CAR PRİCE PREDİCTİON

- Importing necessary Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from scipy.stats import ttest_1samp,shapiro
import plotly.express as px
import plotly.graph_objects as go
import math
from warnings import filterwarnings
filterwarnings("ignore")
```

- Loading and Exploring the dataset

In [3]:
```
df = pd.read_csv("CarPrice_Assignment.csv")
```

In [4]:
```
def check(df):
    l=[]
    columns=df.columns
    for col in columns:
        dtypes=df[col].dtypes
        nunique=df[col].nunique()
        sum_null=df[col].isnull().sum()
        l.append([col,dtypes,nunique,sum_null])
    df_check=pd.DataFrame(l)
    df_check.columns=['column','dtypes','nunique','sum_null']
    return df_check
check(df)
```

Out[4]:

| | column | dtypes | nunique | sum_null |
|---|---|---|---|---|
| 0 | car_ID | int64 | 205 | 0 |
| 1 | symboling | int64 | 6 | 0 |
| 2 | CarName | object | 147 | 0 |
| 3 | fueltype | object | 2 | 0 |
| 4 | aspiration | object | 2 | 0 |
| 5 | doornumber | object | 2 | 0 |
| 6 | carbody | object | 5 | 0 |
| 7 | drivewheel | object | 3 | 0 |
| 8 | enginelocation | object | 2 | 0 |
| 9 | wheelbase | float64 | 53 | 0 |
| 10 | carlength | float64 | 75 | 0 |
| 11 | carwidth | float64 | 44 | 0 |
| 12 | carheight | float64 | 49 | 0 |
| 13 | curbweight | int64 | 171 | 0 |

| | | | | |
|---|---|---|---|---|
| **14** | enginetype | object | 7 | 0 |
| **15** | cylindernumber | object | 7 | 0 |
| **16** | enginesize | int64 | 44 | 0 |
| **17** | fuelsystem | object | 8 | 0 |
| **18** | boreratio | float64 | 38 | 0 |
| **19** | stroke | float64 | 37 | 0 |
| **20** | compressionratio | float64 | 32 | 0 |
| **21** | horsepower | int64 | 59 | 0 |
| **22** | peakrpm | int64 | 23 | 0 |
| **23** | citympg | int64 | 29 | 0 |
| **24** | highwaympg | int64 | 30 | 0 |
| **25** | price | float64 | 189 | 0 |

In [5]:
```
df.describe()    #statistical information
```

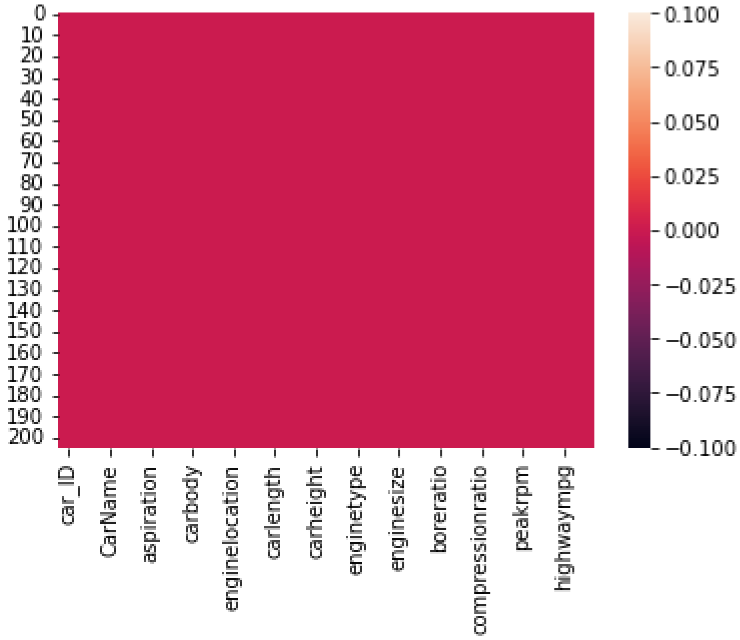
Out[5]:

| | car_ID | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | borerati |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.00000 |
| **mean** | 103.000000 | 0.834146 | 98.756585 | 174.049268 | 65.907805 | 53.724878 | 2555.565854 | 126.907317 | 3.32975 |
| **std** | 59.322565 | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 0.27084 |
| **min** | 1.000000 | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 2.54000 |
| **25%** | 52.000000 | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 3.15000 |
| **50%** | 103.000000 | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 3.31000 |
| **75%** | 154.000000 | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 3.58000 |
| **max** | 205.000000 | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 3.94000 |

- Checking for missing value

In [6]:
```
sns.heatmap(df.isnull())
```

Out[6]:
```
<AxesSubplot:>
```



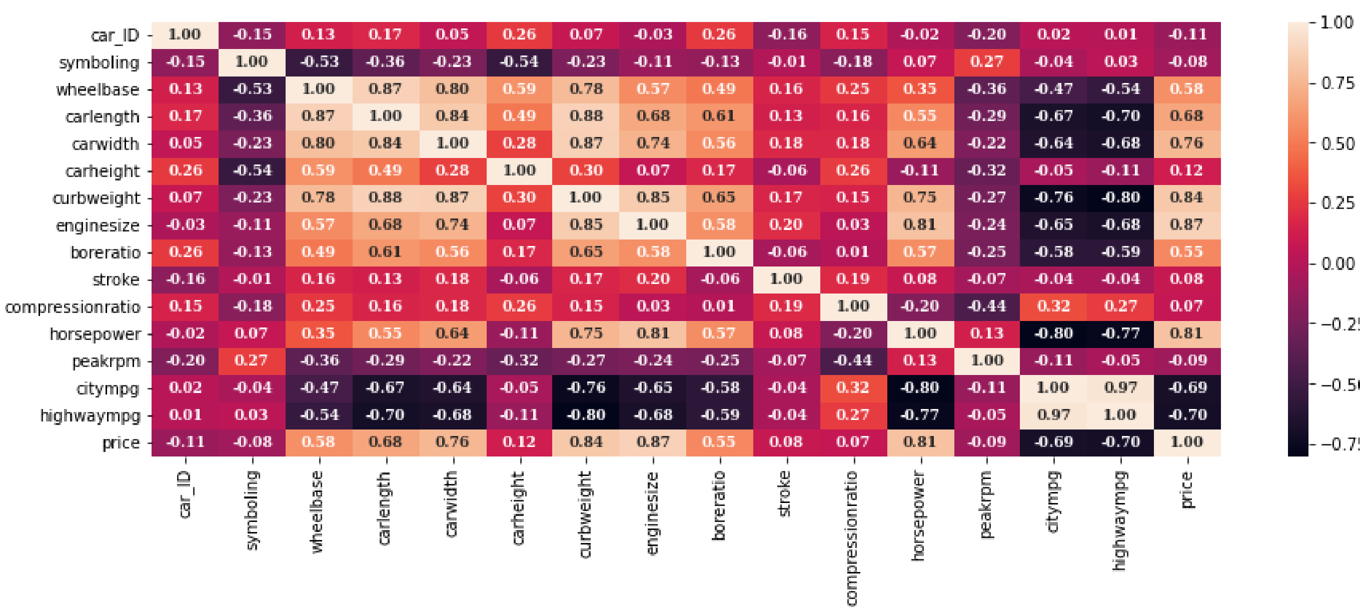

It seems that there is no missing value

CORRELATİON

In [7]:

```
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True,fmt=".2f",
                annot_kws={
                        "fontsize":9,
                        "fontweight":"bold",
                        "fontfamily":"serif"})
```

Out[7]: <AxesSubplot:>




NOTE: For linear regression, the dependent variable is price and the independent variable is horsepower.

- Creating a new data frame with the data we will use

In [8]:

```
yeni_df=df[["peakrpm","horsepower","price","enginesize"]]
```

- Exploring the dataset

In [9]:

```python
def check(df):
    l=[]
    columns=df.columns
    for col in columns:
        dtypes=df[col].dtypes
        nunique=df[col].nunique()
        sum_null=df[col].isnull().sum()
        l.append([col,dtypes,nunique,sum_null])
    df_check=pd.DataFrame(l)
    df_check.columns=['column','dtypes','nunique','sum_null']
    return df_check
check(yeni_df)
```

Out[9]:

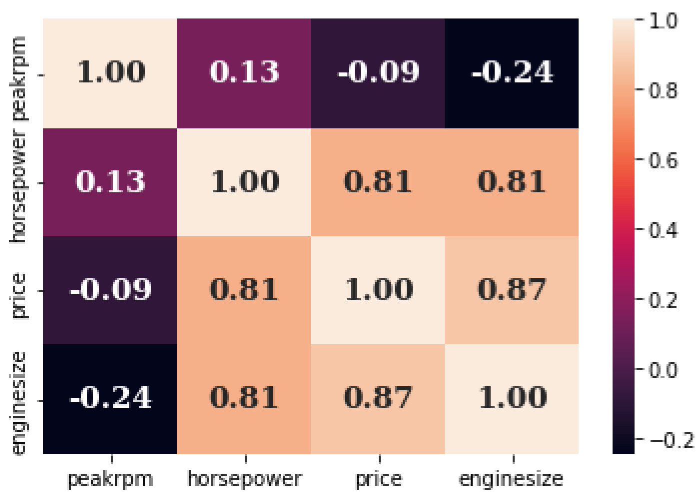
| | column | dtypes | nunique | sum_null |
|---|---|---|---|---|
| **0** | peakrpm | int64 | 23 | 0 |
| **1** | horsepower | int64 | 59 | 0 |
| **2** | price | float64 | 189 | 0 |
| **3** | enginesize | int64 | 44 | 0 |

- Looking at heatmap and pairplot for correlation and linear regression

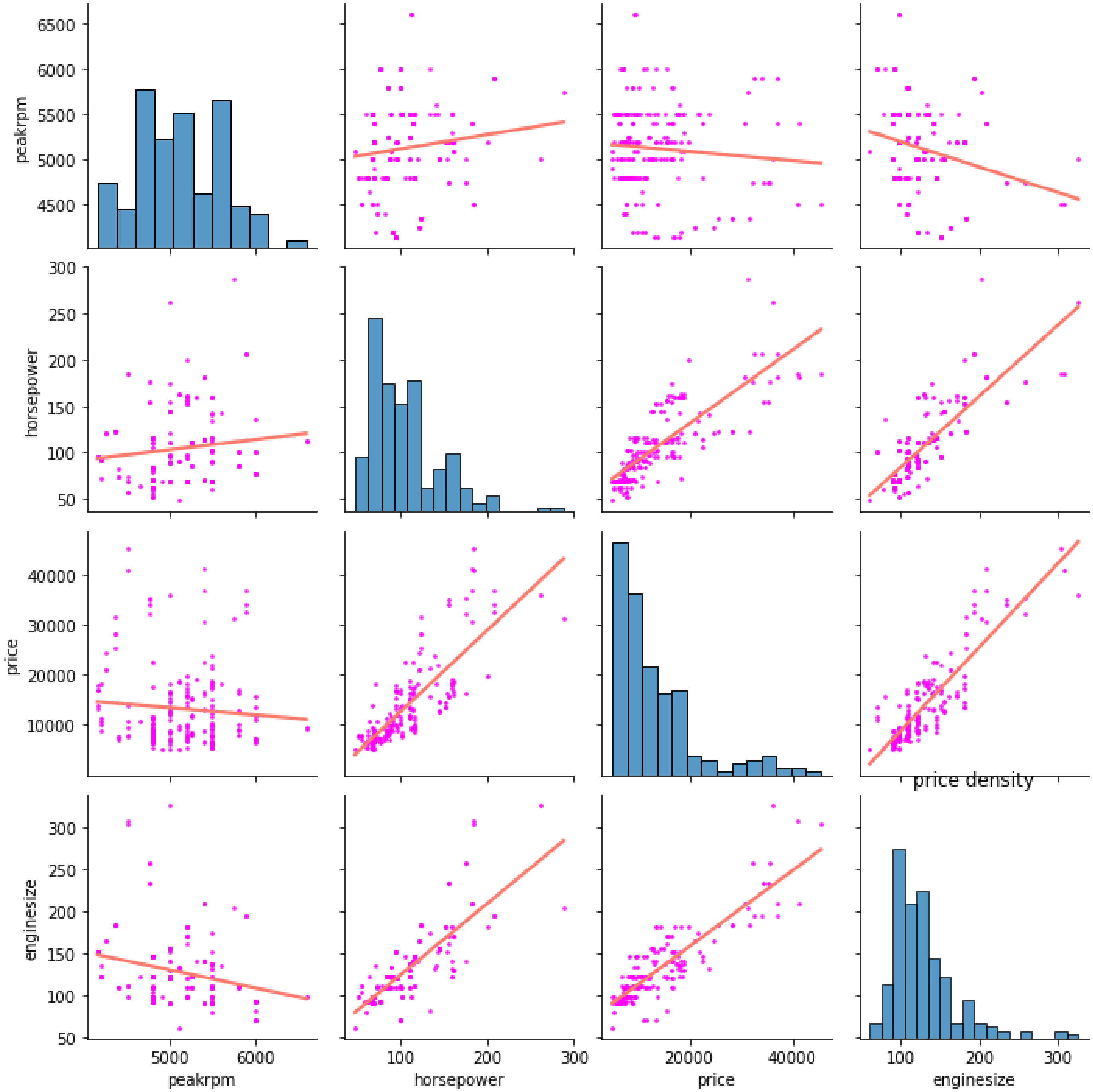
In [10]:

```python
yeni_df.head()
sns.heatmap(yeni_df.corr(),annot=True,fmt=".2f",
                annot_kws={
                        "fontsize":15,
                        "fontweight":"bold",
                        "fontfamily":"serif"})


plt.figure(figsize=(12,9))
sns.pairplot(yeni_df,kind="reg",plot_kws={'ci':None, 'color': 'xkcd:salmon',
                        'scatter_kws': {'color': 'fuchsia', 's': 3}})
plt.title("price density")
plt.show()
```

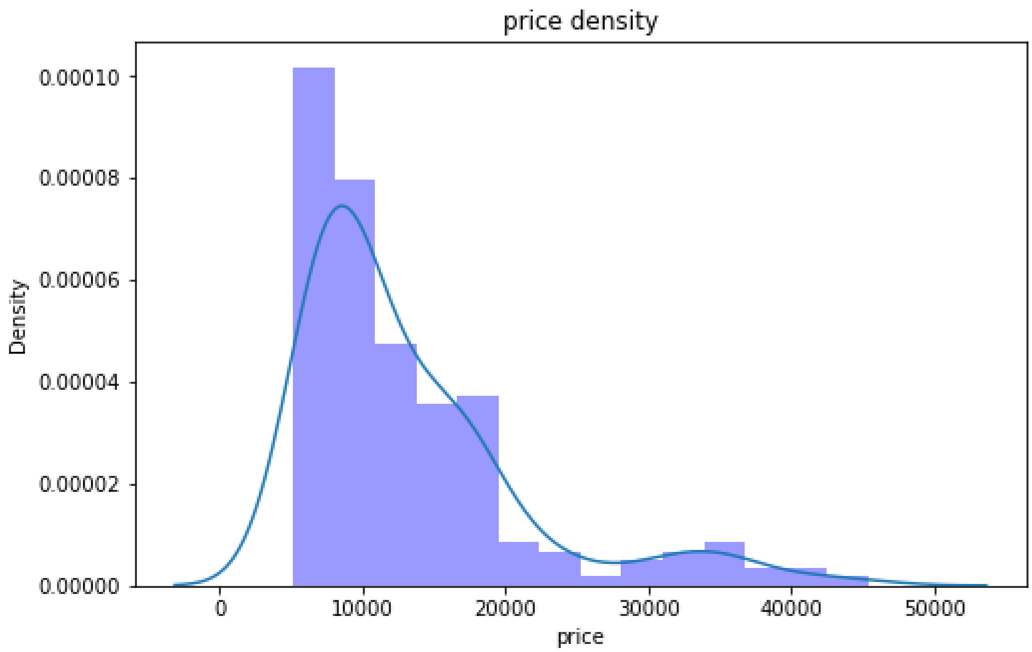



```
<Figure size 864x648 with 0 Axes>
```




• Determining whether the dependent variable(y) fits the normal distribution

In [11]:
```
plt.figure(figsize=(8,5))
sns.distplot(yeni_df["price"],hist_kws={"color":"b"})
plt.title("price density")
plt.show()
```




The distribution that can be understood from the graph is a right skewed distribution and not a normal distribution. but let's refer to the Shapiro-Wilk test just to be sure

In [12]:
```
test_stat,p_value=shapiro(yeni_df["horsepower"])
print("Price p value=%.8f"%p_value)
```

```
Price p value=0.00000000
```

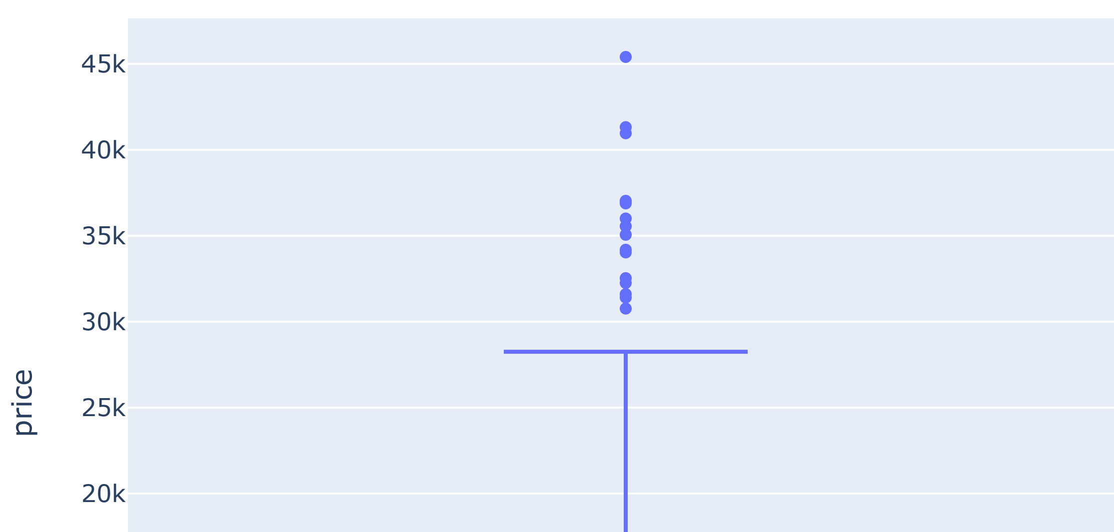
Since the p value is less than 0.05, we can say that the independent variable does not fit the normal distribution.

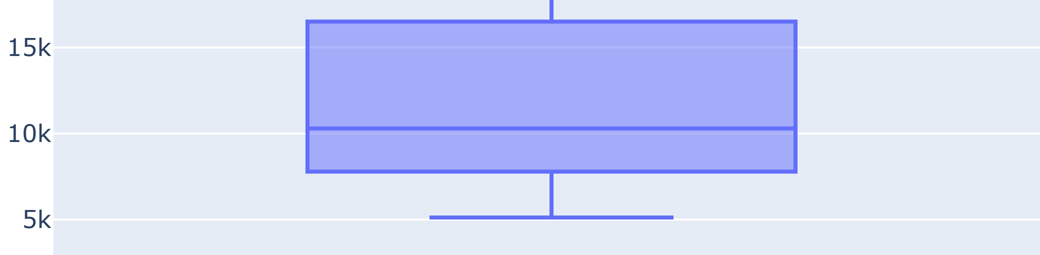
.

OUTLİERS

In [13]:
```
df_price=df["price"].copy()
```

In [14]:
```
import plotly.express as px
fig = px.box(df, y='price')
fig.show()
```






- Extreme values

In [15]:
```
Q1=df_price.quantile(0.25)
Q3=df_price.quantile(0.75)
IQR=Q3-Q1

lower_bound=Q1 - 1.5*IQR
upper_bound=Q3 + 1.5*IQR
print("lower bound is " + str(lower_bound))
print("upper bound is " + str(upper_bound))
print("Q1 ",Q1)
print("Q3 ",Q3)
```

```
lower bound is -5284.5
upper bound is 29575.5
Q1  7788.0
Q3  16503.0
```

In [16]:
```
outliers_vector=(df_price < (lower_bound)) | (df_price > (upper_bound))
outliers_vector
```

Out[16]:
```
0       False
1       False
2       False
3       False
4       False
        ...
200     False
201     False
202     False
203     False
204     False
Name: price, Length: 205, dtype: bool
```

In [17]:
```
outliers = df_price[outliers_vector]
outliers.index
```

Out[17]:
```
Int64Index([15, 16, 17, 47, 48, 49, 70, 71, 72, 73, 74, 126, 127, 128, 129], dtype='int6
4')
```

- Deletion of outlier observation

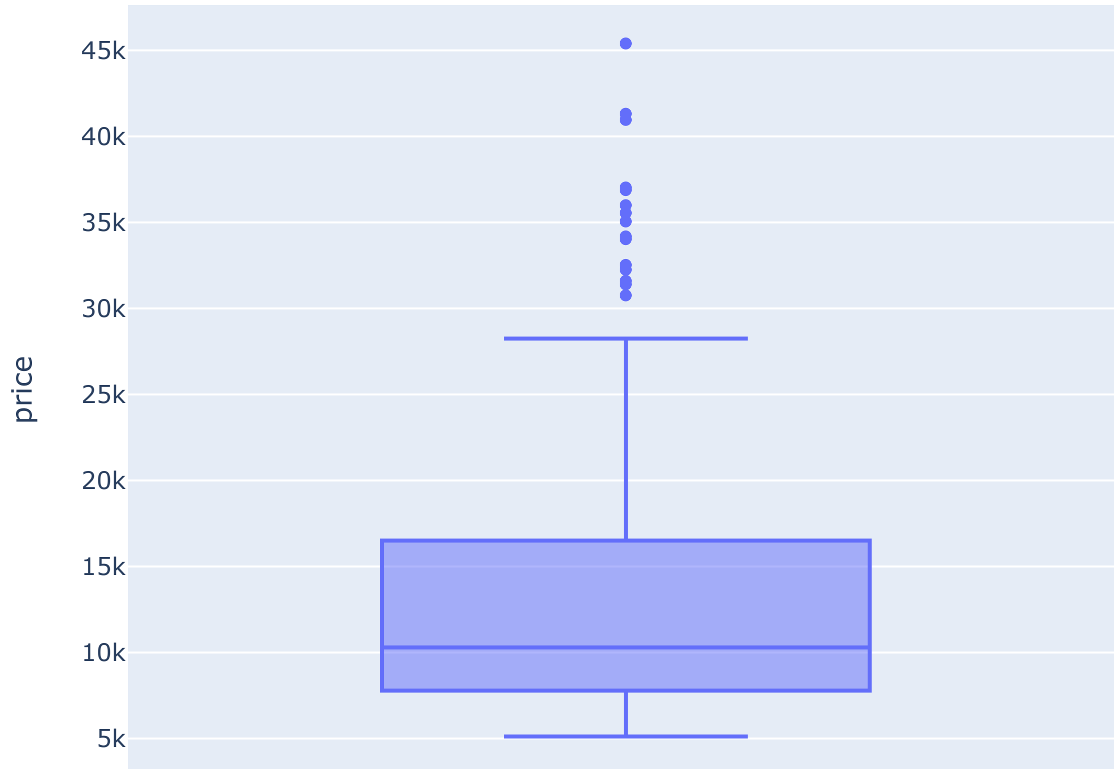
In [18]:
```
clean_df_price = df_price[~(df_price < (lower_bound)) | (df_price > (upper_bound))]
clean_df_price.shape
```

Out[18]:
```
(205,)
```

- Fill with average

In [19]:
```
import plotly.express as px
```

```
fig = px.box(df, y='price')
fig.show()
```




In [20]:
```
df_price.mean()
```

Out[20]:
```
13276.710570731706
```

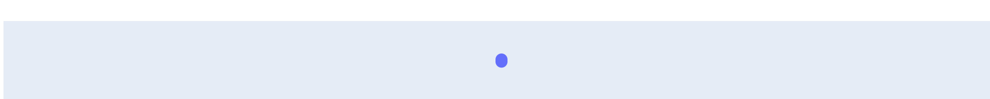
In [21]:
```
df_price[outliers_vector]=df_price.mean()
df_price[outliers_vector].head()
```

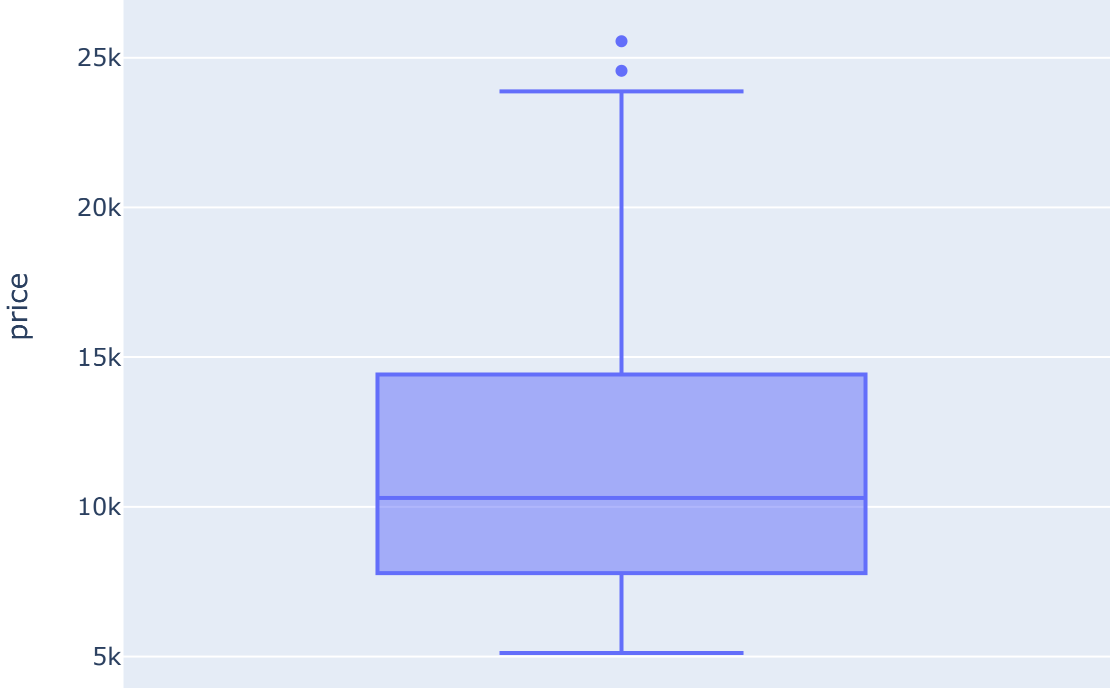
Out[21]:
```
15    13276.710571
16    13276.710571
17    13276.710571
47    13276.710571
48    13276.710571
Name: price, dtype: float64
```

In [22]:
```
ilk_price=df["price"]
```

In [23]:
```
df["price"]=df_price
```

In [24]:
```
import plotly.express as px
figg= px.box(df, y='price')
figg.show()
```



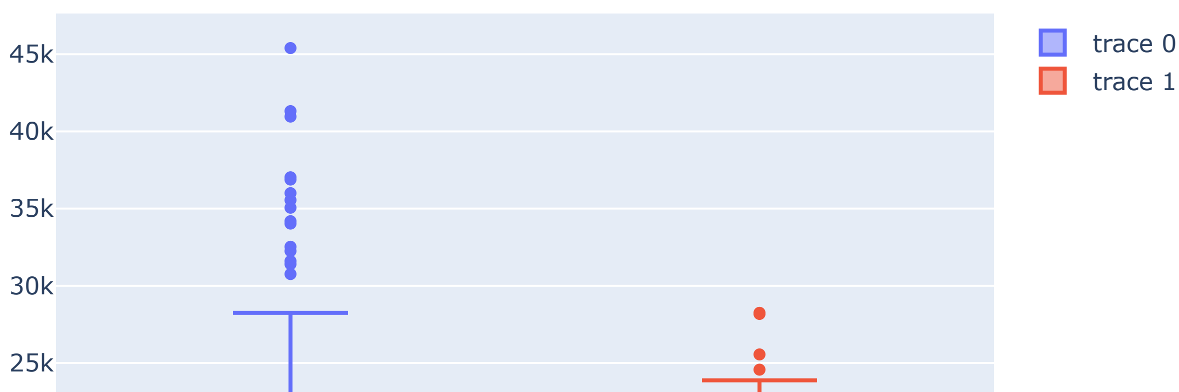


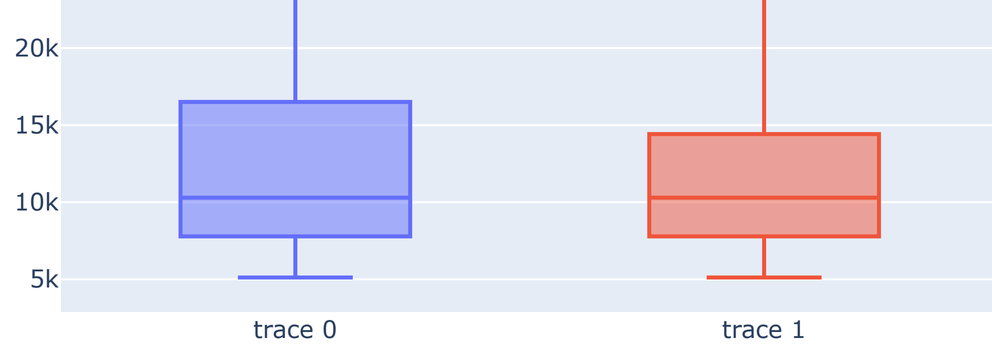

In [25]:
```
df_price.describe()
```

Out[25]:
```
count      205.000000
mean     11638.716222
std       4804.496843
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      14399.000000
max      28248.000000
Name: price, dtype: float64
```

- Comparison

In [26]:
```
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Box(y=ilk_price))
fig.add_trace(go.Box(y=df.price))
fig.show()
```





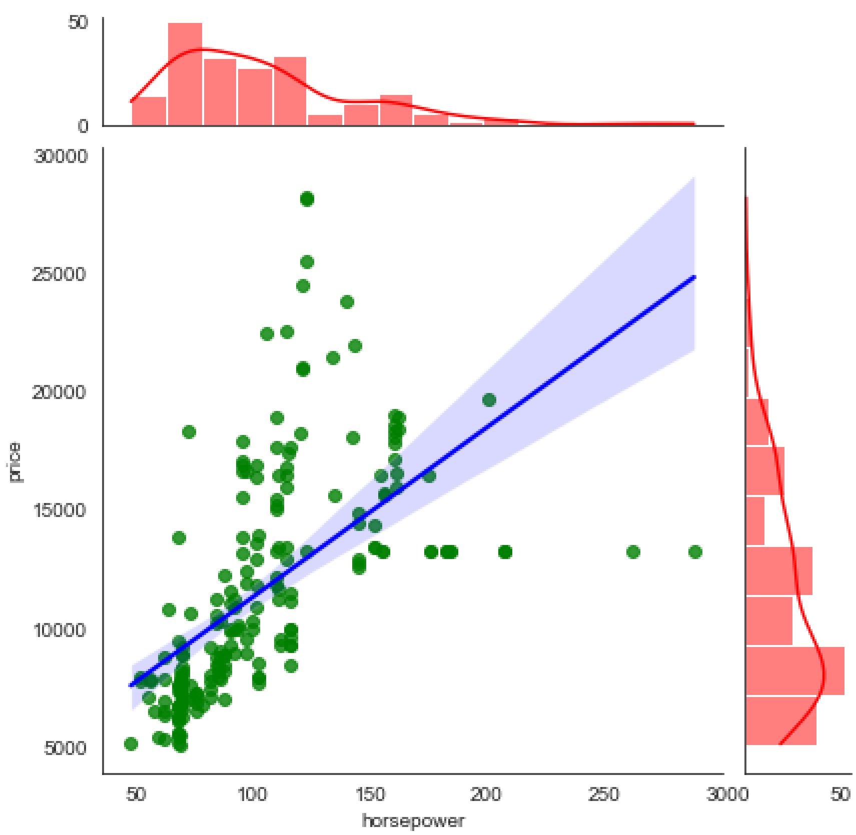

.

Regression Analysis 1) Simple linear regression 1.1 Simple linear regression with statsmodel 1.2 Simple linear regression with sklearn

1) SİMPLE LİNEAR REGRESSİON

In [27]:
```
sns.set_style("white")
sns.jointplot(x="horsepower",y="price",
                data=df,
                kind="reg",
                marginal_kws={"color":"r"},
                line_kws={"color":"b"},
                color="g",
                marginal_ticks=True)
```

Out[27]: <seaborn.axisgrid.JointGrid at 0x19628acd400>




1.1 SİMPLE LİNEAR REGRESSİON WİTH STATSMODEL

In [28]:
```
x = yeni_df.horsepower
y_gercek =df.price
```

In [29]:
```
import statsmodels.api as sm
x = sm.add_constant(x)
model = sm.OLS(y_gercek,x).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.351
Model:                            OLS   Adj. R-squared:                  0.348
Method:                 Least Squares   F-statistic:                     109.7
Date:                Sat, 04 Feb 2023   Prob (F-statistic):           8.38e-21
Time:                        11:38:42   Log-Likelihood:                -1983.9
No. Observations:                 205   AIC:                             3972.
Df Residuals:                     203   BIC:                             3979.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4145.6695    764.951      5.420      0.000    2637.401    5653.938
horsepower    71.9675      6.870     10.475      0.000      58.421      85.514
==============================================================================
Omnibus:                       54.973   Durbin-Watson:                   0.550
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              116.619
Skew:                           1.259   Prob(JB):                     4.75e-26
Kurtosis:                       5.705   Cond. No.                         314.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

• Preliminary information

• If RC~0, the distribution is symmetrical with respect to the mean, • If RC < 0, the distribution is skewed to the left, that is, it tends to (-) direction, • If RC > 0, the distribution is skewed to the right, that is, it tends to the (+) direction.

• Mesokurtic : The Kurtosis Coefficient of the standard normal distribution is equal to 3. In this case The distribution of the data means that it is in accordance with the SND. • Lepokurtic: means BK > 3. The distribution is sharper than SND. • Playkurtic: means BK < 3. The distribution is flatter than SND.

• TABLE INTERPRETATİON

The model is significant since the probe (F-statistic) is 8.38e-21<0.05.

The change in horsepower can explain 35% of the change in price.

The horsepower p value is less than 0.05, so it is significant for the model.

Since the probe (Omnibus) value is less than 0.05, the errors are not normally distributed.

Since the skew value is 1.259 (RC>0), it is skewed to the right.

Since Kurtosis is 5.705 (BK>3), the distribution is sharper than the standard normal distribution.

We find the model equation as y=4145.6695+71.9675*horsepower.

• Examination of error values

In [30]:
```
y_pred_stat = model.predict(x)
karsilastirma_statsmodel = pd.DataFrame({'Gercek_Degerler': y_gercek, 'Tahmin_Degerler':
karsilastirma_statsmodel["tahminleme_hatalari"] = karsilastirma_statsmodel.Gercek_Degerl
karsilastirma_statsmodel["hata_kareler"]=karsilastirma_statsmodel["tahminleme_hatalari"]
print(karsilastirma_statsmodel)
```

```
     Gercek_Degerler  Tahmin_Degerler  tahminleme_hatalari  hata_kareler
0            13495.0     12134.063343          1360.936657  1.852149e+06
1            16500.0     12134.063343          4365.936657  1.906140e+07
2            16500.0     15228.666371          1271.333629  1.616289e+06
3            13950.0     11486.355733          2463.644267  6.069543e+06
4            17450.0     12421.933392          5028.066608  2.528145e+07
..               ...              ...                  ...           ...
200          16845.0     12349.965880          4495.034120  2.020533e+07
201          19045.0     15660.471445          3384.528555  1.145503e+07
202          21485.0     13789.316126          7695.683874  5.922355e+07
203          22470.0     11774.225782         10695.774218  1.143996e+08
204          22625.0     12349.965880         10275.034120  1.055763e+08

[205 rows x 4 columns]
```

In [31]:
```
print("Mean Squared Error      : ",karsilastirma_statsmodel.hata_kareler.mean())
print("Root Mean Squared Error : ",math.sqrt(karsilastirma_statsmodel.hata_kareler.mean(
```

```
Mean Squared Error      :  14910976.776812812
Root Mean Squared Error :  3861.473394549393
```

.

## 1.2 SİMPLE LİNEAR REGRESSİON WİTH SKLEARN

In [32]:
```
from sklearn.linear_model import LinearRegression
from sklearn import metrics

lr = LinearRegression()
lr.fit(x,y_gercek)

y_pred_sklearn = lr.predict(x)
karsilastirma_sklearn = pd.DataFrame({'Gercek_Degerler': y_gercek, 'Tahmin_Degerler': y_
karsilastirma_sklearn["tahminleme_hatalari"] = karsilastirma_sklearn.Gercek_Degerler - k
karsilastirma_sklearn["hata_kareler"]=karsilastirma_sklearn["tahminleme_hatalari"]**2
karsilastirma_sklearn=pd.DataFrame(karsilastirma_sklearn)
karsilastirma_sklearn
```

Out[32]:

| | Gercek_Degerler | Tahmin_Degerler | tahminleme_hatalari | hata_kareler |
|---|---|---|---|---|
| **0** | 13495.0 | 12134.063343 | 1360.936657 | 1.852149e+06 |
| **1** | 16500.0 | 12134.063343 | 4365.936657 | 1.906140e+07 |
| **2** | 16500.0 | 15228.666371 | 1271.333629 | 1.616289e+06 |
| **3** | 13950.0 | 11486.355733 | 2463.644267 | 6.069543e+06 |
| **4** | 17450.0 | 12421.933392 | 5028.066608 | 2.528145e+07 |
| **...** | ... | ... | ... | ... |
| **200** | 16845.0 | 12349.965880 | 4495.034120 | 2.020533e+07 |
| **201** | 19045.0 | 15660.471445 | 3384.528555 | 1.145503e+07 |
| **202** | 21485.0 | 13789.316126 | 7695.683874 | 5.922355e+07 |
| **203** | 22470.0 | 11774.225782 | 10695.774218 | 1.143996e+08 |
| **204** | 22625.0 | 12349.965880 | 10275.034120 | 1.055763e+08 |

205 rows × 4 columns

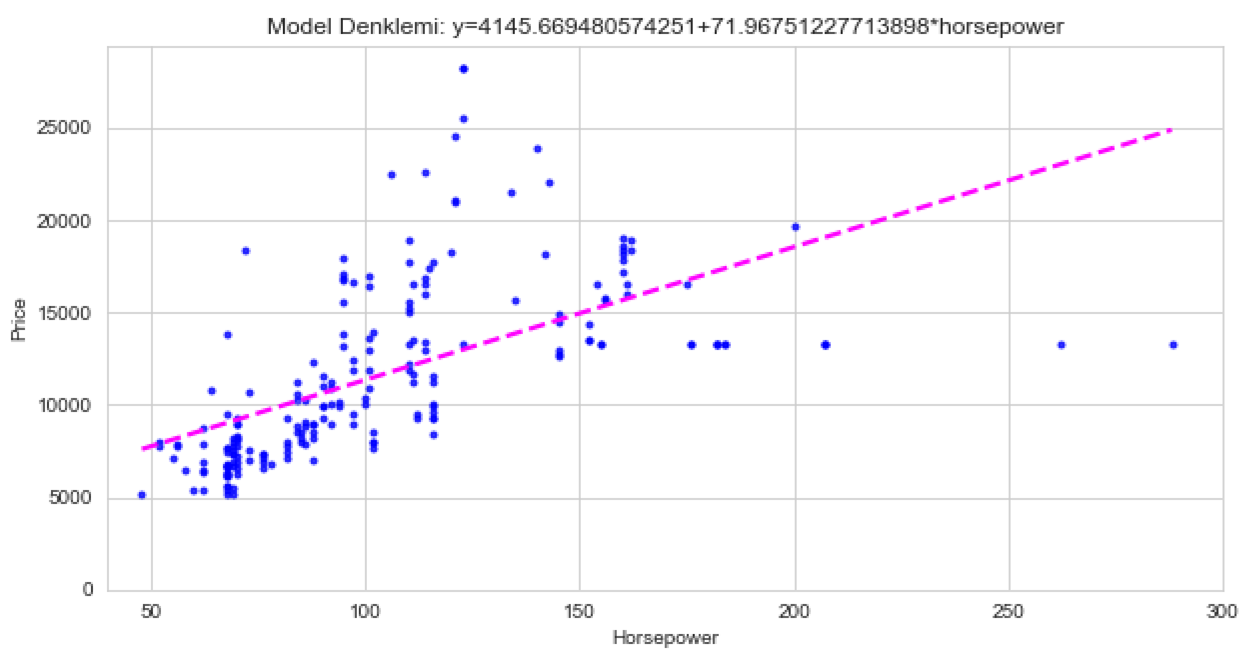
In [33]:
```
print("Mean Squared Error      : ",karsilastirma_sklearn.hata_kareler.mean())
print("Root Mean Squared Error : ",math.sqrt(karsilastirma_sklearn.hata_kareler.mean()))
```

```
Mean Squared Error      :  14910976.776812803
Root Mean Squared Error :  3861.4733945493917
```

In [34]:
```
print("coefficients : ",lr.coef_[1])  # katsayı
print("intercept    :",lr.intercept_) # Kesim noktası
print("r2 score     :",metrics.r2_score(y_gercek,y_pred_sklearn))
print('Mean Squared Error     :', metrics.mean_squared_error(y_gercek, y_pred_sklearn))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_gercek, y_pred_sk
print('mean_absolute_error    :', metrics.mean_absolute_error(y_gercek, y_pred_sklearn))
```

```
coefficients :  71.96751227713898
intercept    : 4145.669480574251
r2 score     : 0.35086658933701964
Mean Squared Error     : 14910976.776812814
Root Mean Squared Error: 3861.473394549393
mean_absolute_error    : 2826.191574446722
```

In [35]:
```
sns.set_style("whitegrid")
plt.figure(figsize=(10,5))
g = sns.regplot(x=df["horsepower"], y=df["price"], scatter_kws={'color': 'b', 's': 9},
                ci=False, line_kws={"color":"fuchsia",'linestyle': '--'})
g.set_title(f"Model Denklemi: y=4145.669480574251+71.96751227713898*horsepower")
g.set_ylabel("Price")
g.set_xlabel("Horsepower")
plt.xlim(40, 300)
plt.ylim(bottom=0)
plt.show()
```




In [42]:
```
#Y=aX+b
Y=lr.coef_*150+lr.intercept_
Y
```

Out[42]:
```
array([ 4145.66948057, 14940.79632215])
```

In [ ]:

In [ ]: