Night Tube Journeys on the London Underground

This notebook aims to gather, process and visualise data relating to the number of passengers which utilised the Night Tube Service from August 2016 - February 2017; the first 6 months for which service was available.

In [15]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

pd.set_option('display.max_rows', 50) #Allow more displayed data by default

%matplotlib inline
#Precautionary debugging - for graphs plotted

os.getcwd() #Prints the working directory to ensure correct relative file paths duri
```

Out[15]:
```
'C:\\Users\\Zaana\\Documents\\CASA\\CASA6\\CW_FA'
```

Data Importation

Data was collected from a TfL Freedom of Information (FOI) public database. The dataset gives values for the number of Night Tube passengers for every weekend, from the first weekend the Night Tube was in service (August 18th & 19th 2016), to the last week of February 2017.

As this data was not included in the regularly released Rolling Origin and Destination Survey (RODS), the file presented is in a 'raw' and unprocessed format. This is shown when the file is loaded below:

In [10]:
```
dataraw = pd.read_csv("FOI-1215.csv")

dataraw.tail(16)
```

Out[10]:

	Date	Station Name	Time	Number of Entries	Number of Exits
58446	18-02-17	Woodford	4:30:00	Fewer than 5	7
58447	18-02-17	Woodford	5:00:00	10	12
58448	18-02-17	Woodford	5:30:00	11	6
58449	18-02-17	Woodford	6:00:00	31	12
58450	18-02-17	Woodside Park	0:30:00	Fewer than 5	23
58451	18-02-17	Woodside Park	1:00:00	Fewer than 5	17
58452	18-02-17	Woodside Park	1:30:00	Fewer than 5	23
58453	18-02-17	Woodside Park	2:00:00	Fewer than 5	17
58454	18-02-17	Woodside Park	2:30:00	Fewer than 5	16
58455	18-02-17	Woodside Park	3:00:00	Fewer than 5	8

	Date	Station Name	Time	Number of Entries	Number of Exits
58456	18-02-17	Woodside Park	3:30:00	Fewer than 5	12
58457	18-02-17	Woodside Park	4:00:00	Fewer than 5	7
58458	18-02-17	Woodside Park	4:30:00	Fewer than 5	8
58459	18-02-17	Woodside Park	5:00:00	Fewer than 5	Fewer than 5
58460	18-02-17	Woodside Park	5:30:00	Fewer than 5	Fewer than 5
58461	18-02-17	Woodside Park	6:00:00	9	Fewer than 5

Based on the formatting used, there are approximately 59,000 values in the dataset.

From first observation, there is no clear value for stations which see "fewer than 5" passengers entering or exiting, meaning an assumption must be made for quantitative analysis. In addition to this, the data is expanded per day, per half hour *and* per station, rather than grouping the dates and times - this results in multiple values of the same station for each weekend, as shown by the last 10 values being solely for 1 night at Woodside Park Station.

Another obstacle found when observing the .csv file in its entirety was the timeframe for which data was provided. As mentioned above, this dataset gives values from the first weekend of Night Tube service. At present, Night Tube service runs on 5 different Underground Lines. Initially, however, Night Tube only ran on 2 lines - the Central and Victoria Lines, followed by Jubilee and Northern Line commencing their service in October and November 2016 respectively, and then finally, Piccadilly Line from December 2016 onward. This will lead to gaps or null values for stations which began providing service after the official launch, when all stations are grouped by date and time accordingly.

Data Cleaning and Validation

In order to analyse the dataset effectively, the following steps are done as shown in the code segments below.

Firstly, a copy of the dataset is created on which cleaning will be performed. For simplicity, the columns are renamed as follows:

In [84]:
```
clean = dataraw.copy()

clean.rename(columns = {'Station Name' : 'St','Number of Entries' : 'En', 'Number of

clean.head(6)
```

Out[84]:

	Date	St	Time	En	Ex
0	19-08-16	Bank	0:30:00	228	77
1	19-08-16	Bank	1:00:00	189	117
2	19-08-16	Bank	1:30:00	184	104
3	19-08-16	Bank	2:00:00	137	74
4	19-08-16	Bank	2:30:00	84	37
5	19-08-16	Bank	3:00:00	90	48

Next, all values from the Entry and Exit columns (now `'En'` and `'Ex'`) which are recorded as **'fewer than 5'** are changed to **3**, where 3 is used as a benchmark / median value.

In later processing, 0 values will denote stations which did not have Night Tube service during the given timeframes, and so this must be segregated from instances during which passengers were recorded entering or exiting stations.

In [85]:
```
clean['En'] = clean['En'].replace(['Fewer than 5'], 3)
clean['Ex'] = clean['Ex'].replace(['Fewer than 5'], 3)

clean.tail(11)
```

Out[85]:

	Date	St	Time	En	Ex
58451	18-02-17	Woodside Park	1:00:00	3	17
58452	18-02-17	Woodside Park	1:30:00	3	23
58453	18-02-17	Woodside Park	2:00:00	3	17
58454	18-02-17	Woodside Park	2:30:00	3	16
58455	18-02-17	Woodside Park	3:00:00	3	8
58456	18-02-17	Woodside Park	3:30:00	3	12
58457	18-02-17	Woodside Park	4:00:00	3	7
58458	18-02-17	Woodside Park	4:30:00	3	8
58459	18-02-17	Woodside Park	5:00:00	3	3
58460	18-02-17	Woodside Park	5:30:00	3	3
58461	18-02-17	Woodside Park	6:00:00	9	3

As this dataset provides a temporal range, the next step is to combine the `'Date'` and `'Time'` columns to form a `'Datetime'` column:

1. Both the `'Date'` and `'Time'` columns are converted from string to their appropriate date and time formats.

In [87]:
```
from datetime import datetime

for date in clean['Date']:
    date = datetime.strptime(date, '%d-%m-%y').date()


for time in clean['Time']:
    time = datetime.strptime(time, '%H:%M:%S').time()

clean[['Date', 'Time']].head(6)
```

Out[87]:

	Date	Time
0	19-08-16	0:30:00
1	19-08-16	1:00:00
2	19-08-16	1:30:00
3	19-08-16	2:00:00
4	19-08-16	2:30:00

	Date	Time
5	19-08-16	3:00:00

2. The `'Date'` and `'Time'` columns are then combined to create `'datetime'`

In [88]:
```
clean['datetime'] = clean["Date"] + " " + clean["Time"]

clean[['datetime']].head(6)
```

Out[88]:

	datetime
0	19-08-16 0:30:00
1	19-08-16 1:00:00
2	19-08-16 1:30:00
3	19-08-16 2:00:00
4	19-08-16 2:30:00
5	19-08-16 3:00:00

In [89]:
```
clean.dtypes
```

Out[89]:
```
Date        object
St          object
Time        object
En          object
Ex          object
datetime    object
dtype: object
```

From above, it shows that all values are objects, however `'datetime'`, `'En'` and `'Ex'` must be adjusted to datetime and integer types respectively, and the original `'Date'` and `'Time'` columns are removed as they are no longer required:

In [90]:
```
for dt in clean['datetime']:
    dt = datetime.strptime(dt, '%d-%m-%y %H:%M:%S')

clean['datetime'] = pd.to_datetime(clean['datetime'], dayfirst = True)

clean['En'] = pd.to_numeric(clean['En'])
clean['Ex'] = pd.to_numeric(clean['Ex'])

clean = clean.drop(['Date', 'Time'], axis = 1)
clean.dtypes
```

Out[90]:
```
St                  object
En                   int64
Ex                   int64
datetime    datetime64[ns]
dtype: object
```

In [91]:
```
clean.head(6)
```

Out[91]:

	St	En	Ex	datetime
0	Bank	228	77	2016-08-19 00:30:00
1	Bank	189	117	2016-08-19 01:00:00
2	Bank	184	104	2016-08-19 01:30:00

	St	En	Ex	datetime
3	Bank	137	74	2016-08-19 02:00:00
4	Bank	84	37	2016-08-19 02:30:00
5	Bank	90	48	2016-08-19 03:00:00

Adding Spatial Properties and Station Details

Now that this dataset is cleaned, further attributes can be added, such as station coordinates, congestion zones, and the lines which each station serves.

For this, a new dataset, taken from 'What Do They Know', provides this information.

In [18]:
```
geom = pd.read_csv("Stations.csv")
geom
```

Out[18]:

	FID	OBJECTID	NAME	EASTING	NORTHING	LINES	NETWORK	Zone	
0	291	477	ABBEY ROAD - DLR	539077	183399	NaN	DLR	3	0.0053
1	258	162	Acton Central	520632	180296	NaN	London Overground	3	-0.2615
2	225	204	Acton Town	519478	179592	District, Piccadilly	London Underground	3	-0.2784
3	455	421	Addington Village	537066	163744	NaN	Tramlink	6	-0.0312
4	422	425	Addiscombe	534188	166297	NaN	Tramlink	5	-0.0716
5	6	84	Aldgate	533613	181262	Metropolitan, Circle	London Underground	1	-0.0742
6	61	249	Aldgate East	533936	181375	Hammersmith & City, District	London Underground	1	-0.0695
7	170	448	ALL SAINTS - DLR	537979	180993	NaN	DLR	2	-0.0114
8	309	13	Alperton	517996	183795	Piccadilly	London Underground	4	-0.2983
9	472	284	Amersham	496372	198181	Metropolitan	London Underground	9	-0.6061
10	365	406	Ampere Way	530664	166485	NaN	Tramlink	4	-0.1221
11	375	349	Anerley	534617	169899	NaN	London Overground	4	-0.0641
12	59	331	Angel	531561	183100	Northern	London Underground	1	-0.1031
13	107	301	Archway	529364	186721	Northern	London Underground	2	-0.1334
14	366	428	Arena	535194	167627	NaN	Tramlink	4	-0.0566
15	310	22	Arnos Grove	529355	192498	Piccadilly	London Underground	4	-0.1314

	FID	OBJECTID	NAME	EASTING	NORTHING	LINES	NETWORK	Zone	
16	80	21	Arsenal	531305	186112	Piccadilly	London Underground	2	-0.1056
17	370	432	Avenue Road	535745	169322	NaN	Tramlink	4	-0.0481
18	14	92	Baker Street	527955	182064	Metropolitan, Bakerloo, Circle, Jubilee, Hamme...	London Underground	1	-0.1554
19	238	304	Balham	528480	173244	Northern	London Underground	3	-0.1510
20	35	47	Bank	532710	181120	Waterloo & City, Northern, Central	London Underground	1	-0.0872
21	67	473	BANK - DLR	532709	181114	NaN	DLR	1	-0.0873
22	9	87	Barbican	532004	181856	Metropolitan, Circle, Hammersmith & City	London Underground	1	-0.0971
23	328	200	Barking	544359	184382	District, Hammersmith & City	London Underground	4	0.0818
24	378	368	Barking	544422	184332	NaN	London Overground	4	0.0827
...	...	...	...	...	...	...	...	...	
454	188	471	WEST INDIA QUAY - DLR	537495	180586	NaN	DLR	2	-0.0185
455	130	242	West Kensington	524685	178398	District	London Underground	2	-0.2038
456	434	75	West Ruislip	508379	186796	Central	London Underground	6	-0.4360
457	283	446	WEST SILVERTOWN - DLR	540460	180156	NaN	DLR	3	0.0239
458	87	36	Westbourne Park	524901	181756	Hammersmith & City, Circle	London Underground	2	-0.1995
459	189	472	WESTFERRY - DLR	537023	180798	NaN	DLR	2	-0.0253
460	42	195	Westminster	530196	179668	District, Circle, Jubilee	London Underground	1	-0.1240
461	197	373	Westompton	525397	178008	NaN	London Overground	2	-0.1937
462	72	105	White City	523308	180748	Central	London Underground	2	-0.2228
463	256	158	White Hart Lane	533693	191333	NaN	London Overground	3	-0.0692
464	124	338	Whitechapel	534698	181851	NaN	London Overground	2	-0.0583

	FID	OBJECTID	NAME	EASTING	NORTHING	LINES	NETWORK	Zone	
465	136	250	Whitechapel	534645	181850	District, Hammersmith & City	London Underground	2	-0.0591
466	234	296	Willesden Green	523329	184883	Jubilee	London Underground	3	-0.2211
467	248	261	Willesden Junction	521858	182960	Bakerloo	London Underground	3	-0.2429
468	478	364	Willesden Junction	521879	182944	NaN	London Overground	0	-0.2426
469	240	333	Wimbledon	524800	170684	District	London Underground	3	-0.2049
470	269	396	Wimbledon (Tramlink)	524838	170637	NaN	Tramlink	3	-0.2043
471	203	100	Wimbledon Park	525282	172100	District	London Underground	3	-0.1974
472	220	193	Wood Green	531015	190428	Piccadilly	London Underground	3	-0.1082
473	121	334	Wood Lane	523374	180519	Hammersmith & City, Circle	London Underground	2	-0.2219
474	323	64	Woodford	540950	191740	Central	London Underground	4	0.0356
475	477	363	Woodgrange Park	541821	185350	NaN	London Overground	0	0.0456
476	424	427	Woodside	534701	167101	NaN	Tramlink	5	-0.0639
477	343	264	Woodside Park	525718	192588	Northern	London Underground	4	-0.1839
478	373	447	WOOLWICH ARSENAL - DLR	543731	178811	NaN	DLR	4	0.0704

479 rows × 10 columns

From this dataset, only the `'NAME'`, `'LINES'`, `'Zone'` and x-y coordinates are required for stations on the London Underground. These values are filtered out to create a new dataframe, `'LUL'`.

The tube stations are filtered out in this step, as the text formatting of DLR, Overground, and Rail stations in the dataset may affect the final dataframe when both datasets are merged.

In [19]:
```
LUL = geom.loc[geom.NETWORK == 'London Underground']

LUL = LUL[['NAME', 'LINES', 'Zone', 'x', 'y']]

LUL.head(6)
```

Out[19]:

	NAME	LINES	Zone	x	y
2	Acton Town	District, Piccadilly	3	-0.278433	51.502137

	NAME	LINES	Zone	x	y
5	Aldgate	Metropolitan, Circle	1	-0.074236	51.513982
6	Aldgate East	Hammersmith & City, District	1	-0.069540	51.514917
8	Alperton	Piccadilly	4	-0.298361	51.540227
9	Amersham	Metropolitan	9	-0.606147	51.673662
12	Angel	Northern	1	-0.103116	51.530980

An `'inner join'` is then performed. This filters the information for only the required stations from the `'LUL'` dataframe and merges it with the passenger dataset.

In [20]:
```
full = pd.merge(left = LUL, right = clean, left_on = 'NAME', right_on = 'St')

full.rename(columns = {'NAME' : 'Station','LINES' : 'Lines'}, inplace = True)
full = full.drop(['St'], axis = 1)

full.head(11)
```

Out[20]:

	Station	Lines	Zone	x	y	En	Ex	datetime
0	Acton Town	District, Piccadilly	3	-0.278433	51.502137	31	175	2016-12-16 00:30:00
1	Acton Town	District, Piccadilly	3	-0.278433	51.502137	31	115	2016-12-16 01:00:00
2	Acton Town	District, Piccadilly	3	-0.278433	51.502137	9	100	2016-12-16 01:30:00
3	Acton Town	District, Piccadilly	3	-0.278433	51.502137	9	95	2016-12-16 02:00:00
4	Acton Town	District, Piccadilly	3	-0.278433	51.502137	12	53	2016-12-16 02:30:00
5	Acton Town	District, Piccadilly	3	-0.278433	51.502137	5	41	2016-12-16 03:00:00
6	Acton Town	District, Piccadilly	3	-0.278433	51.502137	3	45	2016-12-16 03:30:00
7	Acton Town	District, Piccadilly	3	-0.278433	51.502137	3	38	2016-12-16 04:00:00
8	Acton Town	District, Piccadilly	3	-0.278433	51.502137	23	30	2016-12-16 04:30:00
9	Acton Town	District, Piccadilly	3	-0.278433	51.502137	32	22	2016-12-16 05:00:00
10	Acton Town	District, Piccadilly	3	-0.278433	51.502137	41	140	2016-12-17 00:30:00

Various stations in the dataframe serve multiple tube lines, some of which do not run Night Tube Service.

For the appropriate stations, line information will be edited to keep only those which serve Night Tube, or in cases where stations have multiple lines which run service, only those which first served Night Tube will remain, as the data does not specifically state which line was used by passengers at a given station.

In [92]:
```
full.loc[full['Lines'].str.contains('Central', case = False), 'Lines'] = 'Central'

full.loc[full['Lines'].str.contains('Victoria', case = False), 'Lines'] = 'Victoria'

full.loc[full['Lines'].str.contains('Jubilee', case = False), 'Lines'] = 'Jubilee'

full.loc[full['Lines'].str.contains('Northern', case = False), 'Lines'] = 'Northern'

full.loc[full['Lines'].str.contains('Piccadilly', case = False), 'Lines'] = 'Piccadi
```

```
full.head(6)
```

Out[92]:

	Station	Lines	Zone	x	y	En	Ex	datetime
0	Acton Town	Piccadilly	3	-0.278433	51.502137	31	175	2016-12-16 00:30:00
1	Acton Town	Piccadilly	3	-0.278433	51.502137	31	115	2016-12-16 01:00:00
2	Acton Town	Piccadilly	3	-0.278433	51.502137	9	100	2016-12-16 01:30:00
3	Acton Town	Piccadilly	3	-0.278433	51.502137	9	95	2016-12-16 02:00:00
4	Acton Town	Piccadilly	3	-0.278433	51.502137	12	53	2016-12-16 02:30:00
5	Acton Town	Piccadilly	3	-0.278433	51.502137	5	41	2016-12-16 03:00:00

Data Analysis

The main dataframes have been processed accordingly, and now various forms of analyses and visualisations can be completed.

Upon observing the values for the number of passengers entering and exiting stations, there is a large range, from under a hundred, to almost tens of thousands of passengers at a given timeframe.

To visualise this information fully, **pivot tables** are created for specific categories. These can include:

- Data based on each line
- Data based on each station per line
- Data based on each zone

Entries and Exits according to Line

In [ ]:
```
#conda install -c plotly plotly=4.6.0
```

In [22]:
```
import plotly.graph_objects as go
```

In [93]:
```
lines_en = (full.pivot_table(index = 'datetime',
                             columns = 'Lines',
                             values = 'En',
                             fill_value = 0))

lines_en = lines_en.resample('W-Fri').sum() #Grouping values on a weekly basis

lines_en = lines_en.reset_index().rename_axis(None, axis=1)
lines_en.head(6)
```

Out[93]:

	datetime	Central	Jubilee	Northern	Piccadilly	Victoria
0	2016-08-19	517.281250	0.0	0.0	0.0	494.222222
1	2016-08-26	1118.625000	0.0	0.0	0.0	1144.236111
2	2016-09-02	1093.937500	0.0	0.0	0.0	1152.222222
3	2016-09-09	1135.263105	0.0	0.0	0.0	1240.777778
4	2016-09-16	1194.500000	0.0	0.0	0.0	1279.111111

	datetime	Central	Jubilee	Northern	Piccadilly	Victoria
5	2016-09-23	1223.281250	0.0	0.0	0.0	1310.777778

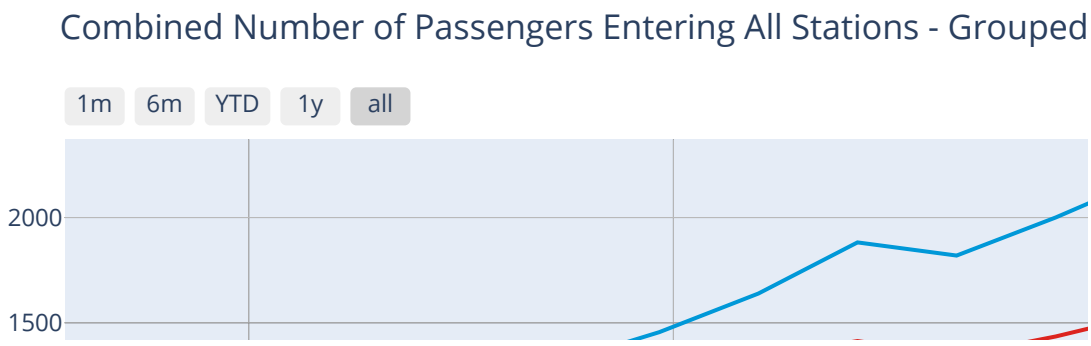
In [94]:
```
lines_ex = (full.pivot_table(index = 'datetime',
                             columns = 'Lines',
                             values = 'Ex',
                             fill_value = 0))

lines_ex = lines_ex.resample('W-Fri').sum() #Grouping values on a weekly basis

lines_ex = lines_ex.reset_index().rename_axis(None, axis=1)
lines_ex.head(6)
```

Out[94]:

	datetime	Central	Jubilee	Northern	Piccadilly	Victoria
0	2016-08-19	541.625000	0.0	0.0	0.0	763.000000
1	2016-08-26	1159.750000	0.0	0.0	0.0	1633.777778
2	2016-09-02	1137.093750	0.0	0.0	0.0	1601.888889
3	2016-09-09	1160.422379	0.0	0.0	0.0	1664.333333
4	2016-09-16	1285.218750	0.0	0.0	0.0	1782.888889
5	2016-09-23	1301.937500	0.0	0.0	0.0	1867.888889

In [25]:
```
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=list(lines_en.datetime), y=list(lines_en.Central), name = "Central"
               line=dict(color='#dc241f')))

fig.add_trace(
    go.Scatter(x=list(lines_en.datetime), y=list(lines_en.Jubilee), name = "Jubilee"
               line=dict(color='#a1a5a7')))

fig.add_trace(
    go.Scatter(x=list(lines_en.datetime), y=list(lines_en.Northern), name = "Norther
               line=dict(color='#000000')))

fig.add_trace(
    go.Scatter(x=list(lines_en.datetime), y=list(lines_en.Piccadilly), name = "Picca
               line=dict(color='#001928')))

fig.add_trace(
    go.Scatter(x=list(lines_en.datetime), y=list(lines_en.Victoria), name = "Victori
               line=dict(color='#0098d8')))

fig.update_layout(
    title_text="Combined Number of Passengers Entering All Stations - Grouped By Lin
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
```

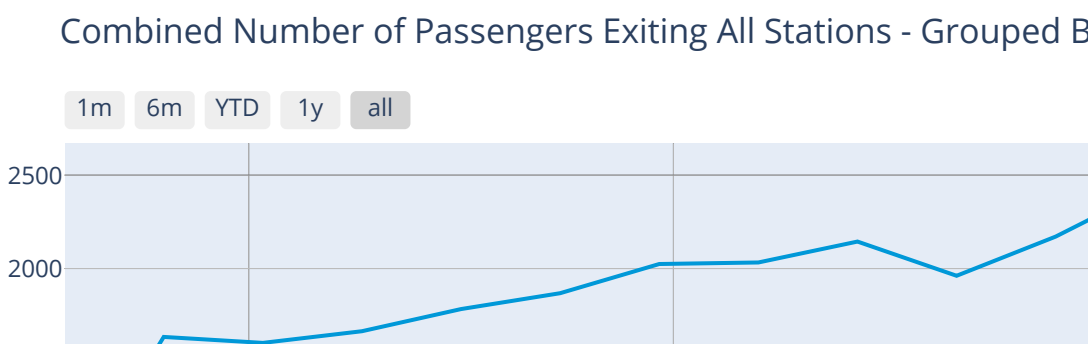
```
                    label="6m",
                    step="month",
                    stepmode="backward"),
                dict(count=1,
                    label="YTD",
                    step="year",
                    stepmode="todate"),
                dict(count=1,
                    label="1y",
                    step="year",
                    stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



In [26]:

```
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=list(lines_ex.datetime), y=list(lines_ex.Central), name = "Central"
               line=dict(color='#dc241f')))

fig.add_trace(
```

```
    go.Scatter(x=list(lines_ex.datetime), y=list(lines_ex.Jubilee), name = "Jubilee"
               line=dict(color='#a1a5a7')))

fig.add_trace(
    go.Scatter(x=list(lines_ex.datetime), y=list(lines_ex.Northern), name = "Norther
               line=dict(color='#000000')))

fig.add_trace(
    go.Scatter(x=list(lines_ex.datetime), y=list(lines_ex.Piccadilly), name = "Picca
               line=dict(color='#001928')))

fig.add_trace(
    go.Scatter(x=list(lines_ex.datetime), y=list(lines_ex.Victoria), name = "Victori
               line=dict(color='#0098d8')))

fig.update_layout(
    title_text="Combined Number of Passengers Exiting All Stations - Grouped By Line
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



From observing both the number of entries and exits across all lines, it is shown that Victoria Line is the most popular, and Piccadilly Line is the least. The popularity can be attributed to the locations which these lines are situated, for example, popular tourist and nightlife destinations. This will be further observed when data is categorised according to station.

The popularity of the Victoria Line can also be due to the fact it only runs through Zones 1 to 3, through Central London and popular train stations, such as Victoria and Vauxhall. This provides a convenient link for commuters travelling via train, both to destinations out of Greater London, or to local airports.

There is a significant drop in values for January 2017. This is due to a TfL strike which occurred across one weekend for 24 hours. Based on the trends, however, the less popular lines were not greatly affected by this strike.

Central Line

In [33]:
```
EN_C = (full.loc[full.Lines == 'Central'].pivot_table(index = 'datetime',
                            columns = 'Station',
                            values = 'En',
                            fill_value = 0))

EN_C = EN_C.resample('W-Fri').sum() #Grouping values on a weekly basis

EN_C = EN_C.reset_index().rename_axis(None, axis=1)
EN_C.head(2)
```

Out[33]:

	datetime	Bank	Barkingside	Bond Street	Buckhurst Hill	Chancery Lane	Ealing Broadway	East Acton	Fairlop	Gants Hill	...
0	2016-08-19	1054	46	476	38	426	268	56	47	136	...
1	2016-08-26	1817	98	1110	68	758	709	153	96	333	...

2 rows × 33 columns

In [34]:
```
EX_C = (full.loc[full.Lines == 'Central'].pivot_table(index = 'datetime',
```

```
                              columns = 'Station',
                              values = 'Ex',
                              fill_value = 0))

EX_C = EX_C.resample('W-Fri').sum() #Grouping values on a weekly basis

EX_C = EX_C.reset_index().rename_axis(None, axis=1) #Converting pivot table back to
EX_C.head(2)
```

Out[34]:

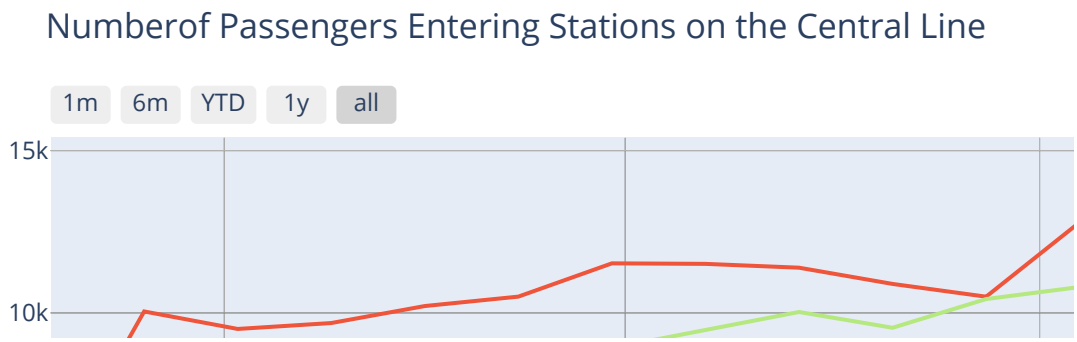
	datetime	Bank	Barkingside	Bond Street	Buckhurst Hill	Chancery Lane	Ealing Broadway	East Acton	Fairlop	Gants Hill	...
0	2016-08-19	580	57	357	160	131	1549	333	47	463	...
1	2016-08-26	1294	128	672	289	320	3386	731	91	940	...

2 rows × 33 columns

In [28]:

```
fig = go.Figure()

for i in EN_C.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EN_C.datetime), y=list(EN_C[i]), name = i))

# Set title
fig.update_layout(
    title_text="Numberof Passengers Entering Stations on the Central Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)
```

```
fig.show()
```



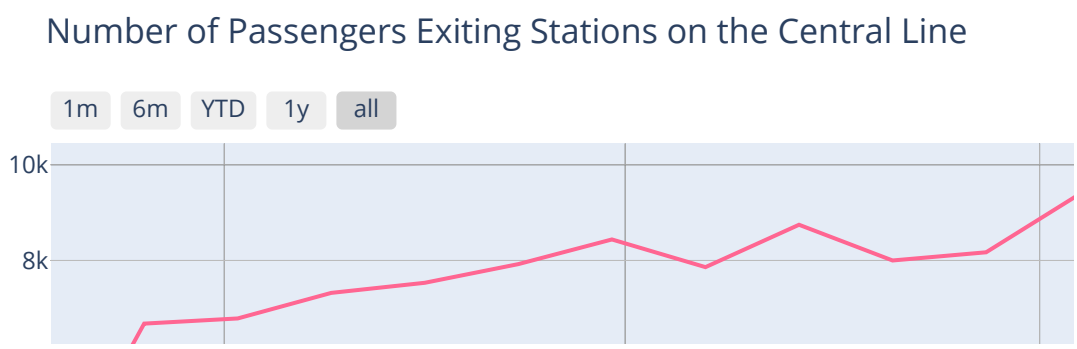
In [29]:

```
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (40,40))
num = 0

try:
    for i in EN_C.drop('datetime', axis = 1):
        num += 1
        plt.subplot(5, 7, num)
        plt.plot(EN_C['datetime'], EN_C[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 20)
        plt.suptitle("Number of Passengers Entering\n Stations on the Central Line",

        plt.ylim(0, 15000)

        if num in range(25):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

From the small multiples plot and range slider above, the most popular stations on the Central Line are Oxford Circus and Tottenham Court Road, with stations including Bank, Bond Street and Holborn trailing closely behind. As all of these stations are located in Zone 1 of Central London, the popularity may be due to the nightlife within Central London, as well as both the accessibility and proximity of these stations to one another.

In [35]:
```
fig = go.Figure()

for i in EX_C.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EX_C.datetime), y=list(EX_C[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Exiting Stations on the Central Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
```

```
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



In [36]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (40,40))
```

```
num = 0

try:
    for i in EX_C.drop('datetime', axis = 1):
        num += 1
        plt.subplot(5, 7, num)
        plt.plot(EX_C['datetime'], EX_C[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 20)
        plt.suptitle("Number of Passengers Exiting\n Stations on the Central Line",

        plt.ylim(30, 10000)

        if num in range(29):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

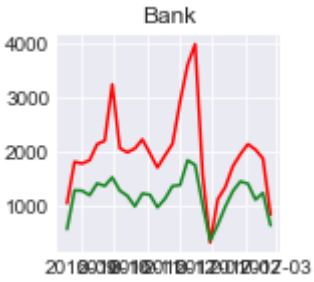
The popularity of stations based on passenger exits varies greatly when compared to those based on entries. From above, the most popular stations are Stratford, Mile End, and Ealing Broadway. These stations serve various lines including Overground and DLR in the case of

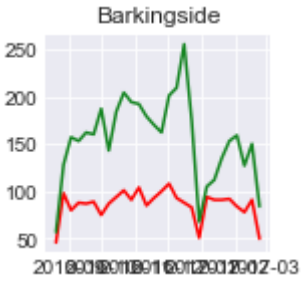
Stratford Station. These additional lines, however, do not run Night Tube Service. The introduction of overnight service therefore provides further accessibility to commuters who would otherwise utilise taxi or Bus services to get to their destinations in these areas.

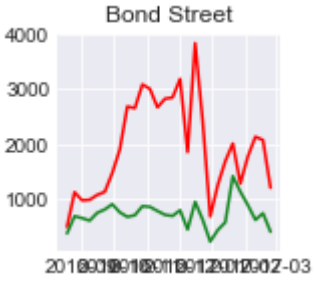
In the plots below, values for passenger entry and exit are compared by station to highlight the variation.

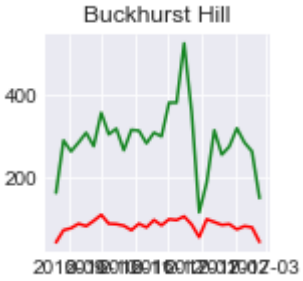
In [37]:
```
for i in EN_C.drop('datetime', axis = 1):
    plt.figure(figsize=(2,2))
    plt.plot(EN_C['datetime'], EN_C[i], 'red') #ENTRIES
    plt.plot(EX_C['datetime'], EX_C[i], '#158522') #EXITS

    plt.title(i)
plt.show()
```

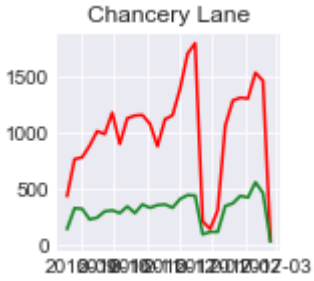
```
C:\Users\Zaana\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning:

