Data Exploration with Python and Jupyter

Basic usage of the Pandas library to download a dataset, explore its contents, clean up missing or invalid data, filter the data according to different criteria, and plot visualizations of the data.

- Part 1: Python and Jupyter
- Part 2: Pandas with toy data
- **Part 3: Pandas with real data**

Press `Spacebar` to go to the next slide (or `?` to see all navigation shortcuts)

Let's download some real data

For some reason, the London Fire Brigade provides a public spreadsheet of all animal rescue incidents since 2009:

https://data.london.gov.uk/dataset/animal-rescue-incidents-attended-by-lfb

They provide a link to the dataset in csv (comma-delimited) format

Let's download some real data

For some reason, the London Fire Brigade provides a public spreadsheet of all animal rescue incidents since 2009:

https://data.london.gov.uk/dataset/animal-rescue-incidents-attended-by-lfb

They provide a link to the dataset in csv (comma-delimited) format

In [1]:
```
# import the Pandas library & matplotlib for plotting

import pandas as pd
import matplotlib.pyplot as plt
```

Let's download some real data

For some reason, the London Fire Brigade provides a public spreadsheet of all animal rescue incidents since 2009:

https://data.london.gov.uk/dataset/animal-rescue-incidents-attended-by-lfb

They provide a link to the dataset in csv (comma-delimited) format

In [1]:
```
# import the Pandas library & matplotlib for plotting

import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
# download a csv file with some data and convert it to a DataFrame
url = "https://data.london.gov.uk/download/animal-rescue-incidents-atter
df = pd.read_csv(url)
```

Suggested workflow / philosophy

- you want to do something
    - if you know / have a guess which function to use, look at its docstring: `?function_name`
    - if you don't have any idea what to try, google `how do I ... in pandas`
    - if in doubt, just try something!
- if you get an error, copy & paste the last bit into google (along with `funtion_name` and/or `pandas`)
    - don't be intimidated by the long and apparently nonsensical error messages
    - almost certainly someone else has had this exact problem
    - almost certainly the solution is waiting for you
- look for a stackoverflow answer with many up-votes
    - ignore the green tick, this just means the person asking the question liked the answer
    - typically an answer with many up-votes is a better option
    - more recent answers can also be better: sometimes a library has changed since an older answer was written

(For anyone who wasn't already doing this, that may be the most useful thing in this course)

Display the DataFrame

Display the DataFrame

In [3]:
```
df
```

Out[3]:

	IncidentNumber	DateTimeOfCall	CalYear	FinYear	TypeOfIncid
0	139091	2009-01-01 03:01:00	2009	2008/09	Special Serv
1	275091	2009-01-01 08:51:00	2009	2008/09	Special Serv
2	2075091	2009-01-04 10:07:00	2009	2008/09	Special Serv
3	2872091	2009-01-05 12:27:00	2009	2008/09	Special Serv
4	3553091	2009-01-06 15:23:00	2009	2008/09	Special Serv
...	...	...	...	...	
9723	096744-30062023	2023-06-30 15:56:00	2023	2023/24	Special Serv
9724	096880-30062023	2023-06-30 20:21:00	2023	2023/24	Special Serv
9725	096884-30062023	2023-06-30 20:31:00	2023	2023/24	Special Serv
9726	096913-30062023	2023-06-30 21:24:00	2023	2023/24	Special Serv
9727	096935-30062023	2023-06-30 22:26:00	2023	2023/24	Special Serv

9728 rows × 31 columns

Column data types

Column data types

In [4]:
```
df.dtypes
```

Out[4]:

```
IncidentNumber                object
DateTimeOfCall                object
CalYear                        int64
FinYear                       object
TypeOfIncident                object
PumpCount                    float64
PumpHoursTotal               float64
HourlyNotionalCost(£)          int64
IncidentNotionalCost(£)      float64
FinalDescription              object
AnimalGroupParent             object
OriginofCall                  object
PropertyType                  object
PropertyCategory              object
SpecialServiceTypeCategory    object
SpecialServiceType            object
WardCode                      object
Ward                          object
BoroughCode                   object
Borough                       object
StnGroundName                 object
UPRN                         float64
Street                        object
USRN                         float64
PostcodeDistrict              object
Easting_m                    float64
Northing_m                   float64
Easting_rounded                int64
Northing_rounded               int64
Latitude                     float64
```

```
Longitude                 float64
dtype: object
```

Convert DateTimeOfCall to a date-time

Convert DateTimeOfCall to a date-time

In [5]:
```
df["DateTimeOfCall"].head()
```

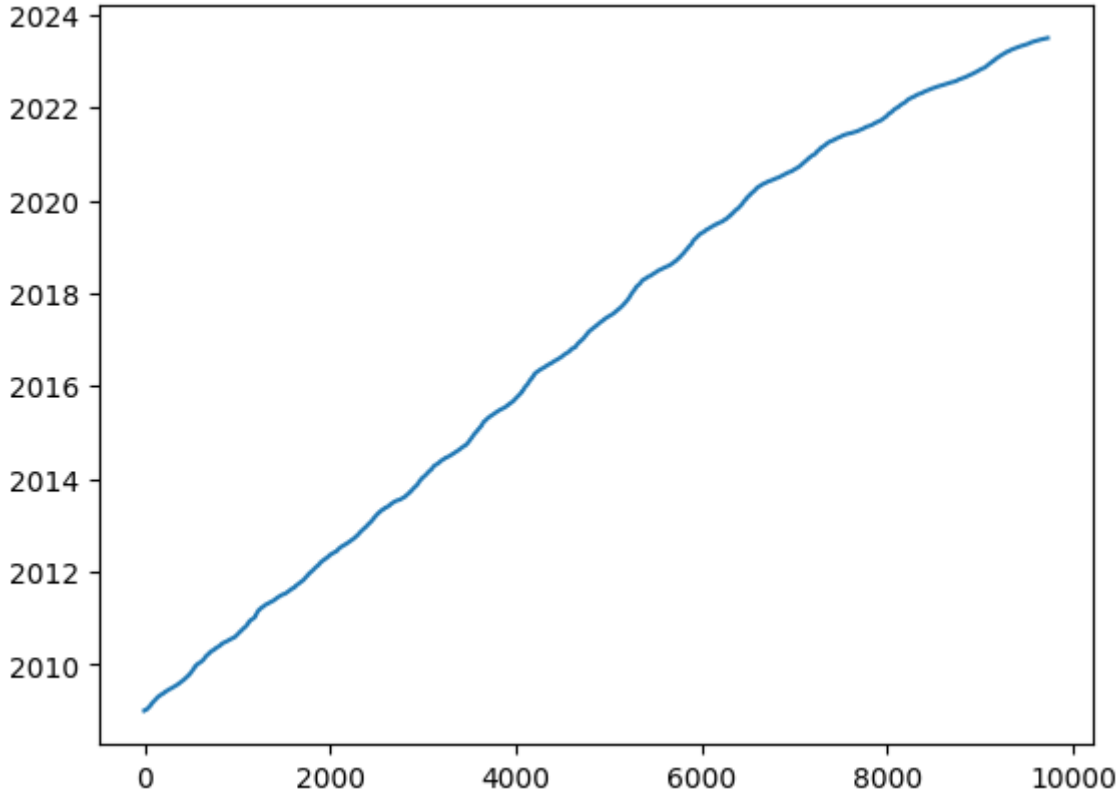
Out[5]:
```
0    2009-01-01 03:01:00
1    2009-01-01 08:51:00
2    2009-01-04 10:07:00
3    2009-01-05 12:27:00
4    2009-01-06 15:23:00
Name: DateTimeOfCall, dtype: object
```

