Business case Introduction

Cars24 is one of the leading used-car buying and selling platform in India.

Current methodology to estimate the selling price has three major components

1. Past records of the "Selling Prices" of the same *Make* and *Model*
2. Rule based calculations for the depreciation of the car
3. Rule based calculation for condition of the vehicle

As the **Pricing Data Scientist working at Cars24, you are given a task**

GOAL: To automate the process of estimating price of the used car

Exploratory Data Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
!gdown 18QuoyuqmNKmR9mba5he4ojPc4KzkIs3t
```

```
Downloading...
From: https://drive.google.com/uc?id=18QuoyuqmNKmR9mba5he4ojPc4KzkIs3t
To: /content/train-cars24-car-price.csv
100% 1.25M/1.25M [00:00<00:00, 119MB/s]
```

Overview

```
df = pd.read_csv("train-cars24-car-price.csv")
df.head()
```

	full_name	selling_price	year	seller_type	km_driven	fuel_type	transmission_type	mileage	en
0	Maruti SX4 Zxi BSIII	2.85	2007.0	Individual	110000	Petrol	Manual	15.00	15
1	Hyundai i20 Sportz 1.4 CRDi	4.70	2012.0	Dealer	70000	Diesel	Manual	21.90	13
2	Maruti Swift VDI BSIV	5.25	2015.0	Individual	70000	Diesel	Manual	25.20	12

Notice that Variable `full_name` contains both `Maker` and `Model` information - need to extract make and model from full_name.

```
df.shape
```

```
(13986, 11)
```

```
df.info();
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13986 entries, 0 to 13985
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   full_name          13986 non-null  object 
 1   selling_price      13986 non-null  float64
 2   year               13986 non-null  float64
 3   seller_type        13986 non-null  object 
 4   km_driven          13986 non-null  int64  
 5   fuel_type          13986 non-null  object 
 6   transmission_type  13986 non-null  object 
 7   mileage            13986 non-null  float64
```

```
 8   engine             13986 non-null  float64
 9   max_power          13986 non-null  float64
 10  seats              13986 non-null  float64
dtypes: float64(6), int64(1), object(4)
memory usage: 1.2+ MB
```

```
display(df.describe())
```

	selling_price	year	km_driven	mileage	engine	max_power	seats
count	13986.000000	13986.000000	1.398600e+04	13986.000000	13986.000000	13986.000000	13986.000000
mean	7.377749	2014.516088	5.797629e+04	19.356221	1473.574905	97.685631	5.313242
std	9.368596	3.256795	5.378180e+04	4.638576	518.289204	45.067944	0.831653
min	0.325000	1991.000000	1.000000e+02	0.000000	0.000000	5.000000	2.000000
25%	3.400000	2013.000000	3.100000e+04	16.840000	1197.000000	73.900000	5.000000
50%	5.100000	2015.000000	5.120000e+04	19.160000	1248.000000	86.700000	5.000000
75%	7.800000	2017.000000	7.397800e+04	22.320000	1582.000000	112.000000	5.000000
max	395.000000	2021.000000	3.800000e+06	120.000000	6752.000000	626.000000	14.000000

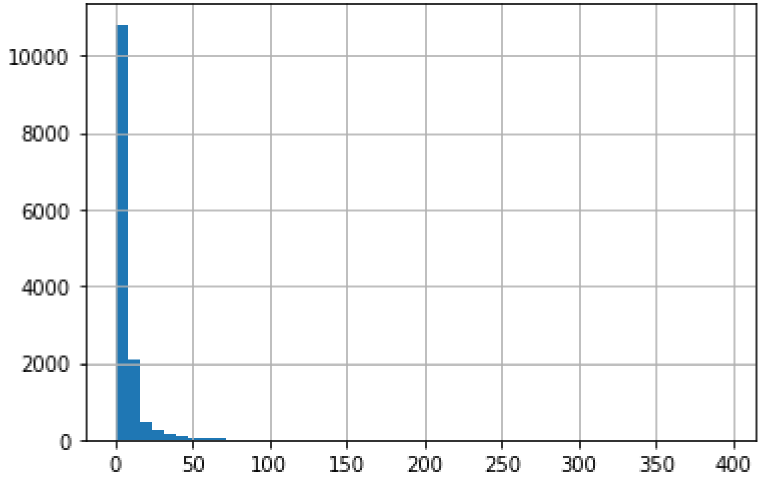
- `selling_price` ranges from [0.33, 395] - the values seem to be in lakhs.
- `mileage` min is 0, max is 120 - need to check for outliers.
- `engine` min is 0, max is 6752 - values appear to be in "cc" - need to check for outliers.

Distributions

Lets check the distributions of these features one by one

```
df['selling_price'].hist(bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a91998350>
```



Lets check the percntage of outliers. We are manually taking a call that price over a 100 (K) is an outlier by looking at the above plot

```
(df.loc[df.selling_price > 100].shape[0] / df.shape[0]) * 100
```

```
0.06435006435006435
```

Lets saturate selling price at **100**.

```
df.loc[df.selling_price > 100, 'selling_price'] = 100
```

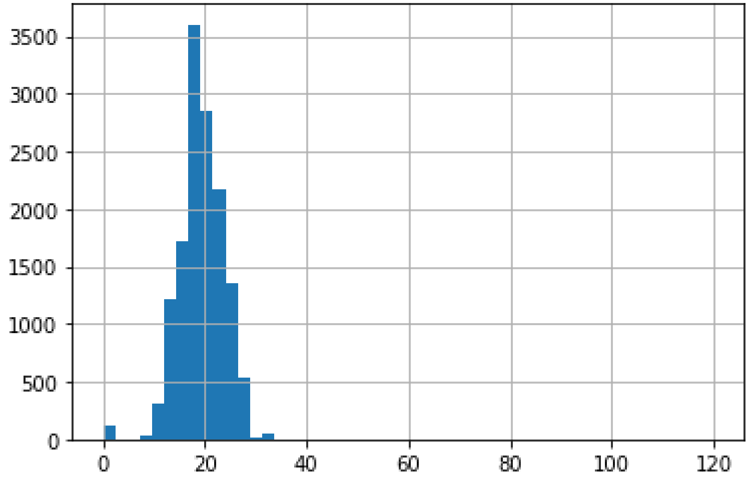
> - Can you think of an automated way of deciding this threshold 100 ? Seems like we made this decision arbitarily
> - **Hint:** use quantiles

Outliers and Missing Values

Q: Which plot(s) is/are suitable for outlier visualisation ?

Similarly lets plot a histogram for **mileage**

```
df["mileage"].hist(bins=50);
```



Another useful plot is called the 'density' plot. Which is also similar to histogram, but with a more *approximate* shape. It tries to fit a suitable distribution over the histogram and hence includes points that might not even be present in data at hand.

