HOJA DE TRABAJO 4 MODELOS DE REGRESION LINEAL

Raul Jimenez 19017 Donaldo Garcia 19683 Oscar Saravia 19322 link al repo:
https://github.com/raulangelj/HT4_MRL

In [ ]:
```
# from re import U
from statsmodels.graphics.gofplots import qqplot
import numpy as np
import pandas as pd
# import pandasql as ps
import matplotlib.pyplot as plt
# import scipy.stats as stats
import statsmodels.stats.diagnostic as diag
# import statsmodels.api as sm
import seaborn as sns
# import random
import sklearn.cluster as cluster
# import sklearn.metrics as metrics
import sklearn.preprocessing
# import scipy.cluster.hierarchy as sch
import pyclustertend
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import normaltest
from sklearn.linear_model import Ridge
from yellowbrick.regressor import ResidualsPlot
# import sklearn.mixture as mixture
# from sklearn import datasets
# from sklearn.cluster import DBSCAN
# from numpy import unique
# from numpy import where
# from matplotlib import pyplot
# from sklearn.datasets import make_classification
# from sklearn.cluster import Birch
# from sklearn.mixture import GaussianMixture


# %matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
```

1. Use los mismos conjuntos de entrenamiento y prueba que usó para los árboles de decisión en la hoja de trabajo anterior

Datos obtenidos de HT3

Preprocesamiento y analisis de datos del lab anterior

In [ ]:
```
train = pd.read_csv('./train.csv', encoding='latin1')
train.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub

5 rows × 81 columns

Se deciden utilizar estas variables debido a que estas son las que nos permiten predecir el comportamiento de este mercad o en un futoro. Con estas variables podemos ver si tiene alguna importanacia en el precio la cantidad del espacio, cantidad de cuartos/baños e incluso el año en el que se termina vendiendo la casa

- SalePrice - **CUANTITATIVO CONTINUO** debido a que el precio puede tener centavos; the property's sale price in dollars. This is the target variable that you're trying to predict.
- LotArea: **CUANTITATIVO CONTINUO** Lot size in square feet
- OverallCond: **CUANTITATIVO DISCRETO** Overall condition rating
- YearBuilt: **CUANTITATIVO DISCRETO** Original construction date
- MasVnrArea: **CUANTITATIVO CONTINUO** Masonry veneer area in square feet
- TotalBsmtSF: **CUANTITATIVO CONTINUO** Total square feet of basement area
- 1stFlrSF: **CUANTITATIVO CONTINUO** First Floor square feet
- 2ndFlrSF: **CUANTITATIVO CONTINUO** Second floor square feet
- GrLivArea: **CUANTITATIVO CONTINUO** Above grade (ground) living area square feet
- TotRmsAbvGrd: **CUANTITATIVO DISCRETO** Total rooms above grade (does not include bathrooms)
- GarageCars: **CUANTITATIVO DISCRETO** Size of garage in car capacity
- WoodDeckSF: **CUANTITATIVO CONTINUO** Wood deck area in square feet
- OpenPorchSF: **CUANTITATIVO CONTINUO** Open porch area in square feet
- EnclosedPorch: **CUANTITATIVO CONTINUO** Enclosed porch area in square feet
- PoolArea: **CUANTITATIVO CONTINUO** Pool area in square feet
- Neighborhood: **CUALITATIVO NOMINAL** Physical locations within Ames city limits

In [ ]:
```
usefullAttr = ['SalePrice', 'LotArea', 'OverallCond', 'YearBuilt', 'MasVnrArea', 'Tota
               '2ndFlrSF', 'GrLivArea', 'TotRmsAbvGrd', 'GarageCars', 'WoodDeckSF', 'C
```

In [ ]:
```
data = train[usefullAttr]
data.head()
```

Out[ ]:

	SalePrice	LotArea	OverallCond	YearBuilt	MasVnrArea	TotalBsmtSF	1stFlrSF	2ndFlrSF	GrLivAre
0	208500	8450	5	2003	196.0	856	856	854	171
1	181500	9600	8	1976	0.0	1262	1262	0	126
2	223500	11250	5	2001	162.0	920	920	866	178
3	140000	9550	5	1915	0.0	756	961	756	171
4	250000	14260	5	2000	350.0	1145	1145	1053	219

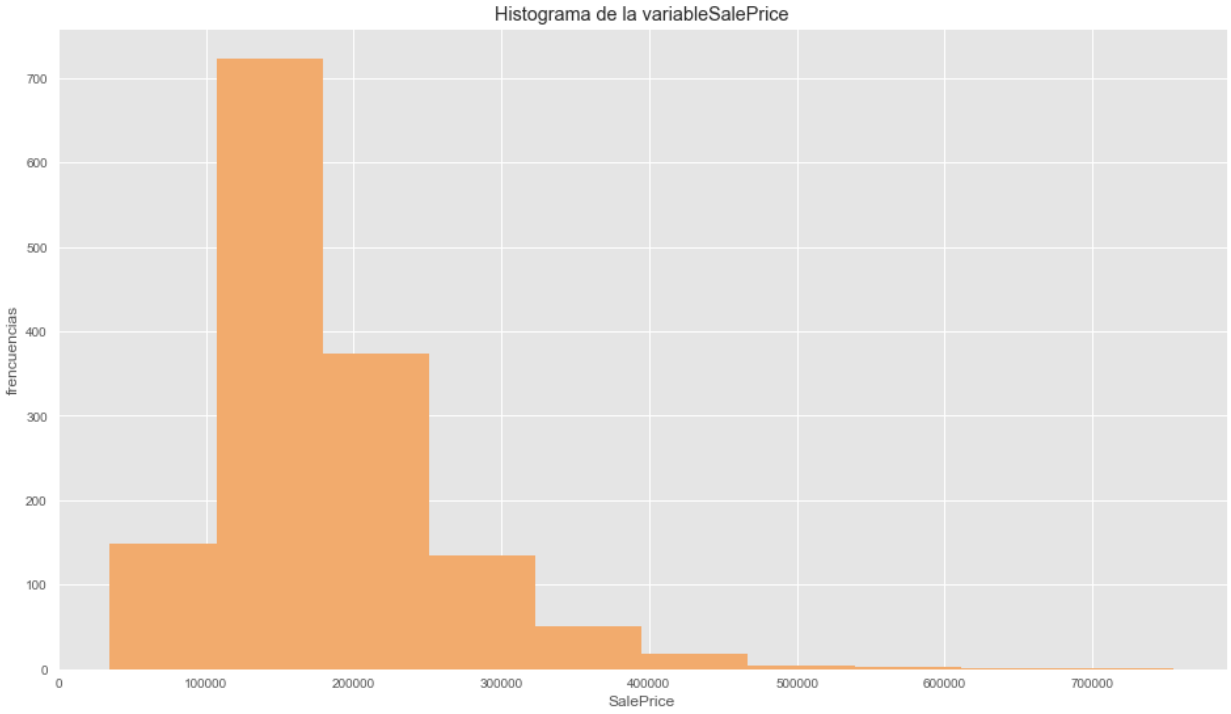
GRAFICAS DE VARIABLES

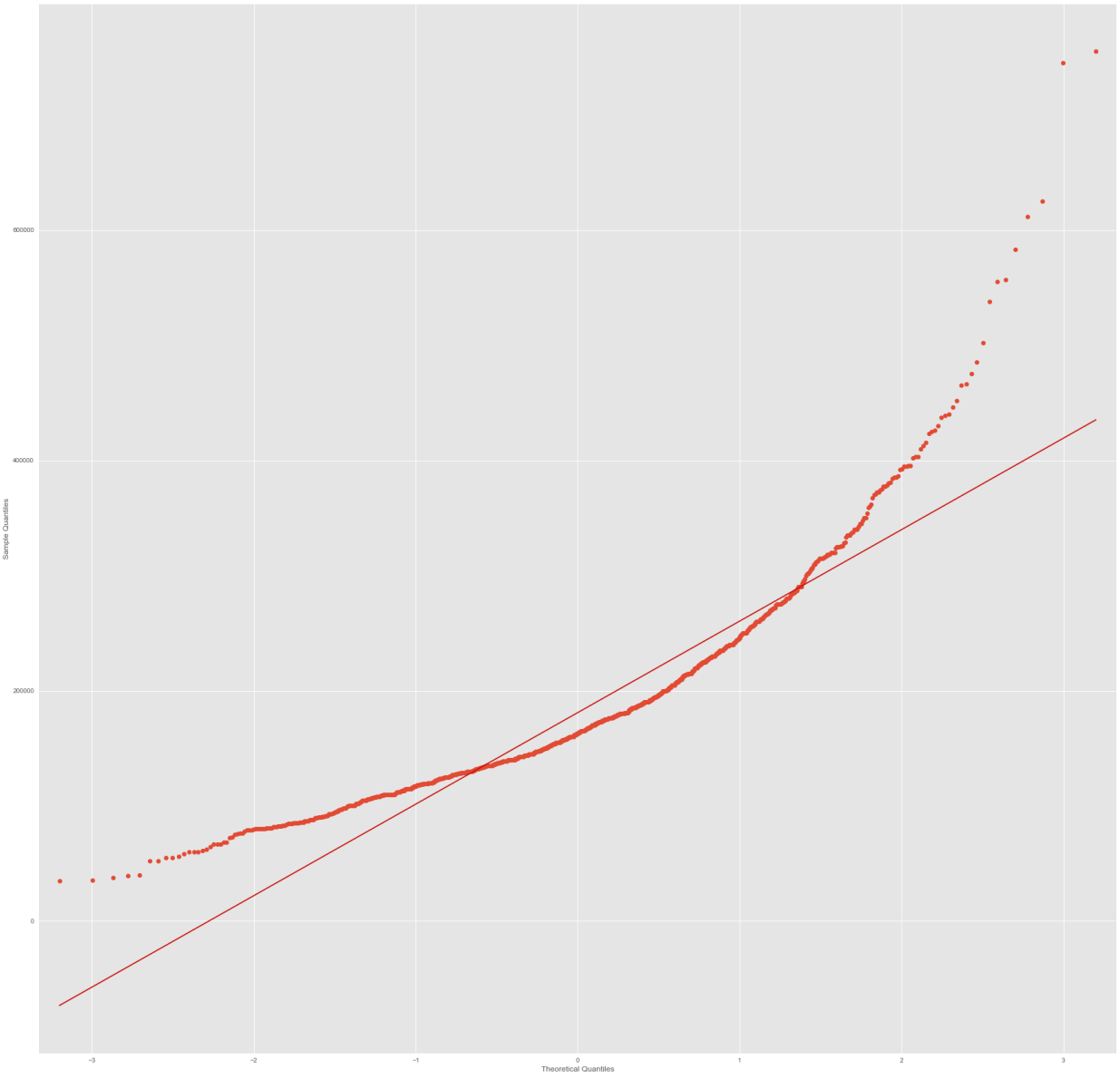
In [ ]:
```
def get_histogram_qq(variable):
    plt.hist(x=data[variable] .dropna(), color='#F2AB6D', rwidth=1)
    plt.title(f'Histograma de la variable{variable}')
    plt.xlabel(variable)
    plt.ylabel('frencuencias')
    plt.rcParams['figure.figsize'] = (30, 30)
    plt.show()

    distribucion_generada = data[variable].dropna()
    # Represento el Q-Q plot
    qqplot(distribucion_generada, line='s')
    plt.show()
```

SalePricee

Se puede determinar que la variable SalePrice no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('SalePrice')
```

