Analysis of British Road Network Use

The assignment will focus on data aggregation and grouping using Pandas library, followed by the creation of plots using Bokeh library.

KATE expects your code to define variables with specific names that correspond to certain things we are interested in.

KATE will run your notebook from top to bottom and check the latest value of those variables, so make sure you don't overwrite them.

- Remember to uncomment the line assigning the variable to your answer and don't change the variable or function names.
- Use copies of the original or previous DataFrames to make sure you do not overwrite them by mistake.

You will find instructions below about how to define each variable.

Once you're happy with your code, upload your notebook to KATE to check your feedback.

Importing Libraries

First of all, we will import `pandas` and `pandas_bokeh` and set them up:

In [1]:
```
import pandas as pd
import pandas_bokeh
from bokeh.plotting import show

from bokeh.plotting import output_notebook
output_notebook()
pd.set_option('plotting.backend', 'pandas_bokeh')

import warnings
warnings.filterwarnings('ignore')
```

(https://bokeh.org/)
BokehJS 2.3.3 successfully loaded.

About the Dataset

You will be analysing a dataset from the UK Department for Transport (https://data.gov.uk/dataset/208c0e7b-353f-4e2d-8b7a-1a7118467acc/gb-road-traffic-counts) on the road network use by different types of vehicles from 1993-2018. Further information on the fields in the dataset can be found in this guide (https://storage.googleapis.com/dft-statistics/road-traffic/all-traffic-data-metadata.pdf), although this isn't necessary for completion of the assignment.

Importing the Dataset

Use `.read_csv()` to get our dataset `data/region_traffic.csv` and assign to DataFrame `df`:

In [2]:
```
df = pd.read_csv('data/region_traffic.csv')
```

Running `df.head()`, `df.tail()` and `df.info()` will show us how the DataFrame is structured:

In [3]:
```
#df.head()
df.head()
```

Out[3]:

	year	region_id	name	ons_code	road_category_id	total_link_length_km	total_link_length_miles	pedal_cycles	two_wh
0	1993	1	South West	E12000009	1	301.339	187.24	0.000000e+00	
1	1993	1	South West	E12000009	3	993.586	617.39	3.579808e+06	
2	1993	1	South West	E12000009	4	3874.924	2407.77	3.866325e+07	
3	1993	1	South West	E12000009	5	3290.200	2044.44	2.435899e+07	
4	1993	1	South West	E12000009	6	40291.500	25035.98	1.613508e+08	

In [4]:
```
#df.tail()
df.tail()
```

Out[4]:

	year	region_id	name	ons_code	road_category_id	total_link_length_km	total_link_length_miles	pedal_cycles	two
1574	2018	11	North East	E12000001	2	2.80	1.74	0.000000e+00	
1575	2018	11	North East	E12000001	3	350.40	217.73	2.216186e+05	
1576	2018	11	North East	E12000001	4	1453.90	903.41	1.431639e+07	
1577	2018	11	North East	E12000001	5	1346.15	836.46	5.017630e+06	
1578	2018	11	North East	E12000001	6	13156.23	8174.90	7.034451e+07	

In [5]:
```
#df.info()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1579 entries, 0 to 1578
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   year                        1579 non-null   int64  
 1   region_id                   1579 non-null   int64  
 2   name                        1579 non-null   object 
 3   ons_code                    1579 non-null   object 
 4   road_category_id            1579 non-null   int64  
 5   total_link_length_km        1579 non-null   float64
 6   total_link_length_miles     1579 non-null   float64
 7   pedal_cycles                1579 non-null   float64
 8   two_wheeled_motor_vehicles  1579 non-null   float64
 9   cars_and_taxis              1579 non-null   float64
 10  buses_and_coaches           1579 non-null   float64
 11  lgvs                        1579 non-null   float64
 12  all_hgvs                    1579 non-null   float64
 13  all_motor_vehicles          1579 non-null   float64
dtypes: float64(9), int64(3), object(2)
memory usage: 172.8+ KB
```

Exploratory Analysis

Q1. Use `.groupby()` to create a DataFrame called `year` which groups `df` by `'year'` and contains the columns `['pedal_cycles', 'cars_and_taxis', 'all_hgvs']`, with the `.sum()` of each of these for each year:

