In [236]:

```
import pandas as pd
import numpy as np
data=pd.read_excel(r"C:\Users\selcu\Desktop\3. sınıf 1. dönem\Data Analytics END 305E\Ödevl
```

In [237]:

```
data
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

```
Sex, Smoker and Region columns are categorical. So, we should add dummy variables for
them:
Sex column includes 2 categories. So I need to add 1 dummy variable as "male".
Smoker column includes 2 categories. So I need to add 1 dummy variable as "smoker_y".
Region column includes 4 categories. So we need to add 3 dummy variable as "northwest",
"northeast" and "southwest".
```

In [238]:

```
data["female"]=np.where(data["sex"].str.contains("female"), 1, 0)
#This line adds a column for dummy of sex column.
#That column will include 1 for female, 0 for others.
```

In [239]:

```
data["smoker_y"]=np.where(data["smoker"].str.contains("yes"), 1, 0)
#This line adds a column for dummy of smoker column.
#That column will include 1 for smokers, 0 for others.
```

In [240]:

```
data["northwest"]=np.where(data["region"].str.contains("northwest"), 1, 0)
#This line adds a column for first dummy of region column.
#That column will include 1 for northwest, 0 for others.
```

In [241]:

```
data["northeast"]=np.where(data["region"].str.contains("northeast"), 1, 0)
#This line adds a column for second dummy of region column.
#That column will include 1 for northeast, 0 for others.
```

In [242]:

```
data["southwest"]=np.where(data["region"].str.contains("southwest"), 1, 0)
#This line adds a column for last dummy of region column.
#That column will include 1 for southwest, 0 for others.
```

In [243]:

```
data.head()
```

Out[243]:

	age	sex	bmi	children	smoker	region	charges	female	smoker_y	northwest
0	19	female	27.900	0	yes	southwest	16884.92400	1	1	0
1	18	male	33.770	1	no	southeast	1725.55230	0	0	0
2	28	male	33.000	3	no	southeast	4449.46200	0	0	0
3	33	male	22.705	0	no	northwest	21984.47061	0	0	1
4	32	male	28.880	0	no	northwest	3866.85520	0	0	1

In [244]:

```
import statsmodels.api as sm
X = data.loc[:,["age","female","bmi","children","smoker_y","northwest","northeast","southwe
#This line defines independent variables of our model.
```

In [245]:

```
X = sm.add_constant(X) #This line adds a constant.
```

In [246]:

```
y = data["charges"] #This line defines the dependent variable of model.
lm = sm.OLS(y,X)    #This line runs OLS method with X and y.
model = lm.fit()    #This line fits the model.
model.summary()     #Prints the summary.
```

Out[246]:

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.751
Model:	OLS	**Adj. R-squared:**	0.749
Method:	Least Squares	**F-statistic:**	500.8
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	15:58:09	**Log-Likelihood:**	-13548.
No. Observations:	1338	**AIC:**	2.711e+04
Df Residuals:	1329	**BIC:**	2.716e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.31e+04	1090.510	-12.017	0.000	-1.52e+04	-1.1e+04
age	256.8564	11.899	21.587	0.000	233.514	280.199
female	131.3144	332.945	0.394	0.693	-521.842	784.470
bmi	339.1935	28.599	11.860	0.000	283.088	395.298
children	475.5005	137.804	3.451	0.001	205.163	745.838
smoker_y	2.385e+04	413.153	57.723	0.000	2.3e+04	2.47e+04
northwest	682.0581	478.959	1.424	0.155	-257.540	1621.657
northeast	1035.0220	478.692	2.162	0.031	95.947	1974.097
southwest	74.9711	470.639	0.159	0.873	-848.305	998.247

Omnibus:	300.366	**Durbin-Watson:**	2.088
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	718.887
Skew:	1.211	**Prob(JB):**	7.86e-157
Kurtosis:	5.651	**Cond. No.**	357.

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

OUTPUT REVIEW:

```
R^2 is 0,751. It means that; %75,1 of dependent variable Y, can be explained by
independent variables.
```

Adjusted R^2 is 0,75. This is the corrected R^2 by using degree of freedom values.

F-statistic is 572,7 and Prob(F-statistic) is 0 so, 0 is less than 0,05 and this means that; at least 1 coeefficient is not equal to 0. And this means; some of the independent variables are significant.

We can't use AIC or BIC measures because we need other models to compare.

We can find our coefficients in the firts column of output part.

We can find t values in the 3rd column of output part. They are calculated by Coef/stdErr.

We can find p values in the 4th column of output. We will use the p values to understand which values are significant for dependent variable.

We can see that; all of the p values except the dummy variables are less than 0,05. So we can understand that; our dependent variables are significant. We would apply backwards stepwise regression and remove the most insignificant variable, if the model included a variable which has a p value larger than 0,05. Shortly, our dependent variables will stay same.

Now, I will find VIF scores to find if there is multicollinearity problem...

In [249]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

print(vif_data)
```

```
     feature        VIF
0      const  43.298082
1        age   1.016822
2     female   1.008900
3        bmi   1.106630
4   children   1.004011
5   smoker_y   1.012074
6  northwest   1.535986
7  northeast   1.531063
8  southwest   1.483083
```

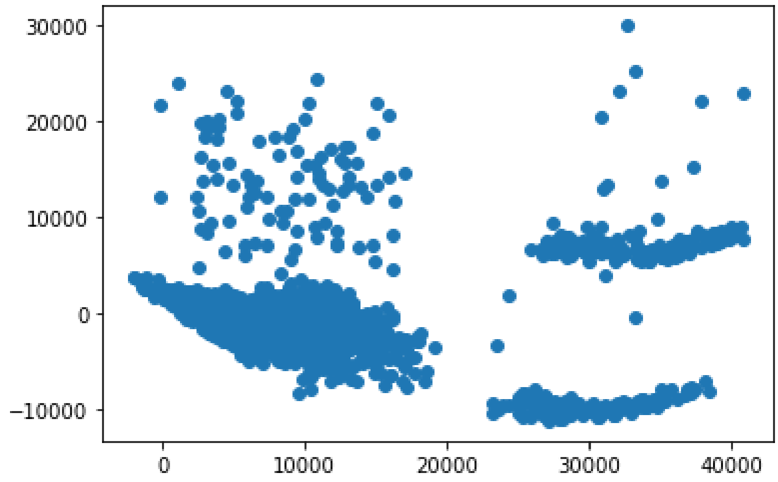
If any VIF score is bigger than 5; we say that there is a problem. But at our model, there is no problem.

In [251]:

```
import matplotlib.pyplot as plt
x=model.fittedvalues
y=model.resid
plt.scatter(x, y)
```

Out[251]:

```
<matplotlib.collections.PathCollection at 0x272a7bcedc0>
```



From our graph's distribution, we can say that; our model has heteroskedasticity problem.
To solve that problem; we can do transformations:

In [259]:

```
X = data.loc[:,["age","female","bmi","children","smoker_y","northwest","northeast","southwe
y = np.sqrt(data["charges"]) #This line calculates root of dependent variable.
X= sm.add_constant(X)
lm = sm.OLS(y, X)
model = lm.fit()
model.summary()
```

Out[259]:

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.780
Model:	OLS	**Adj. R-squared:**	0.778
Method:	Least Squares	**F-statistic:**	587.4
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	16:26:51	**Log-Likelihood:**	-6059.7
No. Observations:	1338	**AIC:**	1.214e+04
Df Residuals:	1329	**BIC:**	1.218e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-7.1633	4.047	-1.770	0.077	-15.102	0.776
age	1.3983	0.044	31.666	0.000	1.312	1.485
female	1.9123	1.236	1.548	0.122	-0.512	4.336
bmi	1.0300	0.106	9.704	0.000	0.822	1.238
children	3.2743	0.511	6.402	0.000	2.271	4.278
smoker_y	90.8717	1.533	59.267	0.000	87.864	93.880
northwest	3.5178	1.777	1.979	0.048	0.031	7.005
northeast	5.8945	1.776	3.318	0.001	2.409	9.379
southwest	0.6930	1.747	0.397	0.692	-2.733	4.119