More than 20 figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume to
o much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
```

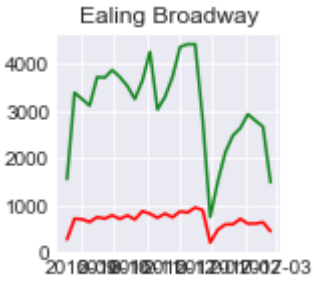


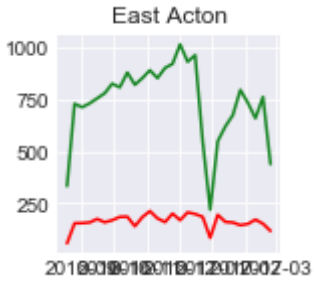


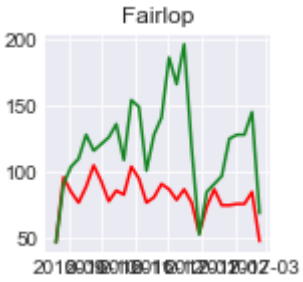


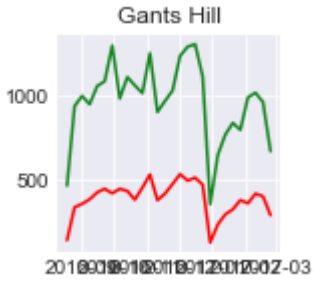


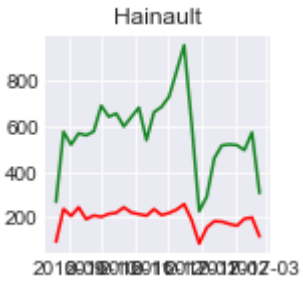

Chancery Lane
1500
1000
500
0

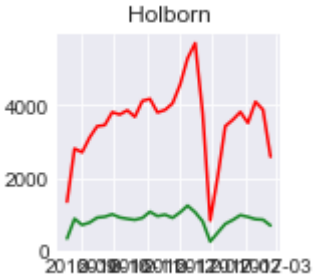

Ealing Broadway
4000
3000
2000
1000
0

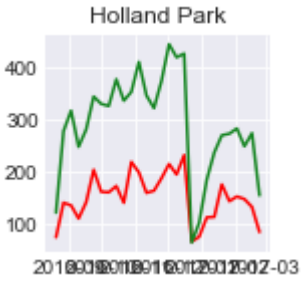

East Acton
1000
750
500
250

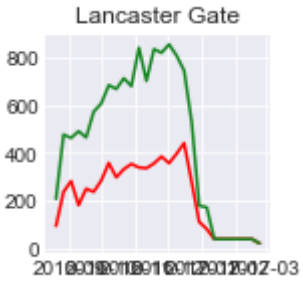

Fairlop
200
150
100
50

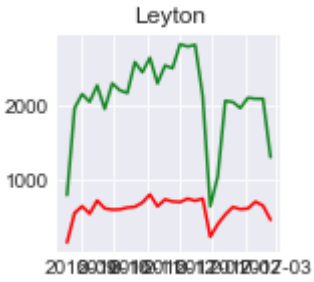

Gants Hill
1000
500

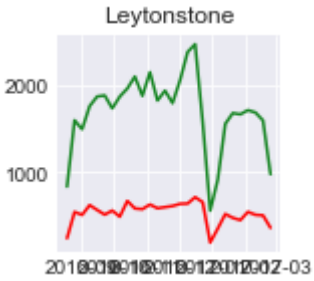

Hainault
800
600
400
200

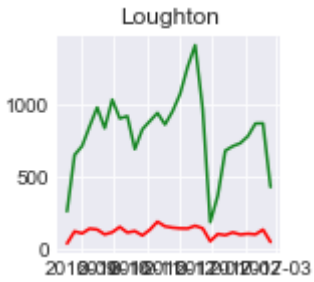

Holborn
4000
2000
0

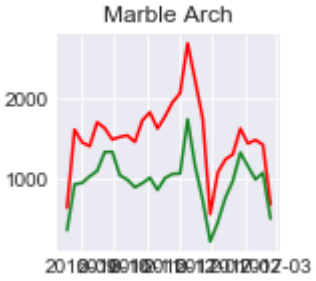

Holland Park
400
300
200
100

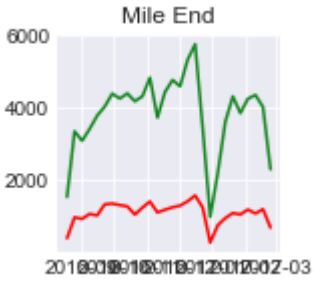

Lancaster Gate
800
600
400
200
0

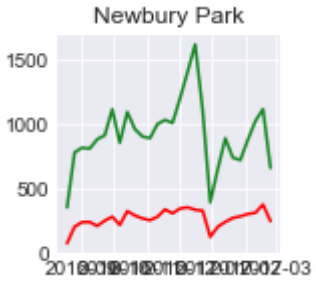

Leyton
2000
1000

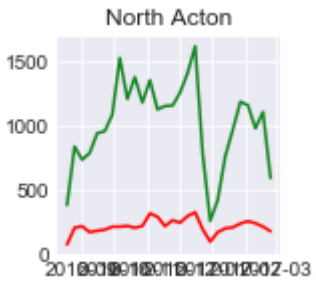

Leytonstone
2000
1000

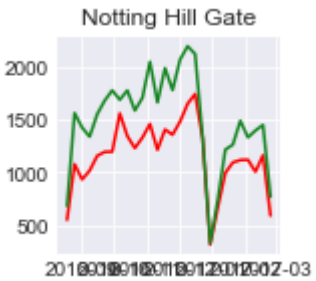

Loughton
1000
500
0

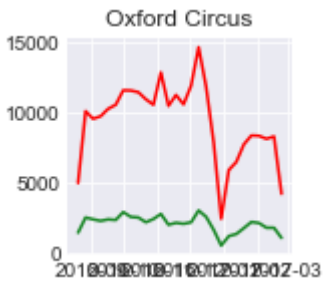

Marble Arch
2000
1000

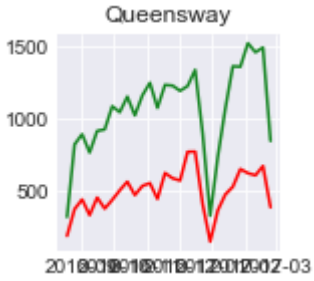

Mile End
6000
4000
2000

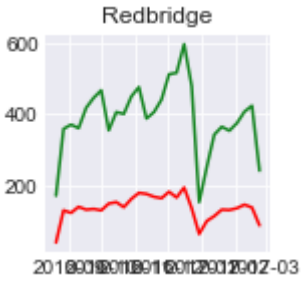

Newbury Park
1500
1000
500
0

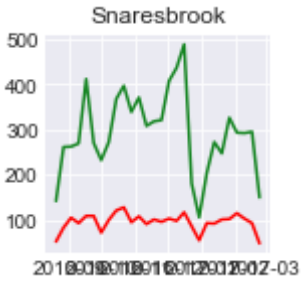

North Acton
1500
1000
500
0

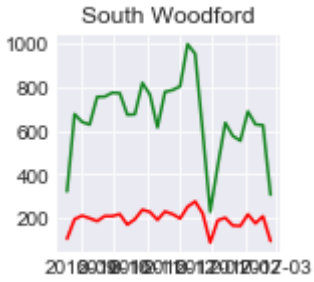

Notting Hill Gate
2000
1500
1000
500

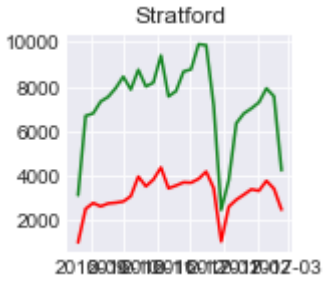

Oxford Circus
15000
10000
5000
0

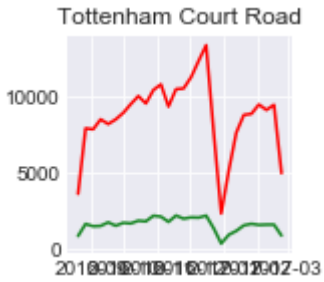

Queensway
1500
1000
500

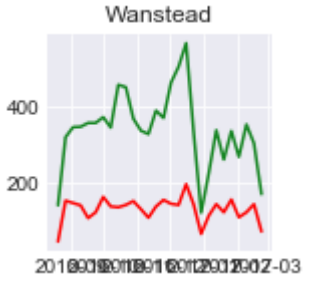

Redbridge
600
400
200

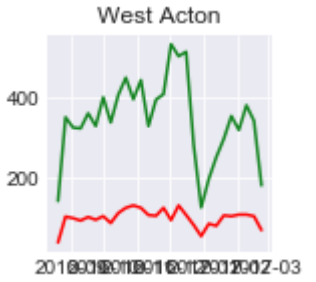

Snaresbrook
500
400
300
200
100

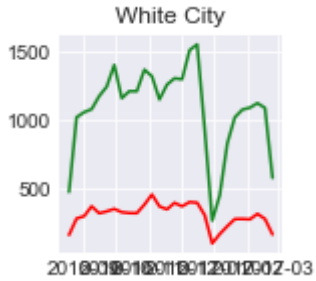

South Woodford
1000
800
600
400
200

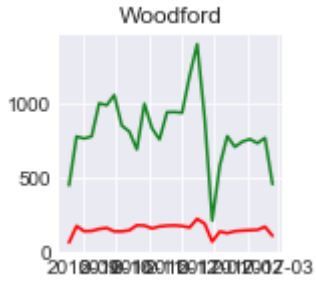

Stratford
10000
8000
6000
4000
2000


Tottenham Court Road
10000
5000
0









Victoria Line

In [38]:
```
EN_V = (full.loc[full.Lines == 'Victoria'].pivot_table(index = 'datetime',
                            columns = 'Station',
                            values = 'En',
                            fill_value = 0))

EN_V = EN_V.resample('W-Fri').sum() #Grouping values on a weekly basis

EN_V = EN_V.reset_index().rename_axis(None, axis=1)
EN_V.head(2)
```

Out[38]:

	datetime	Blackhorse Road	Green Park	Highbury & Islington	Pimlico	Seven Sisters	Stockwell	Tottenham Hale	Walthamstow Central	V
0	2016-08-19	168	1350	736	109	483	581	148	259	
1	2016-08-26	281	2929	1761	254	1358	1491	380	593	

In [39]:
```
EX_V = (full.loc[full.Lines == 'Victoria'].pivot_table(index = 'datetime',
                            columns = 'Station',
```

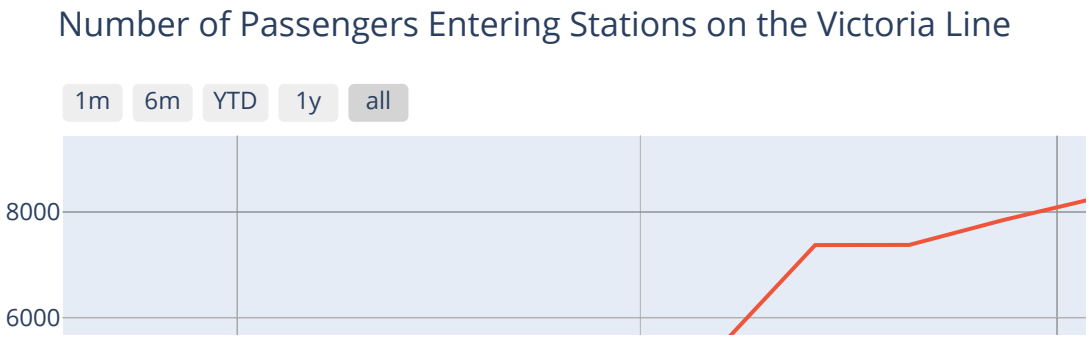
```
                              values = 'Ex',
                              fill_value = 0))

EX_V = EX_V.resample('W-Fri').sum() #Grouping values on a weekly basis

EX_V = EX_V.reset_index().rename_axis(None, axis=1)
EX_V.head(2)
```

Out[39]:

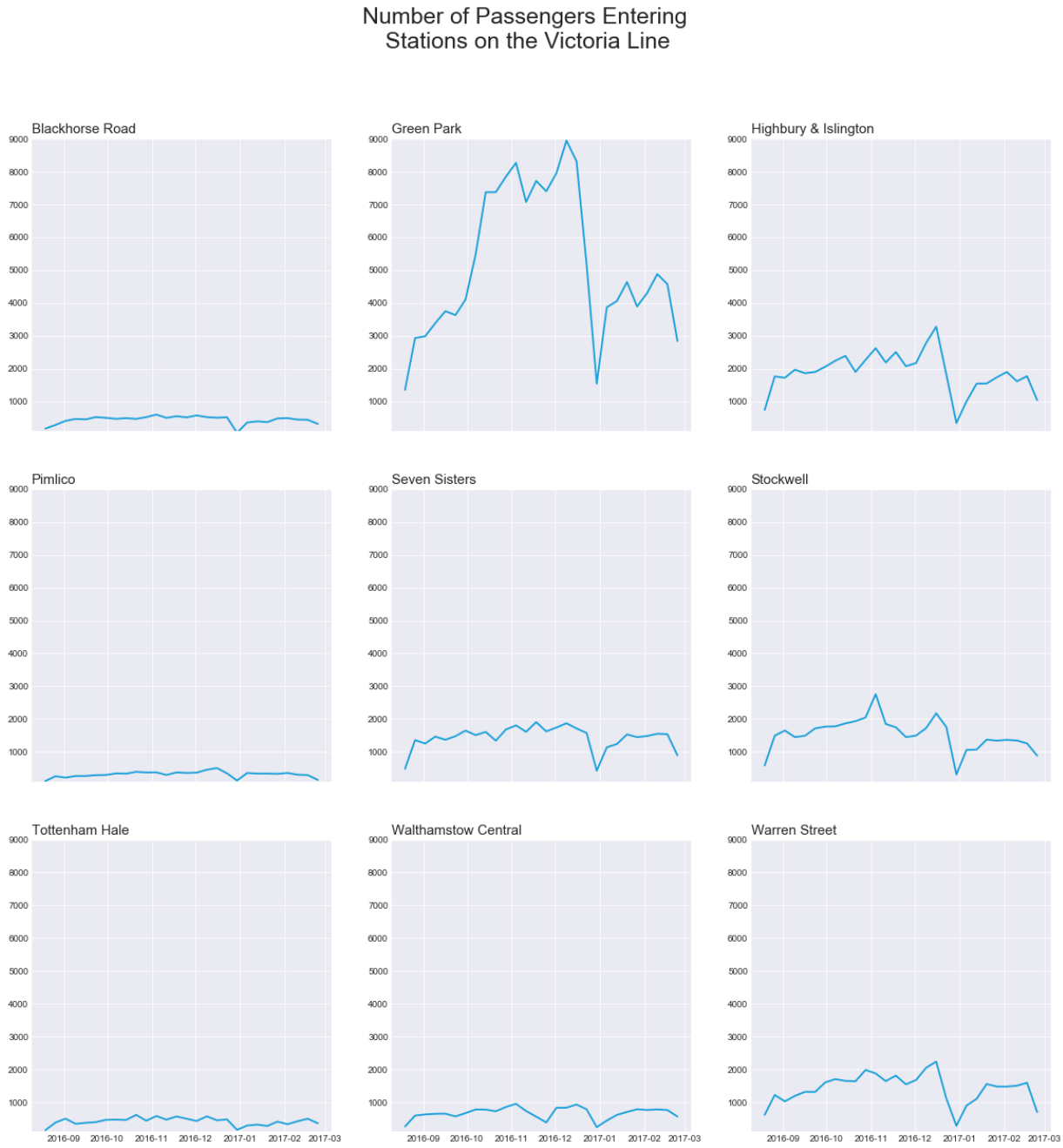
	datetime	Blackhorse Road	Green Park	Highbury & Islington	Pimlico	Seven Sisters	Stockwell	Tottenham Hale	Walthamstow Central	V
0	2016-08-19	517	437	972	290	1482	1352	493	994	
1	2016-08-26	691	1051	2181	630	3304	2827	1004	2224	

In [40]:

```
fig = go.Figure()

for i in EN_V.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EN_V.datetime), y=list(EN_V[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Entering Stations on the Victoria Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



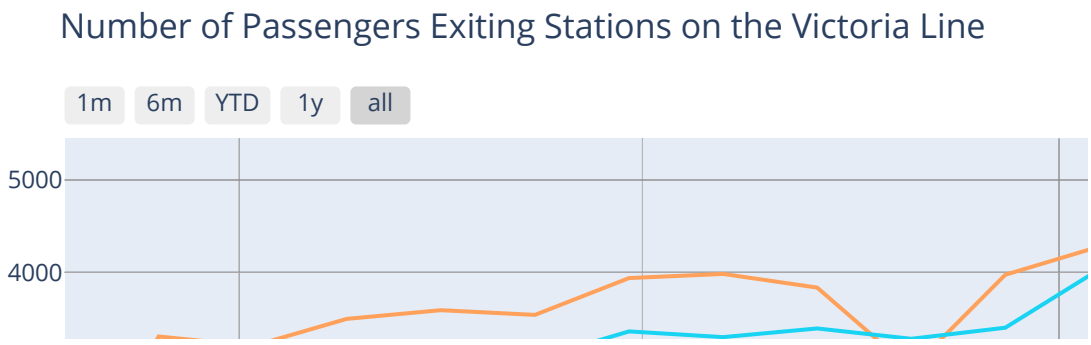
In [41]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EN_V.drop('datetime', axis = 1):
        num += 1
        plt.subplot(3, 3, num)
        plt.plot(EN_V['datetime'], EN_V[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Entering\n Stations on the Victoria Line"

        plt.ylim(100, 9000)

        if num in range(7):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```



The most popular station for passenger entry is Green Park. This can again be credited due to location, as Green Park is located near Buckingham Palace, as well as other stations such as Piccadilly Circus and Leicester Square. As Victoria line ran Night Tube Service before both Piccadilly and Northern Lines, this would have resulted in an influx of passengers.

In [42]:
```
fig = go.Figure()

for i in EX_V.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EX_V.datetime), y=list(EX_V[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Exiting Stations on the Victoria Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
```

```
                    label="1m",
                    step="month",
                    stepmode="backward"),
                dict(count=6,
                    label="6m",
                    step="month",
                    stepmode="backward"),
                dict(count=1,
                    label="YTD",
                    step="year",
                    stepmode="todate"),
                dict(count=1,
                    label="1y",
                    step="year",
                    stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



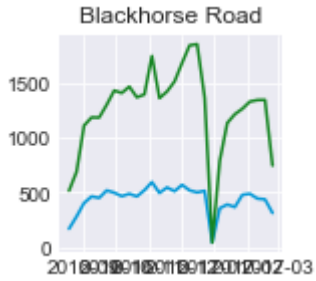
In [43]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0
```

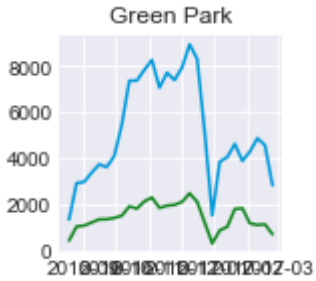
```
try:
    for i in EX_V.drop('datetime', axis = 1):
        num += 1
        plt.subplot(3, 3, num)
        plt.plot(EX_V['datetime'], EX_V[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Exiting\n Stations on the Victoria Line",

        plt.ylim(300, 6000)

        if num in range(7):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

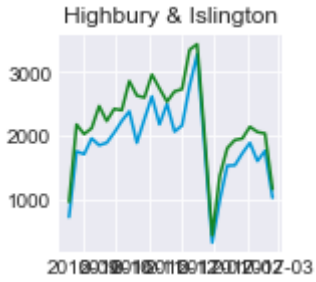
From the graphs above, the number of passenger exits remain fairly within the same range, with Seven Sisters being the most popular. Again, this can be linked to the accessibility of other transport links, as Seven Sisters provides train services to Stansted Airport.

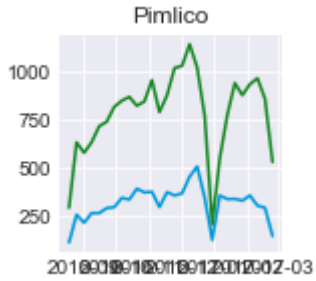
In [44]:
```
for i in EN_V.drop('datetime', axis = 1):
    plt.figure(figsize=(2,2))
```

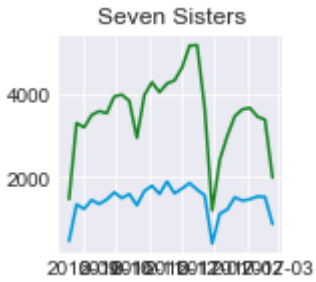
```
    plt.plot(EN_V['datetime'], EN_V[i], color = '#0098d8') #ENTRIES
    plt.plot(EX_V['datetime'], EX_V[i], '#158522') #EXITS

    plt.title(i)
plt.show()
```

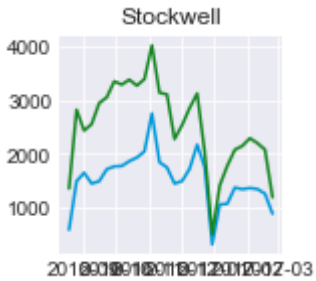


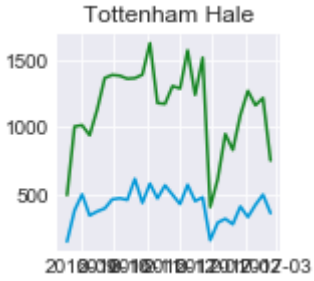


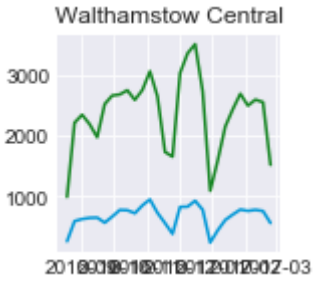


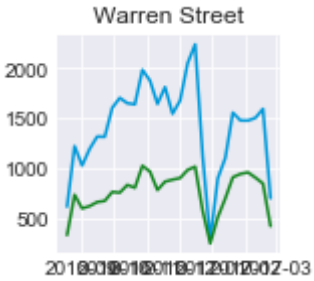












Jubilee Line

In [45]:
```
EN_J = (full.loc[full.Lines == 'Jubilee'].pivot_table(index = 'datetime',
                            columns = 'Station',
                            values = 'En',
                            fill_value = 0))

EN_J = EN_J.resample('W-Fri').sum() #Grouping values on a weekly basis

EN_J = EN_J.reset_index().rename_axis(None, axis=1)
EN_J.head(2)
```

Out[45]:

	datetime	Baker Street	Bermondsey	Canada Water	Canning Town	Canons Park	Dollis Hill	Finchley Road	Kilburn	Kingsbury
0	2016-10-07	587	189	344	351	54	102	143	206	66
1	2016-10-14	974	439	715	1036	112	193	310	526	151

In [46]:
```
EX_J = (full.loc[full.Lines == 'Jubilee'].pivot_table(index = 'datetime',
                            columns = 'Station',
                            values = 'Ex',
                            fill_value = 0))

EX_J = EX_J.resample('W-Fri').sum() #Grouping values on a weekly basis

EX_J = EX_J.reset_index().rename_axis(None, axis=1)
EX_J.head(2)
```

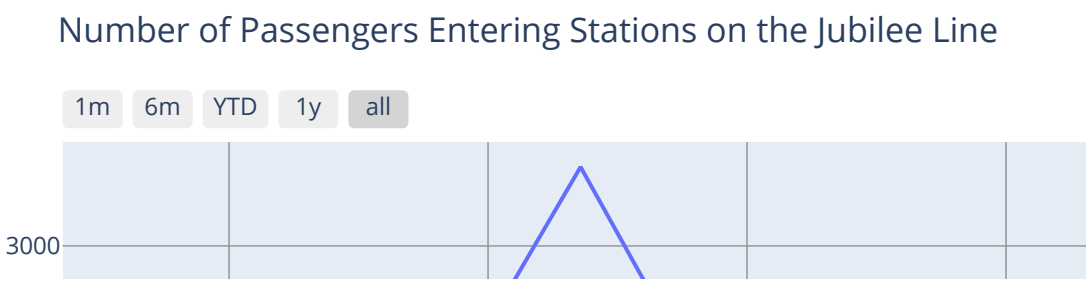
Out[46]:

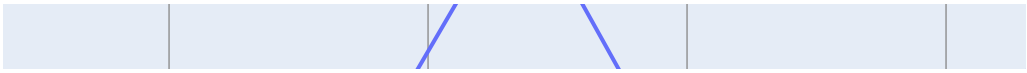
	datetime	Baker Street	Bermondsey	Canada Water	Canning Town	Canons Park	Dollis Hill	Finchley Road	Kilburn	Kingsbury

	datetime	Baker Street	Bermondsey	Canada Water	Canning Town	Canons Park	Dollis Hill	Finchley Road	Kilburn	Kingsbury
0	2016-10-07	374	569	1378	1198	190	315	395	777	220
1	2016-10-14	896	1330	2819	2858	407	660	884	1753	574

In [47]:

```
fig = go.Figure()

for i in EN_J.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EN_J.datetime), y=list(EN_J[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Entering Stations on the Jubilee Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```

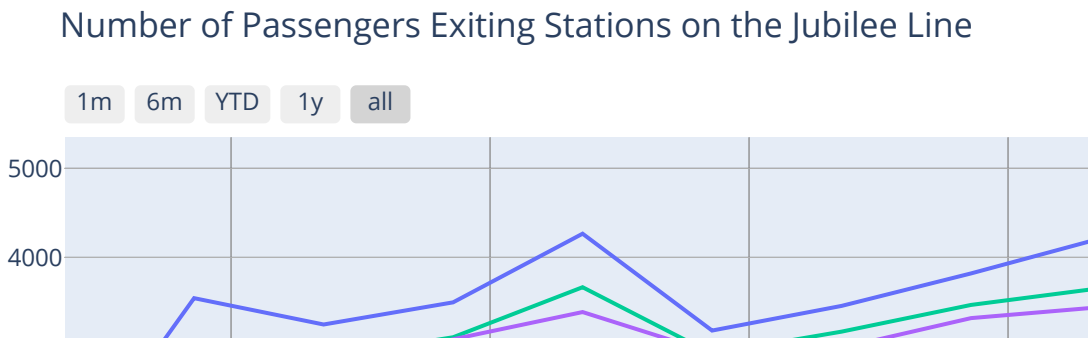




In [48]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EN_J.drop('datetime', axis = 1):
        num += 1
        plt.subplot(3, 7, num)
        plt.plot(EN_J['datetime'], EN_J[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Entering\n Stations on the Jubilee Line",

        plt.ylim(50, 3700)

        if num in range(15):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

In [49]:
```
fig = go.Figure()

for i in EX_J.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EX_J.datetime), y=list(EX_J[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Exiting Stations on the Jubilee Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
```

```
                    step="month",
                    stepmode="backward"),
               dict(count=1,
                    label="YTD",
                    step="year",
                    stepmode="todate"),
               dict(count=1,
                    label="1y",
                    step="year",
                    stepmode="backward"),
               dict(step="all")
           ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



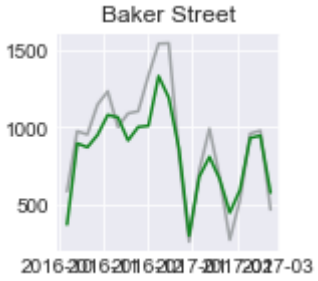
In [50]:

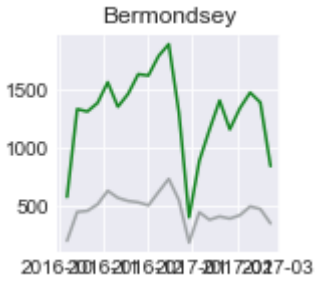
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EX_J.drop('datetime', axis = 1):
        num += 1
        plt.subplot(3, 7, num)
        plt.plot(EX_J['datetime'], EX_J[i], marker = '', linewidth = 1.9, alpha = 0.
```

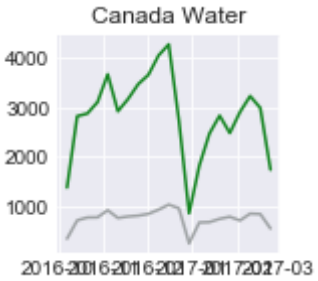
```
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Exiting\n Stations on the Jubilee Line",

        plt.ylim(50, 5500)

        if num in range(15):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

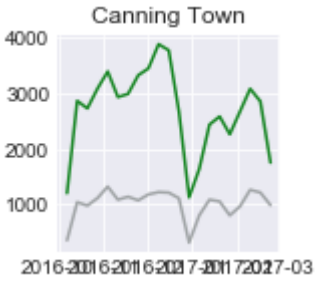
In [51]:

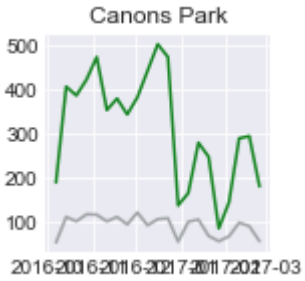
```
for i in EN_J.drop('datetime', axis = 1):
    plt.figure(figsize=(2,2))
    plt.plot(EN_J['datetime'], EN_J[i], color = '#a1a5a7') #ENTRIES
    plt.plot(EX_J['datetime'], EX_J[i], '#158522') #EXITS

    plt.title(i)
plt.show()
```

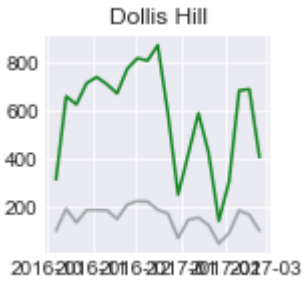

Baker Street
1500
1000
500
2016-03
2017-03

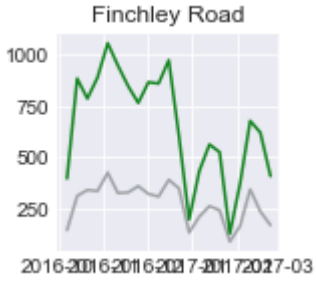

Bermondsey
1500
1000
500
2016-03
2017-03

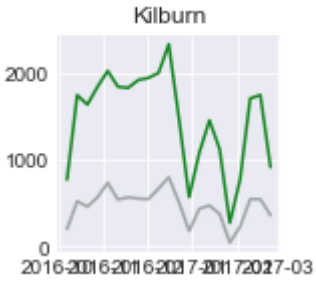

Canada Water
4000
3000
2000
1000
2016-03
2017-03

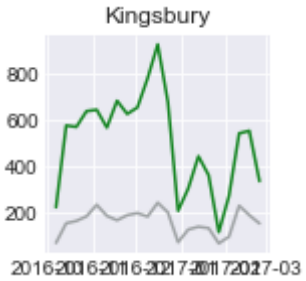

Canning Town
4000
3000
2000
1000
2016-03
2017-03

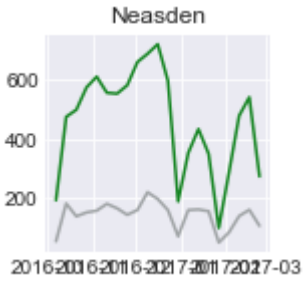

Canons Park
500
400
300
200
100
2016-03
2017-03

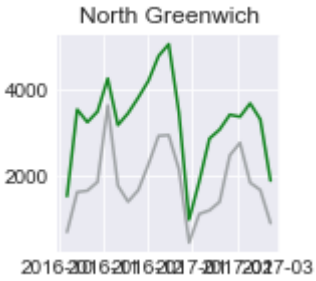

Dollis Hill
800
600
400
200
2016-03
2017-03

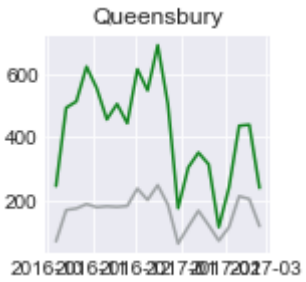

Finchley Road
1000
750
500
250
2016-03
2017-03

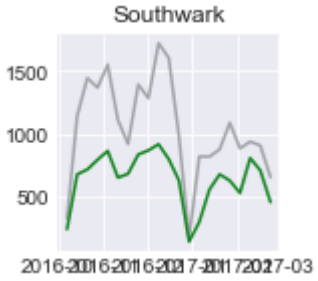

Kilburn
2000
1000
0
2016-03
2017-03

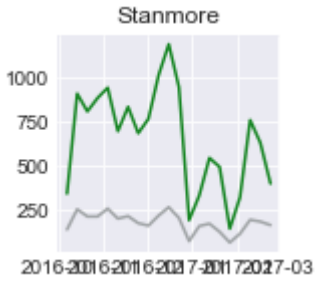

Kingsbury
800
600
400
200
2016-03
2017-03

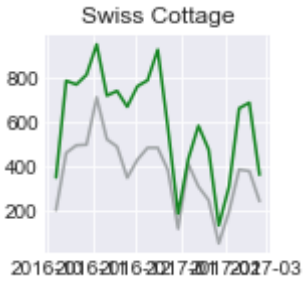

Neasden
600
400
200
2016-03
2017-03

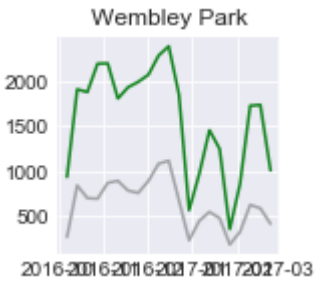

North Greenwich
4000
2000
2016-03
2017-03

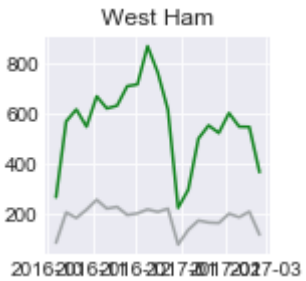

Queensbury
600
400
200
2016-03
2017-03

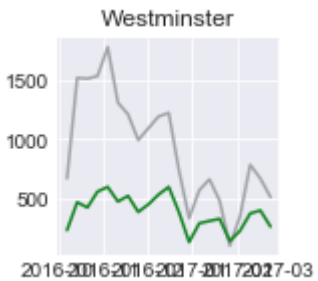

Southwark
1500
1000
500
2016-03
2017-03

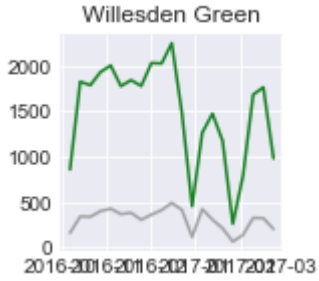

Stanmore
1000
750
500
250
2016-03
2017-03


Swiss Cottage
800
600
400
200
2016-03
2017-03


Wembley Park
2000
1500
1000
500
2016-03
2017-03


West Ham
800
600
400
200
2016-03
2017-03


Westminster
1500
1000
500
2016-03
2017-03



Northern Line

In [52]:
```
EN_N = (full.loc[full.Lines == 'Northern'].pivot_table(index = 'datetime',
                                                       columns = 'Station',
                                                       values = 'En',
                                                       fill_value = 0))

