Regression

En este laboratorio se trabajará un conjunto de datos que indica ciertas características físicas y rutinarias de las personas. Se pretende crear varios modelos de regresión lineal y polinomial. En base a dichos modelos, desarollados con las características de distintos usuarios, será posible predecir los 'charges' de cada persona.

Este conjunto de datos contiene las siguientes características:

- 'Age': edad la persona
- 'BMI': índice de masa corporal de la persona
- 'Children': si tiene hijos o no
- 'Smoker': si fuma o no
- 'Sex': si el consumidor es mujer u hombre
- 'Region': región del consumidor
- 'Charges': cargos de del ususario

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

In [2]:

```
df = pd.read_csv('insurance.csv')
```

Check the head of ad_data

In [3]:

```
df.head() #visualizacion de algunas filas
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520

Preparación de los datos - datos faltantes

In [4]:

```
print("Cantidad de registros: ", len(df))
```

```
Cantidad de registros:  348
```

In [5]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 348 entries, 0 to 347
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       348 non-null    int64  
 1   sex       348 non-null    int64  
 2   bmi       348 non-null    float64
 3   children  348 non-null    int64  
 4   smoker    348 non-null    int64  
 5   region    348 non-null    int64  
 6   charges   348 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 19.2 KB
```

In [6]:
```
1 df.describe() #visualizacion de algunas filas
```

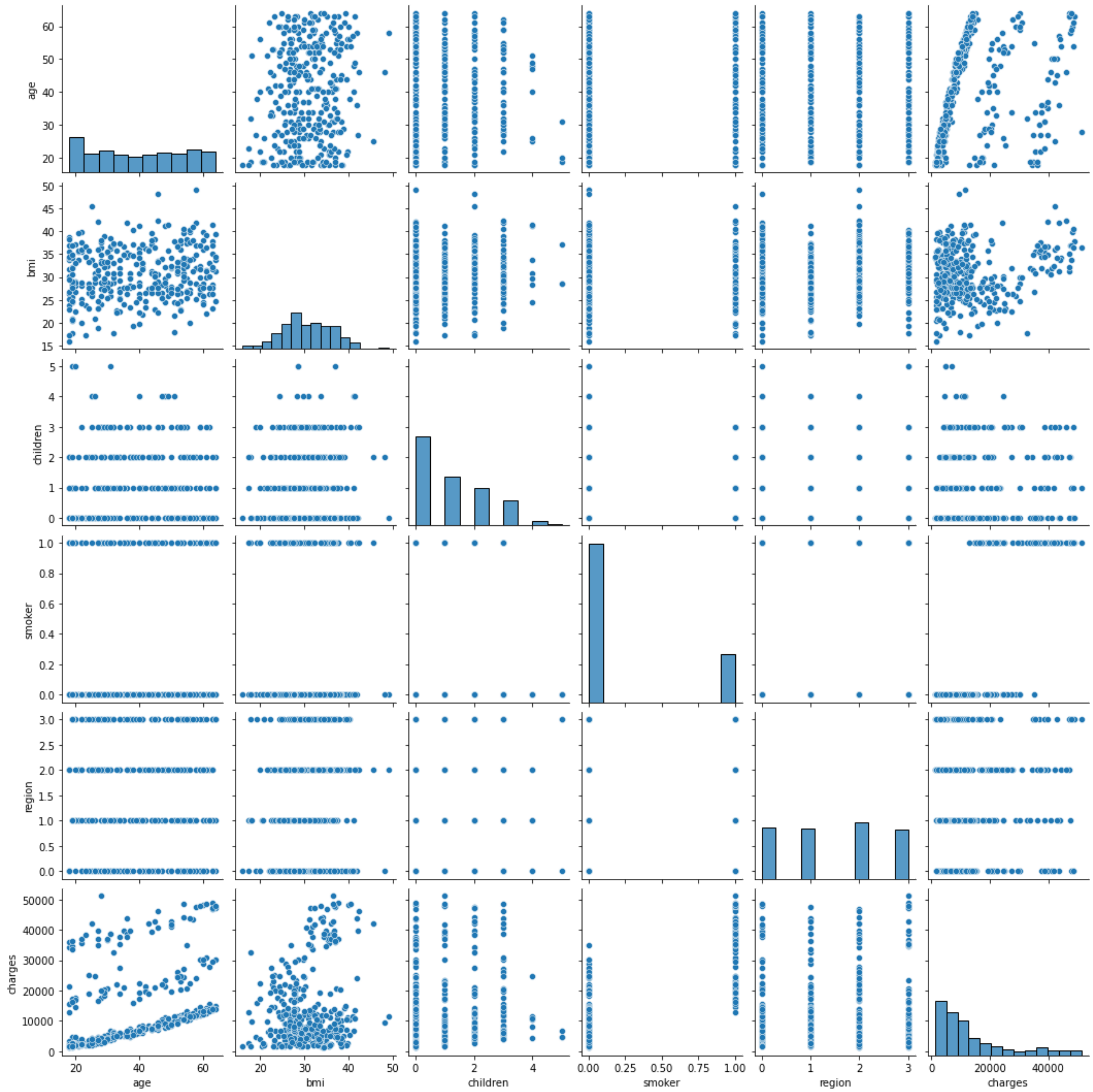
Out[6]:

	age	sex	bmi	children	smoker	region	charges
count	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000
mean	39.591954	0.508621	30.676552	1.091954	0.232759	1.497126	14016.426293
std	14.417015	0.500646	5.625850	1.192021	0.423198	1.104089	12638.887852
min	18.000000	0.000000	15.960000	0.000000	0.000000	0.000000	1137.011000
25%	27.000000	0.000000	26.782500	0.000000	0.000000	1.000000	4888.466125
50%	40.000000	1.000000	30.300000	1.000000	0.000000	2.000000	9719.305250
75%	53.000000	1.000000	34.777500	2.000000	0.000000	2.000000	19006.316150
max	64.000000	1.000000	49.060000	5.000000	1.000000	3.000000	51194.559140

