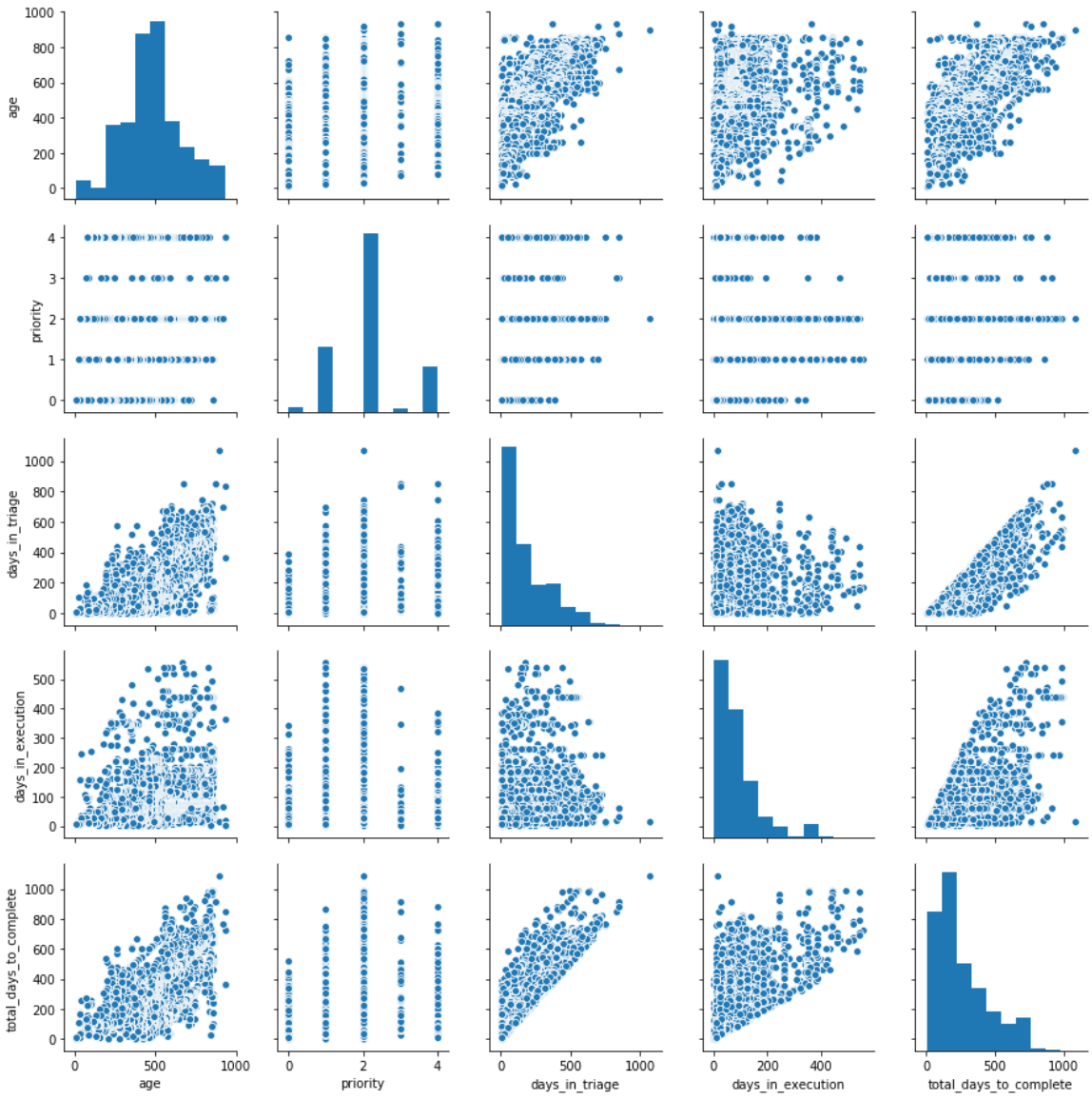
Building the regression model

In [44]:

```
# pairplot being used to highlight any possible correlations within the data and visualise the relationships
sns.pairplot(regdata)
```

Out[44]:

```
<seaborn.axisgrid.PairGrid at 0x1359f978>
```



There are a few interesting points from the above plots I can gain:

- There does not appear to an correlation between 'days in execution' and 'days in triage. The plot does not have a distinguishable pattern.
- The above confirms the new columns I created have continuous data
- The correction between 'total days' and 'days in triage' is stronger than 'days in execution'

A correlation matrix should confirm my above observations.