See below code syntax for some guidance:

```
year = DataFrame_Name.groupby(by=...)[list_of_cols].sum()
```

In [7]:
```
#add your code below
#year =

year = df.groupby("year")['pedal_cycles', 'cars_and_taxis', 'all_hgvs'].sum()
year
```

Out[7]:

year	pedal_cycles	cars_and_taxis	all_hgvs
1993	2.489981e+09	2.100849e+11	1.507144e+10
1994	2.495693e+09	2.143886e+11	1.539442e+10
1995	2.573601e+09	2.181758e+11	1.581009e+10
1996	2.531690e+09	2.236457e+11	1.630137e+10
1997	2.536137e+09	2.272964e+11	1.668684e+10
1998	2.456836e+09	2.302792e+11	1.723609e+10
1999	2.534734e+09	2.345330e+11	1.747849e+10
2000	2.574585e+09	2.336574e+11	1.753572e+10
2001	2.608860e+09	2.368867e+11	1.742001e+10
2002	2.707001e+09	2.426824e+11	1.757117e+10
2003	2.755144e+09	2.423140e+11	1.765987e+10
2004	2.558371e+09	2.449631e+11	1.819330e+10
2005	2.680653e+09	2.439972e+11	1.798489e+10
2006	2.797313e+09	2.469057e+11	1.804688e+10
2007	2.550980e+09	2.472724e+11	1.818077e+10
2008	2.839879e+09	2.454109e+11	1.777312e+10
2009	2.966308e+09	2.447908e+11	1.631027e+10
2010	3.003657e+09	2.397883e+11	1.636518e+10
2011	3.070393e+09	2.406898e+11	1.592679e+10
2012	3.108492e+09	2.402709e+11	1.553736e+10
2013	3.128983e+09	2.399580e+11	1.567433e+10
2014	3.457418e+09	2.449598e+11	1.607863e+10
2015	3.248147e+09	2.476949e+11	1.667027e+10
2016	3.170050e+09	2.516441e+11	1.682620e+10
2017	3.269204e+09	2.543942e+11	1.702446e+10
2018	3.329109e+09	2.550127e+11	1.708263e+10

Q2. We want to look at the change over time of each of these forms of transport relative to the earliest values (year 1993).

To do so, we will create an *index*. An index allows us to inspect the growth over time of a variable relative to some starting value (known as the *base*). By convention, this starting value is `100.0` . If the value of our variable doubles in some future time period, then the value of our index in that future time period would be `200.0` .

- create a new DataFrame called `year_index` as a `.copy()` of `year`
- for the index, select **1993** as the **base year**. This means that all values for 1993 should be equal to `100.0` . All subsequent years should be relative to that

See below code syntax for some guidance:

```
base = year_index.iloc[0]
year_index = (year_index/base)*100
```

Below snippet showcases how the data in `year_index` DataFrame should look like after the changes, you do not need to apply any rounding.

```
      pedal_cycles    cars_and_taxis    all_hgvs
year
1993    100.000000    100.000000    100.000000
1994    100.229413    102.048581    102.143030
1995    103.358260    103.851256    104.900983
```

In [9]:

```
#add your code below
#year_index = year.copy()
#base = year_index.iloc[0]
#year_index = (year_index/base)*100
#year_index.head()

year_index = year.copy()
base = year_index.iloc[0]
year_index = (year_index/base)*100
year_index.head()
```

Out[9]:

year	pedal_cycles	cars_and_taxis	all_hgvs
1993	100.000000	100.000000	100.000000
1994	100.229413	102.048581	102.143030
1995	103.358260	103.851256	104.900983
1996	101.675079	106.454909	108.160667
1997	101.853694	108.192646	110.718300

Q3. Having already imported and set up `pandas_bokeh` at the start of the notebook, we can now create a Bokeh plot of `year_index` DataFrame simply using the `.plot()` method and saving to variable `yi_fig`.