EN_N = EN_N.resample('W-Fri').sum()
EN_N = EN_N.reset_index().rename_axis(None, axis = 1)
EN_N.head(2)
```

Out[52]:

	datetime	Archway	Belsize Park	Brent Cross	Burnt Oak	Camden Town	Chalk Farm	Clapham Common	Clapham North	Clapham South	...	Le
0	2016-11-18	282	57	30	63	3398	271	659	448	132	...	
1	2016-11-25	551	145	85	103	6484	645	1986	1260	380	...	

2 rows × 34 columns

In [53]:
```
EX_N = (full.loc[full.Lines == 'Northern'].pivot_table(index = 'datetime',
                                                       columns = 'Station',
                                                       values = 'Ex',
                                                       fill_value = 0))

EX_N = EX_N.resample('W-Fri').sum()
EX_N = EX_N.reset_index().rename_axis(None, axis = 1)
EX_N.head(2)
```

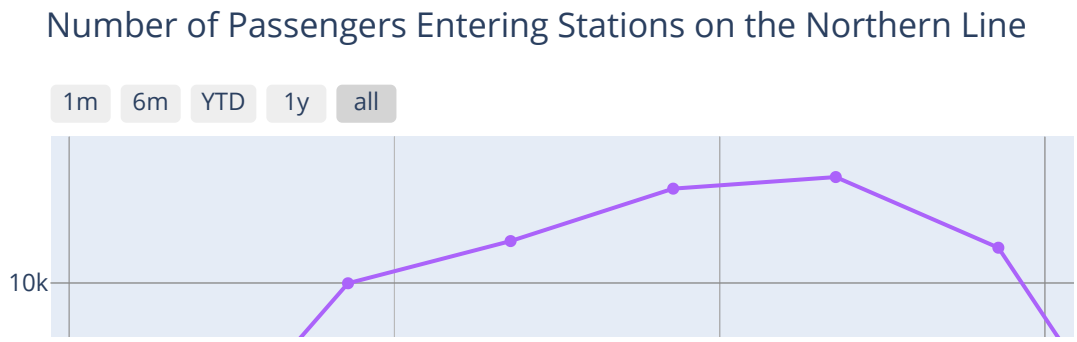
Out[53]:

	datetime	Archway	Belsize Park	Brent Cross	Burnt Oak	Camden Town	Chalk Farm	Clapham Common	Clapham North	Clapham South	...	Le
0	2016-11-18	704	265	158	214	998	199	852	657	607	...	
1	2016-11-25	1528	547	316	529	1932	484	1876	1467	1380	...	

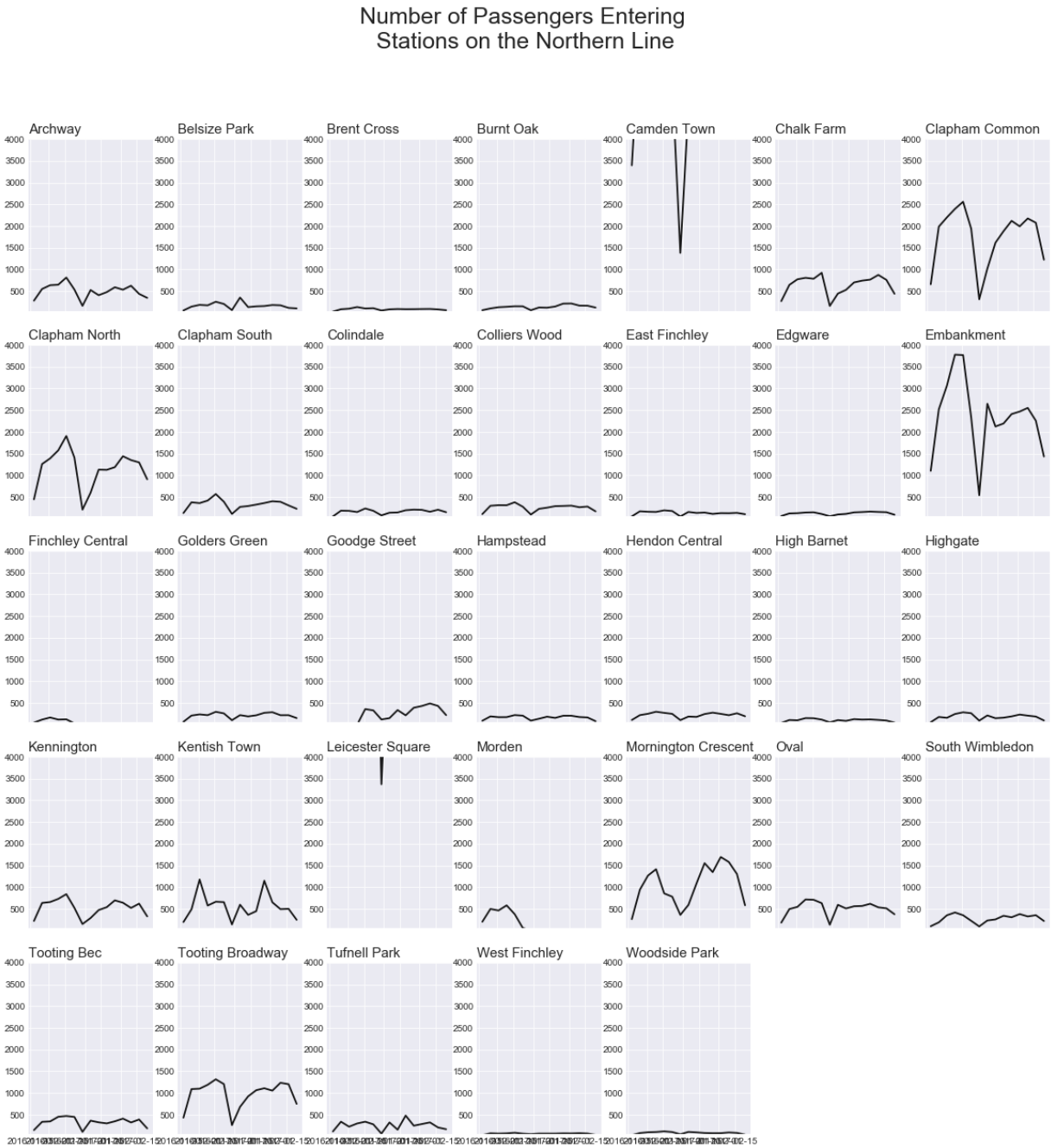
2 rows × 34 columns

In [54]:
```
fig = go.Figure()

for i in EN_N.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EN_N.datetime), y=list(EN_N[i]), name = i))

# Set title
fig.update_layout(
```

```
    title_text="Number of Passengers Entering Stations on the Northern Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



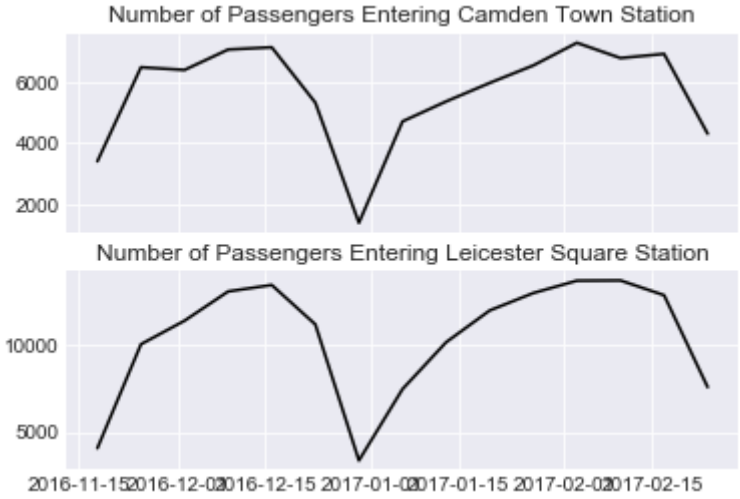
In [55]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EN_N.drop('datetime', axis = 1):
        num += 1
        plt.subplot(5, 7, num)
        plt.plot(EN_N['datetime'], EN_N[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Entering\n Stations on the Northern Line"

        plt.ylim(50, 4000)

        if num in range(29):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```



Plots for Camden Town and Leicester Square Station are shown separately, due to the high volume of passengers

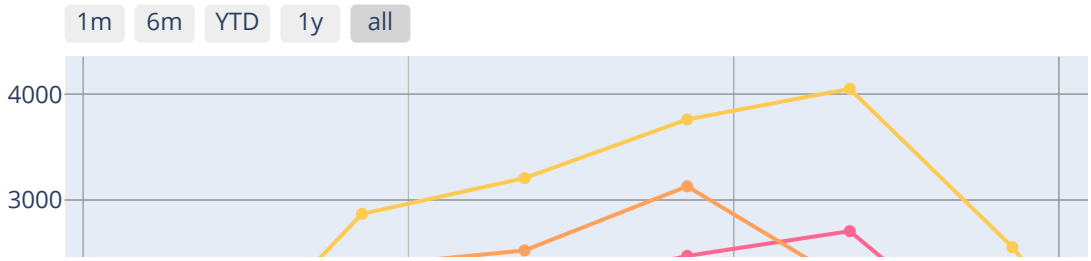
In [56]:
```
fig, axs = plt.subplots(2, sharex = True)

axs[0].plot(EN_N['datetime'], EN_N['Camden Town'], '#000000')
axs[0].set_title("Number of Passengers Entering Camden Town Station")

axs[1].plot(EN_N['datetime'], EN_N['Leicester Square'], '#000000')
axs[1].set_title("Number of Passengers Entering Leicester Square Station")
```

Out[56]:
```
Text(0.5, 1.0, 'Number of Passengers Entering Leicester Square Station')
```



In [57]:
```
fig = go.Figure()

for i in EX_N.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EX_N.datetime), y=list(EX_N[i]), name = i))

# Set title
fig.update_layout(
    title_text="Number of Passengers Exiting Stations on the Northern Line"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```

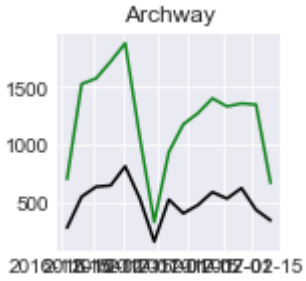
Number of Passengers Exiting Stations on the Northern Line

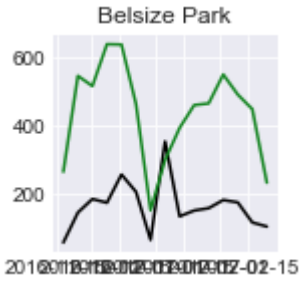


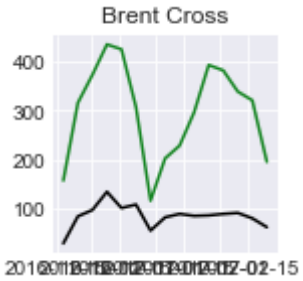
In [58]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EX_N.drop('datetime', axis = 1):
        num += 1
        plt.subplot(5, 7, num)
        plt.plot(EX_N['datetime'], EX_N[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Exiting\n Stations on the Northern Line",

        plt.ylim(10, 4000)

        if num in range(29):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

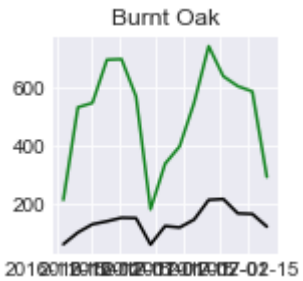
In [59]:
```
for i in EN_N.drop('datetime', axis = 1):
    plt.figure(figsize=(2,2))
    plt.plot(EN_N['datetime'], EN_N[i], color = '#000000') #ENTRIES
    plt.plot(EX_N['datetime'], EX_N[i], '#158522') #EXITS

    plt.title(i)
plt.show()
```

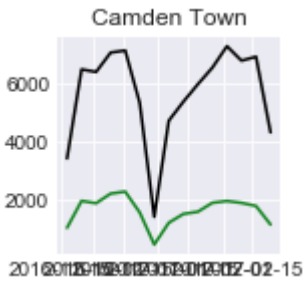
```
C:\Users\Zaana\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning:

More than 20 figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume to
o much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
```

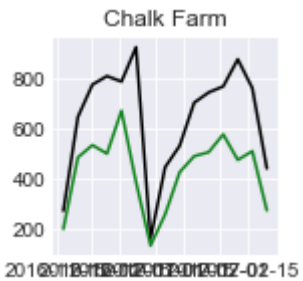

Archway
1500
1000
500
2012-02-15

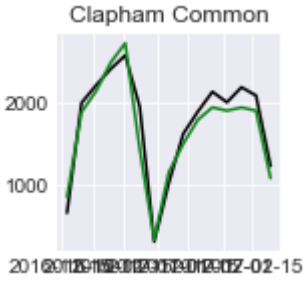

Belsize Park
600
400
200
2012-02-15

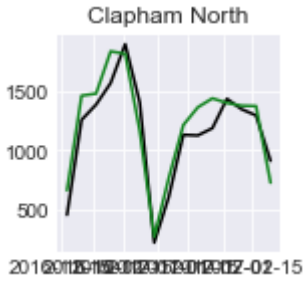

Brent Cross
400
300
200
100
2012-02-15

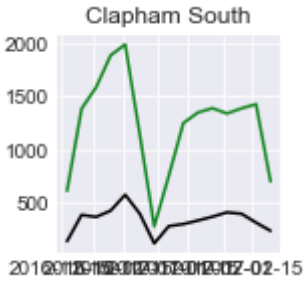

Burnt Oak
600
400
200
2012-02-15

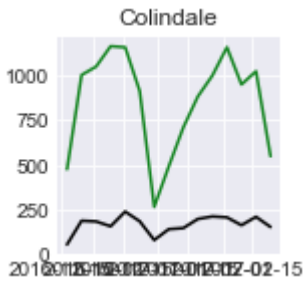

Camden Town
6000
4000
2000
2012-02-15

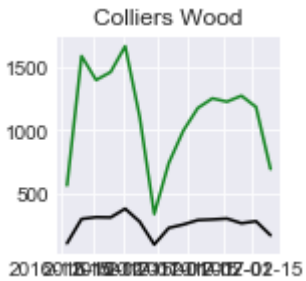

Chalk Farm
800
600
400
200
2012-02-15

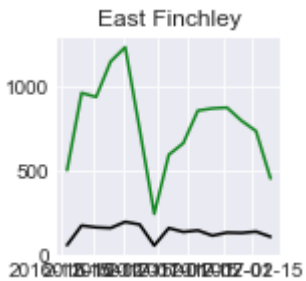

Clapham Common
2000
1000
2016-02-15

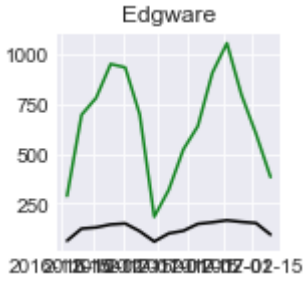

Clapham North
1500
1000
500
2016-02-15

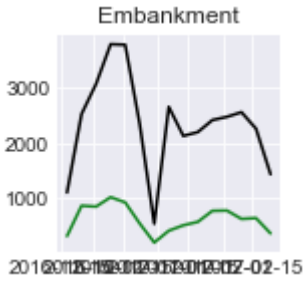

Clapham South
2000
1500
1000
500
2016-02-15

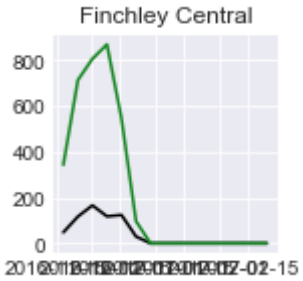

Colindale
1000
750
500
250
0
2016-02-15

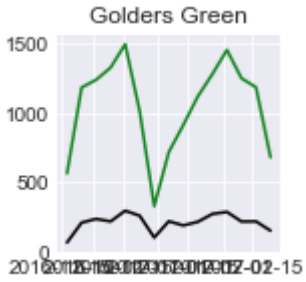

Colliers Wood
1500
1000
500
2016-02-15

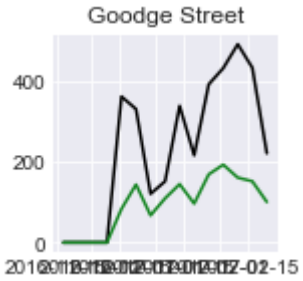

East Finchley
1000
500
0
2016-02-15

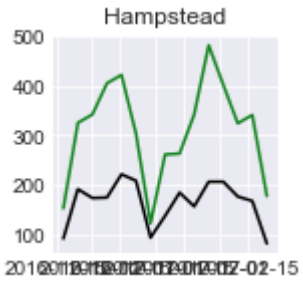

Edgware
1000
750
500
250

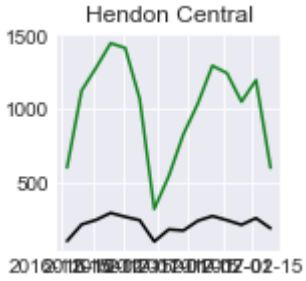

Embankment
3000
2000
1000

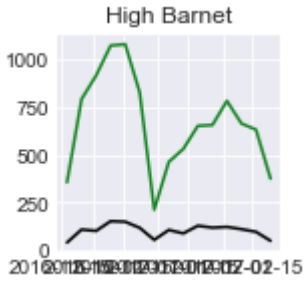

Finchley Central
800
600
400
200
0

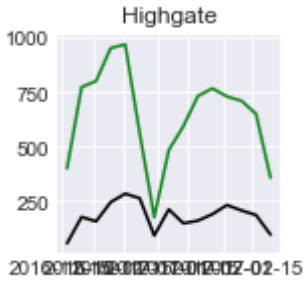

Golders Green
1500
1000
500
0

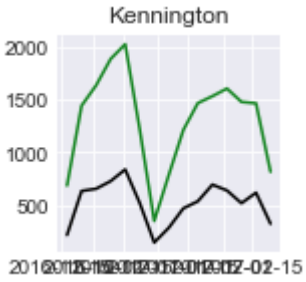

Goodge Street
400
200
0

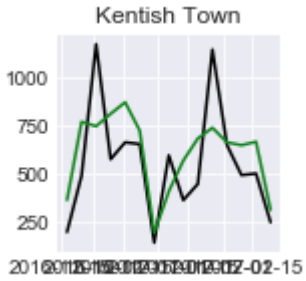

Hampstead
500
400
300
200
100

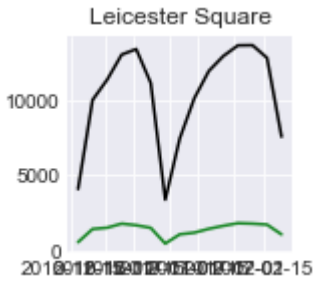

Hendon Central
1500
1000
500

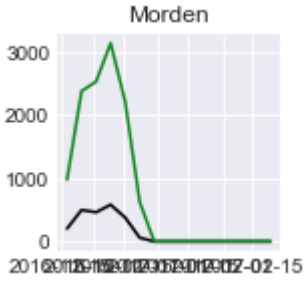

High Barnet
1000
750
500
250
0

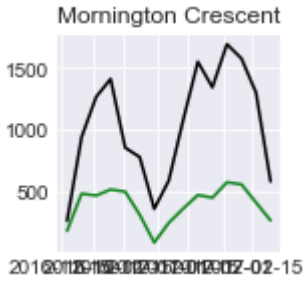

Highgate
1000
750
500
250

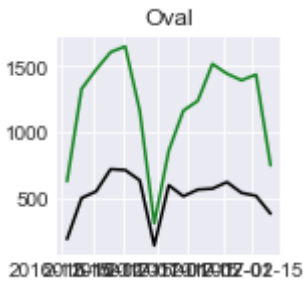

Kennington
2000
1500
1000
500

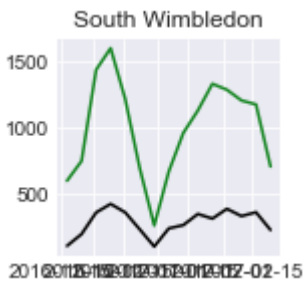

Kentish Town
1000
750
500
250

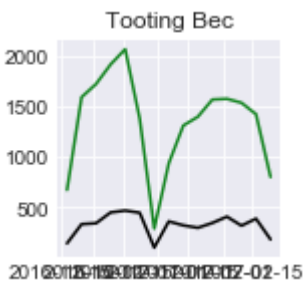

Leicester Square
10000
5000
0

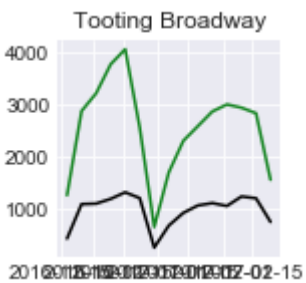

Morden
3000
2000
1000
0

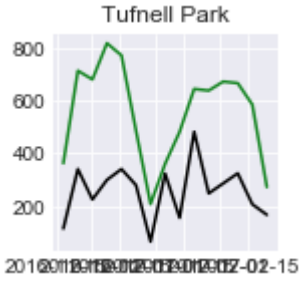

Mornington Crescent
1500
1000
500

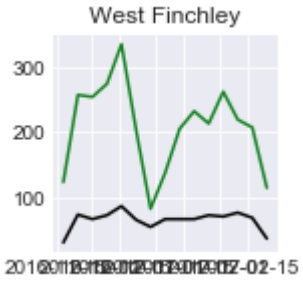

Oval
1500
1000
500


South Wimbledon
1500
1000
500


Tooting Bec
2000
1500
1000
500


Tooting Broadway
4000
3000
2000
1000





Piccadilly Line

In [60]:
```
EN_P = (full.loc[full.Lines == 'Piccadilly'].pivot_table(index = 'datetime',
                                                         columns = 'Station',
                                                         values = 'En',
                                                         fill_value = 0))

EN_P = EN_P.resample('W-Fri').sum()
EN_P = EN_P.reset_index().rename_axis(None, axis = 1)
EN_P.head(2)
```

Out[60]:

	datetime	Acton Town	Arnos Grove	Arsenal	Barons Court	Boston Manor	Bounds Green	Caledonian Road	Cockfosters	Covent Garden	...
0	2016-12-16	158	69	60	141	74	103	183	52	916	...
1	2016-12-23	371	143	123	314	142	292	293	90	1212	...

2 rows × 30 columns

In [61]:
```
EX_P = (full.loc[full.Lines == 'Piccadilly'].pivot_table(index = 'datetime',
                                                         columns = 'Station',
                                                         values = 'Ex',
                                                         fill_value = 0))

EX_P = EX_P.resample('W-Fri').sum()
EX_P = EX_P.reset_index().rename_axis(None, axis = 1)
EX_P.head(2)
```

Out[61]:

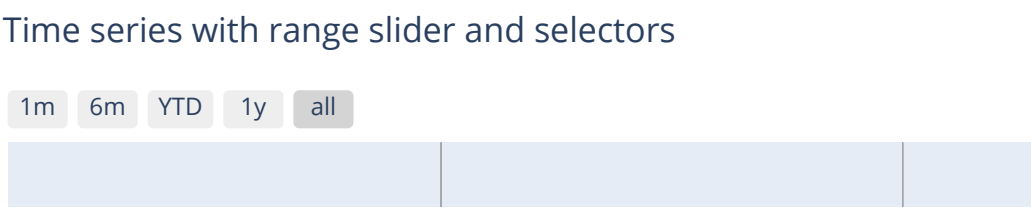
	datetime	Acton Town	Arnos Grove	Arsenal	Barons Court	Boston Manor	Bounds Green	Caledonian Road	Cockfosters	Covent Garden	...

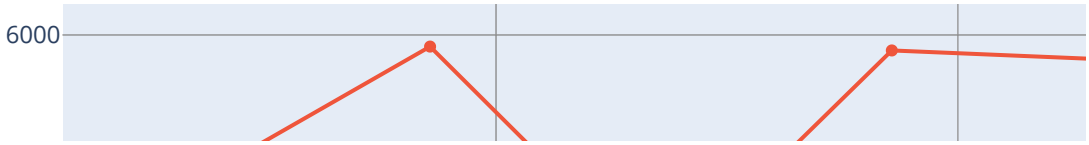
	datetime	Acton Town	Arnos Grove	Arsenal	Barons Court	Boston Manor	Bounds Green	Caledonian Road	Cockfosters	Covent Garden	...
0	2016-12-16	714	473	179	614	217	694	505	235	92	...
1	2016-12-23	1097	662	266	725	274	1006	585	305	122	...

2 rows × 30 columns

In [62]:

```
fig = go.Figure()

for i in EN_P.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EN_P.datetime), y=list(EN_P[i]), name = i))

# Set title
fig.update_layout(
    title_text="Time series with range slider and selectors"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



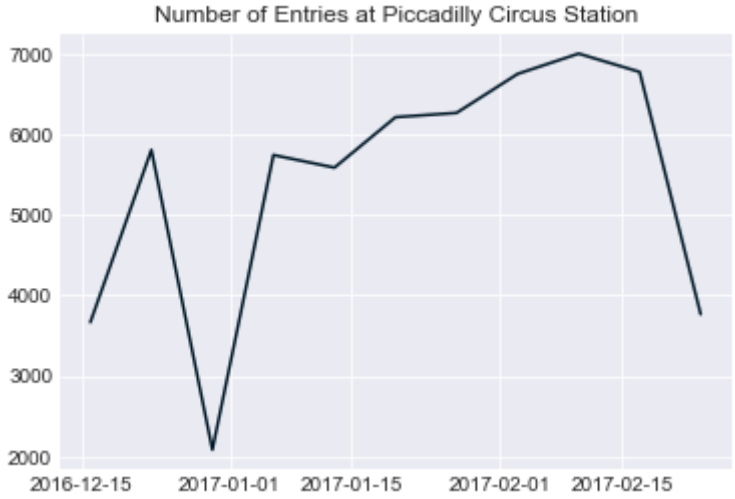


In [63]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EN_P.drop('datetime', axis = 1):
        num += 1
        plt.subplot(4, 7, num)
        plt.plot(EN_P['datetime'], EN_P[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Entering\n Stations on the Piccadilly Lin

        plt.ylim(50, 1500)

        if num in range(22):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

Piccadilly Station is plotted separately, as the number of passengers ranges between 2000 and 7000, well above the average range across the other stations on the Piccadilly Line:

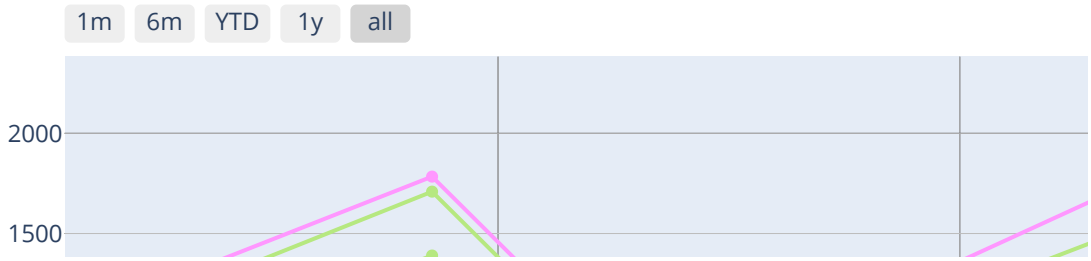
In [64]:
```
plt.plot(EN_P['datetime'], EN_P['Piccadilly Circus'], '#001928')
plt.title("Number of Entries at Piccadilly Circus Station")
```

Out[64]:
```
Text(0.5, 1.0, 'Number of Entries at Piccadilly Circus Station')
```

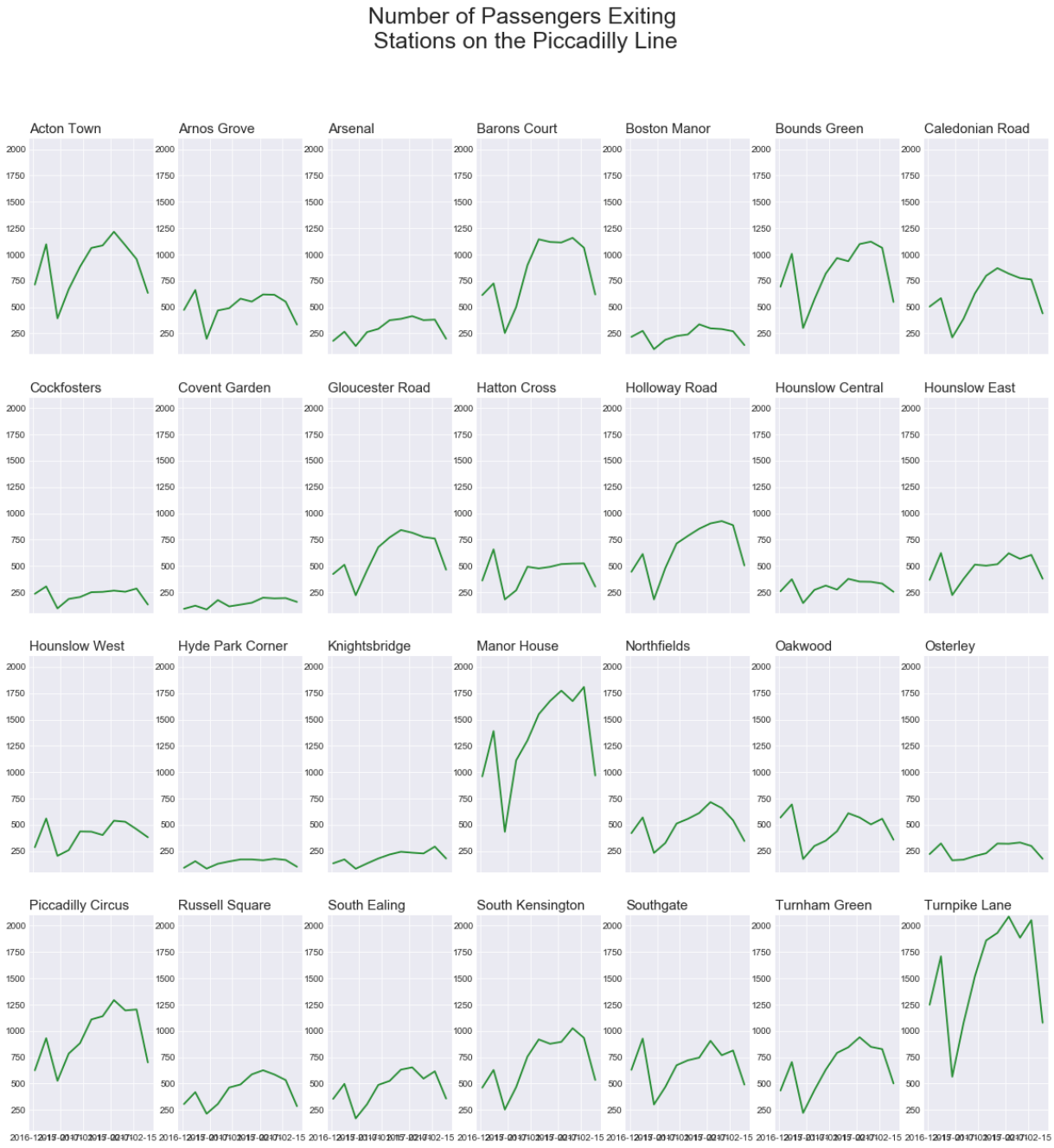


In [65]:
```
fig = go.Figure()

for i in EN_P.drop('datetime', axis = 1):
    fig.add_trace(
        go.Scatter(x=list(EX_P.datetime), y=list(EX_P[i]), name = i))

# Set title
fig.update_layout(
    title_text="Time series with range slider and selectors"
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```

Time series with range slider and selectors

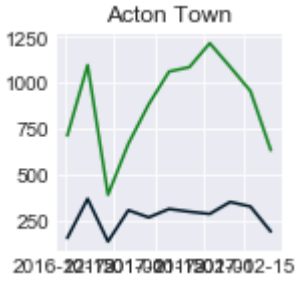


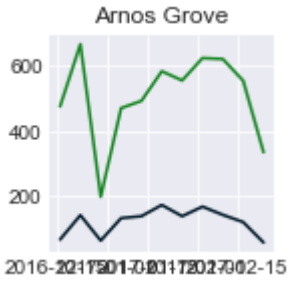
In [66]:
```
plt.style.use('seaborn-darkgrid')
plt.figure(figsize = (20,20))
num = 0

try:
    for i in EX_P.drop('datetime', axis = 1):
        num += 1
        plt.subplot(4, 7, num)
        plt.plot(EX_P['datetime'], EX_P[i], marker = '', linewidth = 1.9, alpha = 0.
        plt.title(i, loc = 'left', fontsize = 15)
        plt.suptitle("Number of Passengers Exiting\n Stations on the Piccadilly Line

        plt.ylim(50, 2100)

        if num in range(22):
            plt.tick_params(axis = 'x', labelbottom = False)

except ValueError:
    print(" ")
```

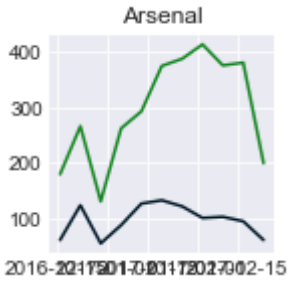


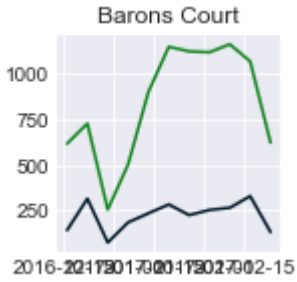
In [67]:
```
for i in EN_P.drop('datetime', axis = 1):
    plt.figure(figsize=(2,2))
    plt.plot(EN_P['datetime'], EN_P[i], color = '#001928') #ENTRIES
    plt.plot(EX_P['datetime'], EX_P[i], '#158522') #EXITS

    plt.title(i)
plt.show()
```

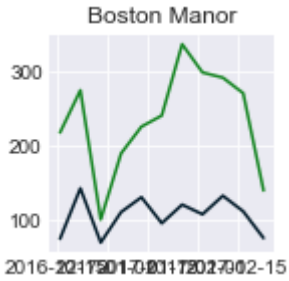
```
C:\Users\Zaana\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning:

