Notebook 1 Exploratory data analysis

June 14, 2022

[3]:
```python
from sklearn.datasets import fetch_openml

dataset=fetch_openml(data_id=42803, as_frame=True)
print(dataset)

df_X=dataset['frame']
```

```
{'data':         Accident_Index  Vehicle_Reference_df_res  Vehicle_Type  \
0       201501BS70001                       1.0          19.0
1       201501BS70002                       1.0           9.0
2       201501BS70004                       1.0           9.0
3       201501BS70005                       1.0           9.0
4       201501BS70008                       1.0           1.0
…                   …                         …            …
363238  2015984141415                      13.0           9.0
363239  2015984141415                      13.0           9.0
363240  2015984141415                      13.0           9.0
363241  2015984141415                      13.0           9.0
363242  2015984141415                      13.0           9.0

        Towing_and_Articulation  Vehicle_Manoeuvre  \
0                           0.0                9.0
1                           0.0                9.0
2                           0.0                9.0
3                           0.0                9.0
4                           0.0               18.0
…                             …                  …
363238                      0.0               18.0
363239                      0.0               18.0
363240                      0.0               18.0
363241                      0.0               18.0
363242                      0.0               18.0

        Vehicle_Location-Restricted_Lane  Junction_Location  \
0                                    0.0                8.0
1                                    0.0                8.0
2                                    0.0                2.0
3                                    0.0                2.0
```

```
4                                  0.0                 8.0
…                                …                 …
363238                             0.0                 0.0
363239                             0.0                 0.0
363240                             0.0                 0.0
363241                             0.0                 0.0
363242                             0.0                 0.0

        Skidding_and_Overturning  Hit_Object_in_Carriageway  \
0                            0.0                        0.0
1                            0.0                        0.0
2                            0.0                        0.0
3                            0.0                        0.0
4                            0.0                        0.0
…                          …                        …
363238                       0.0                        0.0
363239                       0.0                        0.0
363240                       0.0                        0.0
363241                       0.0                        0.0
363242                       0.0                        0.0

        Vehicle_Leaving_Carriageway  …  Age_Band_of_Casualty  \
0                               0.0  …                   7.0
1                               0.0  …                   5.0
2                               0.0  …                   6.0
3                               0.0  …                   2.0
4                               0.0  …                   8.0
…                             …  …                 …
363238                          5.0  …                   1.0
363239                          5.0  …                   5.0
363240                          5.0  …                   4.0
363241                          5.0  …                   6.0
363242                          5.0  …                   4.0

        Casualty_Severity  Pedestrian_Location  Pedestrian_Movement  \
0                     3.0                  5.0                  1.0
1                     3.0                  9.0                  9.0
2                     3.0                  1.0                  3.0
3                     3.0                  5.0                  1.0
4                     2.0                  0.0                  0.0
…                   …                  …                  …
363238                3.0                  0.0                  0.0
363239                3.0                  0.0                  0.0
363240                3.0                  0.0                  0.0
363241                3.0                  0.0                  0.0
363242                3.0                  0.0                  0.0

        Car_Passenger  Bus_or_Coach_Passenger  \
```

```
0                 0.0                    0.0
1                 0.0                    0.0
2                 0.0                    0.0
3                 0.0                    0.0
4                 0.0                    0.0
…               …                    …
363238            2.0                    0.0
363239            0.0                    0.0
363240            0.0                    0.0
363241            0.0                    0.0
363242            0.0                    0.0

        Pedestrian_Road_Maintenance_Worker  Casualty_Type  \
0                                      2.0            0.0
1                                      2.0            0.0
2                                      2.0            0.0
3                                      2.0            0.0
4                                      0.0            1.0
…                                    …            …
363238                                 0.0            9.0
363239                                 0.0            9.0
363240                                 0.0            9.0
363241                                 0.0            9.0
363242                                 0.0            9.0

        Casualty_Home_Area_Type  Casualty_IMD_Decile
0                           NaN                  NaN
1                           1.0                  3.0
2                           1.0                  6.0
3                           1.0                  2.0
4                           1.0                  3.0
…                         …                  …
363238                      1.0                  NaN
363239                      1.0                  2.0
363240                      2.0                  5.0
363241                      3.0                  NaN
363242                      1.0                  4.0

[363243 rows x 66 columns], 'target': 0         1.0
1         1.0
2         1.0
3         1.0
4         1.0
         …
363238    2.0
363239    2.0
363240    2.0
363241    2.0
```

```
363242    2.0
Name: Sex_of_Driver, Length: 363243, dtype: object, 'frame':
Accident_Index  Vehicle_Reference_df_res  Vehicle_Type  \
0       201501BS70001                        1.0          19.0
1       201501BS70002                        1.0           9.0
2       201501BS70004                        1.0           9.0
3       201501BS70005                        1.0           9.0
4       201501BS70008                        1.0           1.0
…               …                          …            …
363238  2015984141415                       13.0           9.0
363239  2015984141415                       13.0           9.0
363240  2015984141415                       13.0           9.0
363241  2015984141415                       13.0           9.0
363242  2015984141415                       13.0           9.0

        Towing_and_Articulation  Vehicle_Manoeuvre  \
0                           0.0                9.0
1                           0.0                9.0
2                           0.0                9.0
3                           0.0                9.0
4                           0.0               18.0
…                          …                 …
363238                      0.0               18.0
363239                      0.0               18.0
363240                      0.0               18.0
363241                      0.0               18.0
363242                      0.0               18.0

        Vehicle_Location-Restricted_Lane  Junction_Location  \
0                                    0.0                8.0
1                                    0.0                8.0
2                                    0.0                2.0
3                                    0.0                2.0
4                                    0.0                8.0
…                                  …                 …
363238                               0.0                0.0
363239                               0.0                0.0
363240                               0.0                0.0
363241                               0.0                0.0
363242                               0.0                0.0

        Skidding_and_Overturning  Hit_Object_in_Carriageway  \
0                            0.0                        0.0
1                            0.0                        0.0
2                            0.0                        0.0
3                            0.0                        0.0
4                            0.0                        0.0
…                          …                         …
```

```
363238                      0.0                        0.0
363239                      0.0                        0.0
363240                      0.0                        0.0
363241                      0.0                        0.0
363242                      0.0                        0.0

        Vehicle_Leaving_Carriageway  …  Age_Band_of_Casualty  \
0                               0.0  …                   7.0
1                               0.0  …                   5.0
2                               0.0  …                   6.0
3                               0.0  …                   2.0
4                               0.0  …                   8.0
…                               …  …                   …
363238                          5.0  …                   1.0
363239                          5.0  …                   5.0
363240                          5.0  …                   4.0
363241                          5.0  …                   6.0
363242                          5.0  …                   4.0

        Casualty_Severity  Pedestrian_Location  Pedestrian_Movement  \
0                     3.0                  5.0                  1.0
1                     3.0                  9.0                  9.0
2                     3.0                  1.0                  3.0
3                     3.0                  5.0                  1.0
4                     2.0                  0.0                  0.0
…                     …                    …                    …
363238                3.0                  0.0                  0.0
363239                3.0                  0.0                  0.0
363240                3.0                  0.0                  0.0
363241                3.0                  0.0                  0.0
363242                3.0                  0.0                  0.0

       Car_Passenger  Bus_or_Coach_Passenger  \
0                0.0                     0.0
1                0.0                     0.0
2                0.0                     0.0
3                0.0                     0.0
4                0.0                     0.0
…                …                       …
363238           2.0                     0.0
363239           0.0                     0.0
363240           0.0                     0.0
363241           0.0                     0.0
363242           0.0                     0.0

        Pedestrian_Road_Maintenance_Worker  Casualty_Type  \
0                                      2.0            0.0
1                                      2.0            0.0
```

```
2                                            2.0           0.0
3                                            2.0           0.0
4                                            0.0           1.0
…                                          …           …
363238                                       0.0           9.0
363239                                       0.0           9.0
363240                                       0.0           9.0
363241                                       0.0           9.0
363242                                       0.0           9.0

        Casualty_Home_Area_Type  Casualty_IMD_Decile
0                           NaN                  NaN
1                           1.0                  3.0
2                           1.0                  6.0
3                           1.0                  2.0
4                           1.0                  3.0
…                         …                  …
363238                      1.0                  NaN
363239                      1.0                  2.0
363240                      2.0                  5.0
363241                      3.0                  NaN
363242                      1.0                  4.0

[363243 rows x 67 columns], 'categories': None, 'feature_names':
['Accident_Index', 'Vehicle_Reference_df_res', 'Vehicle_Type',
'Towing_and_Articulation', 'Vehicle_Manoeuvre', 'Vehicle_Location-
Restricted_Lane', 'Junction_Location', 'Skidding_and_Overturning',
'Hit_Object_in_Carriageway', 'Vehicle_Leaving_Carriageway',
'Hit_Object_off_Carriageway', '1st_Point_of_Impact',
'Was_Vehicle_Left_Hand_Drive?', 'Journey_Purpose_of_Driver', 'Age_of_Driver',
'Age_Band_of_Driver', 'Engine_Capacity_(CC)', 'Propulsion_Code',
'Age_of_Vehicle', 'Driver_Home_Area_Type', 'Location_Easting_OSGR',
'Location_Northing_OSGR', 'Longitude', 'Latitude', 'Police_Force',
'Accident_Severity', 'Number_of_Vehicles', 'Number_of_Casualties', 'Date',
'Day_of_Week', 'Time', 'Local_Authority_(District)',
'Local_Authority_(Highway)', '1st_Road_Class', '1st_Road_Number', 'Road_Type',
'Speed_limit', 'Junction_Detail', 'Junction_Control', '2nd_Road_Class',
'2nd_Road_Number', 'Pedestrian_Crossing-Human_Control', 'Pedestrian_Crossing-
Physical_Facilities', 'Light_Conditions', 'Weather_Conditions',
'Road_Surface_Conditions', 'Special_Conditions_at_Site', 'Carriageway_Hazards',
'Urban_or_Rural_Area', 'Did_Police_Officer_Attend_Scene_of_Accident',
'LSOA_of_Accident_Location', 'Vehicle_Reference_df', 'Casualty_Reference',
'Casualty_Class', 'Sex_of_Casualty', 'Age_of_Casualty', 'Age_Band_of_Casualty',
'Casualty_Severity', 'Pedestrian_Location', 'Pedestrian_Movement',
'Car_Passenger', 'Bus_or_Coach_Passenger', 'Pedestrian_Road_Maintenance_Worker',
'Casualty_Type', 'Casualty_Home_Area_Type', 'Casualty_IMD_Decile'],
'target_names': ['Sex_of_Driver'], 'DESCR': 'Data reported to the police about
the circumstances of personal injury road accidents in Great Britain from 1979,
```

