Análise exploratória de dados com Python

A Análise Exploratória de Dados (EDA), também conhecida como Exploração de Dados, é uma etapa onde várias técnicas são usadas para entender melhor o conjunto de dados utilizado.

"Entender o conjunto de dados" pode se referir a várias coisas, incluindo, mas não se limitando a:

- Extrair variáveis importantes e deixar para trás variáveis inúteis.
- Identificar outliers, valores ausentes ou erros humanos.
- Compreender as relações, ou a falta dela, entre as variáveis.
- Maximizar seus insights sobre um conjunto de dados e minimizar potenciais erros que possam ocorrer mais tarde no processo.

Leonardo Carvalho
leofacebook17@gmail.com

Por que isso é importante?

Ao realizar a EDA, você pode transformar um conjunto de dados **quase** utilizável em um conjunto de dados **completamente** utilizável. É aqui que vamos identificar os dados que precisam ser limpos, e entender melhor nosso conjunto

O que você precisa definir antes

Antes de sair fazendo uma EDA, defina o seu problema. Uma análise exploratória pode ser um verdadeiro buraco negro se você não define o que está procurando antes. Vou dar aqui algumas possibilidades pra você se inspirar:

- Quero entender melhor como está a saúde dos meus principais KPIs. Aqui, entenda o problema (qual business?) e os principais KPIs a serem acompanhados. A partir disso você pode começar a fazer uma análise,

sabendo que vai olhar cada KPI ao longo do tempo, encontrar possíveis causas para algum problema em algum deles, correlacionar, etc.

- Quero fazer um modelo de machine learning. Nesse caso, você precisa analisar a saúde do seu dado, ver se tem algum valor absurdo, quais tipos de dados tem, se suas variáveis se correlacionam com o target de uma forma mais direta, se há feature engineering para fazer, etc

Componentes da EDA

Para mim, qualquer que seja seu problema, existem componentes principais na exploração de dados:

- Compreender suas variáveis e saúde dos dados
- Limpar seu conjunto de dados caso necessário
- Analisar relações entre as variáveis

Vamos a cada uma delas!

Leonardo Carvalho

1. Compreendendo Suas Variáveis (Univariada)

Primeiro, importamos todas as bibliotecas necessárias para uma análise e realize algumas análises preliminares.

```
#Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns#Understanding my variables

#Entenda o dado
print(df.shape)
print(df.head())
print(df.columns)
print(df.info())
```

`.shape` retorna o número de linhas pelo número de colunas do meu conjunto de dados.

`.columns` retorna o nome de todas as suas colunas no conjunto de dados.

Shape of the Dataset

```
In [11]: df.shape

Out[11]: (15411, 13)
```

Our data has 15411 rows and 13 different features

Show features/columns of the Dataset

```
In [13]: df.columns

Out[13]: Index(['car_name', 'brand', 'model', 'vehicle_age', 'km_driven', 'seller_type',
       'fuel_type', 'transmission_type', 'mileage', 'engine', 'max_power',
       'seats', 'selling_price'],
      dtype='object')
```

`.head()` retorna as primeiras 5 linhas do meu conjunto de dados. Isso é útil se você quiser ver alguns valores de exemplo para cada variável.

```
In [8]: df.head()
```

Leonardo Carvalho

Out[8]:

	car_name	brand	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats	selling_price
0	Maruti Alto	Maruti	Alto	9	120000	Individual	Petrol	Manual	19.70	796	46.30	5	120000
1	Hyundai Grand	Hyundai	Grand	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	5	550000
2	Hyundai i20	Hyundai	i20	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	5	215000
3	Maruti Alto	Maruti	Alto	9	37000	Individual	Petrol	Manual	20.92	998	67.10	5	226000
4	Ford Ecosport	Ford	Ecosport	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	5	570000

`.info()` retorna o tipo das colunas e a quantidade de valores não nulos que temos nelas.

Check Null Value Counts and DataTypes of the features

In [15]:
```
df.info()
```

```
RangeIndex: 15411 entries, 0 to 15410
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   car_name           15411 non-null  object
 1   brand              15411 non-null  object
 2   model              15411 non-null  object
 3   vehicle_age        15411 non-null  int64
 4   km_driven          15411 non-null  int64
 5   seller_type        15411 non-null  object
 6   fuel_type          15411 non-null  object
 7   transmission_type  15411 non-null  object
 8   mileage            15411 non-null  float64
 9   engine             15411 non-null  int64
 10  max_power          15411 non-null  float64
 11  seats              15411 non-null  int64
 12  selling_price      15411 non-null  int64
dtypes: float64(2), int64(5), object(6)
memory usage: 1.5+ MB
```

Depois de conhecer todas as variáveis no conjunto de dados, podemos entender melhor os diferentes valores para cada variável.

`df.nunique(axis=0)`
`df.describe()`

`.nunique(axis=0)` retorna o número de valores únicos para cada variável.

`.describe()` resume a contagem, média, desvio padrão, mínimo e máximo para variáveis numéricas.

Summary of Data

In [14]:

```
### Display statistics of the DataFrame
df.describe()
```

Out[14]:

	vehicle_age	km_driven	mileage	engine	max_power	seats	selling_price
count	15411.000000	1.541100e+04	15411.000000	15411.000000	15411.000000	15411.000000	1.541100e+04
mean	6.036338	5.561648e+04	19.701151	1486.057751	100.588254	5.325482	7.749711e+05
std	3.013291	5.161855e+04	4.171265	521.106696	42.972979	0.807628	8.941284e+05
min	0.000000	1.000000e+02	4.000000	793.000000	38.400000	0.000000	4.000000e+04
25%	4.000000	3.000000e+04	17.000000	1197.000000	74.000000	5.000000	3.850000e+05
50%	6.000000	5.000000e+04	19.670000	1248.000000	88.500000	5.000000	5.560000e+05
75%	8.000000	7.000000e+04	22.700000	1582.000000	117.300000	5.000000	8.250000e+05
max	29.000000	3.800000e+06	33.540000	6592.000000	626.000000	9.000000	3.950000e+07

Fiz esse mesmo passo em um outro dataset que também fala sobre carros. Acompanhei esse código com uma breve análise. Veja abaixo

Leonardo Carvalho
leofacebook17@gmail.com

	price	year	odometer	lat	long
count	525839.000000	524399.000000	427248.000000	513618.000000	513618.000000
mean	61966.045503	2009.375184	101150.248338	38.470853	-94.259216
std	9949703.964406	8.975889	105525.184435	5.895637	17.742010
min	0.000000	1900.000000	0.000000	-83.668334	-176.748047
25%	3900.000000	2006.000000	49009.750000	34.669400	-108.386684
50%	8999.000000	2011.000000	94240.000000	39.213214	-88.544050
75%	17900.000000	2015.000000	138000.000000	42.445565	-80.990754
max	3048344231.000000	2020.000000	10000000.000000	78.928357	132.078349

Imediatamente, notamos um problema com preço, ano e odômetro. Por exemplo, o preço mínimo e máximo são $0,00 e $3.048.344.231,00, respectivamente. Você verá como lidamos com isso na próxima seção.

