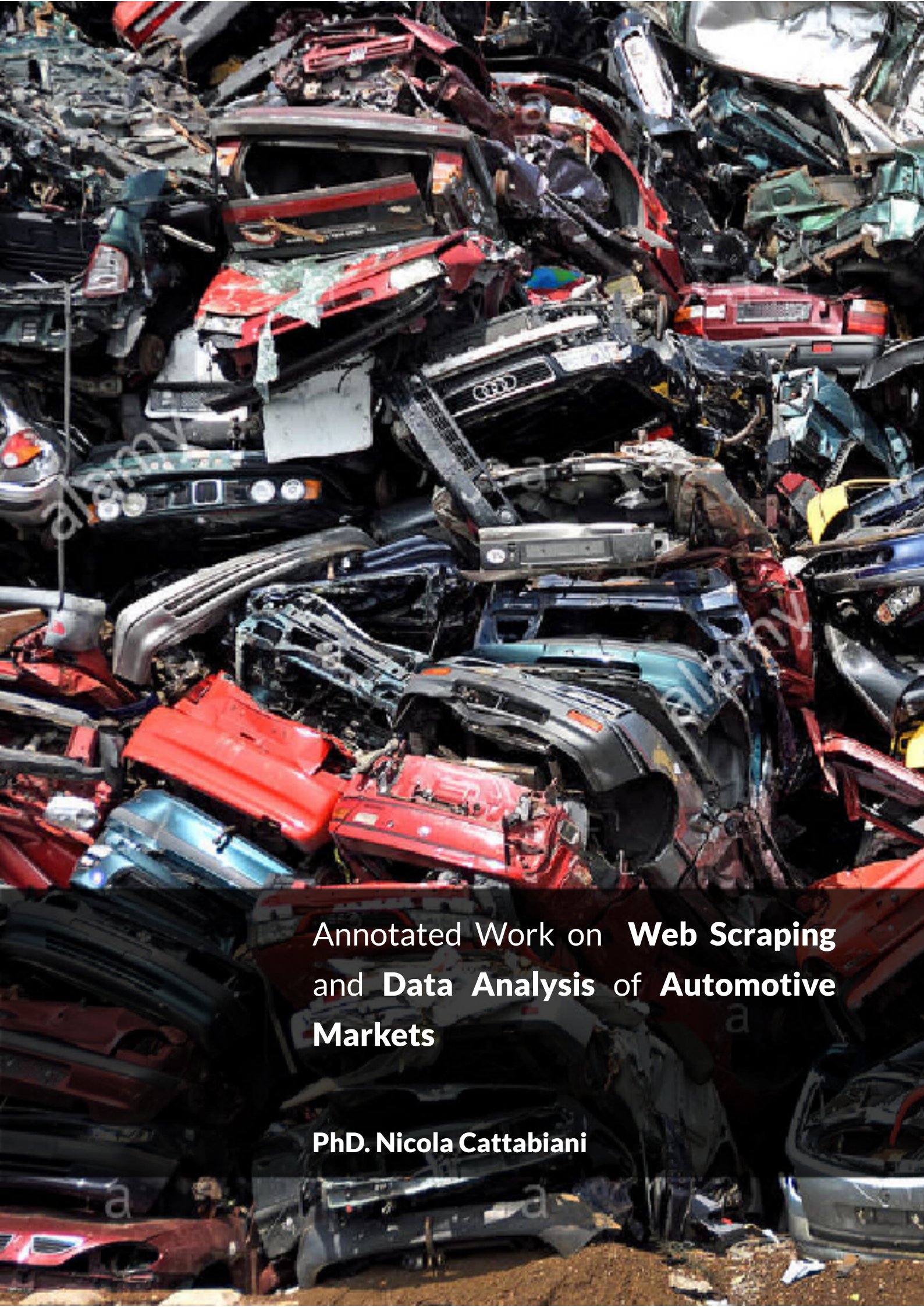


Annotated Work on **Web Scraping** and **Data Analysis** of **Automotive Markets**

PhD. Nicola Cattabiani

Contents

1 SCRAPING OF WEB MARKETPLACES

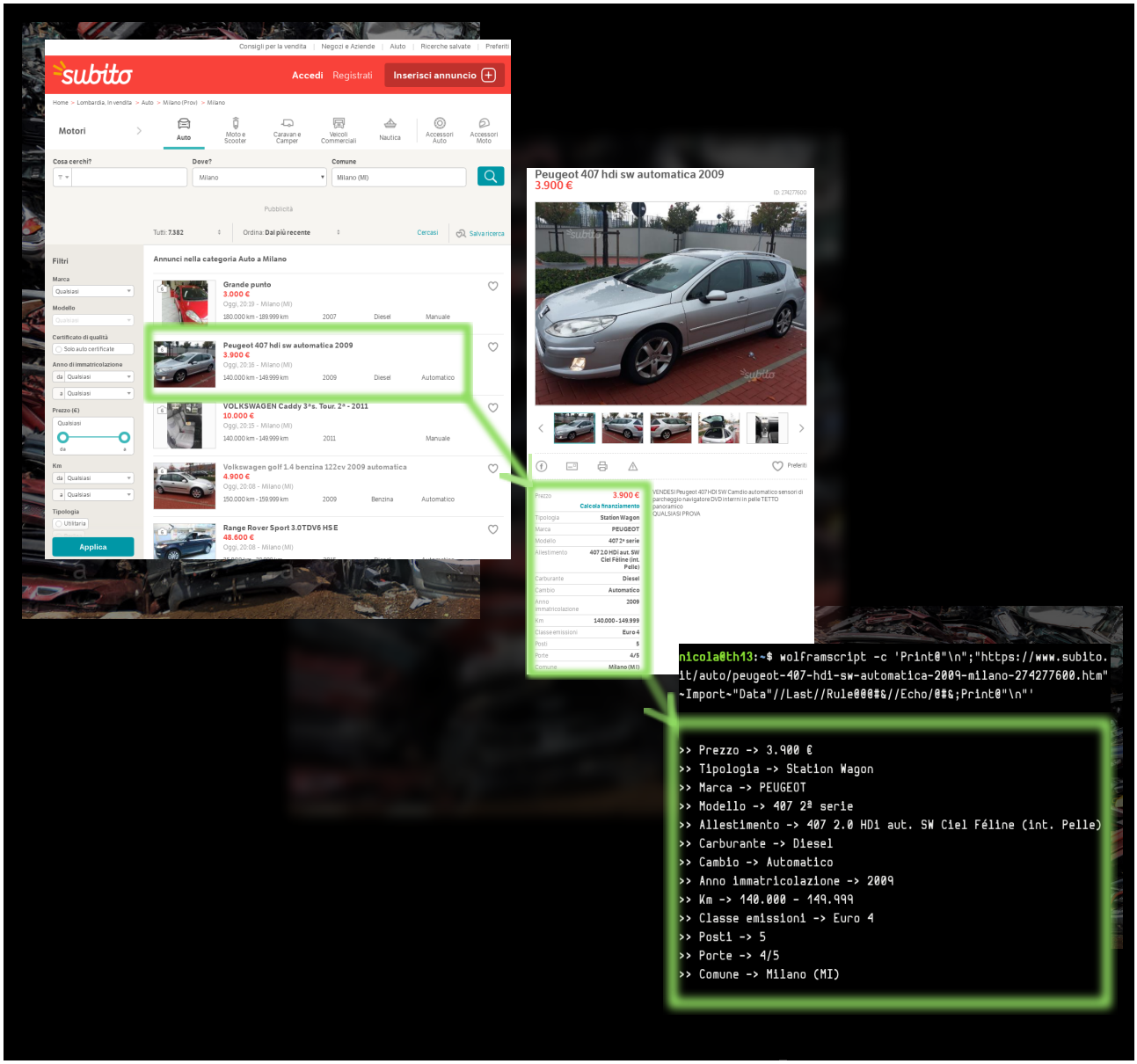
The goal of this work is to gain insights on the second-hand car market by means of data visualization and statistical modeling. The data are stored in a Relational Database (RDB) obtained through scraping the ads on *Subito.it*.

The document is organized in sections of different themes that also contain code snippets. In every section we have brief descriptions that can be read independently of the code, while the headers of the snippets highlight implementation details that facilitate the understanding of the code.

Most of the code is written in the Wolfram Language with few additional Bash and Linux command line utilities.

1.1 DATA COLLECTION

The figure below describes the first step of the scraping process, that is retrieving the data we need and store that data as an entry of a RDB.



In this section we scrape the Milan area. Scraping the whole Italian region has a slightly more convoluted implementation and it will be discussed in the statistical modeling section §3. Nevertheless, both implementations are based on these three steps:

1. Define the list of search result pages that contains the links to the ads. The addresses of these pages are constructed by joining the URL of the latest updated page with an API key.

SNIPPET 1: The API key is "?o=n" where the page index n goes from 1 to highest index found on the latest updated page.

```
resultPages =
        "https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano" //
        {#, Max[ToExpression /@ #~Import~"Hyperlinks"~
                StringCases~("?o=" ~~ x__ → x)]} & //
        {#1~ConstantArray~#2, "/?o=" <> # & /@ ToString /@ Range@#2} & @@ # & //
        StringJoin @@@ Thread@# &
```

```
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=1
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=2
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=3
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=4
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=5
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=6
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=7
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=8
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=9
...
...
https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano/?o=250
```

2. From every page listed in resultPages, extract the links to the ads.

SNIPPET 2: The wildcard pattern that is common to the URLs of the ads is:

```
__ ~~ auto ~~ __ ~~ .htm
```

```
adLinks = #~Import~"Hyperlinks"~
          StringCases~(__ ~~ "auto" ~~ __ ~~ ".htm")~Level~{2} &~
          ParallelMap~resultPages //
           Flatten // DeleteDuplicates
```

Here are the latest 10 ads found:

```
https://www.subito.it/auto/nissan-qashqai-1-5-dci-n-tec-milano-275125072.htm
https://www.subito.it/auto/fiesta-1-4-tdci-milano-275124466.htm
https://www.subito.it/auto/smart-fortwo-1000-mhd-passion-doppi-gomme-milano-275124371.htm
https://www.subito.it/auto/nissan-almera-2-2-2002-unico-prop-121-000km-milano-275124358.htm
https://www.subito.it/auto/hyundai-tucson-2-0-diesel-crdi-anno-2010-milano-274617357.htm
https://www.subito.it/auto/audi-a1-1-2-tfsi-ambition-milano-273973112.htm
https://www.subito.it/auto/land-rover-range-rover-sport-2-7-tdv6-tds-milano-272534592.htm
https://www.subito.it/auto/volvo-v60-2015-navigatore-sensor-parch-manuale-milano-273019323.htm
https://www.subito.it/auto/fiat-sedici-2-0-mjt-16v-dpf-4x4-dynamic-milano-275123237.htm
https://www.subito.it/auto/ford-ka-1-2-8v-69cv-titanium-milano-275122538.htm
. .
. .
```

The search result pages containes 33 ads per page, that is a total of $\simeq 8000$ ads in Milan.

