Crystal Case Study

Setup

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import seaborn as sns
import datetime
import warnings


warnings.filterwarnings("ignore")


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

Import data

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
df
```

	datecrawled	name	seller	offertype	pr
0	2016-03-24T11:52:17	Golf_3_1.6	privat	Angebot	
1	2016-03-24T10:58:45	A5_Sportback_2.7_Tdi	privat	Angebot	18
2	2016-03-14T12:52:21	Jeep_Grand_Cherokee_"Overland"	privat	Angebot	9
3	2016-03-17T16:54:04	GOLF_4_1_4__3T?ER	privat	Angebot	1
4	2016-03-31T17:25:20	Skoda_Fabia_1.4_TDI_PD_Classic	privat	Angebot	3
...	...	...	...	...	
199995	2016-03-27T07:57:15	Fiat_Stilo_Active_viele_Extras_wenig_km_Rentne...	privat	Angebot	2
199996	2016-03-07T15:47:49	BMW_318i	privat	Angebot	
199998	2016-04-	Nissan_Qashqai_AHK	privat	Angebot	11

Data cleaning

Fix dataset column offset

```
# Shift DataFrame
index_to_shift = df[df['name'].str.contains('?privat', regex=False)].index

rows_not_shift = df.iloc[index_to_shift,:2]
rows_to_shift = df.iloc[index_to_shift,2:]
df.iloc[index_to_shift,:] = pd.concat([rows_not_shift, rows_to_shift], axis=1)
df.iloc[index_to_shift,:].head()
```

	datecrawled	name	seller	offertype	price	abtest	ve
1409	2016-03-24T14:54:19	Audi_Coup?privat	NaN	Angebot	2000	test	
4469	2016-03-	Mercedes_C_Coup?privat	NaN	Angebot	5250	control	

```
# Clean column name
df['name'] = list(map(lambda x: x.replace('?privat',''), df['name']))
```

Drop columns

	2016-03-	BMW e30 Karosse 2Tuerer Coup?						

```
df['seller'].value_counts()
```

```
privat        199827
gewerblich         3
Name: seller, dtype: int64
```

```
df['nrofpictures'].value_counts()
```

```
FALSE    200000
Name: nrofpictures, dtype: int64
```

```
df['offertype'].value_counts()
```

```
Angebot    199994
Gesuch          6
Name: offertype, dtype: int64
```

```
df = df.drop(columns = ['name', 'seller', 'nrofpictures', 'offertype'])
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

	datecrawled	price	abtest	vehicletype	yearofregistration	gearbox	powerps	mode
0	2016-03-24T11:52:17	480	test	NaN	1993	manual	0	go

Variable types

```
df.info()
```

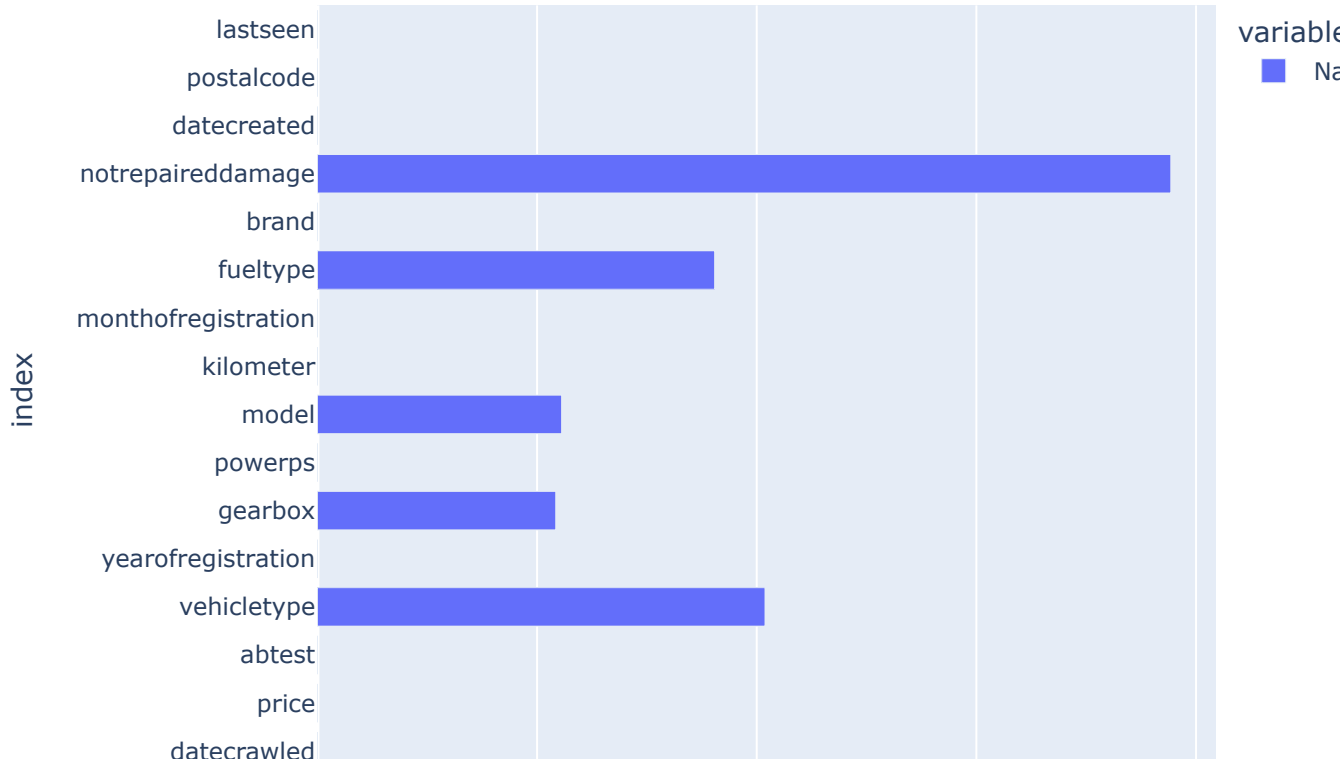
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 16 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   datecrawled          200000 non-null  object
 1   price                200000 non-null  object
 2   abtest               200000 non-null  object
 3   vehicletype          179622 non-null  object
 4   yearofregistration   200000 non-null  object
 5   gearbox              189151 non-null  object
 6   powerps              200000 non-null  object
 7   model                188883 non-null  object
 8   kilometer            200000 non-null  object
 9   monthofregistration  200000 non-null  object
 10  fueltype             181917 non-null  object
 11  brand                200000 non-null  object
 12  notrepaireddamage    161149 non-null  object
 13  datecreated          200000 non-null  object
 14  postalcode           200000 non-null  object
 15  lastseen             200000 non-null  object
dtypes: object(16)
memory usage: 24.4+ MB
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
nans = {}
for i in df.columns:
  nans[i] = df[i].isna().sum()

