

```
In [62]: import json
import psycopg2
import urllib.parse as up
import pandas as pd
import re
import urllib3
from datetime import date
import os
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
```

```
In [63]: #Connect to Database and convert relevant data into a DataFrame
#Connect to Database
parentDir = os.path.dirname(os.getcwd())
cred = json.load(open(parentDir+'/PSQLCredentials.json'))
url = up.urlparse(cred[1]['connection'])
conn = psycopg2.connect(database=url.path[1:], user=url.username, password=url.password, host=url.hostname, port=url.port )
cursor = conn.cursor()
cursor.execute("select * from (select caryear, mileage, model, ownercount, accidentcount, price, Case When transmission like '%Manual%' Then 'Manual' When transmission like '%Automatic%' Then 'Automatic' Else 'Unknown' End transm
conn.commit()
df = pd.DataFrame(cursor.fetchall())
cursor.close()
conn.close()
#Add column names to dataframe
df = df.rename(columns = {0:'Year', 1:'Mileage', 2: 'Maker', 3:'OwnerCount', 4:'AccidentCount', 5:'Price', 6:'Transmission'})
#Convert all numerical data into float
df[['Price']] = df[['Price']].apply(pd.to_numeric)
#Create dummy data for categorical data types
df[['Maker']] = df[['Maker']].map({'Toyota 86': 0, 'Subaru BRZ': 1})
df[['Transmission']] = df[['Transmission']].map({'Manual': 0, 'Automatic': 1})
```

Steps for creating multiple regression model(least squares): 1) Selecting variables 2) Selecting form of multiple regression equation 3) Checking for violations of underlying assumptions

1) Selecting variables:

- checking for collinearity

```
In [65]: coordf = df.drop(columns=["Price"])
coordf.corr()
```

Out[65]:

	Year	Mileage	Maker	OwnerCount	AccidentCount	Transmission
Year	1.000000	-0.767859	-0.186477	-0.501957	-0.013443	0.004221
Mileage	-0.767859	1.000000	0.131412	0.487455	0.066658	-0.043709
Maker	-0.186477	0.131412	1.000000	0.041309	0.151918	-0.428189
OwnerCount	-0.501957	0.487455	0.041309	1.000000	0.122678	-0.016429
AccidentCount	-0.013443	0.066658	0.151918	0.122678	1.000000	-0.085728
Transmission	0.004221	-0.043709	-0.428189	-0.016429	-0.085728	1.000000

2) Selecting and creating best-fitting model using forwards selection Credits: [https://planspace.org/20150423-forward\\_selection\\_with\\_statsmodels/](https://planspace.org/20150423-forward_selection_with_statsmodels/)

```
In [66]: import statsmodels.formula.api as smf

def forward_selected(data, response):
    """Linear model designed by forward selection.

    Parameters:
    -----
    data : pandas DataFrame with all possible predictors and response

    response: string, name of response column in data

    Returns:
    -----
    model: an "optimal" fitted statsmodels linear model
           with an intercept
           selected by forward selection
           evaluated by adjusted R-squared
    """
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = 0.0, 0.0
    while remaining and current_score <= best_new_score:
        scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {} + {}".format(response,
                                             *'.join(selected + [candidate]))

            score = smf.ols(formula, data).fit().rsquared_adj
            scores_with_candidates.append((score, candidate))
        scores_with_candidates.sort()
        best_new_score, best_candidate = scores_with_candidates.pop()
        if current_score < best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
        formula = "{} ~ {} + {}".format(response,
                                         *'.join(selected))
        model = smf.ols(formula, data).fit()
    return model

model = forward_selected(df, 'Price')
print(model.model.formula)
print(model.rsquared_adj)
```

Price ~ Year + Mileage + Maker + 1  
0.9515181044241184

```
In [67]: import statsmodels.api as sm
X = df[['Year', 'Mileage', 'Maker']]
X = sm.add_constant(X)
Y = df[['Price']]
mod = sm.OLS(Y, X)
res = mod.fit()

model = res.params
print(model)
intercept = model[0]
yearCoef = model[1]
mileageCoef = model[2]
makerCoef = model[3]

const      -3.094214e+06
Year        1.549272e+03
Mileage     -9.334981e-02
Maker        4.013720e+02
dtype: float64
```

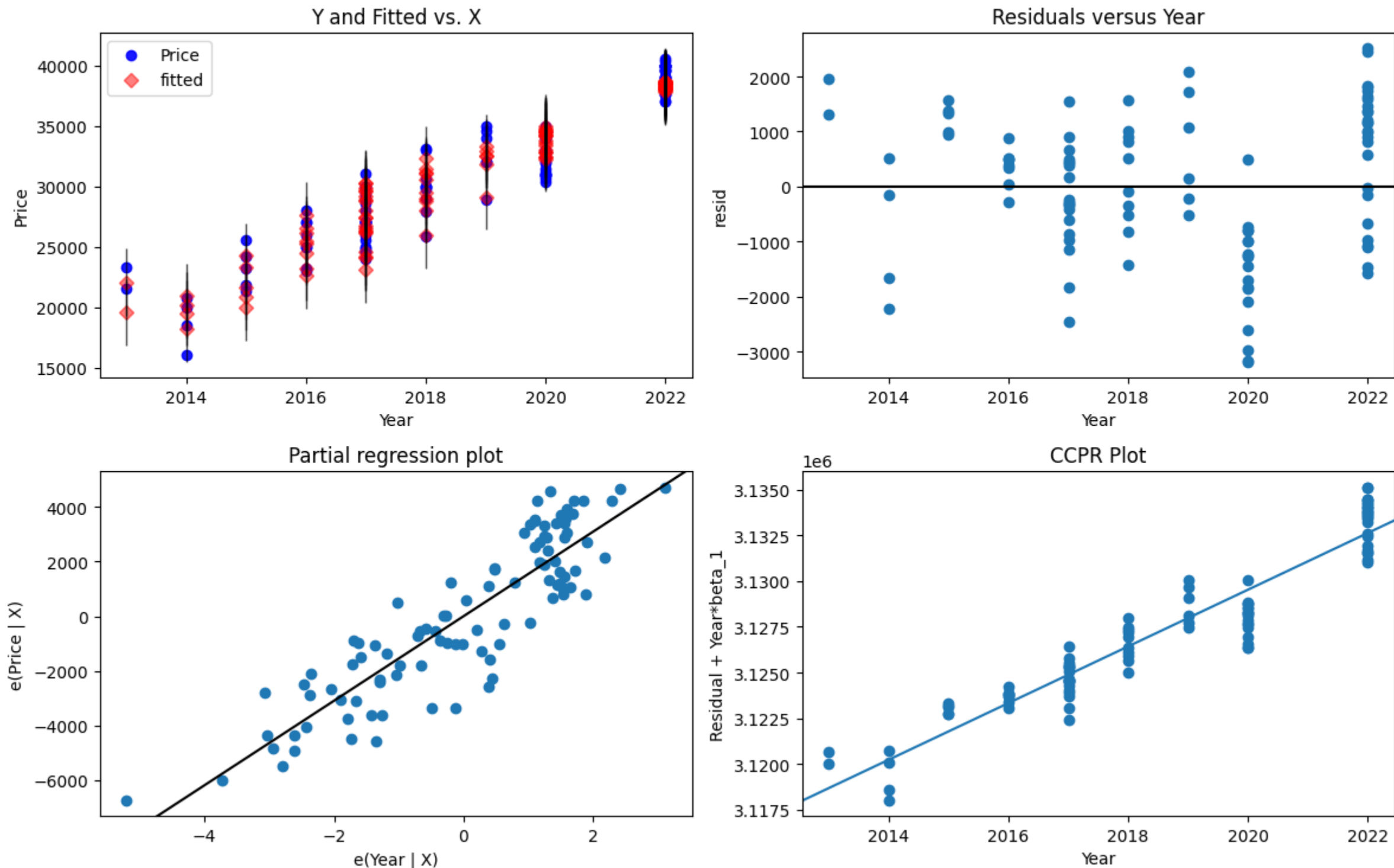
3) Performing Regression Assumptions:

- testing for heteroscedasticity of residuals
- checking values are close to predicted regression values

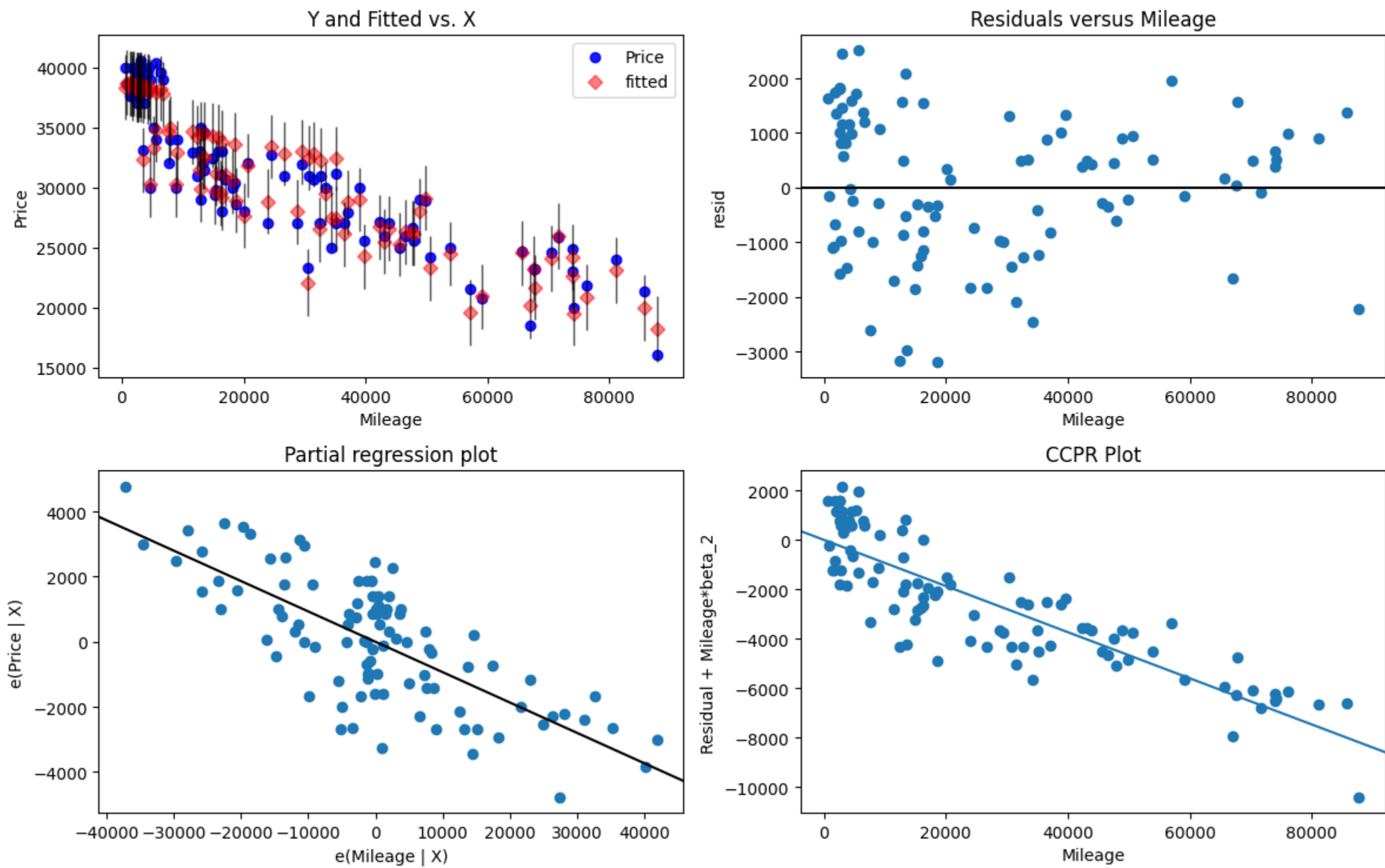
```
In [68]: fig = plt.figure(figsize = (12,8))
fig1 = plt.figure(figsize = (12,8))
fig2 = plt.figure(figsize = (12,8))
fig = sm.graphics.plot_regress_exog(res, 'Year', fig = fig)
fig = sm.graphics.plot_regress_exog(res, 'Mileage', fig = fig1)
fig = sm.graphics.plot_regress_exog(res, 'Maker', fig = fig2)
```

eval\_env: 1  
eval\_env: 1  
eval\_env: 1

Regression Plots for Year



Regression Plots for Mileage



Regression Plots for Maker