Exploración de los datos

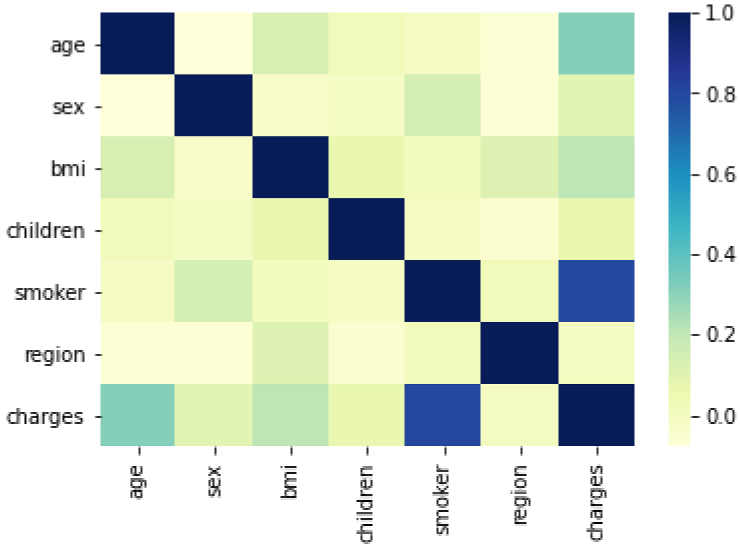
In [15]:

```
sns.pairplot(df[['age','bmi','children','smoker','region','charges']])
plt.show()
```



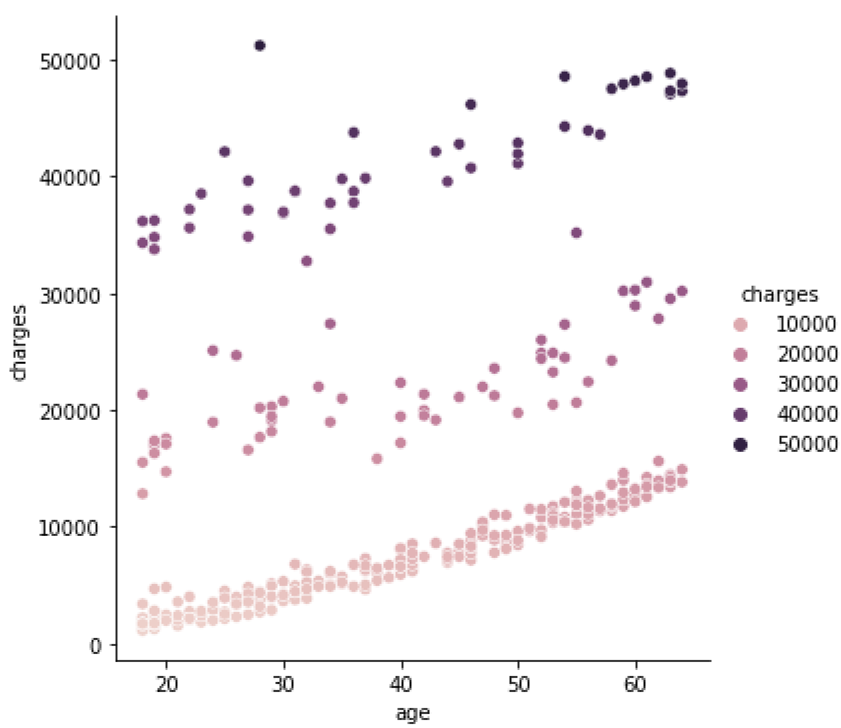
In [16]:

```
sns.heatmap(df.corr(),cmap="YlGnBu")
plt.show()
```



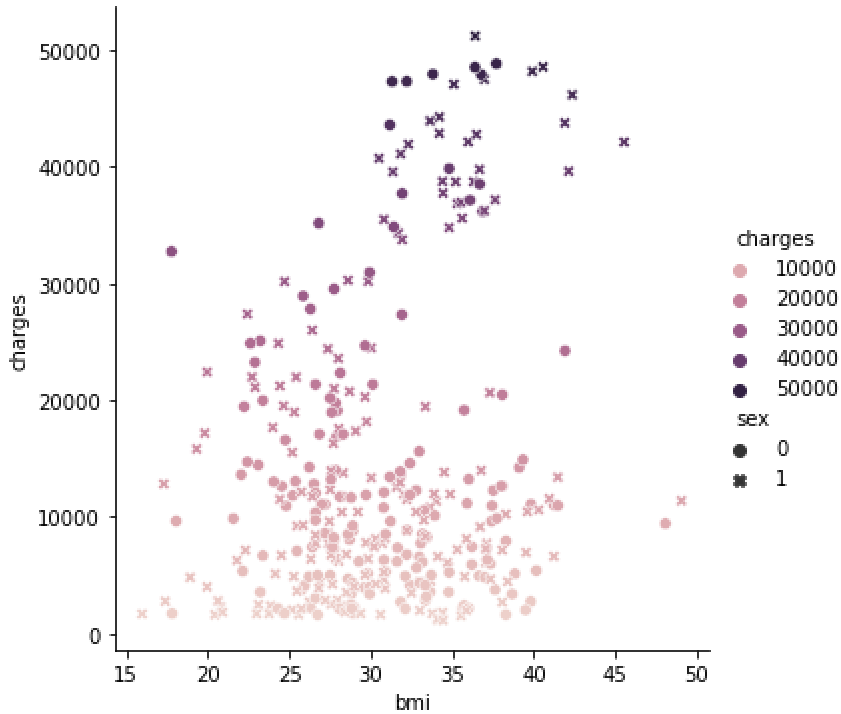
In [17]:

```
#age v charges
sns.relplot(data=df,x="age",y="charges",hue="charges")
plt.show()
```