and the maker and model information of vehicles involved in the respective accident.\n\nThis version includes data up to 2015.\n\nDownloaded from openml.org.', 'details': {'id': '42803', 'name': 'road-safety', 'version': '4', 'description_version': '1', 'format': 'arff', 'creator': 'P. Cerda and G. Varoquaux', 'collection_date': '2015', 'upload_date': '2021-02-17T02:39:58', 'language': 'English', 'licence': 'Public', 'url': 'https://old.openml.org/data/v1/download/22045029/road-safety.arff', 'file_id': '22045029', 'default_target_attribute': 'Sex_of_Driver', 'citation': 'Cerda, P., & Varoquaux, G. (2020). Encoding high-cardinality string categorical variables. IEEE Transactions on Knowledge and Data Engineering.', 'visibility': 'public', 'original_data_url': 'https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data', 'minio_url': 'http://openml1.win.tue.nl/dataset42803/dataset_42803.pq', 'status': 'active', 'processing_date': '2021-02-17 02:40:18', 'md5_checksum': 'f942023301b7793cb34f104c53974923'}, 'url': 'https://www.openml.org/d/42803'}

1 Structure investigation

[5]:
```
# First look at general structure of dataset
df_X.shape
```

[5]:
```
(363243, 67)
```

[6]:
```
# how many different data types do these 67 features contain

import pandas as pd

pd.value_counts(df_X.dtypes)
```

[6]:
```
float64    61
object      6
dtype: int64
```

1.0.1 Structure of non numerical features

data types can be numerical as well as non-numerical. First let's take a closer look at non-numerical entries

[7]:
```
# Display non numeical features
df_X.select_dtypes(exclude="number").head()
```

[7]:
```
  Accident_Index Sex_of_Driver        Date   Time Local_Authority_(Highway)  \
0  201501BS70001           1.0  12/01/2015  18:45                 E09000020
1  201501BS70002           1.0  12/01/2015  07:50                 E09000020
2  201501BS70004           1.0  12/01/2015  18:08                 E09000020
3  201501BS70005           1.0  13/01/2015  07:40                 E09000020
4  201501BS70008           1.0  09/01/2015  07:30                 E09000020
```

```
  LSOA_of_Accident_Location
0                 E01002825
1                 E01002820
2                 E01002833
3                 E01002874
4                 E01002814
```

[8]:
```python
# Change data type of 'Sex_of_driver'

df_X['Sex_of_Driver']=df_X['Sex_of_Driver'].astype("float")
```

[9]:
```python
df_X.select_dtypes(exclude="number")
```

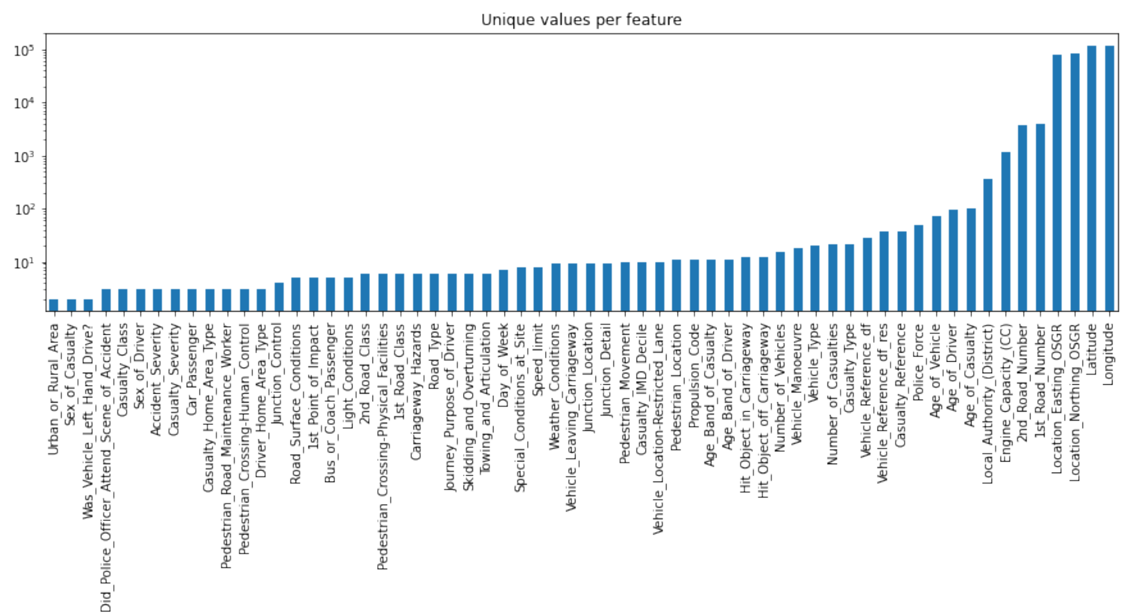
[9]:
```
       Accident_Index        Date   Time Local_Authority_(Highway)  \
0       201501BS70001  12/01/2015  18:45                 E09000020
1       201501BS70002  12/01/2015  07:50                 E09000020
2       201501BS70004  12/01/2015  18:08                 E09000020
3       201501BS70005  13/01/2015  07:40                 E09000020
4       201501BS70008  09/01/2015  07:30                 E09000020
…                 …         …    …                        …
363238  2015984141415  31/12/2015  16:37                 S12000006
363239  2015984141415  31/12/2015  16:37                 S12000006
363240  2015984141415  31/12/2015  16:37                 S12000006
363241  2015984141415  31/12/2015  16:37                 S12000006
363242  2015984141415  31/12/2015  16:37                 S12000006

       LSOA_of_Accident_Location
0                      E01002825
1                      E01002820
2                      E01002833
3                      E01002874
4                      E01002814
…                           …
363238                      None
363239                      None
363240                      None
363241                      None
363242                      None

[363243 rows x 5 columns]
```

1.0.2 Structure of numerical features

Lets see how many unique values each of these dataset has. This process will give more insights about the number of binary(2 uniqye values), ordinal(3-10 unique values) and continuous(more than 10 unique values) features in the dataset

[11]:
```python
# for each numerical feature compute number of unique entries
unique_values=df_X.select_dtypes(include="number").nunique().sort_values()

# plot information with y-axis in log scale
unique_values.plot.bar(logy=True,figsize=(15,4), title="Unique values per␣
↪feature")
```

[11]:
```
<AxesSubplot:title={'center':'Unique values per feature'}>
```



[12]:
```python
df_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 363243 entries, 0 to 363242
Data columns (total 67 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   Accident_Index                      363243 non-null  object
 1   Vehicle_Reference_df_res            363243 non-null  float64
 2   Vehicle_Type                        363181 non-null  float64
 3   Towing_and_Articulation             362864 non-null  float64
 4   Vehicle_Manoeuvre                   363059 non-null  float64
 5   Vehicle_Location-Restricted_Lane    363067 non-null  float64
 6   Junction_Location                   363159 non-null  float64
 7   Skidding_and_Overturning            363067 non-null  float64
 8   Hit_Object_in_Carriageway           363080 non-null  float64
 9   Vehicle_Leaving_Carriageway         363084 non-null  float64
 10  Hit_Object_off_Carriageway          363242 non-null  float64
```

```
11  1st_Point_of_Impact                          363037 non-null  float64
12  Was_Vehicle_Left_Hand_Drive?                 361760 non-null  float64
13  Journey_Purpose_of_Driver                    363221 non-null  float64
14  Sex_of_Driver                                363243 non-null  float64
15  Age_of_Driver                                327374 non-null  float64
16  Age_Band_of_Driver                           327374 non-null  float64
17  Engine_Capacity_(CC)                         269897 non-null  float64
18  Propulsion_Code                              270607 non-null  float64
19  Age_of_Vehicle                               256138 non-null  float64
20  Driver_Home_Area_Type                        302534 non-null  float64
21  Location_Easting_OSGR                        319793 non-null  float64
22  Location_Northing_OSGR                       319793 non-null  float64
23  Longitude                                    319793 non-null  float64
24  Latitude                                     319793 non-null  float64
25  Police_Force                                 319866 non-null  float64
26  Accident_Severity                            319866 non-null  float64
27  Number_of_Vehicles                           319866 non-null  float64
28  Number_of_Casualties                         319866 non-null  float64
29  Date                                         319866 non-null  object
30  Day_of_Week                                  319866 non-null  float64
31  Time                                         319822 non-null  object
32  Local_Authority_(District)                   319866 non-null  float64
33  Local_Authority_(Highway)                    319866 non-null  object
34  1st_Road_Class                               319866 non-null  float64
35  1st_Road_Number                              319866 non-null  float64
36  Road_Type                                    319866 non-null  float64
37  Speed_limit                                  319866 non-null  float64
38  Junction_Detail                              319864 non-null  float64
39  Junction_Control                             189222 non-null  float64
40  2nd_Road_Class                               188201 non-null  float64
41  2nd_Road_Number                              318416 non-null  float64
42  Pedestrian_Crossing-Human_Control            319547 non-null  float64
43  Pedestrian_Crossing-Physical_Facilities      319564 non-null  float64
44  Light_Conditions                             319866 non-null  float64
45  Weather_Conditions                           319866 non-null  float64
46  Road_Surface_Conditions                      319350 non-null  float64
47  Special_Conditions_at_Site                   319626 non-null  float64
48  Carriageway_Hazards                          319653 non-null  float64
49  Urban_or_Rural_Area                          319866 non-null  float64
50  Did_Police_Officer_Attend_Scene_of_Accident  319842 non-null  float64
51  LSOA_of_Accident_Location                    298758 non-null  object
52  Vehicle_Reference_df                         363243 non-null  float64
53  Casualty_Reference                           363243 non-null  float64
54  Casualty_Class                               363243 non-null  float64
55  Sex_of_Casualty                              363109 non-null  float64
56  Age_of_Casualty                              357674 non-null  float64
57  Age_Band_of_Casualty                         357674 non-null  float64
58  Casualty_Severity                            363243 non-null  float64
```

```
 59  Pedestrian_Location                        363241 non-null  float64
 60  Pedestrian_Movement                        363241 non-null  float64
 61  Car_Passenger                              362481 non-null  float64
 62  Bus_or_Coach_Passenger                     363197 non-null  float64
 63  Pedestrian_Road_Maintenance_Worker         363077 non-null  float64
 64  Casualty_Type                              363243 non-null  float64
 65  Casualty_Home_Area_Type                    323448 non-null  float64
 66  Casualty_IMD_Decile                        293666 non-null  float64
dtypes: float64(62), object(5)
memory usage: 185.7+ MB
```