3. Fetch the HTML tables.

SNIPPET 3: By importing the previous URLs as Data we effectively fetch just the HTML tables

```
scrapeData = #~Import~"Data"&~ParallelMap~adLinks
```

The script ran on a AWS c5.2xlarge instance on 8 parallel threads. In Fig.1 we see a summary of the data collection job, including the server machine specs and a print of the script. The job completed in approximately 5 minutes. On a Raspberry Pi 3 Model B connected to a (slow) home network and parallelized on its 4 ARM cores the timing has been 10 times the AWS one.

```
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ uname -srv
Linux 4.4.0-1066-aws #76-Ubuntu SMP Thu Aug 16 16:21:21 UTC 2018
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ lscpu | head -16
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz
Stepping:              4
CPU MHz:               3000.000
BogoMIPS:              6000.00
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ ls -lh
total 4.0K
-rw-r--r-- 1 root root 813 Nov 26 20:43 gba.wls
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ cat gba.wls
LaunchKernels@8;

resultPages = Table[#1<>"/?o="<>ToString@i,{i,Range@#2}]&@@{
              #,
              Max[ToExpression/@#~Import~"Hyperlinks"~StringCases~("?o="~~x__->x)~
              Level~{2}]}&@
"https://www.subito.it/annunci-lombardia/vendita/auto/lombardia/milano"

fullinks = #~Import~"Hyperlinks"~
           StringCases~(__~~"auto"~~__~~".htm")~Level~{2}&~
           ParallelMap~resultPages//
           Flatten//DeleteDuplicates//
     AbsoluteTiming//(Echo[
     ToString@Length@Last@#<>" URLs retreived in "<>ToString@First@#<>" seconds"];
     Last@#)&

fulldata = Import[#,"Data"]&~ParallelMap~fullinks//
     AbsoluteTiming//(Echo[
     ToString@Length@Last@#<>" HTML tables saved in "<>ToString@First@#<>" seconds"];
     Last@#)&

Export["fulldata.csv",fulldata]//Echo

CloseKernels[];
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ wolframscript -f gba.wls
>> 7483 URLs retreived in 79.6639 seconds
>> 7483 HTML tables saved in 268.181 seconds
>> fulldata.csv
ubuntu@ip-172-31-42-148:~/Wolfram/GBA$ ls -lhtr
total 3.8M
-rw-r--r-- 1 root   root    813 Nov 26 20:43 gba.wls
-rw-rw-rw- 1 ubuntu ubuntu 3.8M Nov 26 20:54 fulldata.csv
```

FIGURE 1: DATA COLLECTION LOG

1.2 CLEAN-UP AND STRUCTURING

The data we fetched in SNIPPET 3 presents some issues like spurious entries, inconsistent types, missing keys and missing values. The next snippet corrects all these irregularities and returns a queryable database named milanDB; Fig.2 is a small extract sorted by price.

SNIPPET 4: Create a RDB from the fetched data. The file fulldata.csv contains lists that are parsed into key-values pairs by means of the **Rule** function; missing values are explicitly added by **KeyUnion** and **Dataset** wrapper allows SQL-*like* operations. For instance, we modify the values and their types along the "Anno immatricolazione" column according to a rule

$$String : \text{“Anni 60”} \rightarrow Real : 1960.0$$

or the "Km" columns according to

$$String : \text{“10.000 - 20.000”} \rightarrow Real : 15000.0$$

and so on.

```
milanDB = Import@"fulldata.csv"
          Last /@ ToExpression@# & //
          Select[#, Head@# == List &] & /@ # & //
          Rule @@@ # & /@ # & // KeyUnion //
          Dataset[#][All, {
                   "Modello" → (If[MissingQ@#, #, ToString@#] &),

                    "Prezzo" → (If[MissingQ@#, #,
                                 #~StringDelete~{".", "€"} //
                                 ToExpression // N] &),

                       "Km" → (If[MissingQ@#, #,
                                  #~StringDelete~"."~StringSplit~" - " //
                                  ToExpression // N // Mean ] &),

    "Anno immatricolazione" → (If[Not@StringQ@#, #,
                                  StringReplace[#, "Anni " → "19"] //
                                  ToExpression // N] &)
        ×
                    "Cambio" → (Which[MissingQ@#, "Manuale",
                                      # == "Altro", "AltroCambio",
                                      True, #] &),

                "Carburante" → (Which[MissingQ@#, "Benzina",
                                      # == "Altro", "AltroCarb",
                                      True, #] &)
    }] &
```

Prezzo	Tipologia	Marca	Modello	Allestimento	Carburante	Cambio	Anno immatricolazione	Km	Classe emissioni	Posti	Porte	Comune
4 300 000	Coupé	BUGATTI	—	—	Benzina	—	2018	0.	—	2	—	Milano (MI)
4 200 000	Altro	BUGATTI	—	—	Benzina	Sequenziale	2018	2499.5	—	2	2/3	Milano (MI)
4 000 000	Coupé	BUGATTI	—	—	Benzina	—	2018	2499.5	—	2	—	Milano (MI)
2 999 999	—	ALFA ROMEO	33	—	—	—	1960	2499.5	—	—	—	Milano (MI)
350 000	Cabrio	PORSCHE	—	—	Benzina	—	2011	2499.5	Euro 5	2	2/3	Milano (MI)
255 000	Cabrio	FERRARI	488	488 Spider	—	Automatico	2017	2499.5	Euro 6	2	2/3	Milano (MI)
200 000	—	FERRARI	458	458 Spider DCT	—	—	2013	12 499.5	—	—	—	Milano (MI)
195 000	Coupé	MERCEDES	—	—	Benzina	Automatico	2018	0.	Euro 6	2	2/3	Milano (MI)
190 000	Coupé	FERRARI	512i BB	512i BB	Benzina	Manuale	1980	77 499.5	Pre–Euro	2	2/3	Milano (MI)
150 000	Coupé	Altro	—	—	Benzina	Automatico	2010	52 499.5	Euro 5	5	2/3	Milano (MI)
142 200	Berlina	Altro	—	—	Altro	Automatico	2017	62 499.5	—	5	4/5	Milano (MI)
140 000	Cabrio	PORSCHE	—	—	—	Manuale	1960	2499.5	—	—	—	Milano (MI)
138 950	Berlina	Altro	—	—	Altro	Automatico	2017	12 499.5	—	5	4/5	Milano (MI)
135 000	Station Wagon	AUDI	A6 5ª serie	—	Benzina	Automatico	2016	57 499.5	Euro 6	5	4/5	Milano (MI)
134 000	Cabrio	PORSCHE	911 (991)	911 3.0 Carrera 4S Cabriolet	Benzina	Automatico	2017	27 499.5	—	4	2/3	Milano (MI)
131 000	Altro	BMW	—	—	Benzina	Automatico	2018	0.	—	5	—	Milano (MI)
129 500	Cabrio	PORSCHE	911 (991)	911 3.0 Targa 4S	Benzina	Automatico	2016	12 499.5	Euro 6	4	—	Milano (MI)
129 500	Cabrio	PORSCHE	911 (991)	911 3.0 Targa 4S	Benzina	Automatico	2016	12 499.5	Euro 6	4	—	Milano (MI)
127 990	Cabrio	PORSCHE	911 (991)	911 3.0 Targa 4	Benzina	Automatico	2016	22 499.5	Euro 6	4	—	Milano (MI)
127 990	Cabrio	PORSCHE	911 (991)	911 3.0 Targa 4	Benzina	Automatico	2016	22 499.5	Euro 6	4	—	Milano (MI)
123 500	Coupé	PORSCHE	911 (991)	911 3.8 GT3	Benzina	Automatico	2016	52 499.5	Euro 6	2	2/3	Milano (MI)
116 000	Coupé	PORSCHE	911 (991)	911 3.8 Carrera GTS Coupé	Benzina	Automatico	2015	17 499.5	Euro 6	4	2/3	Milano (MI)
110 000	Fuoristrada/SUV	MERCEDES	Classe G(G461/G463)	G 63 AMG S.W. G Force	Benzina	Automatico	2016	7499.5	Euro 6	5	4/5	Milano (MI)
103 000	Coupé	PORSCHE	911 (997)	911 Turbo S Coupé	Benzina	Automatico	2010	57 499.5	Euro 5	4	—	Milano (MI)
99 999	Coupé	HYUNDAI	Veloster	Veloster 1.6 GDI DCT Sport	Benzina	Manuale	2011	7499.5	Euro 2	4	2/3	Milano (MI)
99 900	Fuoristrada/SUV	MERCEDES	Classe G(G461/G463)	G 63 AMG S.W.	Benzina	Automatico	2015	47 499.5	—	5	4/5	Milano (MI)
98 000	Coupé	MERCEDES	GT (C/R190)	GT AMG	Benzina	Automatico	2017	17 499.5	Euro 6	2	—	Milano (MI)
97 000	Coupé	ASTON MARTIN	Rapide S	Rapide S Coupé Touchtronic	Benzina	Automatico	2013	57 499.5	Euro 5	4	—	Milano (MI)
95 000	Cabrio	ALFA ROMEO	Spider	2.	Benzina	Manuale	1950	97 499.5	—	4	2/3	Milano (MI)
88 000	Coupé	PORSCHE	911 (997)	—	Benzina	—	1970	175 000.	—	—	2/3	Milano (MI)
88 000	Coupé	PORSCHE	901/911/912('63–88)	—	Benzina	Manuale	1970	32 499.5	—	—	—	Milano (MI)
87 993	Fuoristrada/SUV	ALFA ROMEO	—	—	Benzina	Automatico	2018	0.	—	5	—	Milano (MI)
86 900	Altro	MERCEDES	—	—	Benzina	Automatico	2017	0.	—	4	—	Milano (MI)
86 490	Fuoristrada/SUV	LAND ROVER	Range Rover 4ªserie	Range Rover 3.0 TDV6 Autobiography	Diesel	Automatico	2015	27 499.5	Euro 6	4	—	Milano (MI)
84 900	Fuoristrada/SUV	LAND ROVER	Range Rover 4ªserie	Range Rover 3.0 TDV6 Vogue	Diesel	Automatico	2016	22 499.5	Euro 6	5	—	Milano (MI)
80 800	Fuoristrada/SUV	ALFA ROMEO	—	—	Benzina	Automatico	2018	0.	—	5	—	Milano (MI)
80 000	Altro	PORSCHE	901/911/912('63–88)	911 Carrera 3.2 Targa	Benzina	Manuale	1970	2499.5	Pre–Euro	4	2/3	Milano (MI)
79 500	Coupé	PORSCHE	911 (991)	—	Altro	Automatico	2013	52 499.5	Euro 5	4	2/3	Milano (MI)
79 000	Berlina	BMW	—	—	Diesel	Automatico	2018	7499.5	Euro 6	5	4/5	Milano (MI)
75 800	Altro	BMW	—	—	Diesel	Automatico	2017	0.	—	5	—	Milano (MI)
75 700	Altro	BMW	—	—	Diesel	Automatico	2018	0.	—	5	—	Milano (MI)
75 000	—	OPEL	Meriva 2ª s.	Meriva 1.4 Turbo 120CV Cosmo	—	—	2011	62 499.5	—	—	—	Milano (MI)
74 900	Berlina	AUDI	A8 2ª serie	—	Diesel	Altro	2016	12 499.5	—	5	4/5	Milano (MI)
71 900	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
69 800	Altro	BMW	—	—	Diesel	Automatico	2017	0.	—	5	—	Milano (MI)
68 000	Fuoristrada/SUV	MASERATI	Levante	Levante V6 Diesel 275 CV AWD	Diesel	—	2016	27 499.5	—	—	—	Milano (MI)
68 000	Cabrio	BENTLEY	Continental	Continental GTC	—	Automatico	2007	114 999.	Euro 4	4	2/3	Milano (MI)
68 000	Altro	AUDI	A8 3ª serie	—	Benzina	Automatico	2015	22 499.5	—	5	—	Milano (MI)
67 900	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
67 500	—	MERCEDES	Classe GLE Coupé	GLE 350 d 4Matic Coupé Premium Plus	Diesel	—	2017	32 499.5	—	—	—	Milano (MI)
66 800	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
66 500	Berlina	BMW	—	—	Diesel	Automatico	2018	2499.5	—	5	4/5	Milano (MI)
66 000	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
65 000	—	PORSCHE	911 (997)	—	—	—	2009	12 499.5	—	—	—	Milano (MI)
64 600	Altro	BMW	—	—	Diesel	Automatico	2018	0.	—	5	—	Milano (MI)
63 400	Station Wagon	AUDI	—	—	Diesel	Automatico	2017	0.	—	5	—	Milano (MI)
62 900	Coupé	BMW	—	—	Altro	Automatico	2016	42 499.5	—	4	2/3	Milano (MI)
61 900	Cabrio	PORSCHE	901/911/912('63–88)	911 Carrera 3.2 Cabriolet	Benzina	Manuale	1988	135 000.	—	4	2/3	Milano (MI)
61 900	Berlina	BMW	—	—	Diesel	Automatico	2018	27 499.5	Euro 6	5	4/5	Milano (MI)
61 500	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
60 300	Altro	BMW	—	—	Benzina	Automatico	2017	0.	—	4	—	Milano (MI)
60 000	Coupé	PORSCHE	901/911/912('63–88)	911 Carrera 3.2 Coupé	Benzina	Manuale	1986	82 499.5	Pre–Euro	4	2/3	Milano (MI)
60 000	Coupé	ALFA ROMEO	—	—	Benzina	—	1960	52 499.5	Pre–Euro	5	2/3	Milano (MI)
60 000	Berlina	PORSCHE	Panamera	Panamera 3.0 Diesel Edition	Diesel	Automatico	2015	12 499.5	Euro 5	4	4/5	Milano (MI)
59 999	Coupé	NISSAN	GT–R	GT–R 3.8 V6 Premium Edition	Benzina	Automatico	2009	17 499.5	Euro 5	4	2/3	Milano (MI)
59 500	Fuoristrada/SUV	BMW	—	—	Diesel	Automatico	2018	0.	Euro 6	5	—	Milano (MI)
59 000	Coupé	PORSCHE	901/911/912('63–88)	911 SC 3.0 Coupé	Benzina	Manuale	1980	2499.5	—	2	—	Milano (MI)
58 960	Berlina	BMW	Serie 7 (G11/G12)	730d xDrive Luxury	Diesel	Automatico	2016	7499.5	Euro 6	—	4/5	Milano (MI)
58 900	Station Wagon	BMW	—	—	Diesel	Automatico	2018	2499.5	Euro 6	5	—	Milano (MI)
58 900	Fuoristrada/SUV	MERCEDES	Classe G(G461/G463)	G 350 BlueTEC S.W.	Diesel	Automatico	2013	92 499.5	Euro 5	5	4/5	Milano (MI)

7481 total ›

FIGURE 2: 70 EXPENSIVE CARS

2 DESCRIPTIVE STATISTICS

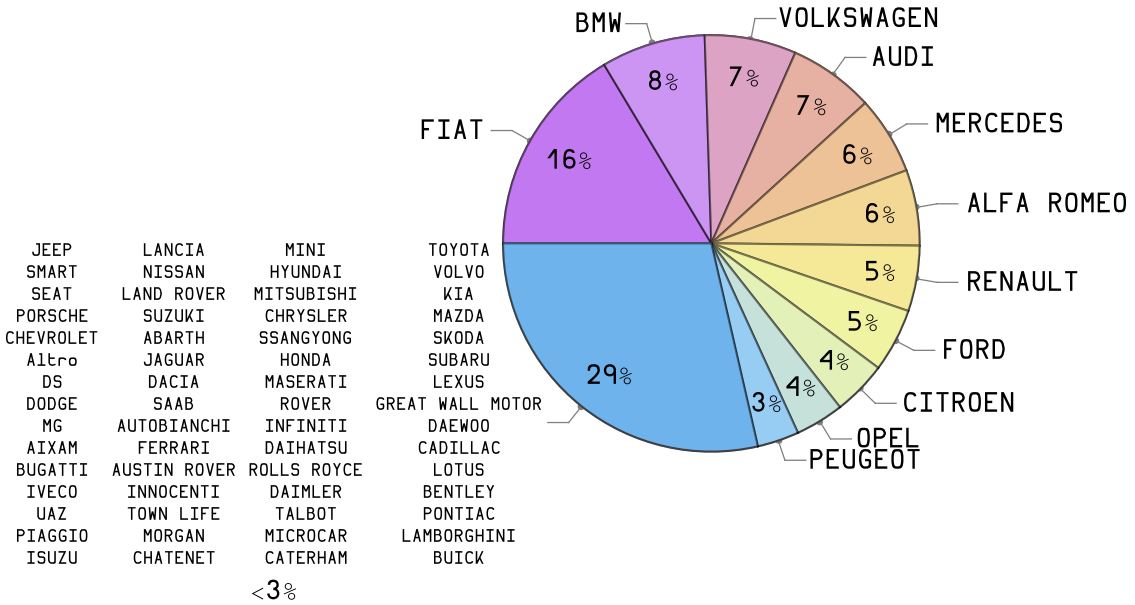
The goal of this section is to create some descriptive representations of the data. The codes in this section automatically generate vector graphics from one or more DB columns. All these graphics are suitable to represent dynamical data insofar the only input is the DB and no post-editing is needed.

2.1 CAR MODELS AND PRICE DISTRIBUTION

In the first two examples we consider the "Marca" (manufacturer) column.

SNIPPET 5: Make a piechart displaying the distribution of manufacturers. Group the manufacturers below 3% into a single slice.

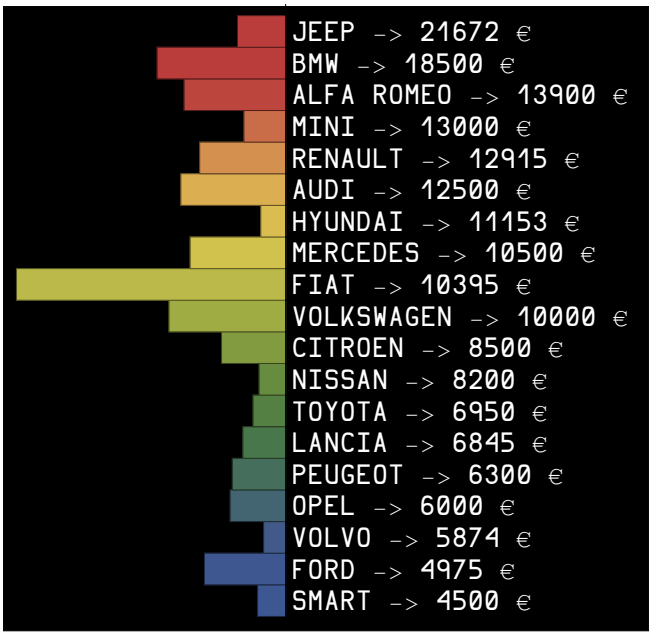
```
milanDB[All, "Marca"] //
          Tally@#~SortBy~Last& //
          Normal // Reverse // Transpose //
          {#1, 100.0 #2 / Total@#2}&@@#& // Transpose //
          GatherBy[#, Last@# < 3&]& //
          Append @@ {#1, {#1~Partition~4 // Grid@#~Style~8& // Labeled[#, "<3%"]&,
                    Total@#2}&@@Transpose@#2}&@@#& //
 PieChart[Last /@ #
    , ChartLabels →
          Placed[Style[#, 12.4, FontFamily → "Hermit"]&@*First /@ #,
                "RadialCallout"]
    , ImagePadding → {{250, 50}, {50, 0}}
    , ImageSize → 600
   ]~Show~
   PieChart[Last /@ #
    , ChartLabels →
       Placed[Style[ToString@# <> "%", 12.4, FontFamily → "Hermit"]&@*Round@*Last /@ #,
              "RadialOuter"]
    , ChartStyle → "Pastel"
    , PlotRangePadding → None]&
```



Here's a different layout that include the median price.

