Fase #2: Construcción de un modelo estadístico base

Elías Garza Valdés A01284041

In [ ]:
```
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
```

Leyendo los datos

In [ ]:
```
df = pd.read_csv(r'../Data/precios_autos.csv')
df.head()
```

Out[ ]:

	symboling	CarName	fueltype	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	...	enginetype	cylir
0	3	alfa-romero giulia	gas	convertible	rwd	front	88.6	168.8	64.1	48.8	...	dohc	
1	3	alfa-romero stelvio	gas	convertible	rwd	front	88.6	168.8	64.1	48.8	...	dohc	
2	1	alfa-romero Quadrifoglio	gas	hatchback	rwd	front	94.5	171.2	65.5	52.4	...	ohcv	
3	2	audi 100 ls	gas	sedan	fwd	front	99.8	176.6	66.2	54.3	...	ohc	
4	2	audi 100ls	gas	sedan	4wd	front	99.4	176.6	66.4	54.3	...	ohc	

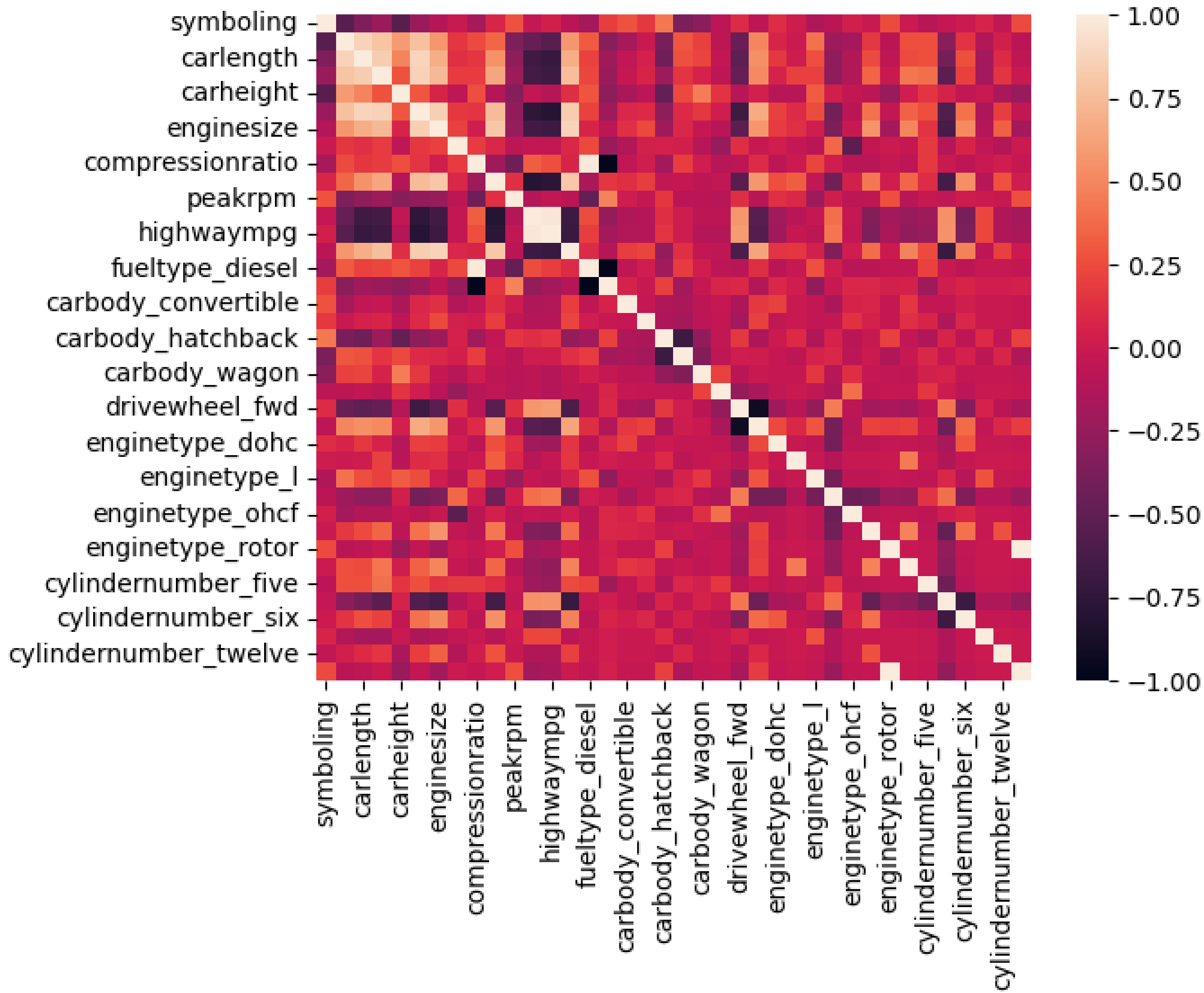
5 rows × 21 columns

In [ ]:

```
df = df.drop(columns = ['CarName', 'enginelocation'])
df = pd.get_dummies(df)
```