[13]:
```
df_X.describe()
```

[13]:
```
       Vehicle_Reference_df_res   Vehicle_Type  Towing_and_Articulation  \
count             363243.000000  363181.000000            362864.000000
mean                   1.696203       9.756953                 0.029766
std                    1.487094       8.315189                 0.294127
min                    1.000000       1.000000                 0.000000
25%                    1.000000       9.000000                 0.000000
50%                    1.000000       9.000000                 0.000000
75%                    2.000000       9.000000                 0.000000
max                   37.000000      98.000000                 5.000000

       Vehicle_Manoeuvre  Vehicle_Location-Restricted_Lane  Junction_Location  \
count      363059.000000                     363067.000000      363159.000000
mean           12.607326                          0.109233           2.609361
std             6.218689                          0.903131           3.249245
min             1.000000                          0.000000           0.000000
25%             6.000000                          0.000000           0.000000
50%            17.000000                          0.000000           1.000000
75%            18.000000                          0.000000           6.000000
max            18.000000                          9.000000           8.000000

       Skidding_and_Overturning  Hit_Object_in_Carriageway  \
count             363067.000000              363080.000000
mean                   0.188139                   0.307480
std                    0.714243                   1.595551
min                    0.000000                   0.000000
25%                    0.000000                   0.000000
50%                    0.000000                   0.000000
75%                    0.000000                   0.000000
max                    5.000000                  12.000000

       Vehicle_Leaving_Carriageway  Hit_Object_off_Carriageway  …  \
count                363084.000000               363242.000000  …
mean                      0.366689                    0.546699  …
std                       1.374107                    2.094845  …
```

```
min                     0.000000                      0.000000  …
25%                     0.000000                      0.000000  …
50%                     0.000000                      0.000000  …
75%                     0.000000                      0.000000  …
max                     8.000000                     11.000000  …

       Age_Band_of_Casualty  Casualty_Severity  Pedestrian_Location  \
count         357674.000000      363243.000000        363241.000000
mean               6.431284           2.875725             0.380731
std                2.157860           0.355195             1.522220
min                1.000000           1.000000             0.000000
25%                5.000000           3.000000             0.000000
50%                6.000000           3.000000             0.000000
75%                8.000000           3.000000             0.000000
max               11.000000           3.000000            10.000000

       Pedestrian_Movement  Car_Passenger  Bus_or_Coach_Passenger  \
count        363241.000000  362481.000000           363197.000000
mean              0.276467       0.281027                0.066127
std               1.294574       0.591239                0.493174
min               0.000000       0.000000                0.000000
25%               0.000000       0.000000                0.000000
50%               0.000000       0.000000                0.000000
75%               0.000000       0.000000                0.000000
max               9.000000       2.000000                4.000000

       Pedestrian_Road_Maintenance_Worker  Casualty_Type  \
count                       363077.000000  363243.000000
mean                             0.032833       7.840080
std                              0.253780       7.366436
min                              0.000000       0.000000
25%                              0.000000       5.000000
50%                              0.000000       9.000000
75%                              0.000000       9.000000
max                              2.000000      98.000000

       Casualty_Home_Area_Type  Casualty_IMD_Decile
count            323448.000000        293666.000000
mean                  1.308186             5.107323
std                   0.657776             2.829458
min                   1.000000             1.000000
25%                   1.000000             3.000000
50%                   1.000000             5.000000
75%                   1.000000             7.000000
max                   3.000000            10.000000

[8 rows x 62 columns]
```

We now have a better understanding of general structure of our dataset. Number of sample and features, what kind of data type each feature has, and how many of them are binary, ordinal, categorical or continuous.

2 Quality investigation

Before focusing on actual content stored in these features, let's take a look at the general quality of the dataset. The goal is to have a global view on the dataset with regards to regards to things like duplicates, missing values and unwanted entries or recording errors

2.0.1 Duplicates

Duplicates are entires that represent the same sample point multipple times. Detecting such duplicates is not always easy, as each dataset might have a unique identifier which you might want to ignore first

[15]:
```
# check number of duplicates while ignoring index feature
n_duplicates=df_X.drop(labels=['Accident_Index'], axis=1).duplicated().sum()
print(f"You seem to have {n_duplicates} duplicates in your dataset")
```

```
You seem to have 22 duplicates in your dataset
```

[16]:
```
# to handle these duplicates you can just simply drop with .drop_duplicates()

# extract column names of all features , except 'Accident_Index'
columns_to_consider=df_X.drop(labels=['Accident_Index'], axis=1).columns

# Drop duplicates based on 'cloumns_to_consider'
df_X=df_X.drop_duplicates(subset=columns_to_consider)
df_X.shape
```

[16]:
```
(363221, 67)
```

2.0.2 Missing values

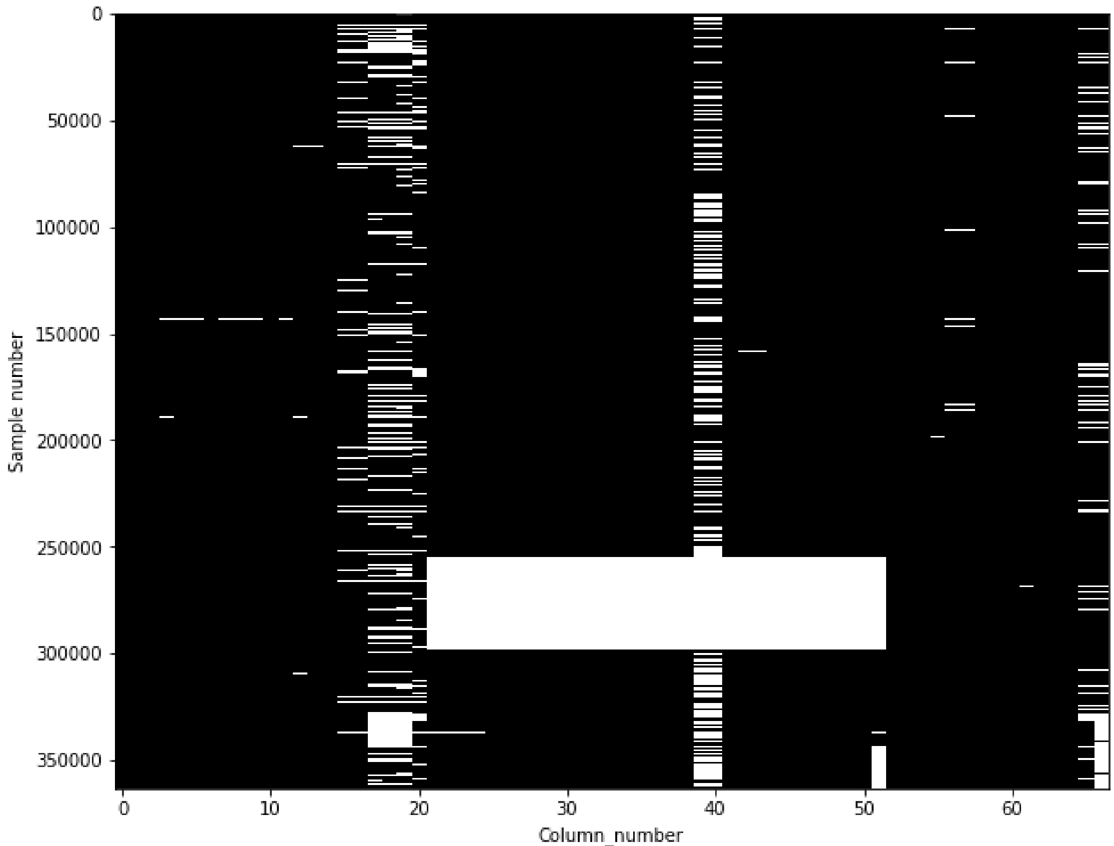
Another quality issue worth to investigate are missing values. Having some missing values is normal. What we want to identify are big holes in the dataset.

Per sample

To look at number of missing values per sample we have multiple options. The most straight forward one is to simply visualize the output of df_X.isna()

[18]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plt.imshow(df_X.isna(), aspect="auto", interpolation="nearest", cmap="gray")
plt.xlabel("Column_number")
plt.ylabel("Sample number")
```

[18]: Text(0, 0.5, 'Sample number')



This figure shows on the y-axis each of the 360'000 individual samples, and on the x-axis if any of the 67 features contains a missing value. While this is already a useful plot, an even better approach is to use the missingno library, to get a plot like this one:

[20]:
```
import missingno as msno

msno.matrix(df_X, labels=True, sort="descending");
```

From both of these plots we can see that the dataset has a huge whole, caused by some samples where more than 50% of the feature values are missing. For those samples, filling the missing values with some replacement values is probably not a good idea.

Therefore, let's go ahead and drop samples that have more than 20% of missing values. The threshold is inspired by the information from the 'Data Completeness' column on the right of this figure.

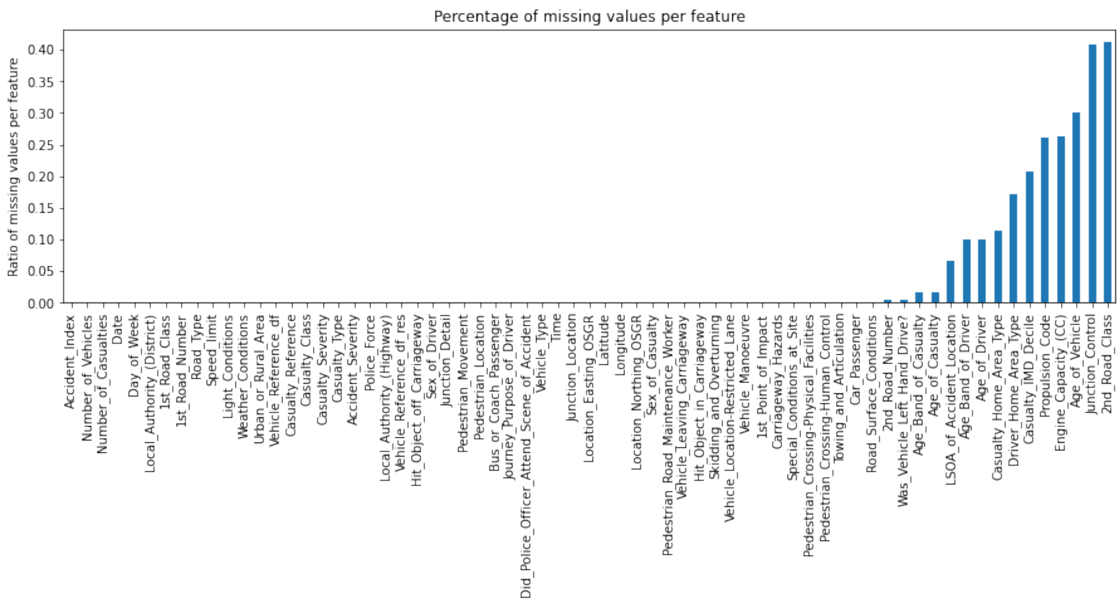
[21]:
```
df_X=df_X.dropna(thresh=df_X.shape[1]*0.8, axis=0).reset_index(drop=True)
df_X.shape
```

[21]: (319790, 67)

Per feature

As a next step, let's now look at the number of missing values per feature. For this we can use some pandas trickery to quickly identify the ratio of missing values per feature.

[22]:
```
df_X.isna().mean().sort_values().plot(
    kind="bar", figsize=(15, 4),
    title="Percentage of missing values per feature",
    ylabel="Ratio of missing values per feature");
```



From this figure we can see that most features don't contain any missing values. Nonetheless, features like 2nd__Road__Class, Junction__Control, Age__of__Vehicle still contain quite a lot of missing values. So let's go ahead and remove any feature with more than 15% of missing values.

[23]:
```
df_X = df_X.dropna(thresh=df_X.shape[0] * 0.85, axis=1)
df_X.shape
```

[23]: (319790, 60)

Missing values: There is no strict order in removing missing values. For some datasets, tackling first the features and than the samples might be better. Furthermore, the threshold at which you decide to drop missing values per feature or sample changes from dataset to dataset, and depends on what you intend to do with the dataset later on.

Also, until now we only addressed the big holes in the dataset, not yet how we would fill the smaller gaps.

2.0.3 Unwanted entries and recording errors

Another source of quality issues in a dataset can be due to unwanted entries or recording errors. It's important to distinguish such samples from simple outliers. While outliers are data points that are unusual for a given feature distribution, unwanted entries or recording errors are samples that shouldn't be there in the first place.

For example, a temperature recording of 45°C in Switzerland might be an outlier (as in 'very unusual'), while a recording at 90°C would be an error. Similarly, a temperature recording from the top of Mont Blanc might be physical possible, but most likely shouldn't be included in a dataset about Swiss cities.

Of course, detecting such errors and unwanted entries and distinguishing them from outliers is not always straight forward and depends highly on the dataset. One approach to this is to take a global view on the dataset and see if you can identify some very unusual patterns.

Numerical features

To plot this global view of the dataset, at least for the numerical features, you can use pandas' .plot() function and combine it with the following parameters:

```
lw=0: lw stands for line width. 0 means that we don't want to show any lines

marker=".": Instead of lines, we tell the plot to use . as markers for each data point

subplots=True: subplots tells pandas to plot each feature in a separate subplot

layout=(-1, 4): This parameter tells pandas how many rows and columns to use for the subplots.

figsize=(15, 30), markersize=1: To make sure that the figure is big enough we recommend to have
```

[24]:
```python
df_X.plot(lw=0, marker=".", subplots=True, layout=(-1, 4),
          figsize=(15, 30), markersize=1);
```


Vehicle_Reference_df_res
Vehicle_Type
Towing_and_Articulation
Vehicle_Manoeuvre
Vehicle_Location-Restricted_Lane
Junction_Location
Skidding_and_Overturning
Hit_Object_in_Carriageway
Vehicle_Leaving_Carriageway
Hit_Object_off_Carriageway
1st_Point_of_Impact
Was_Vehicle_Left_Hand_Drive?
Journey_Purpose_of_Driver
Sex_of_Driver
Age_of_Driver
Age_Band_of_Driver
Location_Easting_OSGR
1e6
Location_Northing_OSGR
Longitude
Latitude
Police_Force
Accident_Severity
Number_of_Vehicles
Number_of_Casualties
Day_of_Week
Local_Authority_(District)
1st_Road_Class
1st_Road_Number
Road_Type
Speed_limit
Junction_Detail
2nd_Road_Number
Pedestrian_Crossing-Human_Contro
Pedestrian_Crossing-Physical_Facilities
Light_Conditions
Weather_Conditions
Road_Surface_Conditions
Special_Conditions_at_Site
Carriageway_Hazards
Urban_or_Rural_Area
Did_Police_Officer_Attend_Scene_of_Accident
Vehicle_Reference_df
Casualty_Reference
Casualty_Class
Sex_of_Casualty
Age_of_Casualty
Age_Band_of_Casualty
Casualty_Severity
Pedestrian_Location
Pedestrian_Movement
Car_Passenger
Bus_or_Coach_Passenger
Pedestrian_Road_Maintenance_Worker
Casualty_Type
Casualty_Home_Area_Type

Each point in this figure is a sample (i.e. a row) in our dataset and each subplot represents a different feature. The y-axis shows the feature value, while the x-axis is the sample index. These kind of plots can give you a lot of ideas for data cleaning and EDA. Usually it makes sense to invest as much time as needed until your happy with the output of this visualization.

Non-numerical features

Identifying unwanted entries or recording errors on non-numerical features is a bit more tricky. Given that at this point, we only want to investigate the general quality of the dataset. So what we can do is take a general look at how many unique values each of these non-numerical features contain, and how often their most frequent category is represented.

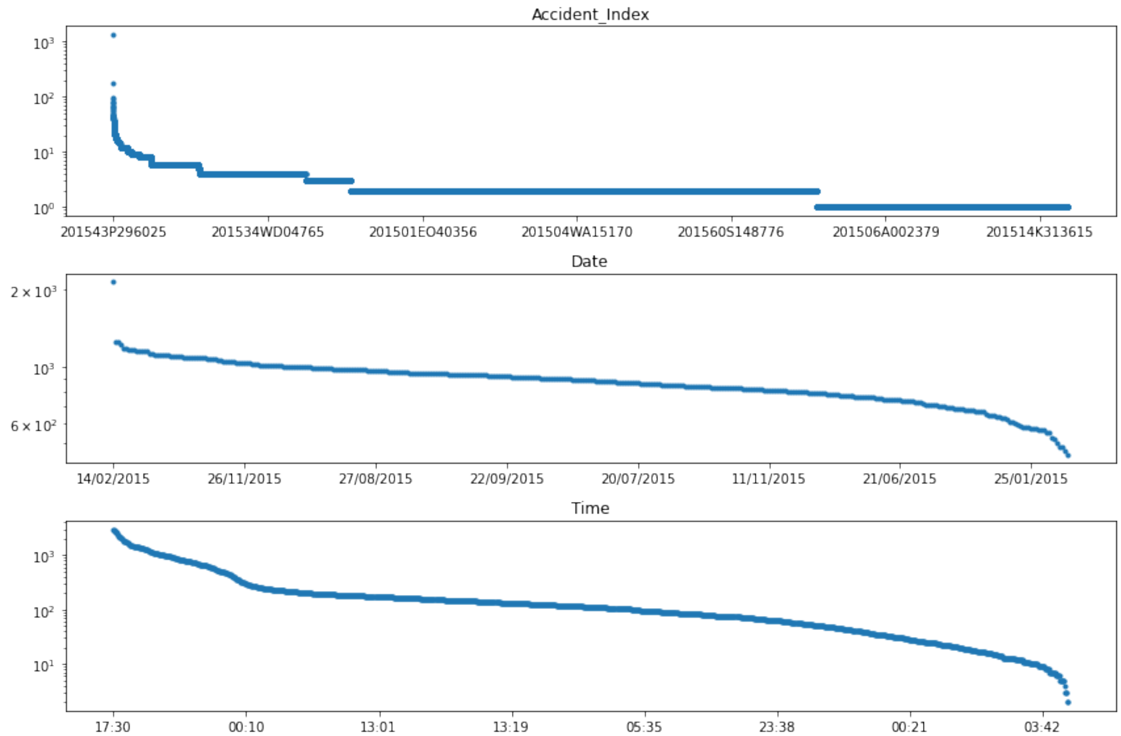
[25]:
```
# Extract descriptive properties of non-numerical features
df_X.describe(exclude=["number", "datetime"])
```

[25]:
```
       Accident_Index        Date    Time Local_Authority_(Highway)  \
count          319790      319790  319746                    319790
unique         123645         365    1439                       204
top      201543P296025  14/02/2015   17:30                 E10000017
freq             1332        2144    2969                      8457

       LSOA_of_Accident_Location
count                     298693
unique                     25977
top                    E01028497
freq                        1456
```

There are multiple ways for how you could potentially streamline the quality investigation for each individual non-numerical features. None of them is perfect, and all of them will require some follow up investigation. But for the purpose of showcasing one such a solution, what we could do is loop through all non-numerical features and plot for each of them the number of occurrences per unique value.

[26]:
```
# Create figure object with 3 subplots
fig, axes = plt.subplots(ncols=1, nrows=3, figsize=(12, 8))

# Identify non-numerical features
df_non_numerical = df_X.select_dtypes(exclude=["number", "datetime"])

# Loop through features and put each subplot on a matplotlib axis object
for col, ax in zip(df_non_numerical.columns, axes.ravel()):

    # Selects one single feature and counts number of occurrences per unique␣
 ↪value
    df_non_numerical[col].value_counts().plot(
```

```
        # Plots this information in a figure with log-scaled y-axis
        logy=True, title=col, lw=0, marker=".", ax=ax)

plt.tight_layout();
```



We can see that the most frequent accident (i.e. Accident_Index), had more than 100 people involved. Digging a bit deeper (i.e. looking at the individual features of this accident), we could identify that this accident happened on February 24th, 2015 at 11:55 in Cardiff UK. A quick internet search reveals that this entry corresponds to a luckily non-lethal accident including a minibus full of pensioners.

The decision for what should be done with such rather unique entries is once more left in the the subjective hands of the person analyzing the dataset. Without any good justification for WHY, and only with the intention to show you the HOW - let's go ahead and remove the 10 most frequent accidents from this dataset.

[27]:
```
# Collect entry values of the 10 most frequent accidents
accident_ids = df_non_numerical["Accident_Index"].value_counts().head(10).index

# Removes accidents from the 'accident_ids' list
df_X = df_X[~df_X["Accident_Index"].isin(accident_ids)]
df_X.shape
```

[27]: (317665, 60)

At the end of this second investigation, we should have a better understanding of the general quality of our dataset. We looked at duplicates, missing values and unwanted entries or recording errors. It is important to point out that we didn't discuss yet how to address the remaining missing values or outliers in the dataset. This is a task for the next investigation.

3 Content investigation

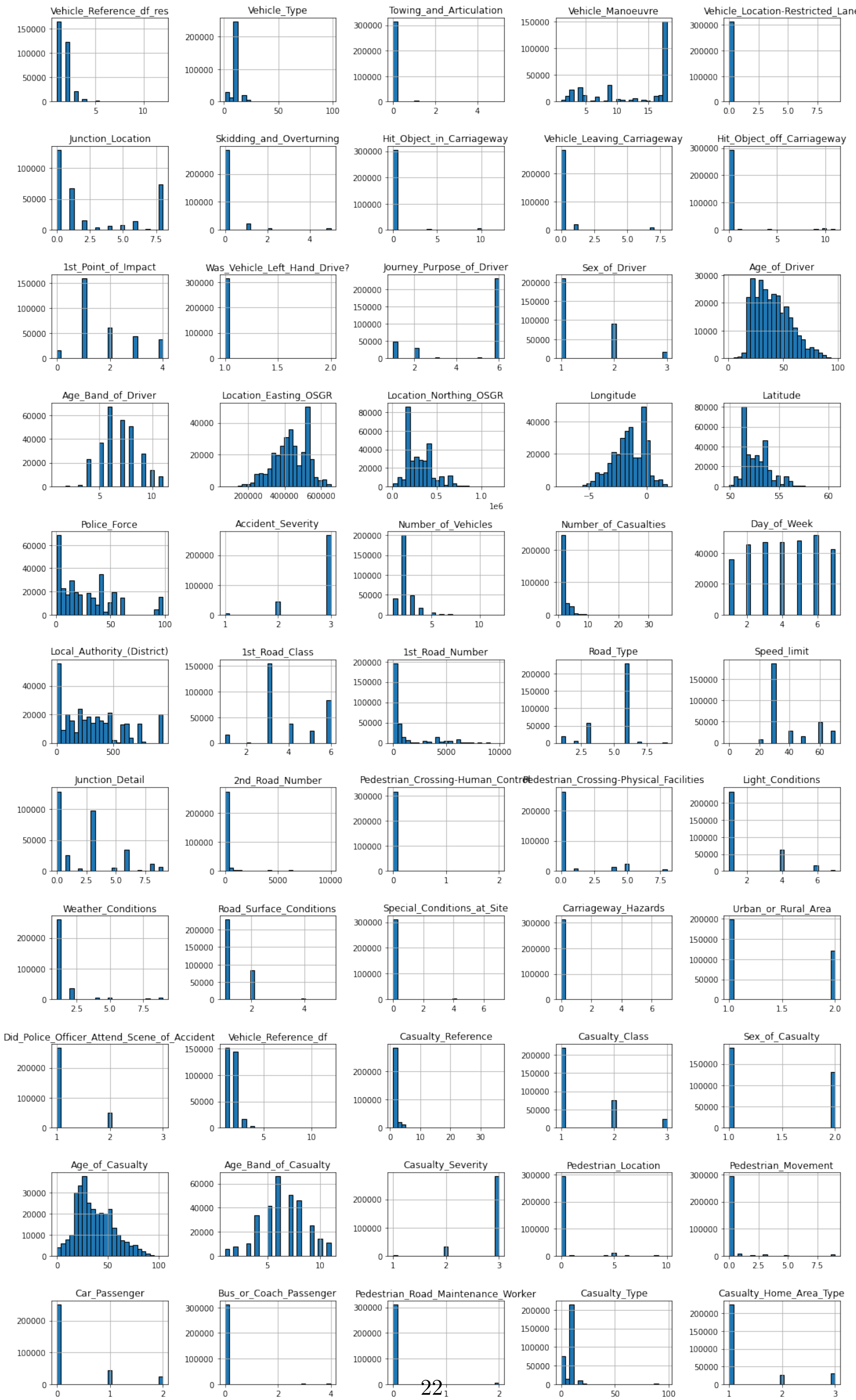
Up until now we only looked at the general structure and quality of the dataset. Let's now go a step further and take a look at the actual content. In an ideal setting, such an investigation would be done feature by feature. But this becomes very cumbersome once you have more than 20-30 features.

For this reason (and to keep this article as short as needed) we will explore three different approaches that can give you a very quick overview of the content stored in each feature and how they relate.

3.0.1 Feature distribution

Looking at the value distribution of each feature is a great way to better understand the content of your data. Furthermore, it can help to guide your EDA, and provides a lot of useful information with regards to data cleaning and feature transformation. The quickest way to do this for numerical features is using histogram plots. Luckily, pandas comes with a builtin histogram function that allows the plotting of multiple features at once.

[28]:
```
# Plots the histogram for each numerical feature in a separate subplot
df_X.hist(bins=25, figsize=(15, 25), layout=(-1, 5), edgecolor="black")
plt.tight_layout();
```

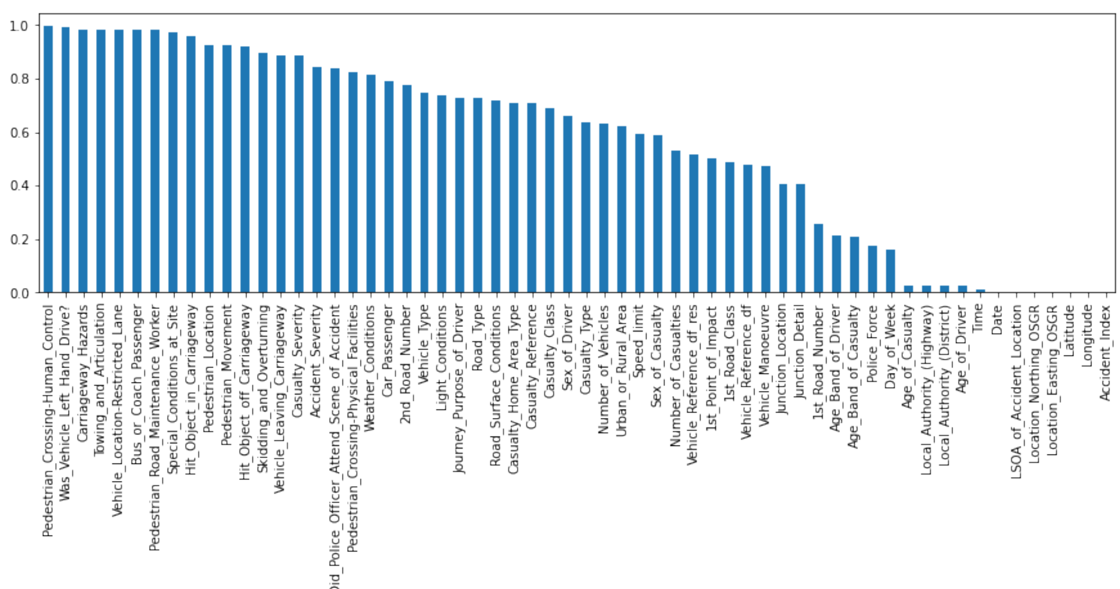

Vehicle_Reference_df_res
Vehicle_Type
Towing_and_Articulation
Vehicle_Manoeuvre
Vehicle_Location-Restricted_Lane
Junction_Location
Skidding_and_Overturning
Hit_Object_in_Carriageway
Vehicle_Leaving_Carriageway
Hit_Object_off_Carriageway
1st_Point_of_Impact
Was_Vehicle_Left_Hand_Drive?
Journey_Purpose_of_Driver
Sex_of_Driver
Age_of_Driver
Age_Band_of_Driver
Location_Easting_OSGR
Location_Northing_OSGR
Longitude
Latitude
Police_Force
Accident_Severity
Number_of_Vehicles
Number_of_Casualties
Day_of_Week
Local_Authority_(District)
1st_Road_Class
1st_Road_Number
Road_Type
Speed_limit
Junction_Detail
2nd_Road_Number
Pedestrian_Crossing-Human_Control
Pedestrian_Crossing-Physical_Facilities
Light_Conditions
Weather_Conditions
Road_Surface_Conditions
Special_Conditions_at_Site
Carriageway_Hazards
Urban_or_Rural_Area
Did_Police_Officer_Attend_Scene_of_Accident
Vehicle_Reference_df
Casualty_Reference
Casualty_Class
Sex_of_Casualty
Age_of_Casualty
Age_Band_of_Casualty
Casualty_Severity
Pedestrian_Location
Pedestrian_Movement
Car_Passenger
Bus_or_Coach_Passenger
Pedestrian_Road_Maintenance_Worker
Casualty_Type
Casualty_Home_Area_Type

There are a lot of very interesting things visible in this plot. For example…

Most frequent entry: Some features, such as Towing_and_Articulation or Was_Vehicle_Left_Hand_Drive? mostly contain entries of just one category. Using the .mode() function, we could for example extract the ratio of the most frequent entry for each feature and visualize that information.

[29]:
```
# Collects for each feature the most frequent entry
most_frequent_entry = df_X.mode()