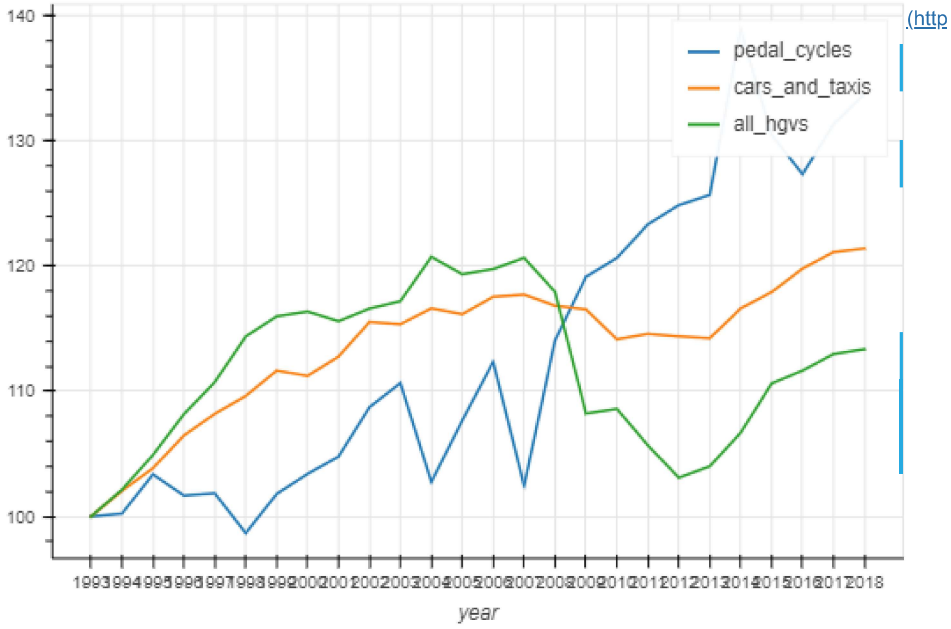
See below code syntax for some guidance:

```
yi_fig = DataFrame_Name.plot()
```

Do not pass any additional arguments to `.plot()`

In [10]:

```
#add your code below
#yi_fig =

yi_fig = year_index.plot()
```



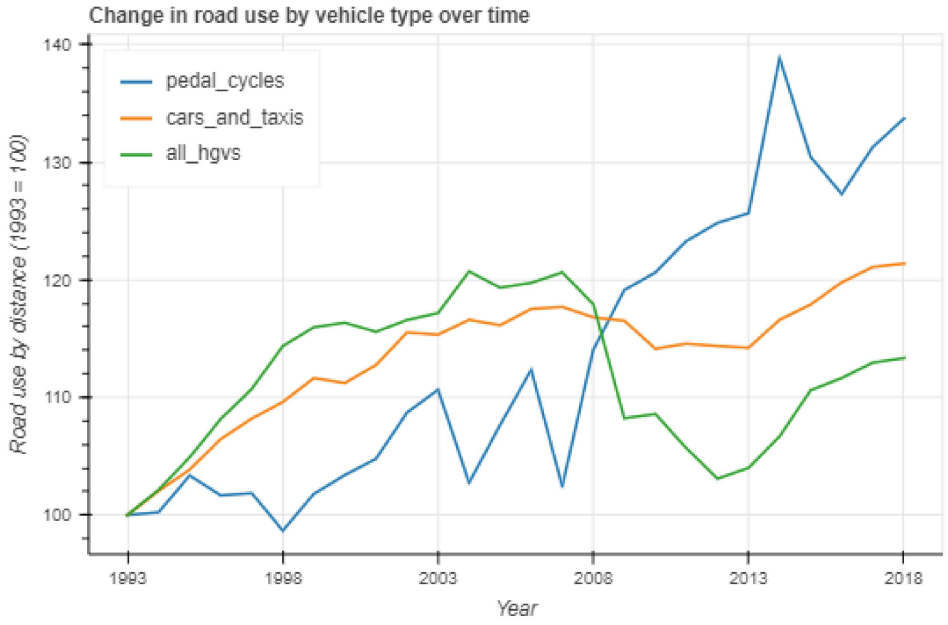
Q4. Now that you have created your `yi_fig` variable using just `.plot()` method, make the following changes to the specified properties of `yi_fig`:

- change the `text` of the `title` to 'Change in road use by vehicle type over time'
- change the `axis_label` of the `yaxis` to 'Road use by distance (1993 = 100)'
- change the `axis_label` of the `xaxis` to 'Year'

- remove the toolbar by changing the `.toolbar_location` attribute to `None`
- change the legend location using `legend.location` attribute to `'top_left'`
- change the `ticker` of the `xaxis` to use the values `[1993, 1998, 2003, 2008, 2013, 2018]`