nans = pd.Series(nans, name='Nans Count')


fig = px.bar(nans, orientation='h')
fig.show()
```



Cleaning the Data

```
df = df.dropna(axis = 0, how = 'any')
```

```
df
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

	datecrawled	price	abtest	vehicletype	yearofregistration	gearbox	powerps
3	2016-03-17T16:54:04	1500	test	kleinwagen	2001	manual	75
4	2016-03-31T17:25:20	3600	test	kleinwagen	2008	manual	69
5	2016-04-04T17:36:23	650	test	limousine	1995	manual	102
6	2016-04-01T20:48:51	2200	test	cabrio	2004	manual	109
7	2016-03-21T18:54:38	0	test	limousine	1980	manual	50
...	...	...	...	...	...	...	...

Data Pre-processing

2016-03-

Powerps

199997 03T11:43:48 3900 control kleinwagen 2008 manual 80

```
df['powerps'] = df['powerps'].astype(int)
```

199998 02T01:54:47 1400 test suv 2012 manual 110

```
df = df[(25 <= df['powerps']) & (df['powerps'] <= 600)]
df['powerps'] = pd.qcut(df['powerps'],q = 4,labels=[0,1,2,3])
```

Kilometer

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
km_list = list(df.kilometer.unique())
km_list.sort()


# Replace kilometer w with their corresponding value from [0,1,2,3...]
df['kilometer'] = df['kilometer'].replace(km_list,list(range(len(km_list))))
```

Month of registration

```
df['monthofregistration'] = df['monthofregistration'].astype(int)
```

Price

```
df['price'] = df['price'].astype(float)


# Drop the rows where the price is not between $1,750 - $50,000
df = df[(1750 <= df['price']) & ( df['price'] <= 50000)]


df['price'].describe()
```

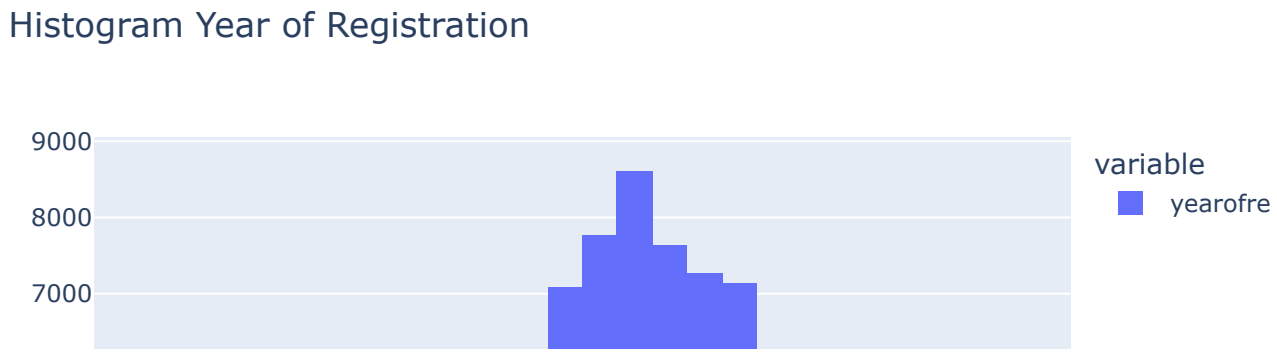
```
count    98672.000000
mean      8539.621909
std       7380.880808
min       1750.000000
25%       3390.000000
50%       5999.000000
75%      11000.000000
max      50000.000000
Name: price, dtype: float64
```

Year of registration

```
df['yearofregistration'] = df['yearofregistration'].astype(int)


# Drop the rows where the year of registration is below 1990
df = df[df['yearofregistration'] > 1990]


px.histogram(df['yearofregistration'],
             title = 'Histogram Year of Registration')
```

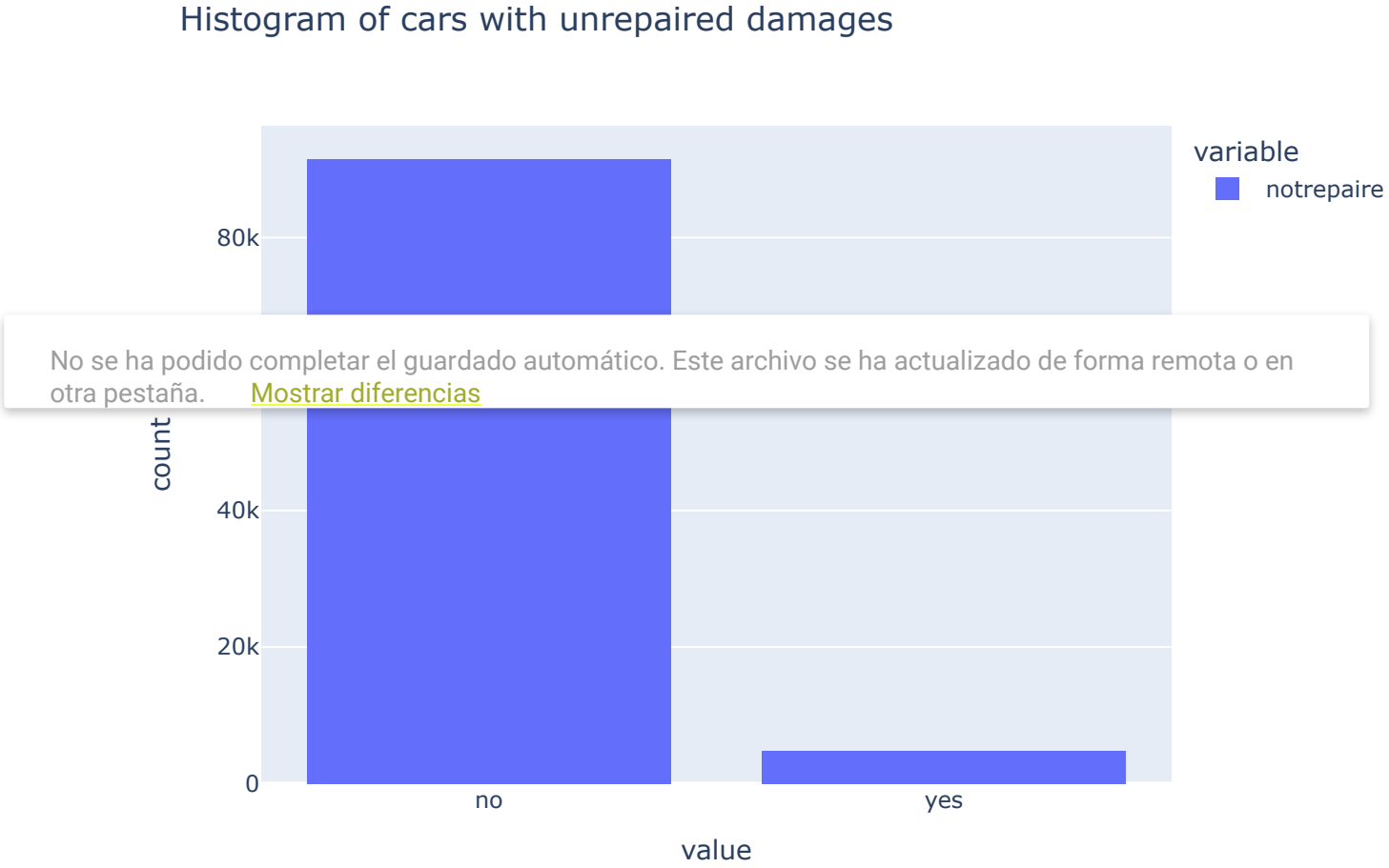


Not repaired damage

```
df['notrepaireddamage'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

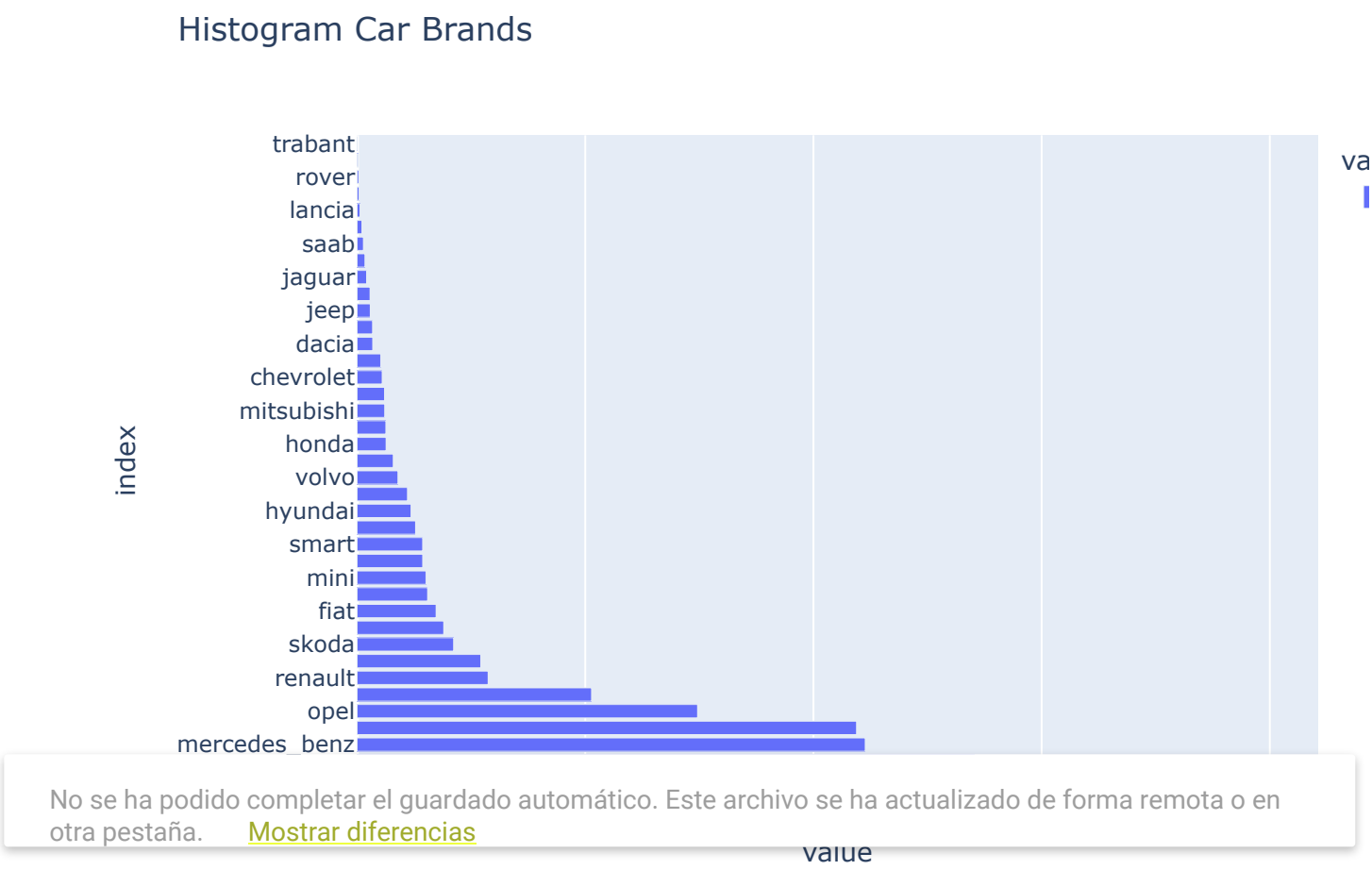
```
px.histogram(df['notrepaireddamage'],
             title = 'Histogram of cars with unrepaired damages')
```



```
dict_damage = {'yes': 1, 'no': 0}
df['notrepaireddamage'] = df['notrepaireddamage'].replace(dict_damage)
```