In [ ]:

```
sns.heatmap(df.corr())
plt.show()
```



Queremos ver que variables son importantes al momento de decidir el precio de un automovil. Por lo tanto, vamos a generar una regresion lineal con las variables para ver si importan al momento de decidir el precio de un automovil.

Ironicamente, la parte más facil será generar el modelo lo cual haremos a continuación. Despues toca verificar el modelo lo cual ya empieza a ser más complejo.

In [ ]:

```
from sklearn.linear_model import LinearRegression, SGDRegressor
model = LinearRegression()
model.fit(df.drop(columns = 'price'), df['price'])
```

Out[ ]:

```
▾ LinearRegression
LinearRegression()
```

In [ ]:

```
[model_stat.intercept_] +  list(model_stat.coef_)
```

Out[ ]:

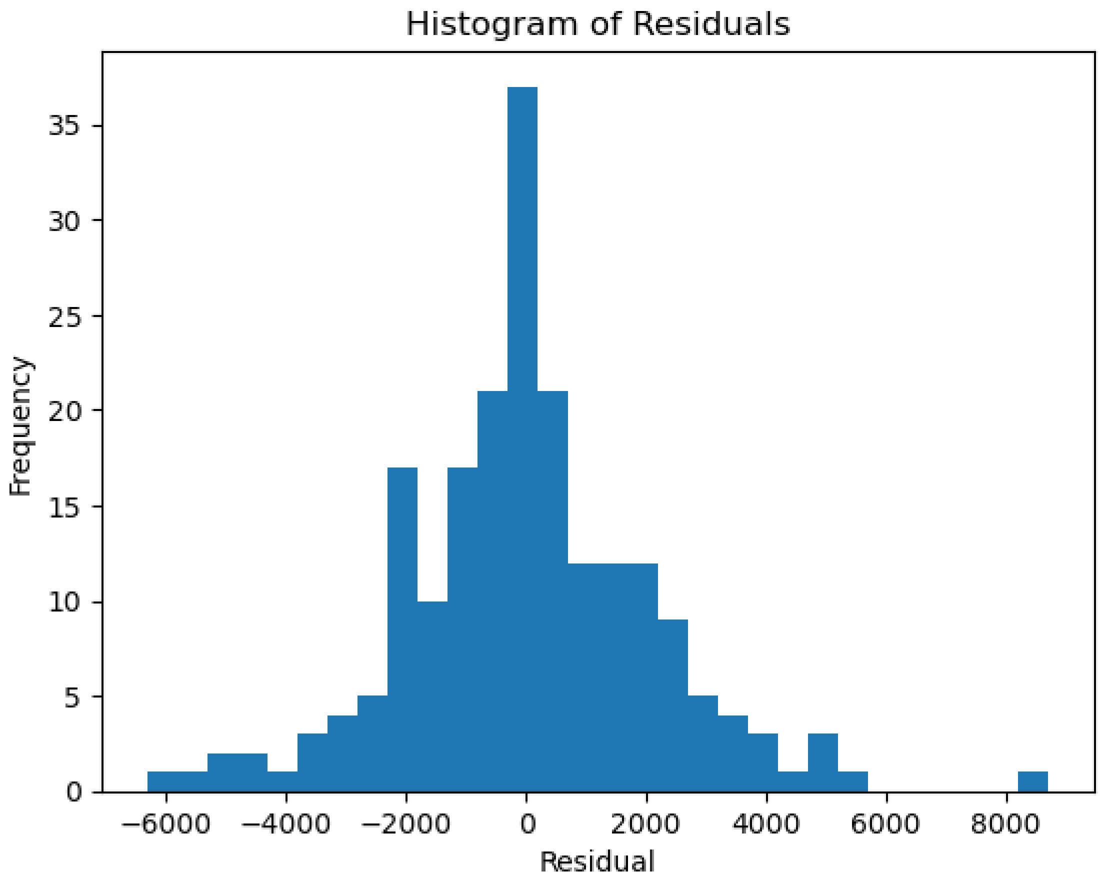
```
[-46298.35495356047,
 -47.150587186348055,
 18.577086953572113,
 -82.2077841717521,
 607.5253710795893,
 186.7355225881091,
 3.515715103081931,
 97.74817546947392,
 -4329.700969603384,
 -243.42650451343053,
 45.58393961205683,
 2.3102647699686853,
 -155.79099865984054,
 225.85765630328902,
 3073.2867269575963,
 -3073.2867269553703,
 2923.609502265532,
 -300.1558218917708,
 -1076.6219209846784,
 -231.47763349034625,
 -1315.3541259112662,
 -193.57422758204194,
 -218.28821859180644,
 411.8624461753853,
 1951.755835812602,
 -13838.964525600866,
 3501.0289830243228,
 5804.986774822784,
 4254.772920402167,
 -4435.1756958764445,
 2761.595707410821,
 8719.804339560087,
 -2902.140337015666,
 -5144.752759285279,
 -214.72897998675813,
 1418.8828720638387,
 -4638.660842755951,
 2761.5957074110174]
```