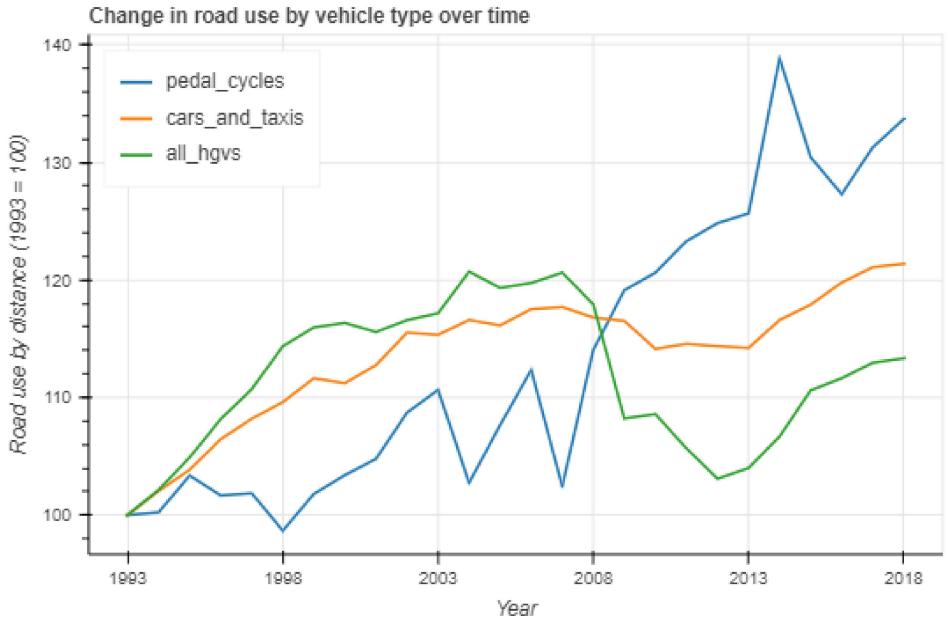
In [15]:

```
#add your code below
#yi_fig.title.text = ...
#yi_fig.yaxis.axis_label = ...
#yi_fig.xaxis.axis_label = ...
#yi_fig.toolbar_location = ...
#yi_fig.legend.location = ...
#yi_fig.xaxis.ticker = ...

yi_fig.title.text = 'Change in road use by vehicle type over time'
yi_fig.yaxis.axis_label = 'Road use by distance (1993 = 100)'
yi_fig.xaxis.axis_label = "Year"
yi_fig.toolbar_location = None
yi_fig.legend.location = "top_left"
xaxis_ticker = [1993, 1998, 2003, 2008, 2013, 2018]
yi_fig.xaxis.ticker =  xaxis_ticker


show(yi_fig)
```



Run the cell below to see that your changes have been implemented as expected:

In [16]:
```
#show(yi_fig)
show(yi_fig)
```



Q5. Create a DataFrame called `green_2018` which:

- uses only the data from `df` for 2018
- groups this 2018 data by `name`
- contains the columns `['pedal_cycles', 'buses_and_coaches']` which have the `.sum()` for each group
- is sorted in *descending* order by the values for `pedal_cycles`
- divide all of the values in the resulting DataFrame by 1000000

See below code syntax for some guidance:

```
DataFrame_Name.groupby(by=...)[list_of_cols].sum().sort_values(by=..., ascending=False)
```

In [47]:
```
#add your code below
#green_2018 =

green_2018 = df[df["year"]==2018].groupby("name")['pedal_cycles', 'buses_and_coaches'].sum().sort_v
green_2018 /= 1000000
green_2018
```

Out[47]:

name	pedal_cycles	buses_and_coaches
South East	556.344401	269.744934
East of England	455.848666	203.142747
London	444.469852	305.159744
South West	357.875642	207.614416
North West	326.663412	185.056717
Yorkshire and The Humber	325.296072	185.086552
East Midlands	246.959834	160.819063
West Midlands	218.618679	192.800382
Scotland	194.348653	316.558012
Wales	112.783546	126.086270
North East	89.900157	145.120040