In [6]:
```
# this looks like what we want..
pd.to_datetime(df["DateTimeOfCall"]).head()
```

Out[6]:
```
0   2009-01-01 03:01:00
1   2009-01-01 08:51:00
2   2009-01-04 10:07:00
3   2009-01-05 12:27:00
4   2009-01-06 15:23:00
Name: DateTimeOfCall, dtype: datetime64[ns]
```

In [7]:
```
# ..but which number is the month and which is the day?
# how can we check if what we just did was correct?
pd.to_datetime(df["DateTimeOfCall"]).plot()
# should be a single monotonically increasing line: looks good!
```

Out[7]:
```
<Axes: >
```



In [8]:
```
# replace DateTimeOfCall column in dataframe with this one
df["DateTimeOfCall"] = pd.to_datetime(df["DateTimeOfCall"])
```

Use the datetime as the index

Use the datetime as the index

In [9]:
```
df.set_index("DateTimeOfCall", inplace=True)
```

Use the datetime as the index

In [9]:
```
df.set_index("DateTimeOfCall", inplace=True)
```

In [10]:
```
df
```

Out[10]:

DateTimeOfCall	IncidentNumber	CalYear	FinYear	TypeOfIncident	Pu
2009-01-01 03:01:00	139091	2009	2008/09	Special Service	
2009-01-01 08:51:00	275091	2009	2008/09	Special Service	
2009-01-04 10:07:00	2075091	2009	2008/09	Special Service	
2009-01-05 12:27:00	2872091	2009	2008/09	Special Service	
2009-01-06 15:23:00	3553091	2009	2008/09	Special Service	
...	...	...	...	...	
2023-06-30 15:56:00	096744-30062023	2023	2023/24	Special Service	
2023-06-30 20:21:00	096880-30062023	2023	2023/24	Special Service	
2023-06-30 20:31:00	096884-30062023	2023	2023/24	Special Service	
2023-06-30 21:24:00	096913-30062023	2023	2023/24	Special Service	

In [11]:
```
# can now use datetime to select rows: here is jan 2021
df.loc["2021-01-01":"2021-01-31", "FinalDescription"]
```

	IncidentNumber	CalYear	FinYear	TypeOfIncident	Pu
22:26:00	30062023	2023	2023/24	Special Service	

9728 rows × 30 columns

Out[11]:

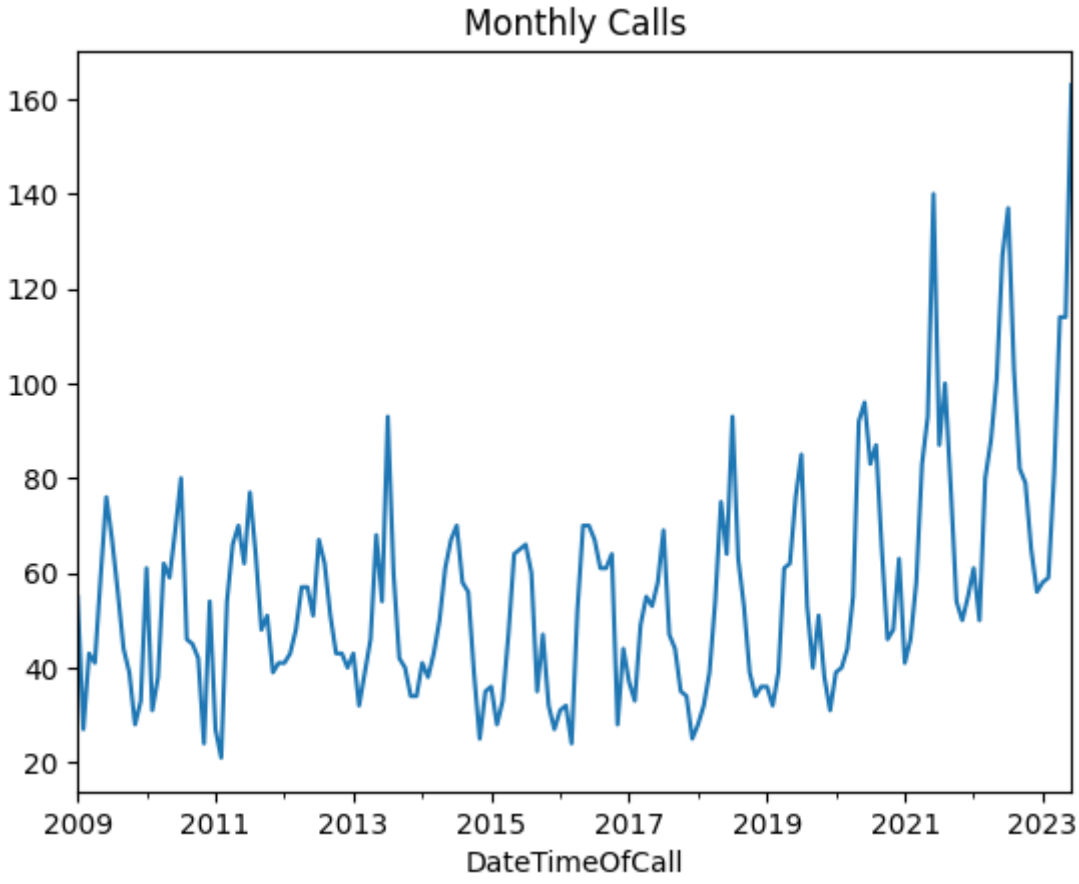
```
DateTimeOfCall
2021-01-01 12:09:00        KITTEN STUCK UP TREE  AL REQUESTED F
ROM SCENE
2021-01-01 14:06:00
Redacted
2021-01-03 18:40:00              CAT WITH LEG TRAPPED IN BATH
PLUGHOLE
2021-01-04 13:39:00
Redacted
2021-01-06 10:22:00
Redacted
2021-01-06 13:09:00    CAT IN DISTRESS ON ROOF - ADDITIONAL APP
LIANCE...
2021-01-06 20:35:00        DOG TRAPPED IN FOX HOLE  - MEET AT C
LUB HOUSE
2021-01-07 23:50:00                 KITTEN STUCK BETWEEN WALL
AND ROOF
2021-01-09 08:01:00                                  DOG STUCK
IN TRENCH
2021-01-10 19:27:00
Redacted
2021-01-12 11:39:00
Redacted
2021-01-12 22:38:00                                CAT TRAPPED
IN DITCH
2021-01-16 18:05:00                           DOG TRAPPED IN POR
TER CABIN
2021-01-17 16:09:00    DOG TRAPPED IN WAREHOUSE AREA - CALLER B
ELIEVE...
2021-01-17 17:09:00      BIRD TRAPPED IN NETTING    CALLER WILL
```

```
2021-01-18 15:17:00                CAT STUCK IN TREE BEING ATTACKED
BY CROWS
2021-01-18 17:06:00    ASSIST RSPCA - SMALL ANIMAL RESUE - BIRD
ENTAN...
2021-01-19 18:28:00                              CAT TRAPPED BEHIND
CUPBOARD
2021-01-19 20:24:00
Redacted
2021-01-19 20:36:00                                   RUNNING CALL A
T ON ROOF
2021-01-20 09:35:00                         CAT STUCK BETWEEN TREE
BRANCHES
2021-01-21 13:15:00                                   SWAN TRAPPED I
N NETTING
2021-01-21 18:23:00                                    CAT TRAPPED I
N CHIMNEY
2021-01-22 14:22:00                      CAT TRAPPED BETWEEN WALL
AND FENCE
2021-01-23 10:18:00                                    CAT TRAPPED I
N CHIMNEY
2021-01-23 15:43:00                                CAT TRAPPED BETW
EEN WALLS
2021-01-23 17:16:00
Redacted
2021-01-25 12:02:00              ASSIST RSPCA WITH FOX STUCK DOW
N CULVERT
2021-01-26 13:42:00          DOG STUCK IN RAILINGS - CALLER WILL
MEET YOU
2021-01-26 18:21:00
Redacted
2021-01-26 22:44:00    BIRDS TRAPPED IN BASKETBALL COURT CALLER
```

```
IS ON...
2021-01-26 23:35:00                FOX TRAPPED IN FENCE IN ALLEYWA
```