In [ ]:
```
y_pred = model.predict(df.drop(columns = 'price'))
residuals = df['price'] - y_pred
```

Out[ ]:
```
0      -267.261957
1      2737.738043
2      4922.835020
3      3403.164879
4       -36.918732
          ...
200   -2021.972000
201   -2108.410803
202    2414.195364
203   -1983.482353
204    3425.708313
Name: price, Length: 205, dtype: float64
```

Ahora vamos a revisar si los residuos forman una distribución normal, validando asi el modelo.

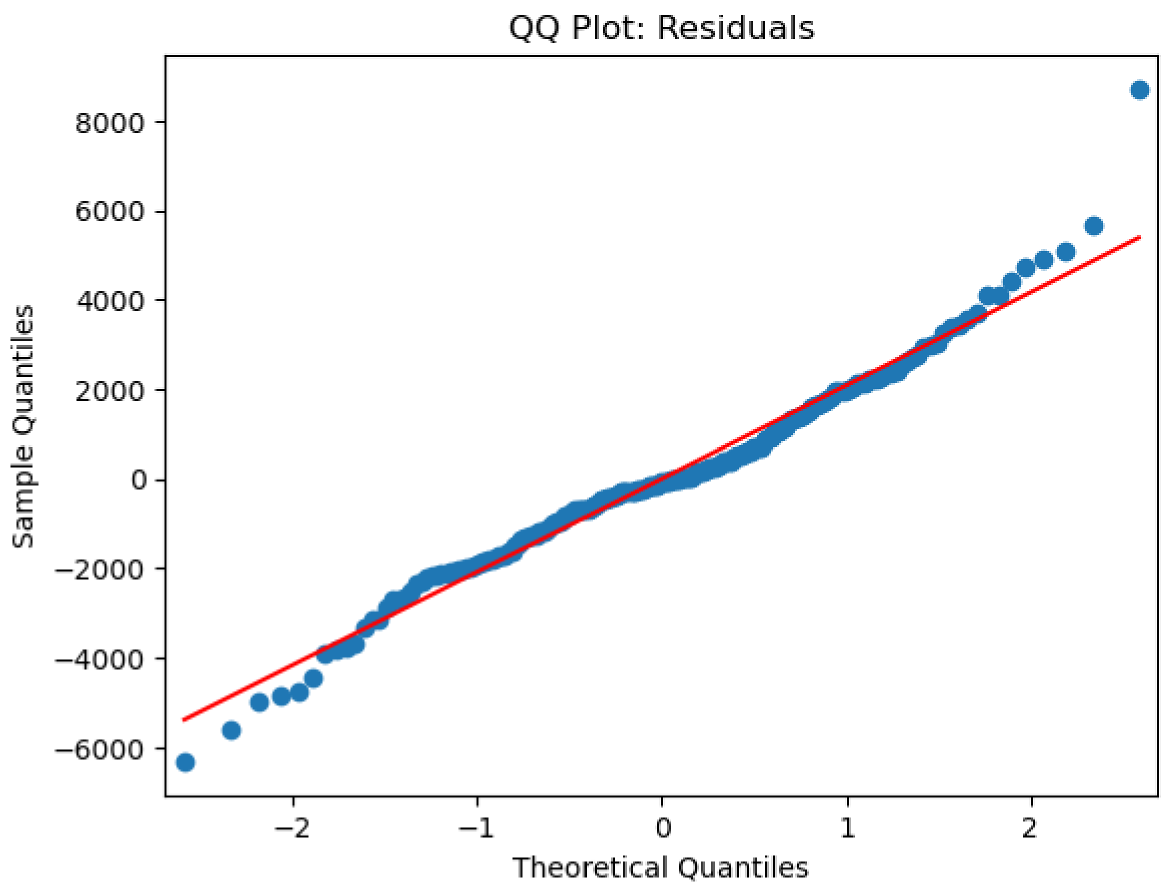
In [ ]:
```
plt.hist(residuals, bins=30)
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()
```