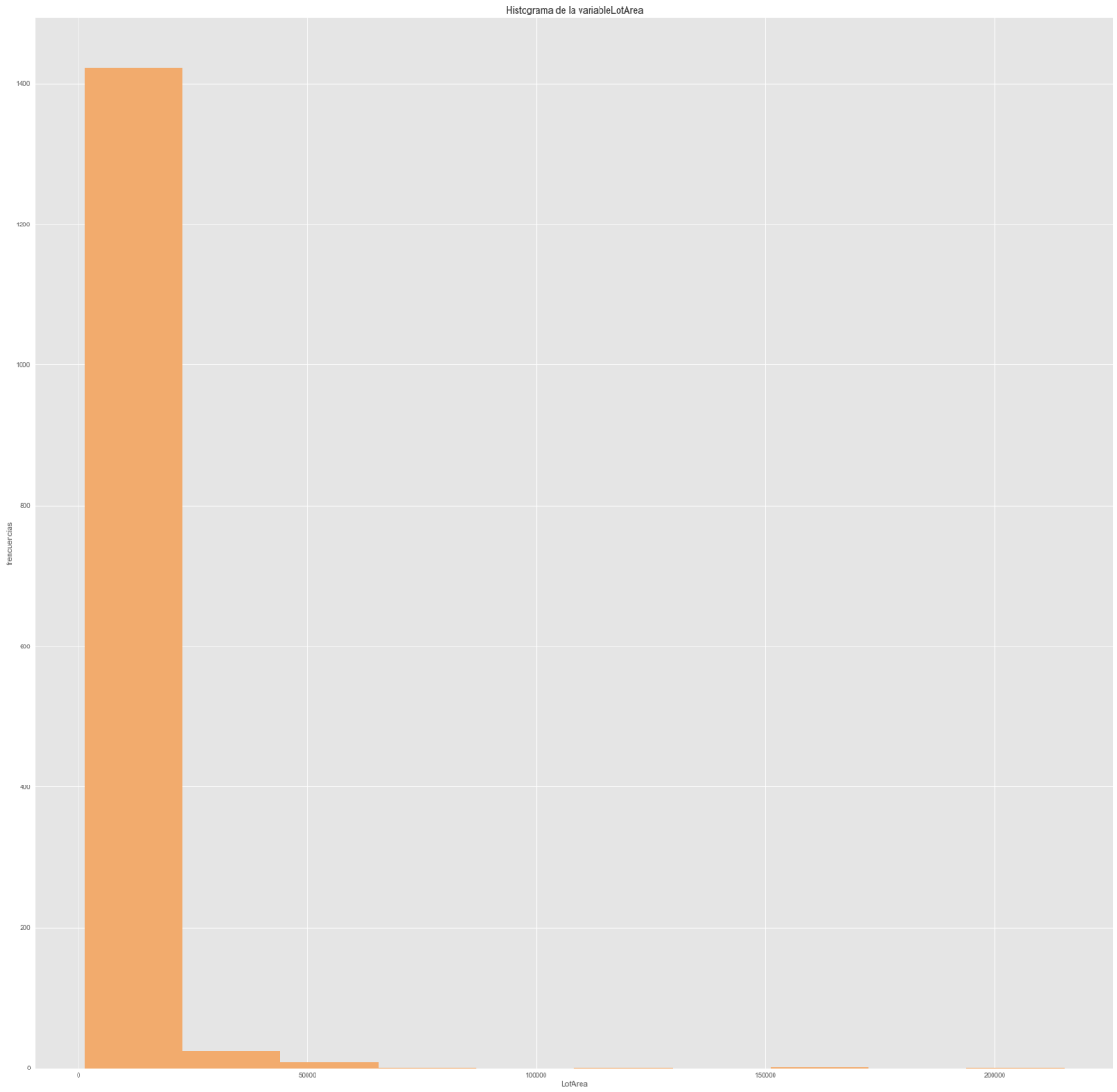

Histograma de la variableSalePrice
frencuencias
SalePrice
0
100
200
300
400
500
600
700
0
100000
200000
300000
400000
500000
600000
700000

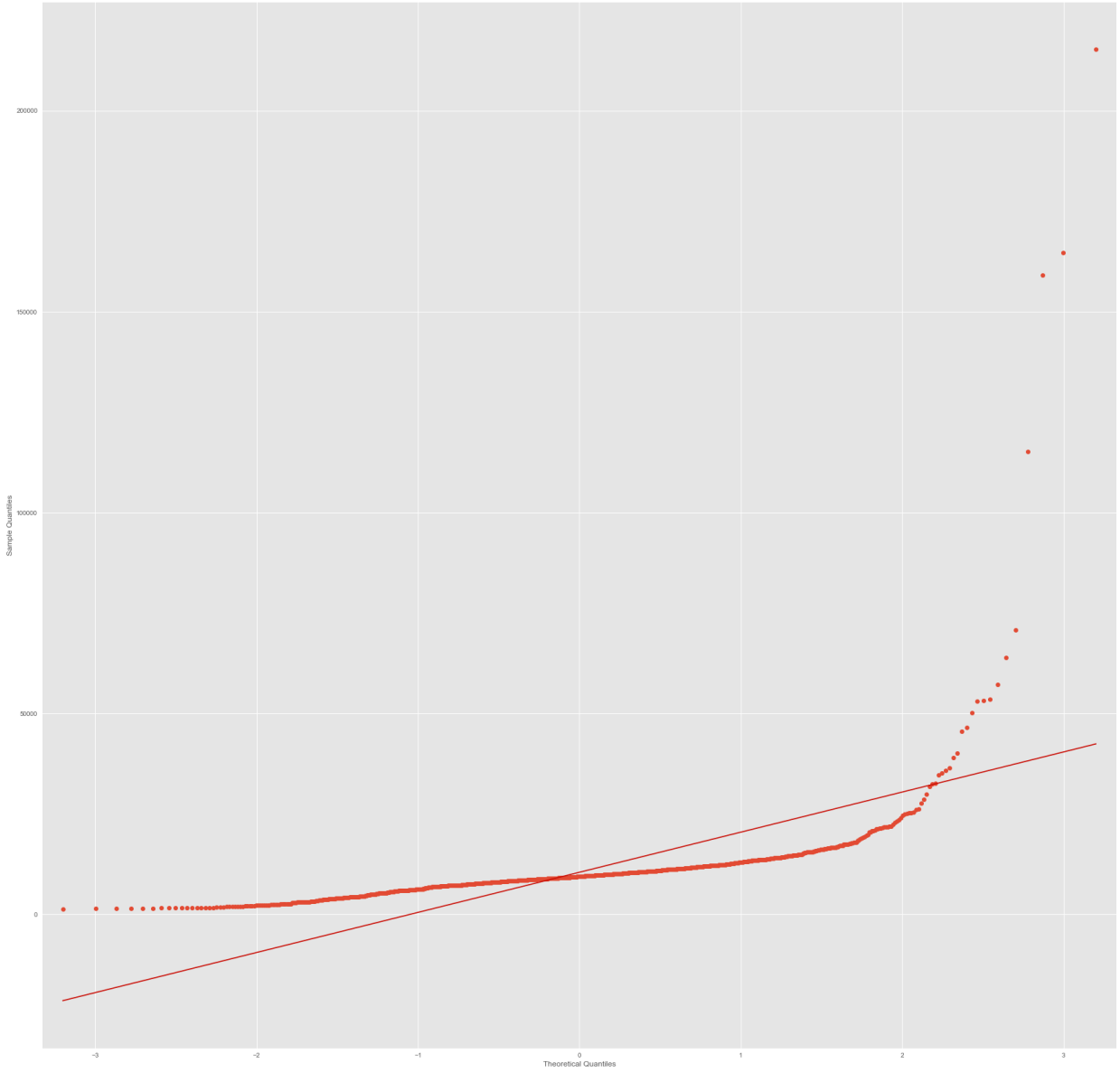

Sample Quantiles
Theoretical Quantiles
0
200000
400000
600000
−3
−2
−1
0
1
2
3

LotArea

Se puede determinar que la variable LotArea no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('LotArea')
```

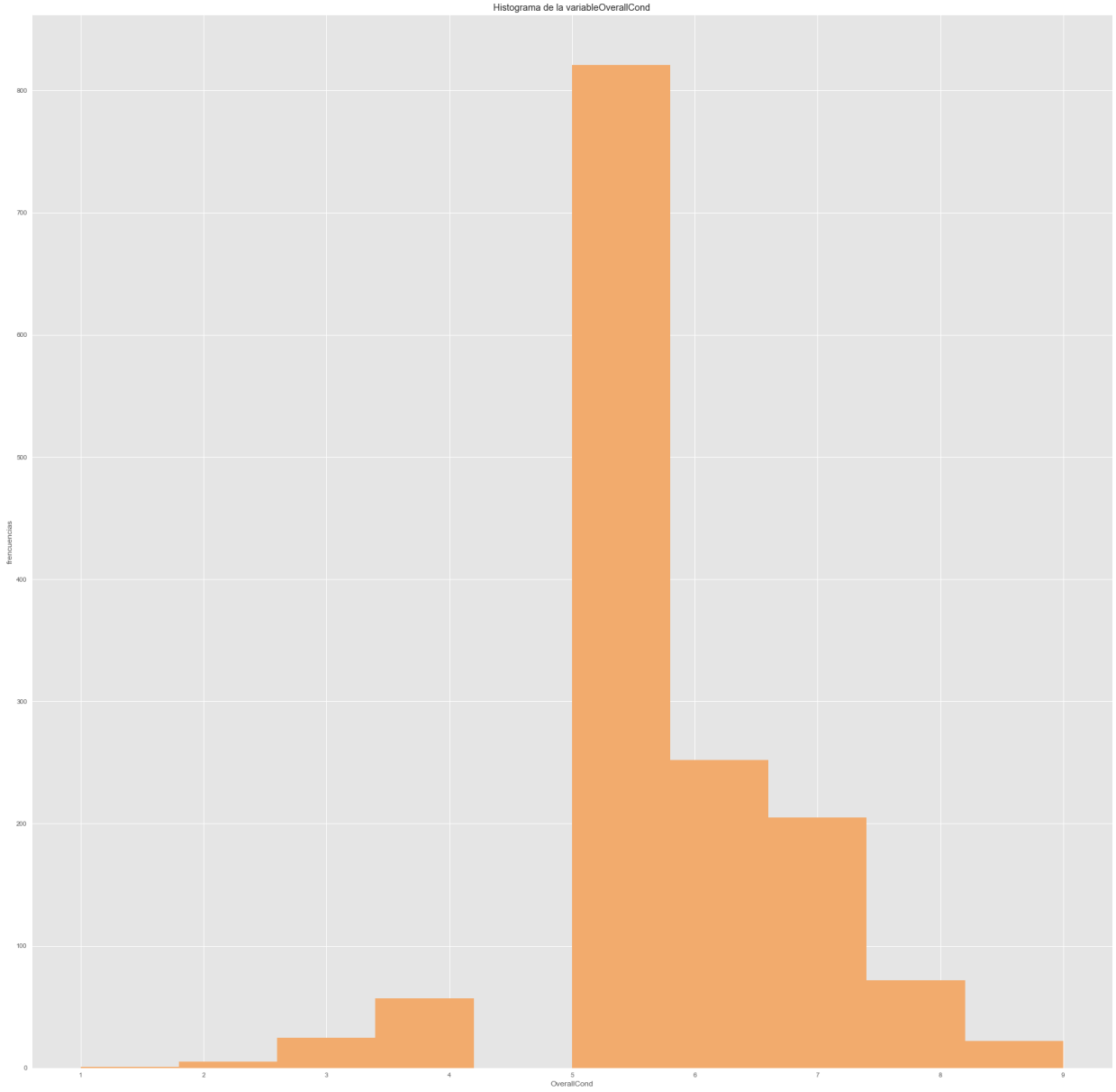


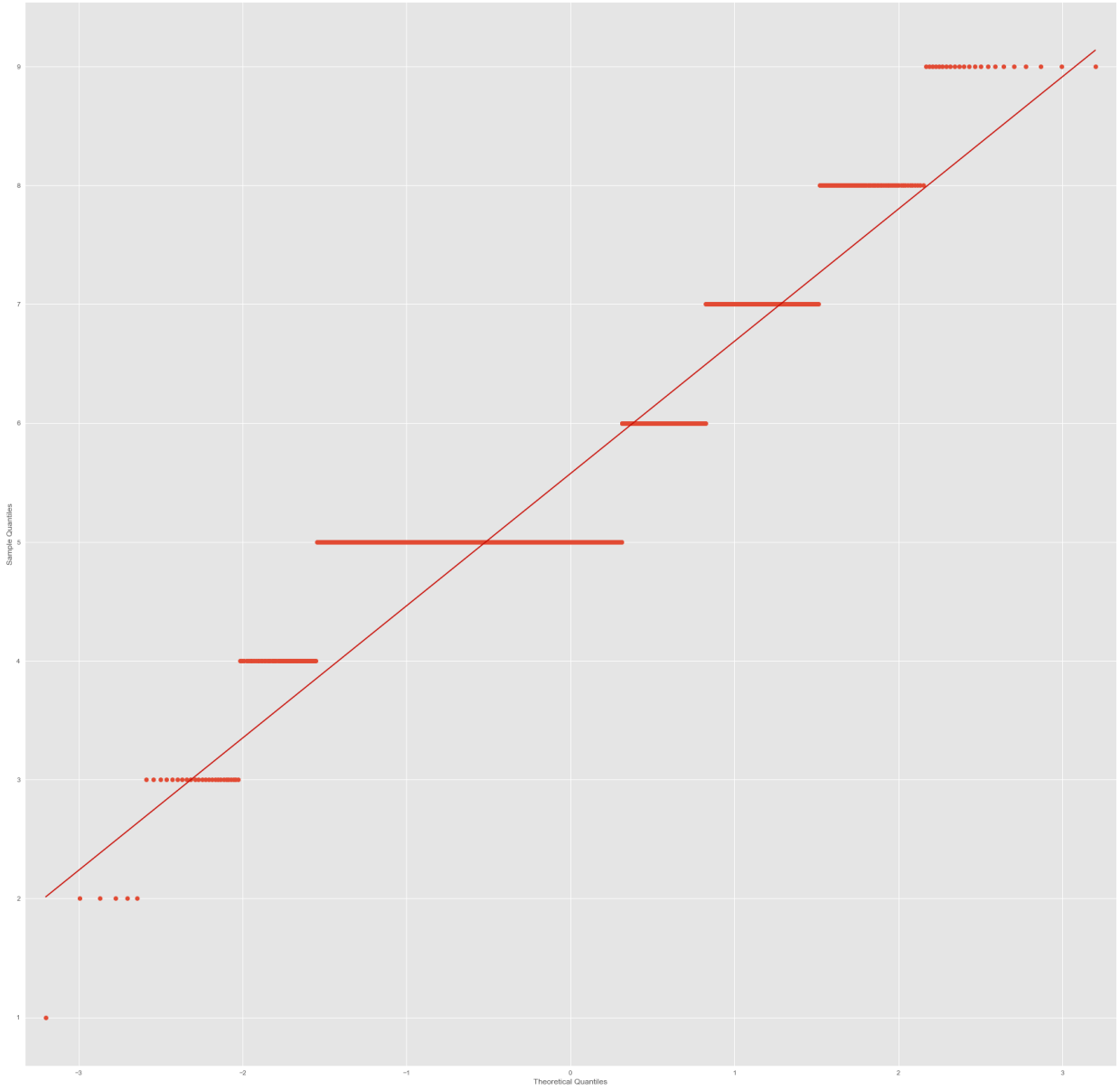


OverallCond

Se puede determinar que la variable OverallCond no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('OverallCond')
```