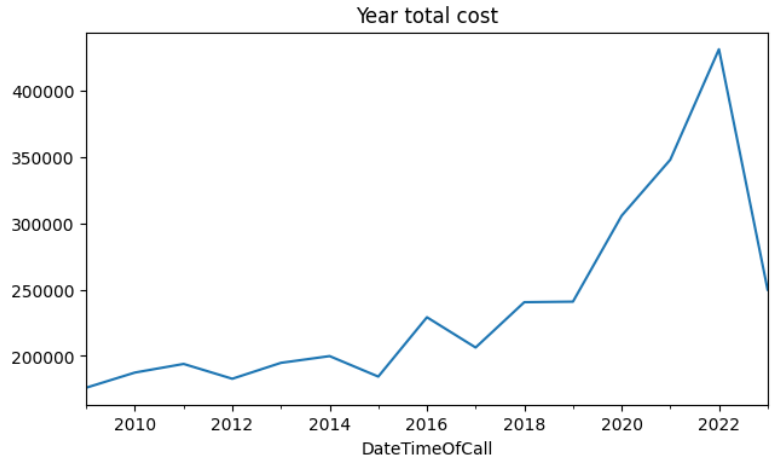
In [12]:
```
# resample the timeseries by month and count incidents
df.resample("M")["IncidentNumber"].count().plot(title="Monthly Calls")
# see https://pandas.pydata.org/docs/user_guide/timeseries.html#timeseri
plt.show()
```

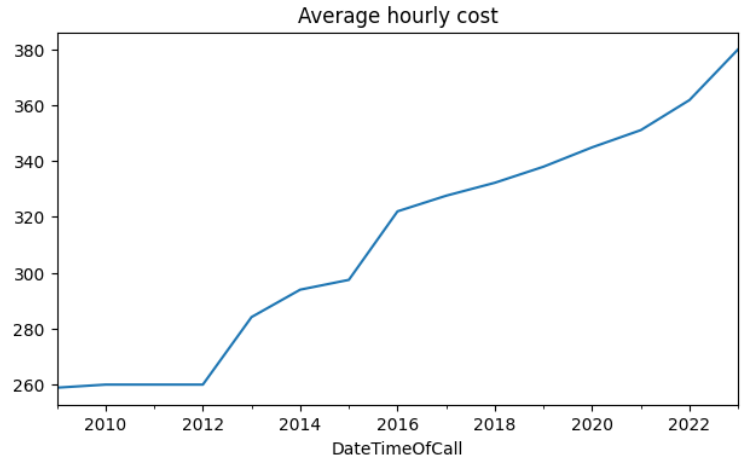
```
2021-01-27 15:22:00                 CAT UP TREE   ASS
IST RSPCA
```




```
OX IN FENCE  IN RE
       CAT STUCK U
IN NETTING - RSPCA
      DOG TRAPPED
CK UP TREE - RSPCA
CK IN GREEN AREA A
```

In [13]:
```
# resample by year, sum total costs, average hourly costs
fig, axs = plt.subplots(figsize=(16, 4), ncols=2)
df.resample("Y")["IncidentNotionalCost(£)"].sum().plot(
    title="Year total cost", ax=axs[0]
)
df.resample("Y")["HourlyNotionalCost(£)"].mean().plot(
    title="Average hourly cost", ax=axs[1]
)
plt.show()
```





Missing data

Different strategies for dealing with missing data:

- Ignore the issue
    - some things may break / not work as expected
- Remove rows/columns with missing data
    - remove all rows with missing data: `df.dropna(axis=0)`
    - remove all columns with missing data: `df.dropna(axis=1)`
- Guess (impute) missing data
    - replace all missing entries with a value: `df.fillna(1)`
    - replace missing entries with mean for that column `df.fillna(df.mean())`
    - replace each missing entry with previous valid entry: `df.fillna(method="pad")`
    - replace missing by interpolating between valid entries: `df.interpolate()`

In [14]:
```
# count missing entries for each column
df.isna().sum()
```

Out[14]:

IncidentNumber	0
CalYear	0
FinYear	0
TypeOfIncident	0
PumpCount	65
PumpHoursTotal	66
HourlyNotionalCost(£)	0
IncidentNotionalCost(£)	66
FinalDescription	5
AnimalGroupParent	0
OriginofCall	0
PropertyType	0
PropertyCategory	0
SpecialServiceTypeCategory	0
SpecialServiceType	0
WardCode	10
Ward	10
BoroughCode	12
Borough	12
StnGroundName	0
UPRN	6127
Street	0
USRN	1156
PostcodeDistrict	0
Easting_m	5108
Northing_m	5108
Easting_rounded	0
Northing_rounded	0
Latitude	5108

```
Longitude                    5108
dtype: int64
```