In [45]:

```
# Correlation matix
regdata.corr()
```

Out[45]:

	age	priority	days_in_triage	days_in_execution	total_days_to
age	1.000000	-0.131201	0.558232	0.272882	
priority	-0.131201	1.000000	0.149377	0.113603	
days_in_triage	0.558232	0.149377	1.000000	0.135978	
days_in_execution	0.272882	0.113603	0.135978	1.000000	
total_days_to_complete	0.588571	0.175628	0.896690	0.560477	

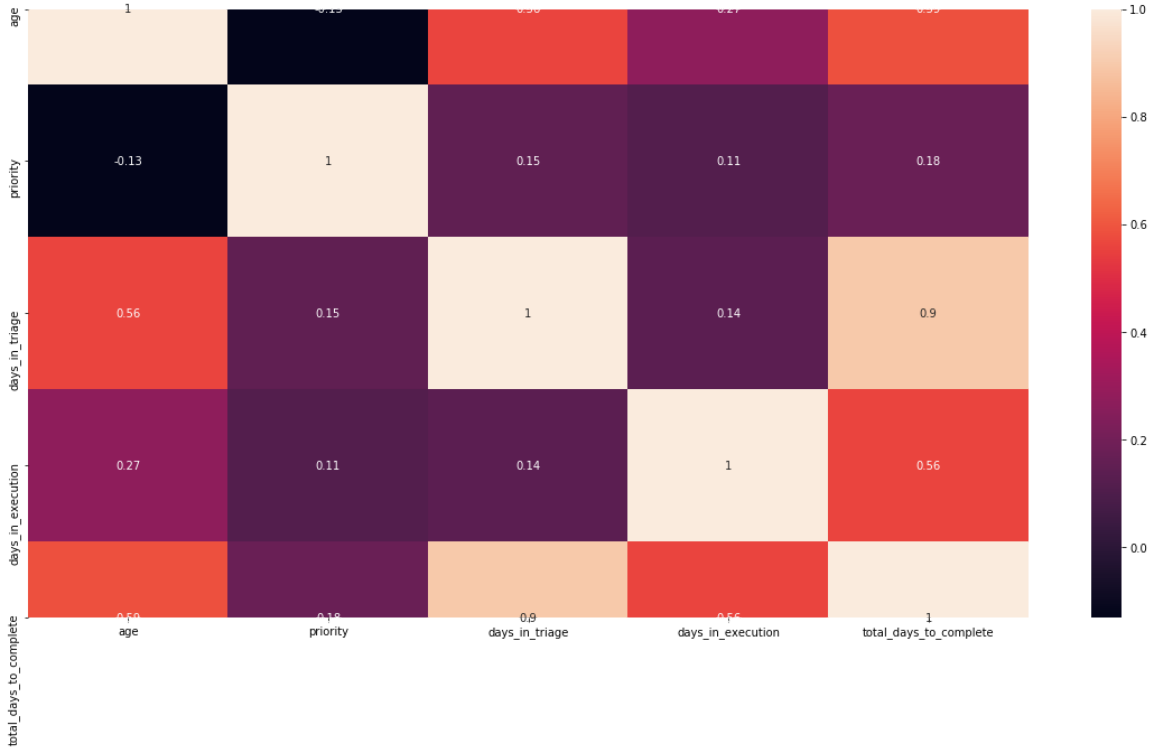
I'm going to represent the above in a visualisation for clarity.

In [46]:

```
regdata_corr = regdata.corr()
plt.figure(figsize = (20, 10))
sns.heatmap(regdata_corr , xticklabels = regdata_corr.columns, yticklabels = regdata_co
rr.columns,  annot = True)
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16c22eb8>
```



There a number of interesting observations from the above plots.

Observations on the correlations:

- 'Days in Triage' and 'total days to complete' are very positive.
- 'Days in execution' does not have a strong correlation with days in triage.
  - This is interesting as my problem statement is concerning this.
- Age has interesting bar plots.
  - I assume this will come down if the team focused on 2019 data.
- In 2020 offboarding are focusing on clearing the ageing RXM requests.
  - Once this is completed and this is rerun with updated data I believe it will demonstrate just how much quicker the processes will be working.
- 'Priority' and 'total days to complete' appear to have a weak correlation.
  - This should be explored once the backlog has been cleared.
- Priority is clearly a categorical data point so I will not be using that for my regression analysis.

In [47]:

```
# coefficient of A & B
regdata['days_in_triage'].corr(regdata['days_in_execution'])
```

Out[47]:

```
0.1359780647249628
```

In [48]:

```
# coefficient of B & A
regdata['days_in_execution'].corr(regdata['days_in_triage'])
```

Out[48]:

```
0.13597806472496277
```

- There is no difference between the coefficients of A & B, and B & A.

Spliting the data for testing

- For the first round of testing I am going to spilt the data 80/20.`
  - Depending on the results of this I might rerun this with a different split due to the data size.
- I have chosen to use the days in triage and execution as these are the performance metrics for the two offboarding teams.
  - The lower the number, the quicker the team can get through cases.