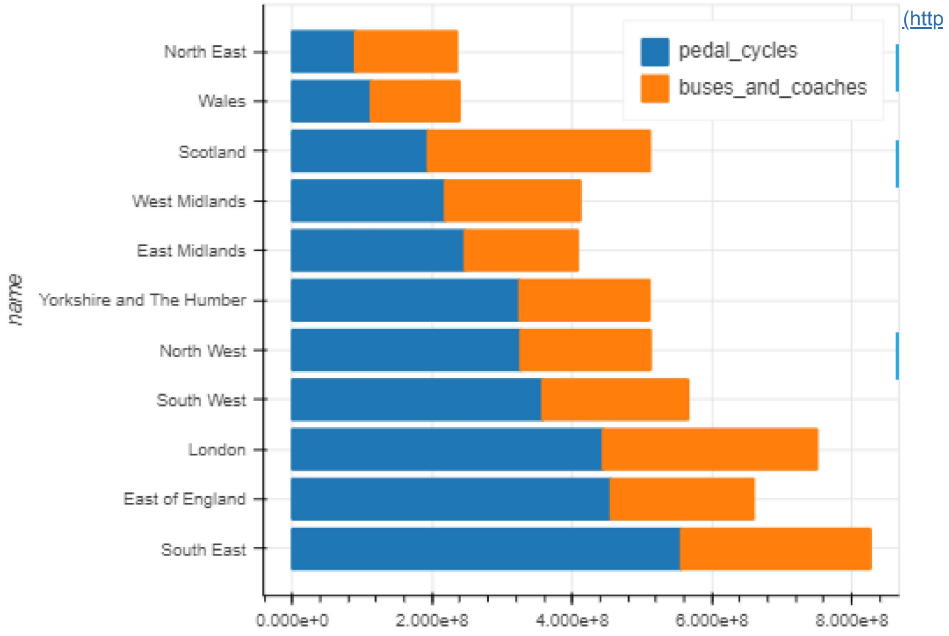
Q6. Use the `.plot()` method to create a *horizontal, stacked* bar chart from the `green_2018` DataFrame, assigning it to `green_bar` variable:

See below code syntax for some guidance:

```
green_bar = DataFrame_Name.plot(stacked=True, kind='barh')
```

- you may find the documentation (https://patrikhlobil.github.io/Pandas-Bokeh/#barplot) useful

In [27]:
```
#add your code below
#green_bar =

green_bar = green_2018.plot(kind="barh", stacked=True)
```



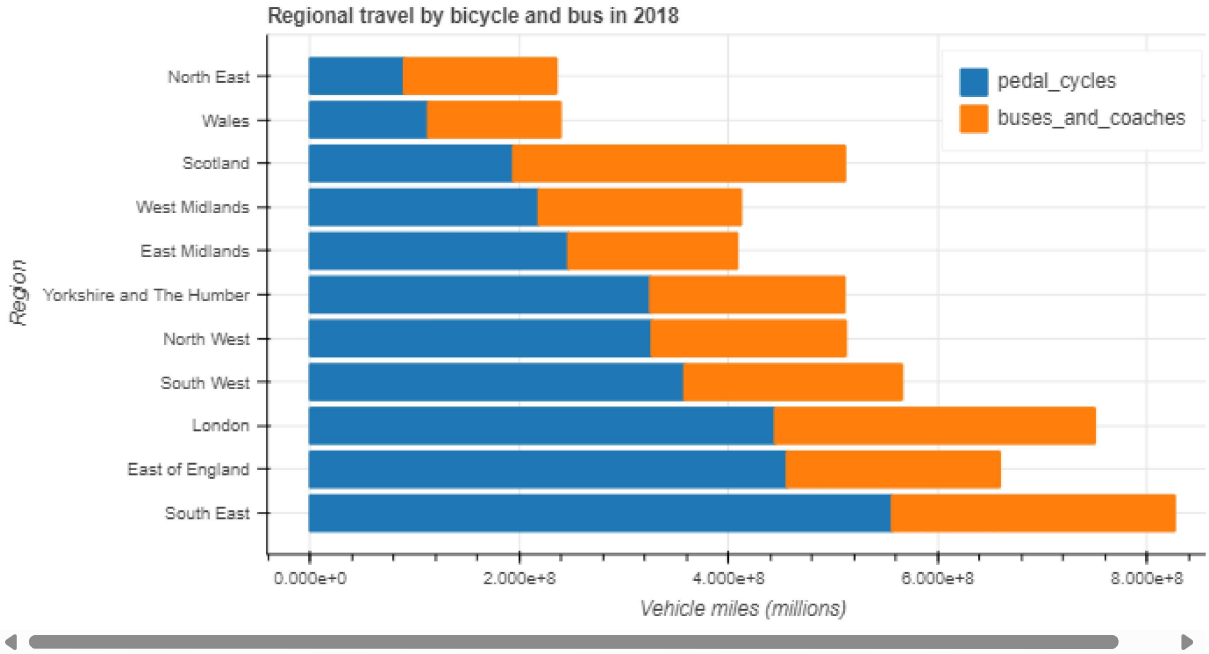
Q7. Once you have created your `green_bar` variable (specifying only that it should be a stacked, horizontal bar plot), modify the following properties of your variable such that:

- the plot `.width` is `800` pixels
- the `axis_label` of the `xaxis` is 'Vehicle miles (millions)'
- the `axis_label` of the `yaxis` is 'Region'
- the `text` of the `title` is 'Regional travel by bicycle and bus in 2018'

In [28]:
```
#add your code below
#green_bar.width = ...
#green_bar.xaxis.axis_label = ...
#green_bar.yaxis.axis_label = ...
#green_bar.title.text = ...


green_bar.width = 800
green_bar.xaxis.axis_label = 'Vehicle miles (millions)'
green_bar.yaxis.axis_label = 'Region'
green_bar.title.text = 'Regional travel by bicycle and bus in 2018'
```

Use `show()` to check that your changes have been made as expected:

In [29]:
```
#show(green_bar)

show(green_bar)
```



Q8. Create a DataFrame called `length_motor` as follows:

- group `df` by `['year', 'name']` with columns for `['total_link_length_miles', 'all_motor_vehicles']` containing the `.sum()` of these:

See below code syntax for some guidance:

```
DataFrame_Name.groupby(by=...)[list_of_cols].sum()
```

- add a new column to `length_motor` DataFrame called **'million_vehicle_miles_per_road_mile'** which is equal to to the following calculation: `(length_motor['all_motor_vehicles'] / 1000000) / length_motor['total_link_length_miles']`

In [32]:
```
#add your code below
#Length_motor =

length_motor = df.groupby(["year", "name"])[['total_link_length_miles', 'all_motor_vehicles']].sum(
length_motor['million_vehicle_miles_per_road_mile'] = (length_motor['all_motor_vehicles'] / 1000000
```

Q9. From `length_motor`, create a new DataFrame called `reg_density` which has a row index of `year` (i.e. one row for each year 1993-2018), and a column for each region (i.e. each unique value in `name`), with the values within the DataFrame being the appropriate `million_vehicle_miles_per_road_mile` for that year in the given region:

- do not change the original `length_motor` DataFrame
- you may find `.reset_index()` and the `.pivot()` method useful
- you can refer to the documentation here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html)

Please note you have been provided with the code for this question to carry out the necessary data manipulation work. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [34]:
```
#add your code below

#reg_density = length_motor.copy()
#reg_density.reset_index(inplace=True)
#reg_density = reg_density.pivot(index='year', columns='name', values='million_vehicle_miles_per_roc
#reg_density.head()

reg_density = length_motor.copy()
reg_density.reset_index(inplace=True)
reg_density = reg_density.pivot(index='year', columns='name', values='million_vehicle_miles_per_roa
reg_density.head()
```

Out[34]:

name	East Midlands	East of England	London	North East	North West	Scotland	South East	South West	Wales	West Midlands	Yorkshire and The Humber
year											
1993	1.064395	1.174043	2.140143	1.043946	1.293883	0.596892	1.514245	0.787532	0.678861	1.274398	1.092595
1994	1.087336	1.201897	2.164728	1.060768	1.314797	0.610051	1.547368	0.807469	0.693933	1.299053	1.114387
1995	1.107626	1.224337	2.161265	1.076316	1.339661	0.621164	1.577301	0.823139	0.706072	1.323180	1.135798
1996	1.140873	1.255611	2.177550	1.096399	1.371051	0.638259	1.625237	0.843202	0.722721	1.355891	1.166726
1997	1.163561	1.282051	2.187643	1.117606	1.396947	0.650531	1.661184	0.856932	0.737682	1.381401	1.185452

Q10. As we did earlier when creating `year_index` DataFrame, create a new DataFrame called `density_index`, which is the same as `reg_density` except the all values are relative to the 1993 value, which should equal `100`. Do not modify `reg_density` DataFrame.