In [15]:

```
# If PumpCount is missing, typically so is PumpHoursTotal
# 66 rows are missing at least one of these
pump_missing = df["PumpCount"].isna() | df["PumpHoursTotal"].isna()
print(pump_missing.sum())
```

```
66
```

In [15]:
```
# If PumpCount is missing, typically so is PumpHoursTotal
# 66 rows are missing at least one of these
pump_missing = df["PumpCount"].isna() | df["PumpHoursTotal"].isna()
print(pump_missing.sum())
```

```
66
```

In [16]:
```
# so we could choose to drop these rows
df1 = df.drop(df.loc[pump_missing == True].index)
# here we made a new dataset df1 with these rows dropped
# to drop the rows from the original dataset df, could do:
#
# df = df.drop(df.loc[pump_missing == True].index)
#
# or:
#
# df.drop(df.loc[pump_missing == True].index, inplace=True)
#
print(len(df1))
```

```
9662
```

In [15]:
```
# If PumpCount is missing, typically so is PumpHoursTotal
# 66 rows are missing at least one of these
pump_missing = df["PumpCount"].isna() | df["PumpHoursTotal"].isna()
print(pump_missing.sum())
```

```
66
```

In [16]:
```
# so we could choose to drop these rows
df1 = df.drop(df.loc[pump_missing == True].index)
# here we made a new dataset df1 with these rows dropped
# to drop the rows from the original dataset df, could do:
#
# df = df.drop(df.loc[pump_missing == True].index)
#
# or:
#
# df.drop(df.loc[pump_missing == True].index, inplace=True)
#
print(len(df1))
```

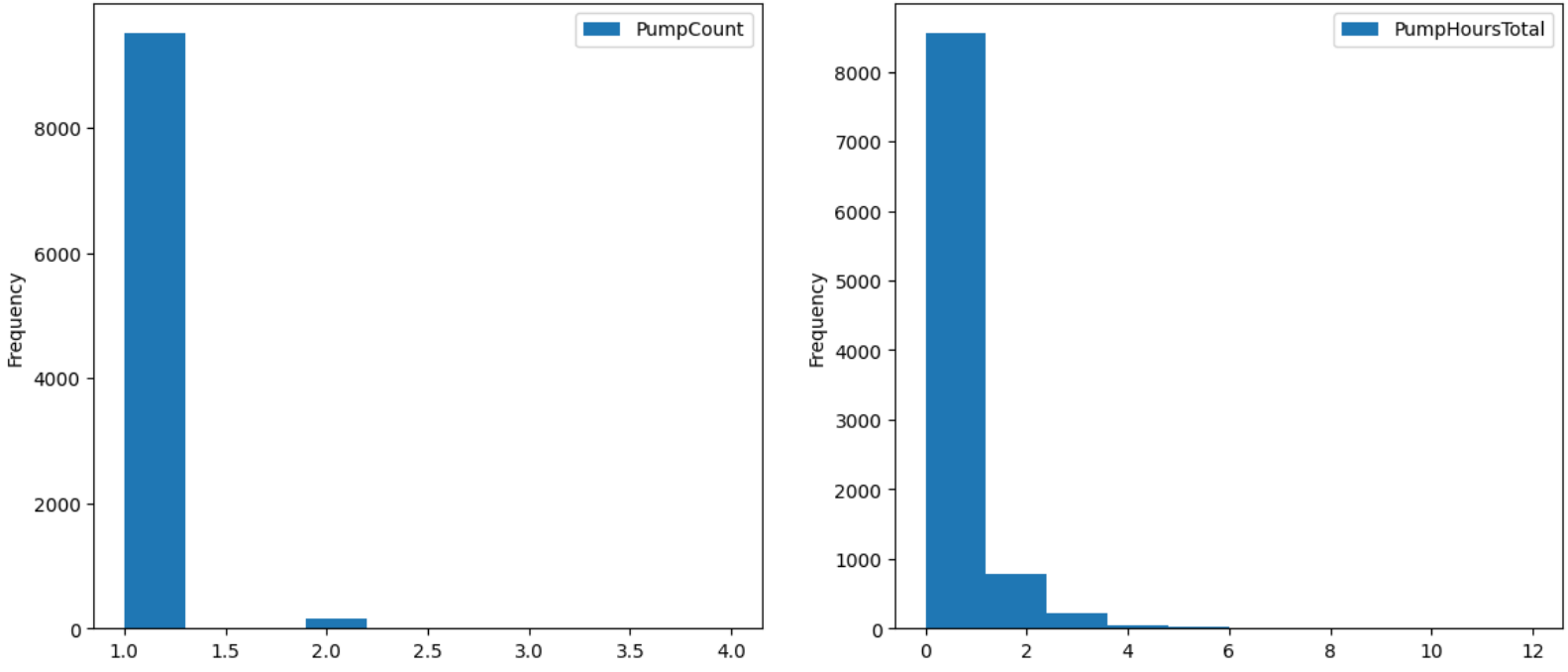
```
9662
```

In [17]:
```
# another equivalent way to do this
df2 = df.dropna(subset=["PumpCount", "PumpHoursTotal"])
print(len(df2))
```

```
9662
```