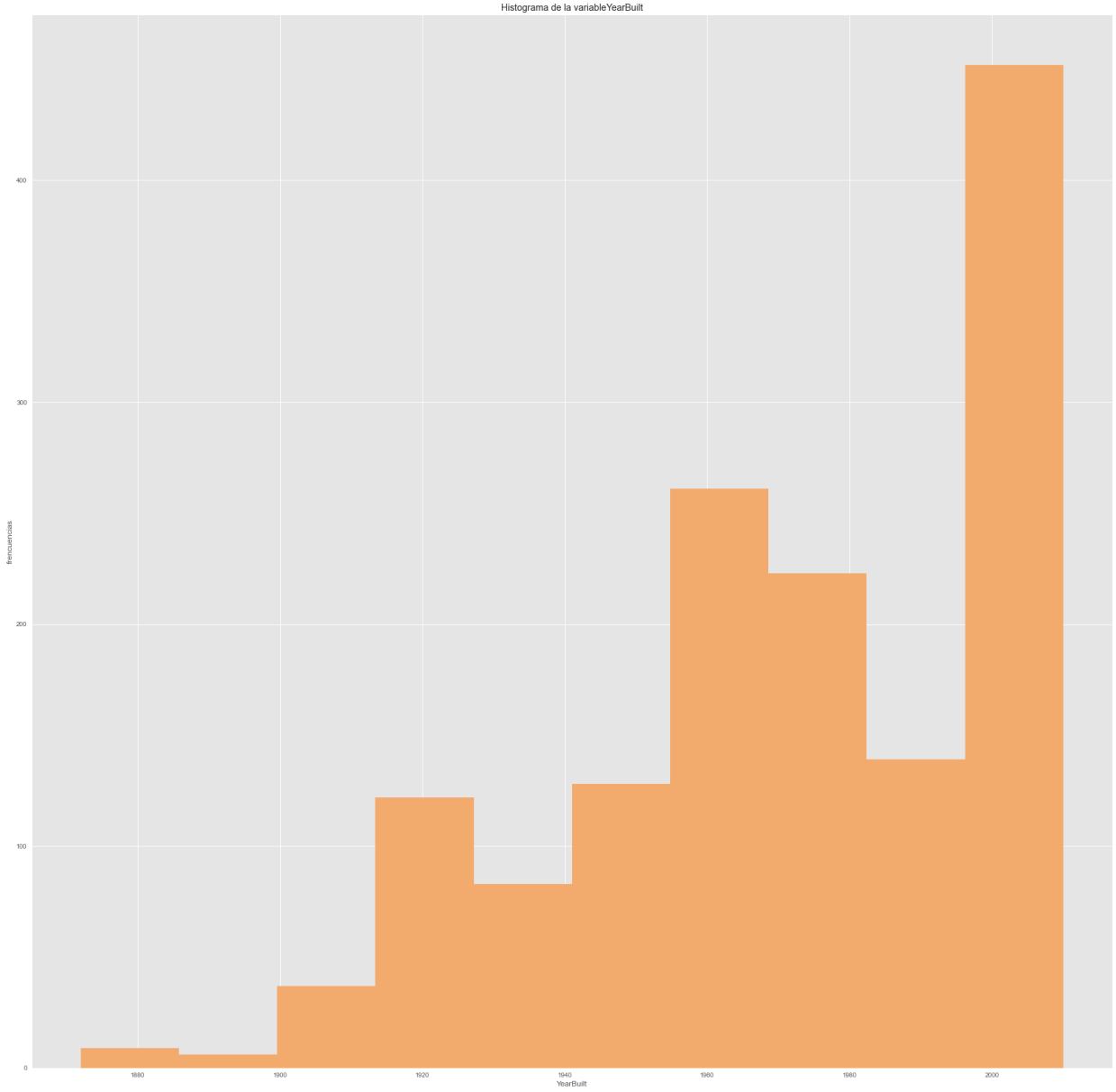

Histograma de la variableOverallCond
frencuencias
OverallCond
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6
7
8
9

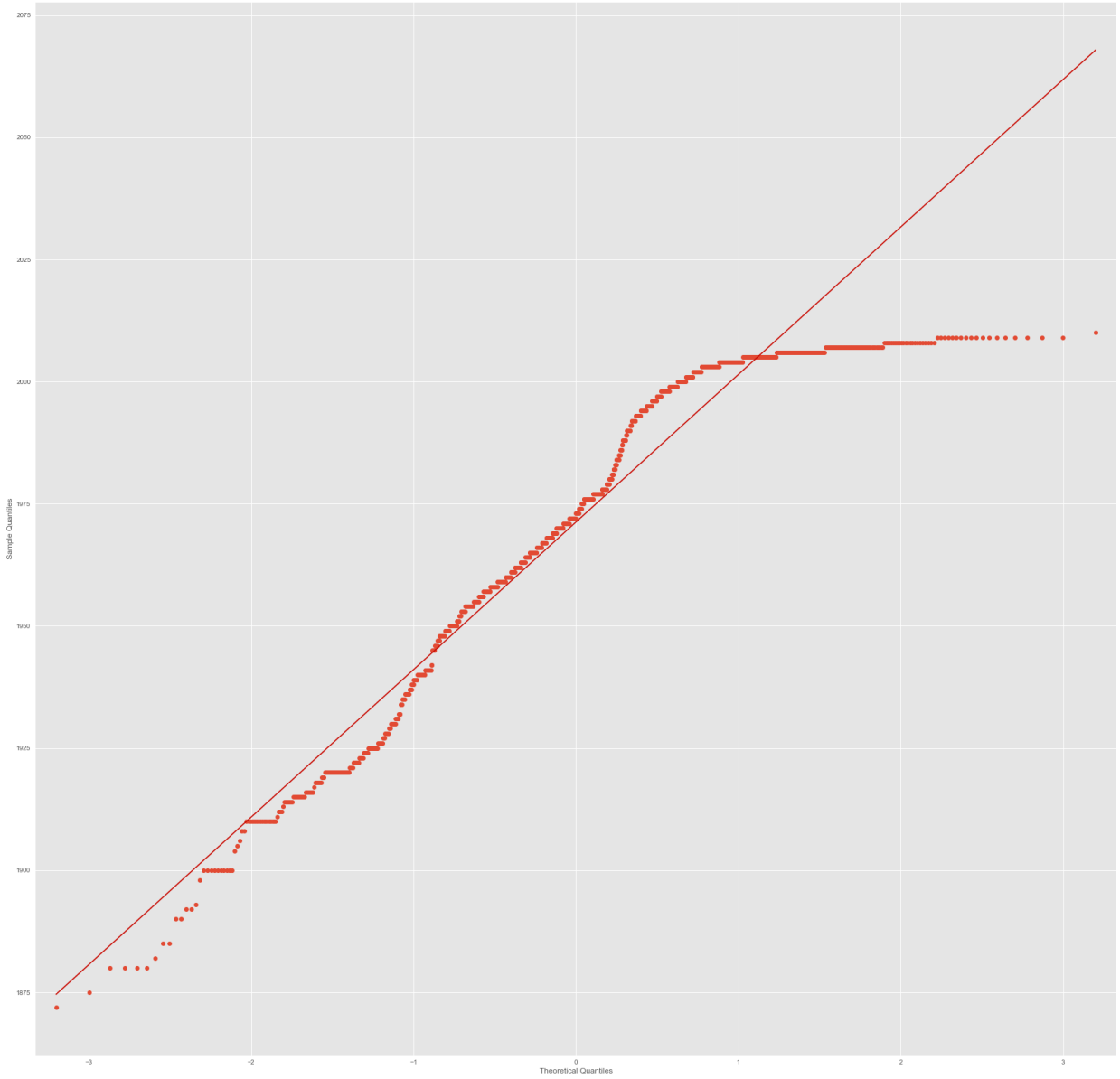


YearBuilt

Se puede determinar que la variable YearBuilt no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('YearBuilt')
```

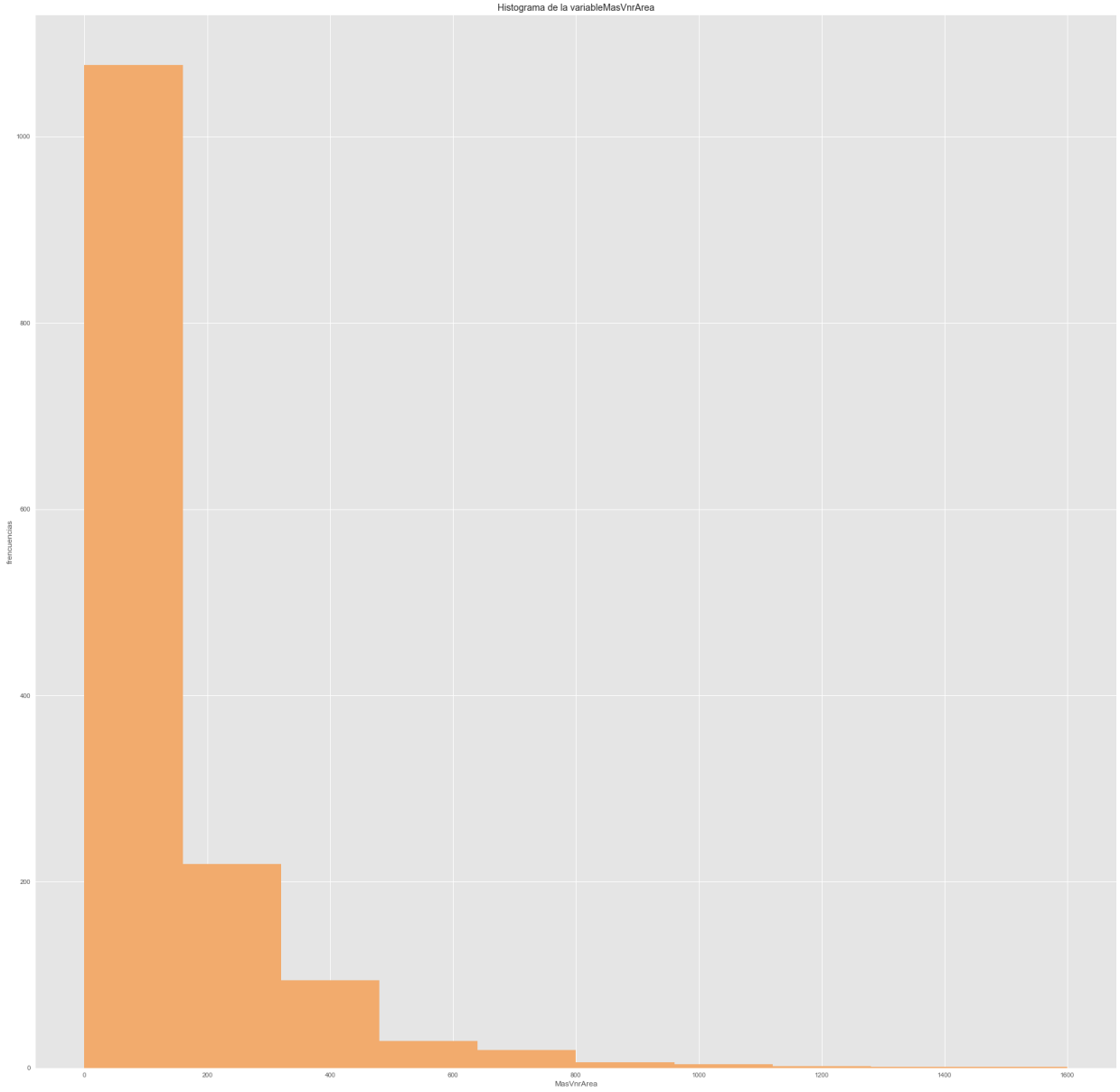

Histograma de la variableYearBuilt
frencuencias
0
100
200
300
400
1880
1900
1920
1940
1960
1980
2000
YearBuilt

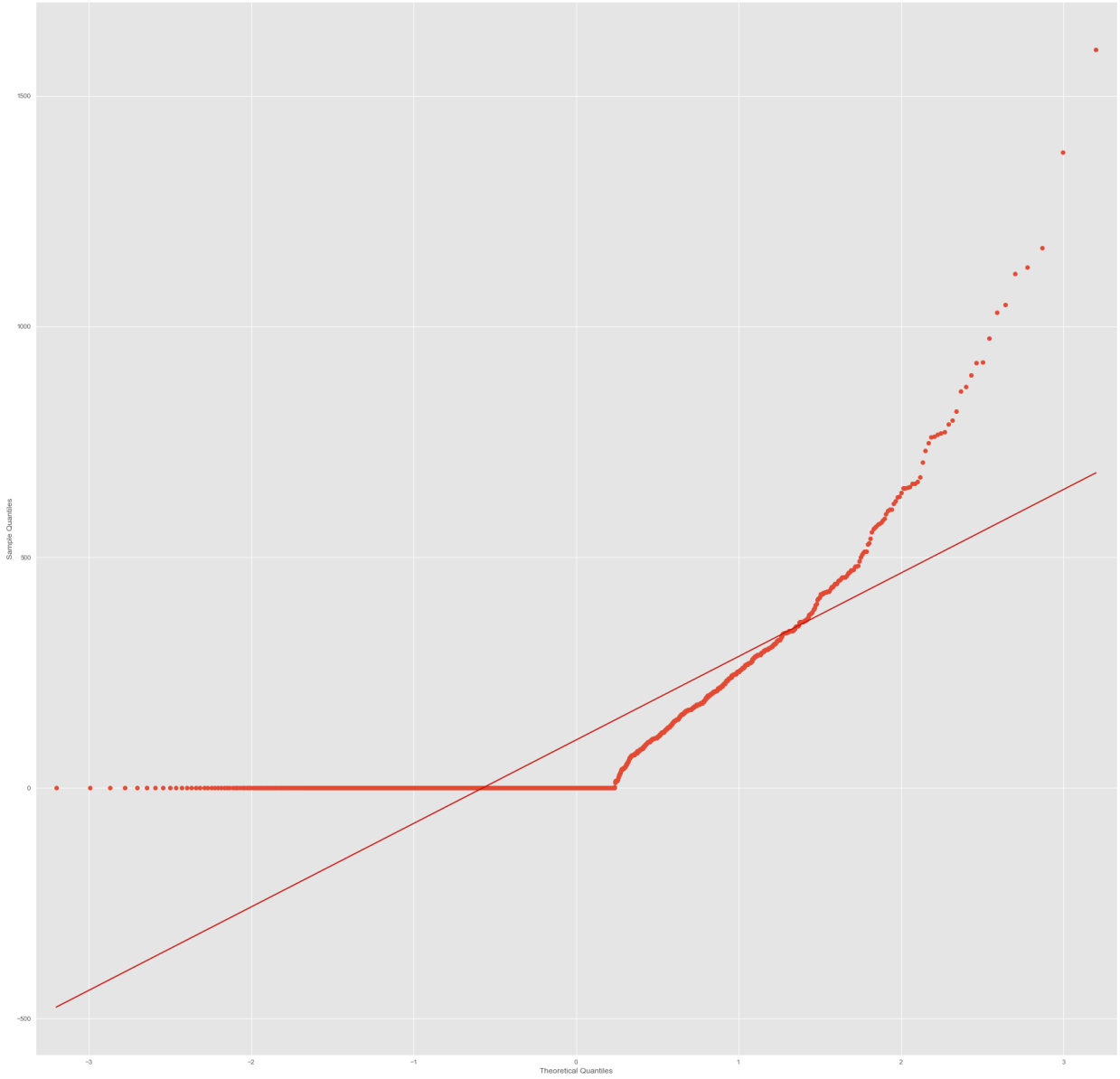


MasVnrArea

Se puede determinar que la variable MasVnrArea no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('MasVnrArea')
```

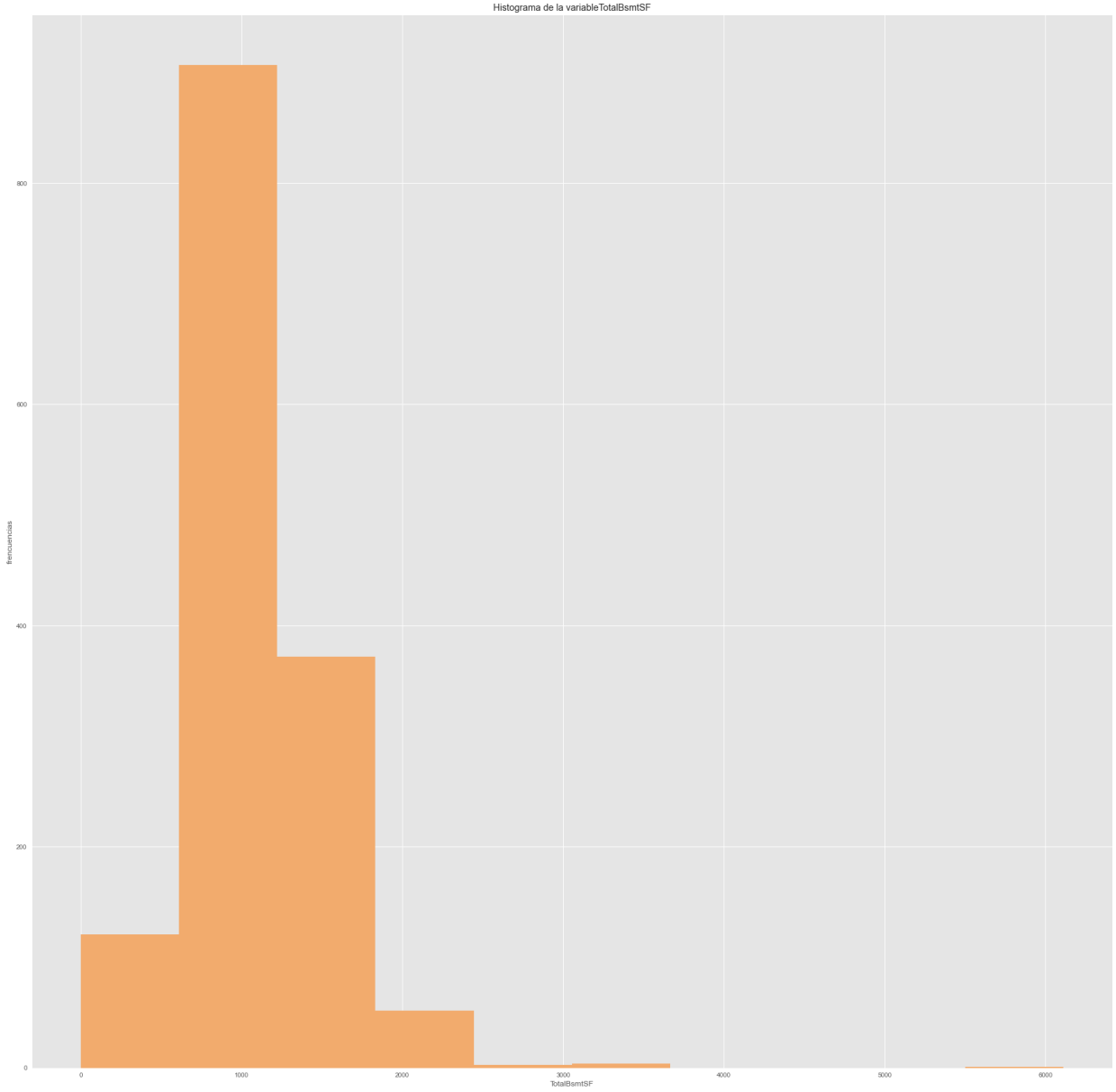

Histograma de la variableMasVnrArea
frecuencias
MasVnrArea
0
200
400
600
800
1000
1200
1400
1600

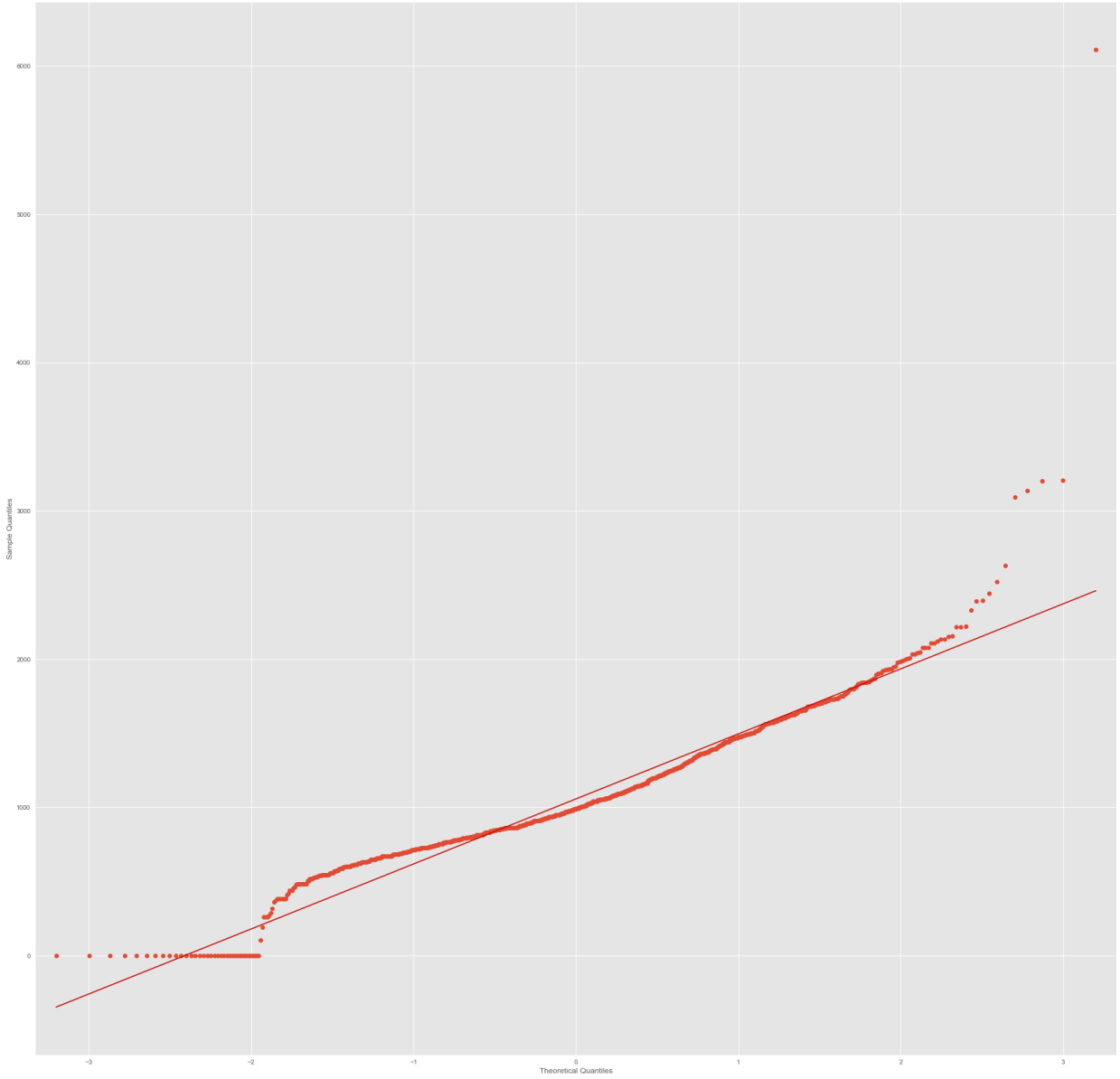


TotalBsmtSF

Se puede determinar que la variable TotalBsmtSF no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('TotalBsmtSF')
```

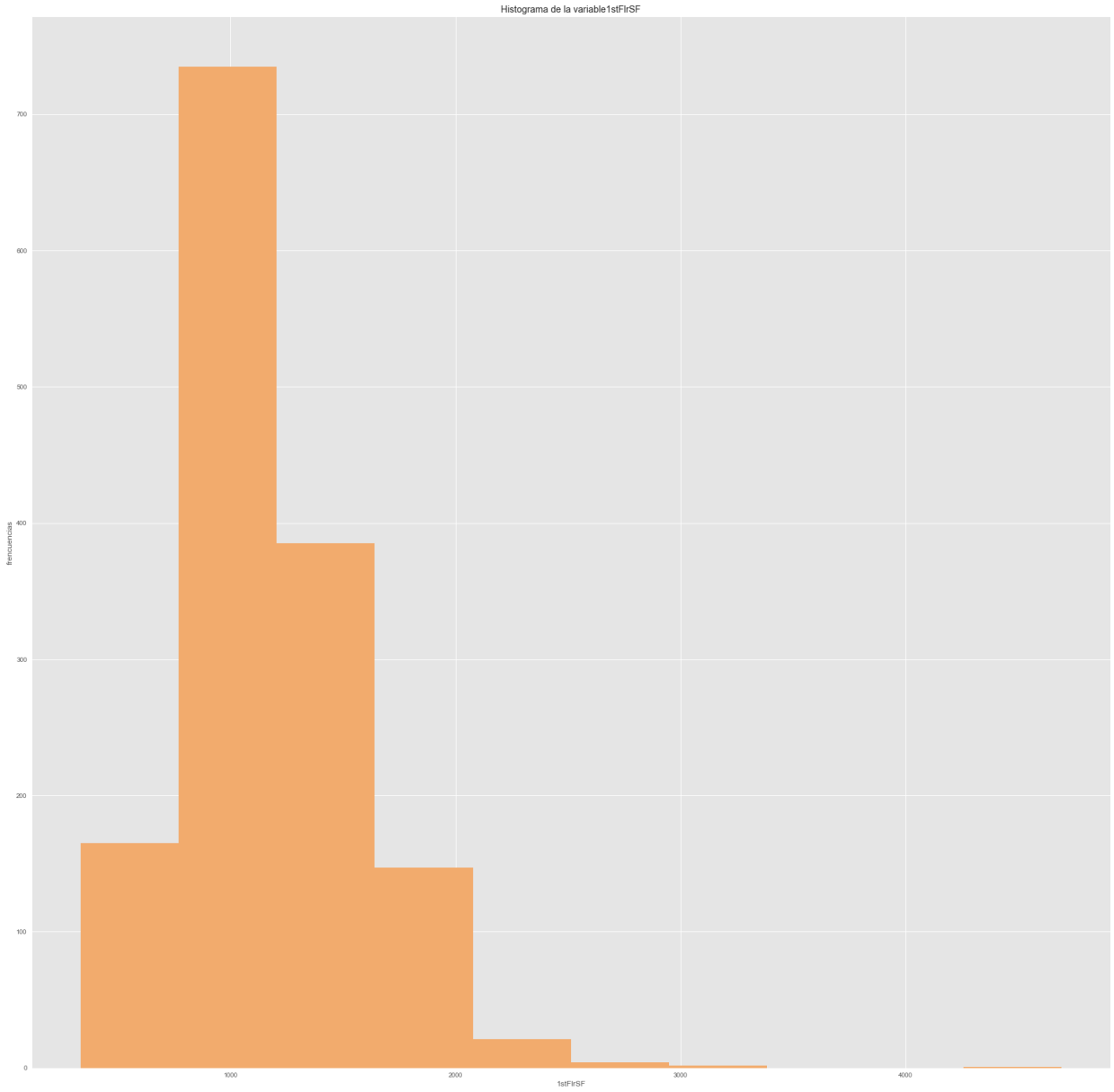

Histograma de la variableTotalBsmtSF
frecuencias
0
200
400
600
800
0
1000
2000
3000
4000
5000
6000
TotalBsmtSF

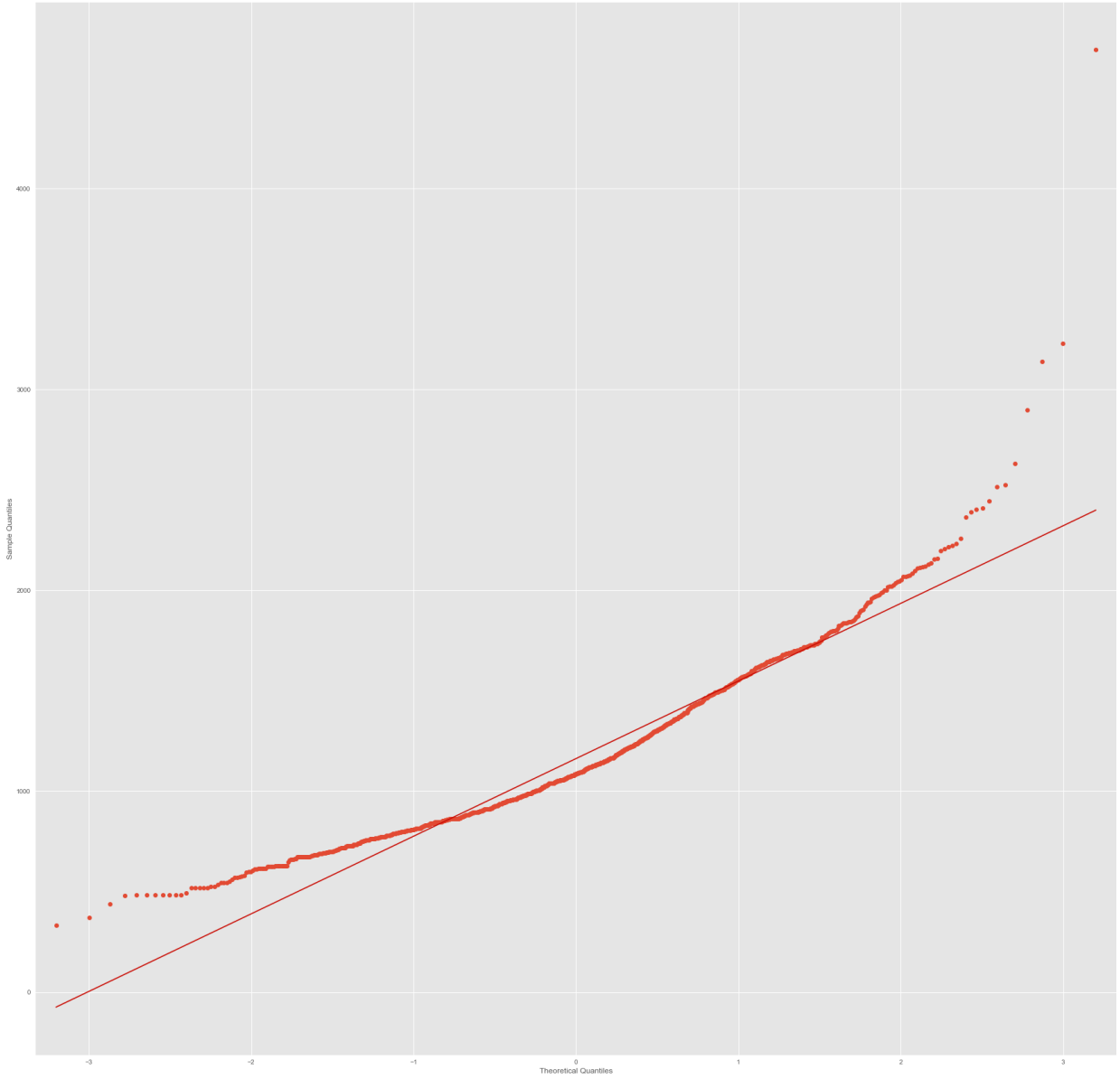


1stFlrSF

Se puede determinar que la variable 1stFlrSF no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('1stFlrSF')
```

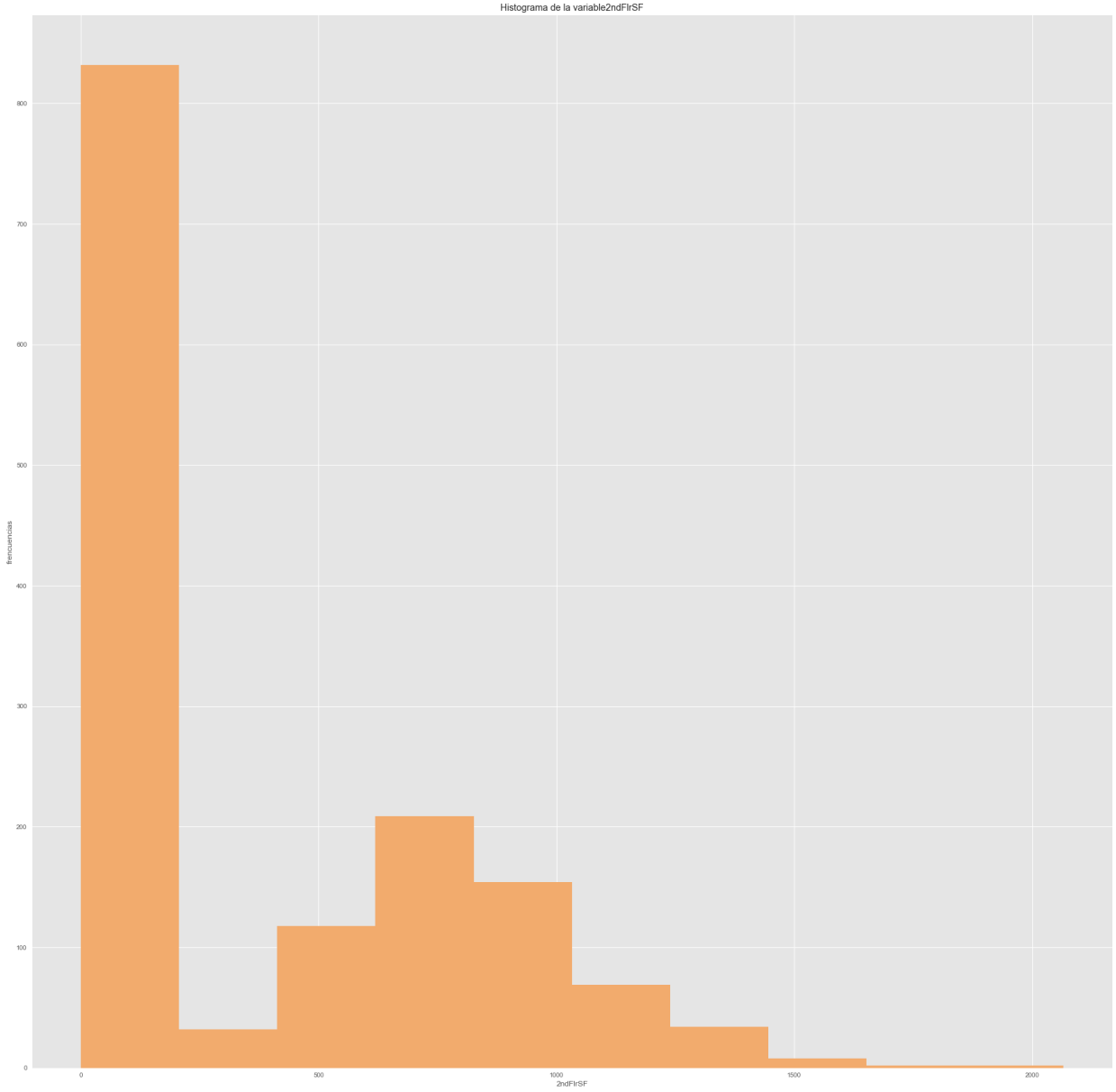

Histograma de la variable1stFlrSF
frencuencias
0
100
200
300
400
500
600
700
1000
2000
3000
4000
1stFlrSF

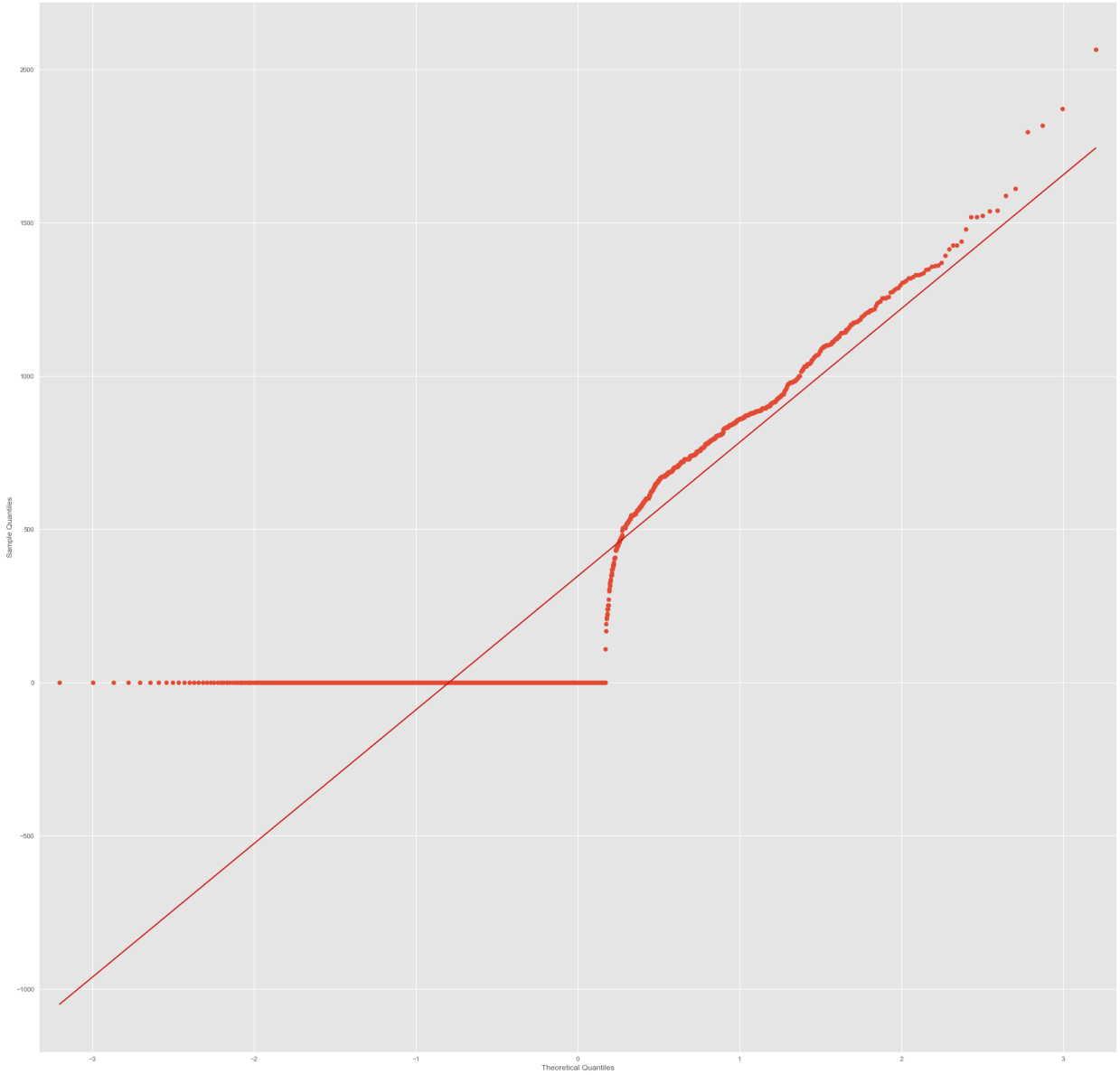


2ndFlrSF

Se puede determinar que la variable 2ndFlrSF no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('2ndFlrSF')
```

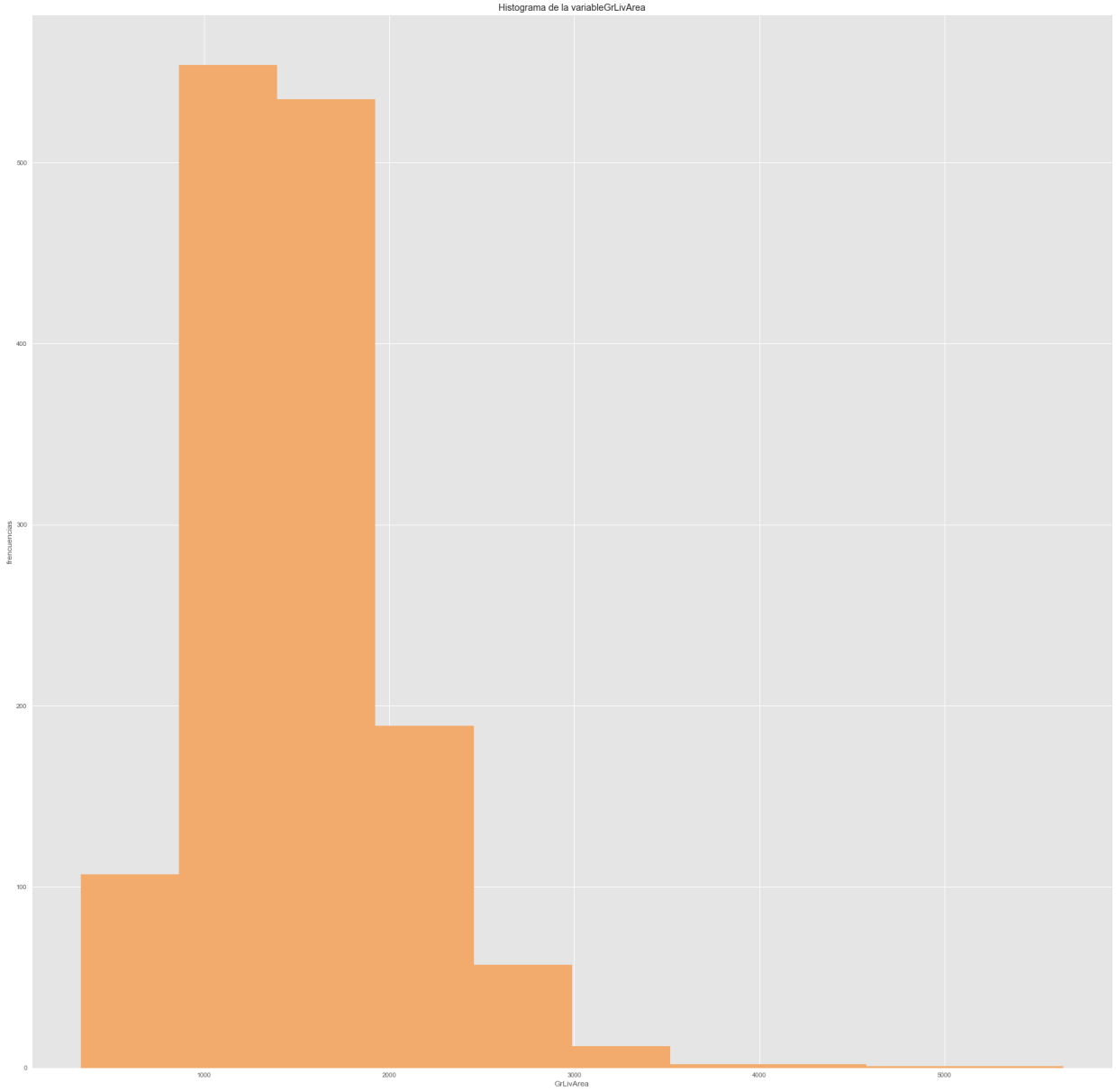

Histograma de la variable2ndFlrSF
frencuencias
0
100
200
300
400
500
600
700
800
0
500
1000
1500
2000
2ndFlrSF

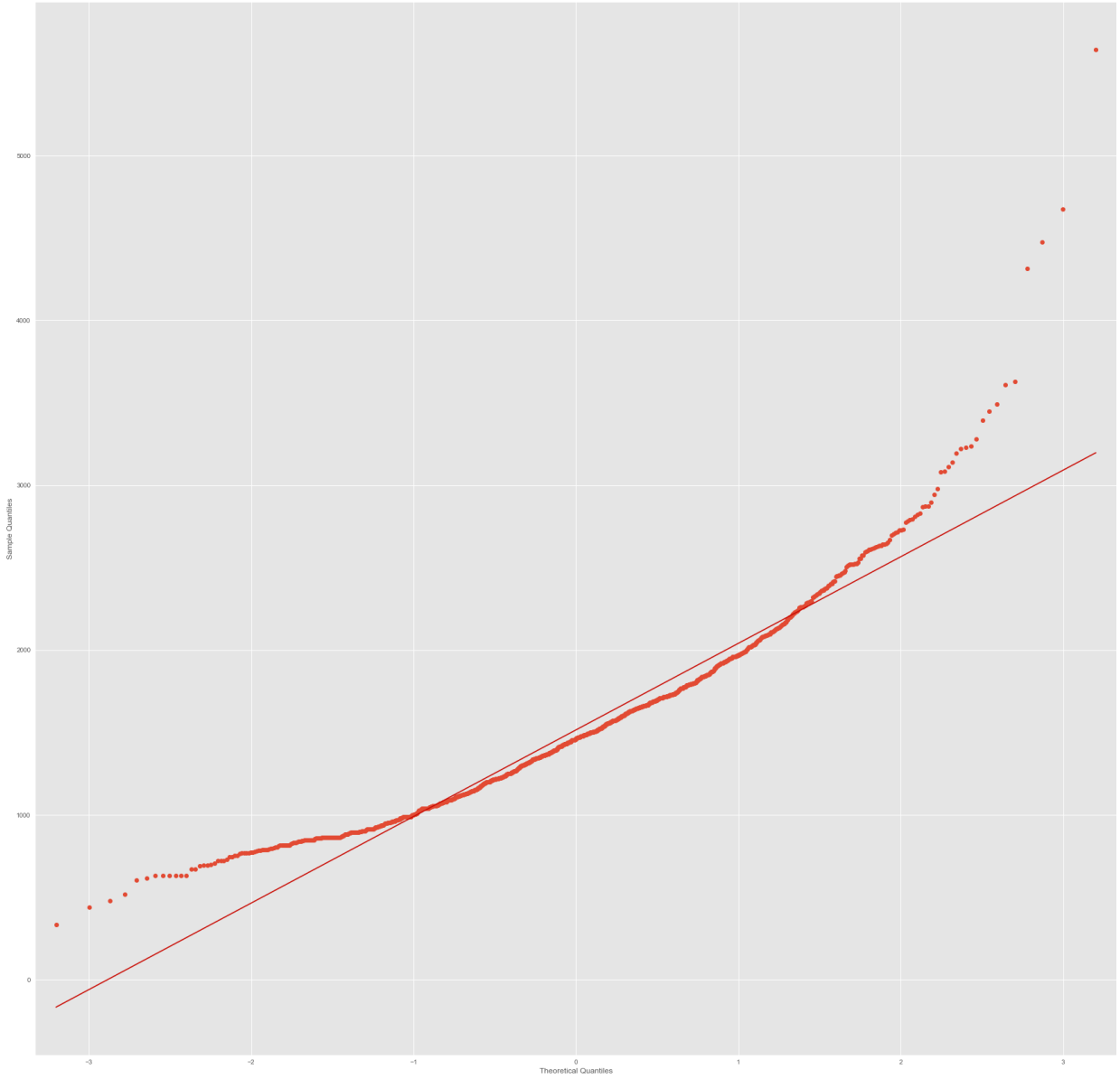


GrLivArea

Se puede determinar que la variable GrLivArea no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('GrLivArea')
```

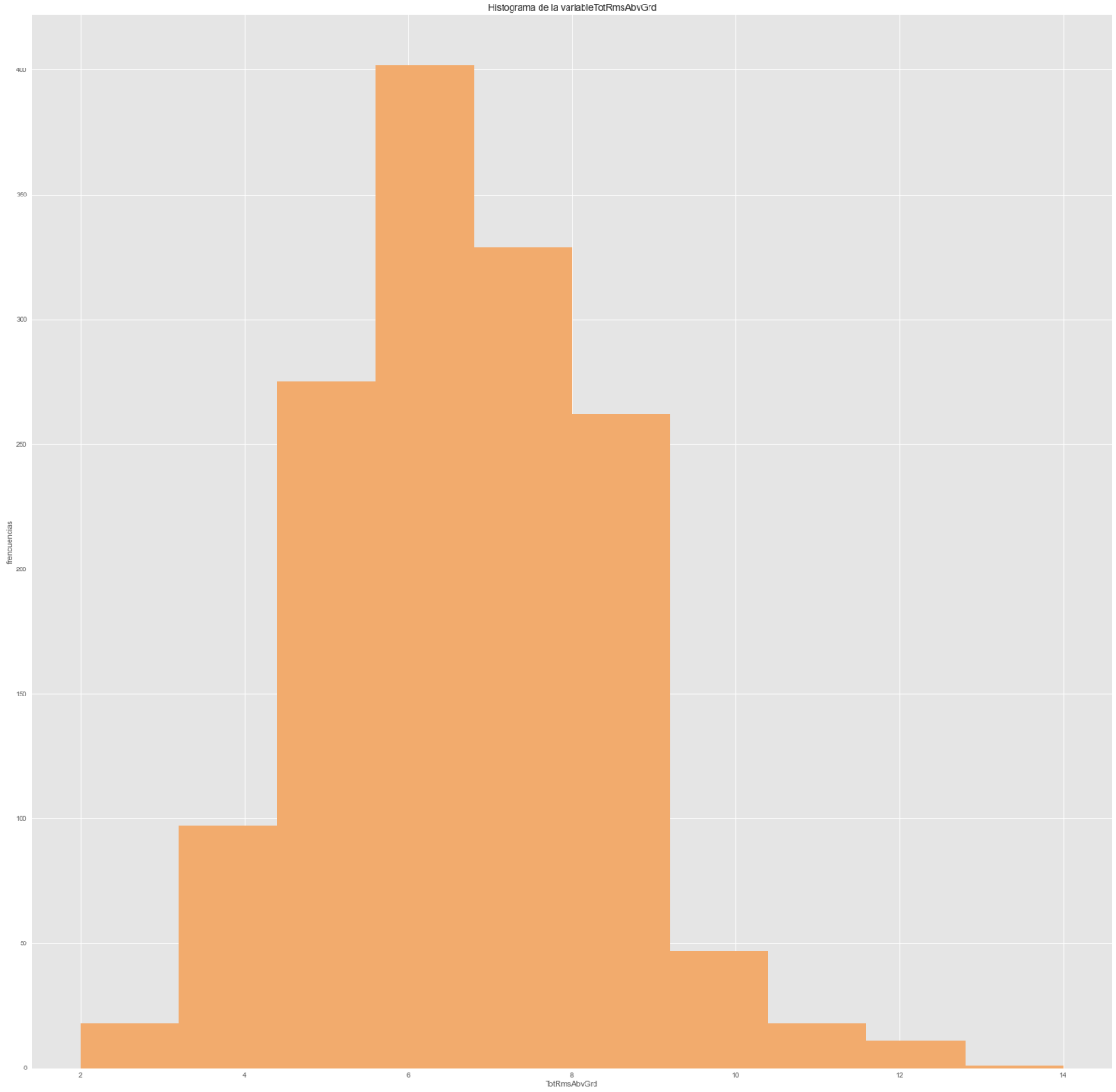

Histograma de la variableGrLivArea
frencuencias
0
100
200
300
400
500
1000
2000
3000
4000
5000
GrLivArea

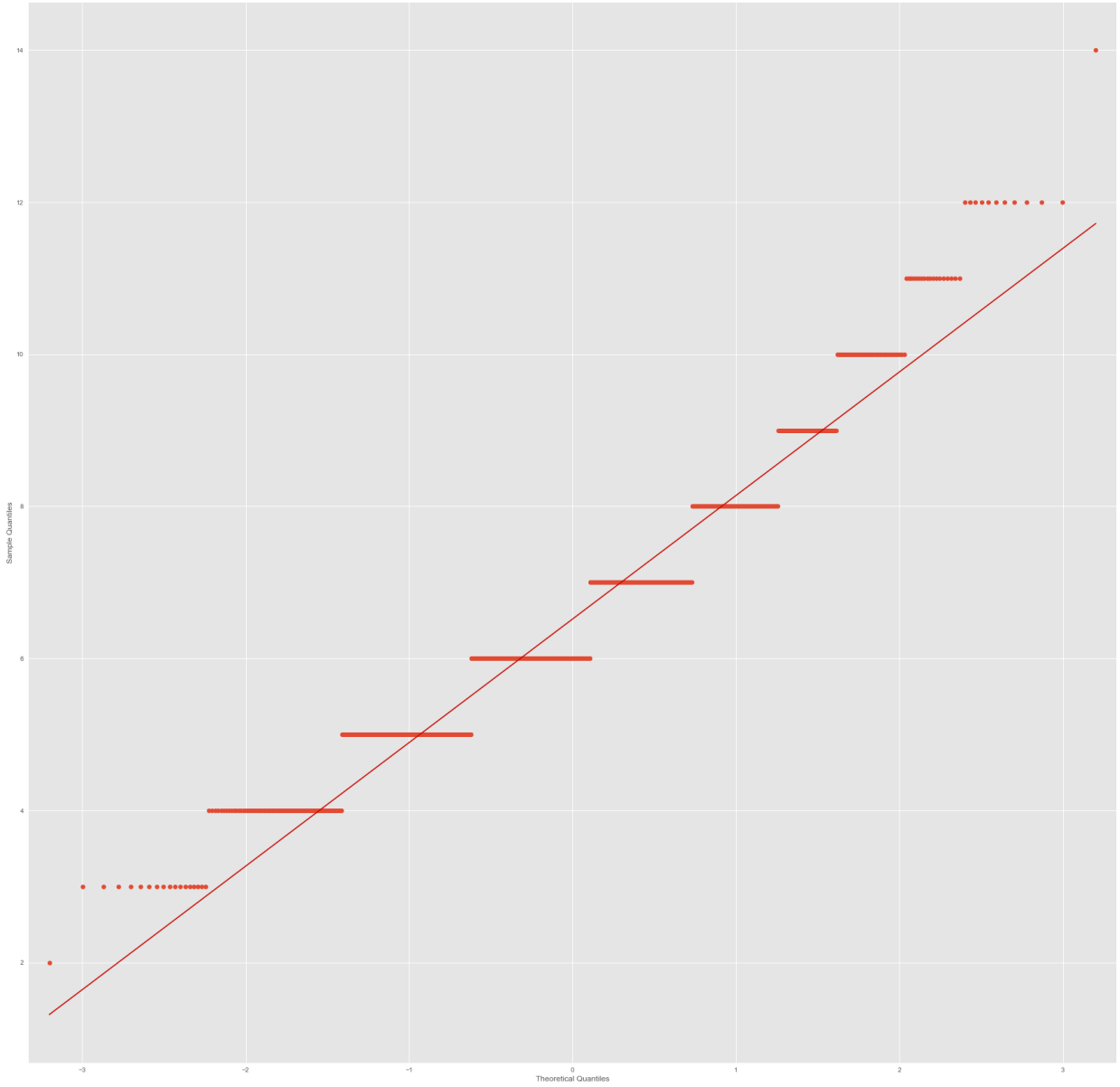


TotRmsAbvGrd

Se puede determinar que la variable TotRmsAbvGrd no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('TotRmsAbvGrd')
```

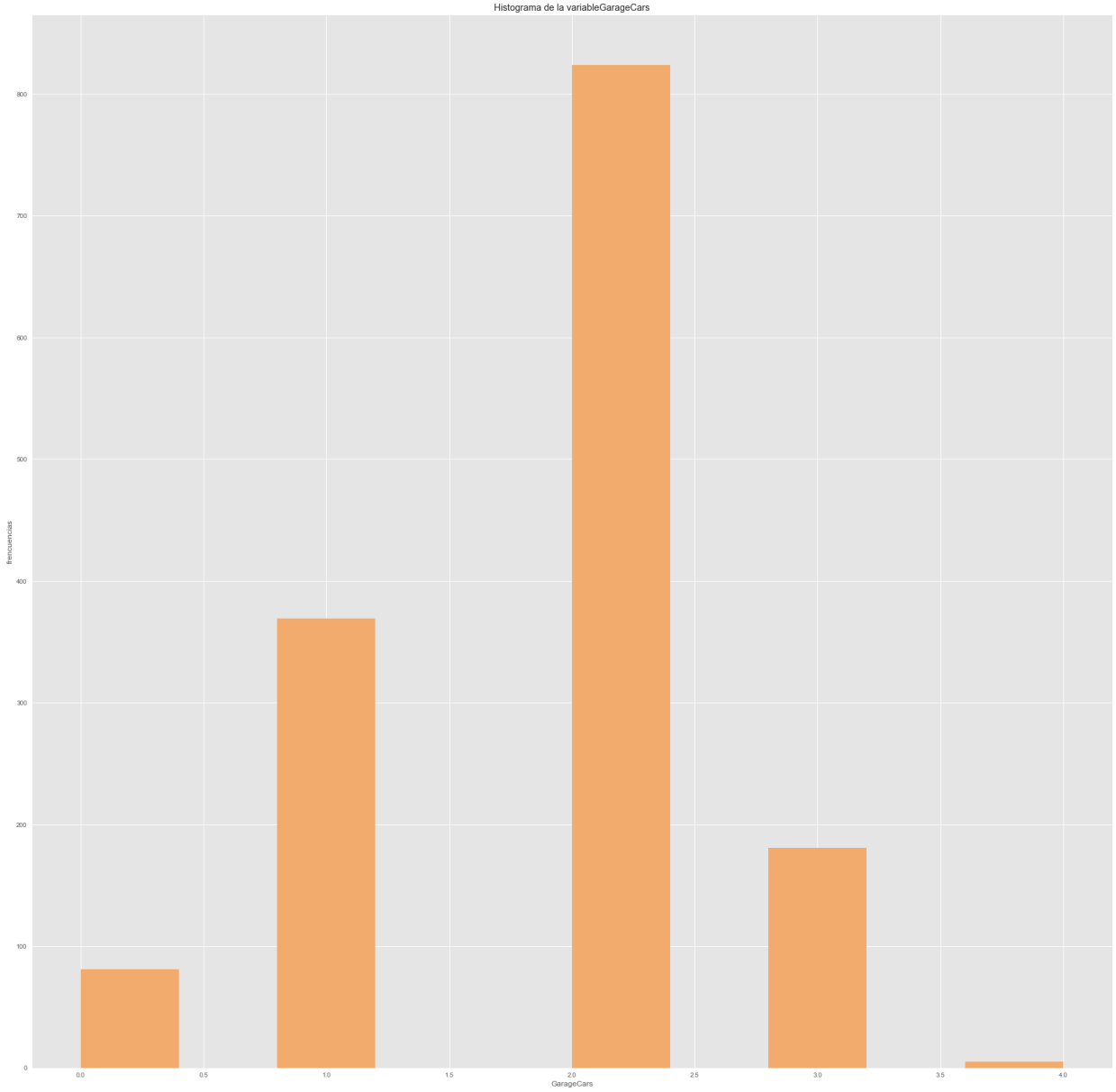

Histograma de la variableTotRmsAbvGrd
frencuencias
TotRmsAbvGrd
0
50
100
150
200
250
300
350
400
2
4
6
8
10
12
14

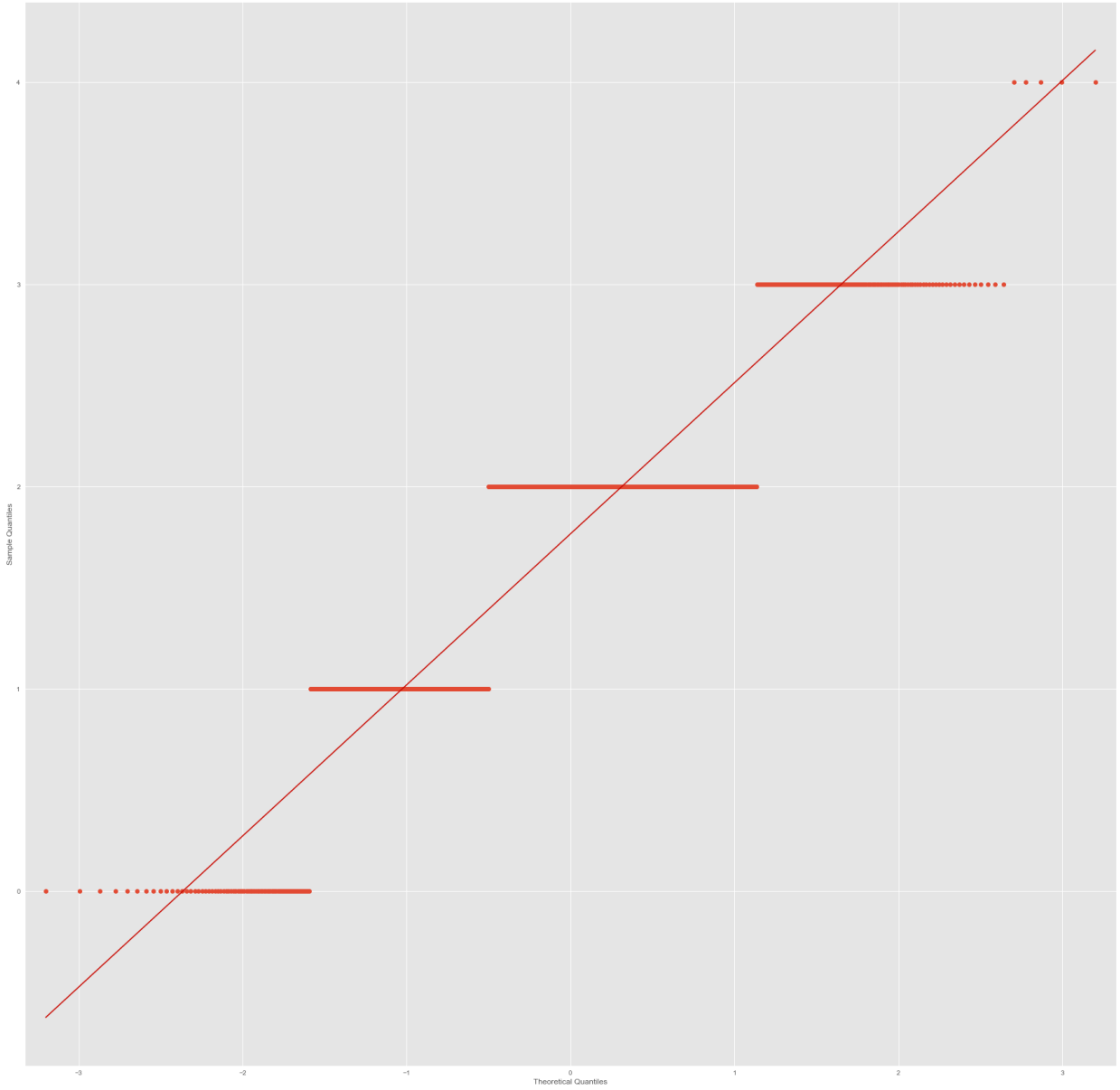


GarageCars

Se puede determinar que la variable GarageCars no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('GarageCars')
```

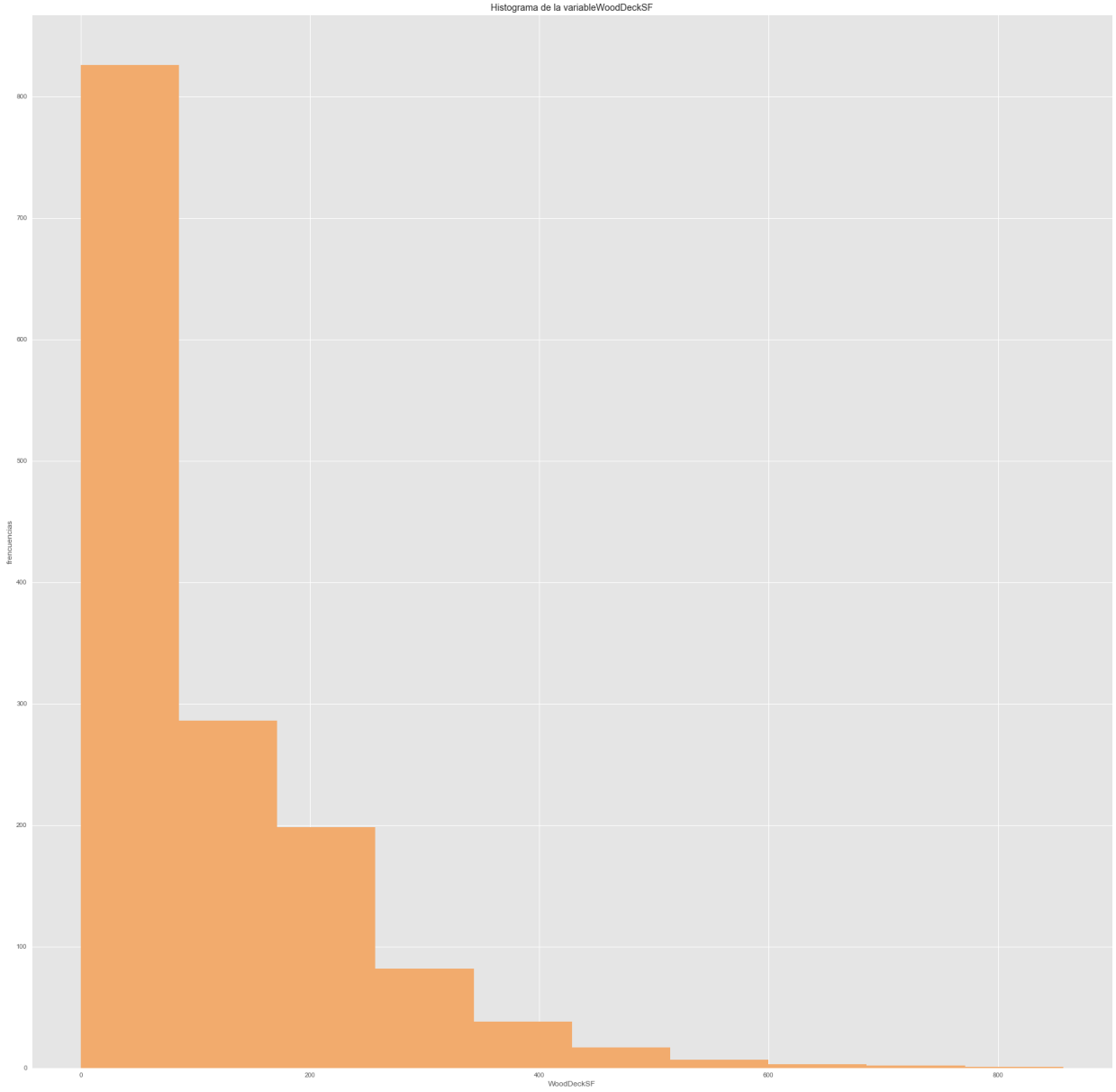

Histograma de la variableGarageCars
frencuencias
GarageCars
0
100
200
300
400
500
600
700
800
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