Ainda, para entender melhor as variáveis discretas:

```
df.condition.unique()
```

```
array(['good', nan, 'excellent', 'like new', 'fair', 'salvage', 'new'],
      dtype=object)
```

Você pode ver que existem muitos sinônimos, como 'excelente' e 'como novo'. Embora este não seja o melhor exemplo, há algumas circunstância em que é ideal agrupar palavras diferentes. Por exemplo, se você estivesse analisando padrões climáticos, talvez queira reclassificar 'nublado', 'cinza', 'nublado com possibilidade de chuva' e 'predominantemente nublado' simplesmente como 'nublado'.

Mais tarde, você verá que vamos omitir esta coluna devido ao fato de ter MUITOS valores nulos, mas se você quisesse reclassificar os valores de condição, poderia usar o código abaixo:

```
# Reclassify condition column
def clean_condition(row):

   good = ['good','fair']
   excellent = ['excellent','like new']

   if row.condition in good:
       return 'good'
   if row.condition in excellent:
       return 'excellent'
   return row.condition# Clean dataframe
def clean_df(playlist):
   df_cleaned = df.copy()
   df_cleaned['condition'] = df_cleaned.apply(lambda row:
clean_condition(row), axis=1)
   return df_cleaned# Get df with reclassfied 'condition' column
df_cleaned = clean_df(df)print(df_cleaned.condition.unique())
```

Leonardo Carvalho
leofacebook17@gmail.com

```
['good' nan 'excellent' 'salvage' 'new']
```

Outra coisas legais de entender nessa etapa:

- Cardinalidade do seu dado - quantos dados de cada categoria você encontra em uma variável categórica. Para isso, use `df.condition.value_counts()`
- Para variáveis numéricas, gosto de ver gráficos como histograma, além do describe que comentamos acima. Para isso, use `df['price'].hist(bins=50)`

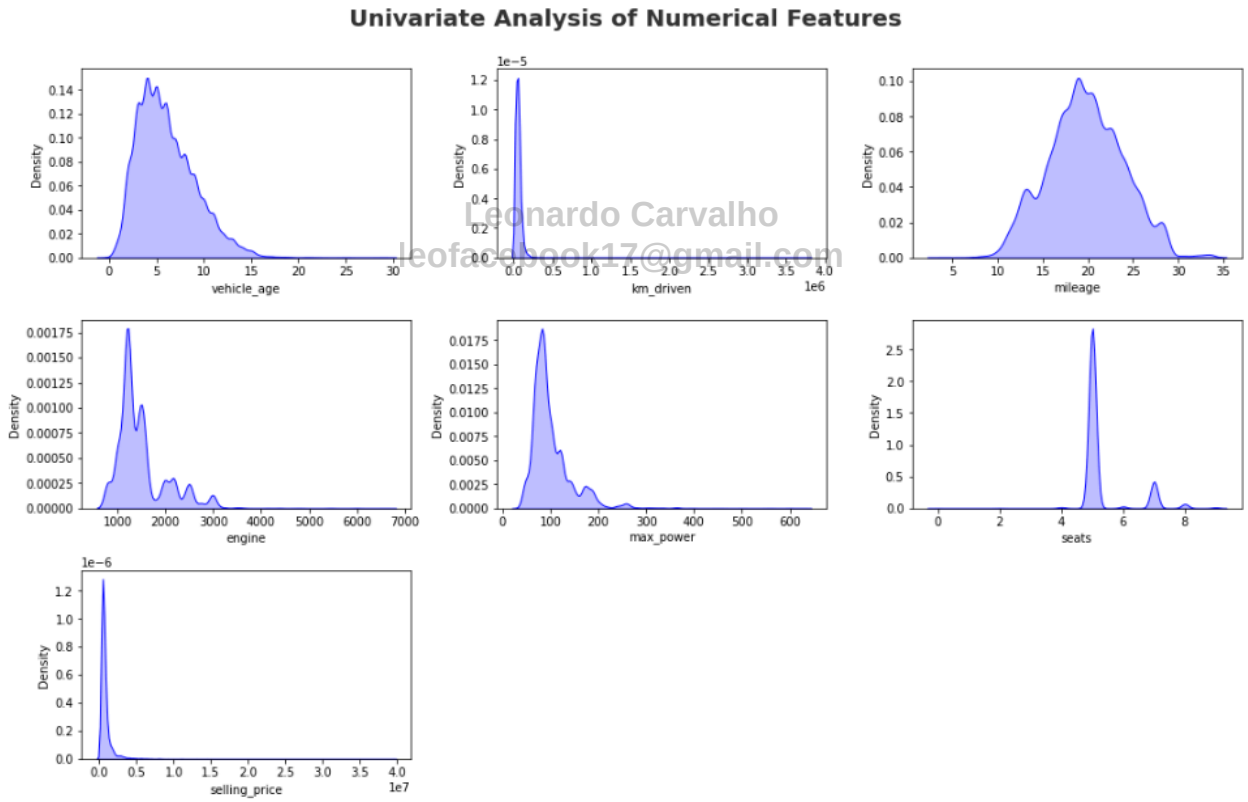
Você que é meu aluno já viu bastante isso, porém veja aqui alguns bons códigos que vão te ajudar nessa tarefa

KDE

Numerical Features

In [27]:

```
plt.figure(figsize=(15,15))
###The suptitle() method figure module of matplotlib library is used to Add a centered title to the figure.
plt.suptitle("Univariate Analysis of Numerical Features",fontsize=20,fontweight='bold',alpha=0.8,y=1.)

for i in range(len(numeric_features)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df[numeric_features[i]],shade=True,color='b')
    plt.xlabel(numeric_features[i])
    plt.tight_layout()
```



Leonardo Carvalho
leofacebook17@gmail.com

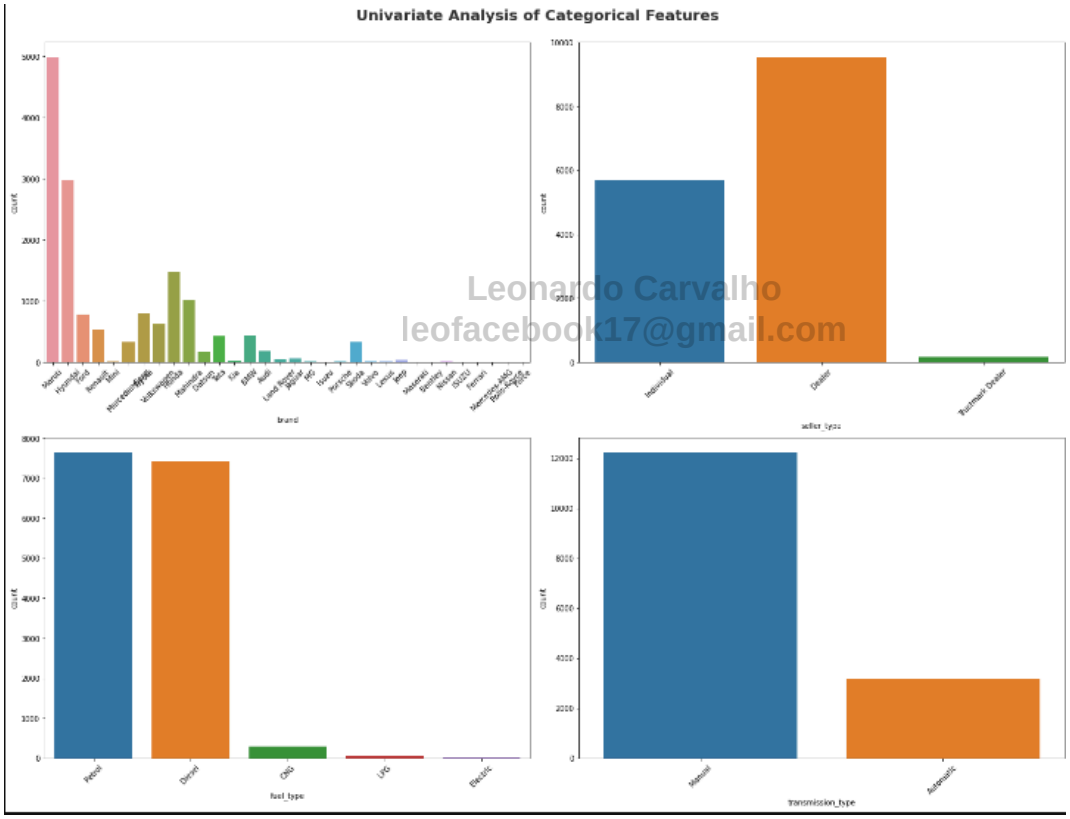
Kernel Density Estimation (KDE) informa sobre a distorção dos dados. No gráfico acima, podemos ver que Km_driven, max_power, sell_price e motor estão distorcidos para a direita e positivamente. Isso significa que há valores discrepantes em km_drive, motor, preço de venda e potência máxima.

Barplot

Categorical Features

In [34]:

```
import warnings
warnings.filterwarnings('ignore')

##categorical features
plt.figure(figsize=(20,15))
plt.suptitle("Univariate Analysis of Categorical Features",fontsize=20,fontweight='bold',alpha=0.8,y=1.)
cat1 = ['brand','seller_type','fuel_type','transmission_type']

for i in range(len(cat1)):
    plt.subplot(2,2,i+1)
    sns.countplot(x=df[cat1[i]])
    plt.xlabel(cat1[i])
    plt.xticks(rotation=45)
    plt.tight_layout()
```



Mostra a contagem de categorias em uma coluna.

2. Limpeza do seu conjunto de dados

Removendo variáveis redundantes

Primeiro, vamos nos livrar de variáveis redundantes. Isso inclui variáveis que não vão ser usadas para nada. Aqui, tome cuidado, pois a depender do seu

problema você pode precisar dessas variáveis. No nosso caso, vou remover url, image_url e city_url.

```
df_cleaned = df_cleaned.copy().drop(['image_url','city_url'], axis=1)
```

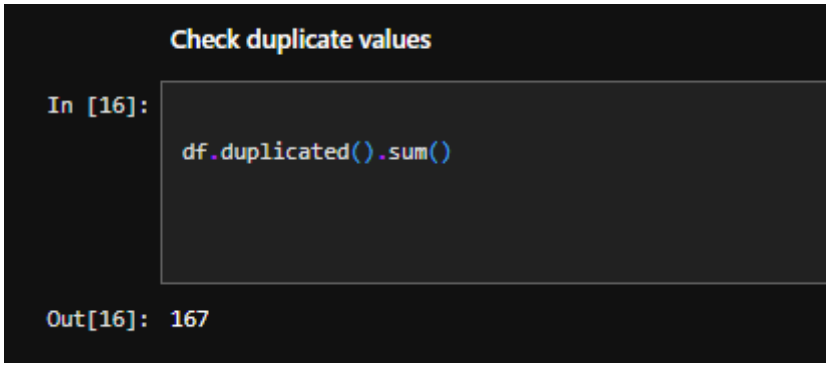
Apenas para exemplificar, um caso que eu não removeria uma variável de ID é o caso que eu precisasse iterar (com um loop for, por exemplo) sobre essas variáveis. Imagine que eu tenho uma série temporal que mostra batimentos cardíacos de vários pacientes (ID) ao longo do tempo. Quero, por exemplo, detectar anomalias. Não faz sentido nesse caso colocar todos os batimentos no mesmo balaio, já que cada pessoa tem uma frequência única.

No caso acima, muito provavelmente eu analisaria anomalias de cada paciente. Poderia misturar tudo para comparar entre pessoas? Sim, a depender do meu approach de análise.

Valores duplicados

Valores duplicados podem afetar a precisão dos modelos de ML e dos resultados da análise de dados, portanto, é uma boa prática sempre verificar se há valores duplicados no conjunto de dados. Em nosso conjunto de dados, existem 167 valores duplicados.

Leonardo Carvalho
leofacebook17@gmail.com



Isso indica que existem 167 linhas completamente idênticas. Nesse caso, use

```
df = df.drop_duplicates()
```

Seleção de Variáveis

Em seguida, precisamos falar sobre os nulos. No meu caso aqui vamos livrar de quaisquer colunas que tivessem **muitos** valores nulos. Isso nem sempre é a melhor prática e exige conhecimento do dataset para tal.

Porém, nesse caso, se a coluna tiver 40% ou mais de seus dados como valores nulos, vamos remove-la. Dependendo da situação, voce pode querer aumentar ou diminuir o limiar.

```
NA_val = df_cleaned.isna().sum() #conta quantos nulos tem nas colunas
def na_filter(na, threshold = .4):
   col_pass = []
   for i in na.keys():
       if na[i]/df_cleaned.shape[0]<threshold:
           col_pass.append(i)
   return col_passdf_cleaned = df_cleaned[na_filter(NA_val)]
```

Mas e aí, faz ou não faz sentido tirar os nulos? De novo, depende da sua situação. Nesse caso, como conheço pouco sobre o problema, optei por tirar. Mas já enfrentei casos em que eu ficaria quase sem dados se tirasse os nulos, por isso optei por tratá-los.