```
df["mileage"].plot.density();
```

We observe a skewed plot with right tail to long. An interesting point here would be mileage > 40, because after that it looks like the values mught just be outliers.

Above we saw that there was an unusual peak at **0**. Lets investigate further. Looking at points where mileage is less than 5

```
df[df["mileage"]<=5].mileage.describe()
```

```
count    119.000000
mean       0.033613
std        0.366679
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        4.000000
Name: mileage, dtype: float64
```

Looks like we just have a number of points with mileage 0. That does not make much sense as it cannot be a real value for car mileage. Investigating mileage=0 cases

```
df.loc[df.mileage == 0].shape[0] / df.shape[0] # 0.8 %
```

```
0.008437008437008437
```

```
df[df["mileage"] == 0.0].shape
```

```
(118, 11)
```

Thats a very small number of points. We can safely impute, or drop these points later during pre-processing

Are the top values errorneous?

Above we had also noticed that there was a significant point at mileage=40. After this there were only very few points and looked like outliers. Lets check them

```
df[df["mileage"] > 40].mileage.describe()
```

```
count      3.000000
mean     116.666667
std        5.773503
min      110.000000
25%      115.000000
50%      120.000000
75%      120.000000
max      120.000000
Name: mileage, dtype: float64
```

```
df[df["mileage"] > 40]
```

	full_name	selling_price	year	seller_type	km_driven	fuel_type	transmission_type	mileage
4291	Mahindra e2o Premium	3.25	2013.0	Individual	50000	Electric	Automatic	120.0
4751	Mahindra e2o T2	3.60	2015.0	Dealer	42862	Electric	Automatic	120.0

```
df[df["mileage"] > 40].fuel_type.value_counts()
```

```
Electric    3
Name: fuel_type, dtype: int64
```

- So all vehicles with mileage > 40 are electric vehicles which makes sense.

- So they are not incorrect (error) values.
- Let us not remove these points for now

Correlations

Now lets look at the correlations of numerical features with each other

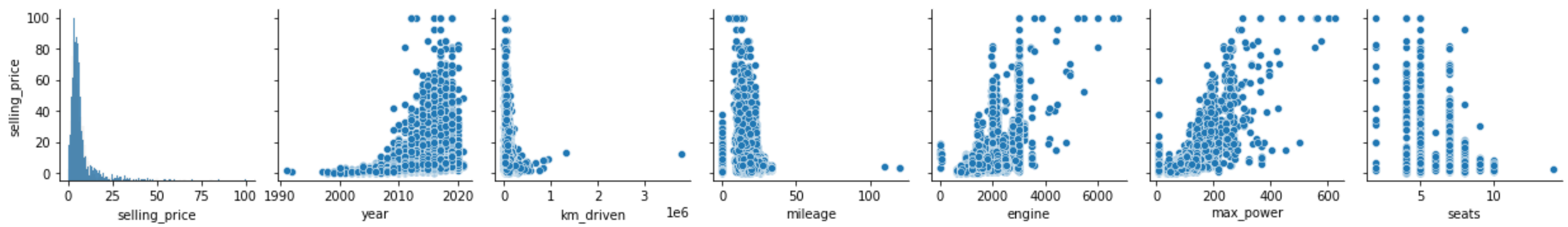
```
plt.figure(figsize=(10,8))
ax = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

We see the following correlations:

- Engine and max-power
- Max power and selling price Now based on our **domain knowledge** (about cars) we can digest these correlations. In this case, these correlations seem to make a lot of sense

Another visualisation we can check, is the scatter plot of each feature with the target. This can be easily done using pairplot from seaborn library


```
sns.pairplot(df, y_vars=["selling_price"]);
```




```
df['seats'].value_counts(normalize=True)
```

```
5.0     0.837123
7.0     0.115902
8.0     0.022237
4.0     0.011440
6.0     0.007293
9.0     0.003432
10.0    0.001502
2.0     0.001001
```

```
14.0    0.000072
Name: seats, dtype: float64
```

Categorical Variables

How can we extract Make and Model from "full_name"

We have noticed that the name contains the informaiton of the brand and the model. Lets extract those and create new features from them. After that we can drop the original column

```
df["make"] = df.full_name.apply(lambda x : x.split()[0])
df["model"] = df.full_name.apply(lambda x : " ".join(x.split()[1:]))
df = df.drop("full_name", axis=1)
df.head(2)
```

	selling_price	year	seller_type	km_driven	fuel_type	transmission_type	mileage	engine	max_po
0	2.85	2007.0	Individual	110000	Petrol	Manual	15.0	1586.0	10

Now let us look at some categorical variables one by one

```
display(df.describe(include="object"))
```

seller_type fuel_type transmission_type make model

```
df["fuel_type"].value_counts()
```

```
Petrol      6870
Diesel      6823
CNG          233
LPG           49
Electric      11
Name: fuel_type, dtype: int64
```

```
df["transmission_type"].value_counts()
```