Omnibus:	480.229	**Durbin-Watson:**	2.090
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1605.806
Skew:	1.783	**Prob(JB):**	0.00
Kurtosis:	7.010	**Cond. No.**	357.

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

Output review:

```
R^2, Adjusted R^2, F Statistics values are increased. AIC and BIC values are decreased.
They all mean that; we had a better model.
```

```
Gender, region variables became more significant; we can understand that from their p
value increase.
```

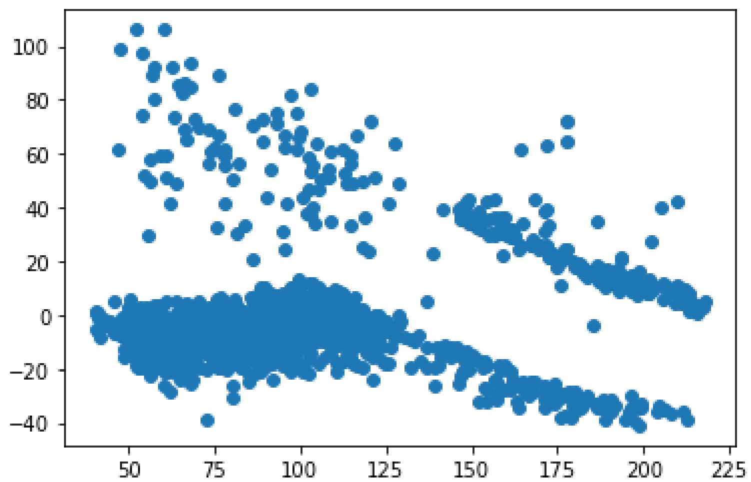
```
We can't be sure without the neccesary tests but; we can say that, maybe we had more
homoscedastic model from the new graph:
```

In [263]:

```
x=model.fittedvalues
y=model.resid
plt.scatter(x, y)
```

Out[263]:

```
<matplotlib.collections.PathCollection at 0x272a8232400>
```



LASSO Regression

```
Actually, LASSO Regression model tries to make coefficients 0 to simpilfy the model. But
my model has already have low number of independent variables and it is a simple model.
So, LASSO is not neccesary but I will use it anyway...
```

In [264]:

```
from sklearn.linear_model import Lasso
X = data.loc[:,["age","female","bmi","children","smoker_y","northwest","northeast","southwe
y = np.sqrt(data["charges"])
X= sm.add_constant(X)                                    # y: sqrt
lasso_model = Lasso(alpha = 0.2).fit(X, y)               #model for alpha=0,2
```

```
Alpha in python is the same thing with lambda. I choose a random alpha(lambda) value as
0,2. Now, let's print the coefficients for alpha = 0,2.
```

In [299]:

```
lasso_model
lasso_model.coef_                          #coefficients for alpha 0,2
```

Out[299]:

```
array([    0.        ,   256.88274201,   125.09878315,   338.66143934,
         474.74706122, 23837.70377167,   658.20696358,  1011.32590064,
          51.82380134])
```

As we can see; these "LASSO coefficients" and "classical OLS method's coefficients" are close, because our alpha value is small. (Already, if alpha was equal to 0, LASSO and OLS would give the same results.)

Now, I will find the optimum alpha value which minimizes the MSE(mean square error):

In [348]:

```
from sklearn.linear_model import LassoCV
lasso_cv_model = LassoCV(alphas = None, cv = 10,
                         max_iter = 10000,
                         normalize = True)
lasso_cv_model.fit(X,y)
lasso_cv_model.alpha_          # finding optimum alpha
```

Out[348]:

```
9.753585560753975e-05
```

We see that; optimum alpha is 9,753585560753975e-05.

In [300]:

```
lasso_tuned = Lasso(alpha = lasso_cv_model.alpha_)
lasso_tuned.fit(X, y)
y_pred = lasso_tuned.predict(X)
```

In [301]:

```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y, y_pred))
```

Out[301]:

```
6041.689900585609
```

As we can see; 6041,689900585609 is the best MSE value.

In [302]:

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha = 7.65e-05).fit(X, y)
```

In [303]:

```
lasso_model
lasso_model.coef_
```

Out[303]:

```
array([    0.        ,   256.85635399,   131.31401739,   339.19342431,
         475.5005037 , 23848.53394591,   682.05683691,  1035.02074503,
          74.96978408])
```

Again, we can see that; coefficients got too close to OLS method's coefficients because alpha is too small.

Now, I will use LASSO with Y is "charges"; not root of "charges":

In [266]:

```
X = data.loc[:,["age","female","bmi","children","smoker_y","northwest","northeast","southwe
y = data["charges"]
X= sm.add_constant(X)
lasso_model = Lasso(alpha = 0.3).fit(X, y) #used a random alpha value
```

I choose a random alpha(lambda) value as 0,3. Now, I will try to find the optimum alpha level which minimizes the MSE(mean square error):

In [269]:

```
from sklearn.linear_model import LassoCV
lasso_cv_model = LassoCV(alphas = None, cv = 10,
                         max_iter = 10000,
                         normalize = True)
lasso_cv_model.fit(X,y)
lasso_cv_model.alpha_
```

Out[269]:

```
1.3904018022702285
```

Optimum alpha value is 1,3904018022702285. Now I will use this value to find the coefficients:

In [270]:

```
X = data.loc[:,["age","female","bmi","children","smoker_y","northwest","northeast","southwe
y = data["charges"]
X= sm.add_constant(X)
lasso_model = Lasso(alpha = 1.3904018022702285).fit(X, y) #used optimum alpha value
```

In [272]:

```
lasso_model
lasso_model.coef_  #LASSO coefficients will be printed
```

Out[272]:

```
array([    0.        ,   256.88274201,   125.09878315,   338.66143934,
         474.74706122, 23837.70377167,   658.20696358,  1011.32590064,
          51.82380134])
```

We can see that; these "LASSO coefficients" and "classical OLS method's coefficients" are close, because our alpha value is small.

RIDGE Regression

Now, I will use Ridge Regression. Lets use alpha=0,4. This will result the same as Lasso. I mean, coefficients of Ridge Regression and OLS Method are close, because alpha is small...

In [276]:

```
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha = 0.4).fit(X, y)     #alfa is 0,4.
ridge_model.coef_
```

Out[276]:

```
array([    0.        ,   256.83331579,   128.47249004,   339.07923251,
         475.5691241 , 23804.10340458,   675.69007019,  1029.45703833,
          69.49749674])
```

I need to find the optimum alpha(alpha is the same thing with lambda) level for Ridge regression.

To find the best alpha value; we will try different alpha values and check their MSE results.

In [280]:

```
from sklearn.linear_model import RidgeCV
MSE = 10**np.linspace(10,-2,100)*0.5
```

In [289]:

```
MSE
```

Out[289]:

```
array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+07, 1.07721735e+07, 8.14875417e+06,
       6.16423370e+06, 4.66301673e+06, 3.52740116e+06, 2.66834962e+06,
       2.01850863e+06, 1.52692775e+06, 1.15506485e+06, 8.73764200e+05,
       6.60970574e+05, 5.00000000e+05, 3.78231664e+05, 2.86118383e+05,
       2.16438064e+05, 1.63727458e+05, 1.23853818e+05, 9.36908711e+04,
       7.08737081e+04, 5.36133611e+04, 4.05565415e+04, 3.06795364e+04,
       2.32079442e+04, 1.75559587e+04, 1.32804389e+04, 1.00461650e+04,
       7.59955541e+03, 5.74878498e+03, 4.34874501e+03, 3.28966612e+03,
       2.48851178e+03, 1.88246790e+03, 1.42401793e+03, 1.07721735e+03,
       8.14875417e+02, 6.16423370e+02, 4.66301673e+02, 3.52740116e+02,
       2.66834962e+02, 2.01850863e+02, 1.52692775e+02, 1.15506485e+02,
       8.73764200e+01, 6.60970574e+01, 5.00000000e+01, 3.78231664e+01,
       2.86118383e+01, 2.16438064e+01, 1.63727458e+01, 1.23853818e+01,
       9.36908711e+00, 7.08737081e+00, 5.36133611e+00, 4.05565415e+00,
       3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
       1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
       3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
       1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
       3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
       1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03])
```

Now, I need to get the alpha which resulted to the minimum MSE:

In [290]:

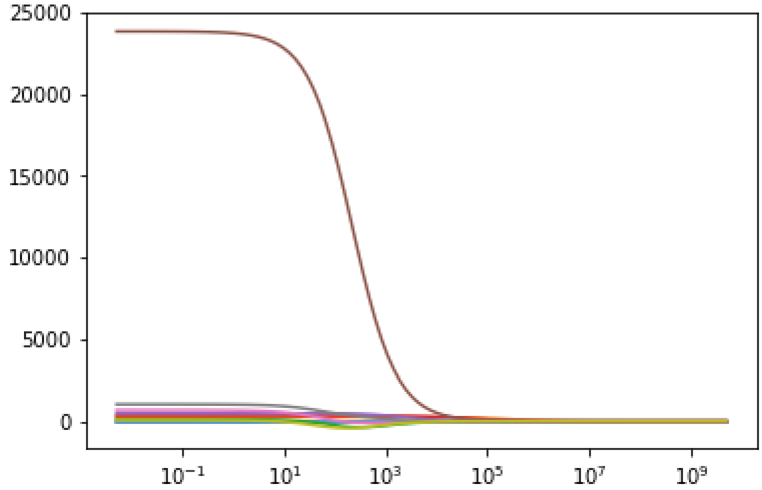
```
ridge_model = Ridge()
katsayilar = []
```

In [291]:

```
for i in MSE:
    ridge_model.set_params(alpha = i)
    ridge_model.fit(X, y)
    katsayilar.append(ridge_model.coef_)
```

In [292]:

```
ax = plt.gca()
ax.plot(MSE, katsayilar)
ax.set_xscale('log')
```




```
Each color represents a different coefficient vector. When we try to make alpha larger and
larger, the coefficients tend to go to 0.
```

In [293]:

```
from sklearn.linear_model import RidgeCV      #optimum alpha minimizes error
ridge_cv=RidgeCV(alphas=MSE, scoring="neg_mean_squared_error", normalize=True)
```

In [294]:

```
ridge_cv.fit(X,y)
```

Out[294]:

```
RidgeCV(alphas=array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.1643
8064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+0...
       1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
       3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
       1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
       3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
       1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03]),
        normalize=True, scoring='neg_mean_squared_error')
```

In [296]:

```
ridge_cv.alpha_
```

Out[296]:

```
0.005
```

Optimimum alpha which minimizes the error is, 0,005. By using that optimum alpha value, I will run Ridge regression and find the coefficients...

In [297]:

```
ridge_model = Ridge(alpha = ridge_cv.alpha_).fit(X, y)
ridge_model.coef_      #coefficients with optimum alfa
```

Out[297]:

```
array([    0.        ,   256.85606443,   131.27873173,   339.19201744,
         475.50140903, 23847.97810377,   681.97810803,  1034.95208712,
          74.9022224 ])
```

Again, we can see that there is no so much difference between these coefficients and OLS method coefficients.