SNIPPET 6: Take the manufacturers that has more than 100 ads, show the median price in descending order next to a bar the length of which represents the relative number of ads of the manufacturer, while the color temperature represents the median price

```
milanDB[GroupBy@"Marca"][Length@# > 100 & // Select] //
    {Median /@ DeleteMissing /@ #[All, All, "Prezzo"] // Sort, Length /@ #} & //
    Round /@ Normal /@ # & //
    BarChart[Last@# /@ Keys@First@#
      , ChartLabels → (# ~ Rotate ~ (3 / 2 Pi) ~ Style ~ {FontFamily → "Hermit", White, 15.1} & /@
                       (ToString@# <> " € " & /@ Normal@First@#))
      , ChartStyle → "DarkRainbow"
      , AspectRatio → .5
      , Background → Black
      , PlotRangePadding → {{-1, -.1}, 60}
      , LabelStyle → "White"
      , PlotTheme → "Minimal"
      ] ~ Rotate ~ (Pi / 2) &
```



The median price per manufacturers may not be very significant though. In the next example we perform a deeper query to display the median price per model.

SNIPPET 7: Show the median price in descending order of the models that have at least 10 ads, but only of those manufacturer that has at least 10 models satisfying the 10 ads per model criterion.

```
milanDB[Not@MissingQ@#Modello && Not@MissingQ@#Prezzo & // Select][
                GroupBy@"Marca"][
                 All, GroupBy@"Modello"][
                All, Length@# > 10 & // Select][
                 All, SortBy@Length][
                 All, Reverse][
                SortBy@Length][
                 Length@# > 10 & // Select][
                Reverse][
               All, All, All, "Prezzo"][
                All, All, Median][
                All,  Reverse@*Sort] //
    {{#[[{1}]], {#[[{2}]], #[[{5}]]}, {#[[{3}]], #[[{4}]]}} & //
    Column[#, Alignment → {Top, Right}] & /@ # & // Row[#, "   "] &
```

FIAT	500X	16 500.
	Tipo	15 010.
	Tipo (2015–––>)	14 630.
	Freemont	13 900.
	QUBO	12 750.
	500L	11 450.
	500 (2015–––>)	10 590.
	Panda	10 340.
	Panda 3ª serie	8895.
	Punto 4ª serie	7250.
	500 (2007–2016)	6900.
	Punto Evo	5220.
	Bravo 2ª serie	4325.
	Panda 2ª serie	3800.
	Punto	3700.
	Doblò	3200.
	Grande Punto	3025.
	Punto 3ª serie	2200.
	600	1850.
	Stilo	1300.
	Punto 2ª serie	1150.

BMW	Serie 1 (F20)	23 490.
	Serie 2 A.T. (F45)	23 000.
	X3 (F25)	19 900.
	Serie 3 (F30/F31)	18 600.
	Serie 5 (F10/F11)	17 400.
	X1 (E84)	15 050.
	X5 (E70)	15 000.
	Serie 3 (E93)	12 900.
	Serie 5 (E60/E61)	8500.
	Serie 3 (E92)	8500.
	Serie 1 (E87)	7300.
	X3 (E83)	6950.
	Serie 3 (E90/E91)	6500.
	Serie 3 (E46)	2750.

AUDI	Q5 1ª serie	20 000.
	A6 4ª serie	18 900.
	A5 1ª serie	16 000.
	A3 3ª serie	15 999.
	A1/S1	13 500.
	Q7 1ª serie	13 250.
	TT 2ª serie	12 950.
	A4 4ª serie	12 700.
	TT 1ª serie	7800.
	A3 2ª serie	7500.
	A6 3ª serie	7245.
	A4 3ª serie	5500.
	A6 2ª serie	3000.

MERCEDES	Classe CLA (C/X117)	25 000.
	Classe A (W176)	18 000.
	Classe E (W/S212)	13 850.
	Classe B(T246/T242)	13 500.
	Classe M (W164)	11 900.
	Classe C (W/S204)	8999.5
	Classe B (T245)	6900.
	Classe A (W/C169)	5650.
	Classe C (W/S203)	3000.
	Classe A (W/V168)	2425.
	Classe C (W/S202)	2000.

VOLKSWAGEN	Golf 7ª serie	15 450.
	Tiguan 1ª serie	15 200.
	Passat 7ª serie	11 100.
	Golf 6ª serie	9450.
	Touareg	9350.
	Maggiolino	9000.
	Polo 5ª serie	8300.
	Touran	6200.
	New Beetle	5950.
	Passat 6ª serie	5600.
	Golf 5ª serie	4200.
	Polo 4ª serie	3500.
	Golf 4ª serie	1500.

These look-up tables can be considered rudimentary price predictors. The problem with predicting a quantity by querying finite data is the absence of a continuous model representing

the non-categorical quantities. We'll tackle this issue in §3.

2.2 GEOGRAPHICAL DISTRIBUTION

In the concluding figure of this section we show something about the sellers instead of the cars on sale. We extract the sellers telephone numbers from the full Italian dataset (see §3.1) and count how many ads these sellers have and where they sell their cars. We need the extracted list of $\{City, Number\}$ pairs we have previously stored in the compressed locNum.mx file

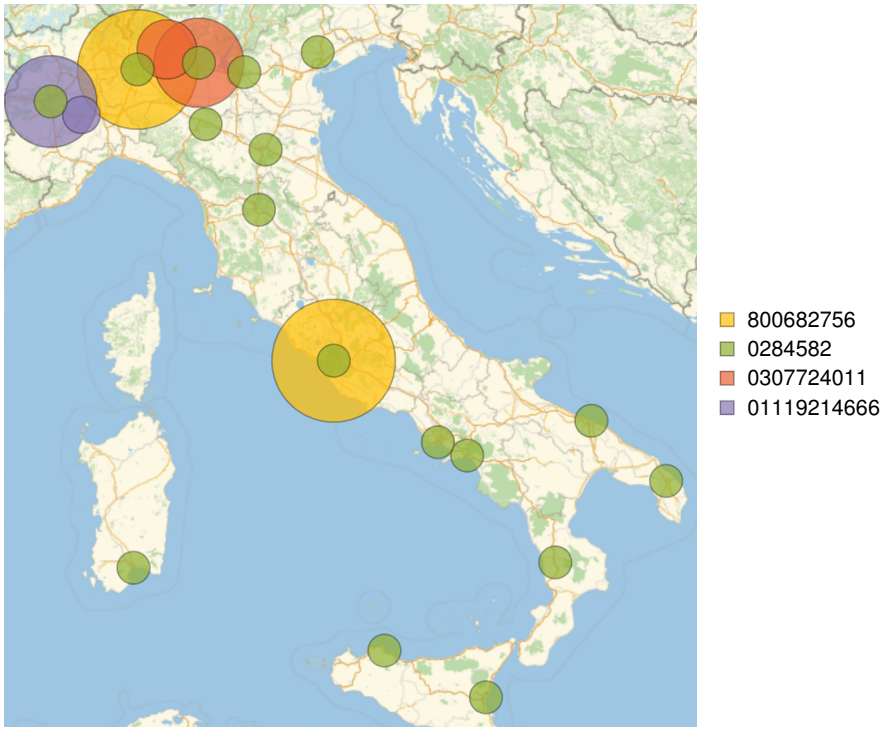
```
<< "locNum.mx"
locNum // Short
```

```
{{"Avellino", "3283086882"}, {"Taranto", "3207464752"}, {"Torino", "999"}, <<388 965>>, {"Alessandria", "999"}, {"Taranto", "350"}}
```

SNIPPET 8: The telephone number is not present in every ad; numbers that are less than 5 digits long are discarded. Then we gather the pairs with the same telephone number, count each group length, sort the list of groups by length and finally take only the biggest 10. The cities geographical coordinates are fetched from the Wolfram server. The map shows only sellers that are present in more than one city; the area of the disks around the cities is proportional to the amount of ads.

```
tallyNum = locNum ~ Select ~ (StringLength@Last@# > 5 &) ~ GatherBy ~ Last ~ SortBy ~ Length ~ Take ~ (-10) //
           Last@Last@# -> Tally[First /@ #] & /@ # & // Reverse;

geodata = (CityData@{First@#, "Italy"} → Last@# & /@ Values@#) & /@ tallyNum //
          Select[#, Length@# > 1 &] &;

bubbles = {MapAt[First, #, {All, 1}] & /@ geodata, tallyNum ~ Select ~ (Length@Values@# > 1 &) // Keys} //
          Transpose // Legended @@@ # & // GeoBubbleChart;

{tallyNum // Association /@ # & // Dataset, bubbles} // Row[#, "      "] &
```

0516861760	Bologna	2965
800682756	Milano	1377
	Roma	1455
0284582	Bari	102
	Bologna	102
	17 total ›	
0307724011	Bergamo	338
	Brescia	770
01119214666	Torino	813
	Asti	133
0432576511	Udine	925
040827032	Trieste	898
04381847111	Treviso	878
0421392211	Venezia	873
0434511211	Pordenone	854



3 INFERENCE STATISTICS

In this section we build continuous statistical models trained on the whole *Subito.it* car market. The data collection job on the whole Italian region has a slightly different implementation; we give some details about this scraping process first.

3.1 THE FULL ITALIAN DB

Due to the limited size of the data collected in the Milan area, we were able to download and parse the HTML tables in one shot (SNIPPET 3). When it comes to scrape hundreds of thousands ads for several hours, it is safer to store the htm pages first. So, instead of SNIPPET 3 we have:

```
URLDownload~ParallelMap~adLinks
```

where adLinks is defined as in SNIPPET 2 and resultPages as in SNIPPET 1 but starting from the "global" search page. Then we translate the htm into plain text with a handy command line utility like w3m

```
Run["w3m -dump " <> # <> " > ~/GoodBuyAutoReport/allTXT/" <> StringDrop[#, -3] <> "txt"] &~ParallelMap~FileNames[]
```

Through wildcard matching we extract the tables from the txt files and create a premilinary DB

```
(StringTrim@StringTake[#, 21] → StringTrim@StringDrop[#, 21])~Check~Nothing[] &/@
 StringSplit[#, "\n"] &@First@
 StringCases[Import[#, "Text"], "\nPrezzo" ~~ ___ ~~ "Comune" ~~ __ ~~ "\n" // Shortest]~Check~Nothing[] &~
 ParallelMap~FileNames[] // KeyUnion // Dataset;
```

which must be curated as described in §1.2 to obtained a complete and cleaned up italDB.

Our statistical models aim to fit a modelDB obtained from the complete database italDB by selecting second hand (Km>0) cars model that has at least 2000 ads without missing information along the Marca, Modello, Km, Carburante, Anno immatricolazione, Km and Prezzo columns.

```
<< "/home/nicola/GoodBuyAutoReport/italDB.mx"

modelDB = italDB[Not[MissingQ@#Modello || MissingQ@#Km || MissingQ@#"Anno immatricolazione" ||
                     MissingQ@#Marca || MissingQ@#Prezzo] & // Select][
                  #Km > 0 & // Select][
                  DeleteDuplicates][
                  GroupBy@"Modello"][
                  Length@# > 2000 & // Select][
                  All, All, {"Marca", "Modello", "Cambio", "Carburante", "Anno immatricolazione", "Km", "Prezzo"}][
                  SortBy@Length];
```

In the next section we extract the design matrix from this modelDB and we will split it into a training set and a test set. Note that our design matrix is *unbalanced* or *nearly-balanced* at best, insofar it has the same amount of objects per Modello, but in each Modello group there might be all sorts of objects. For instance, older car models do not have any representatives with hybrid or electric motors etc.

3.2 DATA ENCODING

We get rid of the relational DB structure and partition the data into two rectangular matrices. The trainingMatrix has exactly 1500 ads per car model, while the rest is left for the testMatrix.

```
{trainingMatrix, testMatrix} = Flatten@MapAt[StringRiffle, #~TakeDrop~2, 1] &/@
                               Values[Join @@ Values@Normal@#] &/@
                               {modelDB, modelDB[All, #~RandomSample~1500 &]} //
                               {Last@#, Complement @@ #} &;
```

Note that the first line merges the Marca and Modello columns.

We chose *one-hot encoding* for the categorical variables; our encoder consists in the following list of substitution rules

```
enc = Function[col, #[[col]] &/@ trainingMatrix // DeleteDuplicates //
                    NetEncoder[{"Class", #, "UnitVector"}]~List~# & //
                    Thread@{#2, #1 /@ #2} &@@ # & // Rule @@@ # &] /@
      {1(*Marca+Modello*), 2(*Cambio*), 3(*Carb*)} // Flatten
```

```
      BMW Serie 3 (F30/F31)      ->  {1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
  ALFA ROMEO Giulietta (2010)    ->  {0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
           FIAT Panda            ->  {0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
        AUDI A3 3ª serie         ->  {0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
   VOLKSWAGEN Golf 7ª serie      ->  {0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
      FIAT Panda 3ª serie        ->  {0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
      FIAT Panda 2ª serie        ->  {0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
        AUDI A4 4ª serie         ->  {0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
     SMART fortwo 2ª serie       ->  {0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
      BMW Serie 3 (E90/E91)      ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
      FIAT 500 (2007-2016)       ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
    LANCIA Ypsilon 2ª serie      ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
       FIAT Grande Punto         ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
      FIAT Punto 3ª serie        ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
        FIAT Punto Evo           ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
          FIAT Punto             ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.}
   MERCEDES Classe A (W/C169)    ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.}
    VOLKSWAGEN Golf 6ª serie     ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.}
        AUDI A3 2ª serie         ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.}
           FIAT 500X             ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.}
           FIAT 500L             ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.}
      ABARTH Grande Punto        ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.}
       ABARTH Punto Evo          ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.}
         ABARTH Punto            ->  {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.}
          Automatico             ->                                    {1., 0., 0., 0.}
            Manuale              ->                                    {0., 1., 0., 0.}
          Sequenziale            ->                                    {0., 0., 1., 0.}
          AltroCambio            ->                                    {0., 0., 0., 1.}
             Diesel              ->                          {1., 0., 0., 0., 0., 0., 0.}
             Benzina             ->                          {0., 1., 0., 0., 0., 0., 0.}
             Metano              ->                          {0., 0., 1., 0., 0., 0., 0.}
               Gpl               ->                          {0., 0., 0., 1., 0., 0., 0.}
             Ibrida              ->                          {0., 0., 0., 0., 1., 0., 0.}
           Elettrica             ->                          {0., 0., 0., 0., 0., 1., 0.}
           AltroCarb             ->                          {0., 0., 0., 0., 0., 0., 1.}
```

These substitution rules are applied to the training and test matrices to create learnable training+test sets:

```
{trainingSet, testSet} = Function[ m, Thread@MapAt[Rescale, Thread@m, {{4}, {5}}] /. enc //
                                      Flatten /@ # & //
                                      Most@# → List@Last@# & /@ # &] /@
                         {trainingMatrix, testMatrix}
```

note that the non-categorical variables (Anno immatricolazione and Km) have been rescaled from 0 to 1. Here's a random element from the training set

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.943182, 0.209999} → {20500.}
```

In summary, the encoding resulted in 37 independent variables: the first 24 represent the car models, then we have 4 variables relative to the gear type, 7 to the fuel type, 1 for the registration date and 1 for the mileage.

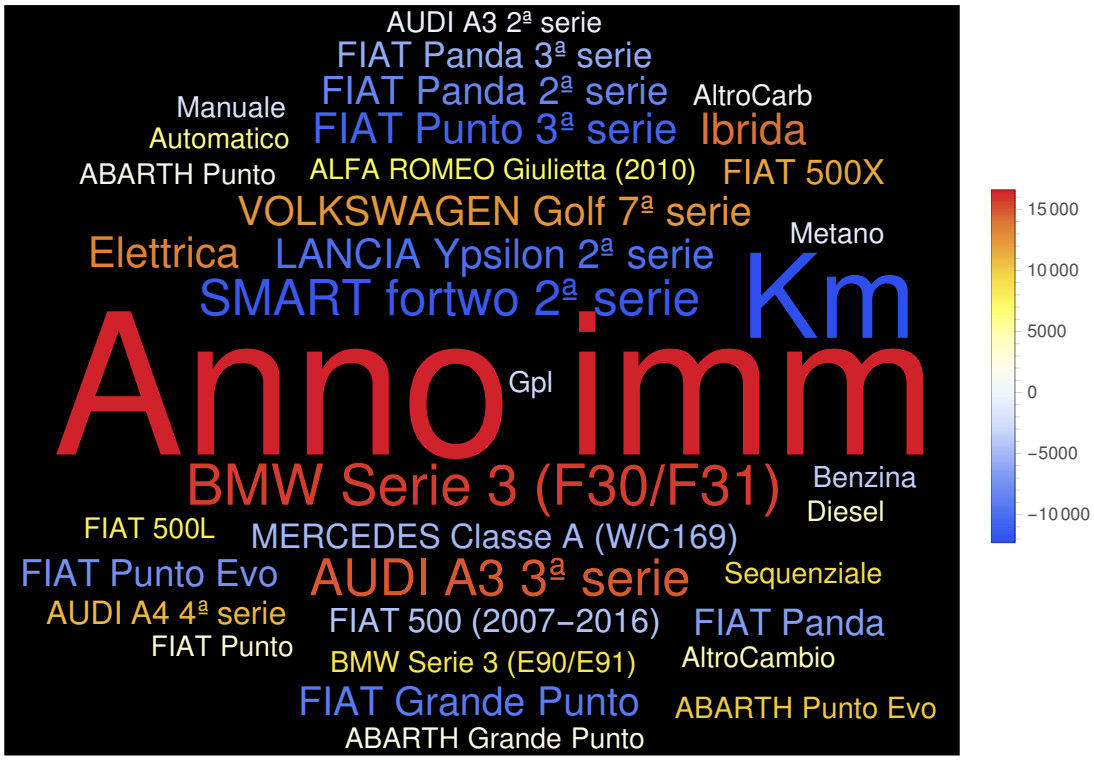
3.3 LINEAR REGRESSION

Here's a linear model that fits the training set:

```
X = Subscript[#, All] & /@ Range[Length[Keys[First[trainingSet]]]];
LinearModelFit[Join @@@ List @@@ trainingSet, #, #] &
```

$$-852.819 - 3194.2\ x_1 + 3524.21\ x_2 - 4008.95\ x_3 + 704.269\ x_4 - 135.136\ x_5 - 5171.91\ x_6 - 6303.97\ x_7 + 844.138\ x_8 - 3369.24\ x_9 - 531.472\ x_{10} - 4681.52\ x_{11} - 1071.77\ x_{12} - 4520.1\ x_{13} - 4182.54\ x_{14} + 7208.45\ x_{15} + 2919.12\ x_{16} - 3753.\ x_{17} + 6466.34\ x_{18} - 4925.66\ x_{19} + 1250.32\ x_{20} + 4797.94\ x_{21} - 827.167\ x_{22} + 1688.77\ x_{23} - 939.739\ x_{24} - 1206.68\ x_{25} + 690.367\ x_{26} + 1401.85\ x_{27} - 471.269\ x_{28} - 474.254\ x_{29} + 5182.19\ x_{30} - 1387.97\ x_{31} - 1326.71\ x_{32} - 1088.56\ x_{33} - 1006.77\ x_{34} + 5364.19\ x_{35} + 16628.2\ x_{36} - 12283.4\ x_{37}$$

Let's make a word cloud where every variable is sized according to coefficient modulus, while the color temperature expresses the actual value of the coefficient.



The word cloud shows that the registration date and mileage are the driving variables.

Let's ignore the variables relative to the car models (x_1 .. x_{28}) , sort the remaining coefficients, group them according to their sign

Km	−12 283.4
Benzina	−1387.97
Gpl	−1326.71
Manuale	−1206.68
Metano	−1088.56
AltroCarb	−1006.77
Diesel	−474.254
AltroCambio	−471.269

Automatico	690.367
Sequenziale	1401.85
Elettrica	5182.19
Ibrida	5364.19
Anno imm	16 628.2

and observe that:

- Electric or hybrid motors add significant value compare to fuels.
- The manual gear is the only gear with a negative sign; also, the gap between manual and automatic/sequential gear is fairly large.

Even though these coefficient make very much sense, we ought to keep in mind that there is a certain correlation between our variables. For instance, the coefficient relative to hybrid motors could be so high just because it is found on newer cars.

In the next section we build a non-linear model and given a minimal adjustment to our dataset.

3.4 NON-LINEAR MODELING

The simplest non-linear model to work with is the Multilayer Perceptron (MLP). Given a D-dimensional input vector **x** and a scalar output y , we can think of a MLP as a linear regression of N *features* $\mathbf{f} = \{f_1(\mathbf{x}), ..f_N(\mathbf{x})\}$:

$$y = \mathbf{w} \cdot \mathbf{f} + \beta$$

where the weights $\mathbf{w} = \{w_1, ..w_N\}$ can either be the least square solution calculated by pseudoinversion or a loss minima found by gradient descend. In both cases, these features do not need to be engineered *a priori*.

Let's consider the pseudoinverse method; as a matter of fact, we define a feature f_i as a random linear combination of the input variables, squished with a sigmoid function σ

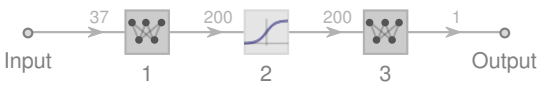
$$f_i = \sigma\left(\mathbf{x} \cdot \mathbf{r}_i + b_i\right)$$

where $\mathbf{r}_i = \{r_{i,1}, ..r_{i,D}\}$ is a set of random weights, and b_i a random bias.

The following function creates a MLP with 1 hidden layer composed of *hN* random neurons (features) that fits a dataset *d* by pseudoinversion of the last layer

