In [3]:

```
# data colelction and preprocessing
from bs4 import BeautifulSoup
import requests
import pandas as pd
import csv
# for data visualisation and statistical analysis
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set_style("white")
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
```

Data Preprocessing & Cleansing

So now we have the RAW data very well gathered, the next step is to preprocess these dataset in order ta make it useful for visualization and training session.

First of all we read our data set into a data frame, so that we can manipulate it easily ...

In [4]:

```
# set the column names
colnames=['price', 'year_model', 'mileage', 'fuel_type', 'mark', 'model', 'fiscal_power', '
# read the csv file as a dataframe
df = pd.read_csv("./data/output.csv", sep=",", names=colnames, header=None)
# let's get some simple vision on our dataset
df.head()
```

Out[4]:

	price	year_model	mileage	fuel_type	mark	model	fiscal_
0	135 000 DH	Année-Modèle:2013	Kilométrage:160 000 - 169 999	Type de carburant:Diesel	Marque:Peugeot	Modèle:508	Pui f
1	53 000 DH	Année-Modèle:2008	Kilométrage:35 000 - 39 999	Type de carburant:Diesel	Marque:Renault	Modèle:Clio	Pui f
2	59 000 DH	Année-Modèle:2007	Kilométrage:180 000 - 189 999	Type de carburant:Diesel	Marque:Citroen	Modèle:C3	Pui fiscal
3	88 000 DH	Année-Modèle:2010	Kilométrage:35 000 - 39 999	Type de carburant:Diesel	Marque:Mercedes-Benz	Modèle:220	Pui f
4	60 000 DH	Année-Modèle:2009	Kilométrage:130 000 - 139 999	Type de carburant:Essence	Marque:Ford	Modèle:Fiesta	Pui fiscal

Starting the Preprocessing

The first thing I have to do is to clean unwanted strings from my columns, then change it to the appropriate type since all of them are string values and finally drop unwanted columns such as sector, type, fiscal_power.

Price Columns

One thing I have noticed is that there are some ads that were published without the price, so the first thing to do is to delete those rows.

In [5]:

```
# remove thos rows doesn't contain the price value
df = df[df.price.str.contains("DH") == True]
# remove the 'DH' caracters from the price
df.price = df.price.map(lambda x: x.rstrip('DH'))
# remove the space on it
df.price = df.price.str.replace(" ","")
# change it to integer value
df.price = pd.to_numeric(df.price, errors = 'coerce', downcast= 'integer')
```

Year Model

In [6]:

```
# remove thos rows doesn't contain the year_model value
df = df[df.year_model.str.contains("Année-Modèle") == True]
# remove the 'Année-Modèle:' from the year_model
df.year_model = df.year_model.map(lambda x: x.lstrip('Année-Modèle:').rstrip('ou plus ancie
# df.year_model = df.year_model.map(lambda x: x.lstrip('Plus de '))
# remove those lines having the year_model not set
df = df[df.year_model != ' -']
df = df[df.year_model != '']
# change it to integer value
df.year_model = pd.to_numeric(df.year_model, errors = 'coerce', downcast = 'integer')
```

mileage

One of the tips I will `mileage` feature engineering is to calculate the mean of it, since I have 2 values the min and the max, so I decided to take the mean so that it could be more explicative in that sence, instead of choosing the minimum or the maximum.

In [7]:

```
# remove thos rows doesn't contain the year_model value
df = df[df.mileage.str.contains("Kilométrage") == True]
# remove the 'Kilométrage:' string from the mileage feature
df.mileage = df.mileage.map(lambda x: x.lstrip('Kilométrage:'))
df.mileage = df.mileage.map(lambda x: x.lstrip('Plus de '))
# remove those lines having the mileage values null or '-'
df = df[df.mileage != '-']
# we have only one value type that is equal to 500 000, all the other ones contain two valu
if any(df.mileage != '500 000'):
    # create two columns minim and maxim to calculate the mileage mean
    df['minim'], df['maxim'] = df.mileage.str.split('-', 1).str
    # remove spaces from the maxim & minim values
    df['maxim'] = df.maxim.str.replace(" ","")
    df['minim'] = df.minim.str.replace(" ","")
    df['maxim'] = df['maxim'].replace(np.nan, 500000)
    # calculate the mean of mileage
    df.mileage = df.apply(lambda row: (int(row.minim) + int(row.maxim)) / 2, axis=1)
    # now that the mileage is calculated so we do not need the minim and maxim values anymo
    df = df.drop(columns=['minim', 'maxim'])
```

Fuel type

In [8]:

```
# remove the 'Type de carburant:' string from the carburant_type feature
df.fuel_type = df.fuel_type.map(lambda x: x.lstrip('Type de carburant:'))
```

Mark & Model

In [9]:

```
# remove the 'Marque:' string from the mark feature
df['mark'] = df['mark'].map(lambda x: x.replace('Marque:', ''))
df = df[df.mark != '-']
# remove the 'Modèle:' string from model feature
df['model'] = df['model'].map(lambda x: x.replace('Modèle:', ''))
```

fiscal power

For the fiscal power we can see that there is exactly 5728 rows not announced, so we will fill them by the mean of the other columns, since it is an important feature in cars price prediction so we can not drop it.

In [10]:

```
df.fiscal_power.value_counts()
```

Out[10]:

```
Puissance fiscale:-                5624
Puissance fiscale:6 CV             1011
Puissance fiscale:8 CV              692
Puissance fiscale:7 CV              495
Puissance fiscale:9 CV              207
Puissance fiscale:11 CV             115
Puissance fiscale:10 CV             109
Puissance fiscale:5 CV              107
Puissance fiscale:12 CV             103
Puissance fiscale:4 CV               22
Puissance fiscale:17 CV              14
Puissance fiscale:13 CV              11
Puissance fiscale:21 CV               8
Puissance fiscale:-                   6
Puissance fiscale:20 CV               5
Puissance fiscale:23 CV               4
Puissance fiscale:14 CV               3
Puissance fiscale:Plus de 48 CV       3
Puissance fiscale:40 CV               2
Puissance fiscale:26 CV               2
Puissance fiscale:15 CV               2
Puissance fiscale:29 CV               2
Puissance fiscale:18 CV               2
Puissance fiscale:31 CV               1
Puissance fiscale:16 CV               1
Puissance fiscale:34 CV               1
Puissance fiscale:19 CV               1
Puissance fiscale:39 CV               1
Puissance fiscale:24 CV               1
Puissance fiscale:28 CV               1
Name: fiscal_power, dtype: int64
```

In [11]:

```
# remove the 'Puissance fiscale:' from the fiscal_power feature
df.fiscal_power = df.fiscal_power.map(lambda x: x.lstrip('Puissance fiscale:Plus de').rstri
# replace the - with NaN values and convert them to integer values
df.fiscal_power = df.fiscal_power.str.replace("-","0")
# convert all fiscal_power values to numerical ones
df.fiscal_power = pd.to_numeric(df.fiscal_power, errors = 'coerce', downcast= 'integer')
# now we need to fill those 0 values with the mean of all fiscal_power columns
df.fiscal_power = df.fiscal_power.map( lambda x : df.fiscal_power.mean() if x == 0 else x )
```

fuel type

In [12]:

```
# remove those lines having the fuel_type not set
df = df[df.fuel_type != '-']
```

drop unwanted columns

the sector, type and city features are not needed to build this model so we will delete them defintely, since they are not very representative in this case, for the model categorical feature we will drop it because of the hugeamount of levels.

In [13]:

```
df = df.drop(columns=['sector', 'type'])
```

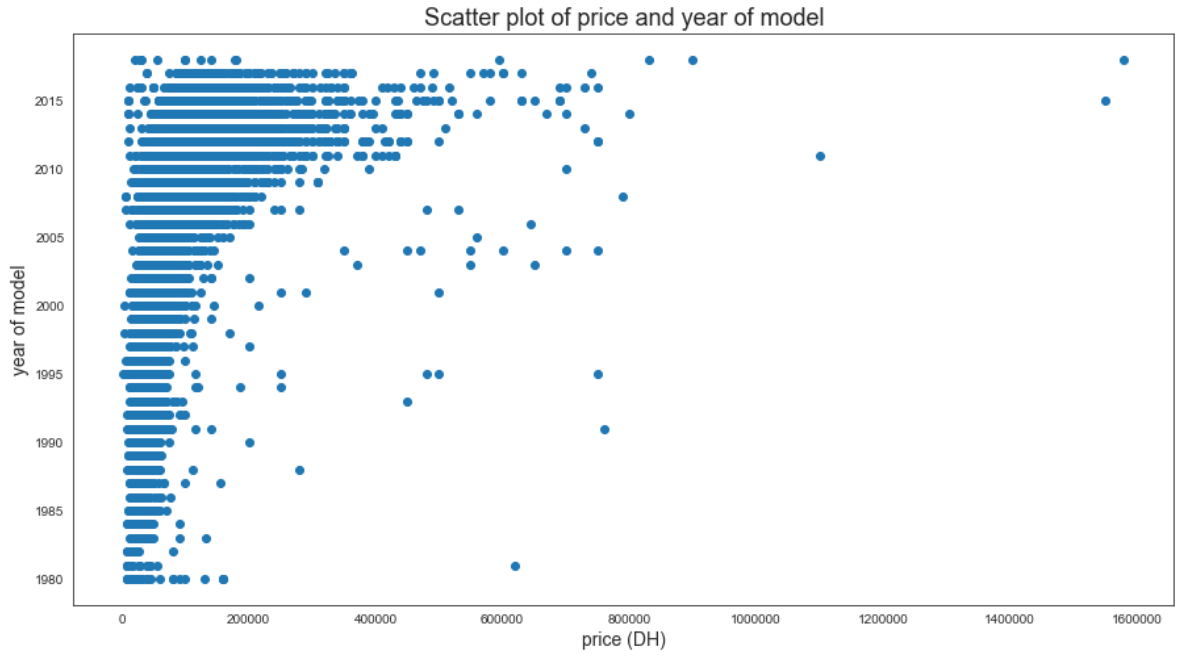
Exploratory data analysis & Visualisation

price distribution by year_model

Let's visualize the distribution of cars price by their year model release, and look how it behaves

In [14]:

```
# here we set the figure size to 15x8
plt.figure(figsize=(15, 8))
# plot two values price per year_model
plt.scatter(df.price, df.year_model)
plt.xlabel("price (DH)", fontsize=14)
plt.ylabel("year of model", fontsize=14)
plt.title("Scatter plot of price and year of model",fontsize=18)
plt.show()
```



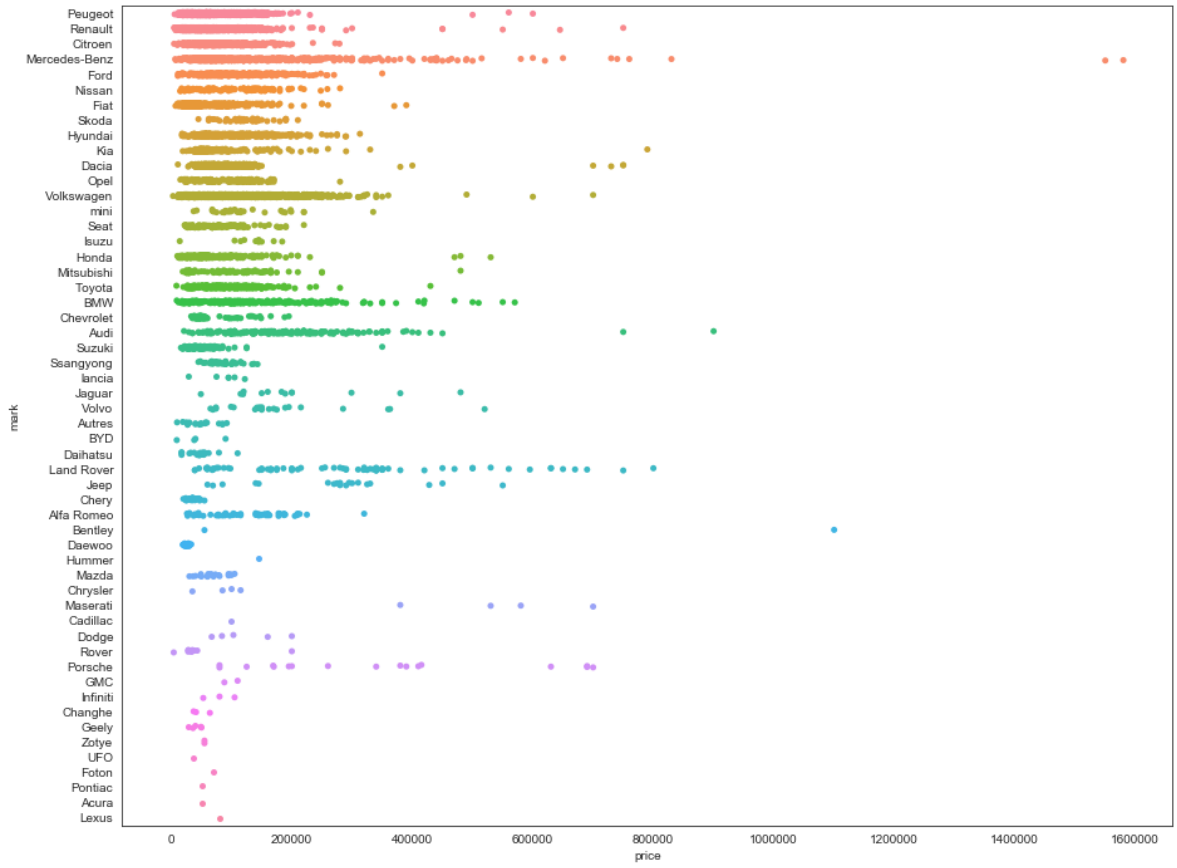
As we can see from the plot above, the cars price increase respectivly by years, and more explicitly we can say that the more the car is recently released, the price augment, while in the other side the oldest cars still have a low price, and this is totally logical since whenever the cars become kind of old from the date of release, so their price start decrease.

Price distribution by mark

Since we're looking to express the cars price by different features, so one of the important plot is to visualize how these prices differs between cars marks.

In [15]:

```
f, ax = plt.subplots(figsize=(15, 12))
sns.stripplot(data = df, x='price', y='mark', jitter=.1)
plt.show()
```



Interpretation

From the plot above, we can extract some the following insights :

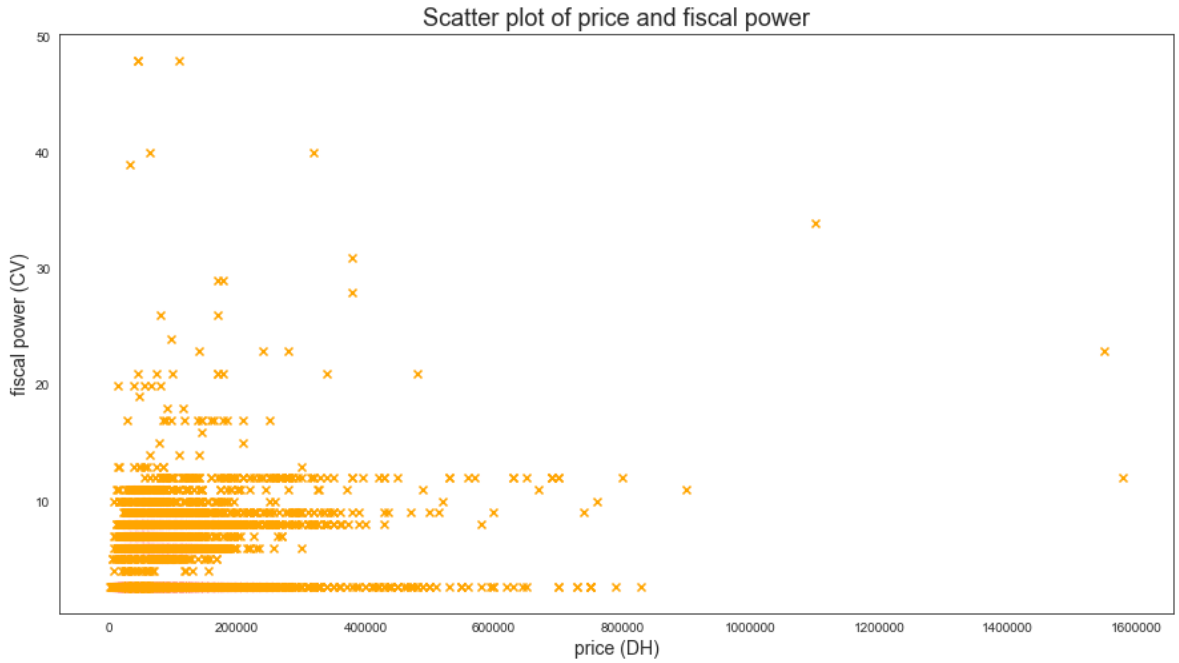
- The popular marks such as Renault, Peugeot, Citroen, Dacia, Hyundai, KIA had a stable range of price, in other words they are not well diversified on the price axis.
- In the opposit side, we can clearly notice that the sophisticated cars are well distributed over the price axis such as the Mercedes-Benz, Land-Rover, Audi, Maserati, Porsche ..., which means that the more the cars from those classes, the more their price augments

price distribution by fiscal_power

Let's visualize the distributions of cars price by their fiscal power and grouping by the fuel_type, and look how it behaves

In [16]:

```
# here we set the figure size to 15x8
plt.figure(figsize=(15, 8))
# plot two values price per year_model
plt.scatter(df.price, df.fiscal_power, c='orange', marker='x')
plt.xlabel("price (DH)", fontsize=14)
plt.ylabel("fiscal power (CV)", fontsize=14)
plt.title("Scatter plot of price and fiscal power",fontsize=18)
plt.show()
```



From the plot above we can notice clearly that there is a huge concentration of points in the range of [2800 DH,800000 DH], and [3 CV, 13 CV], which could be interpreted as first the huge domination of medium fiscal power cars in the market with correct price and second the more the fiscal power increase the price do too.

Top 20 Mark Distribution

For the mark feature I have 54 marks, so plotting it all is not a good option for visual purpose, I will plot only the top 20 mark

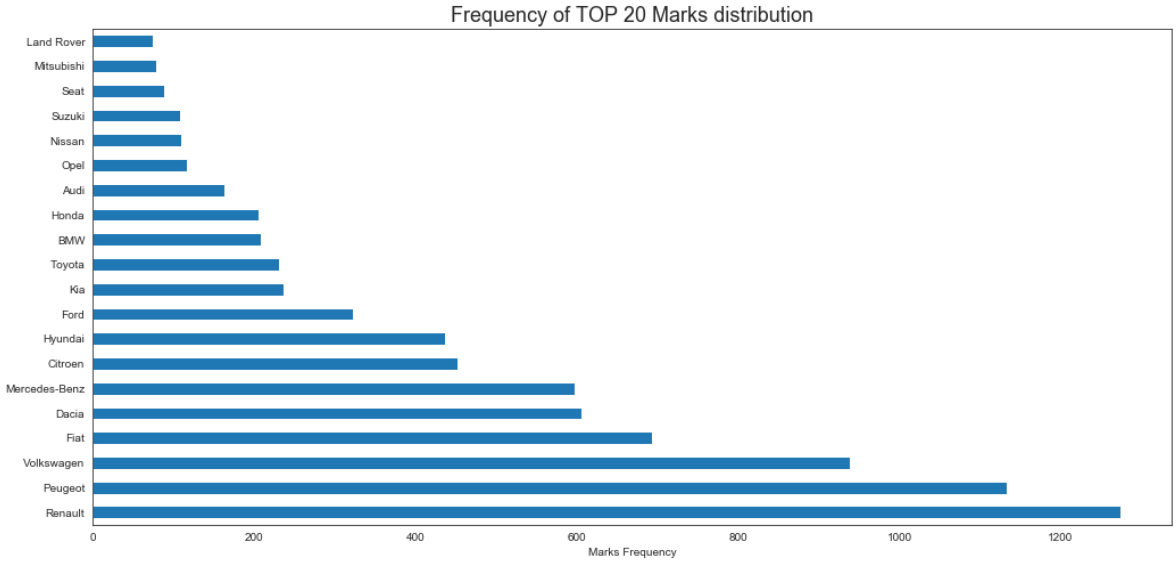
In [17]:

```
print('The length of unique marks feature is',len(df.mark.unique()))
```

```
The length of unique marks feature is 54
```

In [18]:

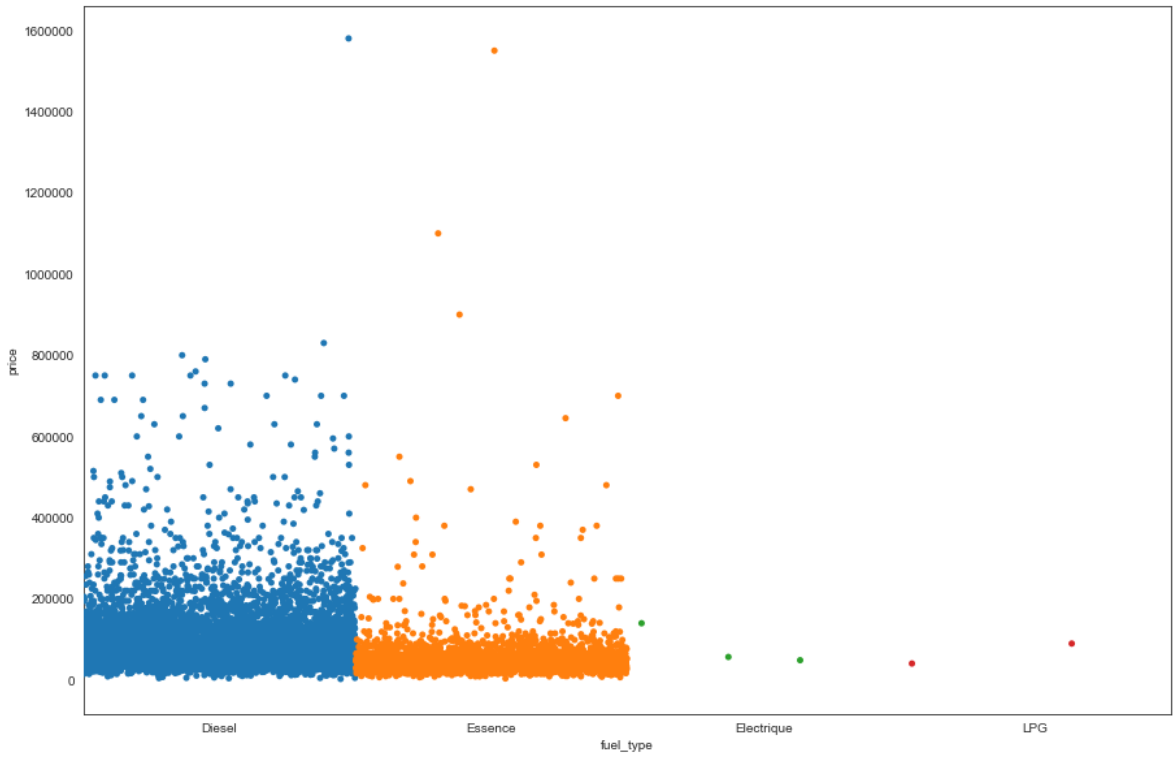
```
plt.figure(figsize=(17,8))
df.mark.value_counts().nlargest(20).plot(kind='barh')
plt.xlabel('Marks Frequency')
plt.title("Frequency of TOP 20 Marks distribution",fontsize=18)
plt.show()
```



Price Distribution by fuel type

In [19]:

```
f, ax = plt.subplots(figsize=(15, 10))
sns.stripplot(data = df, x='fuel_type', y='price', jitter=.5)
plt.show()
```

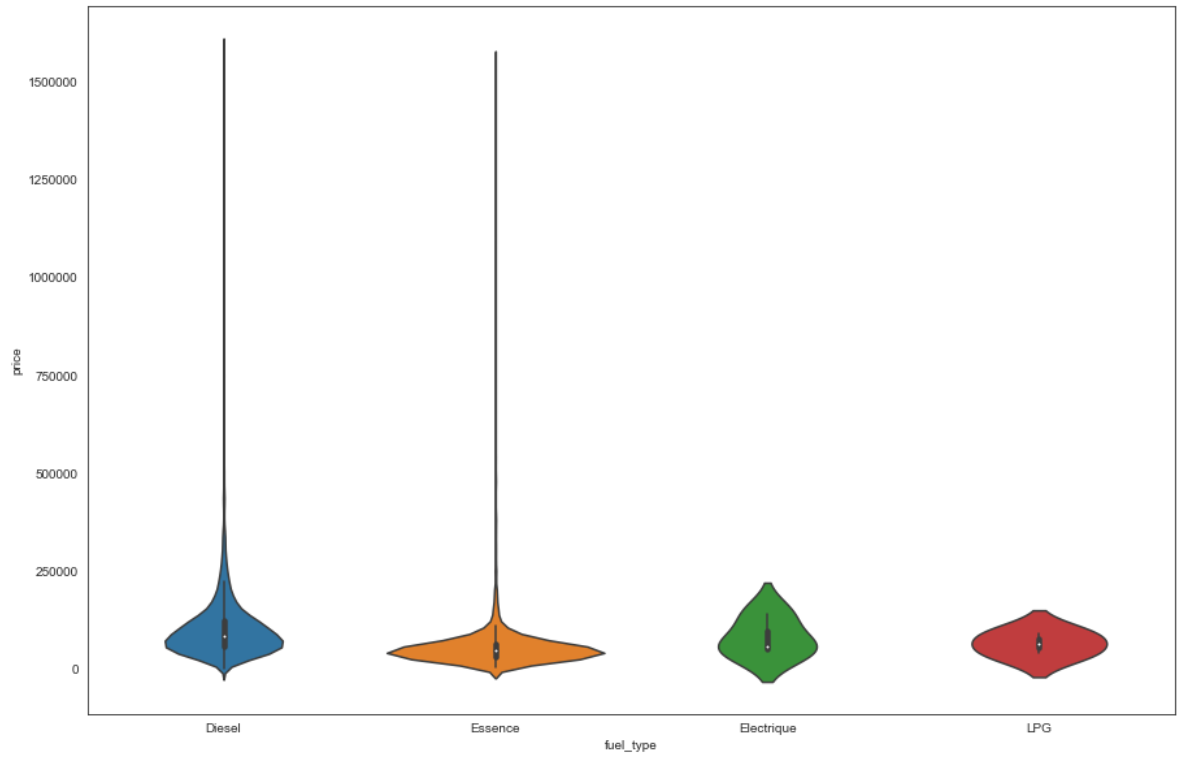


Some Insights with Violin plot

This chart is a combination of a Box Plot and a Density Plot that is rotated and placed on each side, to show the distribution shape of the data.

In [20]:

```
f, ax = plt.subplots(figsize=(15, 10))
sns.violinplot(data = df, x='fuel_type', y='price')
plt.show()
```



From the plot above, we can clearly visualise a lot of information such as the minimum, maximum price for 'Diesel' cars and also get perception on the Median values, but more particularly what we got in violin plot other than teh box plot, is the density plot width known as Kernel Density Plots.