# Checks for each entry if it contains the most frequent entry
df_freq = df_X.eq(most_frequent_entry.values, axis=1)

# Computes the mean of the 'is_most_frequent' occurrence
df_freq = df_freq.mean().sort_values(ascending=False)

# Show the 5 top features with the highest ratio of singular value content
display(df_freq.head())

# Visualize the 'df_freq' table
df_freq.plot.bar(figsize=(15, 4));
```

```
Pedestrian_Crossing-Human_Control     0.995259
Was_Vehicle_Left_Hand_Drive?          0.990137
Carriageway_Hazards                   0.983646
Towing_and_Articulation               0.983221
Vehicle_Location-Restricted_Lane      0.982088
dtype: float64
```



Skewed value distributions: Certain kind of numerical features can also show strongly non-gaussian distributions. In that case you might want to think about how you can transform these values to make them more normal distributed. For example, for right skewed data you could use a log-transformation.

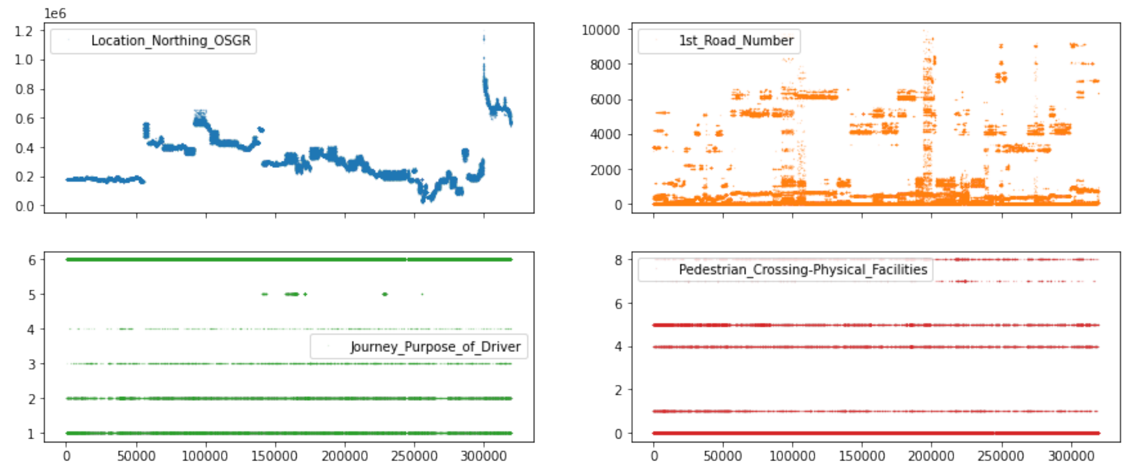
3.0.2 Feature patterns

Next step on the list is the investigation of feature specific patterns. The goal of this part is two fold:

```
1) Can we identify particular patterns within a feature that will help us to decide if some en

2) Can we identify particular relationships between features that will help us to better unders
```

But before we dive into these two questions, let's take a closer look at a few 'randomly selected' features.

[30]:
```
df_X[["Location_Northing_OSGR", "1st_Road_Number",
      "Journey_Purpose_of_Driver", "Pedestrian_Crossing-Physical_Facilities"]].
↪plot(
    lw=0, marker=".", subplots=True, layout=(-1, 2), markersize=0.1,␣
↪figsize=(15, 6));
```



In the top row, we can see features with continuous values (e.g. seemingly any number from the number line), while in the bottom row we have features with discrete values (e.g. 1, 2, 3 but not 2.34).

While there are many ways we could explore our features for particular patterns, let's simplify our option by deciding that we treat features with less than 25 unique features as discrete or ordinal features, and the other features as continuous features.

[31]:
```python
# Creates mask to identify numerical features with more or less than 25 unique␣
 ↪features
cols_continuous = df_X.select_dtypes(include="number").nunique() >= 25
```

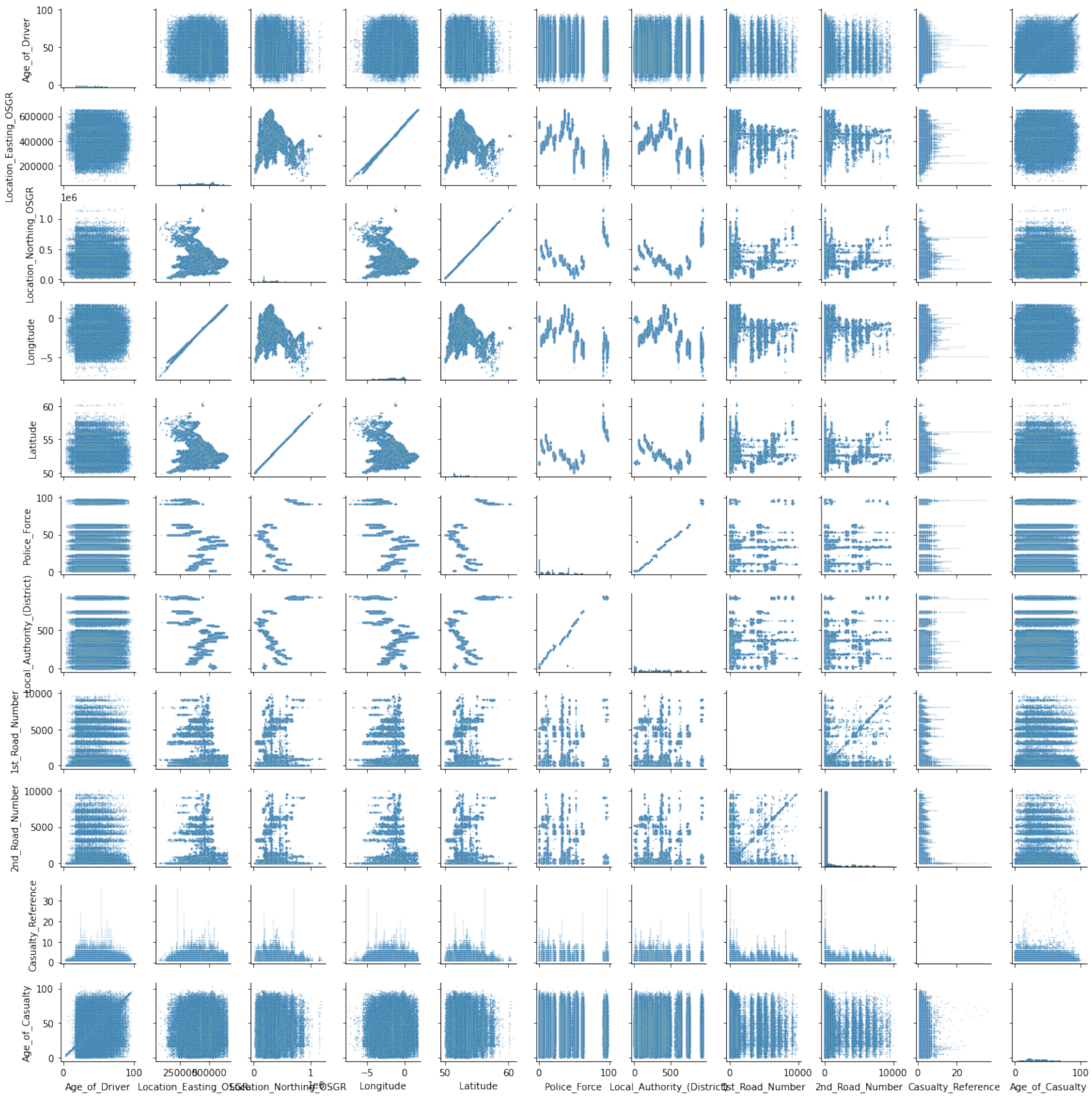
Continuous features Now that we have a way to select the continuous features, let's go ahead and use seaborn's pairplot to visualize the relationships between these features. Important to note, seaborn's pairplot routine can take a long time to create all subplots. Therefore we recommend to not use it for more than ~10 features at a time.

[32]:
```python
# Create a new dataframe which only contains the continuous features
df_continuous = df_X[cols_continuous[cols_continuous].index]
df_continuous.shape
```

[32]:
```
(317665, 11)
```

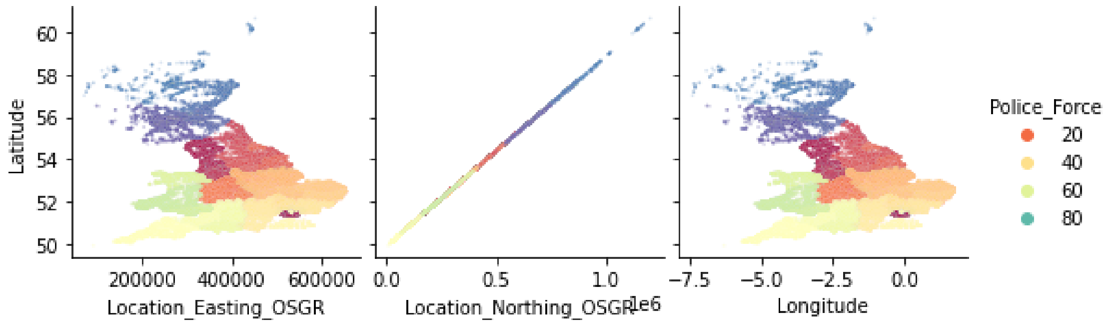
Given that in our case we only have 11 features, we can go ahead with the pairplot. Otherwise, using something like df_continuous.iloc[:, :5] could help to reduce the number of features to plot.

[33]:
```python
import seaborn as sns

sns.pairplot(df_continuous, height=1.5, plot_kws={"s": 2, "alpha": 0.2});
```



There seems to be a strange relationship between a few features in the top left corner. Location_Easting_OSGR and Longitude, as well as Location_Easting_OSGR and Latitude seem to have a very strong linear relationship.

[34]:
```
sns.pairplot(
    df_X, plot_kws={"s": 3, "alpha": 0.2}, hue="Police_Force",␣
↪palette="Spectral",
    x_vars=["Location_Easting_OSGR", "Location_Northing_OSGR", "Longitude"],
    y_vars="Latitude");
```



Knowing that these features contain geographic information, a more in-depth EDA with regards to geolocation could be fruitful. However, for now we will leave the further investigation of this pairplot to the curious reader and continue with the exploration of the discrete and ordinal features.

Discrete and ordinal features

Finding patterns in the discrete or ordinal features is a bit more tricky. But also here, some quick pandas and seaborn trickery can help us to get a general overview of our dataset. First, let's select the columns we want to investigate.

[35]:
```
# Create a new dataframe which doesn't contain the numerical continuous features
df_discrete = df_X[cols_continuous[~cols_continuous].index]
df_discrete.shape
```

[35]: (317665, 44)

As always, there are multiple way for how we could investigate all of these features. Let's try one example, using seaborn's stripplot() together with a handy zip() for-loop for subplots.

Note, to spread the values out in the direction of the y-axis we need to chose one particular (hopefully informative) feature. While the 'right' feature can help to identify some interesting patterns, usually any continuous feature should do the trick. The main interest in this kind of plot is to see how many samples each discrete value contains.

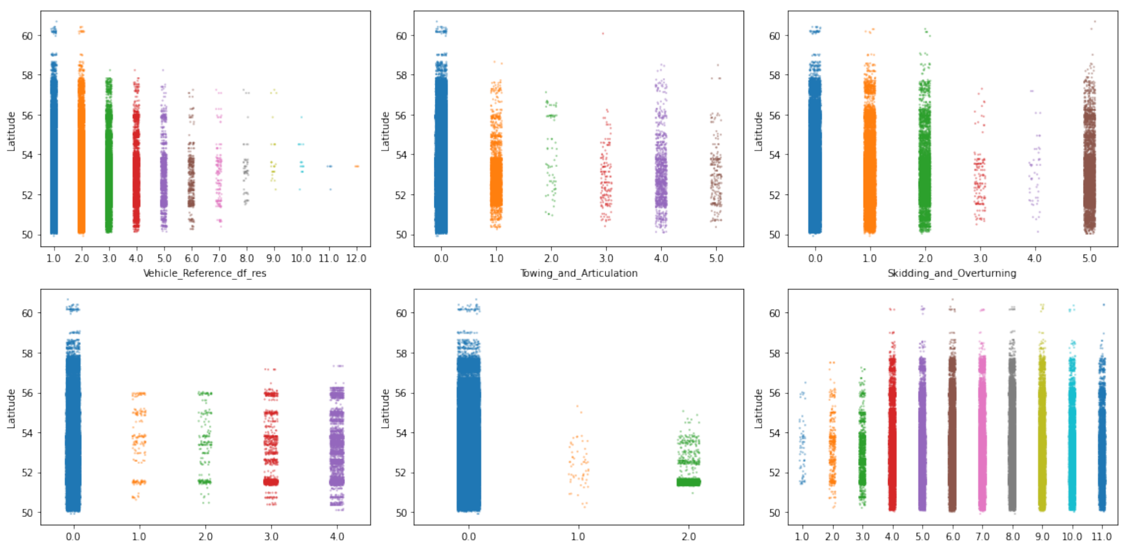
[36]:
```
import numpy as np

# Establish number of columns and rows needed to plot all features
n_cols = 5
n_elements = len(df_discrete.columns)
n_rows = np.ceil(n_elements / n_cols).astype("int")

# Specify y_value to spread data (ideally a continuous feature)
y_value = df_X["Age_of_Driver"]

# Create figure object with as many rows and columns as needed
fig, axes = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(15, n_rows * 2.5))
```

```python
# Loop through features and put each subplot on a matplotlib axis object
for col, ax in zip(df_discrete.columns, axes.ravel()):
    sns.stripplot(data=df_X, x=col, y=y_value, ax=ax, palette="tab10", size=1, alpha=0.5)
plt.tight_layout();
```


Age_of_Driver
Vehicle_Reference_df_res
Vehicle_Type
Towing_and_Articulation
Vehicle_Manoeuvre
Vehicle_Location-Restricted_Lane
Junction_Location
Skidding_and_Overturning
Hit_Object_in_Carriageway
Vehicle_Leaving_Carriageway
Hit_Object_off_Carriageway
1st_Point_of_Impact
Was_Vehicle_Left_Hand_Drive?
Journey_Purpose_of_Driver
Sex_of_Driver
Age_Band_of_Driver
Accident_Severity
Number_of_Vehicles
Number_of_Casualties
Day_of_Week
1st_Road_Class
Road_Type
Speed_limit
Junction_Detail
Pedestrian_Crossing-Human_Control
Pedestrian_Crossing-Physical_Facilities
Light_Conditions
Weather_Conditions
Road_Surface_Conditions
Special_Conditions_at_Site
Carriageway_Hazards
Urban_or_Rural_Area
Did_Police_Officer_Attend_Scene_of_Accident
Vehicle_Reference_df
Casualty_Class
Sex_of_Casualty
Age_Band_of_Casualty
Casualty_Severity
Pedestrian_Location
Pedestrian_Movement
Car_Passenger
Bus_or_Coach_Passenger
Pedestrian_Road_Maintenance_Worker
Casualty_Type
Casualty_Home_Area_Type

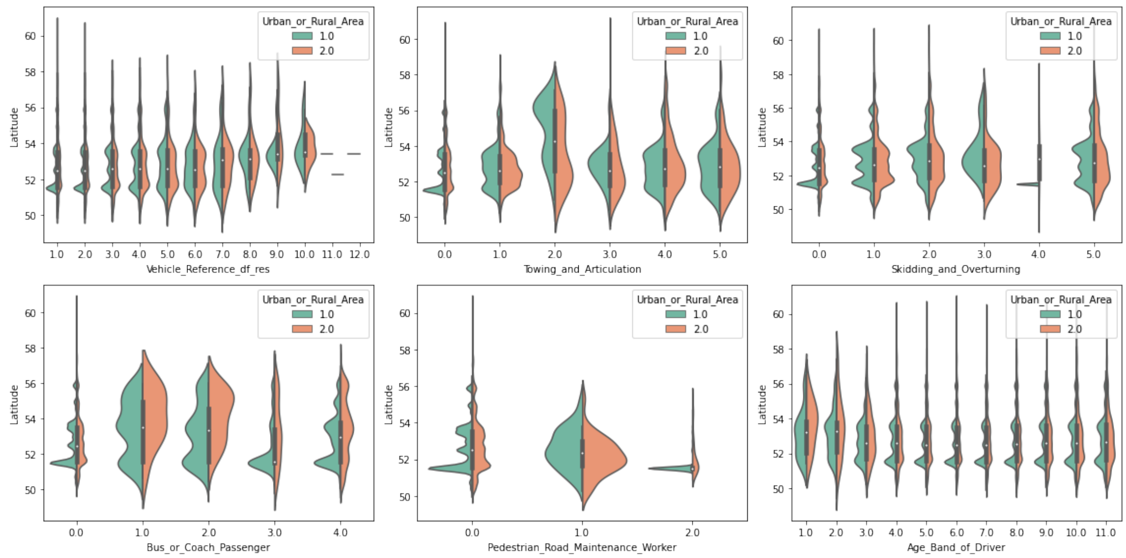
There are too many things to comment here, so let's just focus on a few. In particular, let's focus on 6 features where the values appear in some particular pattern or where some categories seem to be much less frequent than others. And to shake things up a bit, let's now use the Longitude feature to stretch the values over the y-axis.

[37]:
```
# Specify features of interest
selected_features = ["Vehicle_Reference_df_res", "Towing_and_Articulation",
                     "Skidding_and_Overturning", "Bus_or_Coach_Passenger",
                     "Pedestrian_Road_Maintenance_Worker", "Age_Band_of_Driver"]

# Create a figure with 3 x 2 subplots
fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(16, 8))

# Loop through these features and plot entries from each feature against␣
↪`Latitude`
for col, ax in zip(selected_features, axes.ravel()):
    sns.stripplot(data=df_X, x=col, y=df_X["Latitude"], ax=ax,
                  palette="tab10", size=2, alpha=0.5)
plt.tight_layout();
```



These kind of plots are already very informative, but they obscure regions where there are a lot of data points at once. For example, there seems to be a high density of points in some of the plots at the 52nd latitude. So let's take a closer look with an appropriate plot, such as violineplot (or boxenplot or boxplot for that matter). And to go a step further, let's also separate each visualization by Urban_or_Rural_Area.

[38]:
```
# Create a figure with 3 x 2 subplots
fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(16, 8))

# Loop through these features and plot entries from each feature against␣
↪`Latitude`
for col, ax in zip(selected_features, axes.ravel()):
    sns.violinplot(data=df_X, x=col, y=df_X["Latitude"], palette="Set2",
                   split=True, hue="Urban_or_Rural_Area", ax=ax)
plt.tight_layout();
```



Interesting! We can see that some values on features are more frequent in urban, than in rural areas (and vice versa). Furthermore, as suspected, there seems to be a high density peak at latitude 51.5. This is very likely due to the more densely populated region around London (at 51.5074°).

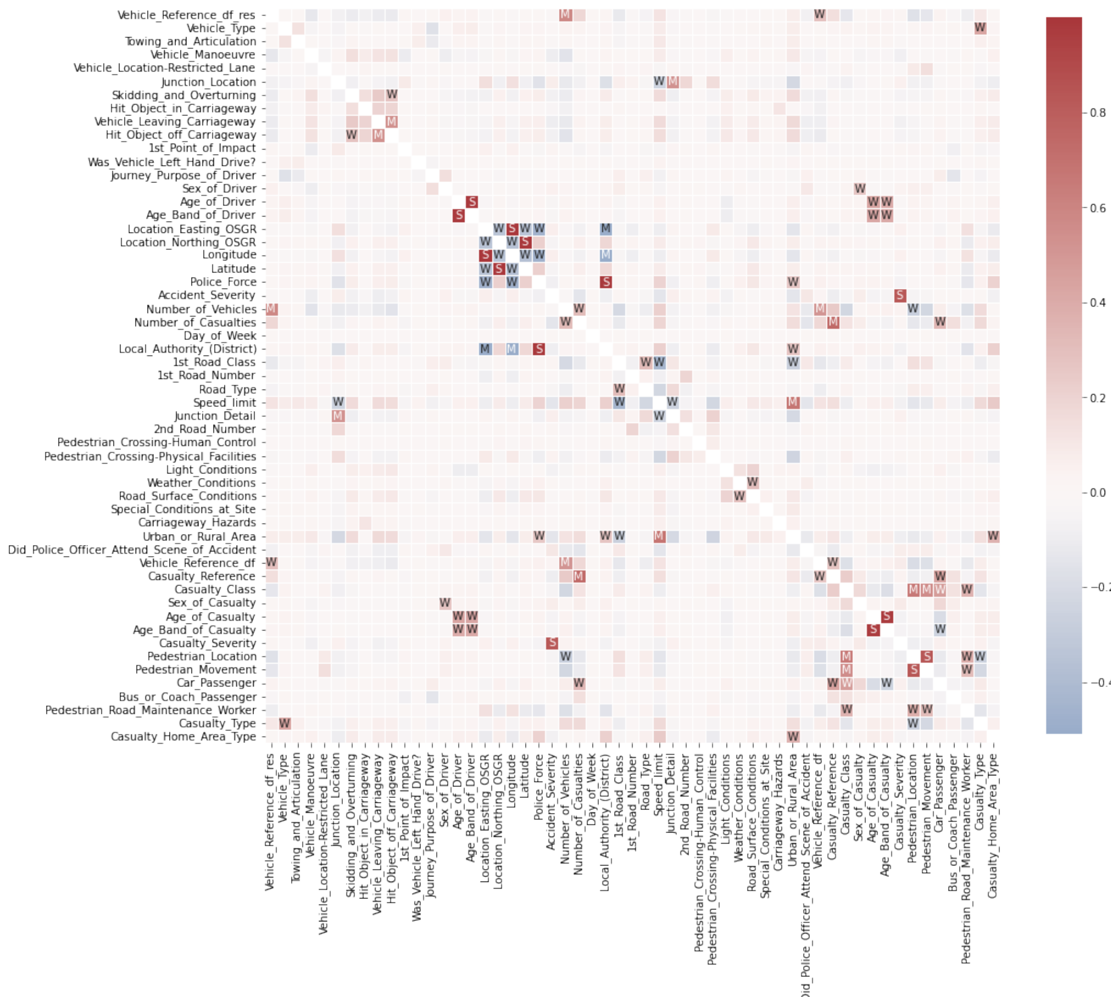
3.0.3 Feature relationships

Last, but not least, let's take a look at relationships between features. More precisely how they correlate. The quickest way to do so is via pandas' .corr() function. So let's go ahead and compute the feature to feature correlation matrix for all numerical features.

[39]:
```
# Computes feature correlation
df_corr = df_X.corr(method="pearson")
```

Note: Depending on the dataset and the kind of features (e.g. ordinal or continuous features) you might want to use the spearman method instead of the pearson method to compute the correlation. Whereas the Pearson correlation evaluates the linear relationship between two continuous variables, the Spearman correlation evaluates the monotonic relationship based on the ranked values for each feature. And to help with the interpretation of this correlation matrix, let's use seaborn's .heatmap() to visualize it.

```
[40]: # Create labels for the correlation matrix
labels = np.where(np.abs(df_corr)>0.75, "S",
                  np.where(np.abs(df_corr)>0.5, "M",
                           np.where(np.abs(df_corr)>0.25, "W", "")))

# Plot correlation matrix
plt.figure(figsize=(15, 15))
sns.heatmap(df_corr, mask=np.eye(len(df_corr)), square=True,
            center=0, annot=labels, fmt='', linewidths=.5,
            cmap="vlag", cbar_kws={"shrink": 0.8});
```



This looks already very interesting. We can see a few very strong correlations between some of the features. Now, if you're interested actually ordering all of these different correlations, you could do something like this:

[41]:
```
#  Creates a mask to remove the diagonal and the upper triangle.
lower_triangle_mask = np.tril(np.ones(df_corr.shape), k=-1).astype("bool")

#  Stack all correlations, after applying the mask
df_corr_stacked = df_corr.where(lower_triangle_mask).stack().sort_values()

#  Showing the lowest and highest correlations in the correlation matrix
display(df_corr_stacked)
```

```
Local_Authority_(District)  Longitude                -0.509343
                            Location_Easting_OSGR    -0.502919
Police_Force                Longitude                -0.471327
                            Location_Easting_OSGR    -0.461112
Speed_limit                 1st_Road_Class           -0.438931
                                                        …
Age_Band_of_Casualty        Age_of_Casualty           0.974397
Age_Band_of_Driver          Age_of_Driver             0.979019
Local_Authority_(District)  Police_Force              0.984819
Longitude                   Location_Easting_OSGR     0.999363
Latitude                    Location_Northing_OSGR    0.999974
Length: 1485, dtype: float64
```

As you can see, the investigation of feature correlations can be very informative. But looking at everything at once can sometimes be more confusing than helpful. So focusing only on one feature with something like df_X.corrwith(df_X["Speed_limit"]) might be a better approach.

Furthermore, correlations can be deceptive if a feature still contains a lot of missing values or extreme outliers. Therefore, it is always important to first make sure that your feature matrix is properly prepared before investigating these correlations.

At the end of this third investigation, we should have a better understanding of the content in our dataset. We looked at value distribution, feature patterns and feature correlations. However, these are certainly not all possible content investigation and data cleaning steps you could do. Additional steps would for example be outlier detection and removal, feature engineering and transformation, and more.

[ ]:
```

```