In [ ]:

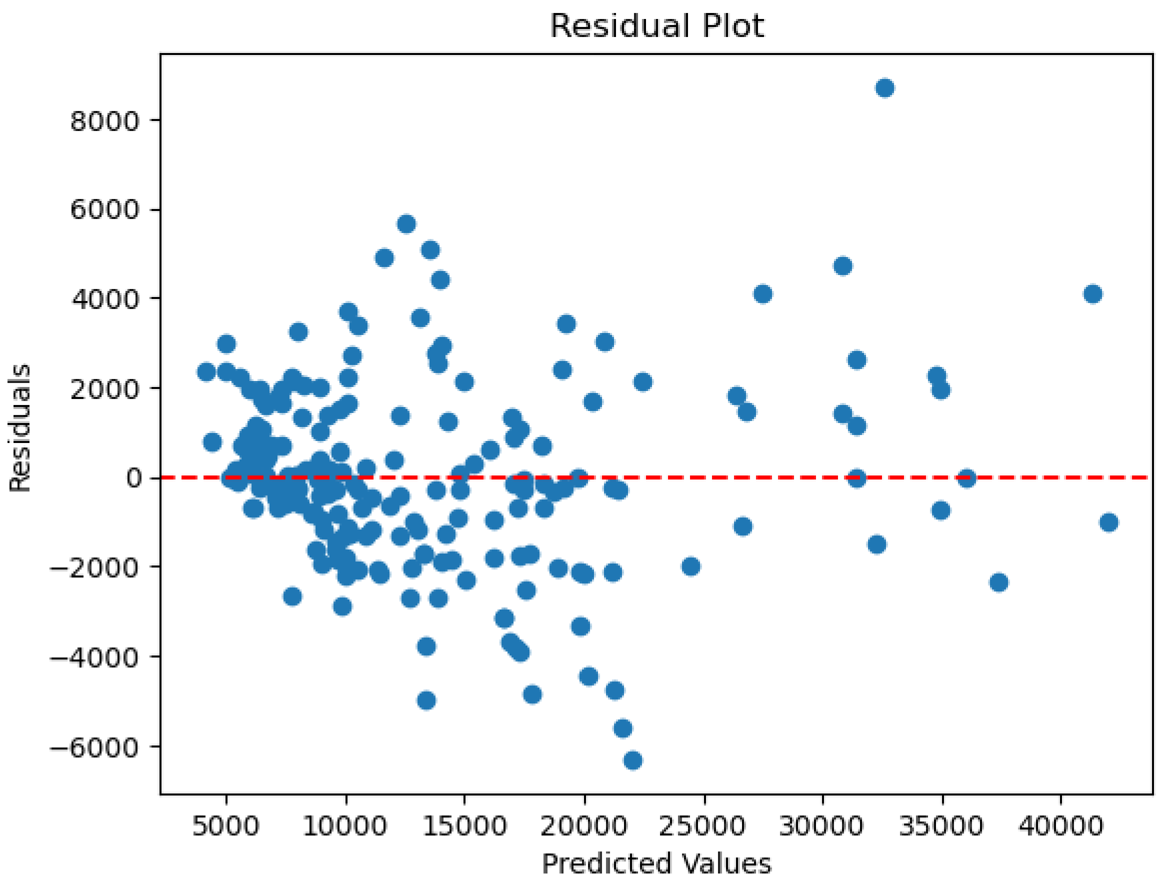
```
sm.qqplot(residuals, line='s')

# Add title and labels
plt.title(f"QQ Plot: Residuals")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")

# Display the plot
plt.show()
result = stats.anderson(residuals)
```

```
print("Is data normally distributed:", result.statistic < result.critical_values[1],
        'via Anderson-Darling test with significance of:' ,100 - result.significance_level[1], '%')

data_standardized = (residuals - np.mean(residuals)) / np.std(residuals)
# Perform the K-S test for normality
ks_statistic, ks_p_value = stats.kstest(data_standardized, 'norm')

print(f"K-S test statistic: {ks_statistic:.4f}")
print(f"P-value: {ks_p_value:.4f}")
```



```
Is data normally distributed: False via Anderson-Darling test with significance of: 90.0 %
K-S test statistic: 0.0732
P-value: 0.2115
```