Please note you have been provided with the code for this question to carry out the necessary data manipulation work. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [36]:
```
#add your code below

#density_index = reg_density.copy()
#base = density_index.iloc[0]
#density_index = (density_index/base)*100
#density_index.head()
density_index = reg_density.copy()
base = density_index.iloc[0]
density_index = (density_index/base)*100
density_index.head()
```

Out[36]:

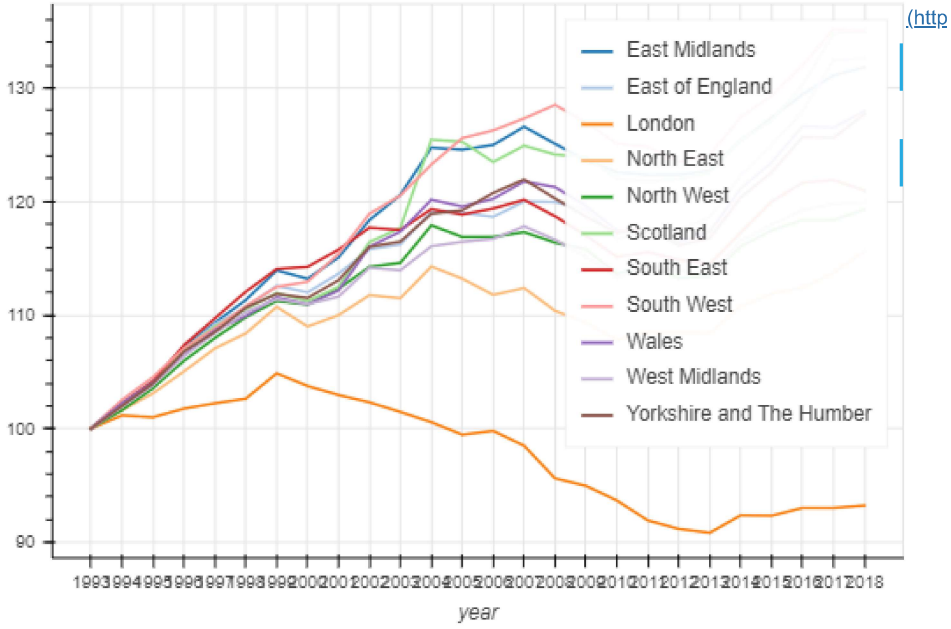
name	East Midlands	East of England	London	North East	North West	Scotland	South East	South West	Wales	Midl
year										
1993	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.00
1994	102.155346	102.372441	101.148749	101.611422	101.616362	102.204494	102.187408	102.531619	102.220089	101.93
1995	104.061565	104.283762	100.986935	103.100738	103.538069	104.066285	104.164175	104.521338	104.008194	103.82
1996	107.185155	106.947597	101.747874	105.024561	105.964080	106.930430	107.329813	107.068876	106.460690	106.39
1997	109.316675	109.199620	102.219469	107.055930	107.965470	108.986346	109.703741	108.812309	108.664595	108.39

Q11. Assign to `density_plot` a figure created by using the `.plot()` method on `density_index` DataFrame, with the parameter `hovertool=False`.

See below code syntax for some guidance:

```
density_plot = DataFrame_Name.plot(hovertool=False)
```

In [38]:
```
#add your code below
#density_plot =

density_plot = density_index.plot(hovertool=False)
```



Q12. Make the following changes to `density_plot` :