Brand

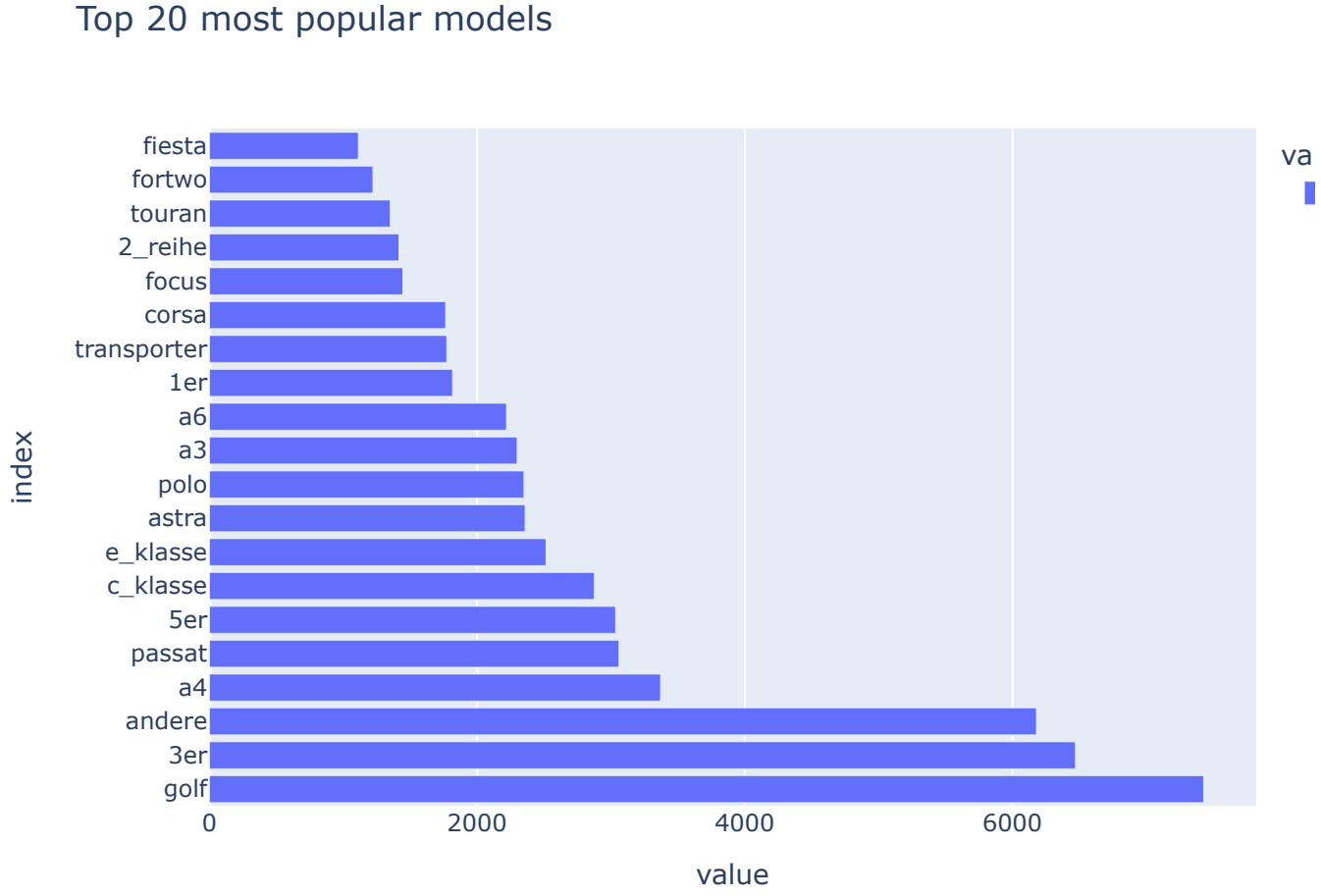
```
top_brands = df.brand.value_counts()
px.bar(top_brands, orientation='h', title = 'Histogram Car Brands')
```




```
# dict_brand = dict(enumerate(df['brand'].unique()))
# # Corrección
# dict_brand = {brand: num for num, brand in dict_brand.items()}
# df['brand'] = df['brand'].replace(dict_brand)
```

Model

```
top_models = df.model.value_counts().iloc[:20]
px.bar(top_models, orientation='h', title = 'Top 20 most popular models')
```

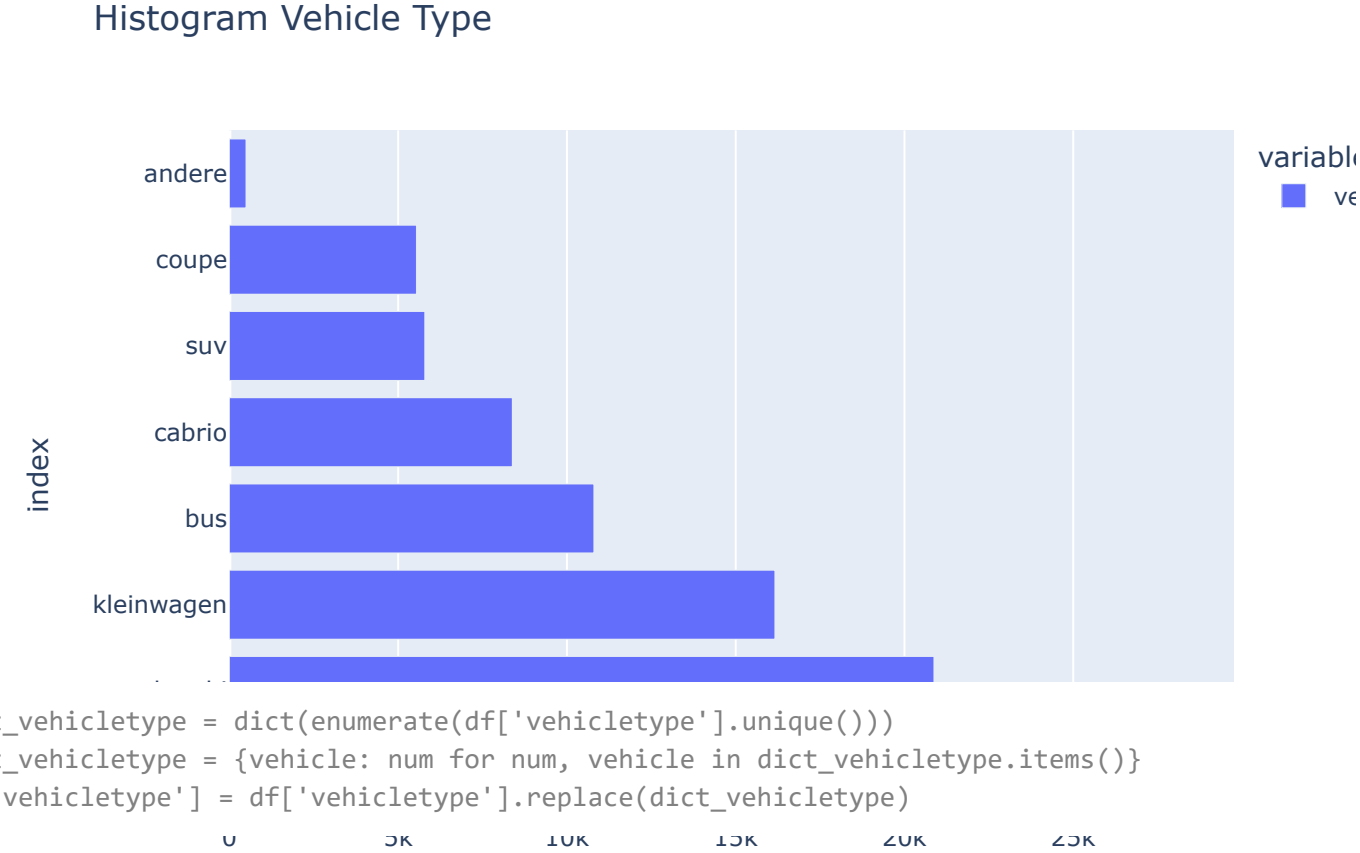



```
# dict_model = dict(enumerate(df['model'].unique()))
# # Corrección
# dict_model = {model: num for num, model in dict_model.items()}
# df['model'] = df['model'].replace(dict_model)
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

Vehicle type

```
top_type = df.vehicletype.value_counts()
px.bar(top_type, orientation='h', title = 'Histogram Vehicle Type')
```




```
dict_vehicletype = dict(enumerate(df['vehicletype'].unique()))
dict_vehicletype = {vehicle: num for num, vehicle in dict_vehicletype.items()}
df['vehicletype'] = df['vehicletype'].replace(dict_vehicletype)
```

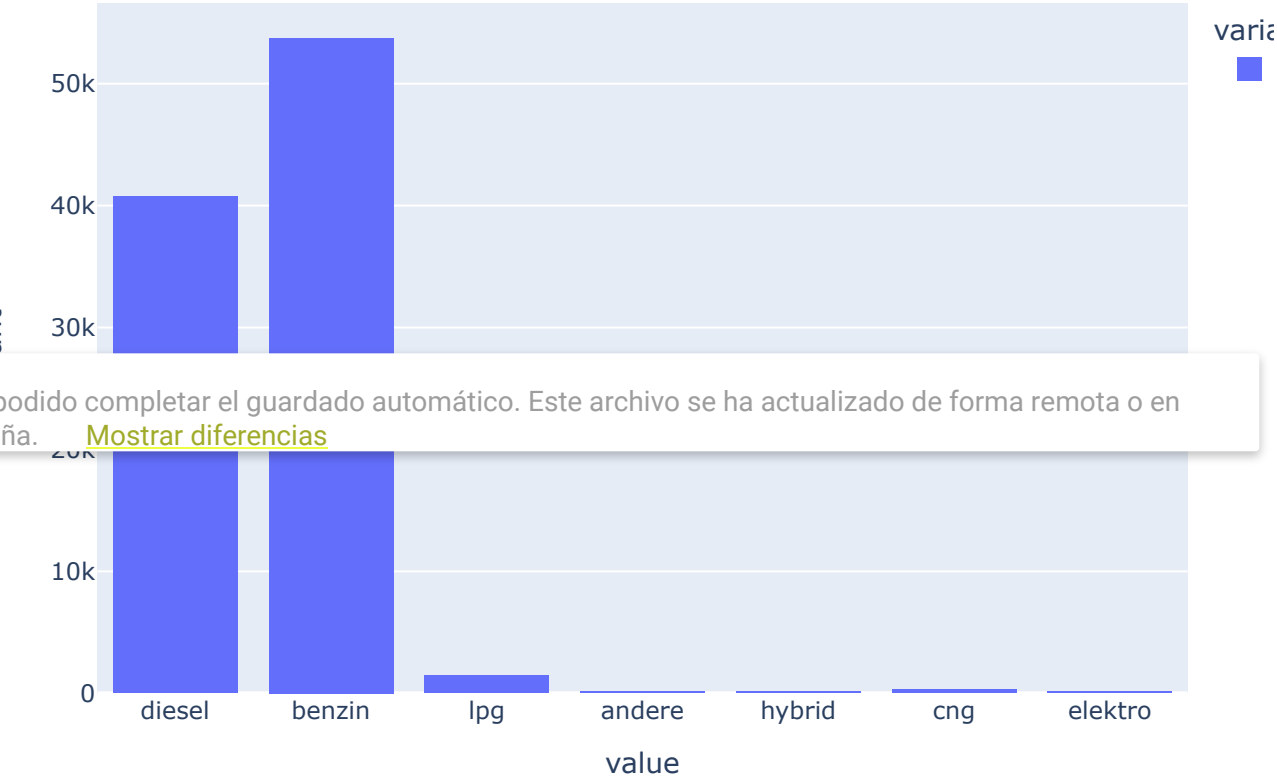
Gearbox

```
px.histogram(df['gearbox'], title = 'Histogram Gearbox')
```


```
dict_gearbox = dict(enumerate(df['gearbox'].unique()))
dict_gearbox = {gearbox: num for num, gearbox in dict_gearbox.items()}
df['gearbox'] = df['gearbox'].replace(dict_gearbox)
```

Fuel type

```
px.histogram(df['fueltype'], title = 'Histogram fuel type')
```




```
dict_fueltype = dict(enumerate(df['fueltype'].unique()))
dict_fueltype = {fueltype: num for num, fueltype in dict_fueltype.items()}
df['fueltype'] = df['fueltype'].replace(dict_fueltype)
```

Dates

```
df2 = df.copy()
```

```
df2.head(5)
```

	datecrawled	price	abtest	vehicletype	yearofregistration	gearbox	powerps	m
4	2016-03-31T17:25:20	3600.0	test	0	2008	0	0	
6	2016-04-01T20:48:51	2200.0	test	1	2004	0	1	2_
10	2016-03-26T19:54:18	2000.0	control	2	2004	0	1	3_
11	2016-04-07T10:06:22	2799.0	control	3	2005	0	2	pa
14	2016-03-21T12:57:01	17999.0	control	4	2011	0	3	na

```
# Convert to datetime format
df2.lastseen = pd.to_datetime(df2['lastseen'])
df2.datecreated = pd.to_datetime(df2['datecreated'])
df2.datecrawled = pd.to_datetime(df2['datecrawled'])
```



```
print((df2.datecreated - df2.datecrawled ).describe())
```

```
DATE CREATED - DATECRAWLED
count                          96306
mean     -1 days +05:38:55.843748054
std        2 days 07:06:13.795585487
min              -370 days +09:17:14
25%                -1 days +03:17:09
50%                -1 days +07:06:03
75%                -1 days +11:02:27
max                -1 days +23:34:44
dtype: object
```

There is no last seen dates before crawled dates (for obvious reasons). What happens with vehicles whose creation date is earlier than its crawled date and sold before the crawl started? The

crawl never registers its last seen which is interpreted as never being sold.

- Option 1: Those sold before the crawl have their datecreate deleted and then there would be no problem -> OK
- Option 2: We do not know if it was sold or not and the last seen date is the last date of the crawl and it is considered as not sold. -> bad

Solution: Throw away all the ones that have been created before the first crawled date

- Rows dropped: 2721

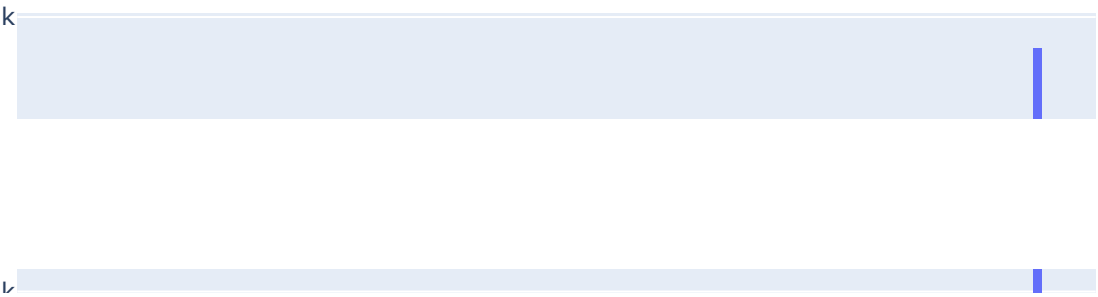
```
len(df2)
```

```
96306
```

```
# Make sure the datacreated after when the crawler started
# The trace will always be after the creation date, so each time this line is
# executed, more data will be deleted, it will be deleted day by day.
df2 = df2[df2['datecreated'] >= df2['datecrawled'].min()]
```

```
# Comparison
fig = go.Figure()

fig.add_trace(go.Histogram(x=df2['lastseen'].dt.date, name = 'LastSeen'))
fig.add_trace(go.Histogram(x=df2['datecrawled'].dt.date, name = 'Crawled'))
fig.add_trace(go.Histogram(x=df2['datecreated'].dt.date, name = 'Created'))
fig.update_layout(barmode='group')
```




```
len(df2)
```

```
93585
```

```
# Remove those that were crawled within the last 3 days, it is not right to
# consider them as unsold if the ad was created at the last moment
df2 = df2[df2['datecrawled'] < pd.to_datetime('2016-04-05')]
# df2 = df2[df2['datecreated'] < pd.to_datetime('2016-04-05')]
```

Already sold

```
# We can say that if the date of the last time the database was crawled is
# greater than a certain date of last seen that vehicle was sold?
# -> We will proceed with this line of thought
```

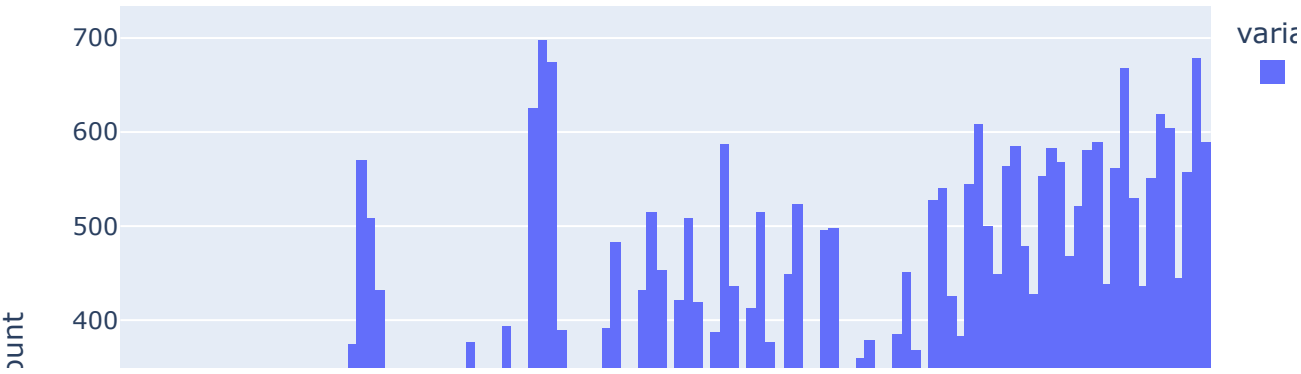
```
df2.datecrawled.max()
```

```
Timestamp('2016-04-04 23:57:46')
```

```
# The rows that have a last seen earlier than when the crawl stopped,
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
px.histogram(df2[df2['alreadysold'] == 1].lastseen, title = 'Histogram Car Sales')
```




```
df2['alreadysold'].value_counts()
```

```
0    49504
1    42434
Name: alreadysold, dtype: int64
```

Time until sale

```
# Subtract LASTSEEN - DATECREATED to find the time to sell
# The values with NaT is that they were not sold
df2['selltime'] = df2[df2.alreadysold == 1].lastseen - df2[df2.alreadysold == 1].datecreated
df2['selltime']
```

```
4                   NaT
6                   NaT
10                  NaT
199980              NaT
199991   17 days 07:18:18
199994   11 days 15:18:03
199997              NaT
199998              NaT
Name: selltime, Length: 91938, dtype: timedelta64[ns]
```

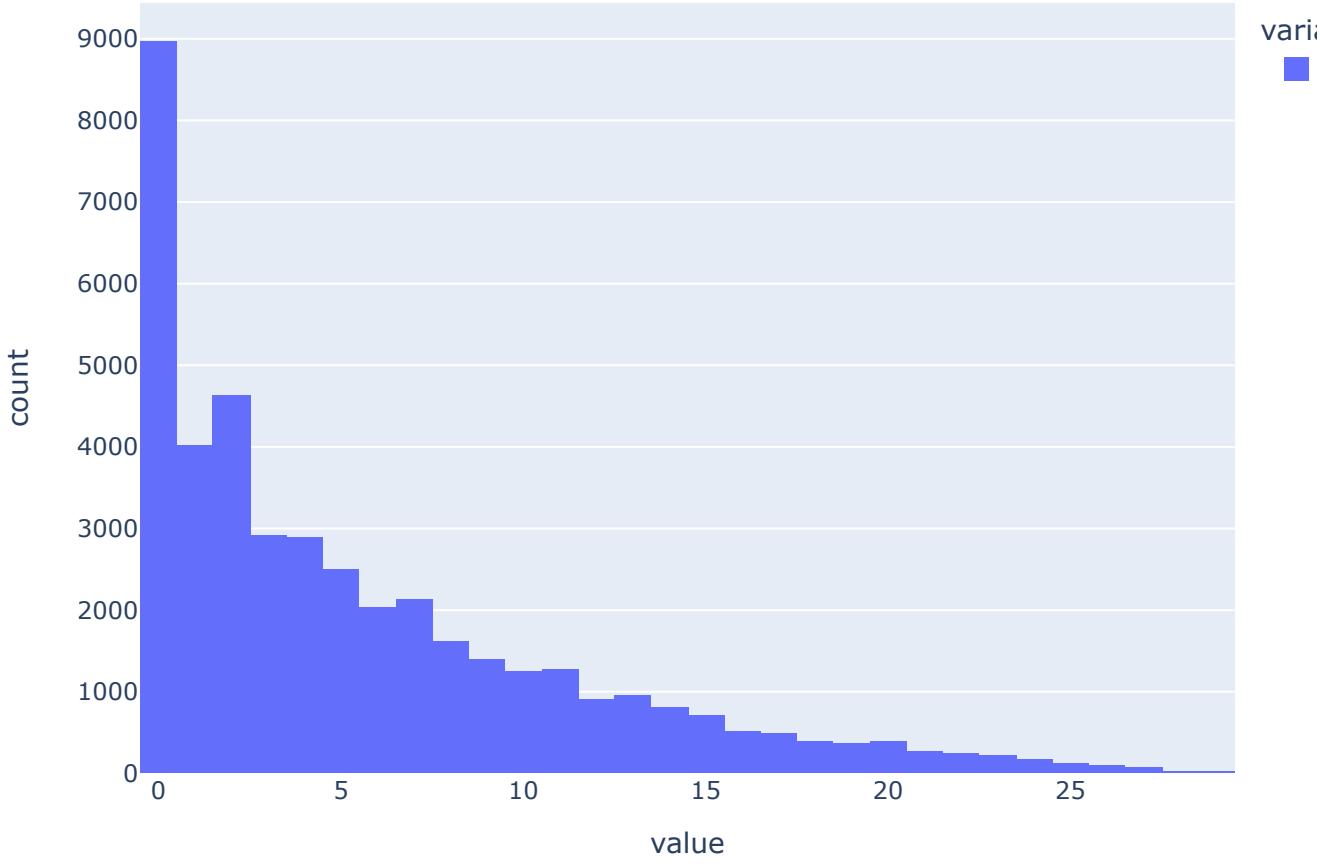
No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
df2.selltime.describe()
```

```
count                        42434
mean     6 days 03:26:38.778338125
std      5 days 20:59:22.424196546
min                0 days 00:36:25
25%         1 days 11:16:45.500000
50%                4 days 05:16:29
75%         9 days 01:45:32.500000
```

```
max              29 days 23:45:35
Name: selltime, dtype: object
```

```
# Of the cars sold, HISTOGRAM of their sale time in days
px.histogram(df2[df2['alreadysold'] == 1].selltime.apply(lambda x : x.days))
```



Implementación del modelo

```
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
```

Data Split

```
X = df2[['gearbox', 'price', 'brand', 'kilometer', 'fueltype', 'yearofregistration']]
y = df2['alreadysold']


dummies_gearbox = pd.get_dummies(X['gearbox'])
dummies_brand = pd.get_dummies(X['brand'])
dummies_fueltype = pd.get_dummies(X['fueltype'])
dummies_yearofregistration = pd.get_dummies(X['yearofregistration'])


X = pd.concat([X[['price', 'kilometer', 'yearofregistration']], dummies_gearbox], axis = 1)


X.head(5)
```

	price	kilometer	yearofregistration	0	1
4	3600.0	9	2008	1	0
6	2200.0	12	2004	1	0
10	2000.0	12	2004	1	0
14	17999.0	7	2011	1	0
17	18000.0	2	2007	0	1

```
y
```

```
4         0
6         0
10        0
14        0
17        0
         ..
199997    0
199998    0
Name: alreadysold, Length: 91938, dtype: int64
```



```
X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.3, random_state=42)
```

Decision Tree

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
from sklearn import tree
```

```
df2.vehicletype
```

```
4         0
6         1
10        2
14        4
17        2
          ..
199980    3
199991    7
199994    0
199997    0
199998    4
Name: vehicletype, Length: 91938, dtype: int64
```

```
# print(tree.export_text(dtc, feature_names = list(X.columns)))
```

```
preds = dtc.predict(X_test)
confusion_matrix(y_test, preds)
```

```
array([[9166, 5737],
       [6435, 6244]])
```

```
dtc.score(X_test, y_test)
```

```
0.5586977013994634
```



```
from sklearn.ensemble import RandomForestClassifier
```

```
X
```

	price	kilometer	yearofregistration	0	1
4	3600.0	9	2008	1	0
6	2200.0	12	2004	1	0
10	2000.0	12	2004	1	0
14	17999.0	7	2011	1	0
17	18000.0	2	2007	0	1
...	...	...	...	...	...
199980	12999.0	8	2011	1	0

```
rnd_for = RandomForestClassifier()


rnd_for.fit(X_train,y_train)
```

```
RandomForestClassifier()
```

```
preds = rnd_for.predict(X_test)


rnd_for.feature_importances_
```

```
array([0.77464131, 0.08848602, 0.13212355, 0.00244372, 0.00230539])
```

XGB Booster

```
import xgboost as xgb
```

No se ha podido completar el guardado automático. Este archivo se ha actualizado de forma remota o en otra pestaña. Mostrar diferencias

```
xgbc0 = xgb.XGBClassifier(objective='binary:logistic',
                          gamma='0.5',
                          learning_rate=0.01,
                          n_estimators=50,
                          booster='gbtree',
                          eval_metric='auc',
                          tree_method='hist',
                          grow_policy='lossguide',
                          use_label_encoder=False)
xgbc0.fit(X_train , y_train)
```

```
XGBClassifier(eval_metric='auc', gamma='0.5', grow_policy='lossguide',
              learning_rate=0.01, n_estimators=50, tree_method='hist',
              use_label_encoder=False)
```

```
default_params = {'gamma': [0,0.5, 1],
              'learning_rate': [0.01, 0.03, 0.06, 0.1, 0.25,  0.6],
              'n_estimators': [50,80,100,150]}

clf0 = GridSearchCV(estimator=xgbc0, scoring='accuracy', param_grid=default_params, return_tr
clf0.fit(X_train, y_train.values.ravel())

# results dataframe
df = pd.DataFrame(clf0.cv_results_)
```

```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
```

```
confusion_matrix(y_test, xgbc0.predict(X_test))
```

```
array([[10009,  4894],
       [ 6626,  6053]])
```

```
xgbc0.score(X_test, y_test)
```

```
0.5823363062867087
```