```
xNetF[hN_, d_] := NetInitialize@NetChain[{
    hN,
    LogisticSigmoid
    }, "Input" → Length@First@Keys@d] // #~NetAppend~LinearLayer[
    1
    , "Weights" → Table[PseudoInverse[#@Keys@d].i, {i, Transpose@Values@d}]
    , "Biases" → 0] &
```

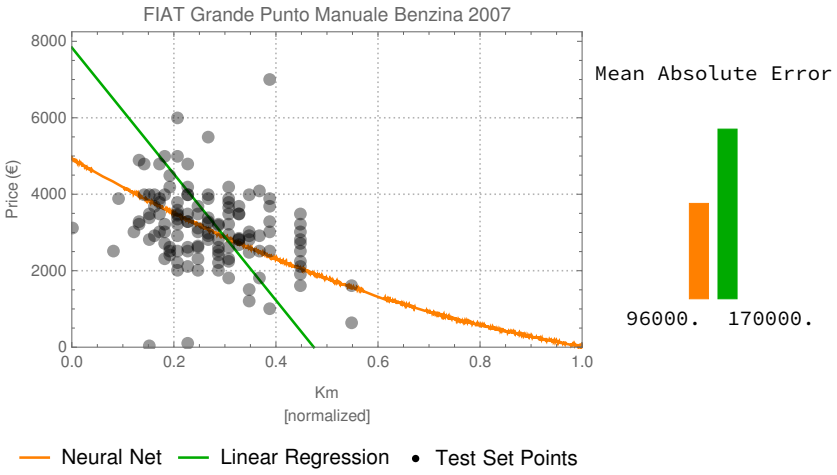
Here's an example of the computational graph for 200 hidden features



Now, the problem with non-linear models is that you never know how they extrapolate. We have very scarce data at high mileage, so anything can happen close to $max(Km)$, skyrocketing prices for instance. To prevent unrealistic predictions, we fabricate fake points that has zero price at 1 million km, that corresponds to a value of 2 in the rescaled mileage variable. In particular, we insert rows like $\{..\ ,\ 2\} \rightarrow \{0\}$ in every class of equal $\{Marca + Modello, Cambio, Carburante, Annoimm\}$. Finally, our non-linear model is constructed with 400 random features that fit adjusted dataset by pseudoinversion:

```
xNetF[400, Join[#~Append~2. → {0.} & /@ Most /@ Keys@#, #] &@trainingSet]
```

Here's typical example of how the linear regression and MLP predict the testSet prices as a function of the mileage for a $\{Marca + Modello, Cambio, Carburante, Annoimm\}$ class:

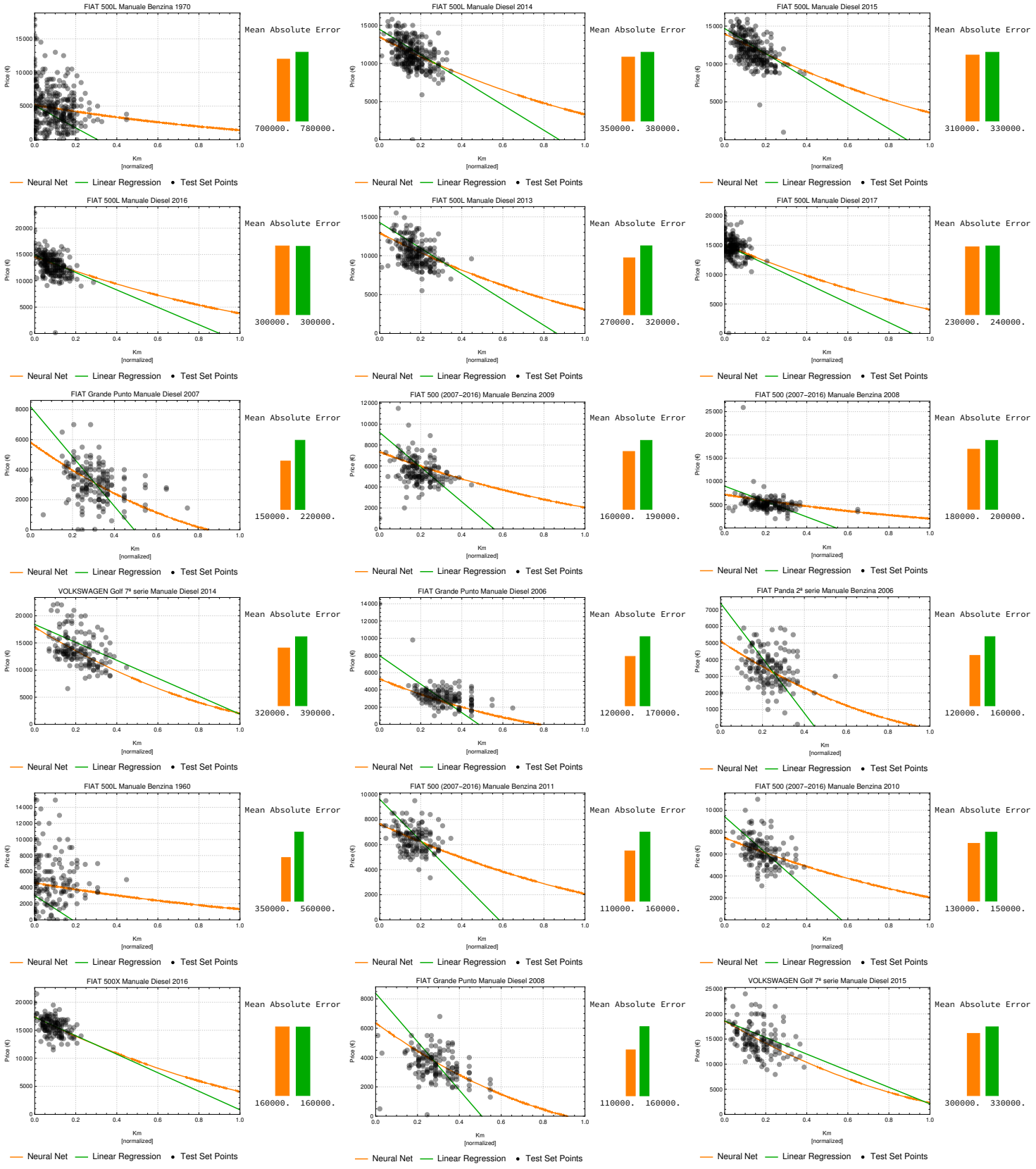


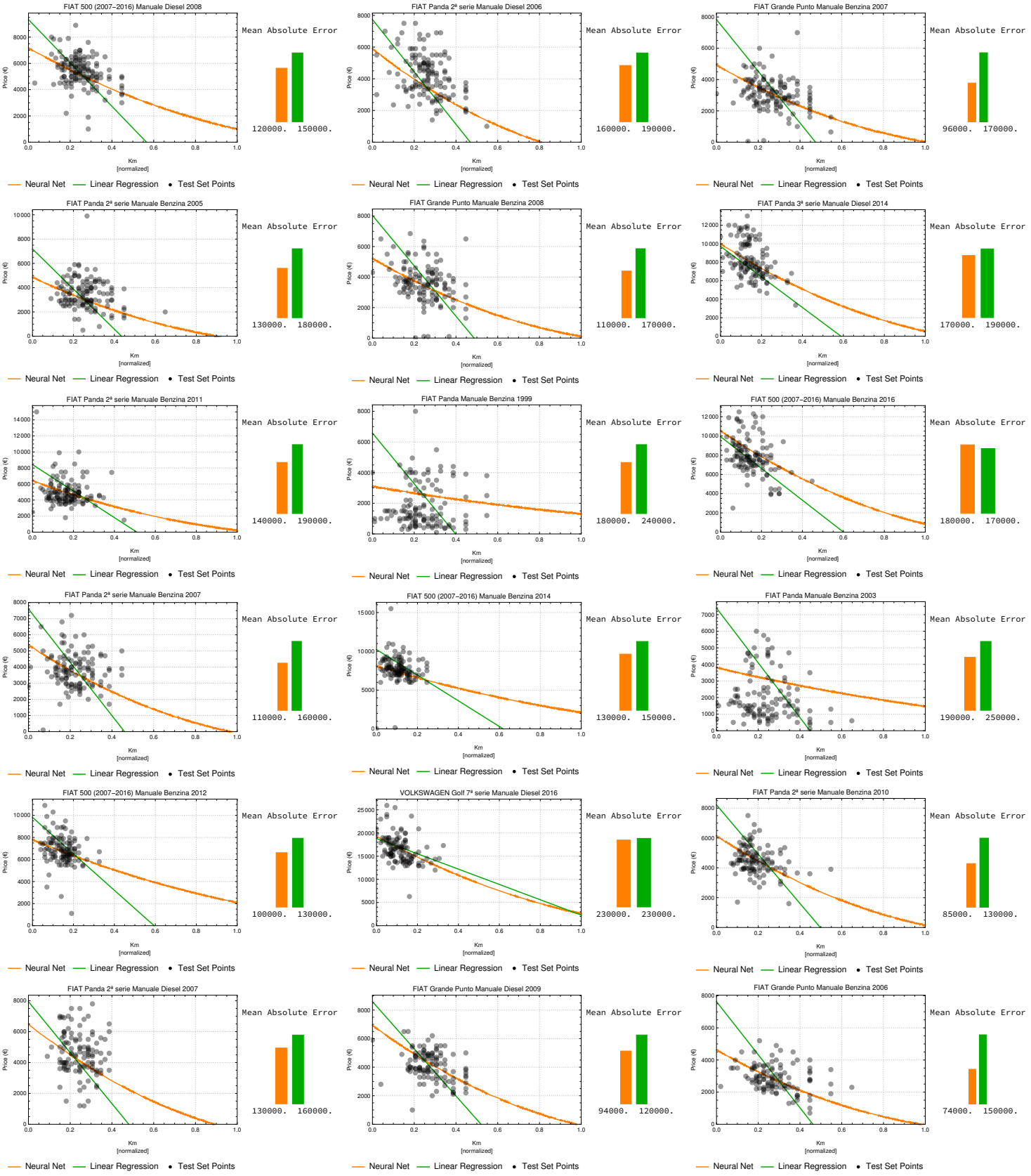
In general, the MLP trained on the adjusted training set finds a better numerical fit and also a more realistic long term prediction compared to the linear regression; as shown in the Appendix, the same holds true for the large majority of classes. When the linear model is trained on the adjusted dataset, its long term prediction are similar to the MLP but it fails to fit the real data-points pretty badly. Only the non-linear model can properly fit the real data while approaching zero price at high mileage.

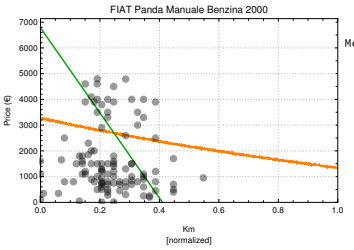
4 APPENDIX

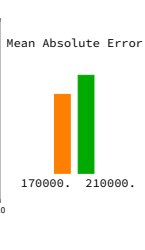
SNIPPET 9: This is the code that produce all the plots presented in the Appendix. The graphics specifications and the mean absolute errors of the models are computed in a function of the class (here called group). This function is mapped on the list of classes, which is created from the test set by gathering all examples that share the class keys. The list of classes is sorted by length and it contains ≈ 1000 classes, many of which contains just few objects. Therefore, only the biggest 72 are shown hereafter.