Price distribution by mileage and fuel type

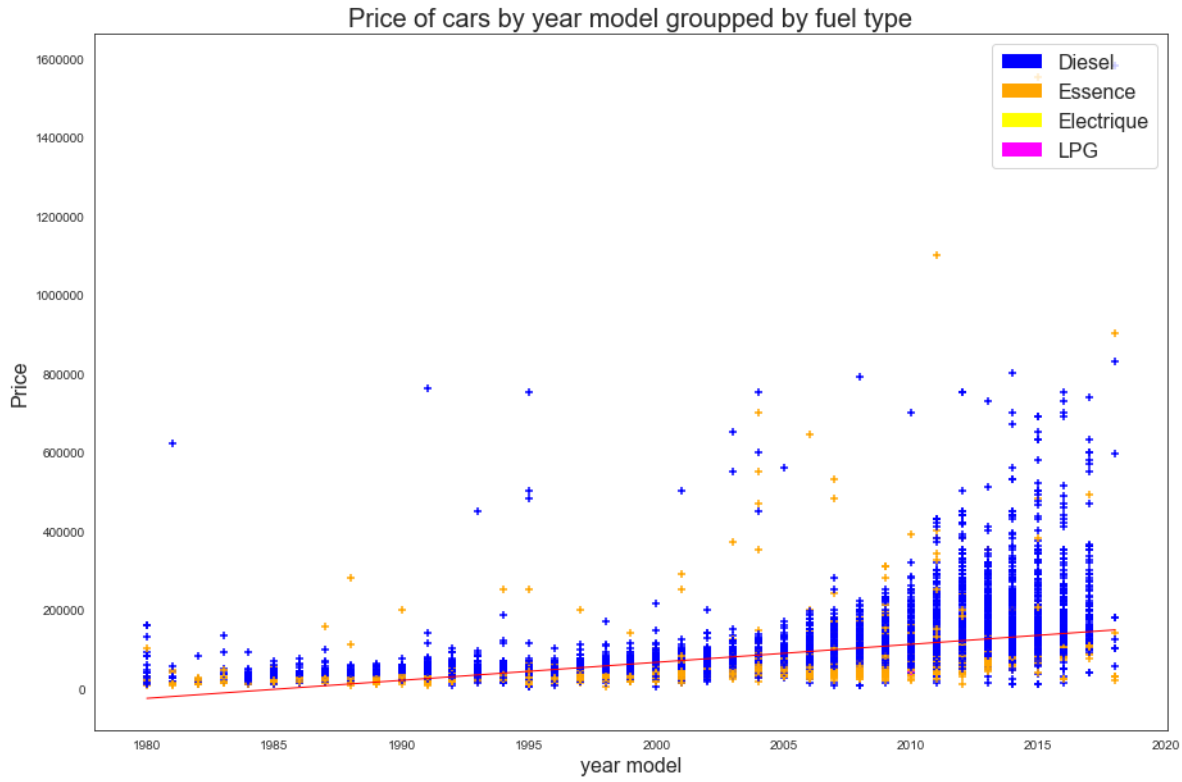
In the following plot we will visualize the price distribution by the mileage values groupping by the fuel type and draw the best fit line that express the price (target feature) by mileage.

In [21]:

```
# define a color dictionarry by fuel_type
color_dict = {'Diesel': 'blue', 'Essence': 'orange', 'Electrique': 'yellow', 'LPG': 'magent
```

In [22]:

```
# set the figure size and plot the price & mileage points with the fit line in red
fig, ax = plt.subplots(figsize = (15,10))
plt.plot(np.unique(df.year_model), np.poly1d(np.polyfit(df.year_model, df.price, 1))(np.uni
plt.scatter(df.year_model, df.price, c = [color_dict[i] for i in df.fuel_type], marker='+')
# get the list of unique fuel type
fuel_type = df.fuel_type.unique()
recs = []
for i in fuel_type:
    recs.append(mpatches.Rectangle((2,2),1,1,fc=color_dict[i]))
    plt.legend(recs,fuel_type,loc=1, fontsize = 16)

plt.title('Price of cars by year model groupped by fuel type',
          fontsize = 20)
plt.ylabel('Price', fontsize = 16)
plt.xlabel('year model', fontsize = 16)

xvals = ax.get_xticks()
ax.set_xticklabels(['{}'.format(int(x)) for x in xvals])

yvals = ax.get_yticks()
ax.set_yticklabels(['{}'.format(int(y)) for y in yvals])

plt.show()
```

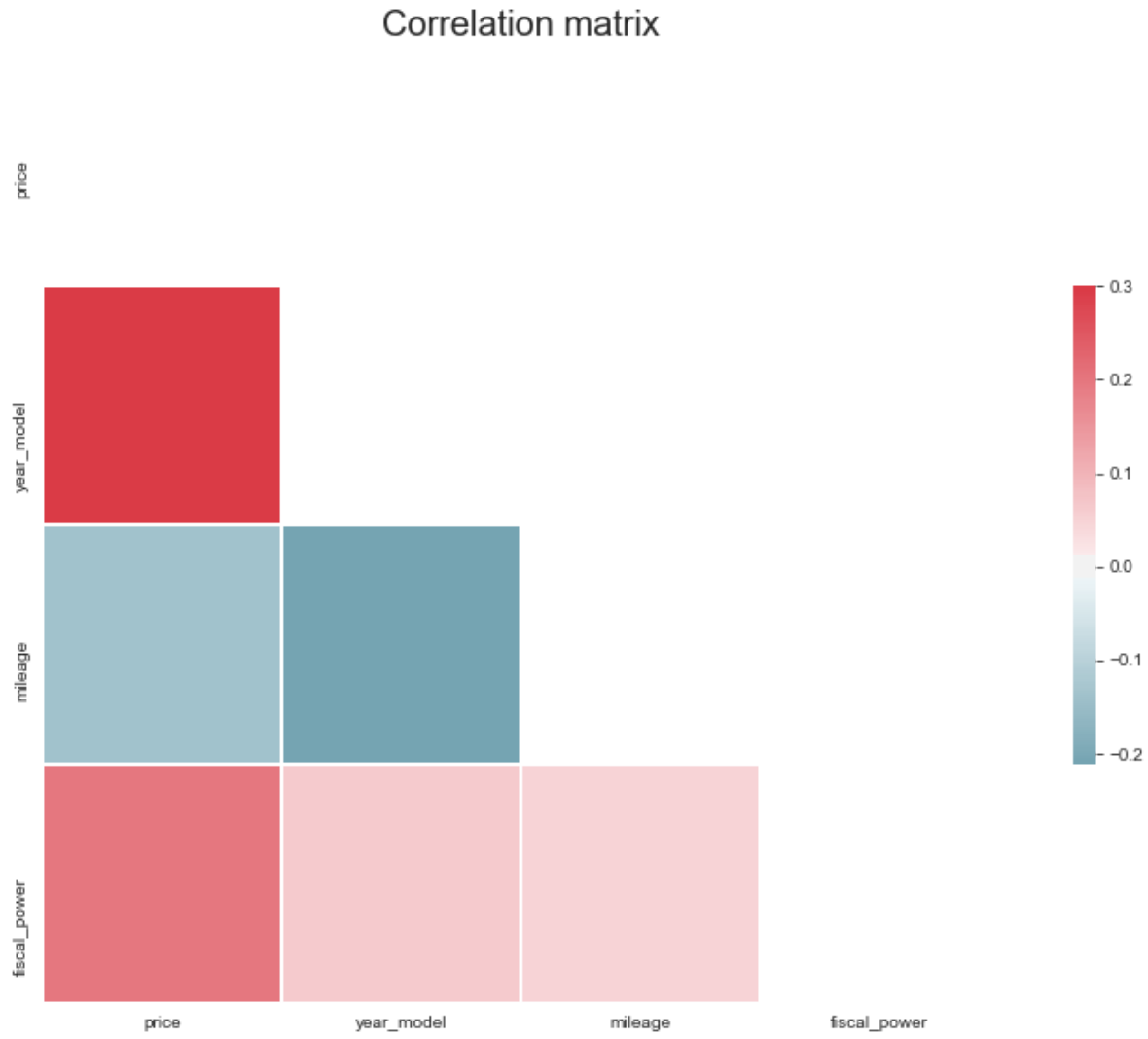


Although weak, it appears that there seems to be a positive relationship. Let's see what is the actual correlation between price and the other data points. We will look at this in 2 ways heatman for visualization and the correlation coefficient score.

Correlation matrix

In [23]:

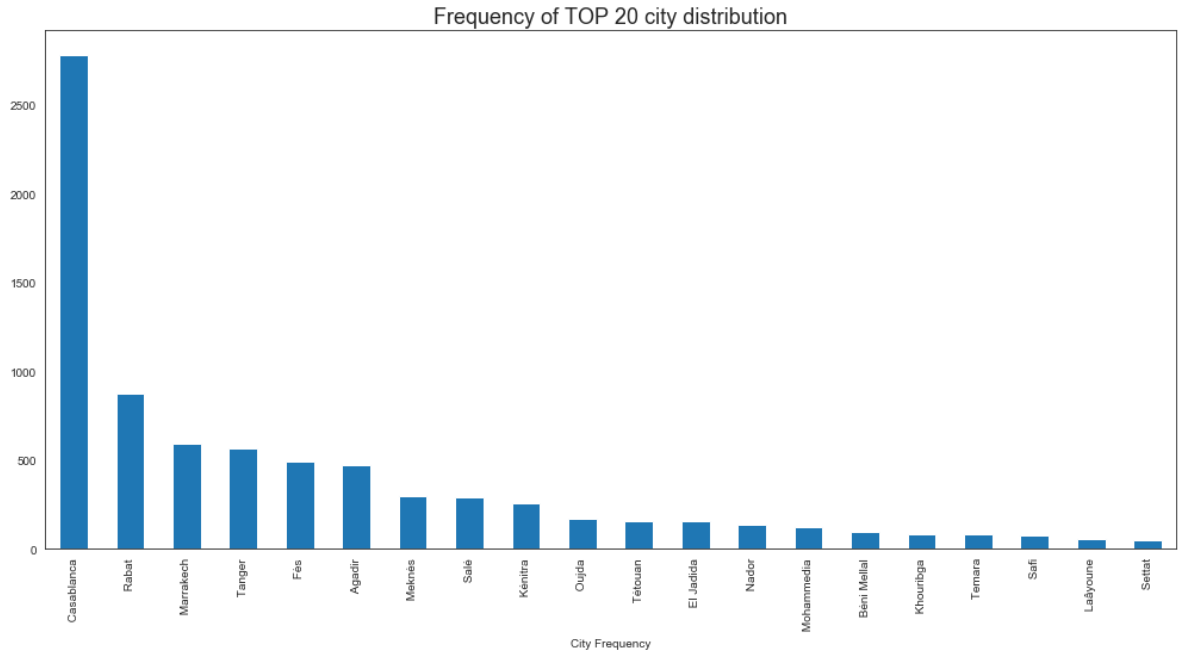
```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(15, 10))
# Compute the correlation matrix
corr = df.corr()
#print(corr)
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title('Correlation matrix',
          fontsize = 20)
plt.show()
```



Ads distribution by city

In [24]:

```
plt.figure(figsize=(17,8))
df.city.value_counts().nlargest(20).plot(kind='bar')
plt.xlabel('City Frequency')
plt.title("Frequency of TOP 20 city distribution",fontsize=18)
plt.show()
```



We can clearly visualize that most of ad publications are coming from Casablanca & Rabat, which is quite normal due to the geographic distribution of the population in those cities, furthermore the economic position of those cities beyond the other ones

Data Modeling

KNN Regression

For the moment we will use the K nearset neighbors regressor model with only numerical features, to get a basic view on our model how it behaves, then we will add other categorical features to improve it later.

In [25]:

```
## create a dataframefor testing
data = df[df.price < 400000]
```

In [26]:

```
data.head()
```

Out[26]:

	price	year_model	mileage	fuel_type	mark	model	fiscal_power	city
0	135000	2013	164999.5	Diesel	Peugeot	508	2.631837	Temara
1	53000	2008	37499.5	Diesel	Renault	Clio	2.631837	Safi
2	59000	2007	184999.5	Diesel	Citroen	C3	6.000000	Fès
3	88000	2010	37499.5	Diesel	Mercedes-Benz	220	2.631837	Nador
4	60000	2009	134999.5	Essence	Ford	Fiesta	7.000000	Fès

In [27]:

```
print(len(data))
print(len(df))
```

```
8456
8553
```

Dealing with Categorical Features

At the moment we still have 3 categorical features which are the fuel_type, mark and model the aim of this section is to pre process those features in order to make them numerical so that they will fit into our model.
In litterature there is two famous kind of categorical variable transformation, the first one is **label encoding**, and the second one is the **one hot encoding**, for this use case we will use the one hot position and the reason why I will choose this kind of data labeling is because I will not need any kind of data normalisation later, and also This has the benefit of not weighting a value improperly but does have the downside of adding more columns to the data set.

In [28]:

```
X = data[['year_model', 'mileage', 'fiscal_power', 'fuel_type', 'mark']]
Y = data.price
X = pd.get_dummies(data=X)
```

In [29]:

```
X.head()
```

Out[29]:

	year_model	mileage	fiscal_power	fuel_type_Diesel	fuel_type_Electrique	fuel_type_Essence
0	2013	164999.5	2.631837	1	0	0
1	2008	37499.5	2.631837	1	0	0
2	2007	184999.5	6.000000	1	0	0
3	2010	37499.5	2.631837	1	0	0
4	2009	134999.5	7.000000	0	0	1

5 rows × 61 columns

Data Splitting

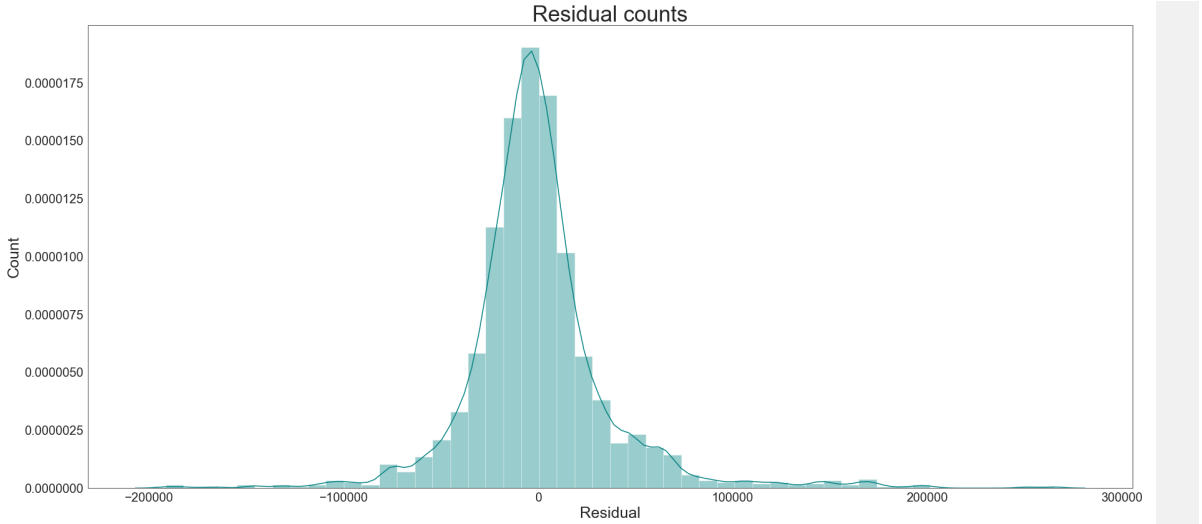
Usually we split our data into three parts : Training , validation and Testing set, but for simplicity we will use only train and test with 20% in test size.

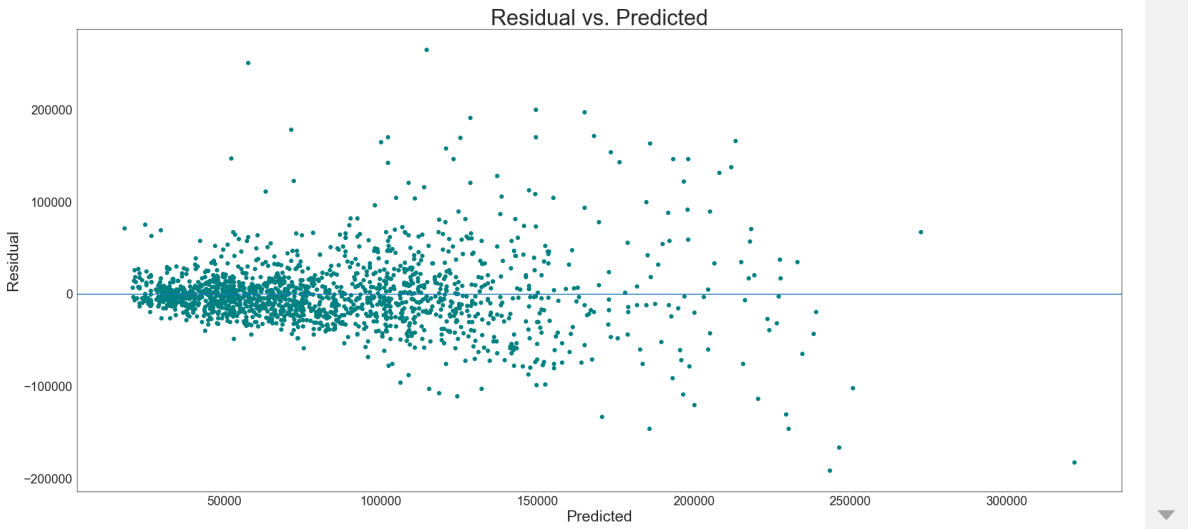
In [30]:

```
# now we use the train_test_split function already available in sklearn library to split ou
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .20, random_state = 4
```

In [31]:

```
from sklearn import neighbors
# the value of n_neighbors will be changed when we plot the histogram showing the lowest RM
knn = neighbors.KNeighborsRegressor(n_neighbors=6)
knn.fit(X_train, Y_train)

predicted = knn.predict(X_test)
residual = Y_test - predicted

fig = plt.figure(figsize=(30,30))
ax1 = plt.subplot(211)
sns.distplot(residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.title('Residual counts',fontsize=35)
plt.xlabel('Residual',fontsize=25)
plt.ylabel('Count',fontsize=25)

ax2 = plt.subplot(212)
plt.scatter(predicted, residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.xlabel('Predicted',fontsize=25)
plt.ylabel('Residual',fontsize=25)
plt.axhline(y=0)
plt.title('Residual vs. Predicted',fontsize=35)

plt.show()

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(Y_test, predicted))
print('RMSE:')
print(rmse)
```






```
RMSE:
37709.6749493281
```

In [32]:

```
from sklearn.metrics import r2_score
print('Variance score: %.2f' % r2_score(Y_test, predicted))
```

```
Variance score: 0.56
```

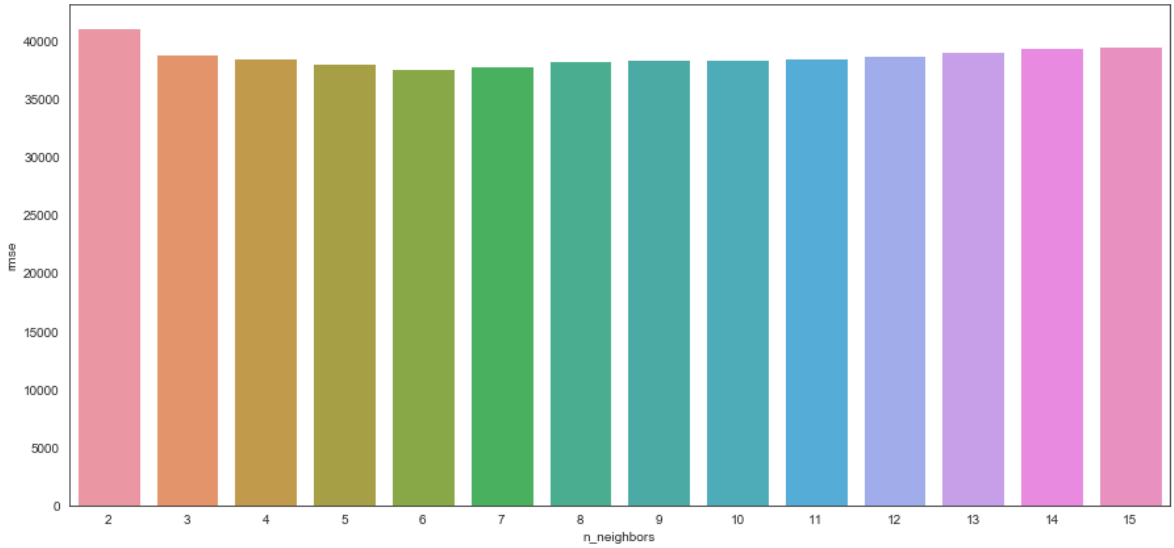
as we can see we got 56% in the r^2 score by using n_neighbors = 6, we still don't know if it's the optimal number of neighors or not, so for that we will plot a histogram of different Root Mean Squared Error by n_neighbors and see who's have the lowest RMSE value, and another thing is that the mean of cross validation values is very low which may indicate that our model had overfitted.

In [33]:

```
rmse_l = []
num = []
for n in range(2, 16):
    knn = neighbors.KNeighborsRegressor(n_neighbors=n)
    knn.fit(X_train, Y_train)
    predicted = knn.predict(X_test)
    rmse_l.append(np.sqrt(mean_squared_error(Y_test, predicted)))
    num.append(n)
```

In [34]:

```
df_plt = pd.DataFrame()
df_plt['rmse'] = rmse_l
df_plt['n_neighbors'] = num
ax = plt.figure(figsize=(15,7))
sns.barplot(data = df_plt, x = 'n_neighbors', y = 'rmse')
plt.show()
```

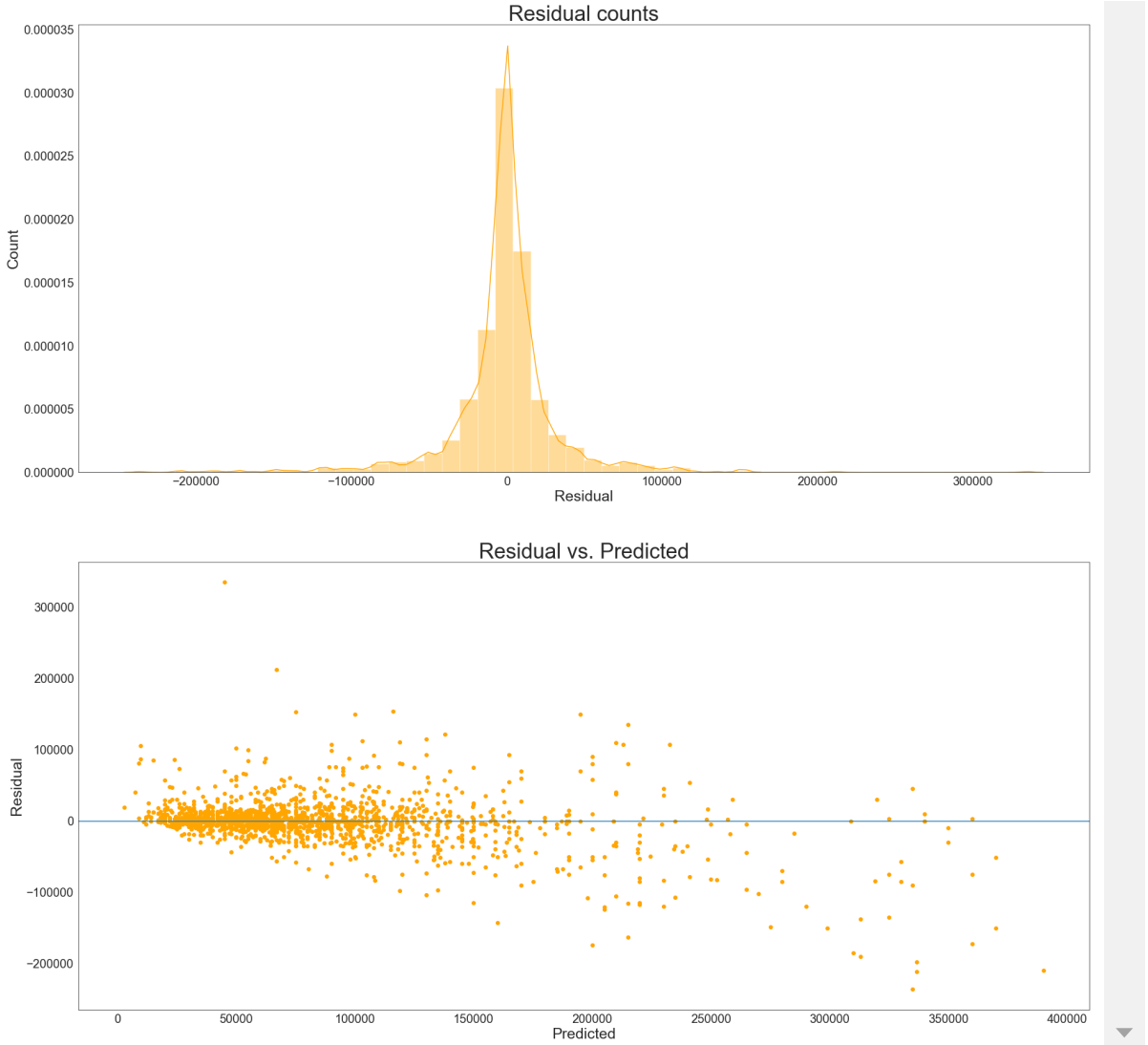


It appears that 6 nearest neighbors is the optimal number of neighbors.

Descision Tree Regression

In [35]:

```
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(max_features='auto')
dtr.fit(X_train, Y_train)
predicted = dtr.predict(X_test)
residual = Y_test - predicted

fig = plt.figure(figsize=(30,30))
ax1 = plt.subplot(211)
sns.distplot(residual, color ='orange')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.title('Residual counts',fontsize=35)
plt.xlabel('Residual',fontsize=25)
plt.ylabel('Count',fontsize=25)

ax2 = plt.subplot(212)
plt.scatter(predicted, residual, color ='orange')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.xlabel('Predicted',fontsize=25)
plt.ylabel('Residual',fontsize=25)
plt.axhline(y=0)
plt.title('Residual vs. Predicted',fontsize=35)

plt.show()

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(Y_test, predicted))
print('RMSE:')
print(rmse)
```




```
RMSE:
34611.061089032795
```

In [36]:

```
print('Variance score: %.2f' % r2_score(Y_test, predicted))
```

```
Variance score: 0.63
```

The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) (or sometimes root-mean-squared error) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed. The RMSD represents the sample standard deviation of the differences between predicted values and observed values. These individual differences are called residuals when the calculations are performed over the data sample that was used for estimation, and are called prediction errors when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. RMSD is a measure of accuracy, to compare forecasting errors of different models for a particular data and not between datasets, as it is scale-dependent. ~ WikiPedia
By comparing the Tree Regression with the KNN Regression we can see that the RMSE was reduced from 37709 to 34392 which let us say that this model is more accurate than the last one, but that's not all of it, we still have to test other regression algorithm to check if there is any improvement in result.

Interpretation

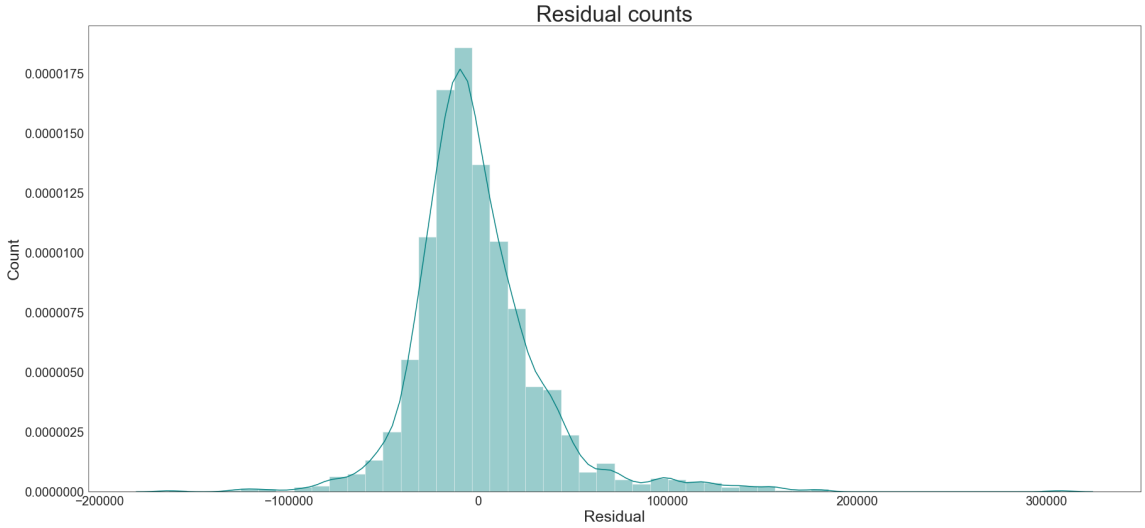
By looking at the last RMSE score we've vast improvements, as you can see from the "Residual vs. Predicted" that the predicted score is closer to zero and is tighter around the lines which means that we are guessing alot closer to the price.

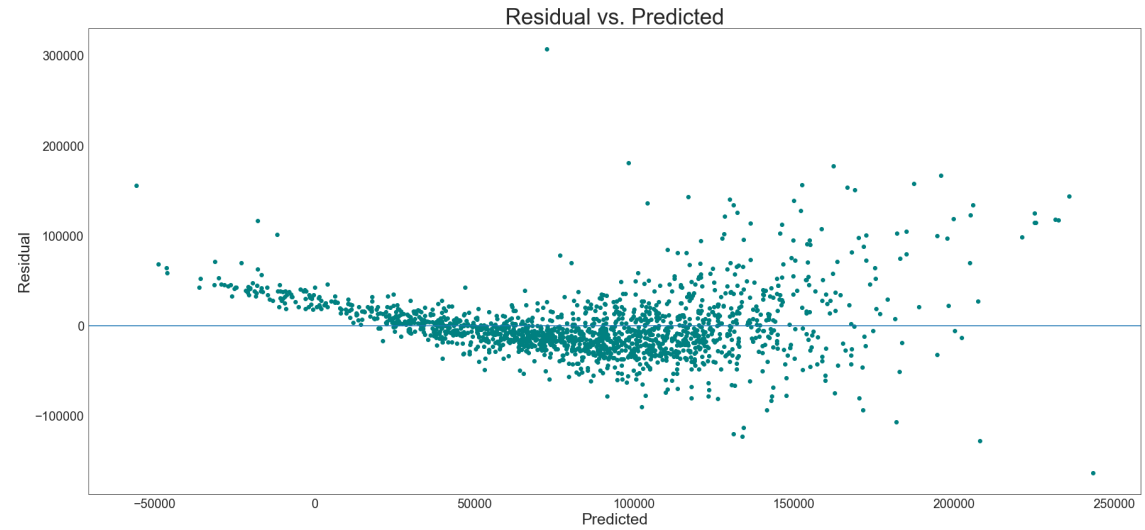
What about Simple Linear Regression

Linear Regression

In [37]:

```
from sklearn import linear_model

regr = linear_model.LinearRegression()
regr.fit(X_train, Y_train)

predicted = regr.predict(X_test)
residual = Y_test - predicted

fig = plt.figure(figsize=(30,30))
ax1 = plt.subplot(211)
sns.distplot(residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.title('Residual counts',fontsize=35)
plt.xlabel('Residual',fontsize=25)
plt.ylabel('Count',fontsize=25)

ax2 = plt.subplot(212)
plt.scatter(predicted, residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.xlabel('Predicted',fontsize=25)
plt.ylabel('Residual',fontsize=25)
plt.axhline(y=0)
plt.title('Residual vs. Predicted',fontsize=35)

plt.show()

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(Y_test, predicted))
print('RMSE:')
print(rmse)
```






```
RMSE:
34865.07324134517
```

In [38]:

```
print('Variance score: %.2f' % r2_score(Y_test, predicted))
```

```
Variance score: 0.62
```

Boosting

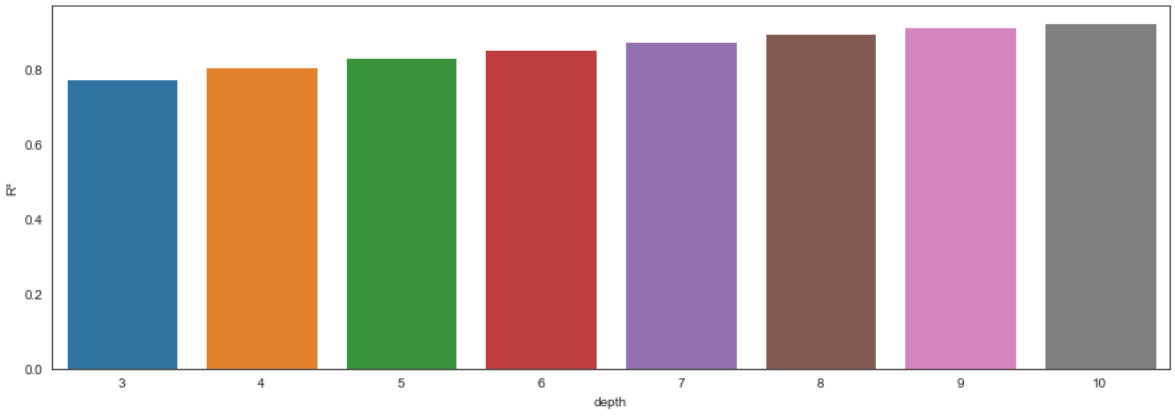
Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. Boosting is based on the question posed by Kearns and Valiant (1988, 1989): Can a set of weak learners create a single strong learner? A weak learner is defined to be a classifier which is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification. ~ WikiPedia)
Let's see if boosting can improve our scores.

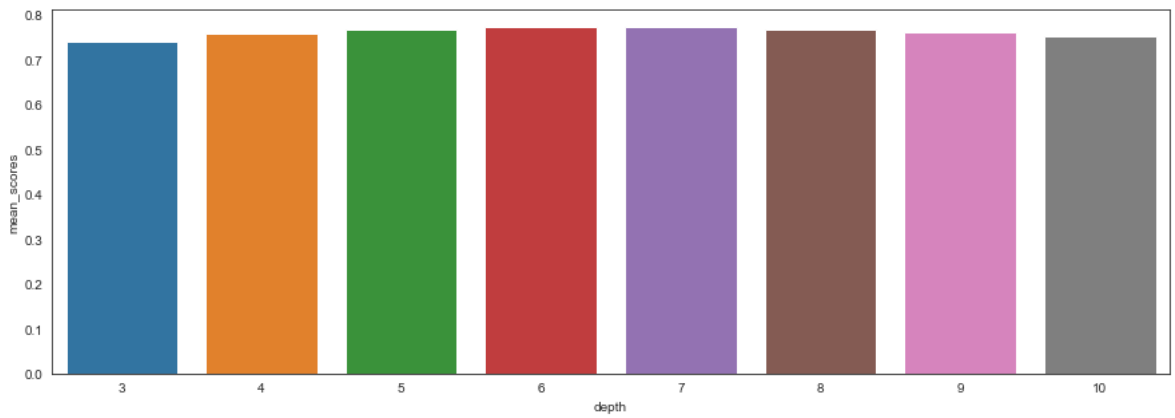
In [39]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score

r_sq = []
deep = []
mean_scores = []

#loss : {‘ls’, ‘lad’, ‘huber’, ‘quantile’}
for n in range(3, 11):
    gbr = GradientBoostingRegressor(loss ='ls', max_depth=n)
    gbr.fit (X, Y)
    deep.append(n)
    r_sq.append(gbr.score(X, Y))
    mean_scores.append(cross_val_score(gbr, X, Y, cv=12).mean())
```

In [40]:

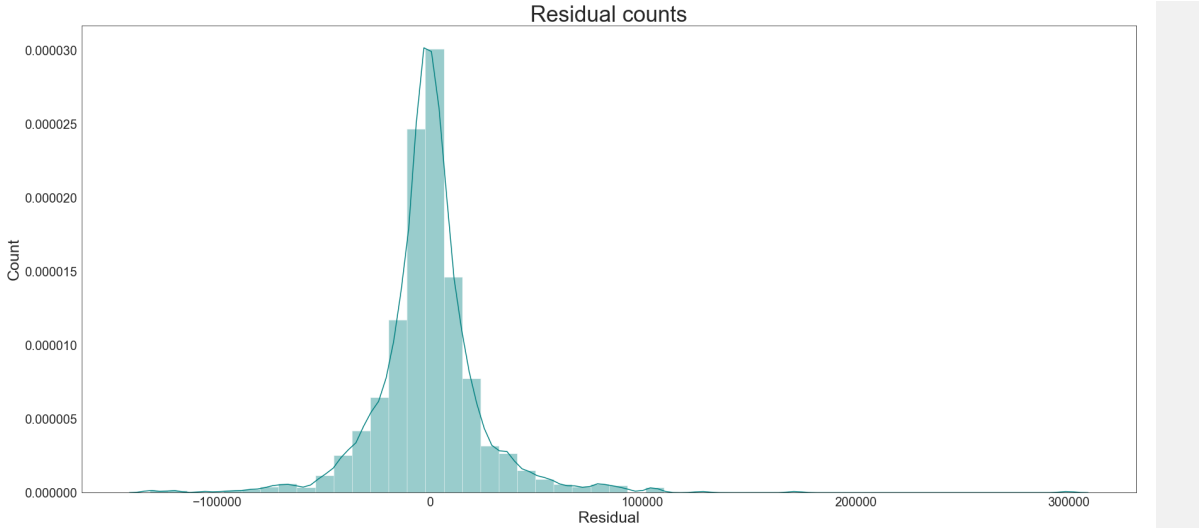
```
plt_gbr = pd.DataFrame()

plt_gbr['mean_scores'] = mean_scores
plt_gbr['depth'] = deep
plt_gbr['R²'] = r_sq

f, ax = plt.subplots(figsize=(15, 5))
sns.barplot(data = plt_gbr, x='depth', y='R²')
plt.show()

f, ax = plt.subplots(figsize=(15, 5))
sns.barplot(data = plt_gbr, x='depth', y='mean_scores')
plt.show()
```

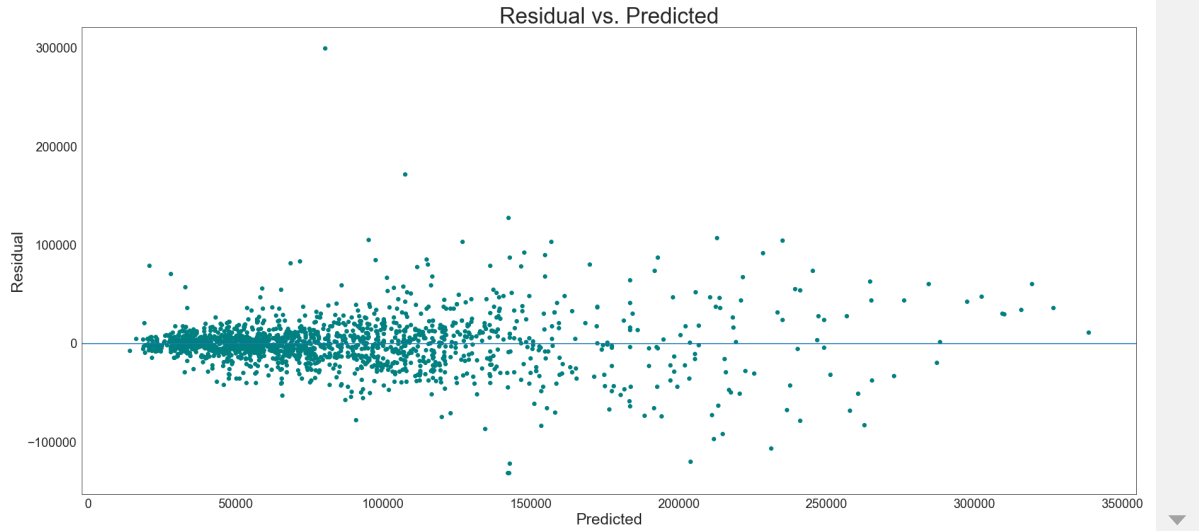




In [41]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score

gbr = GradientBoostingRegressor(loss ='ls', max_depth=6)
gbr.fit (X_train, Y_train)
predicted = gbr.predict(X_test)
residual = Y_test - predicted

fig = plt.figure(figsize=(30,30))
ax1 = plt.subplot(211)
sns.distplot(residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.title('Residual counts',fontsize=35)
plt.xlabel('Residual',fontsize=25)
plt.ylabel('Count',fontsize=25)

ax2 = plt.subplot(212)
plt.scatter(predicted, residual, color ='teal')
plt.tick_params(axis='both', which='major', labelsize=20)
plt.xlabel('Predicted',fontsize=25)
plt.ylabel('Residual',fontsize=25)
plt.axhline(y=0)
plt.title('Residual vs. Predicted',fontsize=35)

plt.show()

rmse = np.sqrt(mean_squared_error(Y_test, predicted))
scores = cross_val_score(gbr, X, Y, cv=12)

print('\nCross Validation Scores:')
print(scores)
print('\nMean Score:')
print(scores.mean())
print('\nRMSE:')
print(rmse)
```






```
Cross Validation Scores:
[0.80449111 0.77468475 0.80062385 0.73325984 0.75030249 0.85785739
 0.80636564 0.73414505 0.77495508 0.74237348 0.67785124 0.81118824]

Mean Score:
0.7723415139514206

RMSE:
25104.762094810307
```

In [42]:

```
print('Variance score: %.2f' % r2_score(Y_test, predicted))
```

```
Variance score: 0.80
```

Prediction VS Real price histogram

First of all we reshape our data to a 1D array then we plot the histogram doing the comparison between the real price and the predicted ones.

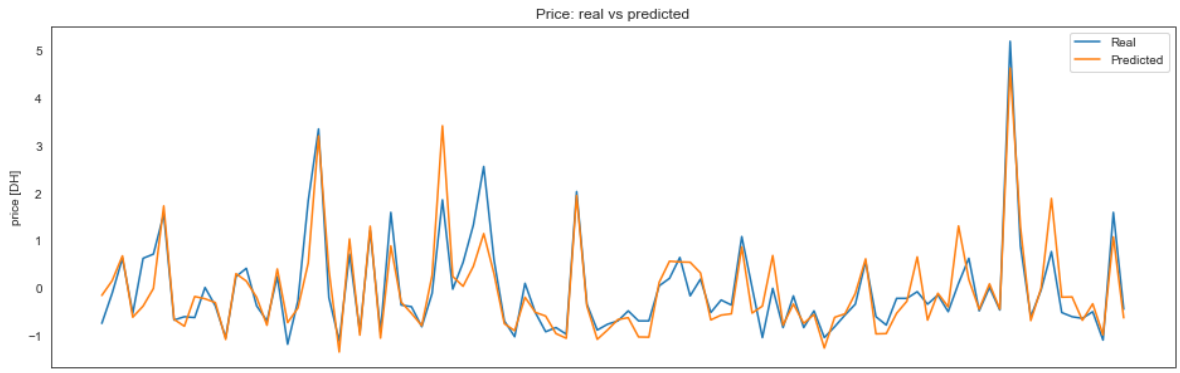
In [51]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
#A = scaler.fit_transform(Y_test.reshape(-1,1))
A_test = pd.DataFrame(Y_test)
A = scaler.fit_transform(A_test.values.reshape(-1,1))
B_pred = pd.DataFrame(predicted)
#A = Y_test.reshape(-1, 1)
B = scaler.fit_transform(B_pred.values.reshape(-1, 1))
```

```
C:\Users\Aman\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: D
ataConversionWarning: Data with input dtype int32 was converted to float64 b
y StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\Aman\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: D
ataConversionWarning: Data with input dtype int32 was converted to float64 b
y StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [52]:

```
plt.rcParams['figure.figsize'] = 16,5
plt.figure()
plt.plot(A[-100:], label="Real")
plt.plot(B[-100:], label="Predicted")
plt.legend()
plt.title("Price: real vs predicted")
plt.ylabel("price [DH]")
plt.xticks(())
plt.show()
```



We can notice clearly that the two line (real vs predicted) fit each other well, with some small differences which let us say that we did a good improvement compared with the first model.

Model Evaluation

It appears that the Gradient Boosting model regressor win the battle with the lowest RMSE value and the highest R^2 score. In the following table we will do a benchmarking resuming all the models tested above.

Model	Variance Score	RMSE

Model	Variance Score	RMSE
KNN	0.56	37709.67
Multiple Linear Regression	0.62	34865.07
Gradient Boosting	0.80	25176.16
Decision Tree	**0.63**	**34551.17**

Since the Gradient Boosting regressor is the winner, we will now inspect its coeficients and interceptors.

Let's predict an observation never seen before

To do that we first build a fucntion that takes a simple user input and transform it to a one hot encoding.

In [53]:

```
# user_input = [2010, 124999.5, 6, 'Diesel', 'BMW']
user_input = {'year_model':2006, 'mileage':82499.5, 'fiscal_power':6, 'fuel_type':'Diesel',
def input_to_one_hot(data):
    # initialize the target vector with zero values
    enc_input = np.zeros(61)
    # set the numerical input as they are
    enc_input[0] = data['year_model']
    enc_input[1] = data['mileage']
    enc_input[2] = data['fiscal_power']
    ###################### Mark #########################
    # get the array of marks categories
    marks = df.mark.unique()
    # redefine the the user inout to match the column name
    redefinded_user_input = 'mark_'+data['mark']
    # search for the index in columns name list
    mark_column_index = X.columns.tolist().index(redefinded_user_input)
    #print(mark_column_index)
    # fullfill the found index with 1
    enc_input[mark_column_index] = 1
    ###################### Fuel Type ####################
    # get the array of fuel type
    fuel_types = df.fuel_type.unique()
    # redefine the the user inout to match the column name
    redefinded_user_input = 'fuel_type_'+data['fuel_type']
    # search for the index in columns name list
    fuelType_column_index = X.columns.tolist().index(redefinded_user_input)
    # fullfill the found index with 1
    enc_input[fuelType_column_index] = 1
    return enc_input
```

In [54]:

```
print(input_to_one_hot(user_input))
```

```
[2.00600e+03 8.24995e+04 6.00000e+00 1.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 1.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
 0.00000e+00]
```

In [55]:

```
a = input_to_one_hot(user_input)
```

In [56]:

```
price_pred = gbr.predict([a])
```

In [57]:

```
price_pred[0]
```

Out[57]:

```
59156.0041509517
```

Save the best Model

In [58]:

```
from sklearn.externals import joblib

joblib.dump(gbr, 'model.pkl')
```

Out[58]:

```
['model.pkl']
```

In [59]:

```
gbr = joblib.load('model.pkl')
```

In [60]:

```
print("the best price for this Dacia is",gbr.predict([a])[0])
```

```
the best price for this Dacia is 59156.0041509517
```