In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

In [0]:

```
!unzip -qq "/content/drive/My Drive/SDS Group Stuff/craigslistVehicles.csv.zip"
```

```
replace craigslistVehicles.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: a
error:  invalid response [a]
replace craigslistVehicles.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: a
error:  invalid response [a]
replace craigslistVehicles.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: a
error:  invalid response [a]
replace craigslistVehicles.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
```

In [0]:

```
import pandas as pd
import numpy as np
import gdown
from skimage import io
import matplotlib.pyplot as plt
from keras.preprocessing import image
from keras.preprocessing.image import save_img
from sklearn.preprocessing import StandardScaler
from sklearn.utils import random
import seaborn as sns

import httplib2
import urllib
from io import BytesIO

import concurrent.futures
import glob
import shutil
import os

sns.set()
```

In [0]:

```
data = pd.read_csv("craigslistVehicles.csv")
```

Data Preparation

Here we prepare the data. We start out by removing the rows where the columns we deemed important for determining the price of a car is NaN. Next filter the price. As we can see, the most expensive car in the dataset is about 3.7 billion dollars and some of them are free. We therefor set a mininum limit of 200 dollars (199) and find a suitable maximum using the 99.5 percentile, which is about 60.000 dollars

In [0]:

```
data =  data.dropna(subset = ["odometer", "manufacturer", "year", "transmission", "fuel"
, "cylinders", "drive", "type", "condition"])
data = data.reset_index(drop=True)
```

In [0]:

```
print("most expensive car:", data["price"].max(), "dollars")
data = data[data["price"] <= 60000]
```

```
data = data[data["price"] >= 199]
print("most expensive car after limiting the data:", data["price"].max(), "dollars")
```

```
most expensive car: 2793940305 dollars
most expensive car after limiting the data: 60000 dollars
```

Here we showcase our price class determination for each car. We spread each price class with a difference of 750 dollars, which make up to 80 price classes, since the most expensive car is 59.995. The reason we do this is to make a more precise and a better estimate of a price for a new car in the future, with our neural network. The price column alone has many different unique values, and is therefore made more compact and more precise.

We end this by mapping the new found price class to a string, that describes the price range.

In [0]:

```
category_span = 750
```

In [0]:

```
floor_price_class = int(data["price"].max() / category_span)
print("class:", (76) * category_span, "to", (76 + 1) * category_span);
```

```
class: 57000 to 57750
```

We save the price class, both as numerical data, and categorical, in order to create indicator variables later with `pd.get_dummies()`.

In [0]:

```
data["price_class"] = data["price"] / category_span
data["price_class"] = data["price_class"].astype(int)
```

In [0]:

```
def price_class_to_string(i):
  return "{0} to {1}".format(i * category_span, (i + 1) * category_span)
```

In [0]:

```
data["price_class_label"] = data["price_class"].apply(lambda x:  price_class_to_string(x
))
```

Picking out samples from the data, checking their image url and gathering the images

In order to collect the images that we are working with, we first pick out 50.000 random samples from our dataset. But we also know that some of the image urls in the dataset does not work, so we have to check that first. We create 1.000 threads (yes, this might be overkill) to check all the links, and save all the samples that are actually working.

We split the random samples in 1000 smaller arrays, and assign one to each thread. This makes sure that the threads won't write to the same slot in memory.

In [0]:

```
# we pick out 50.000 samples from our dataset
samples = random.sample_without_replacement(len(data.index), 50000, random_state = 42)
```

In [0]:

```
samples_splits = []

splits = 1000
samples_len = len(samples)
```

```
for i in range(splits):
  samples_splits.append([])

for i in range(samples_len):
  samples_splits[int(i / (samples_len / splits))].append(samples[i])
```

After the samples are split, we prepare a similar array for all the working samples. This is being filled by the threads, and they try to connect with the url, belonging to each sample

In [0]:

```
working_samples_lists = []

for i in range(splits):
  working_samples_lists.append([])


def thread_function(name, sample_indexes, working_samples):
  for i in sample_indexes:
    with urllib.request.urlopen(data.iloc[i, 18]) as url:
      if url.code != 404:
        working_samples.append(i)

with concurrent.futures.ThreadPoolExecutor(max_workers=splits) as executor:
  executor.map(thread_function, range(splits),
               samples_splits,
               working_samples_lists)
```

Now we prepare to collect all the images, we again use 1.000 threads. Here we combine the index together with the url and split it into 1.000 seperate arrays, one for each thread. In this manor, we ensure that the threads won't interfere with one another.

We did this over two arrays to work more effeciently, since the indexes was needed for coding and testing in the baseline ML models and our textual neural network model. Therefore the image retrieval was put on hold, since it took much more time.

In [0]:

```
image_lists = []

for i in range(splits):
  image_lists.append([])

url_lists = []

for i in range(len(working_samples_lists)):
  url_lists.append([])
  for k in range(len(working_samples_lists[i])):
    url_lists[i].append([working_samples_lists[i][k], data.iloc[working_samples_lists[i]
[k], 18]])
```

In [0]:

```
# just making sure, that the index stored in the url list actually yields the same result
,
# when the url is taken from the dataset with that index
sample_index = url_lists[0][0][0]
print(data.iloc[sample_index, 18])
print(url_lists[0][0][1])
```

```
https://images.craigslist.org/00C0C_6lKdRpdGSw4_600x450.jpg
https://images.craigslist.org/00C0C_6lKdRpdGSw4_600x450.jpg
```

Here we start the search for the images with our threads. We see that the `url_lists` and `image_lists` are passed to the threads.

In [0]:

```
def thread_function(name, urls, images):
  for i in urls:
    with urllib.request.urlopen(i[1]) as url:
      images.append([i[0], image.load_img(BytesIO(url.read()), target_size=(128, 128))])

with concurrent.futures.ThreadPoolExecutor(max_workers=splits) as executor:
  executor.map(thread_function, range(splits),
               url_lists,
               image_lists)
```

In [0]:

```
#just checking that the images and indexes still match
sample_index = image_lists[56][12][0]
print(data.iloc[sample_index, 18])
image_lists[56][12][1]
```

https://images.craigslist.org/00J0J_d7MtSYneqwj_600x450.jpg

Out[0]:

Saving the images

We actually save the images in two different ways, as we were unsure which format to use when building the networks.

Saving the car images, based on price category

This first way of saving the images splits the images in a train and test folder, as well as a sub folder with the price category. This makes it easy to use a generator in keras to train with the images.

In [0]:

```
# This piece of code, just tries to delete the folder named "car_images", and then create
it again.
# This is just a help, in case we end up saving the images wrong, then it is easy to clea
n up.
try:
  shutil.rmtree('car_images')
except OSError:
  pass

try:
  os.mkdir("car_images")
  os.mkdir("car_images/test")
  os.mkdir("car_images/train")
except OSError:
  pass
```

In [0]:

```
flat_image_list = []

for sublist in image_lists:
    for k in sublist:
        flat_image_list.append(k)
```

In [0]:

```
train_max_index = int(len(flat_image_list) * 0.7)

for i in range(train_max_index):
  folderpath = "car_images/train/{0}".format(data.iloc[flat_image_list[i][0]]["price_cla
ss_label"])

  try:
      os.mkdir(folderpath)
  except OSError:
      pass

  path = "{0}/image_{1}.jpg".format(folderpath, flat_image_list[i][0])
  save_img(path, flat_image_list[i][1])

for i in range(train_max_index, len(flat_image_list)):
  folderpath = "car_images/test/{0}".format(data.iloc[flat_image_list[i][0]]["price_clas
s_label"])

  try:
      os.mkdir(folderpath)
  except OSError:
      pass

  path = "{0}/image_{1}.jpg".format(folderpath, flat_image_list[i][0])
  save_img(path, flat_image_list[i][1])
```

In [0]:

```
Here we zip the file and move it to our shared drive folder, so that it is easily accesi
ble from another notebook
```

In [0]:

```
!zip -r "car_images.zip" "car_images"
```

In [0]:

```
shutil.move("car_images.zip", "/content/drive/My Drive/SDS Group Stuff")
```

Out[0]:

```
'/content/drive/My Drive/SDS Group Stuff/car_images.zip'
```

Saving a sorted car image list

Here we save the images in sorted order, and with a name that indicates which index in the pruned dataset, that they belong to

In [0]:

```
sorted_flat_image_list = sorted(flat_image_list, key=lambda x: x[0])
```

In [0]:

```
sorted_flat_image_list[7][0]
```

Out[0]:

```
34
```

In [0]:

```
shutil.rmtree('sorted_car_images')
```

In [0]:

```
for i in range(len(sorted_flat_image_list)):
  path = "sorted_car_images/{0}.jpg".format(sorted_flat_image_list[i][0])
  save_img(path, sorted_flat_image_list[i][1])
```

Again we zip the folder with the images and move it to our drive folder

In [0]:

```
!zip -r "sorted_car_images.zip" "sorted_car_images"
```

In [0]:

```
shutil.move("sorted_car_images.zip", "/content/drive/My Drive/SDS Group Stuff")
```

Out[0]:

```
'/content/drive/My Drive/SDS Group Stuff/sorted_car_images.zip'
```

We also needed the indexes, where each image belongs, in a format that is easy readable. We therefore write all the indexes to a csv, and move it to our drive folder

In [0]:

```
img_list = []
for i in range(len(sorted_flat_image_list)):
  img_list.append(sorted_flat_image_list[i][0])
```

In [0]:

```
df = pd.DataFrame(img_list)
df.to_csv("sorted_image_list.csv")
```

In [0]:

```
shutil.move("sorted_image_list.csv", "/content/drive/My Drive/SDS Group Stuff")
```

Out[0]:

```
'/content/drive/My Drive/SDS Group Stuff/sorted_image_list.csv'
```