Leonardo Carvalho
leofacebook17@gmail.com

Vou falar sobre o tratamento de nulos mais adiante

Removendo Outliers que consideramos errados

Neste caso, usei minha intuição para determinar valores que são absurdos e estão errados

```
df_cleaned = df_cleaned[df_cleaned['price'].between(500, 200000)] #vamos
manter o que está entre $500 a $200.000
df_cleaned = df_cleaned[df_cleaned['year'] > 1950]  vamos manter o que
está foi produzido depois de 1950
df_cleaned = df_cleaned[df_cleaned['odometer'] < 899999.00]
df_cleaned.describe().apply(lambda s: s.apply(lambda x: format(x, 'f')))
```

Como outliers são polêmicos, quero dar um outro cenário aqui. Vamos analisar a coluna de preços de um outro dataset.

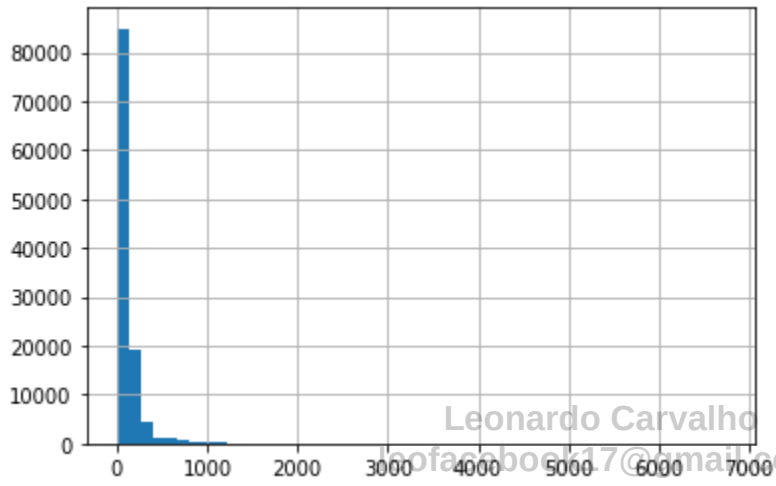
```
df['price']

0           58.90
1          239.90
2          199.00
3           12.99
4          199.90
            ...
112645     299.99
112646     350.00
112647      99.90
112648      55.99
112649      43.00
Name: price, Length: 112650, dtype: float64
```

Note que estamos falando dos preços de diversos produtos, em pedidos ao longo de vários anos. Analisando um histograma dos dados, e também pelas estatísticas do método describe,

```
print(df['price'].describe())
df['price'].hist(bins=50)
```

Leonardo Carvalho
leofacebook17@gmail.com

```
count    112650.000000
mean        120.653739
std         183.633928
min           0.850000
25%          39.900000
50%          74.990000
75%         134.900000
max        6735.000000
Name: price, dtype: float64
```



Leonardo Carvalho
leofacebook17@gmail.com

Podemos notar que a maior parte dos pedidos se concentra em faixas de preço abaixo de R$ 500, embora tenhamos algumas vendas com produtos de preço mais alto.

Pelo método describe, notamos que há um item com preço de R$ 6736,00. Entretanto, a média de preços dos produtos da base está em R$ 120, sendo 50% dos produtos comprados com preços até R$ 74.99.

A diferença entre a média e a mediana está no fato de que os produtos de maior preço criam uma distorção, puxando o preço médio para um valor mais alto. Se calcularmos o percentil 95 desses preços:

```
df['price'].quantile(.95)
```

o valor correspondente será R$ 349.9.

Tirar ou não tirar o Outlier? Eis a questão.
Já falamos sobre isso antes, mas veja aqui um resumo da ópera:

- Evite ao máximo remover outlier que não é um erro. Nesse caso, é plenamente possível ter um produto com esse valor
- Retire sem medo se você souber que é um erro (ex: uma coluna de alturas de mulheres, temos uma mulher de 3m de altura)
- Você pode removê-lo caso esteja atrapalhando sua análise - o que geralmente acontece com métricas sensíveis ao outlier, como média. Se sua intenção é entender como a maioria dos dados se comporta, você pode fazer sua análise com outlier versus sem outlier. Mas nunca se esqueça dele, ok?

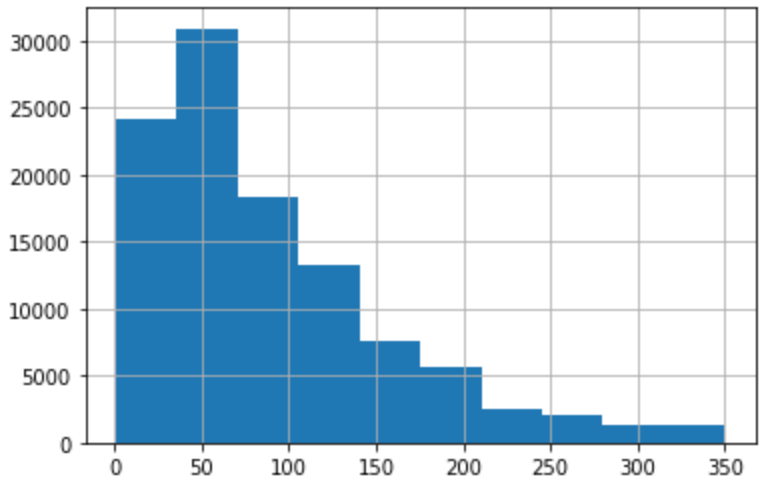
Se quiser remover o que está abaixo do percentil 95:

Leonardo Carvalho
leofacebook17@gmail.com

```
df.query(f'price <= {df.price.quantile(.95)}')['price'].describe()
```

```
count    107145.000000
mean         90.170653
std          68.709243
min           0.850000
25%          39.000000
50%          69.900000
75%         120.000000
max         349.900000
Name: price, dtype: float64
```

Observe que agora temos a mediana de R$ 70 e a média de R$ 90. A diferença entre as duas ficou menor, pois embora a distribuição seja de cauda à direita, agora está truncada num percentil que contempla 95% dos casos.



Isso quer dizer que os produtos de valor mais alto são irrelevantes? É claro que não. Cada venda desses produtos de preço alto corresponde a dezenas ou centenas de vendas de produtos no preço médio, e são muito importantes para a receita bruta dos vendedores. A questão é que não representam o preço médio mais frequente, e vendem num volume muito menor de unidades. O tratamento ou remoção de outliers deve ser usado e interpretado com cuidado, para responder as perguntas corretas e compreender as distorções que causam nas estatísticas do conjunto de dados.

Leonardo Carvalho
leofacebook17@gmail.com

Veja também esses artigos para complementar seu conhecimento:

Dicas de Data Cleaning

Como fazer uma limpeza de dados completa em Python

Removendo ou tratando linhas com valores nulos

Vamos conversar sobre tratamento nulos.

Você precisa decidir se vai excluir a linha que está nula ou se (.dropna(axis=0)) ou se vai preenchê-los (com média, mediana ou KNN imputer). Também seria importante investir se tem algum motivo desses nulos estarem acontecendo - por exemplo, alguma coluna que a resposta seja "não" está sendo representada como nula. Nesse caso, não encontrei um padrão entre os nulos

Como nosso objetivo aqui é apenas ANALISAR e, depois, vamos correlacionar variáveis (análise bivariada), a princípio vou remover nulos. Porém, se seu objetivo posterior é um modelo de ML, tome muito cuidado com essa decisão. Eu recomendo remover nulos apenas se:

- O nulo não significa nada mesmo! Se for algum tipo de resposta, não remova
- Para análise, se o passo anterior for verificado, você pode removê-lo pois ele pode acabar prejudicando sua análise.
- Para modelos de ML, vamos supor que sua coluna_A tenha nulos. Nesse caso, recomendo criar uma coluna dizendo "coluna_A_is_null" - essa coluna indica um booleano, em que será 0 se o valor da coluna_A não for nulo e 1 se for nulo. Nos nulos da coluna_A, você pode preenchê-los
- Alternativamente, se você não achar relevante levar para o modelo a informação de que a coluna_A continha nulos naquela célula, pode remover a linha - minha regrinha: 5% de nulos eu geralmente removo se o passo 1 for satisfeito e se eu realmente tiver certeza que não faz diferença levar a informação do nulo para o modelo

Leonardo Carvalho
leofacebook17@gmail.com

Nesse caso, como nosso objetivo é análise, removeremos

```
df_cleaned = df_cleaned.dropna(axis=0)

df_cleaned.shape
```

Após o código abaixo, passei de 371982 para 208765 linhas.

Veja esse post que fiz no Instagram. Aqui tem absolutamente tudo que você precisa saber sobre isso:
https://www.instagram.com/p/Cr5wSxRouXx/?utm_source=ig_web_copy_link&igshid=MzRlODBiNWFlZA%3D%3D

Remoção de Ruídos

Em análises de dados e aprendizado de máquina, o problema de ruído nos dados é uma questão crítica que pode levar a conclusões imprecisas e modelos preditivos ineficientes. Ruído nos dados refere-se a informações inconsistentes ou contraditórias. Por exemplo, em um conjunto de dados de

classificação de filmes, se um filme for categorizado de maneiras diferentes, isso pode confundir os algoritmos de aprendizado de máquina (caso você aplique posteriormente), resultando em classificações incorretas.

Além disso, o ruído pode indicar problemas mais amplos na coleta ou processamento de dados, como falhas na entrada de dados ou mudanças nas metodologias de classificação. Portanto, a identificação e correção de ruídos são essenciais para manter a integridade e confiabilidade das análises.

Suponha que temos um conjunto de dados contendo informações sobre notícias, incluindo um identificador único (Id), título (Title), conteúdo (Content), data de publicação (Date) e uma etiqueta indicando se a notícia é verdadeira (0) ou falsa (1). Nosso objetivo é identificar notícias rotuladas inconsistentemente, ou seja, como verdadeiras e falsas ao mesmo tempo.

	id	label	title	content	date
0	id-1000000	NaN	Germany and Russia: natural partners, so why n...	This [post] (https : //www.fort-russ.com/20...	2019-01-01
1	id-1000001	1.0	Netflix Pulls Muslim Comedy Show After Saudi C...	The in Hollywood continues to eat itself , as ...	2019-01-01
2	id-1000002	NaN	New Study Reports that Grass Really IS Greener...	While the adage that the grass is always green...	2019-01-01
3	id-1000003	0.0		NaN Daniel Mihailescu , AFP | The government build...	2019-01-01
4	id-1000004	0.0	Nasri completes West Ham move as transfer wind...	Former France midfielder Samir Nasri has compl...	2019-01-01

Leonardo Carvalho
leofacebook17@gmail.com

Para detectar essas inconsistências, excluímos as colunas 'id' e 'date', pois não influenciam o conteúdo da notícia. O código Python a seguir é usado para encontrar ruídos:

```
df['unique_labels'] =
df.groupby(['title','content'])['label'].transform('nunique')
noisy_data = df[df['unique_labels'] > 1].drop('unique_labels', axis=1)
```

Neste código, criamos uma nova coluna 'unique_labels', que conta o número de tags únicas para cada combinação de título e conteúdo de notícia. Se houver mais de uma etiqueta única para a mesma notícia, isso indica um ruído.

Segue abaixo um exemplo de linhas com ruídos:

4527	id-1004527	0.0	Donald Trump's Nationalist Moment	When President Donald Trump flew into Houston ...	2019-01-03
4563	id-1004563	1.0	Donald Trump's Nationalist Moment	When President Donald Trump flew into Houston ...	2019-01-03

Após identificar os dados ruidosos, o próximo passo é removê-los para limpar o conjunto de dados:

```
# Remover os ruídos identificados
df_cleaned = df_cleaned.drop(noisy_data.index)
df_cleaned.drop(columns = ['unique_labels'], inplace = True)
```

Aqui, removemos as linhas ruidosas identificadas do conjunto de dados original e, em seguida, descartamos a coluna 'unique_labels', pois ela já cumpriu seu propósito.

A remoção de ruídos é um passo crítico no processo de limpeza de dados. Ao eliminar dados inconsistentes ou errôneos, garantimos que as análises e modelos desenvolvidos sejam baseados em informações precisas e confiáveis, o que é fundamental para a obtenção de resultados válidos e confiáveis.

Veja um outro exemplo com o dataset de carros.

Show Categories in columns

Leonardo Carvalho
leofacebook17@gmail.com

In [20]:

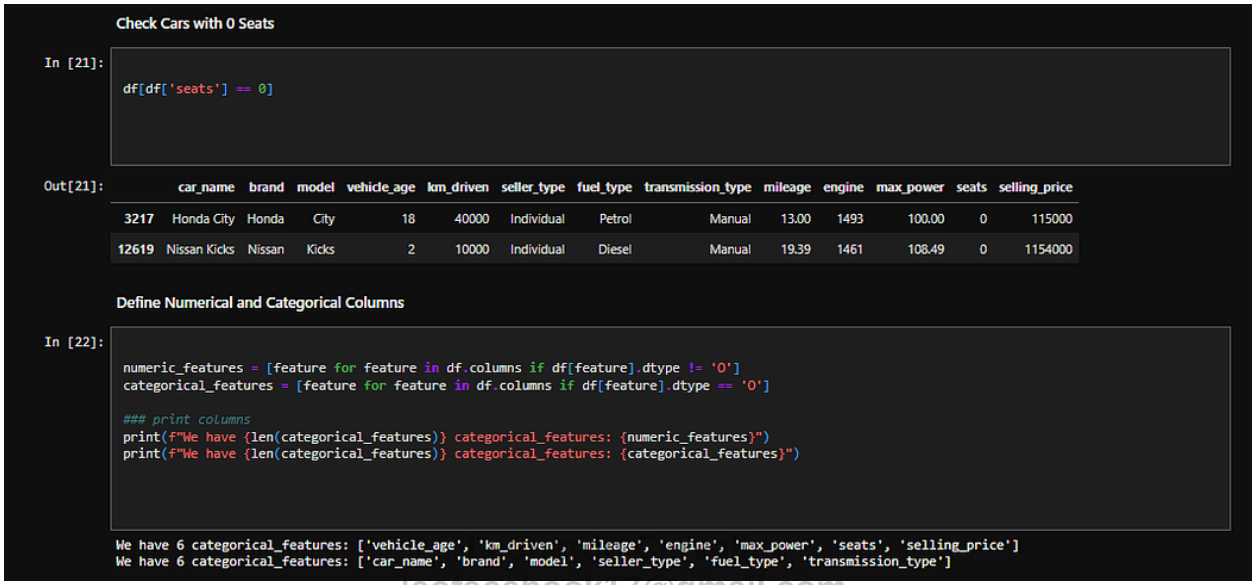
```
print("Categories in 'seller_type' variable: ",end = " ")
print(df['seller_type'].unique())
print("\n")

print("Categories in 'fuel_type' variable: ",end = " ")
print(df['fuel_type'].unique())
print("\n")

print("Categories in 'transmission_type' variable: ",end = " ")
print(df['transmission_type'].unique())
print("\n")

print("Categories in 'seats' variable: ",end = " ")
print(df['seats'].unique())
```

```
Categories in 'seller_type' variable:  ['Individual' 'Dealer' 'Trustmark Dealer']


Categories in 'fuel_type' variable:  ['Petrol' 'Diesel' 'CNG' 'LPG' 'Electric']


Categories in 'transmission_type' variable:  ['Manual' 'Automatic']


Categories in 'seats' variable:  [5 8 7 6 4 2 9 0]
```

Como podemos ver, existe um assento de categoria 0 que não é possível. Portanto, podemos tentar explorar a coluna do assento onde o assento é zero

e podemos então decidir remover essas linhas ou substituir os valores 0 por outros valores como modo, média, etc.

Além disso, podemos dividir as colunas em Numérico ou Categórico, o que irá ser útil para encontrar padrões nos dados.



Check Cars with 0 Seats

In [21]:
```
df[df['seats'] == 0]
```

Out[21]:

	car_name	brand	model	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats	selling_price
3217	Honda City	Honda	City	18	40000	Individual	Petrol	Manual	13.00	1493	100.00	0	115000
12619	Nissan Kicks	Nissan	Kicks	2	10000	Individual	Diesel	Manual	19.39	1461	108.49	0	1154000

Define Numerical and Categorical Columns

In [22]:
```
numeric_features = [feature for feature in df.columns if df[feature].dtype != 'O']
categorical_features = [feature for feature in df.columns if df[feature].dtype == 'O']

### print columns
print(f"We have {len(categorical_features)} categorical_features: {numeric_features}")
print(f"We have {len(categorical_features)} categorical_features: {categorical_features}")
```

```
We have 6 categorical_features: ['vehicle_age', 'km_driven', 'mileage', 'engine', 'max_power', 'seats', 'selling_price']
We have 6 categorical_features: ['car_name', 'brand', 'model', 'seller_type', 'fuel_type', 'transmission_type']
```

leofacebook17@gmail.com

3. Analisando relações entre variáveis mais a fundo (multivariada)

Variável x Variável

A primeira coisa que gosto de fazer ao analisar minhas variáveis é visualizá-las através de uma matriz de correlação, pois é a maneira mais rápida de desenvolver um entendimento geral de todas as minhas variáveis. Para revisar, correlação é uma medida que descreve a relação entre duas variáveis — se você quiser aprender mais sobre isso, pode conferir minha folha de dicas de estatísticas aqui.) Assim, uma matriz de correlação é uma tabela que mostra os coeficientes de correlação entre várias variáveis. Usei sns.heatmap() para traçar uma matriz de correlação de todas as variáveis no conjunto de dados de carros usados.

In [37]:

```
df[numeric_features].corr()
```

Out[37]:

	vehicle_age	km_driven	mileage	engine	max_power	seats	selling_price
vehicle_age	1.000000	0.333891	-0.257394	0.098965	0.005208	0.030791	-0.241851
km_driven	0.333891	1.000000	-0.105239	0.192885	0.044421	0.192830	-0.080030
mileage	-0.257394	-0.105239	1.000000	-0.632987	-0.533128	-0.440280	-0.305549
engine	0.098965	0.192885	-0.632987	1.000000	0.807368	0.551236	0.585844
max_power	0.005208	0.044421	-0.533128	0.807368	1.000000	0.172257	0.750236
seats	0.030791	0.192830	-0.440280	0.551236	0.172257	1.000000	0.115033
selling_price	-0.241851	-0.080030	-0.305549	0.585844	0.750236	0.115033	1.000000

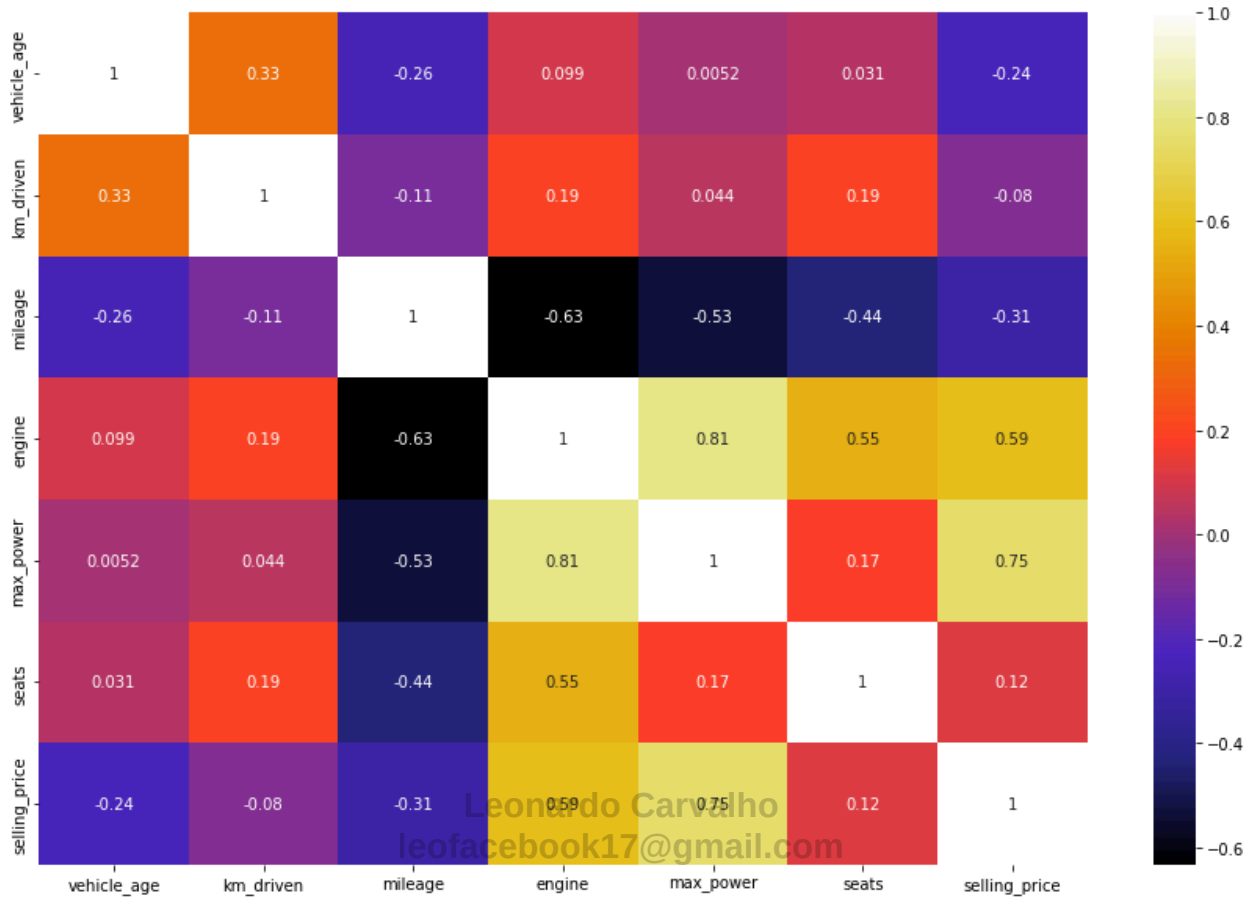
Correlation

Leonardo Carvalho
leofacebook17@gmail.com

In [38]:

```
plt.figure(figsize = (15,10))
sns.heatmap(df.corr(),cmap='CMRmap',annot=True)
plt.show()
```

Heatmap Code



Leonardo Carvalho
leofacebook17@gmail.com

Vejam que engine e mileage são bem correlacionadas negativamente, enquanto que max_power e selling_price são correlacionadas positivamente.

Lembrando que lá no EBA - Estatística do Básico ao Avançado, eu te ensino em detalhes sobre correlações, gráficos de dispersão e muito mais!

Saber a correlação é importante também porque, digamos, que temos 2 colunas independentes A e B. A e B estão 99% correlacionadas ou 95% correlacionadas, portanto, durante a seleção de recursos, podemos eliminar qualquer uma dessas colunas e não afetar o meu modelo de machine learning. A nível de análise, talvez nem faça sentido analisar as 2... Uma só, de repente, já bastaria.

Variável x Target

Vamos supor agora que estou mirando no selling_price. Quero fazer um modelo preditivo que, dado as características do carro, eu seja capaz de saber

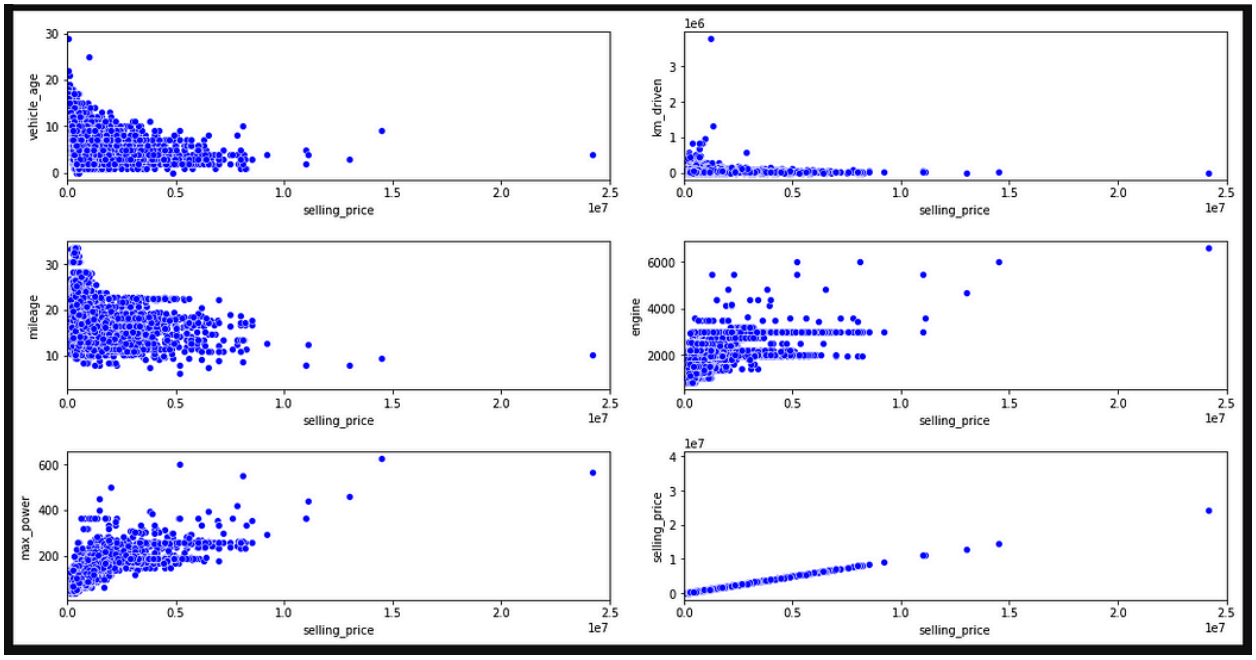
qual seria o valor adequado para vendê-lo. Uma boa ideia nesse caso é olhar a relação de cada característica do carro com essa variável.

Para as variáveis contínuas, podemos ver

```
In [54]:
continuous_features = []
for feature in numeric_features:
    if len(df[feature].unique()) >= 10:
        continuous_features.append(feature)

continuous_features

Out[54]: ['vehicle_age', 'km_driven', 'mileage', 'engine', 'max_power', 'selling_price']
```

```
In [64]:
fig = plt.figure(figsize=(15, 20))

for i in range(0, len(continuous_features)):
    ax = plt.subplot(8, 2, i+1)

    sns.scatterplot(data= df ,x='selling_price', y=continuous_features[i], color='b')
    plt.xlim(0,25000000) # Limit to 2.5 cr Rupees to view clean
    plt.tight_layout()
```

Leonardo Carvalho
leofacebook17@gmail.com

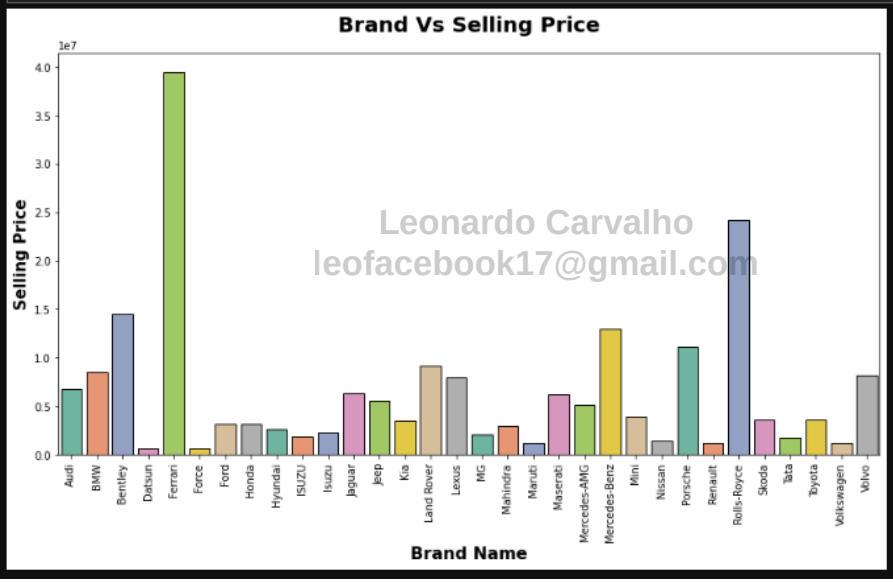


Aqui, no gráfico de dispersão acima, podemos ver as seguintes observações:

1. A idade mais baixa do veículo tem mais preço de venda do que o veículo com mais idade.
2. O motor CC tem um efeito positivo no preço.
3. Kms percorridos tem um efeito negativo no preço de venda.

Podemos ver também com variáveis categóricas

```
plt.subplots(figsize = (14,7))
sns.barplot(x=brand.index,y=brand.values,ec='black',palette='Set2')
plt.title("Brand Vs Selling Price", weight="bold", fontsize=20, pad=20)
plt.ylabel("Selling Price", weight='bold', fontsize=15)
plt.xlabel("Brand Name", weight="bold", fontsize=16)
plt.xticks(rotation=90)
plt.show()
```



Leonardo Carvalho
leofacebook17@gmail.com

Testes de hipótese aqui são extremamente bem-vindos. Com eles, conseguimos entender de fato a importância da variável e usá-las para fazer regras de negócios. Nesse caso em que nosso target é numérico contínuo, poderíamos fazer testes de hipótese com as variáveis categóricas. Ou seja, ver se realmente temos diferença de média de preço para as marcas de carro, a nível populacional, de acordo com a amostra que temos.

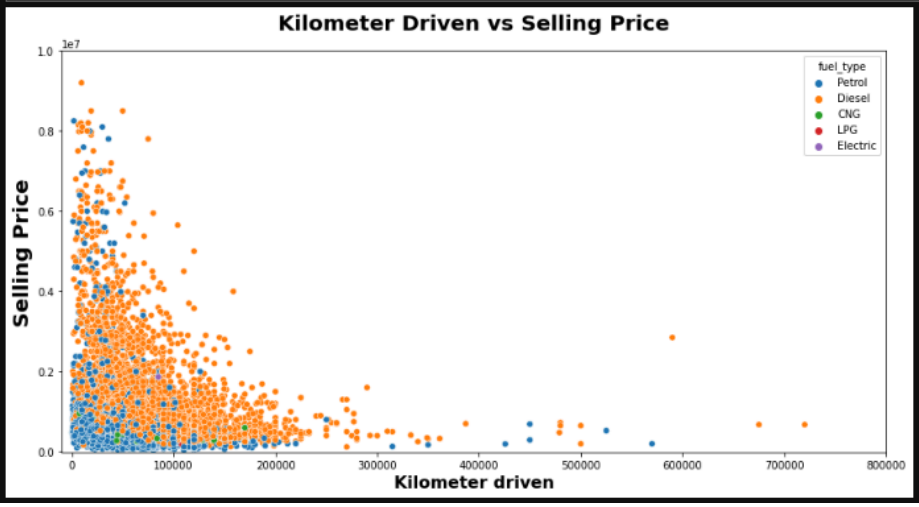
Variável x Variável x Target

Aqui a história fica mais complexa, mas podemos usar 2 variáveis e ver em relação ao target

Kilometer driven vs Selling Price

In [90]:

```
plt.subplots(figsize=(14,7))
sns.scatterplot(x="km_driven", y='selling_price', data=df,ec = "white",color='b', hue='fuel_type')
plt.title("Kilometer Driven vs Selling Price", weight="bold",fontsize=20, pad=20)
plt.ylabel("Selling Price", weight="bold", fontsize=20)
plt.xlim(-10000,800000) #used limit for better visualization
plt.ylim(-10000,10000000)
plt.xlabel("Kilometer driven", weight="bold", fontsize=16)
plt.show()
```



Leonardo Carvalho
leofacebook17@gmail.com

Muitos carros foram vendidos com km entre 0 a 20 mil quilômetros e carros com baixa quilometragem têm preços de venda mais altos em comparação com carros com mais km rodados.

Tenha muito cuidado ao adicionar mais variáveis nessa análise pois pode ser que sua EDA comece a ficar muito confusa. Considere também por aqui uma regressão linear para extrair o p-valor de suas features em relação ao target e ver os coeficientes da regressão para entender o "peso" de cada feature na predição.

Veja esse artigo para enriquecer mais seu conhecimento

https://medium.com/@ugursavci/complete-exploratory-data-analysis-using-python-9f685d67d1e4