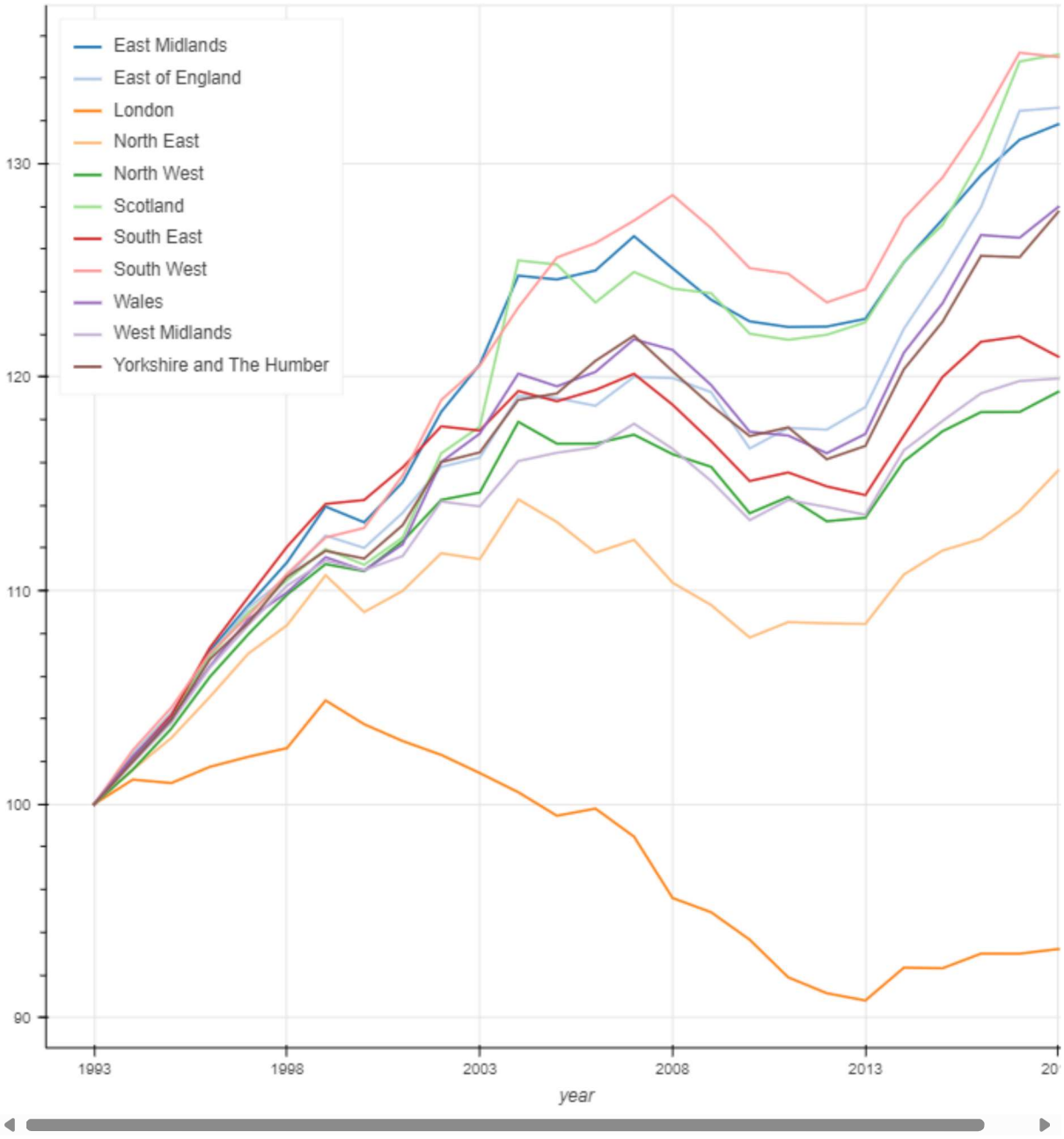
- make the `height` and `width` both `800`
- remove the toolbar by changing the `.toolbar_location` attribute to `None`
- change the legend location using `legend.location` attribute to `'top_left'`
- change the `ticker` of the `xaxis` to use the values `[1993, 1998, 2003, 2008, 2013, 2018]`

In [39]:
```
#add your code below
#density_plot.height = ...
#density_plot.width = ...
#density_plot.toolbar_location = ...
#density_plot.legend.location = ...
#density_plot.xaxis.ticker = ...

density_plot.height = 800
density_plot.width = 800
density_plot.toolbar_location = None
density_plot.legend.location = "top_left"
ticker_xaxis = [1993, 1998, 2003, 2008, 2013, 2018]
density_plot.xaxis.ticker = ticker_xaxis
```

Run the following cell to check your changes have been applied as expected:

In [40]:

```
#show(density_plot)

show(density_plot)
```



In [ ]: