Time Series Analysis Project - Offboarding Requests

Problem Statement

The offboarding team receives requests to deactivate client data all year round. The team gets especially busy over the holiday periods as the team can process more requests as traditionally the market is quiet at this time a year. Regulatory programs tend to have their deadlines at the end of the calendar year which further increases the demand on the team.

The goal for this project is to determine what are the busiest times of the year for the offboarding team and to see if I can use this information to accurately predict how many requests offboarding could get in the future. I am going to attempt to predict the demand on offboard team for the next three months.

Overview

An Offboarding request is submitted by the business or another one of our stakeholders when they need a data attribute removed from the UBS client data master systems; Entity Master or MasterFiles. These requests can come in at different levels within the client data hierarchy. They can be:

- Xref - Identifiers which enable a client to trade particular products on an account in MasterFiles.
- GL - Sometimes known as a cconsol. This is a client account which is stored in MasterFiles.
- RXM - A credit instrument aggregator. This is used to the work out credit limits and other related risks.
- CE - A consolidated entity. These are stored in the UBS legal master system (Entity Master). These are the onboarded client legal entities. Larger clients have multiple legal entities based around the world onboarded with UBS.
- PL - These are the identifiers used by the sales team which can contain multiple legal entities within a relationship for clients.

The initial plan is to work with a simple data set containing all requests received into the offboarding pipeline, regardless of completion state. I'm going to attempt to show the seasonality of the requests. Then I will try and go into more detail to see if there is a time of year where different requests inputs are submitted more often.

Time series analysis is suitable for this problem as the data is based on a series of singular inputs over an extended time frame. Experience tells me that offboarding tend to see spikes in requests and similar points of the year (June and December) and a time series analysis can help prove or disprove this.

Sourcing the data

- The data has bee sourced from the two Client Middle Office (CMO) Offboarding request trackers.
  - SharePoint (SP)
  - Client Lifecycle Management Tool (CLMT)
- No Client Identifying Data (CID) is in this dataset.

I have to make sure that I am in line with data sharing practices, therefore, I will:

- Make sure No CID is sent externally to UBS.
- All CID is removed or masked before submission

If I fail to complete the above I will be in breach of UBS' stringent data sharing guidelines.

Tasks

To complete this analysis successfully I will need to carry out the following tasks:

1. Load, explore and visualise the data.
    - This is required to make sure all the data has been imported properly and explore any areas of interest in the data which need to be explained or could affect the time series prediction when it's created.

1. Decomposition
    - The offboarding team has been getting an increasing number of requests whilst increasing throughput. Performing decomposition analysis on the data might provide greater insight.

1. Model and Transform
    - I will need to test for stationarity so check if I need to apply any logarithmic transformation to create my ARMIA model.

1. Prepare ARMIA model
    - To for future predictions to be made.

1. Forecast
    - I am going to attempt to predict the next three months of offboarding requests using the ARMIA model

1. Refine the Model
    - If my model does not perform well using the initial data I will try and refine the inputs to try and improve the results.

1. Conclusion
    - How has my model performed? How can these results be used in my current role? What future steps can I potentially do continue to improve this analysis?

Exploring the data

Importing the libaries

In [1]:

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

# importing matplotlib and seaborn for creating data visualisations
import matplotlib.pylab as plt
import matplotlib.pyplot as plot
import seaborn as sns

#the below code allows visualistions to be directly loaded into the Python notebook
%matplotlib inline

#importing datetime functionality for the time series analysis to be performed
from datetime import datetime

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

Loading the data

In [2]:

```
# load the offboarding request data from the two data points.
data = pd.read_csv("requests.csv",  encoding='ANSI')
```

- The offboarding team currently has two databases used to track offboarding requests.
- The team is migrating fully onto CLMT once the required functionality has been developed and is performing correctly after User Acceptance Testing (UAT) has been passed.
- This will allow for the full decommission and archiving of the SP site which will mean this code will have to be amended to work solely with CLMT.
- The dataset used is a simplified consolidation of both sources.

Exploring the data

- I am checking for data quality issues.

In [3]:

```
# showing the first few rows of data and the headings
data.head(13)
```

Out[3]:

	date	number of requests
0	01/01/2017	0
1	02/01/2017	0
2	03/01/2017	1
3	04/01/2017	0
4	05/01/2017	0
5	06/01/2017	2
6	07/01/2017	0
7	08/01/2017	0
8	09/01/2017	0
9	10/01/2017	0
10	11/01/2017	0
11	12/01/2017	0
12	13/01/2017	0

- The data looks to have been loaded correctly. Visualising this later will confirm this.
- There several days where no requests have been processed.
  - These will be weekends and weekdays when higher priority work took precedent.

In [4]:

```
data.shape
```

Out[4]:

```
(1055, 2)
```

- The shape is accurate as CMO Offboarding come into existence at the end of 2016 so I was expecting around one thousand days of data.

In [5]:

```
# showing the column headings to make sure they do not need amending
list(data.columns.values)
```

Out[5]:

```
['date', 'number of requests']
```

- I will rename the 'number of requests' column for ease.

In [6]:

```
# rename the number of requests column for ease
data.rename(columns = {'number of requests': 'requests'}, inplace=True)
```

In [7]:

```
# checking for null (NaN) data point
data.isnull().values.any()
```

Out[7]:

```
False
```

- There are no missing values in this data. I do not need to treat this.
- Based on my experience in the CMO offboarding team I am aware I lead a small team who consistently has a high level of demand.

In [8]:

```
# converting the 'Month' column in to the time series type
data['date'] = pd.to_datetime(data['date'], format = '%d/%m/%Y')
```

In [9]:

```
# Checking the conversation has worked correctly
data.dtypes
```

Out[9]:

```
date        datetime64[ns]
requests             int64
dtype: object
```

- The conversion of the 'date' column into a time-series has been successful.

In [10]:

```
#setting the 'Month' column as the index
data.set_index('date', inplace = True)
```

- Setting the 'date' column to the index is required to make sure the time series can be run correctly and be visualised accurately.

In [11]:

```
# checking the amendments have been successful by showing more than one month of data
data.head(32)
```

Out[11]:

date	requests
2017-01-01	0
2017-01-02	0
2017-01-03	1
2017-01-04	0
2017-01-05	0
2017-01-06	2
2017-01-07	0
2017-01-08	0
2017-01-09	0
2017-01-10	0
2017-01-11	0
2017-01-12	0
2017-01-13	0
2017-01-14	0
2017-01-15	0
2017-01-16	0
2017-01-17	0
2017-01-18	0
2017-01-19	0
2017-01-20	1
2017-01-21	550
2017-01-22	0
2017-01-23	0
2017-01-24	0
2017-01-25	0
2017-01-26	0
2017-01-27	0
2017-01-28	0
2017-01-29	0
2017-01-30	0
2017-01-31	0
2017-02-01	0

- You can see the first bulk population was received on the 21st of January 2017.
- As the team was in its infancy at this point more regular BAU (business as usual) requests would start to be received as more people were made aware of its creation, and the change of the process required to deactivate account data.

In [12]:

```
# Creating a new variable for the requests numbers only
ts = data['requests']
```

In [13]:

```
# Checking the type of variable
type(ts)
```

Out[13]:

```
pandas.core.series.Series
```

In [14]:

```
# checking the new variable creation has been a success.
ts.head(32)
```

Out[14]:

```
date
2017-01-01      0
2017-01-02      0
2017-01-03      1
2017-01-04      0
2017-01-05      0
2017-01-06      2
2017-01-07      0
2017-01-08      0
2017-01-09      0
2017-01-10      0
2017-01-11      0
2017-01-12      0
2017-01-13      0
2017-01-14      0
2017-01-15      0
2017-01-16      0
2017-01-17      0
2017-01-18      0
2017-01-19      0
2017-01-20      1
2017-01-21    550
2017-01-22      0
2017-01-23      0
2017-01-24      0
2017-01-25      0
2017-01-26      0
2017-01-27      0
2017-01-28      0
2017-01-29      0
2017-01-30      0
2017-01-31      0
2017-02-01      0
Name: requests, dtype: int64
```

In [15]:

```
# basic statistical analysis on the new variable
ts.describe()
```

Out[15]:

```
count    1055.000000
mean      114.790521
std       572.336059
min         0.000000
25%         0.000000
50%        21.000000
75%        68.500000
max      8961.000000
Name: requests, dtype: float64
```

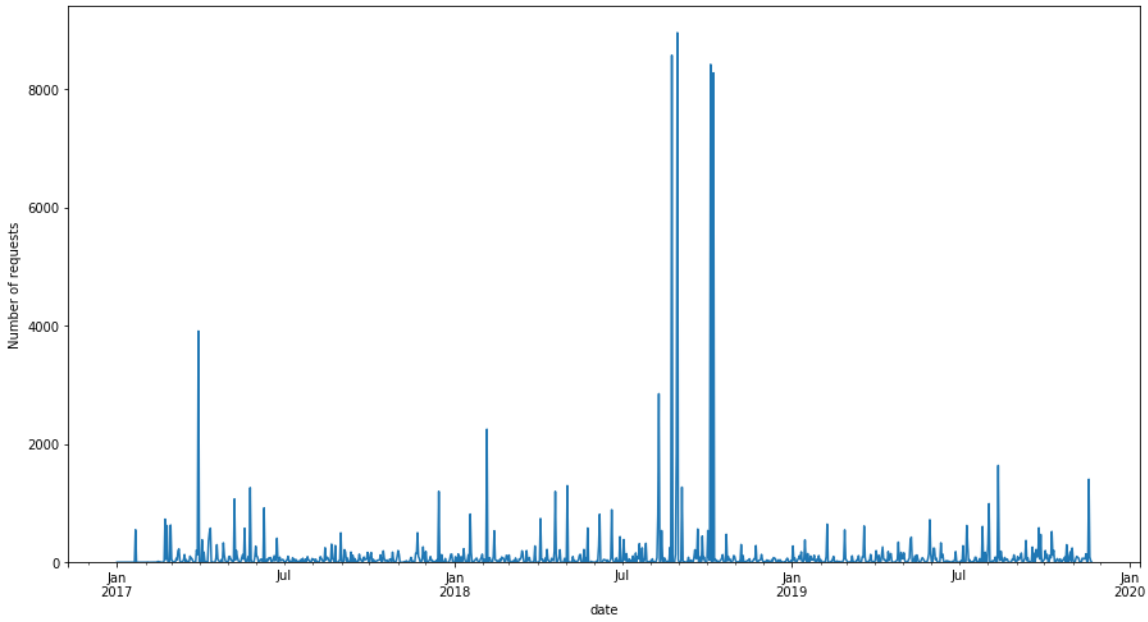
- Analysing the description table I can see that the Max amount is 8,961.
  - This is a lot higher than the 75% mark (69).
- I'm going to plot this out to identify this population.
- This population will have an effect on the mean, which could alter my results.
  - My domain knowledge tells me the mean for requests processed in a day is too high. The bulk populations are causing this distortion. Offboarding tends to get small bulk populations throughout the year however there are expectations where these populations are very large.

In [16]:

```
# plotting the the variable
plt.figure(figsize = (15,8))
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
ts.plot()
plt.ylim(0)
```

Out[16]:

```
(0, 9409.05)
```



- The outliers in this population are distorting the graph.
- The initial spike around April 2017 is the first regulatory population the CMO Offboard team received to complete by the end of 2017.
- There two large spikes around May and August in 2018.
- These spikes will affect the analysis as they appear to be distorting the mean. It might affect the model later on so will need to be mindful of these when the analysis is done.

Based on my experience in my current role on the CMO Offboarding team I can identify these two large populations as being relating to regulatory program relating to Brexit.

At this time in 2018 offboarding received two very large requests from the Brexit program. Brexit is a unique event which forced changes to UBS' legal structure which led to these large populations being submitted for offboarding. I do not expect numbers like this to be accepted in the future as there is currently no regulatory programmes which are expected to cause such a dramatic impact to UBS and as a result as such large offboarding request numbers.

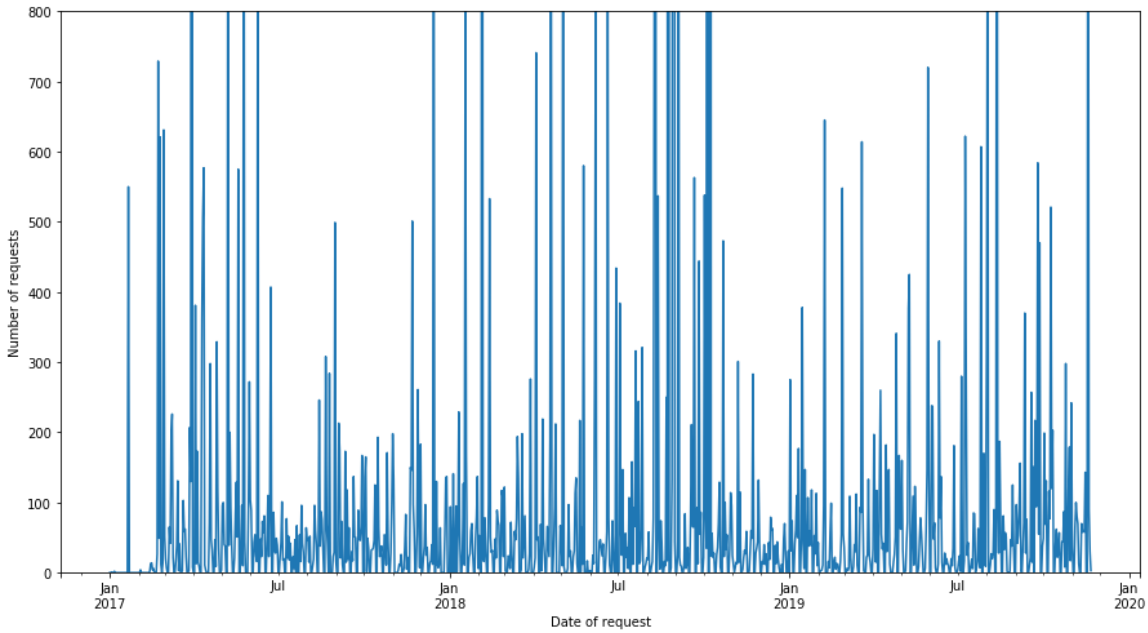
- I am going to limit the y-axis so I can view the regular inputs easier.

In [17]:

```
# plotting the the variable
plt.figure(figsize = (15,8))
ts.plot()
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.ylim(0,800)
```

Out[17]:

```
(0, 800)
```



- I have reduced the limit of the y-axis see the standard populations of requests being processed.
- Smaller sized bulk populations are more common than I originally thought.
    - This is going to have an impact on the model's prediction.
- I can use the above visualisation to make management aware of just how common large populations are requested. There will be an impact on the team's ability to meet SLA targets as the clock will start for all these requests as soon as they are entered into the pipeline. Not being able to stagger these requests hinders the team.

In [18]:

```
# calling the variable to identify the largest populations
data.sort_values(by = ['requests'], ascending = False)
```

Out[18]:

date	requests
2018-08-30	8961
2018-08-24	8579
2018-10-05	8423
2018-10-08	8279
2017-03-30	3912
2018-08-10	2852
2018-02-05	2252
2018-08-29	1848
2019-08-12	1638
2019-11-18	1405
2018-05-03	1297
2018-09-04	1271
2017-05-25	1262
2017-12-15	1199
2018-04-20	1198
2017-05-08	1071
2019-08-02	992
2017-06-09	919
2018-06-20	889
2018-01-18	815
2018-06-07	813
2018-08-09	800
2018-04-04	741
2017-02-22	729
2019-05-30	720
2019-02-08	645
2017-02-28	631
2019-07-09	622
2017-02-24	621
2019-03-20	614
...	...
2018-04-21	0
2018-04-22	0
2018-04-26	0
2019-03-03	0
2019-03-02	0

date	requests
2018-04-27	0
2018-04-28	0
2018-03-26	0
2018-03-18	0
2018-01-28	0
2018-02-24	0
2018-02-03	0
2018-02-04	0
2018-02-10	0
2018-02-11	0
2019-04-07	0
2019-04-06	0
2018-02-17	0
2018-02-18	0
2018-02-25	0
2019-03-23	0
2019-03-31	0
2019-03-30	0
2018-03-03	0
2018-03-04	0
2018-03-10	0
2018-03-11	0
2018-03-17	0
2019-03-24	0
2017-01-01	0

1055 rows × 1 columns

- August and May 2018 are clear outliers in this situation. I would also class October as an outlier too.
- As I stated above, these requests are related to Brexit.

The initial deadline for the UK to leave the EU was 29th March 2019. Due to this UBS decided to close down their UK entity, UBS Limited. The deadline for all the accounts to be closed was the end of December 2018. This meant all accounts not being closed would be migrated to the new Germany based entity "UBS Europe SE".

Decomposition

- I am going to visualise my data as smaller populations.
  - The time-series analysis might be affected as the above visualisations have not shown any clear patterns.
- I would argue that my data is multiplicative as I am aware that demand for the offboarding team from 2017 until now has been increasing year on year. The team have been completing more offboarding requests whilst simultaneously receiving more requests.
- I am going to visualise the decomposition breakdown of the data to try and provide more insight and back up my claim.

In [19]:

```
# importing seasonal decompose from statesmodels
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [20]:

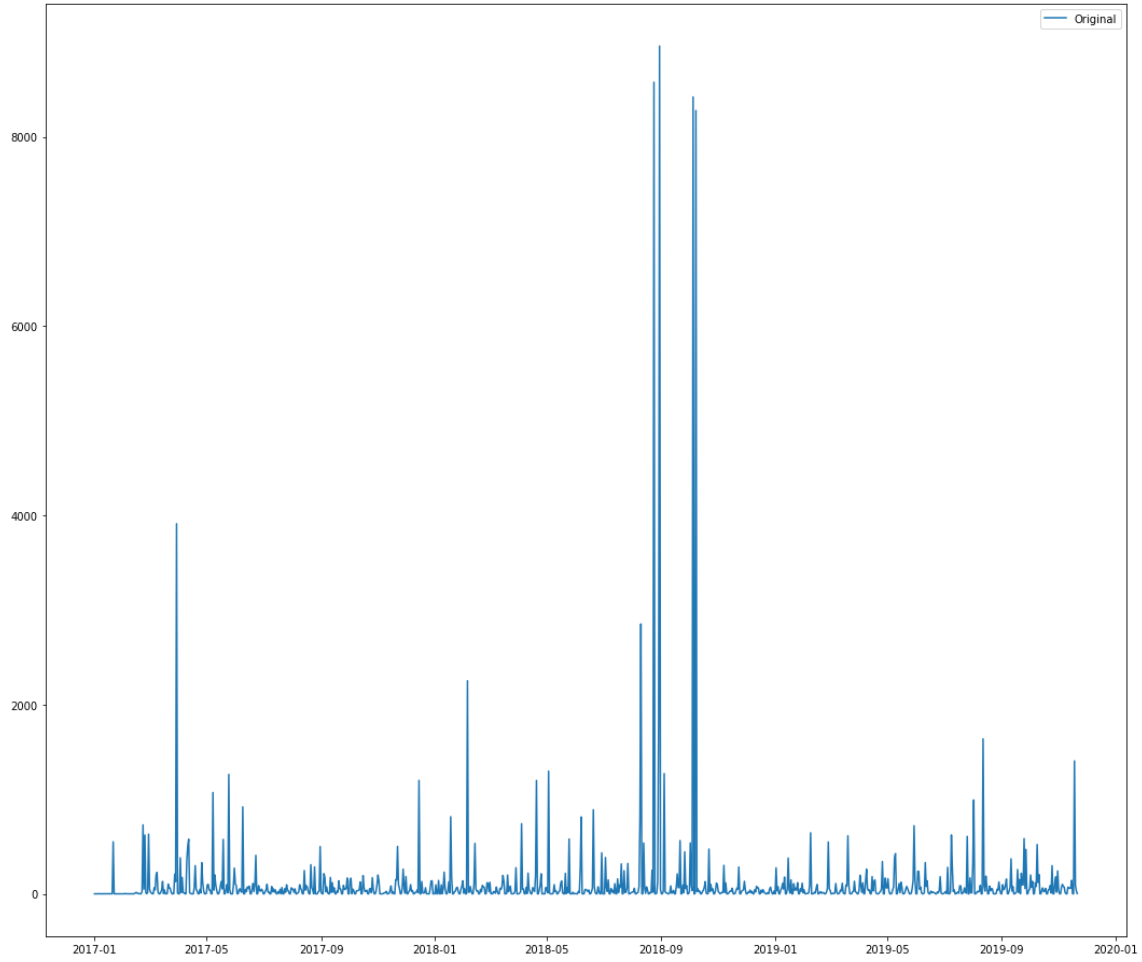
```
# creating new variable for decomosition
decomposition = seasonal_decompose(ts, freq = 7)
```

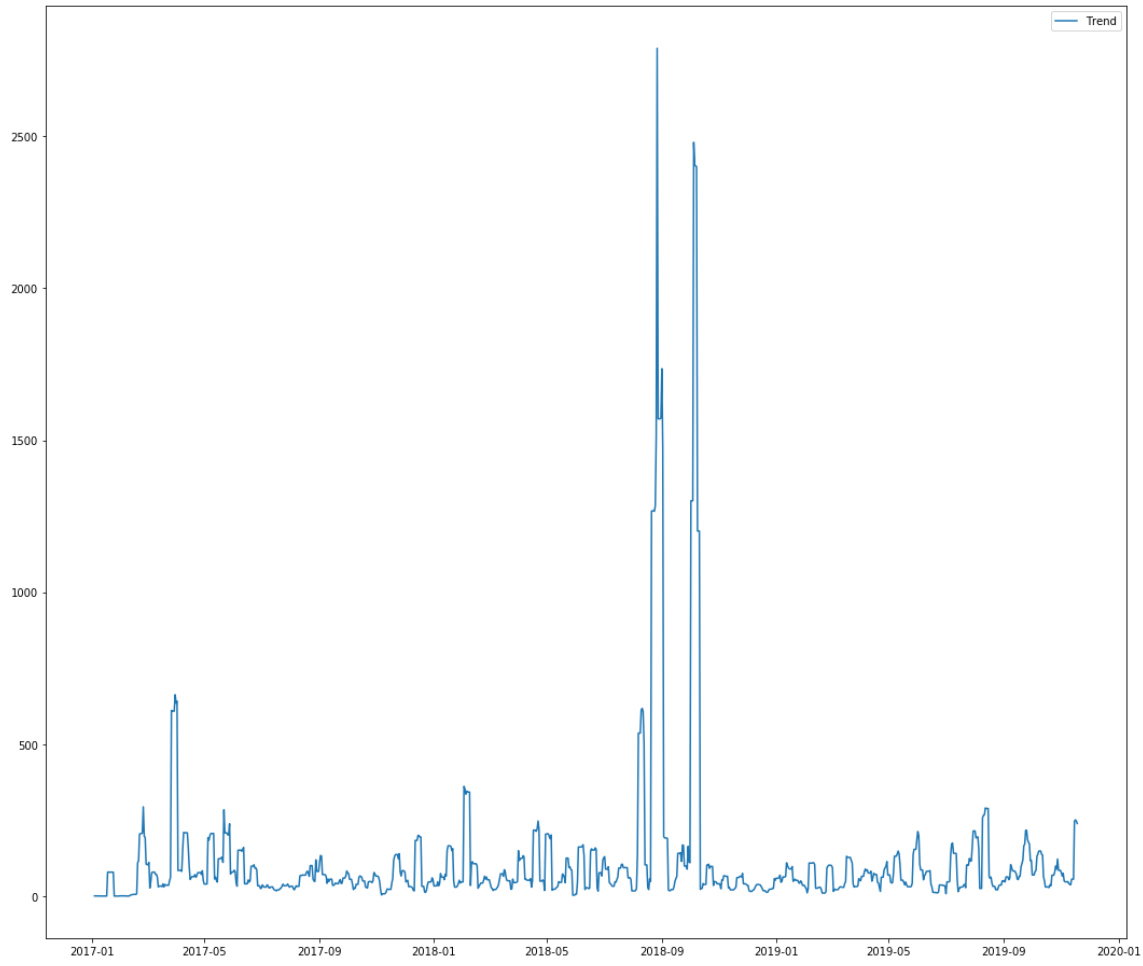
In [21]:

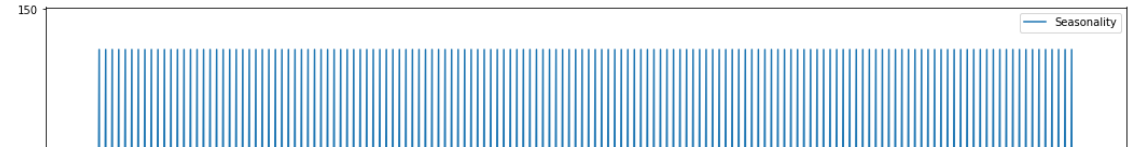
```
# creating new variables for the three types of decomposition
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

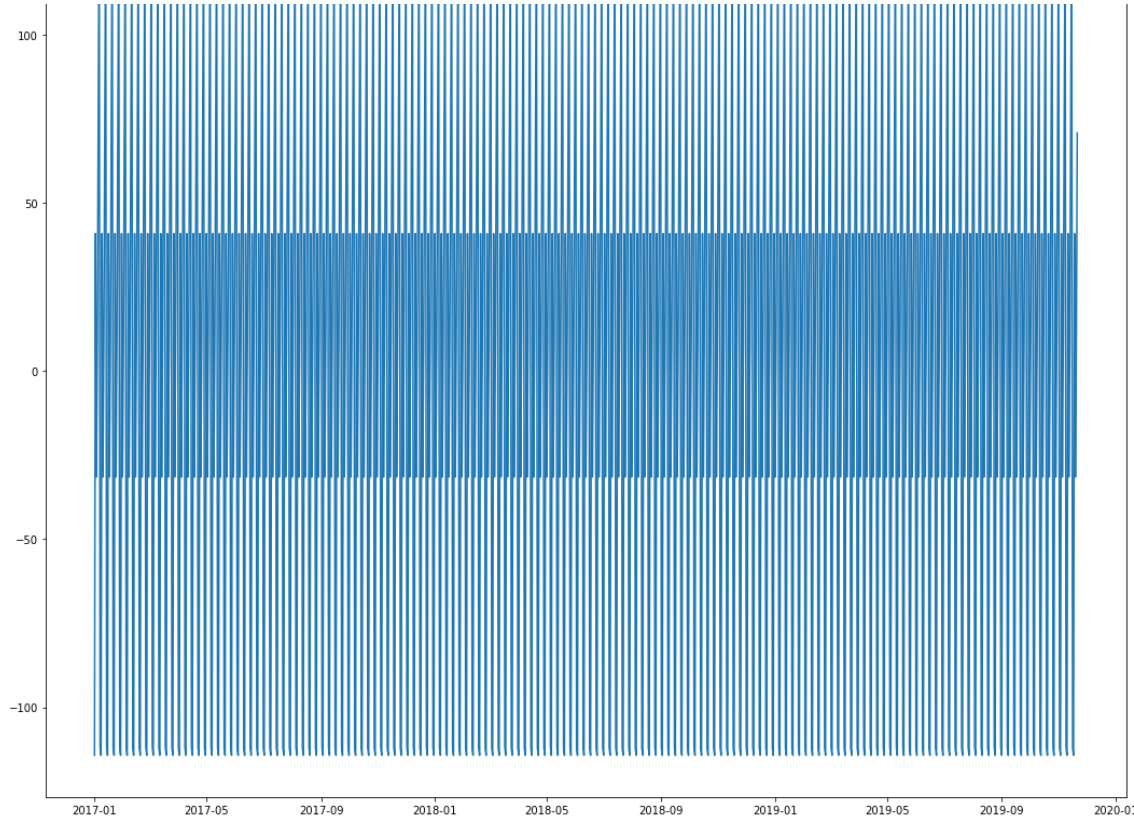
In [22]:

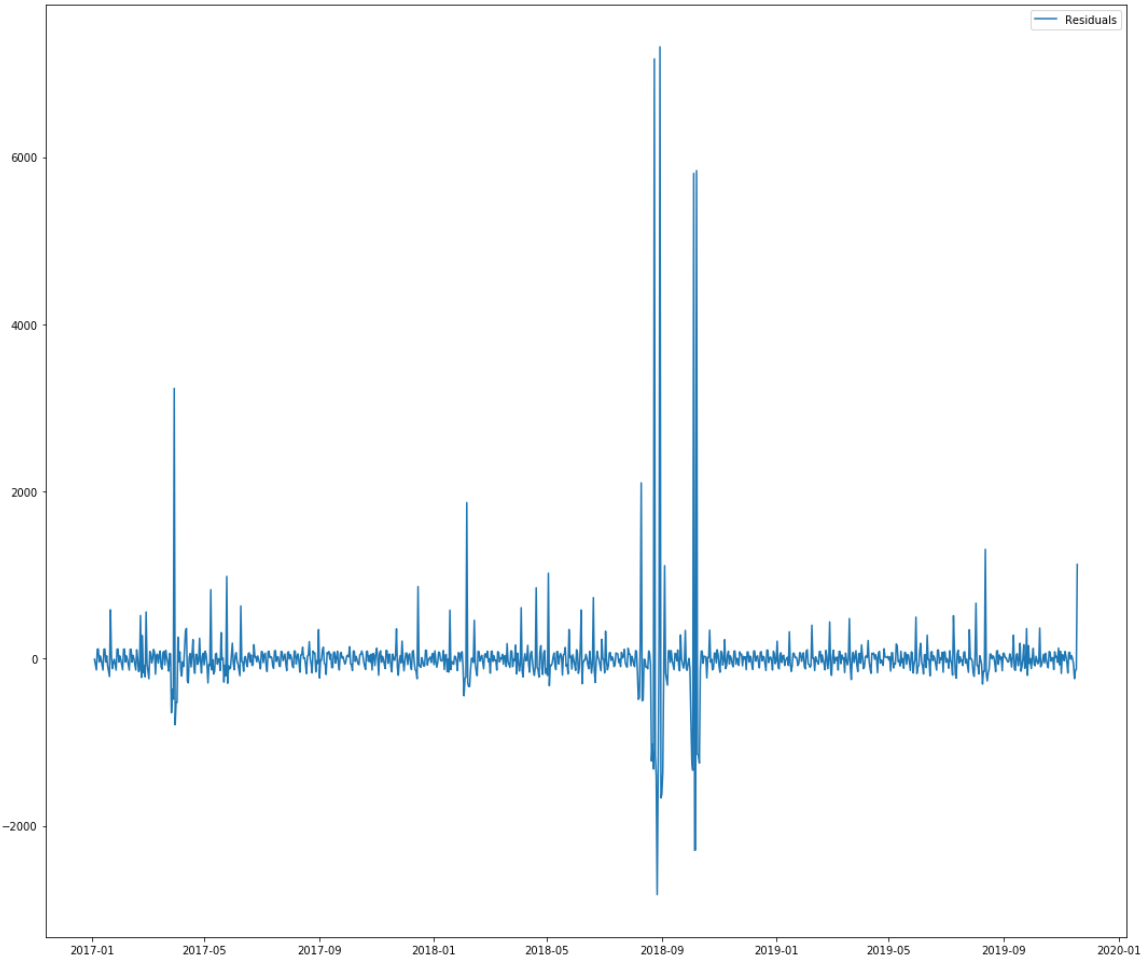
```
plt.figure(figsize=(15,50))
plt.subplot(411)
plt.plot(ts, label = 'Original')
plt.legend(loc = 'best')
plt.subplot(412)
plt.plot(trend, label = 'Trend')
plt.legend(loc = 'best')
plt.subplot(413)
plt.plot(seasonal, label = 'Seasonality')
plt.legend(loc = 'best')
plt.subplot(414)
plt.plot(residual, label = 'Residuals')
plt.legend(loc = 'best')
plt.tight_layout()
```


Original
8000
6000
4000
2000
0
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01


Trend
2500
2000
1500
1000
500
0
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01


Seasonality
150


100
50
0
−50
−100
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01


Residuals
6000
4000
2000
0
−2000
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01

- I am going to continue my analysis with this original data set and look at the results.
  - This will then inform me of any changes I will need to make to try and negate the potential issues that might be caused by this.
- There is no clear trend upwards however this has been affected by the large Brexit populations.
  - The visualisation shows potential peaks and troughs within the trend line through the years.
- The seasonality breakdown looks consistent as there are weekends included in this data.
  - No requests are processed on weekends.
- The residual plot confirms the Brexit related outliers might going to affect the model.
  - Once my initial model is completed I might drop these values in order and retest to see what effect these have.
- I am concerned that the residual analysis has confirmed some negative values.
  - This will need further investigation.

Model and transform

Testing for stationarity

In [23]:

```
# import the KPSS test form statesmodel
from statsmodels.tsa.stattools import kpss
import warnings
warnings.filterwarnings('ignore')
```

In [24]:

```
# run KPSS for stationarity
kpss(ts)
```

Out[24]:

```
(0.16778208321988378,
 0.1,
 22,
 {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

- The P-value has come back at 0.1. As this greater than 0.05, I can conclude that the data is stationary.
- I do not need to apply any logarithmic transformation to this data set so can move on to create my ARMIA model.

ARIMA model

- The ARIMA model is based on the concept that you can predict the next value in a time series by using information about the most recent data point which fits. Based on this I am going to try to predict future requests.

In [25]:

```
# import mean squared error to work out model accuracy
from sklearn.metrics import mean_squared_error
```

In [26]:

```
# import ARMIA from statsmodels to create the predicition model
from statsmodels.tsa.arima_model import ARIMA
```

In [27]:

```
# This finds the Mean Squared Error (MSE) of a single ARIMA model.
def evaluate_arima_model(data, arima_order):
    split = int(len(data) * 0.8) # an integer is required because it is later used as a
n index.
    train, test = data[0:split], data[split:len(data)]
    past = [x for x in train]
    # make predictions
    predictions = list()
    for i in range(len(test)):#timestep - wise comparison between test data and one-ste
p prediction ARIMA model.
        model = ARIMA(past, order = arima_order)
        model_fit = model.fit(disp = 0)
        future = model_fit.forecast()[0]
        predictions.append(future)
        past.append(test[i])

    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error

    # evaluate the ARIMA models with several different p, d, and q values.
def evaluate_models(dataset, p_values, d_values, q_values):
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    return print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
```

- There are two ways I can work out the best values for the ARIMA model:
  - Autocorrelation and partial autocorrelation.
  - Evaluation of the model using code.

Autocorrelation and Partial Autocorrelation

In [28]:

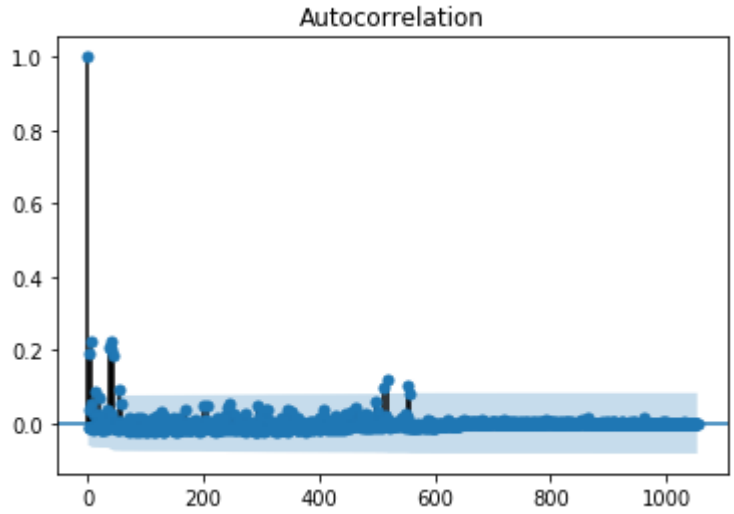
```
from statsmodels.graphics import tsaplots
```

In [29]:

```
tsaplots.plot_acf(data['requests'])
plt.figure(figsize = (15,8))
```

Out[29]:

```
<Figure size 1080x576 with 0 Axes>
```




```
<Figure size 1080x576 with 0 Axes>
```

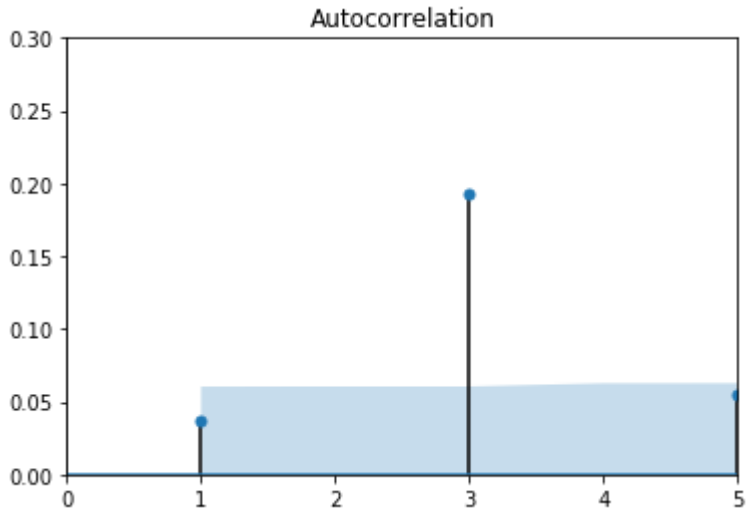
The axis values in this plot make the data harder to read. You can see some outliers between 400-600 on the x-axis. I am going to limit the axis values to zoom in on the first values.

In [30]:

```
tsaplots.plot_acf(data['requests'])
plt.xlim(0,5)
plt.ylim(0,0.3)
plt.figure(figsize = (15,8))
```

Out[30]:

```
<Figure size 1080x576 with 0 Axes>
```




```
<Figure size 1080x576 with 0 Axes>
```

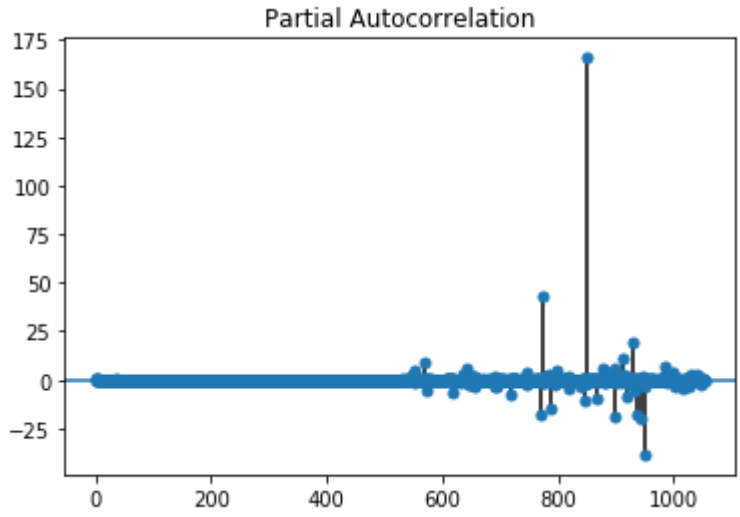
There is a significant correlation in leg 1. I will use the 'evaluate_models' function to confirm this.

In [31]:

```
tsaplots.plot_pacf(data['requests'])
#plt.xlim(0,5)
#plt.ylim(0,0.3)
plt.figure(figsize = (15,8))
```

Out[31]:

```
<Figure size 1080x576 with 0 Axes>
```




```
<Figure size 1080x576 with 0 Axes>
```

- This partial autocorrelation plot does not appear to be suitable for this model based on the plot above. I will finalise my inputs using the evaluate models function.

In [32]:

```
# I am choosing a couple of values to try for each parameter.
p_values = [x for x in range(0, 2)]
d_values = [x for x in range(0, 2)]
q_values = [x for x in range(0, 2)]
```

In [33]:

```
# this code evaluates the best ARIMA values for my model
evaluate_models(ts, p_values, d_values, q_values)
```

```
ARIMA(0, 0, 1) MSE=38746.321
ARIMA(0, 1, 1) MSE=39070.100
ARIMA(1, 0, 0) MSE=38750.158
ARIMA(1, 0, 1) MSE=38709.099
ARIMA(1, 1, 0) MSE=59337.224
ARIMA(1, 1, 1) MSE=39064.467
Best ARIMA(1, 0, 1) MSE=38709.099
```

In [34]:

```
# define my parameters with the best ARIMA values (with the lowest error rate)
p = 1
d = 0
q = 1
model = ARIMA(ts, order=(p,d,q))
model_fit = model.fit()
forecast = model_fit.forecast(90)
```

In [35]:

```
model_fit.summary()
```

Out[35]:

ARMA Model Results

Dep. Variable:	requests	**No. Observations:**	1055
Model:	ARMA(1, 1)	**Log Likelihood**	-8190.809
Method:	css-mle	**S.D. of innovations**	569.554
Date:	Sat, 14 Dec 2019	**AIC**	16389.618
Time:	14:06:44	**BIC**	16409.463
Sample:	01-01-2017	**HQIC**	16397.141
	- 11-21-2019		

	coef	std err	z	P>|z|	[0.025	0.975]
const	114.7933	18.347	6.257	0.000	78.833	150.754
ar.L1.requests	-0.6080	0.104	-5.844	0.000	-0.812	-0.404
ma.L1.requests	0.6826	0.094	7.271	0.000	0.499	0.867

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-1.6446	+0.0000j	1.6446	0.5000
MA.1	-1.4650	+0.0000j	1.4650	0.5000

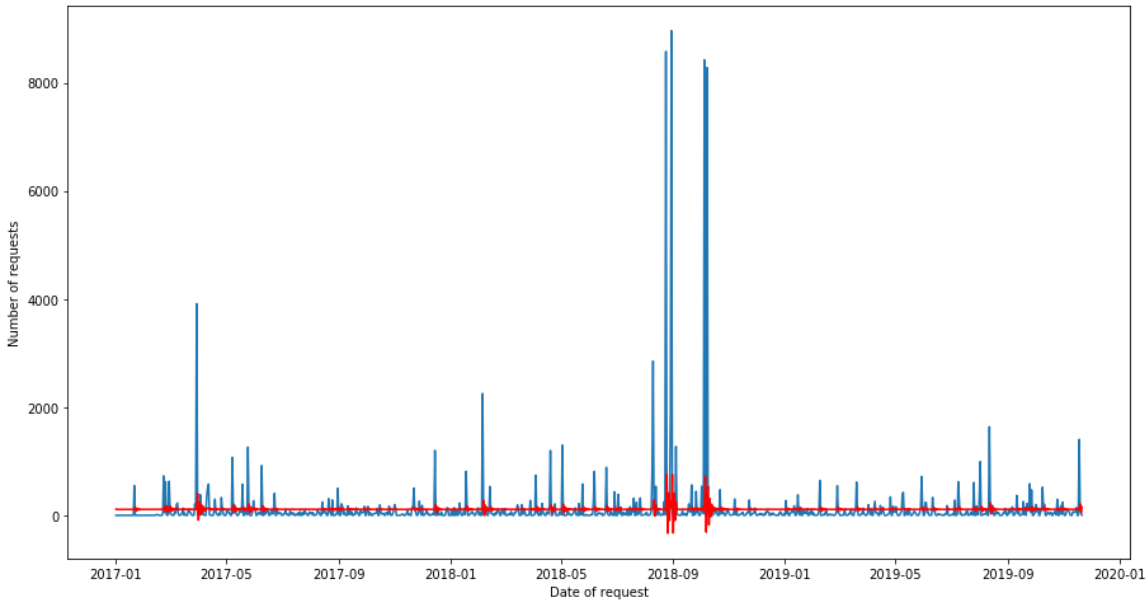
- I now need to plot my model's predictions on a graph to see how successful it is.

In [36]:

```
# plot the model against the actual numbers
plt.figure(figsize = (15,8))
plt.plot(ts)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.plot(model_fit.predict(), color = 'red')
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0xc31be48>]
```



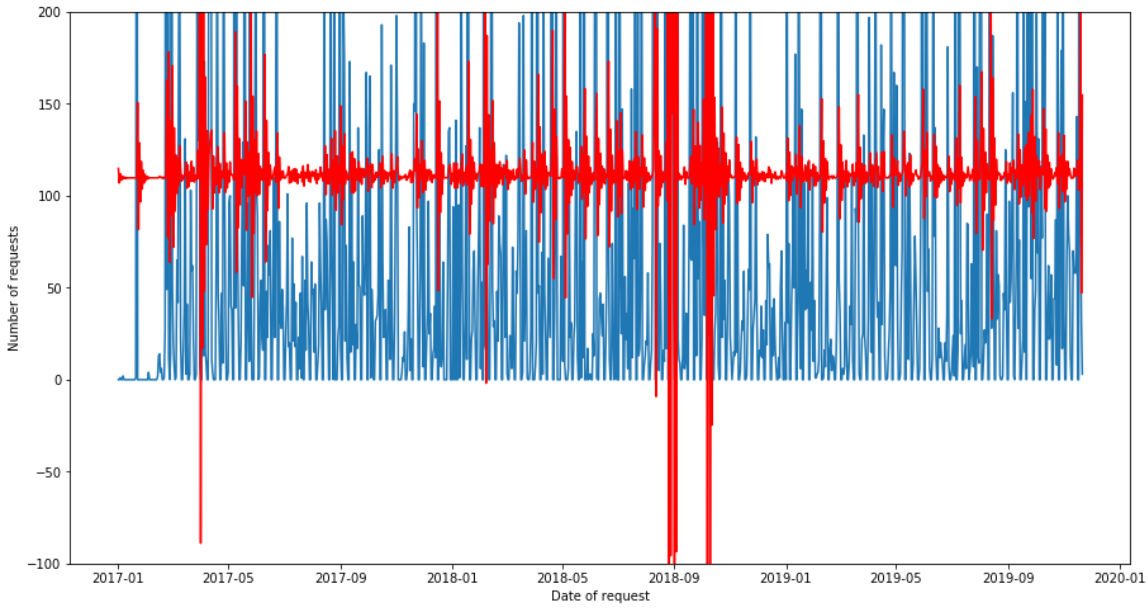
- I need to zoom in on this plot to analyse this further. I can see there are smaller spikes where bulk populations have been submitted.

In [37]:

```
# plot the model against the actual numbers
plt.figure(figsize = (15,8))
plt.plot(ts)
plt.plot(model_fit.predict(), color = 'red')
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.ylim(-100,200)
```

Out[37]:

```
(-100, 200)
```



- The negative predictions are happening due to the large outliers in this population.
- Removing these from the data might help with the model's accuracy however I am going to carry on with this model to see what effect this has.
- Once the SP tracker has decommissioned and these massive bulk populations will not be included in the analysis which should improve the model's future runs.

Forecasting

- I am going to use my model to try to see if we can predict the next three months of offboarding requests.

In [38]:

```
# going to predict the next three months of data to mid 2021
periods_to_forecast = 90
```

In [39]:

```
# Create a range of future dates that is the length of
# the periods you've chosen to forecast
date_range = pd.date_range(ts.index[-1],
                           periods = periods_to_forecast,
              freq='D').strftime("%Y/%m/%d").tolist()

# Turn that range into a dataframe that includes your predictions
future_days = pd.DataFrame(date_range, columns = ['date'])
future_days['date'] = pd.to_datetime(future_days['date'])
future_days.set_index('date', inplace = True)
```

In [40]:

```
future_days['prediction'] = forecast[0]
```

In [41]:

```
future_days
```

Out[41]:

date	prediction
2019-11-21	79.193111
2019-11-22	136.439564
2019-11-23	101.631603
2019-11-24	122.796130
2019-11-25	109.927315
2019-11-26	117.752031
2019-11-27	112.994314
2019-11-28	115.887182
2019-11-29	114.128211
2019-11-30	115.197730
2019-12-01	114.547423
2019-12-02	114.942834
2019-12-03	114.702410
2019-12-04	114.848597
2019-12-05	114.759710
2019-12-06	114.813756
2019-12-07	114.780894
2019-12-08	114.800875
2019-12-09	114.788726
2019-12-10	114.796113
2019-12-11	114.791621
2019-12-12	114.794353
2019-12-13	114.792692
2019-12-14	114.793702
2019-12-15	114.793088
2019-12-16	114.793461
2019-12-17	114.793234
2019-12-18	114.793372
2019-12-19	114.793288
2019-12-20	114.793339
...	...
2020-01-20	114.793320
2020-01-21	114.793320
2020-01-22	114.793320
2020-01-23	114.793320
2020-01-24	114.793320

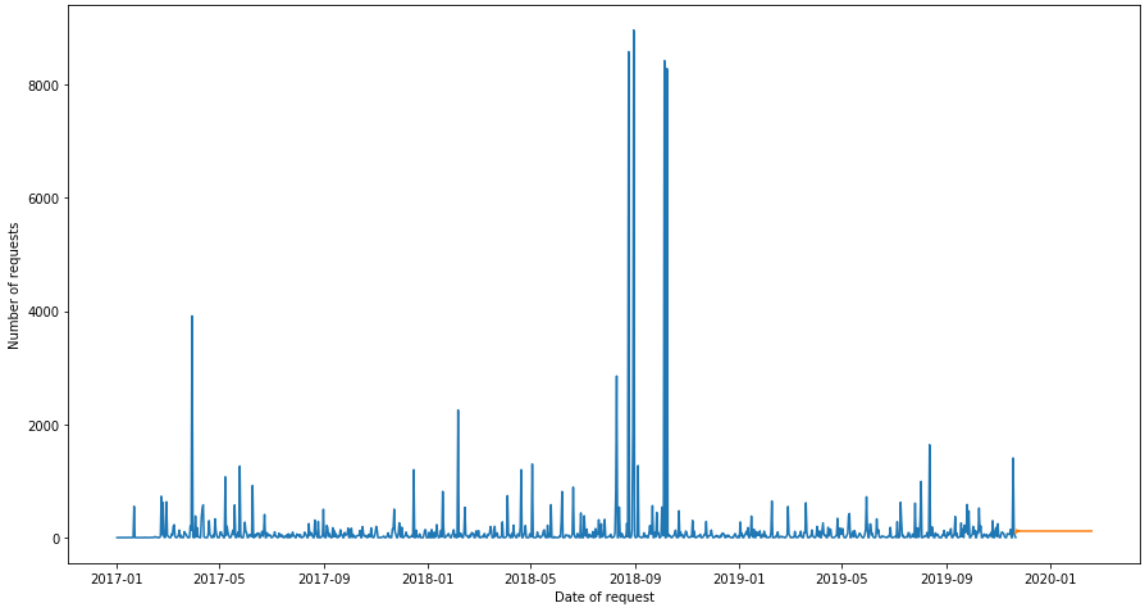
date	prediction
2020-01-25	114.793320
2020-01-26	114.793320
2020-01-27	114.793320
2020-01-28	114.793320
2020-01-29	114.793320
2020-01-30	114.793320
2020-01-31	114.793320
2020-02-01	114.793320
2020-02-02	114.793320
2020-02-03	114.793320
2020-02-04	114.793320
2020-02-05	114.793320
2020-02-06	114.793320
2020-02-07	114.793320
2020-02-08	114.793320
2020-02-09	114.793320
2020-02-10	114.793320
2020-02-11	114.793320
2020-02-12	114.793320
2020-02-13	114.793320
2020-02-14	114.793320
2020-02-15	114.793320
2020-02-16	114.793320
2020-02-17	114.793320
2020-02-18	114.793320

90 rows × 1 columns

- The predictions normalise after ten days.
  - The model appears to be going to the mean for all future dates.
- This is likely an underlying issue with the model or the data. I will test this later by running the model with new data.

In [42]:

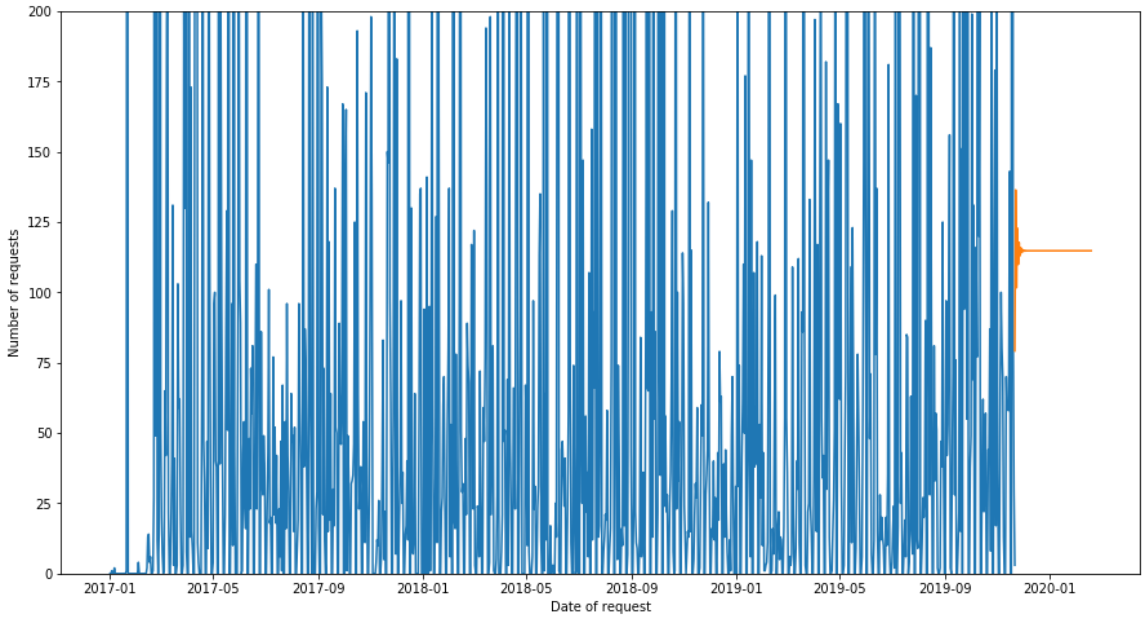
```
# add the predicted numbers to the current data
plt.figure(figsize = (15,8))
plt.plot(ts)
plt.plot(future_days)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.show()
```



- The predictions have been added to the current numbers however due to the size of the brexit populations I need to zoom in on them.

In [43]:

```
# add the predicted numbers to the current data
plt.figure(figsize=(15,8))
plt.plot(ts)
plt.plot(future_days)
plt.ylim(0,200)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.show()
```



- This model is not performing to an acceptable level. I will need to revisit it.

My first amendment is removing the bulk populations. This will help predict BAU populations as bulk populations are distorting the data. Once the model is rerun I can identify if it is a problem with data or the model.

Refining the model

I am going to reformulate the model using a dataset where I am going to remove the bulk requests.

In [44]:

```
# reimporting the data
data2 = pd.read_csv("requests.csv",  encoding='ANSI')
```

In [45]:

```
# changing the date to dateime
data2['date'] = pd.to_datetime(data2['date'], format = '%d/%m/%Y')
```

In [46]:

```
#resetting the index
data2.set_index('date', inplace = True)
```

In [47]:

```
#rename the reqeust column
data2.rename(columns = {'number of requests': 'requests'}, inplace=True)
```

In [48]:

```
# removing all the values over 100 so the large bulk request are cancelledout
data2['requests'].values[data2['requests'] > 100] = 0
```

In [49]:

```
# checking the values have been removed
data2.sort_values(by = ['requests'], ascending = False)
```

Out[49]:

date	requests
2018-10-24	100
2019-11-05	100
2018-02-12	100
2017-05-03	100
2019-02-15	99
2017-10-31	99
2017-11-23	98
2018-05-09	97
2019-09-02	97
2017-12-06	97
2018-10-31	96
2017-05-02	96
2019-02-07	96
2017-05-23	96
2017-08-09	96
2017-07-26	96
2017-06-22	96
2019-09-24	95
2018-01-08	95
2018-01-02	94
2019-09-23	94
2017-05-11	93
2019-03-18	93
2018-04-24	93
2018-09-24	93
2019-03-21	93
2018-07-18	93
2019-09-05	92
2019-04-03	91
2019-11-19	90
...	...
2017-11-05	0
2018-12-16	0
2018-12-15	0
2017-11-06	0
2017-11-11	0

date	requests
2017-11-12	0
2017-11-18	0
2017-11-19	0
2018-12-09	0
2018-12-08	0
2017-11-20	0
2017-11-21	0
2017-11-22	0
2017-11-25	0
2017-11-26	0
2018-12-02	0
2018-12-01	0
2017-11-28	0
2018-11-29	0
2017-12-01	0
2017-12-02	0
2017-12-03	0
2018-11-25	0
2018-11-24	0
2018-11-23	0
2017-12-09	0
2017-12-10	0
2017-12-15	0
2018-11-18	0
2017-01-01	0

1055 rows × 1 columns

- The bulk populations have been succesfully removed.

In [50]:

```
# new varible
ts2 = data2['requests']
```

In [51]:

```
# show informaiton
ts2.describe()
```

Out[51]:

```
count    1055.000000
mean       18.982938
std        26.879866
min         0.000000
25%         0.000000
50%         1.000000
75%        32.500000
max       100.000000
Name: requests, dtype: float64
```

- This has brought the mean of requests down substantially.
- I am redoing the decomposition to see what effect this change as had.

In [52]:

```
# creating new variable for decomosition
decomposition = seasonal_decompose(ts2, freq = 7)
```

In [53]:

```
# creating new variables for the three types of decomposition
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

In [54]:

```
plt.figure(figsize=(15,8))
plt.subplot(411)
plt.plot(ts2, label = 'Original')
plt.legend(loc = 'best')
plt.subplot(412)
plt.plot(trend, label = 'Trend')
plt.legend(loc = 'best')
plt.subplot(413)
plt.plot(seasonal, label = 'Seasonality')
plt.legend(loc = 'best')
plt.subplot(414)
plt.plot(residual, label = 'Residuals')
plt.legend(loc = 'best')
plt.tight_layout()
```

- There is no clear trend with the amended data.
- Seasonality has remained the same
- Residuals suggest that there are now more outliers

In [55]:

```
# run KPSS for stationarity
kpss(ts2)
```

Out[55]:

```
(0.2802829875182456,
 0.1,
 22,
 {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

- The new data is not stationary.

In [56]:

```
# This method finds the MSE of a single ARIMA model.
def evaluate_arima_model(data2, arima_order):
    split = int(len(data2) * 0.8) # Needs to be an integer because it is later used as
 an index.
    train, test = data2[0:split], data2[split:len(data2)]
    past = [x for x in train]
    # make predictions
    predictions = list()
    for i in range(len(test)):#timestep-wise comparison between test data and one-step
 prediction ARIMA model.
        model = ARIMA(past, order = arima_order)
        model_fit = model.fit(disp = 0)
        future = model_fit.forecast()[0]
        predictions.append(future)
        past.append(test[i])

    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error

    # This method evaluates ARIMA models with several different p, d, and q values.
def evaluate_models(dataset, p_values, d_values, q_values):
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    return print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
```

- I am going to use evaluate_models to define the parameters for my model.

In [57]:

```
# choosing a couple of values to try for each parameter.
p_values = [x for x in range(0, 2)]
d_values = [x for x in range(0, 2)]
q_values = [x for x in range(0, 2)]
```

In [58]:

```
evaluate_models(ts2, p_values, d_values, q_values)
```

```
ARIMA(0, 0, 1) MSE=808.940
ARIMA(0, 1, 1) MSE=832.923
ARIMA(1, 0, 0) MSE=813.068
ARIMA(1, 0, 1) MSE=807.618
ARIMA(1, 1, 0) MSE=1213.224
Best ARIMA(1, 0, 1) MSE=807.618
```

- This has provided the best values for my model.

In [59]:

```
# the above test shows that that below has the lowest error rate
p = 0
d = 0
q = 1
model = ARIMA(ts2, order = (p,d,q))
model_fit = model.fit()
forecast = model_fit.forecast(90)
```

In [60]:

```
model_fit.summary()
```

Out[60]:

ARMA Model Results

Dep. Variable:	requests	**No. Observations:**	1055
Model:	ARMA(0, 1)	**Log Likelihood**	-4956.835
Method:	css-mle	**S.D. of innovations**	26.562
Date:	Sat, 14 Dec 2019	**AIC**	9919.671
Time:	14:08:02	**BIC**	9934.555
Sample:	01-01-2017	**HQIC**	9925.313
	- 11-21-2019		

	coef	std err	z	P>|z|	[0.025	0.975]
const	18.9781	0.946	20.053	0.000	17.123	20.833
ma.L1.requests	0.1574	0.031	5.043	0.000	0.096	0.219

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-6.3523	+0.0000j	6.3523	0.5000

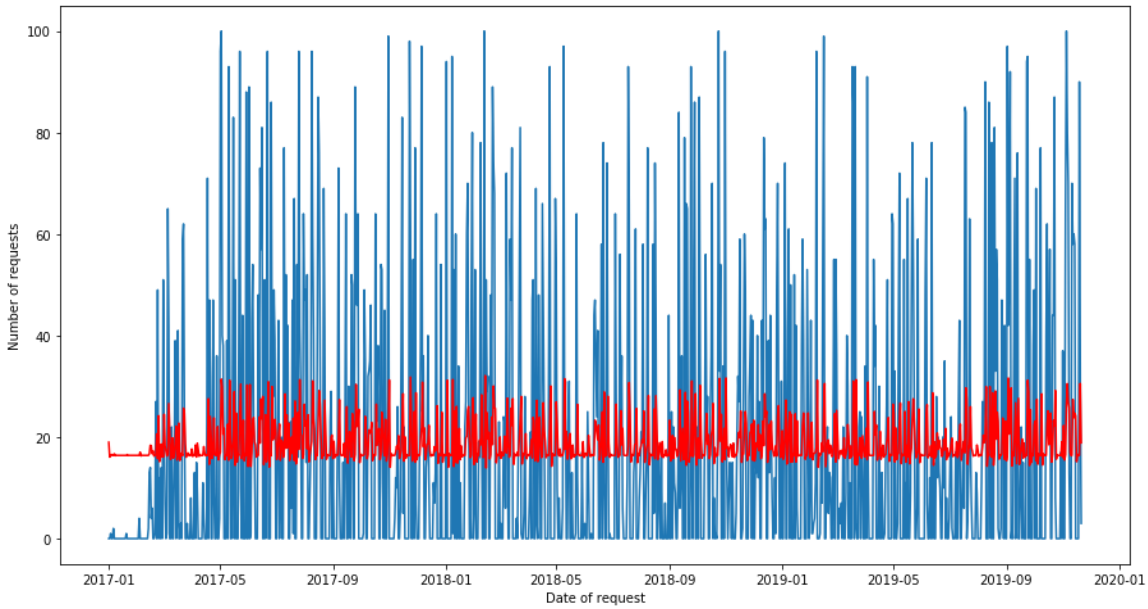
- Plotting the new prediction on a graph to see if the new data has improved the model's performance.

In [61]:

```
# plot the model against the actual numbers
plt.figure(figsize = (15,8))
plt.plot(ts2)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.plot(model_fit.predict(), color = 'red')
```

Out[61]:

```
[<matplotlib.lines.Line2D at 0xd6e1f28>]
```



- The predictions from the model look more promising however they still do not look accurate. I believe this might be due to the weekends being in this data.

In [62]:

```
# going to predict the next three months of data
periods_to_forecast = 90
```

In [63]:

```
# Create a range of future dates that is the length of
# the periods you've chosen to forecast
date_range = pd.date_range(ts2.index[-1],
                           periods = periods_to_forecast,
              freq='D').strftime("%Y/%m/%d").tolist()

# Turn that range into a dataframe that includes your predictions
future_days2 = pd.DataFrame(date_range, columns = ['date'])
future_days2['date'] = pd.to_datetime(future_days2['date'])
future_days2.set_index('date', inplace = True)
```

In [64]:

```
future_days2['prediction'] = forecast[0]
```

In [65]:

```
future_days2
```

Out[65]:

date	prediction
2019-11-21	16.476772
2019-11-22	18.978129
2019-11-23	18.978129
2019-11-24	18.978129
2019-11-25	18.978129
2019-11-26	18.978129
2019-11-27	18.978129
2019-11-28	18.978129
2019-11-29	18.978129
2019-11-30	18.978129
2019-12-01	18.978129
2019-12-02	18.978129
2019-12-03	18.978129
2019-12-04	18.978129
2019-12-05	18.978129
2019-12-06	18.978129
2019-12-07	18.978129
2019-12-08	18.978129
2019-12-09	18.978129
2019-12-10	18.978129
2019-12-11	18.978129
2019-12-12	18.978129
2019-12-13	18.978129
2019-12-14	18.978129
2019-12-15	18.978129
2019-12-16	18.978129
2019-12-17	18.978129
2019-12-18	18.978129
2019-12-19	18.978129
2019-12-20	18.978129
...	...
2020-01-20	18.978129
2020-01-21	18.978129
2020-01-22	18.978129
2020-01-23	18.978129
2020-01-24	18.978129

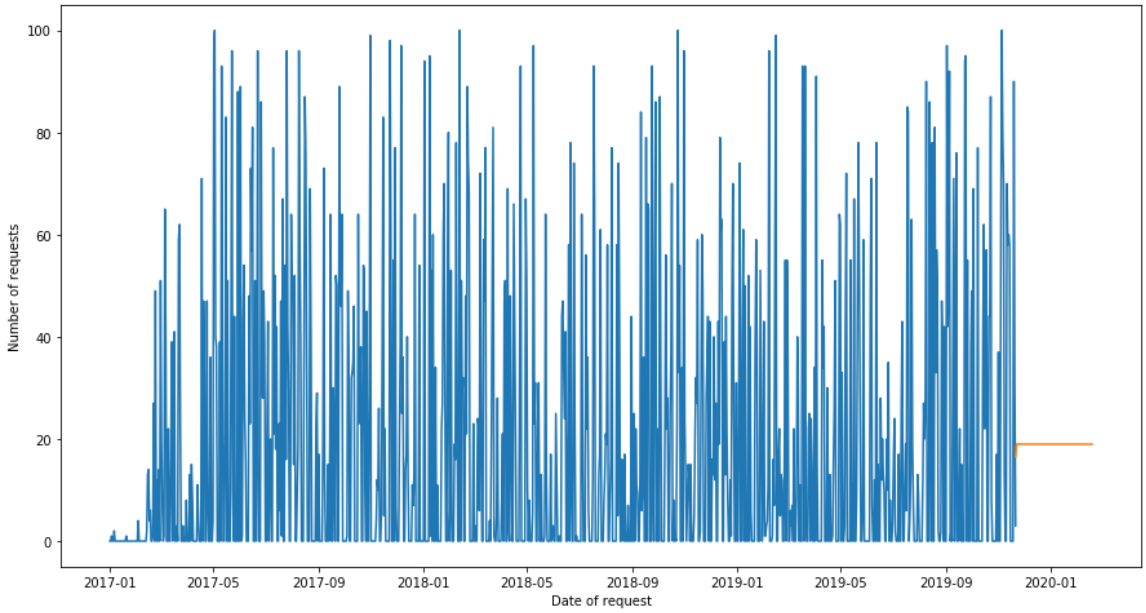
date	prediction
2020-01-25	18.978129
2020-01-26	18.978129
2020-01-27	18.978129
2020-01-28	18.978129
2020-01-29	18.978129
2020-01-30	18.978129
2020-01-31	18.978129
2020-02-01	18.978129
2020-02-02	18.978129
2020-02-03	18.978129
2020-02-04	18.978129
2020-02-05	18.978129
2020-02-06	18.978129
2020-02-07	18.978129
2020-02-08	18.978129
2020-02-09	18.978129
2020-02-10	18.978129
2020-02-11	18.978129
2020-02-12	18.978129
2020-02-13	18.978129
2020-02-14	18.978129
2020-02-15	18.978129
2020-02-16	18.978129
2020-02-17	18.978129
2020-02-18	18.978129

90 rows × 1 columns

- The predications look to normalise very quickly with the new data set. I will plot these out to further evidence this.

In [66]:

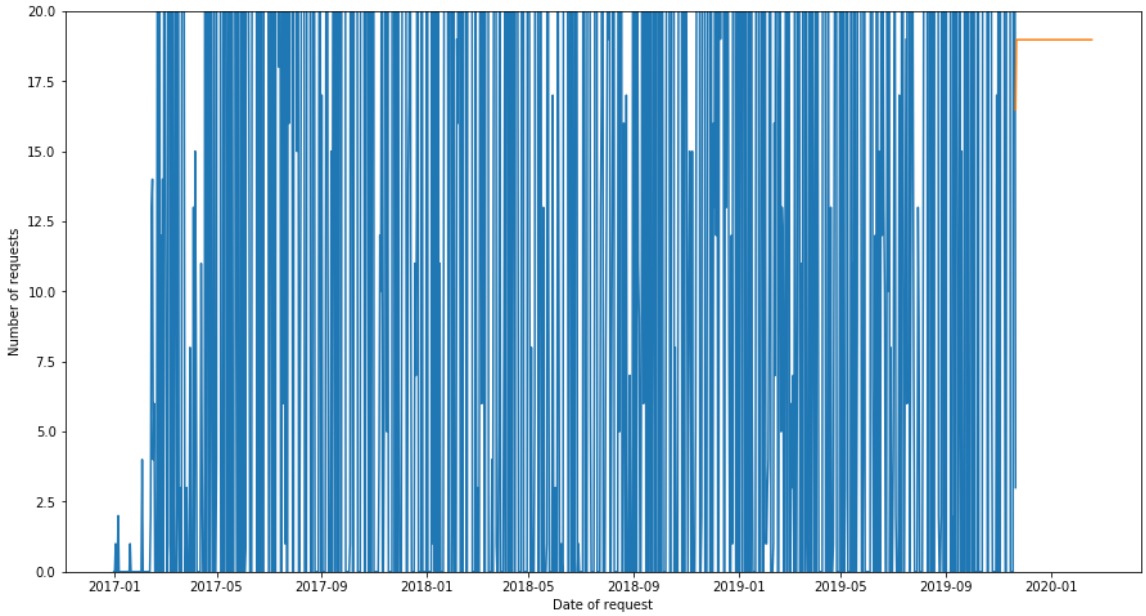
```
# add the predicted numbers to the current data
plt.figure(figsize=(15,8))
plt.plot(ts2)
plt.plot(future_days2)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.show()
```



- I can see the predictions on the above plot look very flat.

In [67]:

```
# add the predicted numbers to the current data
plt.figure(figsize=(15,8))
plt.plot(ts2)
plt.plot(future_days2)
plt.ylim(0,20)
plt.xlabel("Date of request")
plt.ylabel("Number of requests")
plt.show()
```



- When zooming in on the predictions I can see that the model has not been improved with the new data. The predictions revert to the mean very quickly.

Conclusion

- My model is not very effective at predicting future populations. It reverts to the mean for all future dates. Even when I have rerun the model with a reduced population the model is referring back to the mean very quickly.
- Offboarding has a population of 11,000 requests due in before the end of the year and the model clearly cannot predict this as there is no president in the past data. This limited data cannot provide this insight.
- In my current role, I can use this analysis to explain to senior management that processing requests on behalf of requestors make predicting input difficult as the team is limited by their capacity. As we move to a self-service offboarding model in CLMT we could potentially get better model analysis as offboarding requests will be drip-fed into the pipeline by stakeholders as and when they need them, rather than be uploaded in bulk populations by a member of the offboarding team. The fewer requests the team process the more capacity they will have to get through the backlog of work.

Potential Future Steps

- I believe the model not functioning well is due to data points for the weekends being included. No requests are processed on the weekend. I think removing these would reduce the amount of noise and therefore not distort the statistical information (e.g. mean). Removing these empty data points and changing the frequency to 5 days would improve the model.
- I would also like to revisit this model once the offboarding team has fully migrated request capture to CLMT and the ageing backlog has cleared. The team is still in a reactionary mode due to large requests and once the process is fully stabilised input and throughput should standardise. This is a clear future step as I cannot envisage this occurring for several years.