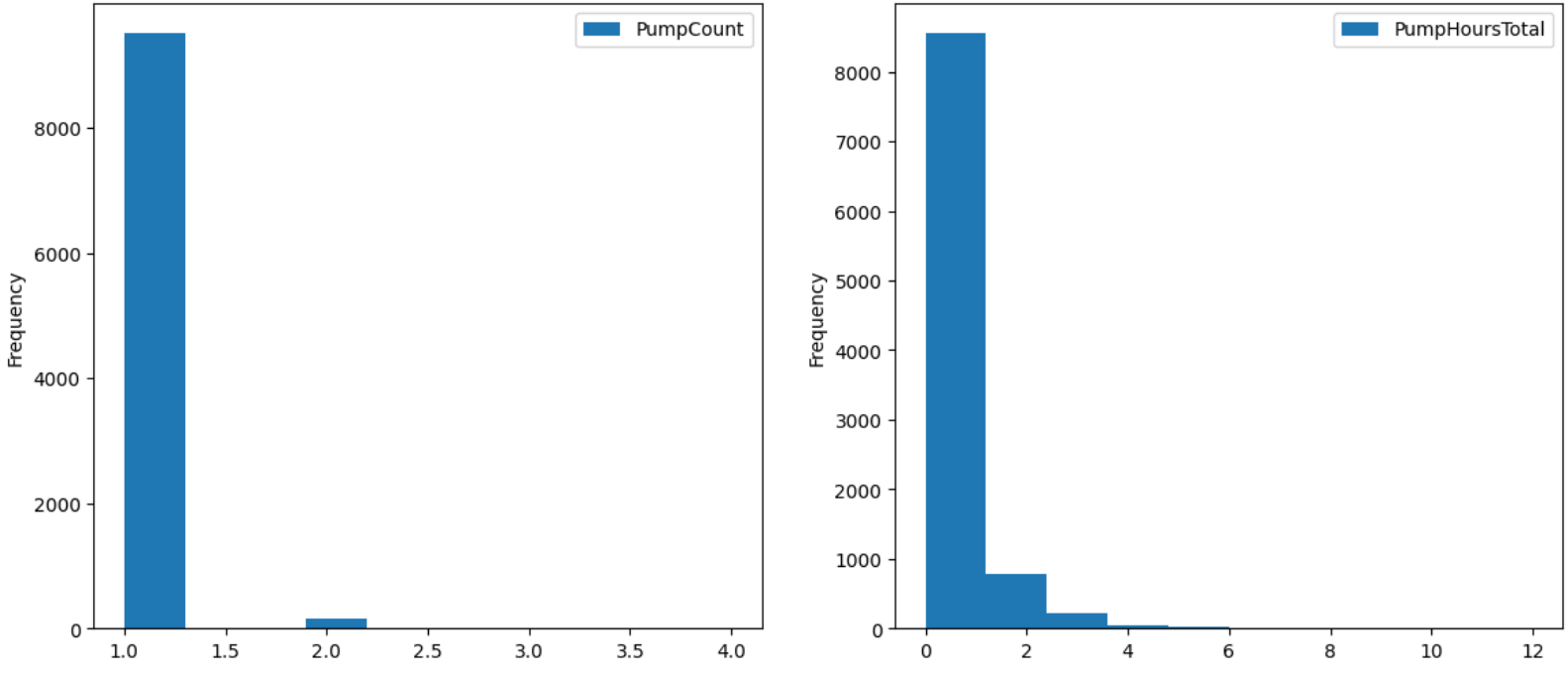
In [18]:
```
# but if we drop them, we lose valid data from other columns
# let's look at the distribution of values:
fig, axs = plt.subplots(1, 2, figsize=(14, 6))
df.plot.hist(y="PumpCount", ax=axs[0])
df.plot.hist(y="PumpHoursTotal", ax=axs[1])
plt.plot()
```

Out[18]:
```
[]
```



In [18]:
```
# but if we drop them, we lose valid data from other columns
# let's look at the distribution of values:
fig, axs = plt.subplots(1, 2, figsize=(14, 6))
df.plot.hist(y="PumpCount", ax=axs[0])
df.plot.hist(y="PumpHoursTotal", ax=axs[1])
plt.plot()
```

Out[18]:
```
[]
```



In [19]:
```
# looks like it would be better to replace missing PumpCount and PumpHou
?df.fillna
df.fillna({"PumpCount": 1, "PumpHoursTotal": 1}, inplace=True)
```

In [20]:
```
df.isna().sum()
```

Out[20]:

IncidentNumber	0
CalYear	0
FinYear	0
TypeOfIncident	0
PumpCount	0
PumpHoursTotal	0
HourlyNotionalCost(£)	0
IncidentNotionalCost(£)	66
FinalDescription	5
AnimalGroupParent	0
OriginofCall	0
PropertyType	0
PropertyCategory	0
SpecialServiceTypeCategory	0
SpecialServiceType	0
WardCode	10
Ward	10
BoroughCode	12
Borough	12
StnGroundName	0
UPRN	6127
Street	0
USRN	1156
PostcodeDistrict	0
Easting_m	5108
Northing_m	5108
Easting_rounded	0
Northing_rounded	0
Latitude	5108

```
Longitude                      5108
dtype: int64
```

Count the unique entries in each column

Count the unique entries in each column

In [21]:
```
df.nunique().sort_values()
```

Out[21]:

TypeOfIncident	1
PumpCount	4
SpecialServiceTypeCategory	4
PropertyCategory	7
OriginofCall	8
PumpHoursTotal	12
HourlyNotionalCost(£)	13
CalYear	15
FinYear	16
SpecialServiceType	24
AnimalGroupParent	28
BoroughCode	37
Borough	70
IncidentNotionalCost(£)	82
StnGroundName	108
PropertyType	187
PostcodeDistrict	277
Northing_rounded	425
Easting_rounded	530
WardCode	759
Ward	1272
UPRN	3446
Northing_m	4188
Easting_m	4254
Longitude	4549
Latitude	4549
FinalDescription	5907
USRN	6496
Street	7172

```
IncidentNumber                9728
dtype: int64
```

In [22]:
```
# "cat" and "Cat" are treated as different animals here:
df["AnimalGroupParent"].unique()
```

Out[22]:
```
array(['Dog', 'Fox', 'Horse', 'Rabbit',
       'Unknown - Heavy Livestock Animal', 'Squirrel', 'Cat',
'Bird',
       'Unknown - Domestic Animal Or Pet', 'Sheep', 'Deer',
       'Unknown - Wild Animal', 'Snake', 'Lizard', 'Hedgehog',
'cat',
       'Hamster', 'Lamb', 'Fish', 'Bull', 'Cow', 'Ferret', 'Bud
gie',
       'Unknown - Animal rescue from water - Farm animal', 'Pig
eon',
       'Goat', 'Tortoise',
       'Unknown - Animal rescue from below ground - Farm anima
l'],
      dtype=object)
```

In [22]:
```
# "cat" and "Cat" are treated as different animals here:
df["AnimalGroupParent"].unique()
```