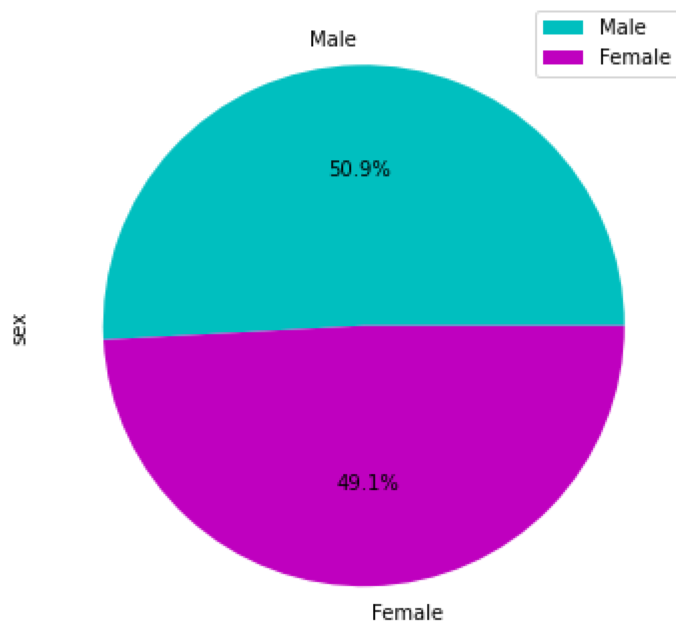
In [19]:

```
#bmi vs Charges diferenciando sex
sns.relplot(data=df,x="bmi",y='charges',hue="charges",style="sex")
plt.show()
```



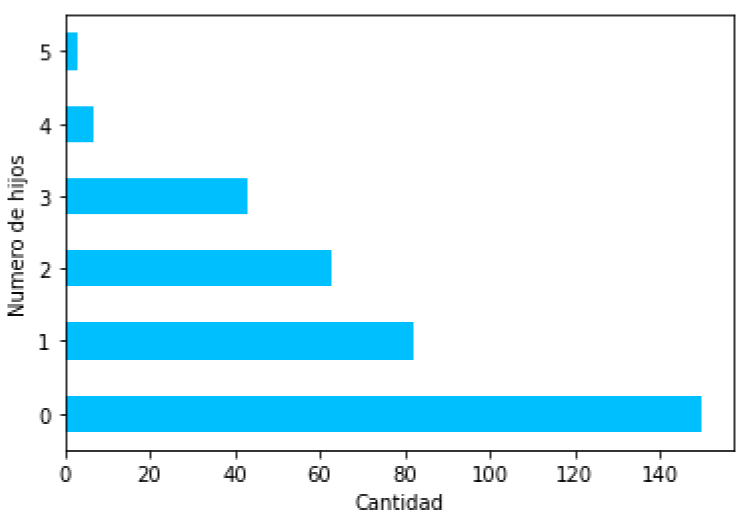
In [10]:

```
#distributuion of sex
plt.figure(figsize=(6,6))
my_lab=["Male","Female"]
mycolors = ["c", "m"]
df.sex.value_counts().plot(kind='pie',labels=my_lab, colors = mycolors, autopct='%1.1f
plt.legend()
plt.show()
```



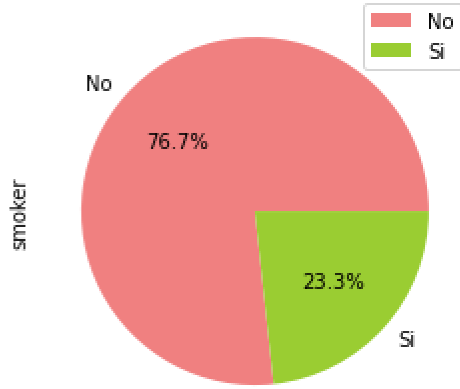
In [8]:

```
#Cantidad of childern
df.children.value_counts().plot(kind="barh",color="deepskyblue")
plt.xlabel("Cantidad")
plt.ylabel("Numero de hijos")
plt.show()
```



In [12]:

```
#Smoker or not
my_lab=["No","Si"]
mycolors = ["lightcoral", "yellowgreen"]
df.smoker.value_counts().plot(kind='pie', labels=my_lab, colors = mycolors, autopct='%
plt.legend()
plt.show()
```



Preparación de los datos – datos categóricos

Label Encoder para codificar los campos de sex y smoker

In [3]:

```
from sklearn.preprocessing import LabelEncoder
le_sex = LabelEncoder()
df["labeled_sex"] = le_sex.fit_transform(df["sex"])
df
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges	labeled_sex
0	19	0	27.900	0	1	3	16884.92400	0
1	18	1	33.770	1	0	2	1725.55230	1
2	28	1	33.000	3	0	2	4449.46200	1
3	33	1	22.705	0	0	1	21984.47061	1
4	32	1	28.880	0	0	1	3866.85520	1
...	...	...	...	...	...	...	...	...
343	63	1	36.765	0	0	0	13981.85035	1
344	49	0	41.470	4	0	2	10977.20630	0
345	34	0	29.260	3	0	2	6184.29940	0
346	33	1	35.750	2	0	2	4889.99950	1
347	46	1	33.345	1	0	0	8334.45755	1

348 rows × 8 columns

In [4]:

```
le_smoker = LabelEncoder()
df["labeled_smoker"] = le_smoker.fit_transform(df["smoker"])
df
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges	labeled_sex	labeled_smoker
0	19	0	27.900	0	1	3	16884.92400	0	1
1	18	1	33.770	1	0	2	1725.55230	1	0
2	28	1	33.000	3	0	2	4449.46200	1	0
3	33	1	22.705	0	0	1	21984.47061	1	0
4	32	1	28.880	0	0	1	3866.85520	1	0
...	...	...	...	...	...	...	...	...	...
343	63	1	36.765	0	0	0	13981.85035	1	0
344	49	0	41.470	4	0	2	10977.20630	0	0
345	34	0	29.260	3	0	2	6184.29940	0	0
346	33	1	35.750	2	0	2	4889.99950	1	0
347	46	1	33.345	1	0	0	8334.45755	1	0

348 rows × 9 columns

On Hot Encoder para el region

In [5]:

```python
#forma 1
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
df3 = pd.DataFrame(ohe.fit_transform(df[["region"]]).toarray())
df_new=pd.concat([df,df3],axis=1)
df_new
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges	labeled_sex	labeled_smoker	0	1	2	3
0	19	0	27.900	0	1	3	16884.92400	0	1	0.0	0.0	0.0	1.0
1	18	1	33.770	1	0	2	1725.55230	1	0	0.0	0.0	1.0	0.0
2	28	1	33.000	3	0	2	4449.46200	1	0	0.0	0.0	1.0	0.0
3	33	1	22.705	0	0	1	21984.47061	1	0	0.0	1.0	0.0	0.0
4	32	1	28.880	0	0	1	3866.85520	1	0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
343	63	1	36.765	0	0	0	13981.85035	1	0	1.0	0.0	0.0	0.0
344	49	0	41.470	4	0	2	10977.20630	0	0	0.0	0.0	1.0	0.0
345	34	0	29.260	3	0	2	6184.29940	0	0	0.0	0.0	1.0	0.0
346	33	1	35.750	2	0	2	4889.99950	1	0	0.0	0.0	1.0	0.0
347	46	1	33.345	1	0	0	8334.45755	1	0	1.0	0.0	0.0	0.0

348 rows × 13 columns

In [6]:

```python
#forma 2
df = pd.get_dummies(df, columns=["region"])
df.head()
```

Out[6]:

	age	sex	bmi	children	smoker	charges	labeled_sex	labeled_smoker	region_0	region_1	region_2
0	19	0	27.900	0	1	16884.92400	0	1	0	0	0
1	18	1	33.770	1	0	1725.55230	1	0	0	0	1
2	28	1	33.000	3	0	4449.46200	1	0	0	0	1
3	33	1	22.705	0	0	21984.47061	1	0	0	1	0
4	32	1	28.880	0	0	3866.85520	1	0	0	1	0

Dividir en training y test - preparación escala

In [7]:

```python
#Splitting the data y elección de la columna "charges" para predecirla
target=df["charges"]
data=np.array(df['bmi'])
data=data.reshape(len(df), 1)
```

Preparación de los datos - escala

In [8]:

```python
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
data=sc.fit_transform(data)
print(data,target)
```

```
[[-4.94245241e-01]
 [ 5.50655000e-01]
 [ 4.13589721e-01]
 [-1.41899085e+00]
 [-3.19798523e-01]
 [-8.78740049e-01]
 [ 4.91912738e-01]
 [-5.22726338e-01]
 [-1.50692010e-01]
 [-8.60939363e-01]
 [-7.93296758e-01]
 [-7.80836278e-01]
 [ 6.62799319e-01]
 [ 1.62759647e+00]
 [ 2.03879231e+00]
 [-1.08166786e+00]
 [ 1.84145023e-02]
 [-1.21606304e+00]
 [ 1.71303977e+00]
```

In [9]:

```python
#Splitting en train y test
from sklearn.model_selection import train_test_split
x_trainA,x_testA,y_trainA,y_testA =train_test_split(data,target,test_size=.3)
x_trainB,x_testB,y_trainB,y_testB =train_test_split(data,target,test_size=.3)
x_trainBp,x_testBp,y_trainBp,y_testBp =train_test_split(data,target,test_size=.3)

print(x_trainA,y_trainB)
```

```
 [-1.3006163 ]
 [-1.43590151]
 [ 0.94761029]
 [ 0.11987841]
 [-0.97931392]
 [ 0.72866186]
 [-0.72565415]
 [ 0.53819452]
 [ 0.44118078]
 [-0.18451331]
 [ 0.27118424]
 [ 1.9996308 ]
 [ 1.14341783]
 [ 0.30589557]
 [ 0.9298096 ]
 [-0.30288787]
 [ 0.52573404]
 [-0.7425648 ]
 [ 1.42199856]
```

Modelación lineal

modelo 1 (sin librería)

In [10]:

```python
from sklearn.metrics import mean_squared_error,r2_score
#Usando bmi
n = x_trainA.shape[1]
r = np.linalg.matrix_rank(x_trainA)
U, sigma, VT = np.linalg.svd(x_trainA, full_matrices=False)
print(U, sigma, VT)
```

```
 [ 0.01388207]
 [ 0.0013782 ]
 [-0.09217755]
 [ 0.12524467]
 [ 0.13032436]
 [-0.09932262]
 [-0.02111761]
 [-0.05399609]
 [ 0.02593937]
 [-0.02111761]
 [-0.01157224]
 [ 0.07115426]
 [ 0.04491846]
 [ 0.02024565]
 [ 0.03827578]
 [-0.01157224]
 [-0.03657998]
 [ 0.08243007]
 [ 0.05161696]
 [ 0.00221551]
```

In [11]:

```python
D_plus = np.diag(np.hstack([1/sigma[:r], np.zeros(n-r)]))
V = VT.T
print(D_plus)
print(V)
```

```
[[0.06271764]]
[[1.]]
```

In [12]:

```
X_plus = V.dot(D_plus).dot(U.T)
w = X_plus.dot(y_trainA)
print(X_plus)
print(w)
```

```
[[-2.98739874e-03  1.66886770e-03 -1.85659117e-03  1.46931343e-03
  -2.43424829e-03  1.00368678e-03 -2.85436255e-03 -2.76333780e-03
  -9.14835029e-04 -4.59711242e-04  1.21787262e-02 -5.30502910e-03
   3.53137428e-03  2.08898197e-03 -1.59401976e-03  4.42761804e-03
  -1.45398167e-03  2.93271145e-03 -1.52400071e-03  2.88719907e-03
   1.38951586e-04 -2.18918163e-03  6.17109224e-03  5.12780848e-03
   9.33667737e-04  4.26307329e-03 -1.24392454e-03  6.45817032e-03
  -9.30661747e-03  3.70642189e-03  3.53137428e-03  3.39833809e-03
   4.06351901e-03  1.86842198e-03  6.24811319e-03  6.63321793e-03
  -3.65257966e-03 -7.60793132e-04  2.88719907e-03 -1.94411498e-03
   8.64373028e-05  1.20674201e-03  6.19209796e-03 -3.05391683e-03
   7.55746932e-03 -5.29730286e-04  1.85791912e-03  1.62685627e-03
  -9.01603844e-03  2.86619336e-03 -1.03043889e-02  4.66218184e-03
  -2.07014926e-03 -3.65257966e-03 -1.65703690e-03 -5.86518145e-03
   5.32386181e-03 -7.51062899e-03 -3.38650729e-03 -1.39096453e-03
  -1.66403880e-03 -5.38205005e-03  2.33404862e-03  7.55746932e-03
  -5.11597768e-03 -5.64812242e-03  3.72742760e-03  4.71542046e-04
  -3.85213393e-03  2.86619336e-03 -2.85436255e-03  2.11698958e-03
   1.73538579e-03 -7.25783610e-04  1.06670392e-03  7.86555311e-03
   4.49763709e-03  1.20324106e-03  3.65740856e-03 -1.19141025e-03
   2.06797625e-03 -2.92088064e-03  5.59343513e-03  3.99700092e-03
  -2.05614545e-03 -1.99312831e-03 -5.29730286e-04 -1.80407689e-03
   4.00750378e-03  7.38942361e-03  1.86842198e-03  1.80190389e-03
   7.02604892e-04  3.86046378e-03 -7.61565756e-03  4.49763709e-03
  -2.18918163e-03 -5.78115860e-03  1.80190389e-03  8.70650598e-04
   8.64373028e-05 -5.78115860e-03  7.85505025e-03  8.17363691e-03
  -6.22928048e-03 -1.32444644e-03 -3.38650729e-03  1.62685627e-03
  -1.32444644e-03 -7.25783610e-04  4.46262756e-03  2.81718003e-03
   1.26975915e-03  2.40056671e-03 -7.25783610e-04 -2.29421020e-03
   5.16981991e-03  3.23729429e-03  1.38951586e-04  1.13672297e-03
   6.39165223e-03 -2.22419116e-03 -4.38077772e-03  2.46708480e-03
   4.70769422e-03 -5.31553196e-03 -1.45748262e-03 -4.45079676e-03
  -1.27120782e-04 -5.92747426e-04  2.47408671e-03  4.39610947e-03
  -4.38427867e-03 -3.45652633e-03  2.16600291e-03  7.24334940e-05
   6.92379697e-03  2.60712289e-03 -5.11597768e-03  8.04132506e-04
  -1.45398167e-03  4.19655520e-03 -3.45652633e-03  5.32736276e-03
  -1.29993977e-03 -1.19141025e-03  4.66218184e-03 -5.98071288e-03
  -2.12266354e-03 -6.83772183e-04 -3.93193150e-04  2.39706576e-03
  -2.32221782e-03  4.52914566e-03 -2.65480828e-03 -2.99440064e-03
  -6.83772183e-04  5.40088276e-03 -4.59711242e-04  8.04132506e-04
   8.64373028e-05  1.20324106e-03 -3.68758918e-03 -4.22673582e-03
   4.86173612e-03 -2.18918163e-03 -2.25569972e-03  7.38942361e-03
   1.28719148e-02 -4.25474344e-03  2.18700863e-03  4.72869993e-03
  -4.31776058e-03 -5.22800815e-03  3.38505862e-04 -2.85436255e-03
  -2.22419116e-03 -2.22419116e-03 -1.59401976e-03  4.19655520e-03
  -4.84990531e-03 -4.58383294e-03  4.93875706e-03  1.56456347e-04
   5.16981991e-03  4.36532524e-04 -1.45748262e-03  1.07020487e-03
  -4.46480057e-03 -2.63657918e-04  8.01959501e-03 -3.13443873e-03
   8.56646789e-04  3.78344284e-03  4.39610947e-03  8.70650598e-04
   1.62685627e-03 -1.99312831e-03  3.86396474e-03  6.71096322e-04
  -3.45652633e-03 -5.29730286e-04 -5.92119669e-03  2.60012099e-03
  -3.31998920e-03 -1.72355499e-03 -1.24392454e-03  2.06797625e-03
   4.26307329e-03 -8.24232800e-03 -4.65035104e-03 -2.84035874e-03
   4.32258948e-03  1.00368678e-03 -5.45907100e-03 -1.39096453e-03
  -4.30375677e-03  3.39833809e-03  6.46589656e-04 -2.58829018e-03
  -2.43424829e-03 -7.51062899e-03 -1.91610736e-03  5.32386181e-03
```