More than 20 figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume to
o much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
```

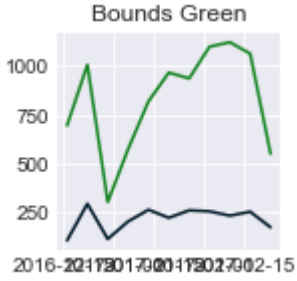

Acton Town
1250
1000
750
500
250

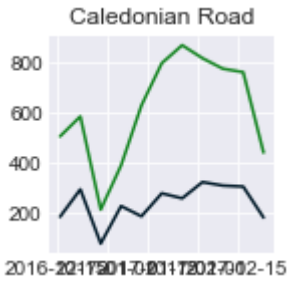

Arnos Grove
600
400
200

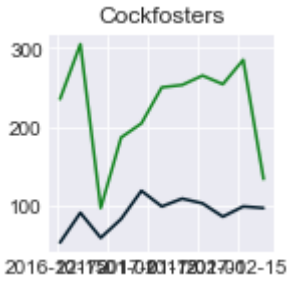

Arsenal
400
300
200
100

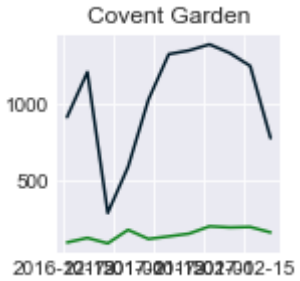

Barons Court
1000
750
500
250

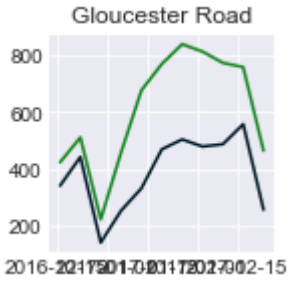

Boston Manor
300
200
100

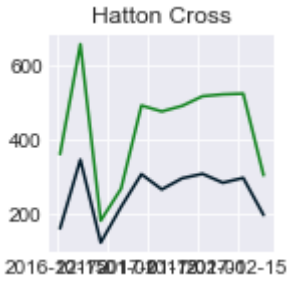

Bounds Green
1000
750
500
250

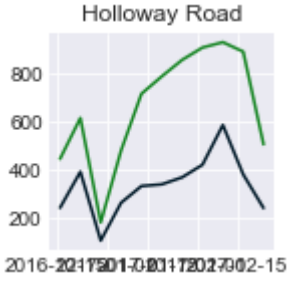

Caledonian Road
800
600
400
200

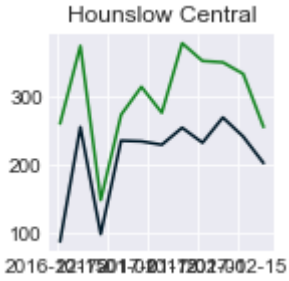

Cockfosters
300
200
100

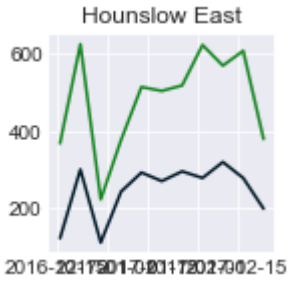

Covent Garden
1000
500

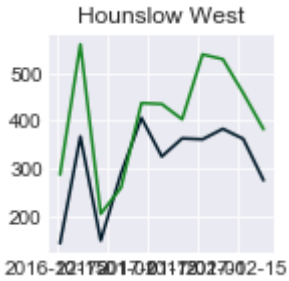

Gloucester Road
800
600
400
200

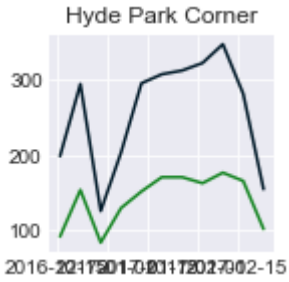

Hatton Cross
600
400
200

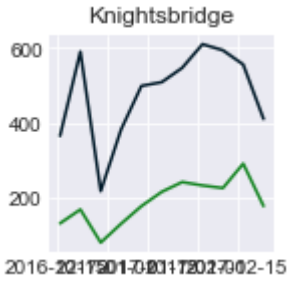

Holloway Road
800
600
400
200

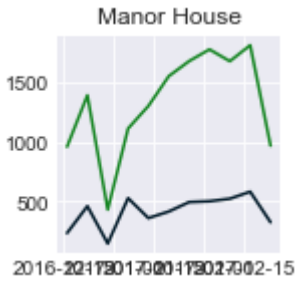

Hounslow Central
300
200
100

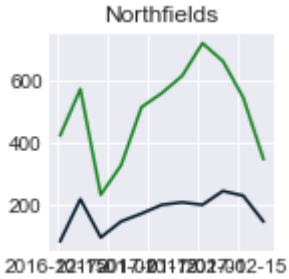

Hounslow East
600
400
200

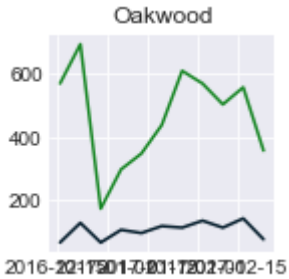

Hounslow West
500
400
300
200

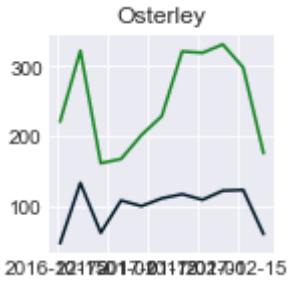

Hyde Park Corner
300
200
100

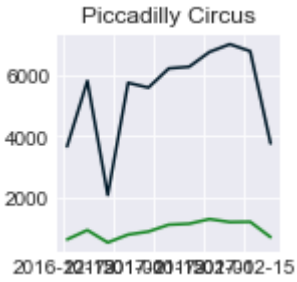

Knightsbridge
600
400
200

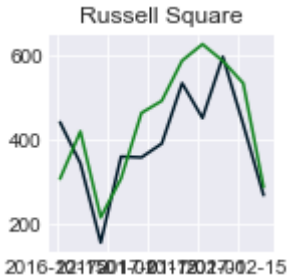

Manor House
1500
1000
500

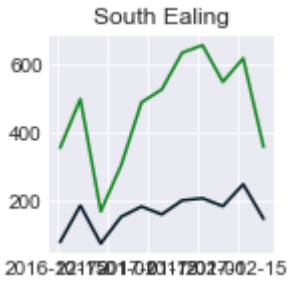

Northfields
600
400
200

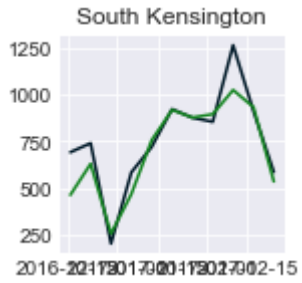

Oakwood
600
400
200

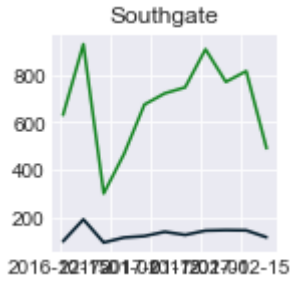

Osterley
300
200
100

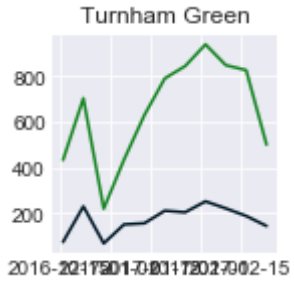

Piccadilly Circus
6000
4000
2000

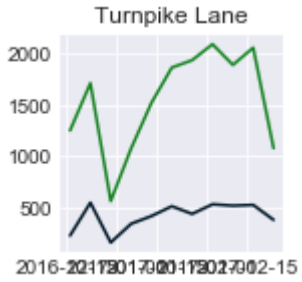

Russell Square
600
400
200

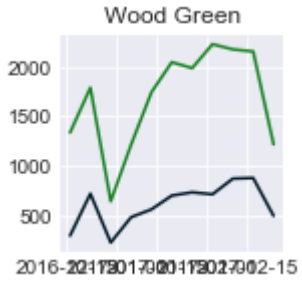

South Ealing
600
400
200











When visualising the passenger data for each station based on line according to entry and exit, the most common trend shown is that stations situated within Central London and Zone 1 have the highest popularity for station entry. This can be credited to the accessbility of the stations for late-night commuters, as it provides further convenience when compared to taxi or bus services, in terms of cost and travel time respectively.

Stations in Zone 2 and 3 have fairly high popularity in terms of the numbers of passenger exits. This may be due to the combination of commuters travelling from Zone 1 and out of Central London, alongside passengers who require transport links to train stations for overnight travel, for example, in the case of Seven Sisters Station as previously mentioned.

To better understand the prevalence of Night Tube travel based on location, travel data can now be analysed when grouped according to congestion zones rather than individual stations.

Grouping Data According to Congestion Zones

In [68]:
```
full.Zone.value_counts()
```

Out[68]:
```
3    12110
1    11198
2    10690
4    10478
5     2699
6      806
Name: Zone, dtype: int64
```

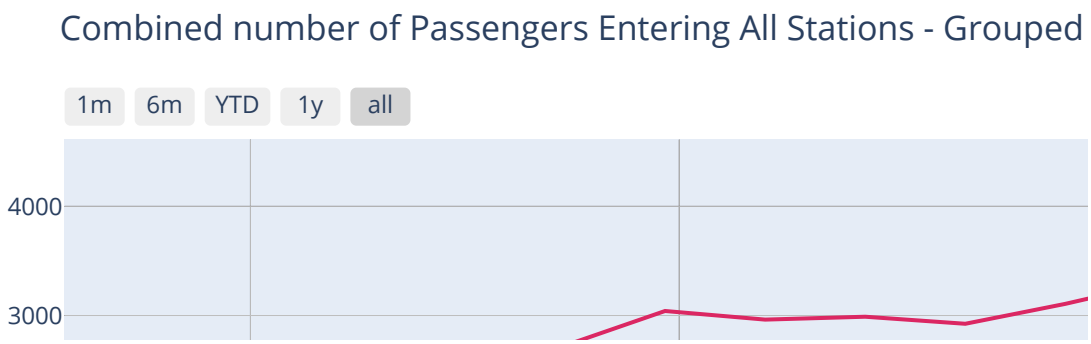
In [70]:
```
Zone_en = (full.pivot_table(index = 'datetime',
                            columns = 'Zone',
                            values = 'En',
                            fill_value = 0))

Zone_en = Zone_en.resample('W-Fri').sum()
Zone_en = Zone_en.reset_index().rename_axis(None, axis = 1)

Zone_en.head(6)
```

Out[70]:

	datetime	1	2	3	4	5	6
0	2016-08-19	1183.153846	335.333333	287.888889	85.363636	38.0	40.0
1	2016-08-26	2472.153846	801.000000	744.611111	203.727273	68.0	124.0
2	2016-09-02	2376.846154	815.333333	800.888889	199.454545	73.0	110.0
3	2016-09-09	2520.384615	854.000000	775.777778	215.272727	84.0	143.0
4	2016-09-16	2686.153846	835.500000	815.222222	204.636364	78.0	137.0
5	2016-09-23	2724.307692	940.666667	816.444444	212.727273	91.0	103.0

In [71]:
```
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[1]), name = "Zone 1",
               line=dict(color='#db2763')))

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[2]), name = "Zone 2",
               line=dict(color='#12eaea')))

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[3]), name = "Zone 3",
               line=dict(color='#b0db43')))

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[4]), name = "Zone 4",
               line=dict(color='#f2a2d4')))

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[5]), name = "Zone 5",
               line=dict(color='#d7bba8')))

fig.add_trace(
    go.Scatter(x=list(Zone_en.datetime), y=list(Zone_en[6]), name = "Zone 6",
               line=dict(color='#3e2a35')))

fig.update_layout(
    title_text="Combined number of Passengers Entering All Stations - Grouped By Zon
)
```

```
# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```

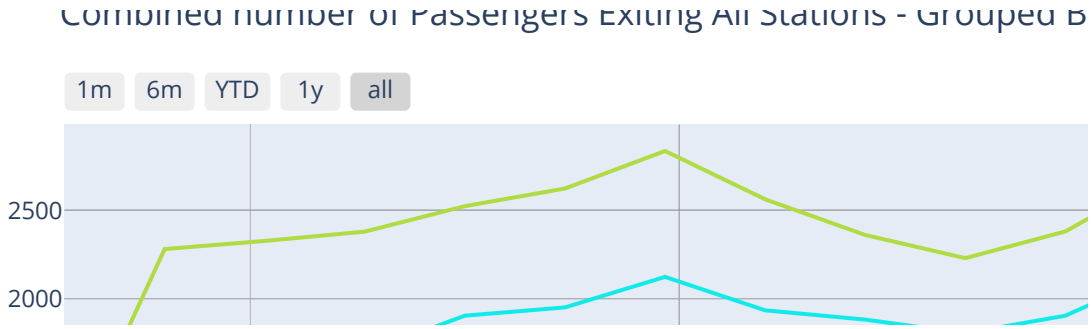


In [74]:
```
Zones = (full.pivot_table(index = 'datetime',
                          columns = 'Zone',
                          values = 'Ex',
                          fill_value = 0))

Zones = Zones.resample('W-Fri').sum()
Zones = Zones.reset_index().rename_axis(None, axis = 1)

Zones
```

Out[74]:

	datetime	1	2	3	4	5	6
0	2016-08-19	482.384615	799.333333	1051.888889	294.727273	160.000000	263.0
1	2016-08-26	1037.307692	1731.833333	2280.402778	590.909091	289.000000	653.0
2	2016-09-02	982.384615	1608.333333	2326.444444	588.000000	262.000000	715.0
3	2016-09-09	971.692308	1693.500000	2379.222222	614.109091	284.000000	852.0
4	2016-09-16	1088.615385	1903.333333	2523.444444	690.909091	308.000000	982.0
5	2016-09-23	1115.384615	1950.166667	2623.222222	686.272727	275.000000	839.0
6	2016-09-30	1238.307692	2123.000000	2835.666667	731.272727	357.000000	1036.0
7	2016-10-07	1113.052885	1933.848485	2562.266667	686.733766	398.333333	906.0
8	2016-10-14	1102.375000	1881.000000	2360.066667	773.428571	545.333333	922.0
9	2016-10-21	1027.312500	1804.181818	2229.066667	757.357143	487.333333	694.0
10	2016-10-28	1168.062500	1903.181818	2379.933333	800.642857	541.333333	831.0
11	2016-11-04	1263.375000	2156.000000	2698.266667	820.857143	577.333333	888.0
12	2016-11-11	1043.750000	1796.454545	2222.266667	699.543956	444.666667	944.0
13	2016-11-18	1140.020833	1707.575758	2113.916667	781.264286	558.200000	864.0
14	2016-11-25	1143.166667	1461.750000	1995.059783	795.500000	563.800000	955.0
15	2016-12-02	1207.833333	1520.875000	2193.750000	878.100000	646.400000	1075.0
16	2016-12-09	1418.630719	1715.875000	2418.833333	1002.350000	773.600000	1262.0
17	2016-12-16	1209.141026	1766.979885	2350.629441	971.957692	827.125000	1210.5
18	2016-12-23	722.847692	1073.631773	1568.290323	636.656154	608.000000	814.5
19	2016-12-30	259.336538	329.330049	520.760120	230.298913	166.375000	185.0
20	2017-01-06	513.900641	752.551724	952.292621	378.269598	276.375000	319.5
21	2017-01-13	684.384615	1049.413793	1369.322581	515.541667	397.750000	587.5
22	2017-01-20	881.174967	1197.252874	1526.870014	529.580303	436.529762	595.0
23	2017-01-27	943.560682	1205.731432	1608.620856	508.529270	436.565476	613.0
24	2017-02-03	907.887492	1271.169270	1682.315515	566.206522	507.839286	650.0
25	2017-02-10	871.667692	1285.103448	1686.419355	605.250000	510.375000	696.5
26	2017-02-17	869.703077	1250.620690	1640.516129	609.672101	465.250000	698.0
27	2017-02-24	509.240385	696.793103	941.258065	356.170290	294.750000	367.5

In [75]:
```
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[1]), name = "Zone 1",
               line=dict(color='#db2763')))

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[2]), name = "Zone 2",
               line=dict(color='#12eaea')))

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[3]), name = "Zone 3",
               line=dict(color='#b0db43')))

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[4]), name = "Zone 4",
               line=dict(color='#f2a2d4')))

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[5]), name = "Zone 5",
               line=dict(color='#d7bba8')))

fig.add_trace(
    go.Scatter(x=list(Zones.datetime), y=list(Zones[6]), name = "Zone 6",
               line=dict(color='#3e2a35')))

fig.update_layout(
    title_text="Combined number of Passengers Exiting All Stations - Grouped By Zone
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)

fig.show()
```



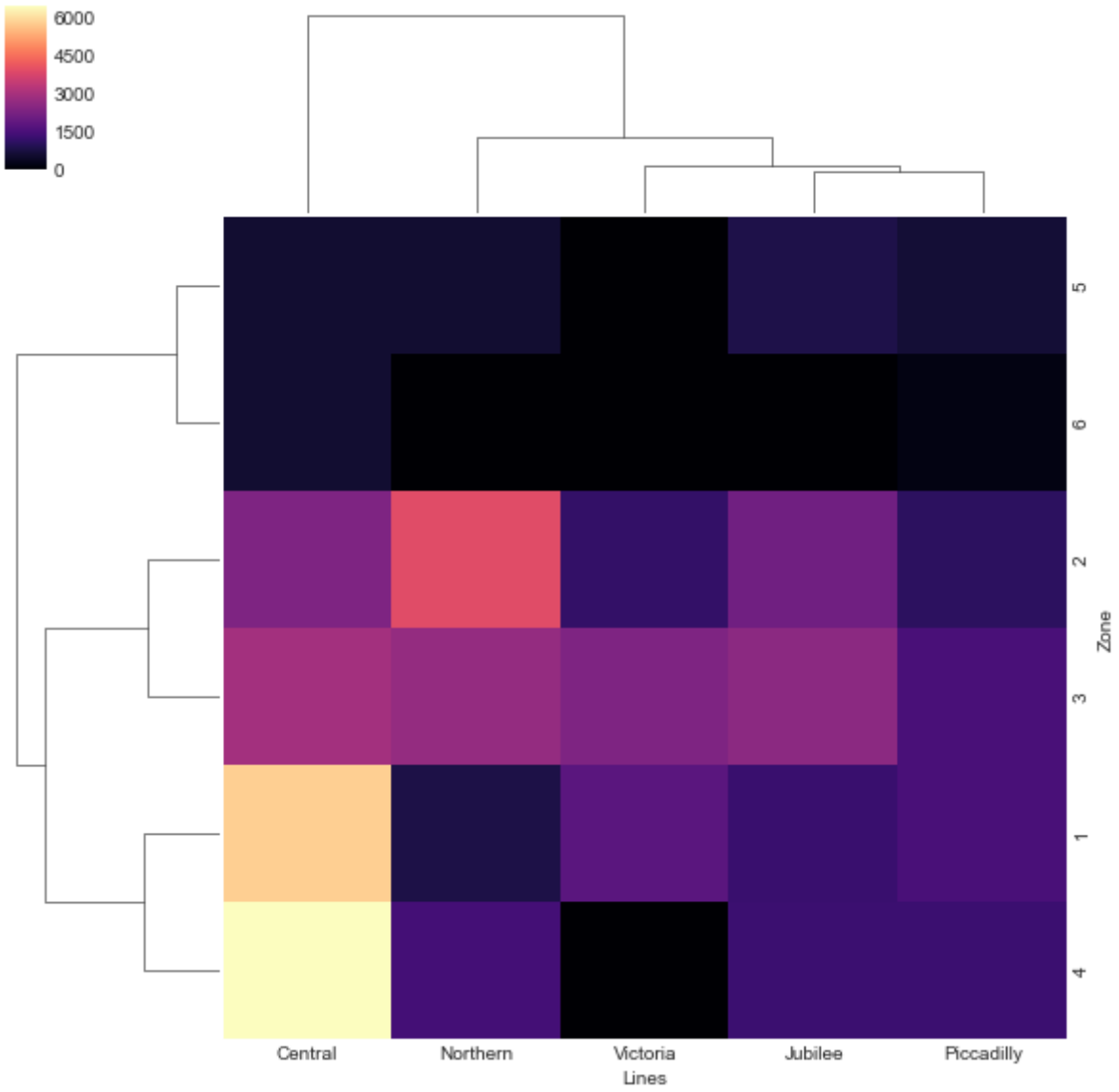
As expected, stations situated within Zone 1 have the highest number of passenger entry, while those situated in Zones 5 and 6 have the lowest. There is a contrast, however, when compared with the popularity based on passenger exits, as Zones 2 and 3 are the most popular, and the number of passengers exiting stations within Zones 5 and 6 are almost double the number of entries.

To understand whether the combined number of passengers has any correlation to tube lines and location, a clustermap is plotted.

In [76]:
```
def zone_popularity(full):

    # Group by Zone and Line
    tube_lines = full.groupby('Zone').Lines.value_counts()
    t = tube_lines.unstack().fillna(0)

    # Sort by lines
    tube_sum=t.sum(axis=0)
    t=t[tube_sum.index]

    # Filter by zone
    zone_sum=t.sum(axis=1)

    # Large number, so let's slice the data.
    return t

t= zone_popularity(full)
```

In [77]:
```
sns.clustermap(t, cmap = "magma")
```

Out[77]:
```
<seaborn.matrix.ClusterGrid at 0x286bce62898>
```

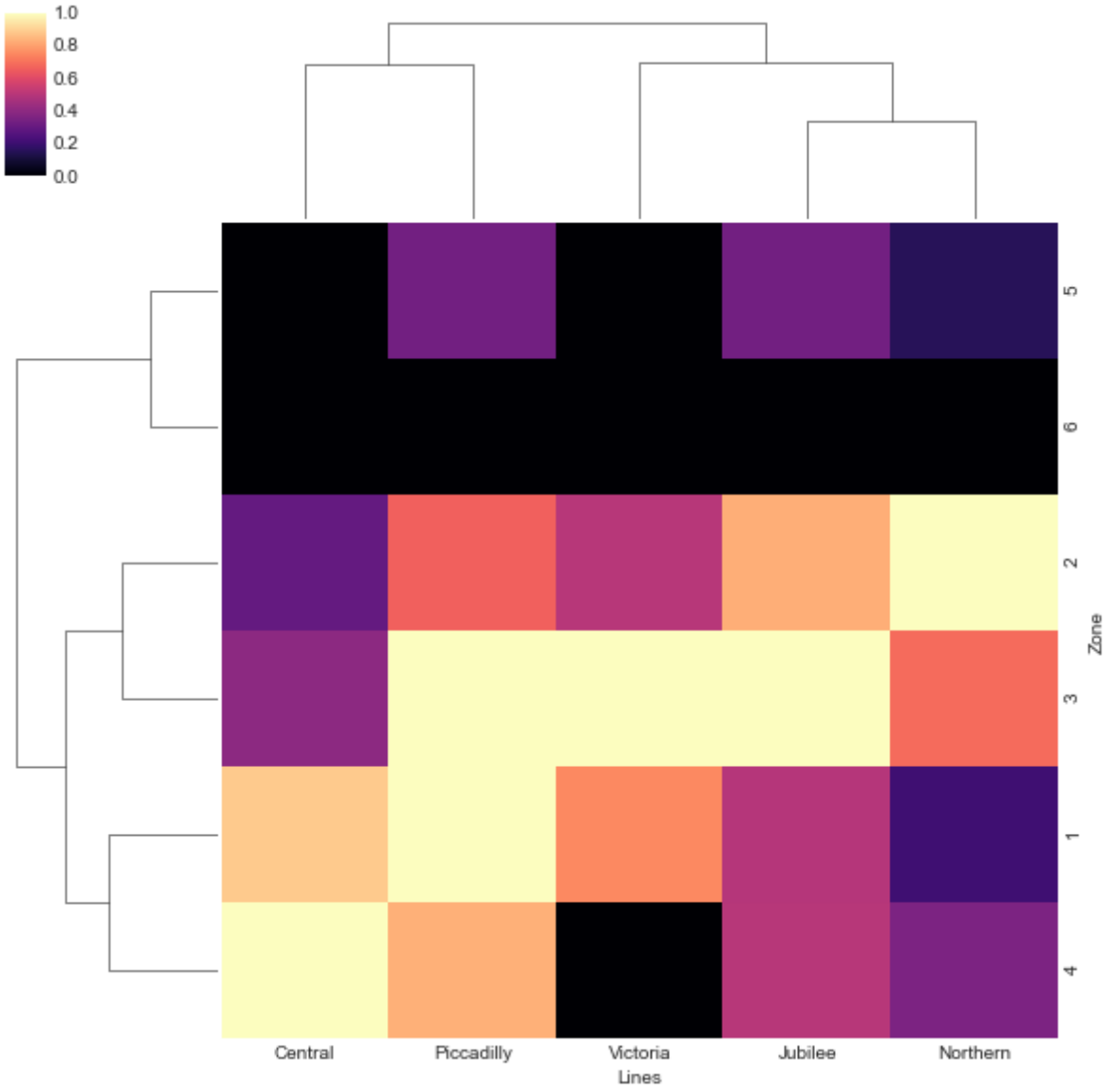


The clustermap above shows that aggregate Central Line use in Zones 1 and 4 is the highest and travel across all lines throughout Zones 3 and 2 follows closely behind. Victoria Line, however, shows low popularity, which is a stark contrast between the data presented according to travel based on Tube Lines.

The contrast shown may be attributed to the extent to which certain lines serve, which can affect the proportion of travel. To counteract this, the clustermap is standardised across Lines and Zones.

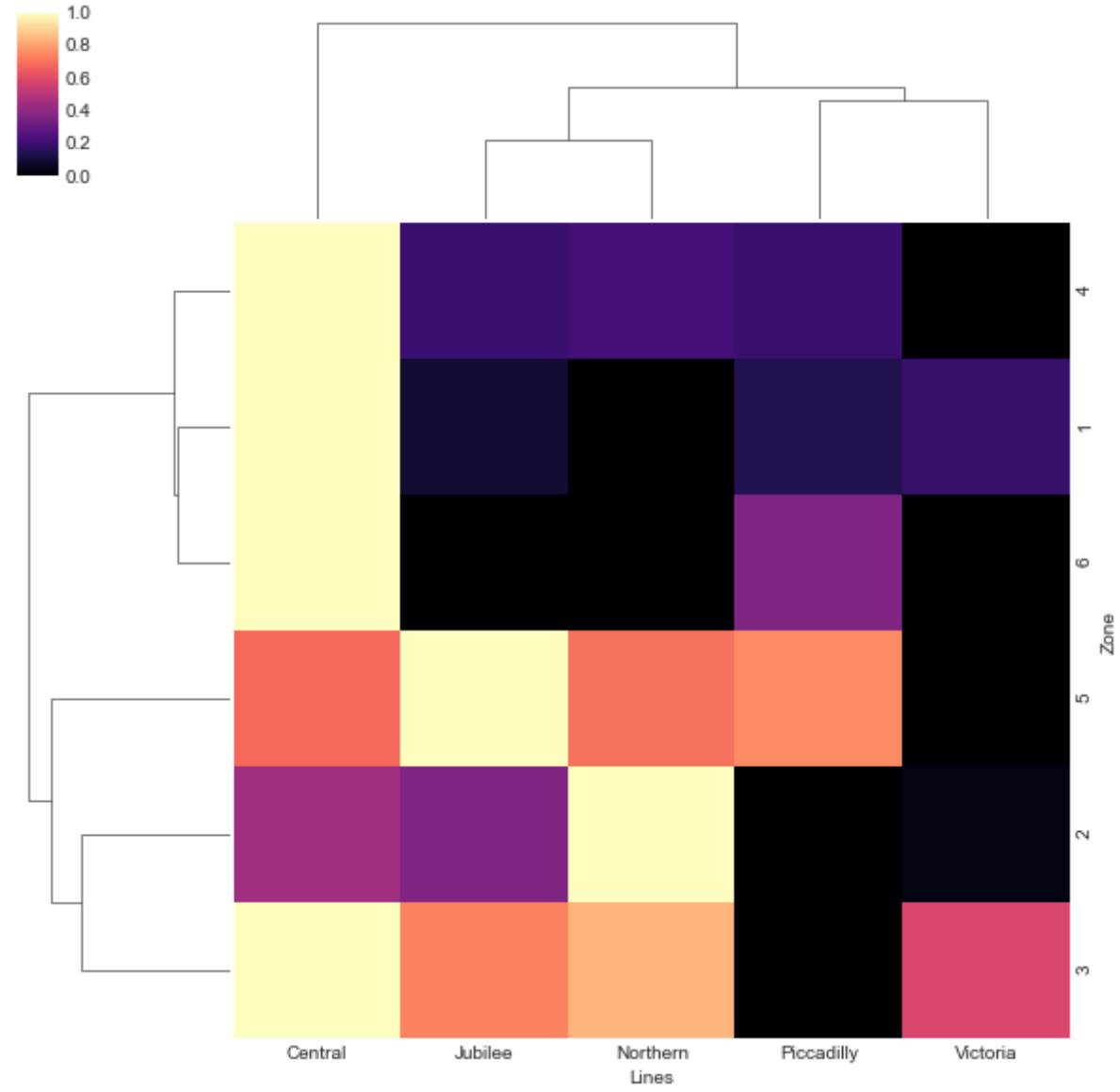
In [78]:
```
sns.clustermap(t, standard_scale = 1, cmap = 'magma')
```

Out[78]:
```
<seaborn.matrix.ClusterGrid at 0x286bbcb51d0>
```



In [79]:
```
sns.clustermap(t, standard_scale = 0, cmap = "magma")
```

Out[79]:
```
<seaborn.matrix.ClusterGrid at 0x286bb9a3a58>
```



Upon standardisation, it is seen that Central Line has the highest popularity across zones, followed by Northern, Jubilee, and lastly, Piccadilly and Victoria. From the clusters, it shows that the proportionality of stations across each tube line affects the overall performance when clustered against geographical region. Additionally, from the clustermaps, it shows that the timeframe for which each line ran service does not have a direct impact on the values when standardised, based on cluster values for Northern Line, for example.

Spatial Visualisation of Data

From the graphs plotted above for each line, the popularity of the night tube increases over time, due to the addition of lines over the inital service period. A screenshot below shows the combined number of passengers entering and exiting stations for the month of August (blank represents 0)

From the screenshot, the most popular stations are situated in central London, with over tens of thousands of passengers accessing the Night Tube in the span of 8 days each month. To visualise the movement of passengers for the entire timeframe, a link to an interactive version of the above map is included below:

In [41]:
```
import webbrowser

tewbs = 'TFLMAP.html'

webbrowser.open(tewbs)
```

Out[41]: True

Conclusions, Limitations and Recommendations

From the datasets analysed above, the largest conclusion which is drawn is that certain stations within Zone 1 are the most popular for passenger entry, regardless of the lines they serve. Alongside this, stations which provide links to other public transport networks outside of Central London, are the most popular for passenger exits, though overall there is a higher number of passenger exits in stations located between Zones 2 and 3.

The presence of Night Tube is highly impactful as it provides accessible and cost-effective transport to areas outside of Central London, which become especially useful for commuters who work late shifts and have to travel long distances between their workplace and residence.

While the data provided is fairly detailed, there are many limitations in the values which had to be accounted for. For example, as many stations provide Night Tube service on multiple lines, the values provided do not specify via which lines passengers travel. In this case, it was assumed that only the first line which ran service was accessed by all commuters entering and exiting the stations. Should these values be separated according to line use, passenger data could be segregated to highlight any possible transfers between stations. One example of this would be for Green Park Station, which serves Victoria, Piccadilly and Jubilee lines, with Victoria being the first to provide Night Tube Service:

In [80]:
```
plt.plot(EN_V['datetime'], EN_V['Green Park'], color = '#0098d8')
plt.plot(EX_V['datetime'], EX_V['Green Park'], color = '#827878')
plt.title("Number of Passengers Entering and Exiting Green Park Station")
bloo = mpatches.Patch(color='#0098d8', label='Entries')
grey = mpatches.Patch(color='#827878', label = 'Exits')
plt.legend(handles=[bloo, grey])

plt.show()
```

As shown above, there is a large peak in values between October and December 2016, around the same time both Jubilee and Piccadilly Lines began service respectively. The influx in

passenger entry could therefore be due to passenger use throughout all 3 lines, however these values are not categorised to determine which line is most frequented.

Another limitation which was initially mentioned was the separation from this data from the RODS survey. The RODS survey, collected over a year, indicates a generalised number of passengers at a station at a given timeframe between 7am and 1:45am the following day, as well as the direction and lines accessed. This can be seen in a snippet below:

In [82]:
```
rods = pd.read_csv("RODS_Snip.csv")
rods.head(6)
```

Out[82]:

	Station	Line	Direction	- 7am	7am-10am	10am-4pm	4pm-7pm	7pm-10pm	10pm+	Total	0500-0515	0515-0530
0	Acton Town	District	E	299	1510	464	386	190	113	2962	10	12
1	Acton Town	District	W	11	278	573	619	236	95	1813	0	0
2	Acton Town	Piccadilly	E	531	2872	2188	1457	494	94	7637	17	22
3	Acton Town	Piccadilly	W	551	2461	2506	3606	1611	797	11532	26	27
4	Aldgate	Circle	I	152	295	676	823	99	31	2076	8	6
5	Aldgate	Circle	O	19	310	815	486	218	10	1858	0	0

From this data, visualisations highlighting passenger travel can be created, specifying which routes and areas are the most popular. Based on the Night Tube analyses, the general movement of passengers is radial, with the most prevalent travel patterns being grouped mainly as travel from Central London / Zone 1, out to Zones 2 and 3.

It would therefore be interesting to visualise the changes in movement during the week, when compared to nightly weekend travel. Certain lines, such as Central and Northern, are considered notoriously popular during a regular workweek, however for Night Tube service, their usage is significantly low.

References

Holtz, Y., 2017. #125 *Small multiples for line chart.* The Python Graph Gallery. URL https://python-graph-gallery.com/125-small-multiples-for-line-chart/ (accessed 5.2.20).

FOI request detail - Transport for London [WWW Document], n.d. URL https://tfl.gov.uk/corporate/transparency/freedom-of-information/foi-request-detail?referenceId=FOI-1215-1718 (accessed 5.1.20).

Coordinates of London Underground stations (including Elizabeth Line stations) - a Freedom of Information request to Transport for London - WhatDoTheyKnow [WWW Document], n.d. URL https://www.whatdotheyknow.com/request/coordinates_of_london_undergroun#incoming-1238210 (accessed 4.30.20).

Transport for London - *TfL Colour Standards*, 2016. URL http://content.tfl.gov.uk/tfl-colour-standards-issue04.pdf (accessed 5.3.20)