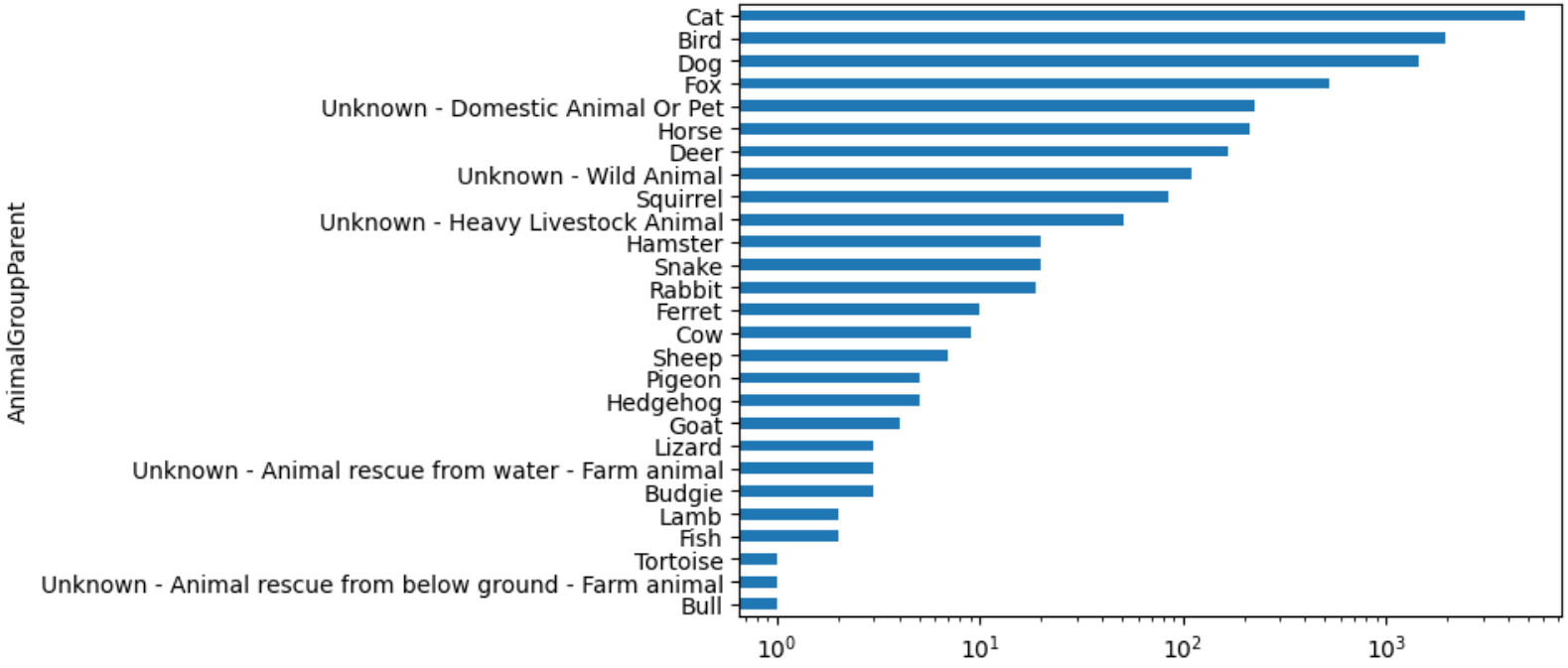
Out[22]:
```
array(['Dog', 'Fox', 'Horse', 'Rabbit',
       'Unknown - Heavy Livestock Animal', 'Squirrel', 'Cat',
'Bird',
       'Unknown - Domestic Animal Or Pet', 'Sheep', 'Deer',
       'Unknown - Wild Animal', 'Snake', 'Lizard', 'Hedgehog',
'cat',
       'Hamster', 'Lamb', 'Fish', 'Bull', 'Cow', 'Ferret', 'Bud
gie',
       'Unknown - Animal rescue from water - Farm animal', 'Pig
eon',
       'Goat', 'Tortoise',
       'Unknown - Animal rescue from below ground - Farm anima
l'],
      dtype=object)
```

In [23]:
```
# select rows where AnimalGroupParent is "cat", replace with "Cat"
df.loc[df["AnimalGroupParent"] == "cat", "AnimalGroupParent"] = "Cat"
```

In [24]:
```
df["AnimalGroupParent"].unique()
```

Out[24]:
```
array(['Dog', 'Fox', 'Horse', 'Rabbit',
       'Unknown - Heavy Livestock Animal', 'Squirrel', 'Cat',
'Bird',
       'Unknown - Domestic Animal Or Pet', 'Sheep', 'Deer',
       'Unknown - Wild Animal', 'Snake', 'Lizard', 'Hedgehog',
'Hamster',
       'Lamb', 'Fish', 'Bull', 'Cow', 'Ferret', 'Budgie',
       'Unknown - Animal rescue from water - Farm animal', 'Pig
eon',
       'Goat', 'Tortoise',
       'Unknown - Animal rescue from below ground - Farm anima
l'],
      dtype=object)
```

In [25]:
```
df.groupby("AnimalGroupParent")["IncidentNumber"].count().sort_values().
    logx=True
)
plt.show()
```

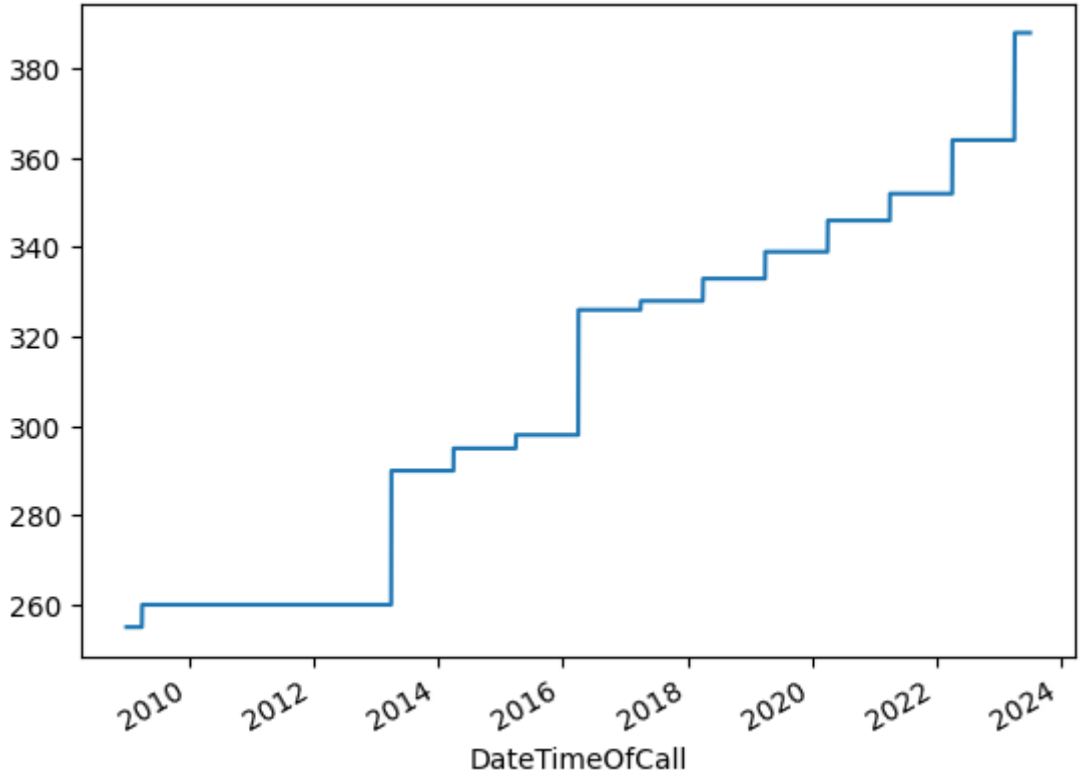


In [26]:
```
# apparently different hourly costs
# does it depend on the type of event? or does it just increase over tim
df["HourlyNotionalCost(£)"].unique()
```

Out[26]:
```
array([255, 260, 290, 295, 298, 326, 328, 333, 339, 346, 352, 3
64, 388])
```

In [27]:
```
# just goes up over time
df["HourlyNotionalCost(£)"].plot.line()
```

Out[27]:
```
<Axes: xlabel='DateTimeOfCall'>
```



In [28]:
```
# Group incidents by fire station & count them
df.groupby("StnGroundName")["IncidentNumber"].count()
```