In [49]:

```
# split the data to train and test the model
X = regdata[['days_in_triage']]
y = regdata[['days_in_execution']]
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.20,
                                                    random_state = 123)
```

- Random state 123 has been chosen in order to replicate the analysis.

In [50]:

```
# show the shape of the data for a sense check
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

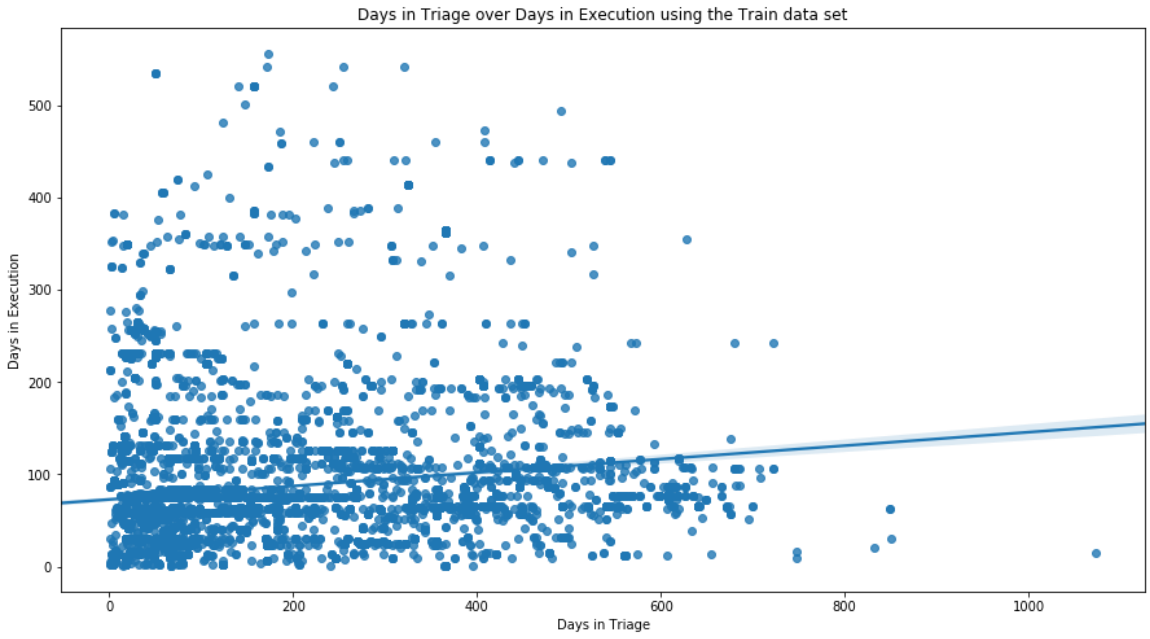
```
(9567, 1) (9567, 1)
(2392, 1) (2392, 1)
```

In [51]:

```
# sense check with the split data
Train = pd.concat([X_train,y_train], axis =1)
plt.figure(figsize=(15,8))
sns.regplot(x = "days_in_triage", y = "days_in_execution", data = Train)
plt.title('Days in Triage over Days in Execution using the Train data set')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[51]:

```
Text(0, 0.5, 'Days in Execution')
```



The above leg plot shows me there is only a slight correlation between 'days in execution' and 'days in triage'. I expected this correlation to be a lot stronger based on my knowledge of the process. This outcome has likely been impacted by the removal of all the inaccurate data previously.

The Linear Regression Model

In [52]:

```
# creating the linear regression variable
regression_model = linear_model.LinearRegression()
```

In [53]:

```
# Train the model with the training data sets
regression_model = regression_model.fit(X_train, y_train)
```

In [54]:

```
# R-Squared error
print('The R-Squared Error is:\n',regression_model.score(X_train, y_train))
```

```
The R-Squared Error is:
 0.018928752115410452
```

- The R squared number is low.
  - This tells us that there is a low strength on the dependence between days in triage and execution.
  - This does not tell us if the model is bad nor if the data is biased.

In [55]:

```
# coefficients
print('Coefficients:\n', regression_model.coef_)
```

```
Coefficients:
 [[0.07287016]]
```

In [56]:

```
#interpect
print('Intercept:\n', regression_model.intercept_)
```

```
Intercept:
 [72.84711852]
```

- The intercept of when X = 0 is 72.
- In real world terms I can interpret this and claim that cases tend to take around 70 days before they are worked by triage. The longest SLA for P4 is 90 days so this is an issue offboarding need to address as a team.
  - Based on this I can argue one of ways I can measure success going forward is to try and lower the amount of time requests are sitting in the pipeline without being worked on.

Predictions

In [57]:

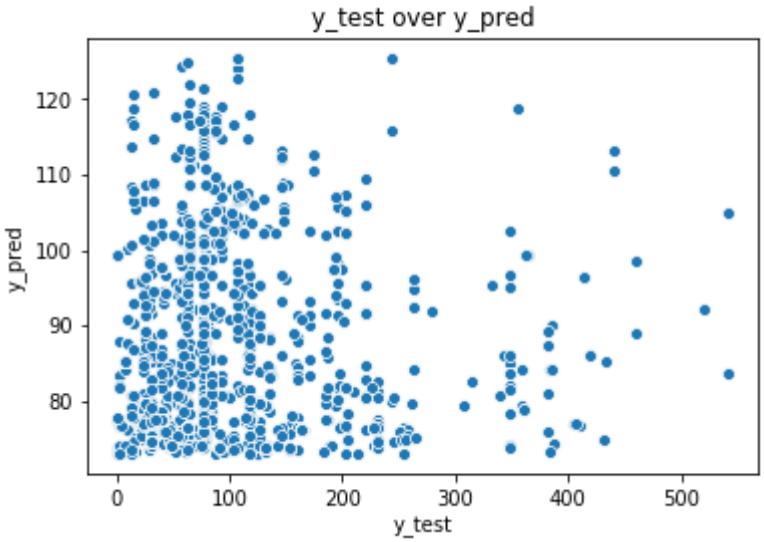
```
# using the testing data to make predicitions
y_pred = regression_model.predict(X_test)
y_pred = pd.DataFrame(y_pred)
```

In [58]:

```
y_test = y_test['days_in_execution'].reset_index()
y_obs_pred = pd.concat([y_test['days_in_execution'], y_pred], axis = 1)
y_obs_pred.columns = ['y_test','y_pred']
y_obs_pred['diff'] = y_obs_pred['y_test'] - y_obs_pred['y_pred']
r = sns.scatterplot (x = 'y_test', y = 'y_pred', data = y_obs_pred)
plt.title('y_test over y_pred')
plt.xlabel('y_test')
plt.ylabel('y_pred')
```

Out[58]:

```
Text(0, 0.5, 'y_pred')
```



- I can see that there a lot of the predicted data points are different to the actual test data.
- I need to run the Root squared mean error for accuracy of the model.

Root-Squard Mean Error (RSME)

In [59]:

```
diff = y_obs_pred['y_pred'] - y_obs_pred['y_test']
diff_sq = diff**2
mean_diff_sq = diff_sq.mean()
RSME = np.sqrt(mean_diff_sq)
RSME
```

Out[59]:

```
85.30399048530627
```

- RMSE is the root mean squared error and is used to measure how well the model has performed.
  - This done by working out the difference between the actual values and predicted values.
  - This is done by measuring the length between the predictions and actual figures.
- The RMSE error for this model was come out at 85.
  - This is a strong outcome and the model can make reasonable accurate predictions for time in execution based of the time spent in triage.

Ordinary Least Squares (OLS)

I am going fit the regression model using ordinary least squares (OLS)

In [60]:

```
regression_model = smf.ols('days_in_triage ~ days_in_execution', data = Train).fit()
print(regression_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         days_in_triage   R-squared:                       0.019
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     184.5
Date:                Tue, 14 Jul 2020   Prob (F-statistic):           1.20e-41
Time:                        12:14:25   Log-Likelihood:                -61946.
No. Observations:                9567   AIC:                         1.239e+05
Df Residuals:                    9565   BIC:                         1.239e+05
Df Model:                           1
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept           156.3723      2.296     68.107      0.000     151.872     160.873
days_in_execution     0.2598      0.019     13.585      0.000       0.222       0.297
==============================================================================
Omnibus:                     1350.220   Durbin-Watson:                   1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1986.599
Skew:                           1.077   Prob(JB):                         0.00
Kurtosis:                       3.586   Cond. No.                         172.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

- The R squared number has come out the same previously

Analysis

My original problem statement was:

- Does the time it takes to complete triage on offboarding request have an impact on the time it takes to execute an offboard request?

The model suggests that this is the case however it does not prove this outright and I have to rely on domain knowledge to justify this. I will go into detail about this in my conclusion.

Model limitations

- Linear regression (LR) models are sensitive to over-fitting however I do not believe that my model has been over fitted as the RSME has come in at 85, which as previously stated is a strong score.
- Secondly LR models are sensitive to outliers and this is an issue which may have a potential impact on my model.

The offboarding process is still within its infancy. It has only been functioning since early 2017 and the last major process transformation was April 2018. From April 2018 to now there has been numerous technological and process changes which will have affected the speed of throughput. This has caused substantial reduction in the time cases spend in the back log. Consequently this data has a lot of old cases in it which are outliers. This is going to negatively impact this model.

As a result of old outliers in the data I'm going to repeat the analysis using data from 2019 only. This due to the backlog of old request continuing to go down as our throughput gets higher.

In [61]:

```
# copy the data used for the regression model
regdata19 = regdata.copy()
```

In [62]:

```
# check the data
regdata19.head(2)
```

Out[62]:

	offboard_type	input_key	date_of_request	age	priority	triage_completed	execution_date
0	Remove Credit Risk	RXM	2017-02-13	859	2	2018-10-23	2019-01-04
1	Remove Credit Risk	RXM	2017-02-13	859	2	2018-11-02	2019-01-18

In [63]:

```
regdata19['year_of_request'] = regdata19['date_of_request'].dt.year
```

In [64]:

```
# drop the cases not from 2019
regdata19.drop(regdata19[regdata19.year_of_request != 2019].index, inplace=True)
```

In [65]:

```
# grouping by priority and input key to see if I can identify any further details
regdata19.groupby(['year_of_request']).count()
```

Out[65]:

year_of_request	offboard_type	input_key	date_of_request	age	priority	triage_completed	e
2019	444	444	444	444	444	444	

In [66]:

```
# check the shape
regdata19.shape
```

Out[66]:

```
(444, 11)
```

The data does not contain enough input for good analysis so I will change this to 2018. I believe this is due to the transition of request capture to the Client Lifecycle Management Tool (CLMT). CLMT is an internal tool which has automated approvals capture when the bank wants to offboard a client. It has a cleaner data set. Analysis on this population will only be worthwhile when the required development has been finished and the SharePoint site has been decommissioned.

In [67]:

```
# copy the data used for the regression model
regdata18 = regdata.copy()
```

In [68]:

```
regdata18['year_of_request'] = regdata18['date_of_request'].dt.year
```

In [69]:

```
# drop all cases taken in to the offboaridng pipeline not in 2018
regdata18.drop(regdata18[regdata18.year_of_request != 2018].index, inplace=True)
```

In [70]:

```
# check the data
regdata18.groupby(['year_of_request']).count()
```

Out[70]:

year_of_request	offboard_type	input_key	date_of_request	age	priority	triage_completed
2018	5972	5972	5972	5972	5972	5972

In [71]:

```
# check the shape
regdata18.shape
```

Out[71]:

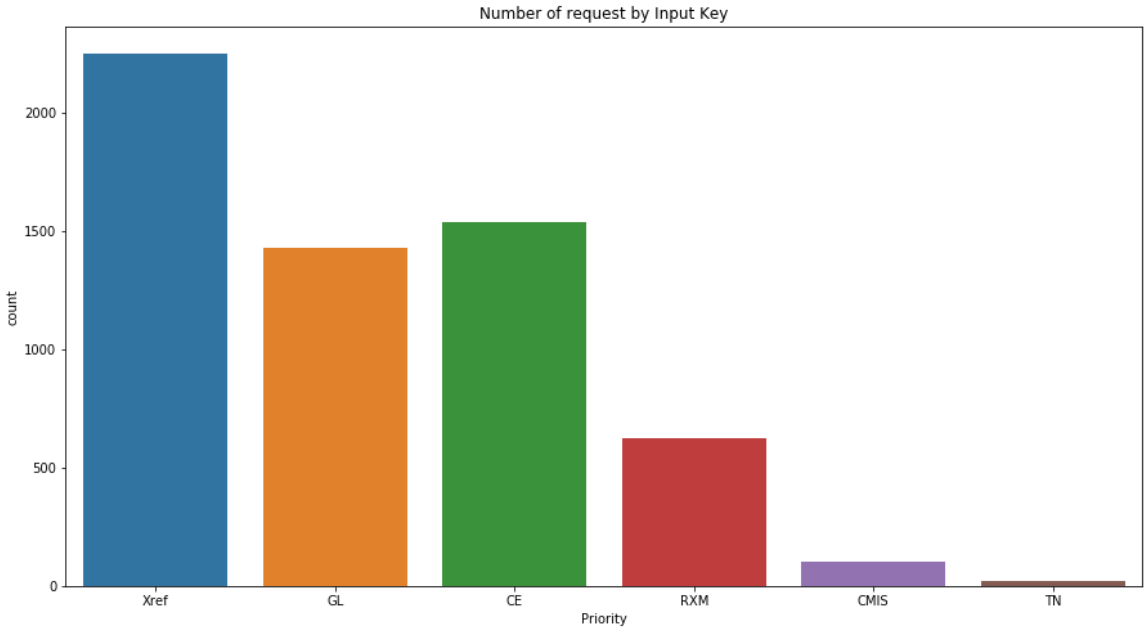
```
(5972, 11)
```

In [72]:

```
# countplot by input key
plt.figure(figsize=(15,8))
sns.countplot(x = 'input_key', data = regdata18)
plt.title("Number of request by Input Key")
plt.xlabel('Priority')
```

Out[72]:

```
Text(0.5, 0, 'Priority')
```



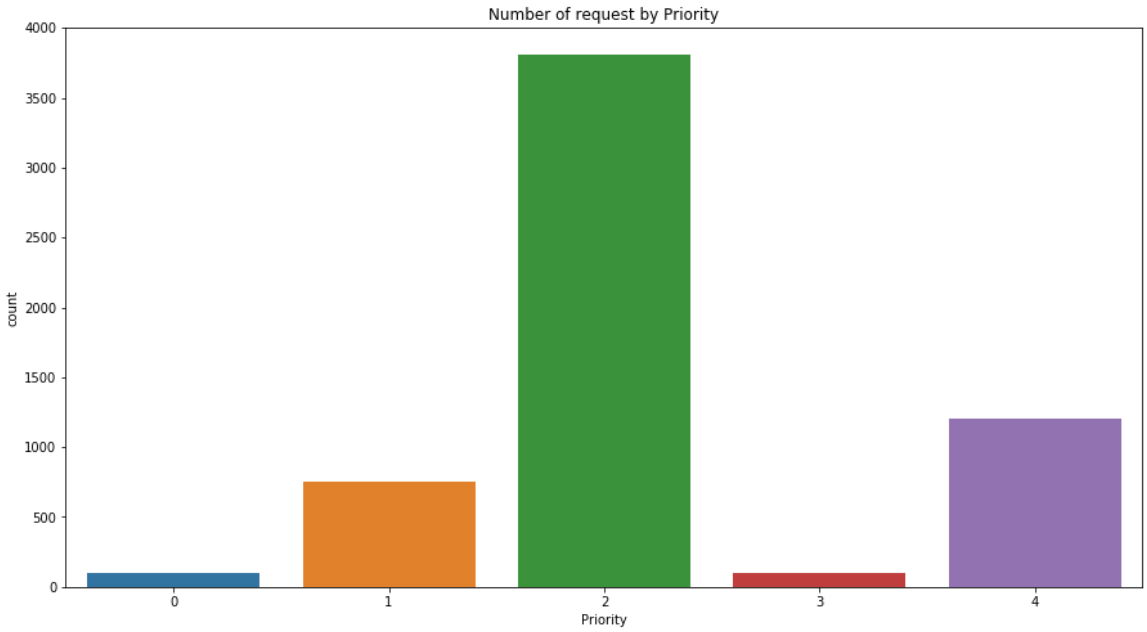
The 2018 data has a different shape compared to the overall population. There are closure amount of GL and CE requests than in the full population. CMIS requests fell dramatically in 2018.

In [73]:

```
# countplot by priority
plt.figure(figsize=(15,8))
sns.countplot(x = 'priority', data = regdata18)
plt.title("Number of request by Priority")
plt.xlabel('Priority')
```

Out[73]:

```
Text(0.5, 0, 'Priority')
```



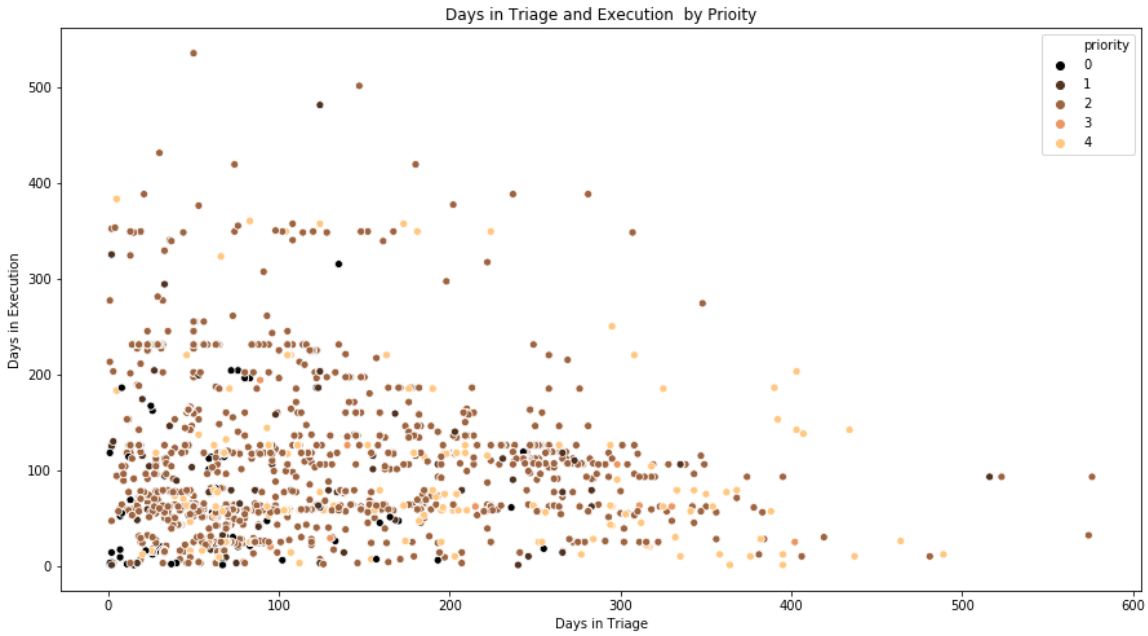
The barplot tell a similar story to the data as a whole, priority 2 and xref requests are the most popular.

In [74]:

```
# replot the data to confirm the negtive values removed, by prioity
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'days_in_triage',y = 'days_in_execution', data = regdata18, palette
= 'copper',hue='priority', legend = "full")
plt.title('Days in Triage and Execution  by Prioity')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[74]:

```
Text(0, 0.5, 'Days in Execution')
```



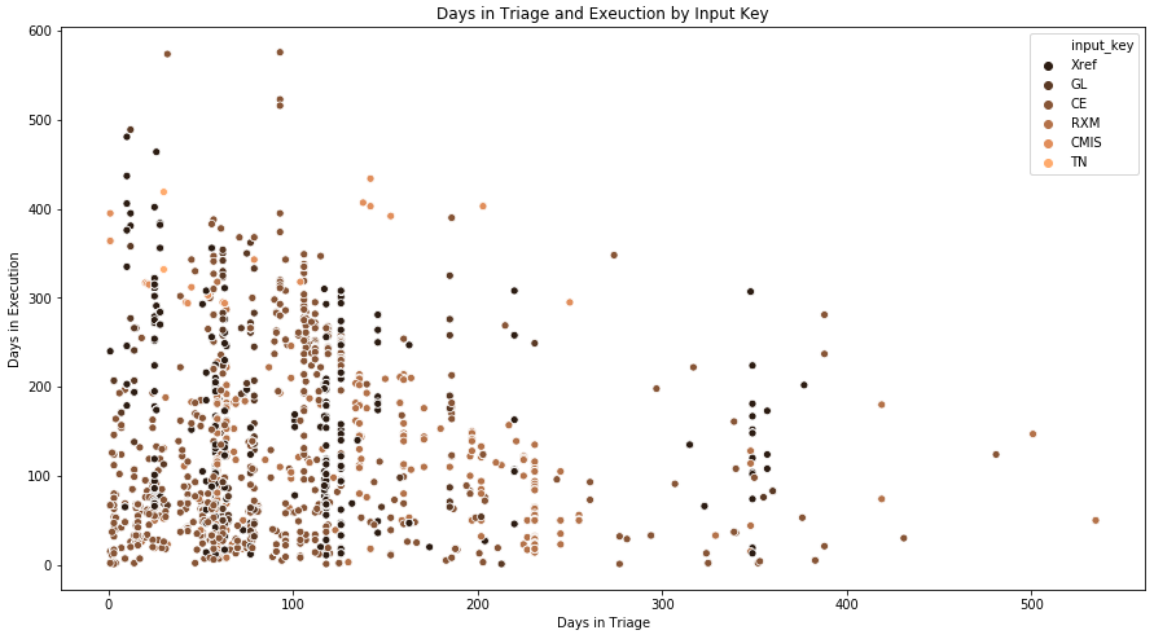
- There no correlations that immediately stand out from this visualisation.

In [75]:

```
# plot the 2019 data by input ket
plt.figure(figsize=(15,8))
sns.scatterplot(x = 'days_in_execution',y = 'days_in_triage', data = regdata18, palette
= 'copper', hue='input_key', legend = "full")
plt.title('Days in Triage and Exeuction by Input Key')
plt.xlabel('Days in Triage')
plt.ylabel('Days in Execution')
```

Out[75]:

```
Text(0, 0.5, 'Days in Execution')
```



There are similar columns in this data especially with the Xref requests that have spent the same amount of time in triage and longer time in execution.

In [76]:

```
# split the data to train and test the model
X = regdata18[['days_in_triage']]
y = regdata18[['days_in_execution']]
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.25,
                                                    random_state = 123)
```

In [77]:

```
# creating the linear regression variable
regression_model = linear_model.LinearRegression()
```

In [78]:

```
# Train the model with the training data sets
regression_model = regression_model.fit(X_train, y_train)
```

In [79]:

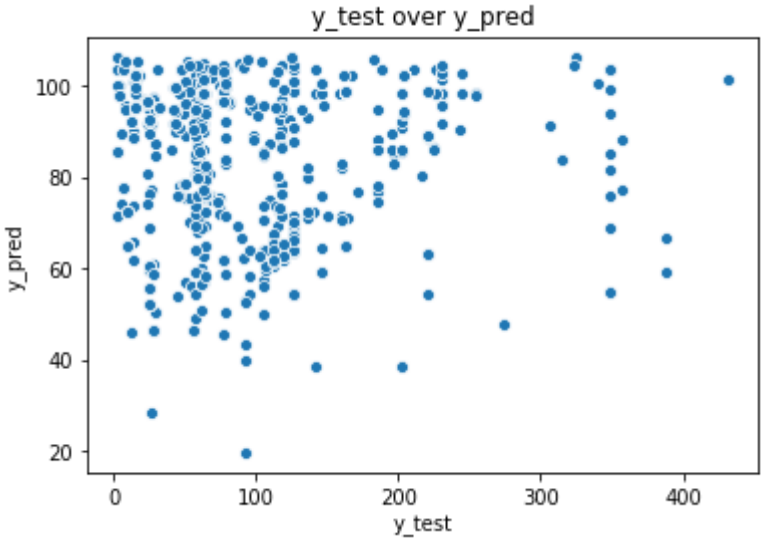
```
# using the testing data to make predicitions
y_pred = regression_model.predict(X_test)
y_pred = pd.DataFrame(y_pred)
```

In [80]:

```
y_test = y_test['days_in_execution'].reset_index()
y_obs_pred = pd.concat([y_test['days_in_execution'], y_pred], axis = 1)
y_obs_pred.columns = ['y_test','y_pred']
y_obs_pred['diff'] = y_obs_pred['y_test'] - y_obs_pred['y_pred']
r = sns.scatterplot (x = 'y_test', y = 'y_pred', data = y_obs_pred)
plt.title('y_test over y_pred')
plt.xlabel('y_test')
plt.ylabel('y_pred')
```

Out[80]:

```
Text(0, 0.5, 'y_pred')
```



- The test data and predications are less accurate when using a smaller data set. This is due to less data being able to inform the prediction outcomes.

In [81]:

```
diff = y_obs_pred['y_pred'] - y_obs_pred['y_test']
diff_sq = diff**2
mean_diff_sq = diff_sq.mean()
RSME = np.sqrt(mean_diff_sq)
RSME
```

Out[81]:

```
73.65263443684623
```

- The RSME is less with the smaller data set.
- This is likely due to the outliers caused by 2018 being the year of the major process changes. The workflow moved to a flexible model so throughput increased.

In [82]:

```
regression_model = smf.ols('days_in_triage ~ days_in_execution', data = Train).fit()
print(regression_model.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:         days_in_triage   R-squared:
0.019
Model:                            OLS   Adj. R-squared:
0.019
Method:                 Least Squares   F-statistic:                      1
84.5
Date:                Tue, 14 Jul 2020   Prob (F-statistic):           1.20
e-41
Time:                        12:14:27   Log-Likelihood:                -61
946.
No. Observations:                9567   AIC:                          1.239
e+05
Df Residuals:                    9565   BIC:                          1.239
e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
===========
                         coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
Intercept           156.3723      2.296     68.107      0.000     151.872
160.873
days_in_execution     0.2598      0.019     13.585      0.000       0.222
0.297
==============================================================================
====
Omnibus:                     1350.220   Durbin-Watson:
1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             198
6.599
Skew:                           1.077   Prob(JB):
0.00
Kurtosis:                       3.586   Cond. No.
172.
==============================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

Conclusion

- Does the time it takes to complete triage on an offboarding request have an impact on the time it takes to execute the same offboarding request?

My analysis has let me to conclude that this hypothesis is true. Based on the prediction from my model and my knowledge of the process I can justify this.

More complicated requests tend to spend longer in triage, regardless of the time spent before they are started to be work on. This complexity is due to different areas of the bank using the clients' accounts. This means more checks must be completed. The shared data model further complicated this it causes a knock on effect in execution as the accounts are likely to be open in more trading systems and this caused the time in execution being longer as the team having to spend more time getting confirmation that client's accounts are closed in more systems.

As triage comes first in the process the days in triage population will be artificially raised as the team can only deal with a certain number of cases at a time. The offboarding team inherited a large backlog of requests and some of the still open requests age back to 2017. These have yet to be cleared due to ongoing data quality issues which need to be resolved first.

Once the process is fully stabilised and the backlog is cleared this hypothesis and model could be reused and tested more effectively as the data will be cleaner. One of the biggest action points I have taken from this project is that the data stored in our request track is not clean enough and a large exercise will need to under taken to correct this.

Potential future steps

In December 2019 automation of the triage process is being implanted which will vastly reduce the amount of time cases are worked. Once the old backlog has been cleared and the team are working on fresh requests redoing this analysis will product more accurate results due to the limitations of the model.

Moving away from Excel as data workflow tool will help the team get through more cases as current workbooks are limited to 1 million rows of data. Once the team is trained to use a system dedicated to data analytics (e.g. Alteryx) this model could be rerun to give more concreate results.

Finally a large data quality exercise on past requests will need to be done if we can to make the model perform better. It can only learn on what inputs it is feed. If the humans creating these inputs are inaccurate it will mean the model will not work effectively.