```
  -3.85213393e-03  3.44735142e-03 -1.03386740e-03 -6.84544807e-03
  -4.68886151e-03  1.07020487e-03  2.39706576e-03 -2.05614545e-03
   8.64373028e-05 -1.87409593e-03 -1.25792835e-03  1.04072444e-02
  -3.76461013e-03 -2.76333780e-03  5.06551568e-04 -1.32444644e-03
  -9.37313557e-03  4.86173612e-03  3.55237999e-03]]
[3796.66725941]
```

In [13]:
```python
error = np.linalg.norm(x_trainA.dot(w) - y_trainA, ord=2) ** 2
print(error)
```

```
93133829264.44078
```

In [14]:
```python
np.linalg.lstsq(x_trainA, y_trainA)
```

```
<ipython-input-14-038c38aaba46>:1: FutureWarning: `rcond` parameter will change to the de
fault of machine precision times ``max(M, N)`` where M and N are the input matrix dimensi
ons.
To use the future default and silence this warning we advise to pass `rcond=None`, to kee
p using the old, explicitly pass `rcond=-1`.
  np.linalg.lstsq(x_trainA, y_trainA)
```

Out[14]:
```
(array([3796.66725941]), array([9.31338293e+10]), 1, array([15.94447688]))
```

modelo 2 (con librería)

In [15]:
```python
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

In [16]:
```python
#Training la data con BMI
lr.fit(x_trainB,y_trainB)
w2 = lr.coef_[0]
print(w2)
```

```
2750.702214787475
```

modelo 3 (con librería)

In [17]:
```python
target_c=df['charges']
data_c=df.drop(columns=['charges'])
data_c=sc.fit_transform(data_c)
```

In [18]:

```
x_trainC,x_testC,y_trainC,y_testC=train_test_split(data_c,target_c,test_size=.3)
x_trainCp,x_testCp,y_trainCp,y_testCp=train_test_split(data_c,target_c,test_size=.3)
print(x_trainC, y_trainC)
```

```
[[-1.49982813  0.98290472  0.550655   ... -0.56850147  1.63191847
  -0.55522129]
 [ 0.0978062  -1.01739261  0.17061036 ... -0.56850147 -0.61277571
  -0.55522129]
 [ 0.0978062  -1.01739261  1.14341783 ... -0.56850147 -0.61277571
   1.80108368]
 ...
 [ 0.44511801 -1.01739261 -0.52272634 ...  1.75901042 -0.61277571
  -0.55522129]
 [-0.11058089  0.98290472  0.71620138 ... -0.56850147 -0.61277571
   1.80108368]
 [ 1.27866635  0.98290472  1.11760683 ...  1.75901042 -0.61277571
  -0.55522129]] 1       1725.55230
228     7358.17565
215     7371.77200
247     1986.93340
341    13352.09980
          ...
50      2211.13075
267    14590.63205
88      8026.66660
129     6082.40500
55     47496.49445
Name: charges, Length: 243, dtype: float64
```

In [19]:

```
lr_c = LinearRegression()
lr_c.fit(x_trainC, y_trainC)
w3 = lr_c.coef_[0]
print(w3)
```

```
4098.962376236883
```

Evaluación de modelos

modelo 1

In [21]:

```
from sklearn.metrics import mean_absolute_error
#train data
print(mean_squared_error(lr.predict(x_trainA),y_trainA))
print(r2_score(lr.predict(x_trainA),y_trainA))
print(mean_absolute_error(lr.predict(x_trainA), y_trainA))
```

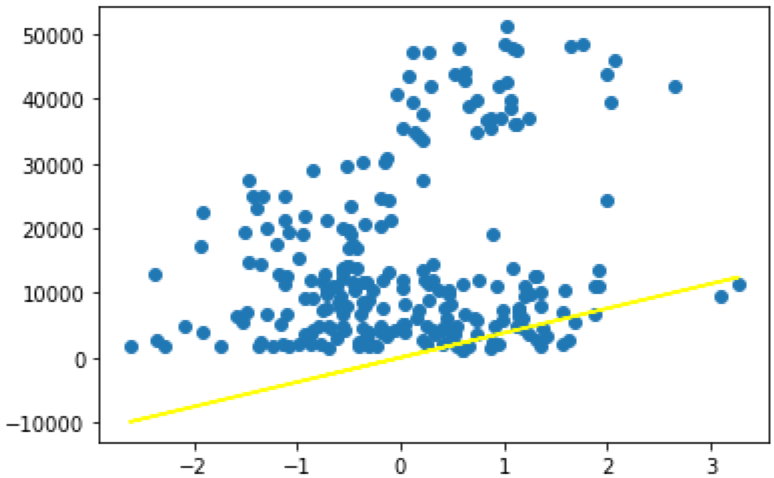
```
171861827.5824608
-20.756528196400208
10664.75573459395
```

In [22]:
```
#test data
print(mean_squared_error(lr.predict(x_testA),y_testA))
print(r2_score(lr.predict(x_testA),y_testA))
print(mean_absolute_error(lr.predict(x_testA), y_testA))
```

```
106911476.47814861
-15.032345689714223
8240.152805604483
```

In [26]:
```
#graficación de los resultados
plt.scatter(x_trainA, y_trainA)
plt.plot(x_trainA, w*x_trainA, c='yellow')
```

Out[26]: [<matplotlib.lines.Line2D at 0x1403dcf0400>]



modelo 2

In [27]:
```
#train data
print(mean_squared_error(lr.predict(x_trainB),y_trainB))
print(r2_score(lr.predict(x_trainB),y_trainB))
print(mean_absolute_error(lr.predict(x_trainB), y_trainB))
```

```
154730820.70977762
-19.89548300409495
9978.182784924107
```

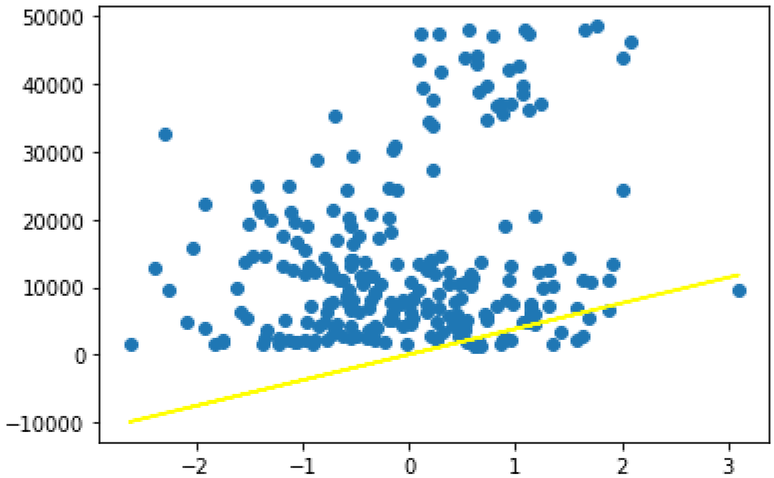
In [28]:
```
#test data
print(mean_squared_error(lr.predict(x_testB),y_testB))
print(r2_score(lr.predict(x_testB),y_testB))
print(mean_absolute_error(lr.predict(x_testB), y_testB))
```

```
146557520.95492962
-17.511210971213572
9829.078774840393
```

In [32]:

```
#graficacion de la data entrenada
plt.scatter(x_trainB, y_trainB)
plt.plot(x_trainB, w*x_trainB, c='yellow')
```

Out[32]: [<matplotlib.lines.Line2D at 0x1403dfcb7c0>]



In [35]:

```
#p-values
import statsmodels.api as sm

X2 = sm.add_constant(df)
est = sm.OLS(target, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 3.216e+31
Date:                Sat, 17 Apr 2021   Prob (F-statistic):               0.00
Time:                        23:52:57   Log-Likelihood:                 8212.5
No. Observations:                 348   AIC:                        -1.641e+04
Df Residuals:                     338   BIC:                        -1.637e+04
Df Model:                           9
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const          -3.183e-12   3.88e-12     -0.820      0.413   -1.08e-11    4.46e-12
age            -2.593e-13   6.17e-14     -4.202      0.000   -3.81e-13   -1.38e-13
sex            -4.547e-13   7.54e-13     -0.603      0.547   -1.94e-12    1.03e-12
bmi             4.192e-13   1.43e-13      2.936      0.004    1.38e-13       7e-13
children       -2.842e-13   6.29e-13     -0.452      0.652   -1.52e-12    9.54e-13
smoker          7.731e-12   1.72e-12      4.486      0.000    4.34e-12    1.11e-11
charges            1.0000   1.23e-16   8.11e+15      0.000       1.000       1.000
labeled_sex    -5.684e-13   7.54e-13     -0.753      0.452   -2.05e-12    9.16e-13
labeled_smoker  6.594e-12   1.72e-12      3.826      0.000     3.2e-12    9.98e-12
region_0       -1.137e-12    1.6e-12     -0.711      0.478   -4.28e-12    2.01e-12
region_1        1.961e-12   1.54e-12      1.270      0.205   -1.08e-12       5e-12
region_2       -3.411e-12    1.7e-12     -2.006      0.046   -6.76e-12   -6.58e-14
region_3        2.274e-13   1.64e-12      0.138      0.890      -3e-12    3.46e-12
==============================================================================
Omnibus:                       67.483   Durbin-Watson:                   1.435
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              102.646
Skew:                          -1.299   Prob(JB):                     5.14e-23
Kurtosis:                       3.576   Cond. No.                     2.37e+20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
[2] The smallest eigenvalue is 2.2e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

modelo 3

In [37]:

```
#train data
print(mean_squared_error(lr_c.predict(x_trainC),y_trainC))
print(r2_score(lr_c.predict(x_trainC),y_trainC))
print(mean_absolute_error(lr_c.predict(x_trainC), y_trainC))
```

```
40211847.45231889
0.6394121730976938
4446.509843020969
```

In [38]:

```
#test data
print(mean_squared_error(lr_c.predict(x_testC),y_testC))
print(r2_score(lr_c.predict(x_testC),y_testC))
print(mean_absolute_error(lr_c.predict(x_testC), y_testC))
```

```
27813558.779042557
0.7940053673577131
4066.795234775514
```

In [40]:

```
X3 = sm.add_constant(df)
est = sm.OLS(target_c, X3)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 3.216e+31
Date:                Sun, 18 Apr 2021   Prob (F-statistic):               0.00
Time:                        00:03:07   Log-Likelihood:                 8212.5
No. Observations:                 348   AIC:                        -1.641e+04
Df Residuals:                     338   BIC:                        -1.637e+04
Df Model:                           9
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const          -3.183e-12   3.88e-12     -0.820      0.413   -1.08e-11    4.46e-12
age            -2.593e-13   6.17e-14     -4.202      0.000   -3.81e-13   -1.38e-13
sex            -4.547e-13   7.54e-13     -0.603      0.547   -1.94e-12    1.03e-12
bmi             4.192e-13   1.43e-13      2.936      0.004    1.38e-13       7e-13
children       -2.842e-13   6.29e-13     -0.452      0.652   -1.52e-12    9.54e-13
smoker          7.731e-12   1.72e-12      4.486      0.000    4.34e-12    1.11e-11
charges            1.0000   1.23e-16   8.11e+15      0.000       1.000       1.000
labeled_sex    -5.684e-13   7.54e-13     -0.753      0.452   -2.05e-12    9.16e-13
labeled_smoker  6.594e-12   1.72e-12      3.826      0.000     3.2e-12    9.98e-12
region_0       -1.137e-12    1.6e-12     -0.711      0.478   -4.28e-12    2.01e-12
region_1        1.961e-12   1.54e-12      1.270      0.205   -1.08e-12       5e-12
region_2       -3.411e-12    1.7e-12     -2.006      0.046   -6.76e-12   -6.58e-14
region_3        2.274e-13   1.64e-12      0.138      0.890      -3e-12    3.46e-12
==============================================================================
Omnibus:                       67.483   Durbin-Watson:                   1.435
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              102.646
Skew:                          -1.299   Prob(JB):                     5.14e-23
Kurtosis:                       3.576   Cond. No.                     2.37e+20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
[2] The smallest eigenvalue is 2.2e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Regresion polinomial

modelo 2

In [42]:

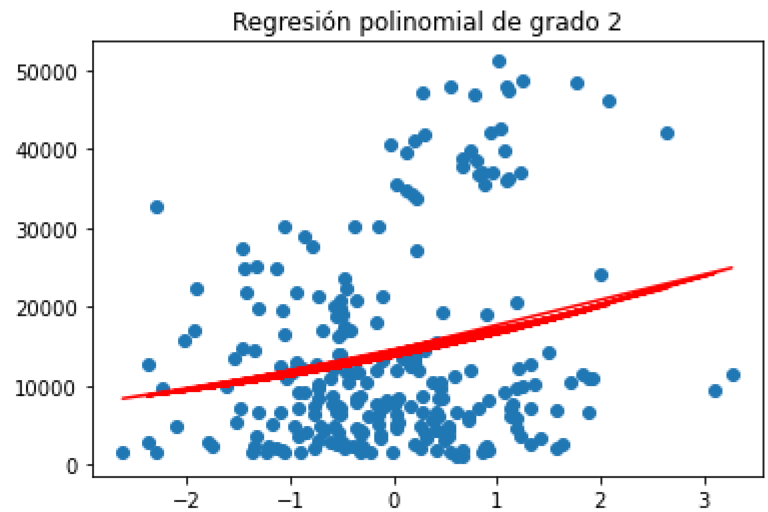
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
grado=2
polyreg=make_pipeline(PolynomialFeatures(grado),LinearRegression())
polyreg.fit(x_trainBp,y_trainBp)
```

Out[42]:

```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures()),
                ('linearregression', LinearRegression())])
```

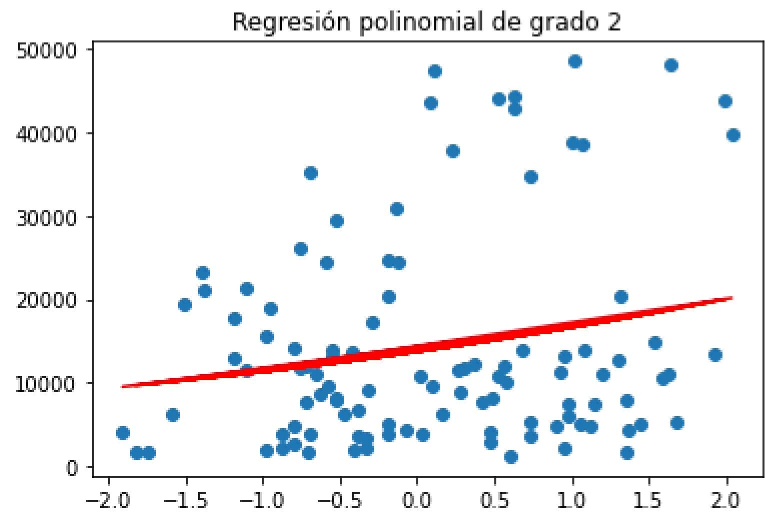
In [44]:

```
#train data
plt.figure()
plt.scatter(x_trainBp,y_trainBp)
plt.plot(x_trainBp,polyreg.predict(x_trainBp),color="red")
plt.title("Regresión polinomial de grado "+str(grado))
plt.show()
print(mean_squared_error(polyreg.predict(x_trainBp),y_trainBp))
print(r2_score(polyreg.predict(x_trainBp),y_trainBp))
print(mean_absolute_error(polyreg.predict(x_trainBp), y_trainBp))
```




```
151094918.98186207
-18.985998061181988
9887.623463530665
```

In [46]:
```
#test data
plt.figure()
plt.scatter(x_testBp,y_testBp)
plt.plot(x_testBp,polyreg.predict(x_testBp),color="red")
plt.title("Regresión polinomial de grado "+str(grado))
plt.show()
print(mean_squared_error(polyreg.predict(x_testBp),y_testBp))
print(r2_score(polyreg.predict(x_testBp),y_testBp))
print(mean_absolute_error(polyreg.predict(x_testBp), y_testBp))
```




```
154562639.28068462
-22.267578920673483
9897.778353929572
```

modelo 3

In [47]:
```
polyreg3=make_pipeline(PolynomialFeatures(grado),LinearRegression())
polyreg3.fit(x_trainCp,y_trainCp)
```

Out[47]:
```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures()),
                ('linearregression', LinearRegression())])
```

In [48]:
```
#train data
print(mean_squared_error(polyreg3.predict(x_trainCp),y_trainCp))
print(r2_score(polyreg3.predict(x_trainCp),y_trainCp))
print(mean_absolute_error(polyreg3.predict(x_trainCp), y_trainCp))
```

```
20410446.439592738
0.874196326212864
2579.5735338683126
```

In [49]:
```
#test data
print(mean_squared_error(polyreg3.predict(x_testCp),y_testCp))
print(r2_score(polyreg3.predict(x_testCp),y_testCp))
print(mean_absolute_error(polyreg3.predict(x_testCp), y_testCp))
```

```
23072669.07394339
0.7875220543392588
2657.909713619047
```