```
Function[group
  , Module[{ex, exG, err1, err2},
           ex = Most@Keys@First@group~Append~x;
           exG = group;

           err1 = Abs /@ Differences /@ Transpose@{trainedNet@Keys@group, Values@group} // Total // Flatten // First;
           err2 = Abs /@ Differences /@ Transpose@{model @@@ (Keys@group), Values@group} // Total // Flatten // First;

           Show[
                Plot[ {trainedNet@ex, model @@ ex}, {x, 0, 1}
                     , PlotStyle → {Orange, Darker@Green}
                     , PlotLegends → Placed[LineLegend[{Orange, Darker@Green}, {"Neural Net", "Linear Regression"}]
                                            , Bottom]
                     , PlotTheme → "Detailed"
                     , FrameLabel → {"Km"~Labeled~"[normalized]", "Price (€)"}
                     , PlotRange → {{0, 1}, {0, All}}
                     , PlotLabel → StringRiffle[testMatrix~Extract~Position[Keys[testSet]⟦All, ;; -2⟧, Most@ex] //
                                                First[#]⟦;; -3⟧ &]
                    , ImageSize → 370 ] ,
                ListPlot[ Flatten@{Last@Keys@#, Values@#} & /@ exG
                         , PlotStyle → {Opacity@.4, Black, PointSize@0.025}
                         , PlotLegends → Placed[PointLegend[{Black}, {"Test Set Points"}], Bottom]]]~

    List~Labeled[Labeled[Graphics@{Orange, Rectangle[{0, 0}, {2, 10}],
                                   Darker@Green, Rectangle[{3, 0}, {5, 10 err2 / err1}]},
                         Row@{err1~PaddedForm~2, " ", err2~PaddedForm~2},
                         Bottom],
                 "Mean Absolute Error\n",
                 Top] // Row]

] /@ (GatherBy[testSet, Keys@#~Take~36 &]~SortBy~Length // Reverse)
```

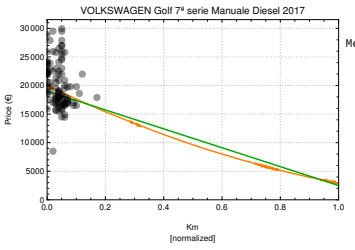

FIAT 500L Manuale Benzina 1970
Mean Absolute Error
700000. 780000.
FIAT 500L Manuale Diesel 2014
Mean Absolute Error
350000. 380000.
FIAT 500L Manuale Diesel 2015
Mean Absolute Error
310000. 330000.
FIAT 500L Manuale Diesel 2016
Mean Absolute Error
300000. 300000.
FIAT 500L Manuale Diesel 2013
Mean Absolute Error
270000. 320000.
FIAT 500L Manuale Diesel 2017
Mean Absolute Error
230000. 240000.
FIAT Grande Punto Manuale Diesel 2007
Mean Absolute Error
150000. 220000.
FIAT 500 (2007–2016) Manuale Benzina 2009
Mean Absolute Error
160000. 190000.
FIAT 500 (2007–2016) Manuale Benzina 2008
Mean Absolute Error
180000. 200000.
VOLKSWAGEN Golf 7ª serie Manuale Diesel 2014
Mean Absolute Error
320000. 390000.
FIAT Grande Punto Manuale Diesel 2006
Mean Absolute Error
120000. 170000.
FIAT Panda 2ª serie Manuale Benzina 2006
Mean Absolute Error
120000. 160000.
FIAT 500L Manuale Benzina 1960
Mean Absolute Error
350000. 560000.
FIAT 500 (2007–2016) Manuale Benzina 2011
Mean Absolute Error
110000. 160000.
FIAT 500 (2007–2016) Manuale Benzina 2010
Mean Absolute Error
130000. 150000.
FIAT 500X Manuale Diesel 2016
Mean Absolute Error
160000. 160000.
FIAT Grande Punto Manuale Diesel 2008
Mean Absolute Error
110000. 160000.
VOLKSWAGEN Golf 7ª serie Manuale Diesel 2015
Mean Absolute Error
300000. 330000.
Price (€)
Km
[normalized]
Neural Net
Linear Regression
Test Set Points

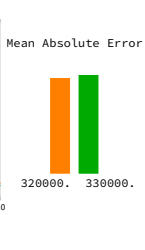

FIAT 500 (2007–2016) Manuale Diesel 2008
Mean Absolute Error
120000. 150000.
FIAT Panda 2ª serie Manuale Diesel 2006
Mean Absolute Error
160000. 190000.
FIAT Grande Punto Manuale Benzina 2007
Mean Absolute Error
96000. 170000.
FIAT Panda 2ª serie Manuale Benzina 2005
Mean Absolute Error
130000. 180000.
FIAT Grande Punto Manuale Benzina 2008
Mean Absolute Error
110000. 170000.
FIAT Panda 3ª serie Manuale Diesel 2014
Mean Absolute Error
170000. 190000.
FIAT Panda 2ª serie Manuale Benzina 2011
Mean Absolute Error
140000. 190000.
FIAT Panda Manuale Benzina 1999
Mean Absolute Error
180000. 240000.
FIAT 500 (2007–2016) Manuale Benzina 2016
Mean Absolute Error
180000. 170000.
FIAT Panda 2ª serie Manuale Benzina 2007
Mean Absolute Error
110000. 160000.
FIAT 500 (2007–2016) Manuale Benzina 2014
Mean Absolute Error
130000. 150000.
FIAT Panda Manuale Benzina 2003
Mean Absolute Error
190000. 250000.
FIAT 500 (2007–2016) Manuale Benzina 2012
Mean Absolute Error
100000. 130000.
VOLKSWAGEN Golf 7ª serie Manuale Diesel 2016
Mean Absolute Error
230000. 230000.
FIAT Panda 2ª serie Manuale Benzina 2010
Mean Absolute Error
85000. 130000.
FIAT Panda 2ª serie Manuale Diesel 2007
Mean Absolute Error
130000. 160000.
FIAT Grande Punto Manuale Diesel 2009
Mean Absolute Error
94000. 120000.
FIAT Grande Punto Manuale Benzina 2006
Mean Absolute Error
74000. 150000.
Price (€)
Km
[normalized]
Neural Net
Linear Regression
Test Set Points



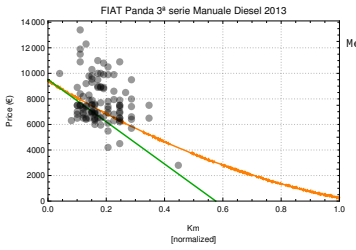


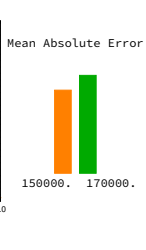
Neural Net — Linear Regression • Test Set Points



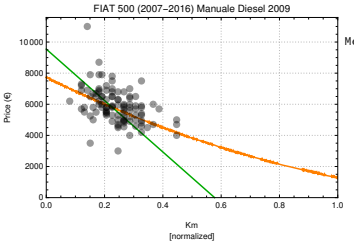


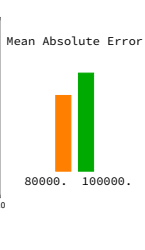
Neural Net — Linear Regression • Test Set Points



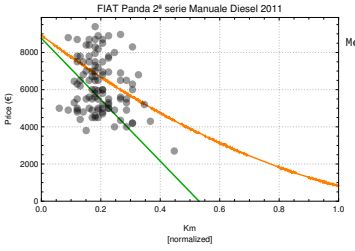


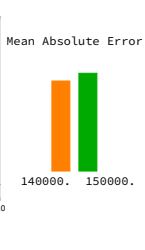
Neural Net — Linear Regression • Test Set Points



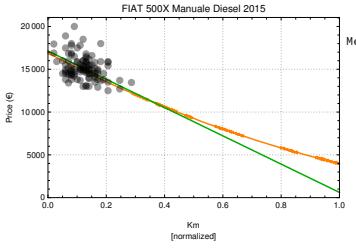


Neural Net — Linear Regression • Test Set Points

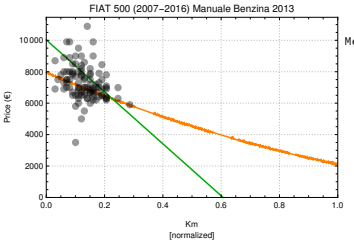


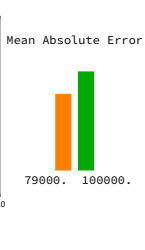


Neural Net — Linear Regression • Test Set Points

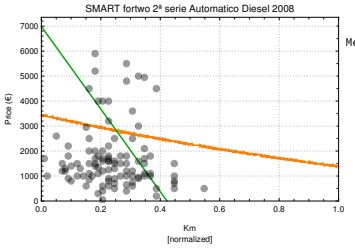


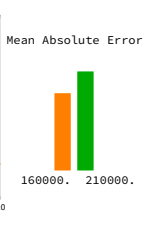
Neural Net — Linear Regression • Test Set Points



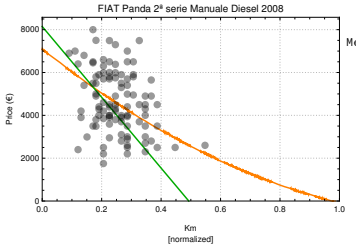


Neural Net — Linear Regression • Test Set Points

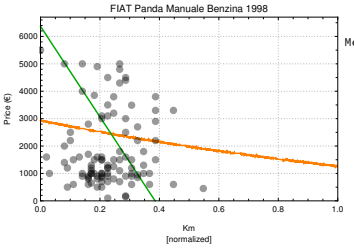


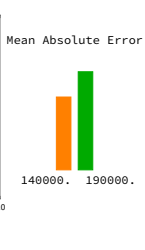


Neural Net — Linear Regression • Test Set Points

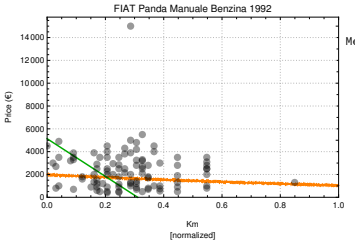


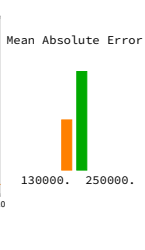
Neural Net — Linear Regression • Test Set Points



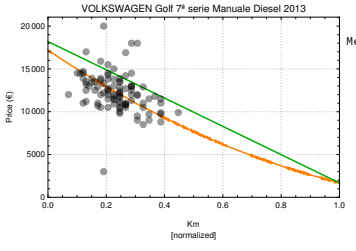


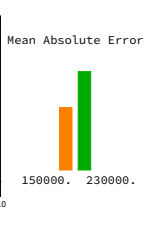
Neural Net — Linear Regression • Test Set Points



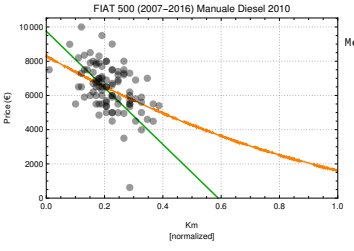


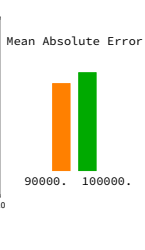
Neural Net — Linear Regression • Test Set Points



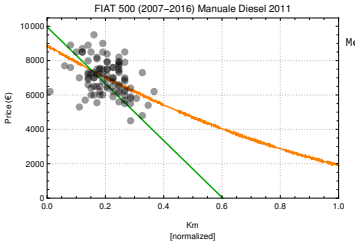


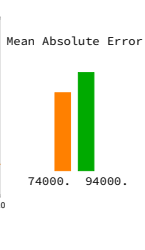
Neural Net — Linear Regression • Test Set Points



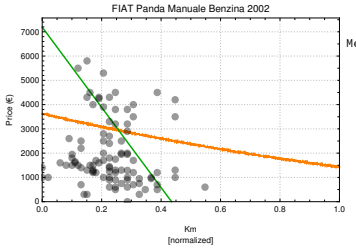


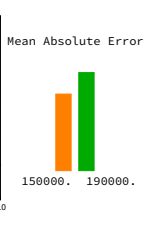
Neural Net — Linear Regression • Test Set Points



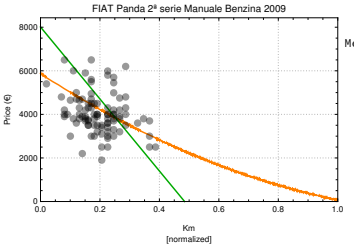


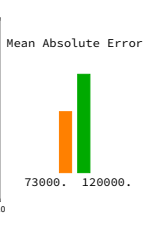
Neural Net — Linear Regression • Test Set Points



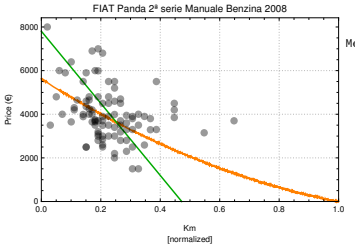


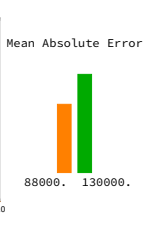
Neural Net — Linear Regression • Test Set Points



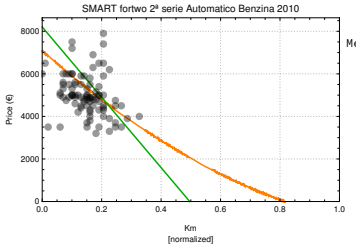


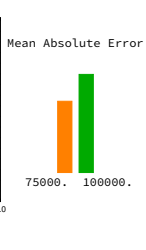
Neural Net — Linear Regression • Test Set Points



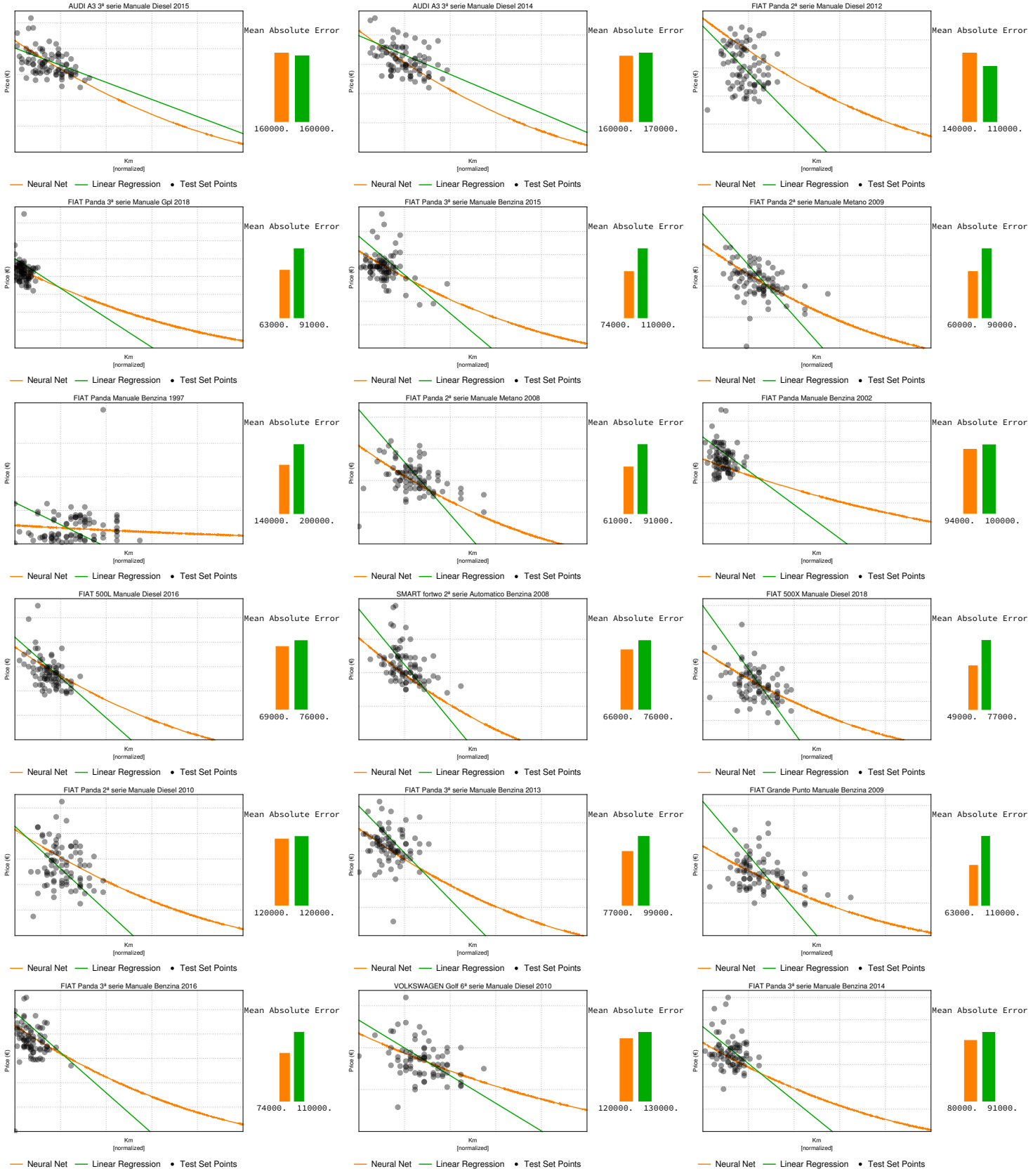


Neural Net — Linear Regression • Test Set Points





Neural Net — Linear Regression • Test Set Points


AUDI A3 3ª serie Manuale Diesel 2015
Mean Absolute Error
160000. 160000.
AUDI A3 3ª serie Manuale Diesel 2014
Mean Absolute Error
160000. 170000.
FIAT Panda 2ª serie Manuale Diesel 2012
Mean Absolute Error
140000. 110000.
FIAT Panda 3ª serie Manuale Gpl 2018
Mean Absolute Error
63000. 91000.
FIAT Panda 3ª serie Manuale Benzina 2015
Mean Absolute Error
74000. 110000.
FIAT Panda 2ª serie Manuale Metano 2009
Mean Absolute Error
60000. 90000.
FIAT Panda Manuale Benzina 1997
Mean Absolute Error
140000. 200000.
FIAT Panda 2ª serie Manuale Metano 2008
Mean Absolute Error
61000. 91000.
FIAT Panda Manuale Benzina 2002
Mean Absolute Error
94000. 100000.
FIAT 500L Manuale Diesel 2016
Mean Absolute Error
69000. 76000.
SMART fortwo 2ª serie Automatico Benzina 2008
Mean Absolute Error
66000. 76000.
FIAT 500X Manuale Diesel 2018
Mean Absolute Error
49000. 77000.
FIAT Panda 2ª serie Manuale Diesel 2010
Mean Absolute Error
120000. 120000.
FIAT Panda 3ª serie Manuale Benzina 2013
Mean Absolute Error
77000. 99000.
FIAT Grande Punto Manuale Benzina 2009
Mean Absolute Error
63000. 110000.
FIAT Panda 3ª serie Manuale Benzina 2016
Mean Absolute Error
74000. 110000.
VOLKSWAGEN Golf 6ª serie Manuale Diesel 2010
Mean Absolute Error
120000. 130000.
FIAT Panda 3ª serie Manuale Benzina 2014
Mean Absolute Error
80000. 91000.
Price (€)
Km
[normalized]
Neural Net
Linear Regression
Test Set Points