In [ ]:
```
plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Como podemos ver. A pesar de que sí tenemos un modelo este no cumple con nuestras supocisiones estadisticas para poder considerarlo asi que no lo vamos a usar tal cual. Es por esto que vamos a intentar hacer algo diferente.

Primero, calcularemos la distancia de mahalanobis de nuestro dataframe.

In [ ]:
```
from scipy.spatial import distance
from scipy.stats import chi2

mahalanobis_distances = df.apply(lambda row: distance.mahalanobis(row, df.mean(),
np.linalg.inv(df.cov())), axis=1)
```

Vamos a hacer un PCA y de las combinaciones lineales vemos que variables son lás más importantes

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=15)

pca.fit(df.drop(columns = 'price'))
components = pd.DataFrame(pca.components_).transpose()
components
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.000669	0.000261	0.012177	0.012530	0.045058	-0.061455	-0.176495	0.086122	-0.127951	-0.549911	0.352773	-0.666671	0.135328
1	0.008079	0.003158	-0.061972	-0.064260	-0.317848	0.221399	0.536870	-0.644318	-0.271120	-0.076157	-0.109444	-0.194331	-0.009480
2	0.017444	0.010535	-0.073878	-0.033185	-0.820849	0.357982	-0.278266	0.318156	-0.044632	0.079534	0.029489	-0.031037	-0.026075
3	0.002880	0.002107	-0.001934	-0.004643	-0.053424	0.061316	0.010688	-0.102158	-0.222095	-0.417149	0.522741	0.631076	0.155907
4	0.001629	-0.000485	-0.039725	-0.014597	-0.126796	0.110067	0.208333	-0.110122	0.909425	-0.234547	0.136389	-0.011378	0.019505
5	0.811765	0.575871	-0.090852	0.004803	0.031548	-0.005649	0.005356	0.008526	0.000025	-0.000097	-0.001138	0.000008	0.000062
6	0.055989	0.037741	0.684410	-0.722002	-0.047499	-0.042849	-0.027351	-0.014741	0.020134	-0.006495	-0.005723	0.000367	-0.002361
7	0.000089	0.000043	0.000227	-0.004549	0.004506	0.005383	-0.009297	-0.009917	-0.030621	-0.034321	-0.024819	0.018454	-0.264452
8	0.002284	-0.002733	-0.042143	-0.039697	0.210044	0.429859	-0.675156	-0.534955	0.082703	0.018217	-0.066778	0.004715	-0.036066
9	0.035111	0.061255	0.713097	0.673494	-0.038115	0.169493	0.035733	-0.029819	0.006887	0.013223	0.009176	-0.011272	-0.003570
10	-0.579827	0.814163	-0.020735	-0.022311	0.002151	0.002381	-0.000509	0.000144	0.000382	0.000020	-0.000131	-0.000052	-0.000096
11	-0.005953	-0.009935	-0.032401	-0.104018	0.281951	0.515916	0.244632	0.161465	-0.028097	0.470137	0.548152	-0.143598	-0.001187
12	-0.006983	-0.010037	-0.020256	-0.079634	0.272220	0.563947	0.188071	0.365404	-0.072382	-0.440750	-0.462318	0.077170	0.017221
13	0.000208	-0.000198	-0.003330	-0.002228	0.015309	0.029071	-0.048022	-0.042334	0.004789	-0.000049	-0.004455	0.000827	-0.019966
14	-0.000208	0.000198	0.003330	0.002228	-0.015309	-0.029071	0.048022	0.042334	-0.004789	0.000049	0.004455	-0.000827	0.019966
15	0.000017	0.000028	0.000688	-0.000200	0.005986	-0.008034	-0.011837	0.011491	0.007452	-0.004562	0.008992	-0.017619	0.057061
16	0.000033	0.000016	0.001986	-0.001186	0.000051	-0.000075	-0.010126	-0.004929	0.006596	-0.012725	0.004546	-0.032349	0.007302
17	-0.000247	-0.000065	0.002973	0.006877	0.020456	-0.018739	0.011961	-0.010107	-0.086726	-0.097010	0.098707	-0.063294	-0.473771
18	0.000103	-0.000009	-0.001553	-0.006086	-0.023490	0.029242	-0.006696	-0.014873	0.002501	0.076911	-0.130041	0.113041	0.590034
19	0.000095	0.000030	-0.004094	0.000594	-0.003003	-0.002394	0.016698	0.018418	0.070177	0.037386	0.017795	0.000221	-0.180626