Out[28]:
```
StnGroundName
Acton          74
Addington      66
Barking        91
Barnet         95
Battersea      82
               ..
Whitechapel    26
Willesden      68
Wimbledon      75
Woodford       95
Woodside       83
Name: IncidentNumber, Length: 108, dtype: int64
```

Plot location of calls on a map

- note: this section uses some more libraries, to install them:
- `pip install geopandas contextily`

In [29]:

```
# drop missing longitude/latitude
df2 = df.dropna(subset=["Longitude", "Latitude"])
# also drop zero values
df2 = df2[df2["Latitude"] != 0]
# convert to geodataframe using geopandas
import geopandas

# set crs to EPSG:4326 to specify WGS84 Latitude/Longitude
gdf = geopandas.GeoDataFrame(
    df2,
    geometry=geopandas.points_from_xy(df2["Longitude"], df2["Latitude"])
    crs="EPSG:4326",
)
gdf.head()
```

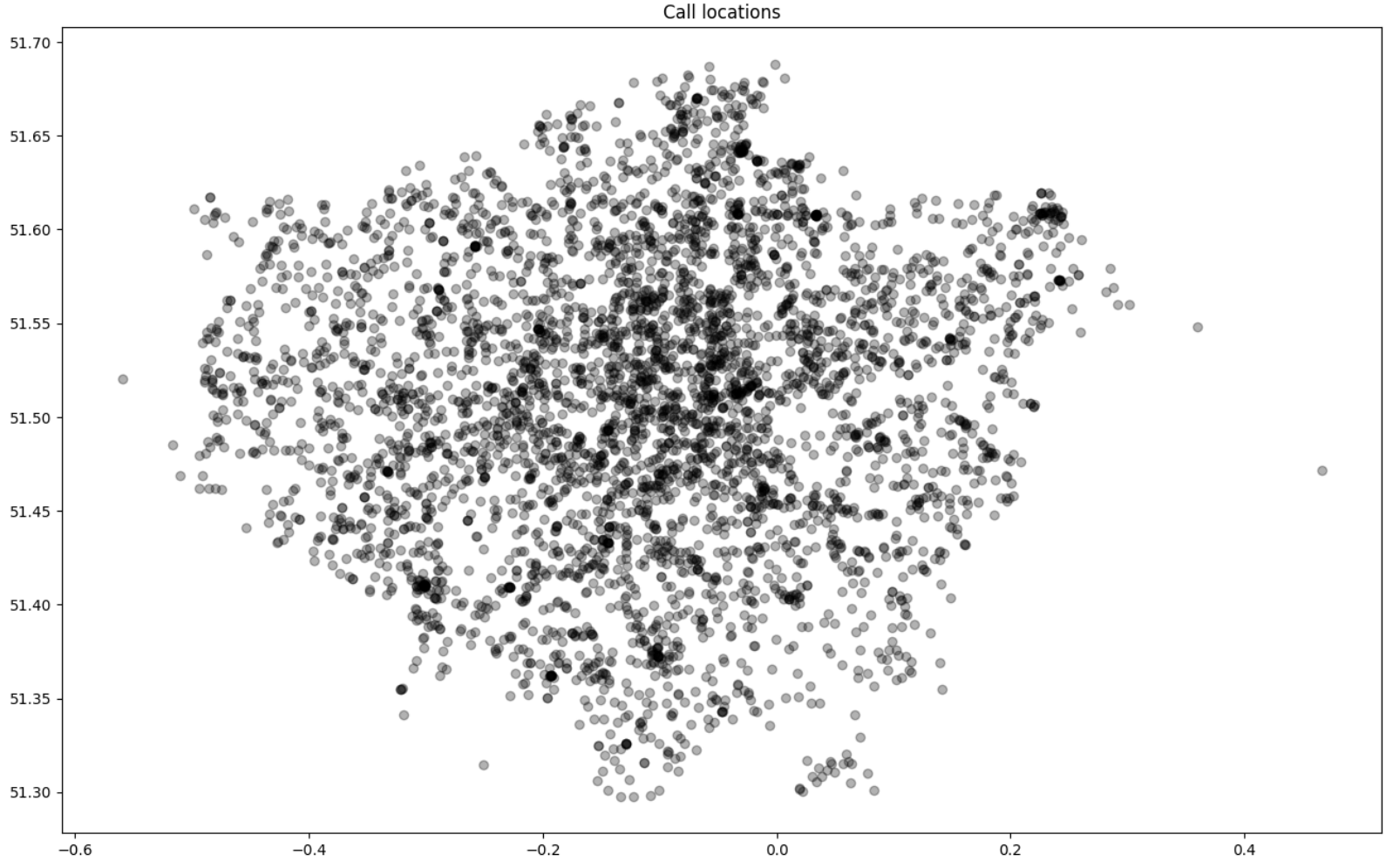
Out[29]:

DateTimeOfCall	IncidentNumber	CalYear	FinYear	TypeOfIncident	Pu
2009-01-01 08:51:00	275091	2009	2008/09	Special Service	
2009-01-04 10:07:00	2075091	2009	2008/09	Special Service	
2009-01-05 12:27:00	2872091	2009	2008/09	Special Service	
2009-01-07 06:29:00	4011091	2009	2008/09	Special Service	
2009-01-07 11:55:00	4211091	2009	2008/09	Special Service	