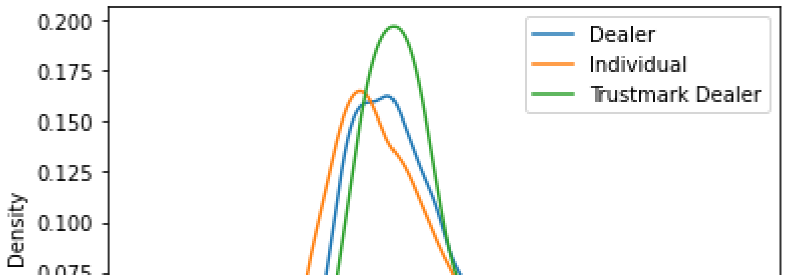
```
Manual       11251
Automatic     2735
Name: transmission_type, dtype: int64
```

```
df["seller_type"].value_counts()
```

```
Dealer              8393
Individual          5450
Trustmark Dealer     143
Name: seller_type, dtype: int64
```

Does selling price vary in some of these groups

```
df.loc[df.selling_price<=df.selling_price.quantile(0.9)].groupby('seller_type')['selling_price'].plot.density();
plt.legend();
```



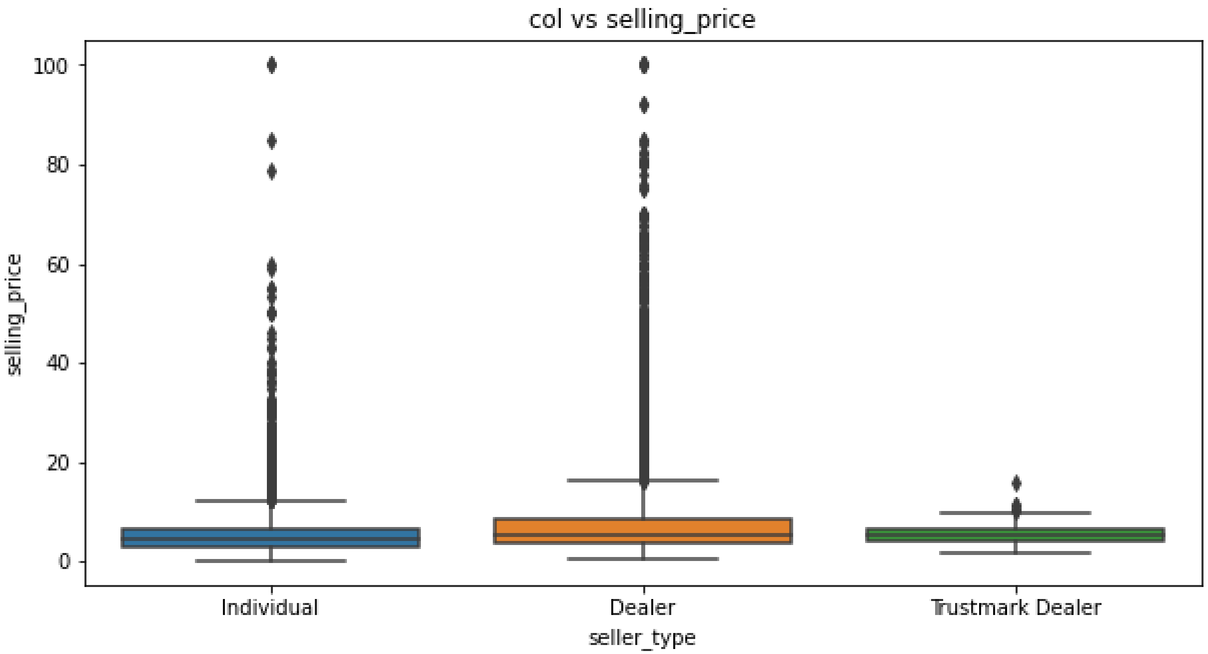
Looks like for different seller type, the selling price distrubution is not very different. Hence one intuition that can be developed from this plot is that the **seller_type** variable will not turn out to be very important for the ML model. (We will check this later)

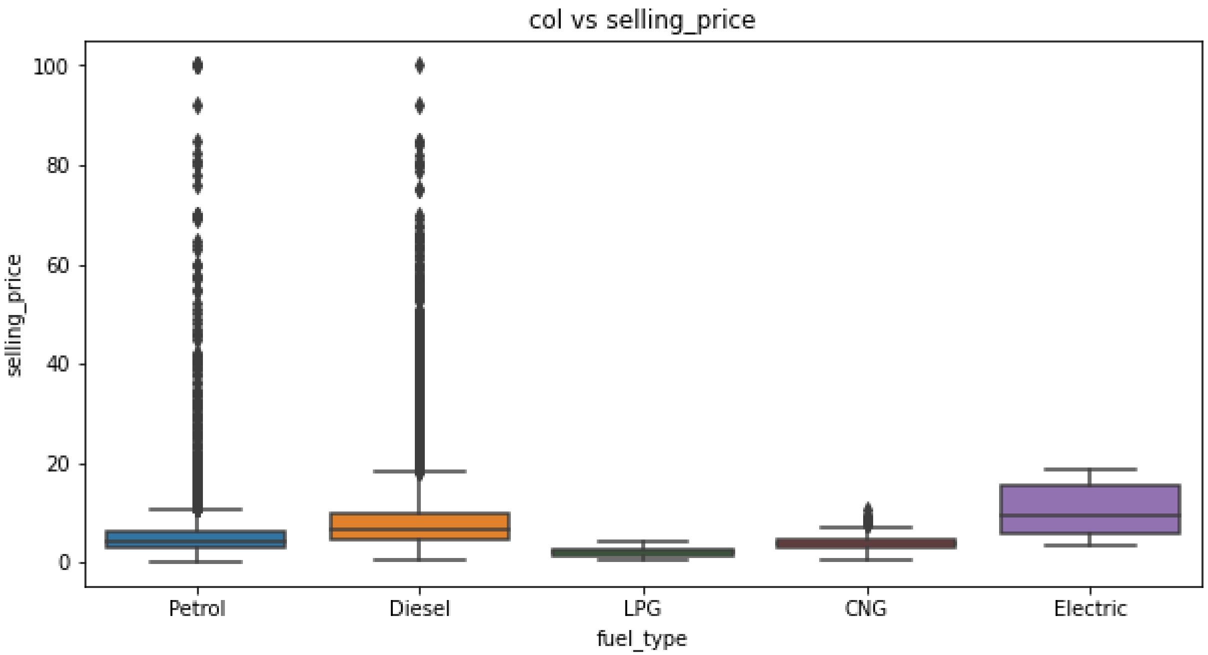
Lets check for all groups?

Another way to visualise the same is to use a plot known as **box-plot**

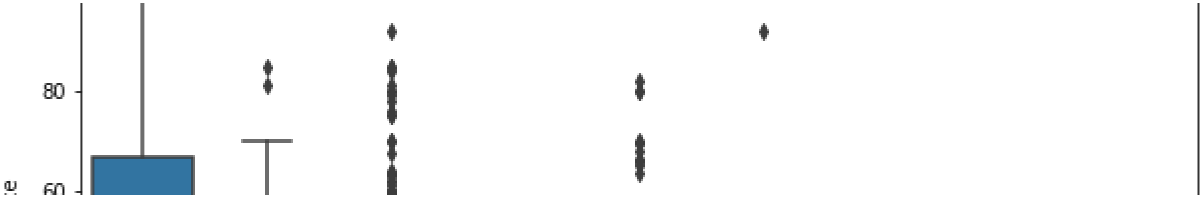
Note: Although, `seats` is numeric, but we will consider it as an ordinal variable and perform similar analysis as we do for other categorical variables.

```
for col in ['seller_type', 'fuel_type', 'seats']:
    plt.figure(figsize=(10,5))
    sns.boxplot(y='selling_price',x=col, data=df)
    plt.title('col vs selling_price')
    plt.show()
```


col vs selling_price
selling_price
seller_type
Individual
Dealer
Trustmark Dealer
0
20
40
60
80
100


col vs selling_price
selling_price
fuel_type
Petrol
Diesel
LPG
CNG
Electric
0
20
40
60
80
100

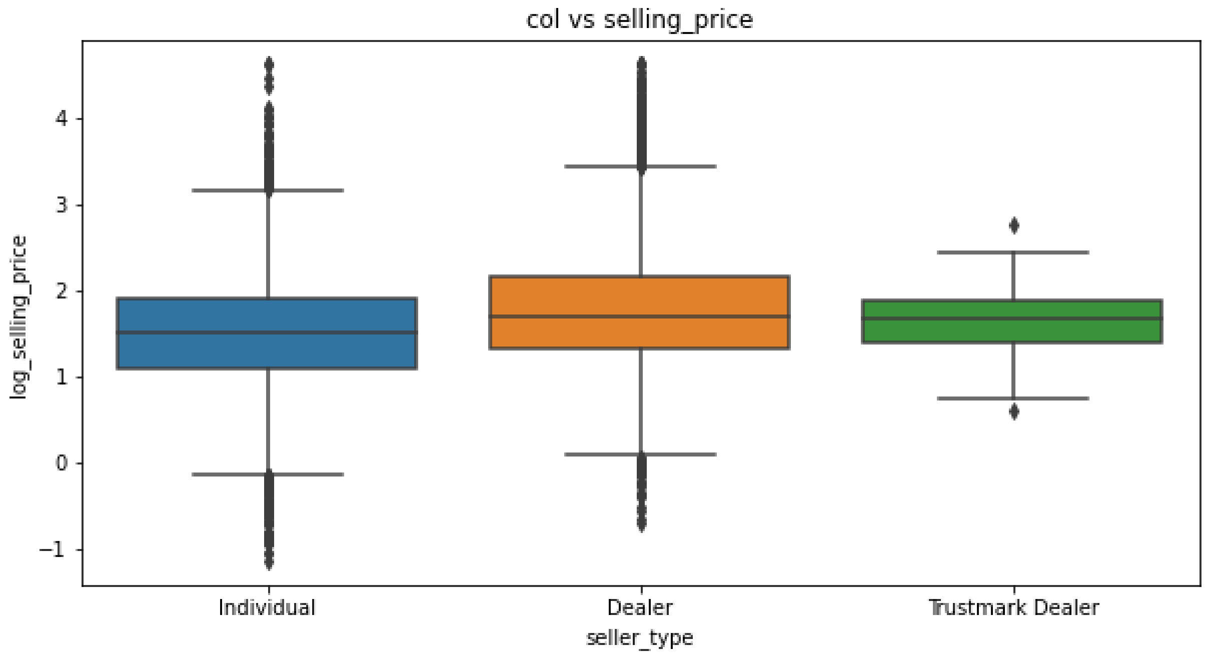

col vs selling_price
100

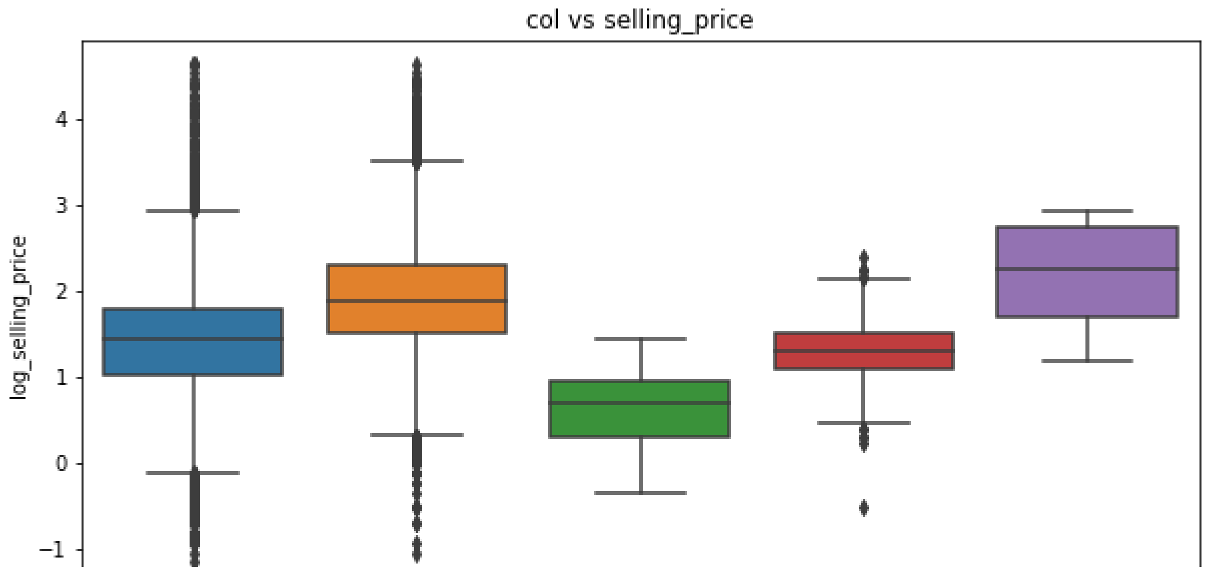


How can we transform the scale of selling-price for better visualisation?

```
df["log_selling_price"] = np.log(df["selling_price"].values)

for col in ['seller_type', 'fuel_type', 'seats']:
    plt.figure(figsize=(10,5))
    sns.boxplot(y='log_selling_price',x=col, data=df)
    plt.title('col vs selling_price')
    plt.show()
```





- `selling_price` of "coupe" is quite variational - important to be kept for predictive modelling

How many categories of "make" in the data?

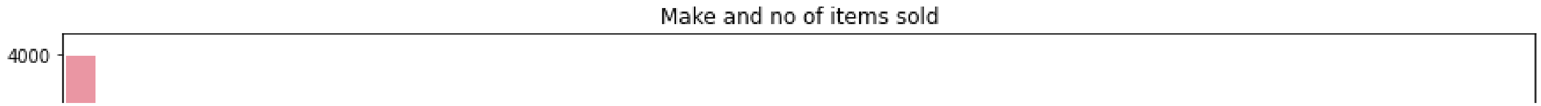
```
len(df["make"].unique())
```

```
41
```

There are 41 car brands in the data - too many levels, think of possible ways to encode them as numbers

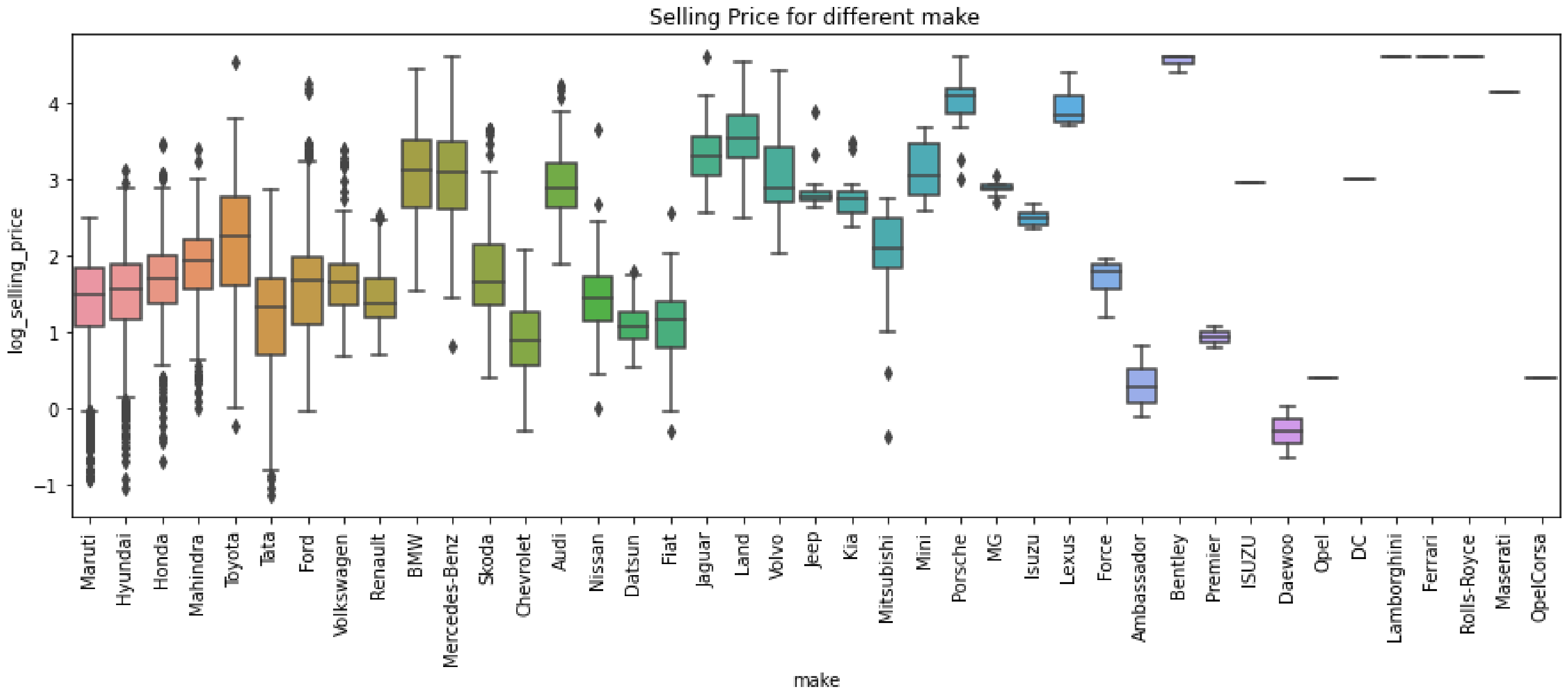
What are item counts for different "make"?

```
plt.figure(figsize=(15,5))
sns.countplot(x='make', data=df, order = df['make'].value_counts().index)
plt.xticks(rotation=90);
plt.title("Make and no of items sold")
plt.show()
```



How is "make" related to the selling price?

```
plt.figure(figsize=(15,5))
sns.boxplot(y='log_selling_price',x='make', data=df, order = df['make'].value_counts().index)
plt.xticks(rotation=90);
plt.title("Selling Price for different make")
plt.show()
```



Lets drop this column for now

```
df.drop("log_selling_price", axis=1, inplace=True)
```

> - How should we treat number of seats? Continuous OR Categorical ?
> - **Answer**:
> -
>   - Number of seats is a variable with a finite discrete set of values. However it is orderd. If we make it categorical, it would be fed into the model as unordered.
> -
>   - But we need to think do we want that order? Example 2 seater cars are most likely more expensive than other cars. So in this sense 2 > 5. We actually would benefit from discarding the order.

> - Can you convert number of seats to a variable which has 9 unique values with 3 categories?
> - **Hint**: Merge some elements. Eg all cars with > 5 seats are on category.
> - Can we do something similar with the "make variable"

Feature Engineering

```
df.head()
```

	selling_price	year	seller_type	km_driven	fuel_type	transmission_type	mileage	engine	max_po
0	2.85	2007.0	Individual	110000	Petrol	Manual	15.00	1586.0	10
1	4.70	2012.0	Dealer	70000	Diesel	Manual	21.90	1396.0	8
2	5.25	2015.0	Individual	70000	Diesel	Manual	25.20	1248.0	7
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

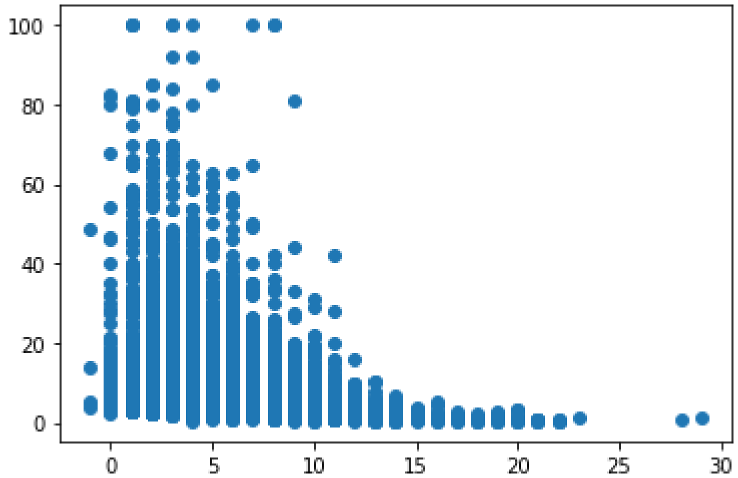
1. Year

Sometimes it is good practice to convert the features into the way we actually understand. This will not necessarily help the model, but helps in us in understanding / interpreting predictions little better

- So lets convert year to more readable ***age***

```
current_year = 2020
age = current_year - df['year']
plt.scatter(age, df.selling_price)
```

```
<matplotlib.collections.PathCollection at 0x7f3a8eafdb10>
```



Higher selling price for newer cars !!

2. Categorical Encoding

Q: How do we encode categorical variables for modelling?

2.1 transmission_type (2 categories)

```
# explains how two category variable can be encoded as a binary variable
transmission_dummy = pd.get_dummies(df["transmission_type"])
df = pd.concat([df, transmission_dummy], axis=1)
df = df.drop(["transmission_type", "Automatic"], axis=1)
df = df.rename(columns={"Manual": "transmission_type"})
df.sample(frac=1).head(2)
```

	selling_price	year	seller_type	km_driven	fuel_type	mileage	engine	max_power	seats	mak
4755	7.75	2017.0	Dealer	114000	Diesel	24.3	1248.0	88.5	5.0	Maru

2.2 fuel_type (5 categores)

```
fuel_type_dummy = pd.get_dummies(df["fuel_type"], prefix="seats")
fuel_type_dummy.sample(frac=1).head(2)
```

	seats_CNG	seats_Diesel	seats_Electric	seats_LPG	seats_Petrol
2013	0	0	0	0	1
9172	0	0	0	0	1

NOTE: We remove 1 of the dummy variables to avoid the dummy variable trap

```
fuel_type_dummy = fuel_type_dummy[fuel_type_dummy.columns[1:]]
fuel_type_dummy.head(2)
```

seats_Diesel	seats_Electric	seats_LPG	seats_Petrol

```
pd.concat([df, fuel_type_dummy], axis=1).sample(frac=1).head(2)
```

	selling_price	year	seller_type	km_driven	fuel_type	mileage	engine	max_power	seats	ı
3564	18.99	2018.0	Dealer	20306	Diesel	15.10	2179.0	152.87	7.0	Mahi
989	7.50	2016.0	Dealer	62563	Diesel	28.09	1248.0	88.50	5.0	M

2.3 make (too many categoies)

```
df.make.nunique()
```

41

> - How to handle so many categories?
> - **Hint:** Here we have too many categories. We can merge categories as in 'seats' variable or we can try target encoding. Dummy encoding would make too many features with highly diluted information.

Or we can try another technique known as **Target Encoding**

```
encoded_make = df.groupby('make')['selling_price'].transform('mean')
```

```
show_df = df[['make']].copy()
```

```
show_df['encoded_make'] = encoded_make
show_df = show_df.drop_duplicates()
print(len(show_df))
show_df.head()
```

41

	make	encoded_make
0	Maruti	4.648484
1	Hyundai	5.344764
3	Honda	5.911203
4	Volkswagen	5.632862
5	Toyota	11.179478

3. Scaling

Q: Now, All these variables have different ranges. Does that work?

```
df.describe()
```

	selling_price	year	km_driven	mileage	engine	max_power	seats
count	13986.000000	13986.000000	1.398600e+04	13986.000000	13986.000000	13986.000000	13986.000000
mean	7.327126	2014.516088	5.797629e+04	19.356221	1473.574905	97.685631	5.313242

Lets scale this data. We will use MinMaxScaler here (in practice you can choose your scaling techniques depending on the problem, or just try all and see what works best)

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler()
scaler.fit(df[['selling_price', 'km_driven', 'mileage']])
scaled_values = scaler.transform(df[['selling_price', 'km_driven', 'mileage']])  # returns numpy.ndarray not df.
scaled_df = pd.DataFrame(scaled_values, columns=['selling_price', 'km_driven', 'mileage'])
scaled_df.head()
```

	selling_price	km_driven	mileage
0	0.025332	0.028922	0.125000
1	0.043893	0.018395	0.182500
2	0.049411	0.018395	0.210000
3	0.009280	0.023659	0.108333
4	0.043391	0.010763	0.137250

```
scaled_df.describe()
```

	selling_price	km_driven	mileage
count	13986.000000	13986.000000	13986.000000
mean	0.070250	0.015231	0.161302
std	0.083378	0.014153	0.038655
min	0.000000	0.000000	0.000000
25%	0.030850	0.008132	0.140333
75%	0.074994	0.019442	0.186000

- Try standard scaler and print "df.describe", and make density plots for both std scaler and minmax scaler

Stichting it all together

```
def merge_seats(x):
    if 2 <= x <= 4:
        return '2-4'
    elif x > 5:
        return '>5'
    else:
        return '5'

def preprocess(df):
    df = df.loc[df.mileage != 0].copy()
    outlier_theshold = df.selling_price.quantile(0.95)
    df.loc[df.selling_price > outlier_theshold, 'selling_price'] = outlier_theshold
    df['age'] = pd.to_datetime('now', utc=True).date().year - df['year']
    df['full_name'] = df['full_name'].str.upper()
    df['seats'] = df['seats'].apply(merge_seats)
    df.drop(columns=['year'])
    return df

def feature_engineering(df):
```

```
    df['make'] = df.full_name.apply(lambda x : x.split()[0])
    df['model'] = df.full_name.apply(lambda x : " ".join(x.split()[1:]))
    one_hot_encode_cols = ['seller_type', 'fuel_type', 'transmission_type', 'seats']
    for column in one_hot_encode_cols:
        df = pd.concat([df, pd.get_dummies(df[column]).iloc[: , 1:]], axis=1)
    df.drop(columns=one_hot_encode_cols + ['full_name'], inplace=True)
    df['make'] = df.groupby('make')['selling_price'].transform('median')
    df['model'] = df.groupby('model')['selling_price'].transform('median')
    scaler = MinMaxScaler()
    df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
    return df
```

```
df = pd.read_csv("train-cars24-car-price.csv")
df = preprocess(df)
df = feature_engineering(df)
df.head()
```

	selling_price	year	km_driven	mileage	engine	max_power	age	make	model	Indiv
0	0.122128	0.517241	0.028922	0.094828	0.234893	0.160515	0.482759	0.179031	0.093108	
1	0.211608	0.689655	0.018395	0.154310	0.206754	0.134879	0.310345	0.203759	0.200726	
2	0.238210	0.793103	0.018395	0.182759	0.184834	0.111111	0.206897	0.179031	0.224184	
3	0.044740	0.448276	0.023659	0.077586	0.198904	0.136876	0.551724	0.233432	0.087062	
4	0.209190	0.793103	0.010763	0.107500	0.177429	0.111111	0.206897	0.218595	0.229504	

```
df.shape
```

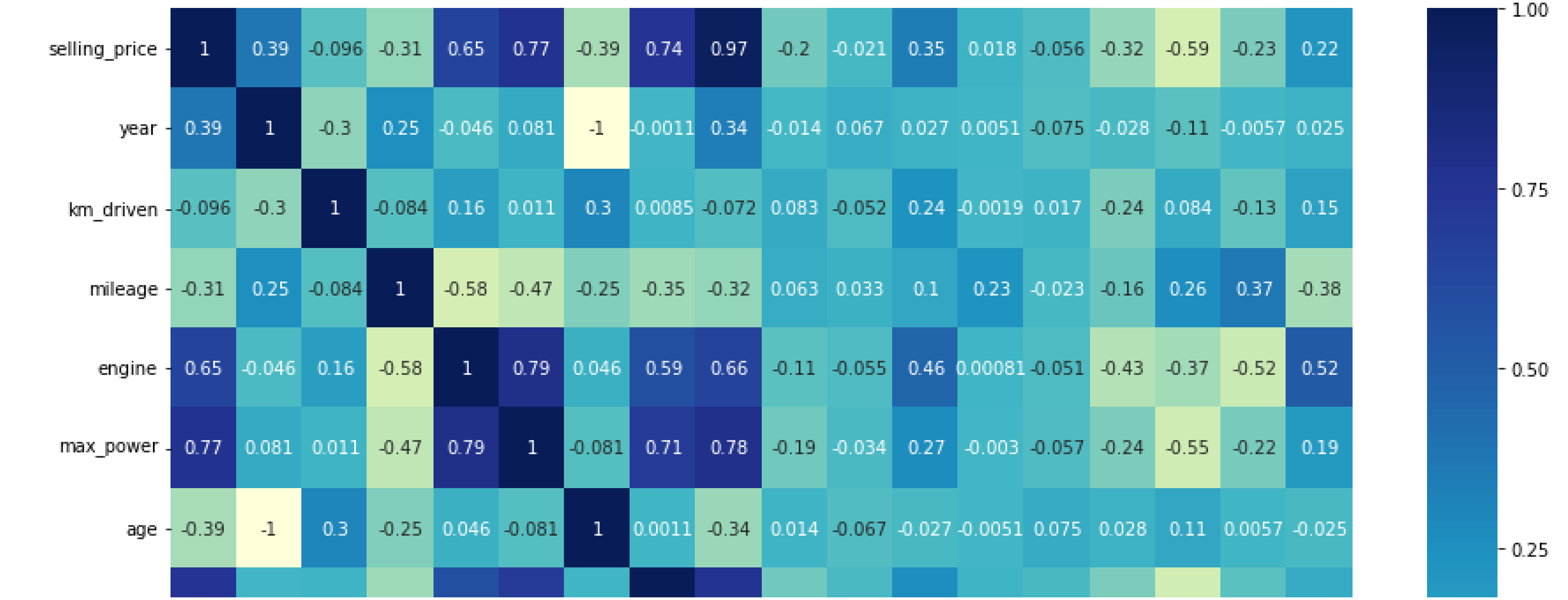
```
(13868, 18)
```

Correlations with all numeric features

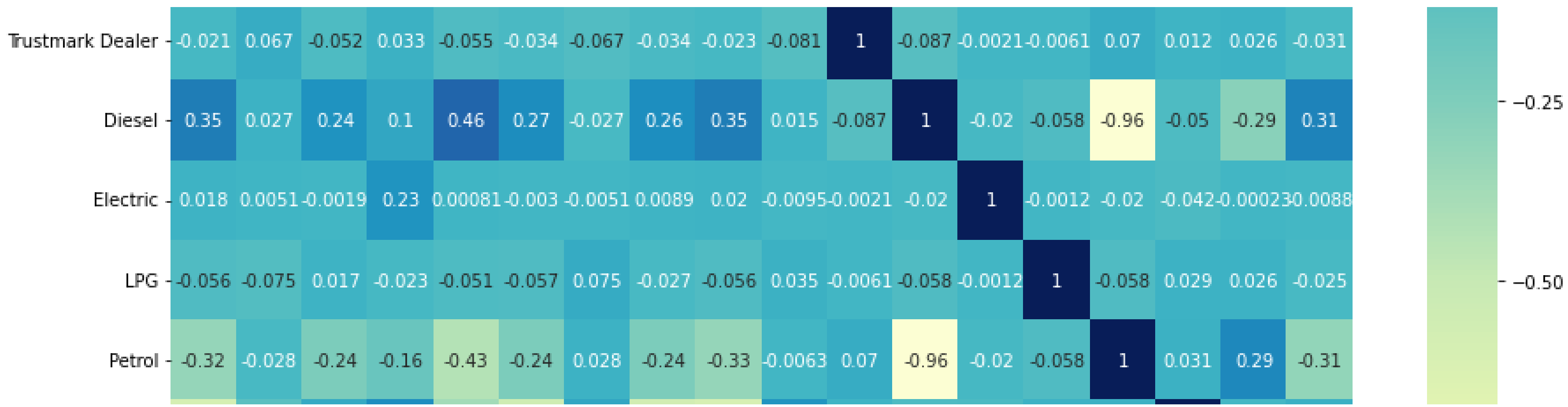
And before we wrap up, lets just see correlations of all variables since now we have all usable information in numerical form

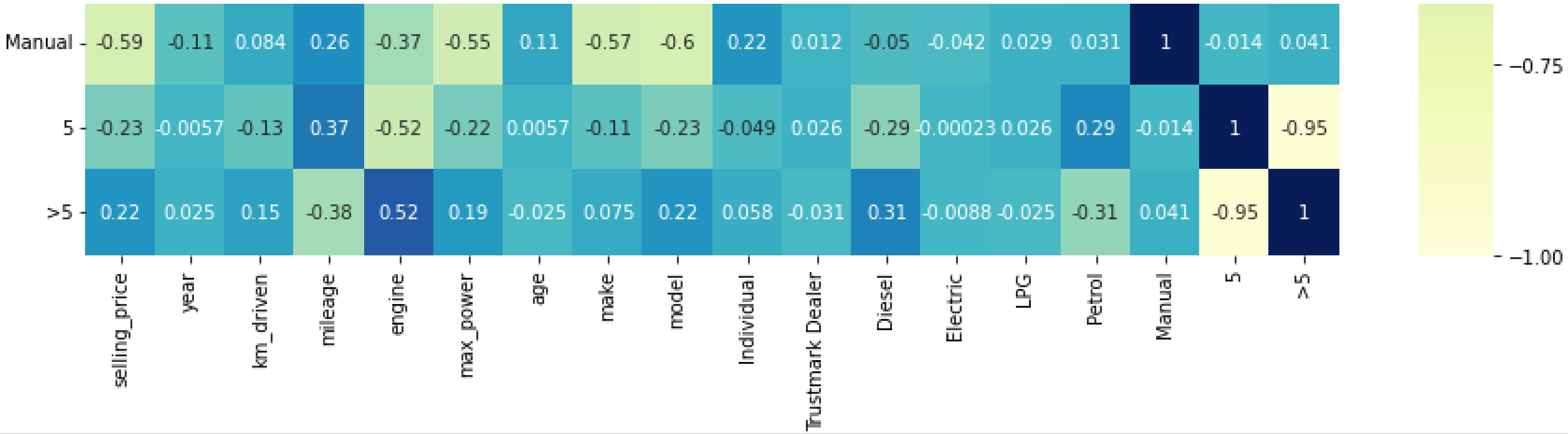
```
plt.figure(figsize=(15, 15))
sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a8eae9690>
```



- `max_power`, `engine` and `transmission_type` are hightest correlated with `selling_price` in same order.
- New encoded features are weak predictor, but they together would might make strong predictors.




Manual
-0.59 -0.11 0.084 0.26 -0.37 -0.55 0.11 -0.57 -0.6 0.22 0.012 -0.05 -0.042 0.029 0.031 1 -0.014 0.041
5
-0.23 -0.0057 -0.13 0.37 -0.52 -0.22 0.0057 -0.11 -0.23 -0.049 0.026 -0.29 -0.00023 0.026 0.29 -0.014 1 -0.95
>5
0.22 0.025 0.15 -0.38 0.52 0.19 -0.025 0.075 0.22 0.058 -0.031 0.31 -0.0088 -0.025 -0.31 0.041 -0.95 1
selling_price
year
km_driven
mileage
engine
max_power
age
make
model
Individual
Trustmark Dealer
Diesel
Electric
LPG
Petrol
Manual
5
>5
−0.75
−1.00