20	0.000023	-0.000035	-0.000996	0.002649	0.003467	-0.012092	0.004362	0.004275	0.032278	0.028418	0.026617	0.022432	0.058751
21	-0.000490	-0.000420	0.000379	-0.005472	-0.009503	0.011252	-0.002657	0.018349	0.005231	-0.080952	0.056100	0.117152	-0.248382
22	0.000467	0.000455	0.000617	0.002823	0.006036	0.000840	-0.001706	-0.022624	-0.037509	0.052534	-0.082718	-0.139584	0.189631
23	0.000037	0.000146	0.000517	0.000996	0.000205	0.001821	-0.004869	0.013121	-0.009133	0.048280	-0.000569	-0.090876	0.021390
24	0.000005	0.000027	0.000606	0.001469	0.001540	0.003917	0.003339	-0.001256	-0.001633	-0.012408	0.001150	0.017234	0.002029

	0	1	2	3	4	5	6	7	8	9	10	11	12
25	0.000122	-0.000014	-0.003349	0.000925	0.002401	0.004336	0.017357	-0.008535	-0.007193	-0.017344	0.001120	-0.070078	0.102045
26	-0.000258	-0.000285	-0.001698	-0.004750	-0.007902	0.010283	0.004392	0.007313	0.002243	-0.063907	-0.050618	0.116349	-0.348795
27	-0.000011	-0.000076	0.001839	0.003211	-0.000463	-0.008167	-0.008487	-0.003866	0.028659	0.046121	0.027664	0.008169	0.156366
28	0.000143	0.000134	0.003015	-0.004274	0.003723	-0.007241	0.000132	0.003881	0.006740	0.006922	0.025710	0.033679	0.044372
29	-0.000038	0.000068	-0.000929	0.002423	0.000496	-0.004950	-0.011863	-0.010657	-0.019683	-0.007664	-0.004457	-0.014477	0.022594
30	0.000083	0.000059	0.001696	-0.002621	-0.000048	-0.000346	0.008529	-0.004042	-0.004072	-0.021272	0.039795	0.017129	0.032275
31	0.000083	0.000089	-0.001541	-0.000556	-0.000240	-0.000380	-0.010276	-0.015114	-0.002768	-0.032605	0.050611	0.082785	-0.015581
32	-0.000304	-0.000449	-0.004265	0.001332	-0.006691	0.003211	0.014113	0.022423	0.003262	0.012528	-0.015522	-0.027604	-0.101728
33	0.000170	0.000231	0.004174	-0.000007	0.003845	0.000151	-0.003500	0.005168	0.024327	0.038149	-0.072129	-0.051860	0.049975
34	-0.000013	-0.000017	0.000090	0.000074	0.001835	0.001651	0.005062	0.002140	0.001353	0.000288	0.003281	-0.014737	0.011630
35	0.000019	0.000020	0.000775	-0.000646	0.000802	0.000664	-0.002066	0.000083	-0.002419	0.010576	-0.001579	0.008763	0.000835
36	-0.000038	0.000068	-0.000929	0.002423	0.000496	-0.004950	-0.011863	-0.010657	-0.019683	-0.007664	-0.004457	-0.014477	0.022594

Los valores de la varianza explicada son:

In [ ]:
```
values = list(pca.singular_values_/pca.singular_values_.sum())
values
```

Out[ ]:
```
[0.5363591633343965,
 0.40274063385100417,
 0.02654605182468482,
 0.013400241162347304,
 0.005872668535607486,
 0.004675202996194573,
 0.0025194770616329453,
 0.002242795816194395,
 0.0016250421968085336,
 0.0009694287606541864,
 0.0008840110386454718,
 0.0008150373077508284,
 0.0004916222740912361,
 0.0004661502432392617,
 0.00039247359674812667]
```

y esta varianza acumulada:

In [ ]:
```
np.array(values).cumsum()
```

Out[ ]:
```
array([0.53635916, 0.9390998 , 0.96564585, 0.97904609, 0.98491876,
       0.98959396, 0.99211344, 0.99435623, 0.99598128, 0.99695071,
       0.99783472, 0.99864975, 0.99914138, 0.99960753, 1.        ])
```

Esto es bastante bueno ya que solo 2 vectores explica 93% de la varianza. Eso quiere decir que 53% del error lo podemos explicar con una regresion lineal (el qqplot de antes si se ve muy lineal pero las colas no nos permitieron que fuera normal).

In [ ]:
```
pca_matrix = pd.DataFrame(np.transpose(pca.components_), columns=[f'PC{i}' for i in range(15)],
index=df.drop(columns = 'price').columns)