5 rows × 31 columns

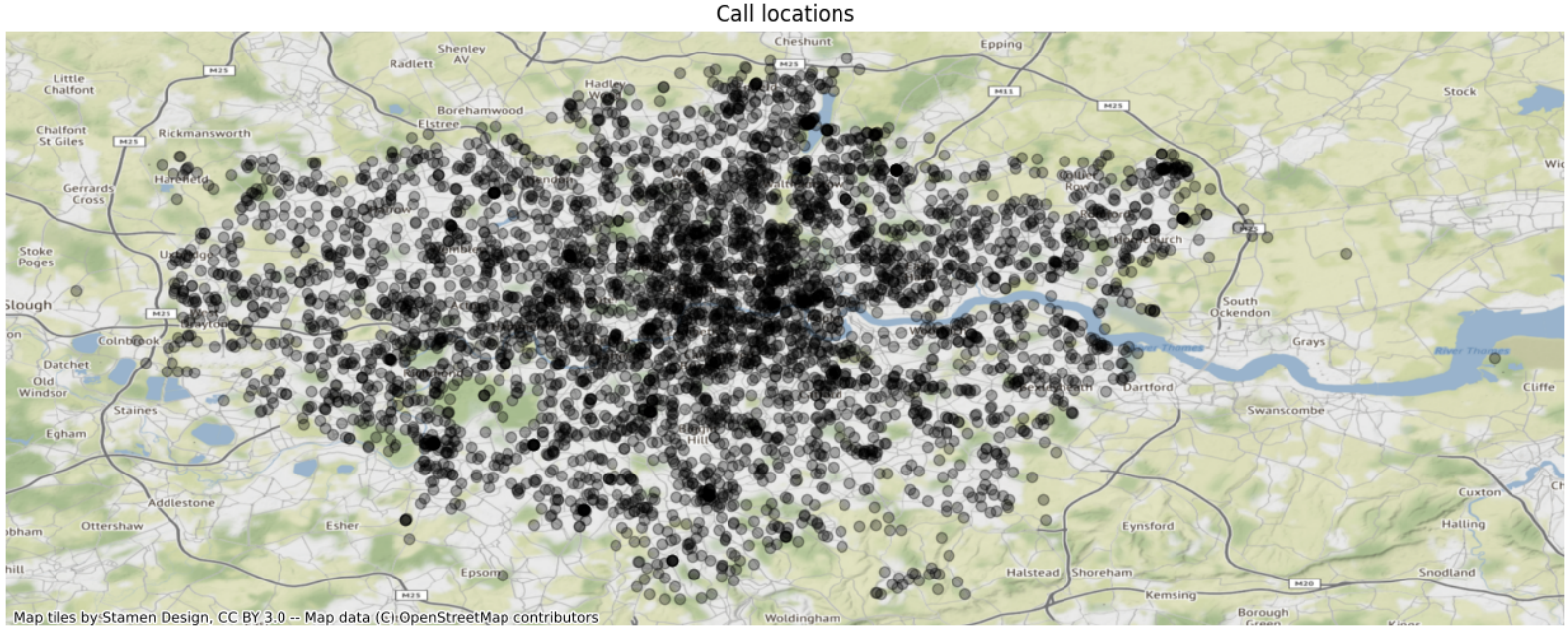
In [30]:

```
f, ax = plt.subplots(figsize=(16, 16))
# plot location of calls involving animals
gdf.plot(ax=ax, color="black", alpha=0.3)
plt.title("Call locations")
# plt.axis("off")
plt.show()
```



In [31]:

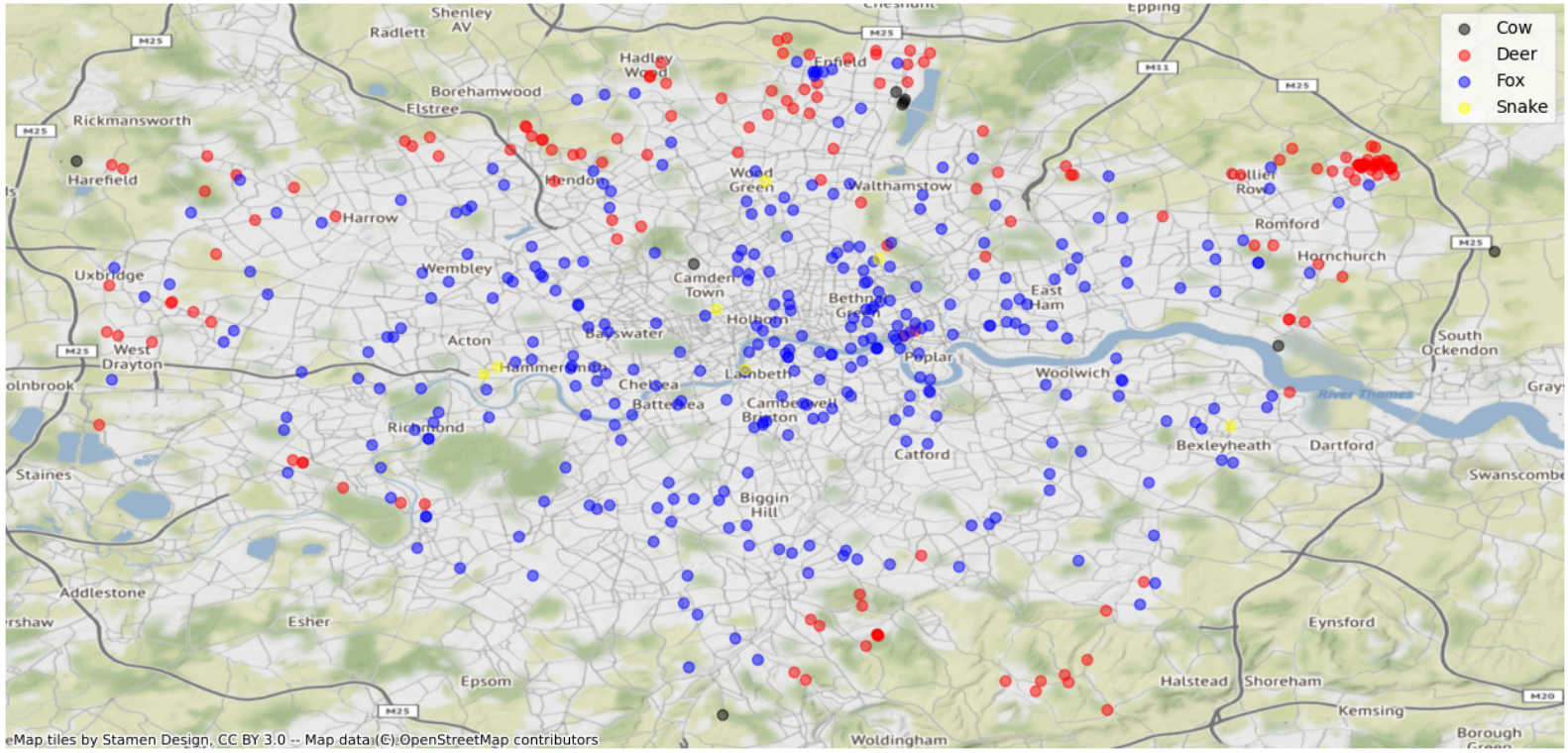
```
import contextily as cx

f, ax = plt.subplots(figsize=(16, 16))
# plot location of calls involving animals
gdf.plot(ax=ax, color="black", alpha=0.3)
# add a basemap of the region using contextily
cx.add_basemap(ax, crs=gdf.crs)
plt.title("Call locations")
plt.axis("off")
plt.show()
```



In [32]:

```
f, ax = plt.subplots(figsize=(16, 16))
# plot location of calls involving animals
for animal, colour in [
    ("Cow", "black"),
    ("Deer", "red"),
    ("Fox", "blue"),
    ("Snake", "yellow"),
]:
    gdf[gdf["AnimalGroupParent"] == animal].plot(
        ax=ax, color=colour, alpha=0.5, label=animal
    )
# add a basemap of the region using contextily
cx.add_basemap(ax, crs=gdf.crs)
plt.title("Call locations by animal")
plt.legend()
plt.axis("off")
plt.show()
```

Call locations by animal



Next steps

- experiment with your own datasets
- read some pandas documentation
    - user guide
- follow a tutorial
    - getting started tutorials
- free interactive kaggle courses
    - pandas
    - data cleaning