pca_matrix = pca_matrix[pca_matrix.columns[:3]]
pca_matrix
```

Out[ ]:

	PC0	PC1	PC2
symboling	-0.000669	0.000261	0.012177
wheelbase	0.008079	0.003158	-0.061972
carlength	0.017444	0.010535	-0.073878
carwidth	0.002880	0.002107	-0.001934
carheight	0.001629	-0.000485	-0.039725
curbweight	0.811765	0.575871	-0.090852
enginesize	0.055989	0.037741	0.684410
stroke	0.000089	0.000043	0.000227
compressionratio	0.002284	-0.002733	-0.042143
horsepower	0.035111	0.061255	0.713097
peakrpm	-0.579827	0.814163	-0.020735
citympg	-0.005953	-0.009935	-0.032401
highwaympg	-0.006983	-0.010037	-0.020256
fueltype_diesel	0.000208	-0.000198	-0.003330
fueltype_gas	-0.000208	0.000198	0.003330
carbody_convertible	0.000017	0.000028	0.000688
carbody_hardtop	0.000033	0.000016	0.001986
carbody_hatchback	-0.000247	-0.000065	0.002973
carbody_sedan	0.000103	-0.000009	-0.001553
carbody_wagon	0.000095	0.000030	-0.004094
drivewheel_4wd	0.000023	-0.000035	-0.000996
drivewheel_fwd	-0.000490	-0.000420	0.000379
drivewheel_rwd	0.000467	0.000455	0.000617
enginetype_dohc	0.000037	0.000146	0.000517
enginetype_dohcv	0.000005	0.000027	0.000606

	PC0	PC1	PC2
enginetype_l	0.000122	-0.000014	-0.003349
enginetype_ohc	-0.000258	-0.000285	-0.001698
enginetype_ohcf	-0.000011	-0.000076	0.001839
enginetype_ohcv	0.000143	0.000134	0.003015
enginetype_rotor	-0.000038	0.000068	-0.000929
cylindernumber_eight	0.000083	0.000059	0.001696
cylindernumber_five	0.000083	0.000089	-0.001541
cylindernumber_four	-0.000304	-0.000449	-0.004265
cylindernumber_six	0.000170	0.000231	0.004174
cylindernumber_three	-0.000013	-0.000017	0.000090
cylindernumber_twelve	0.000019	0.000020	0.000775
cylindernumber_two	-0.000038	0.000068	-0.000929

Ahora veremos las 5 variables que más aportan al primer componente principal (en valor absoluto) para saber cuales son las más relevantes.

In [ ]:
```
pca_matrix.sort_values(by = 'PC0', key = lambda x: abs(x)).tail(5)
```

Out[ ]:

	PC0	PC1	PC2
carlength	0.017444	0.010535	-0.073878
horsepower	0.035111	0.061255	0.713097
enginesize	0.055989	0.037741	0.684410
peakrpm	-0.579827	0.814163	-0.020735
curbweight	0.811765	0.575871	-0.090852

In [ ]:
```
pca_matrix.sort_values(by = 'PC1', key = lambda x: abs(x)).tail(5)
```

Out[ ]:

	PC0	PC1	PC2
carlength	0.017444	0.010535	-0.073878
enginesize	0.055989	0.037741	0.684410
horsepower	0.035111	0.061255	0.713097
curbweight	0.811765	0.575871	-0.090852
peakrpm	-0.579827	0.814163	-0.020735

Vemos que en ambos vectores son casi las mismas variables por lo que podemos decir que estas son las variables importantes.