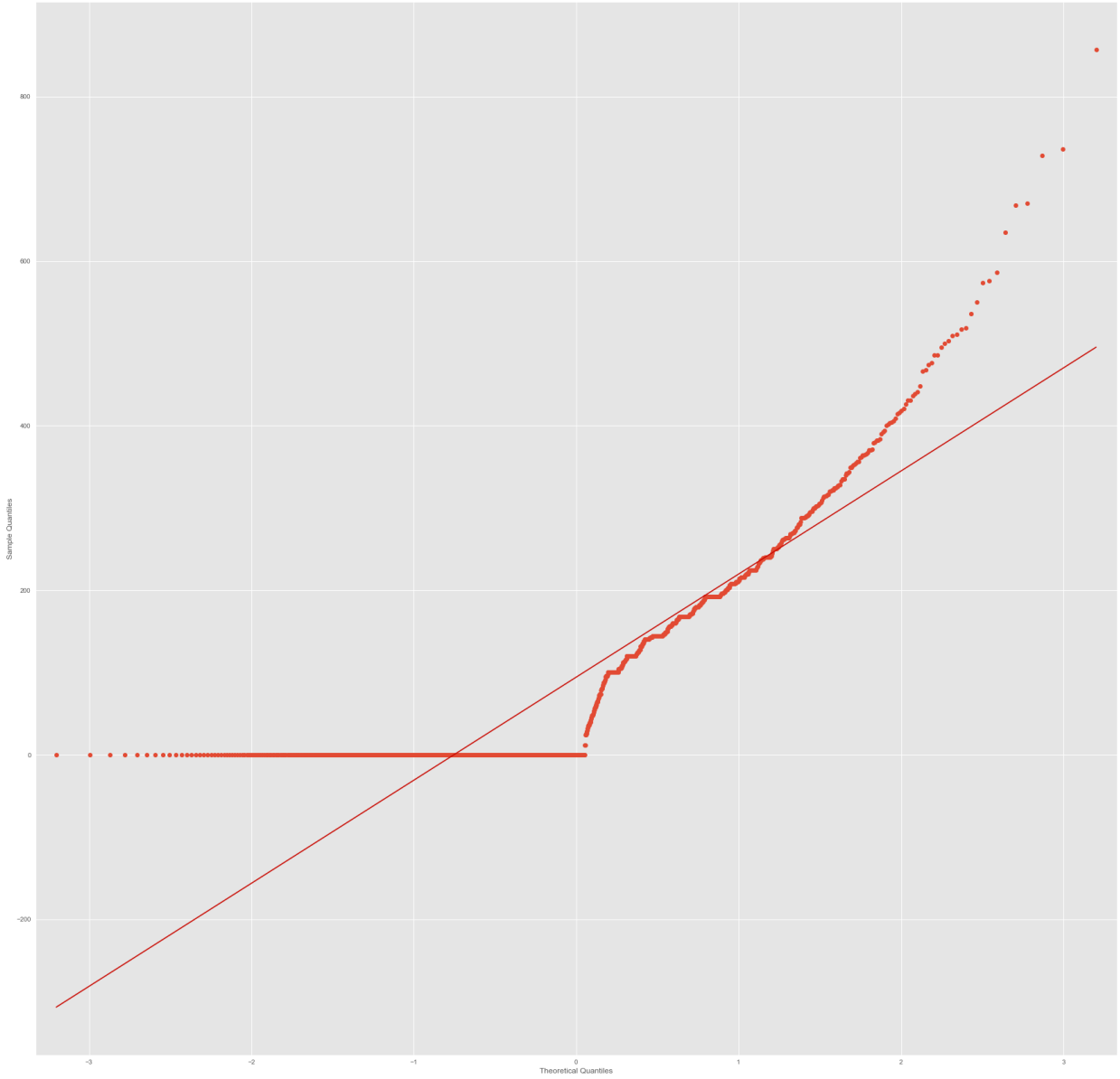


WoodDeckSF

Se puede determinar que la variable WoodDeckSF no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('WoodDeckSF')
```

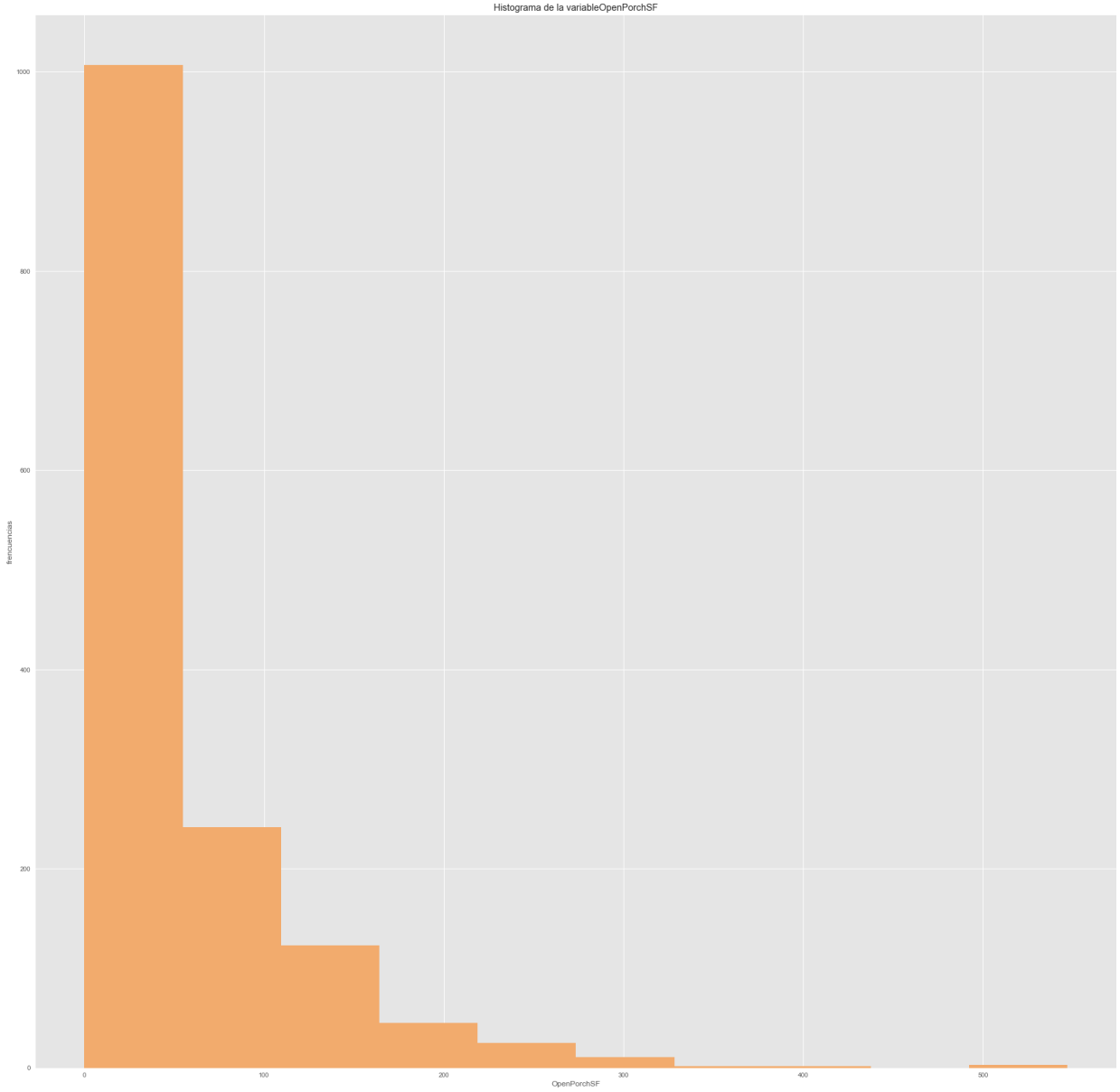

Histograma de la variableWoodDeckSF
frencuencias
0
100
200
300
400
500
600
700
800
0
200
400
600
800
WoodDeckSF

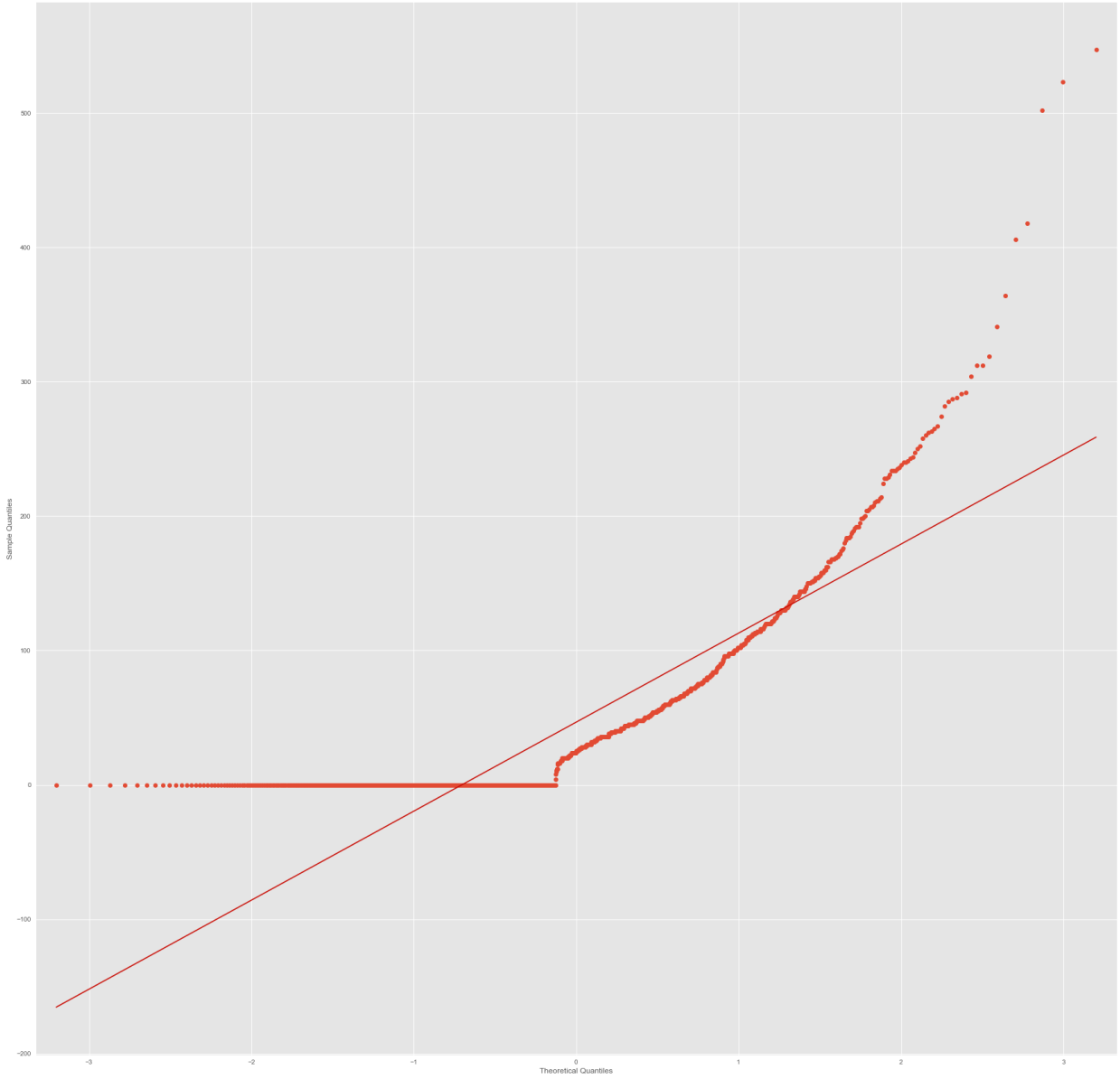


OpenPorchSF

Se puede determinar que la variable OpenPorchSF no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('OpenPorchSF')
```

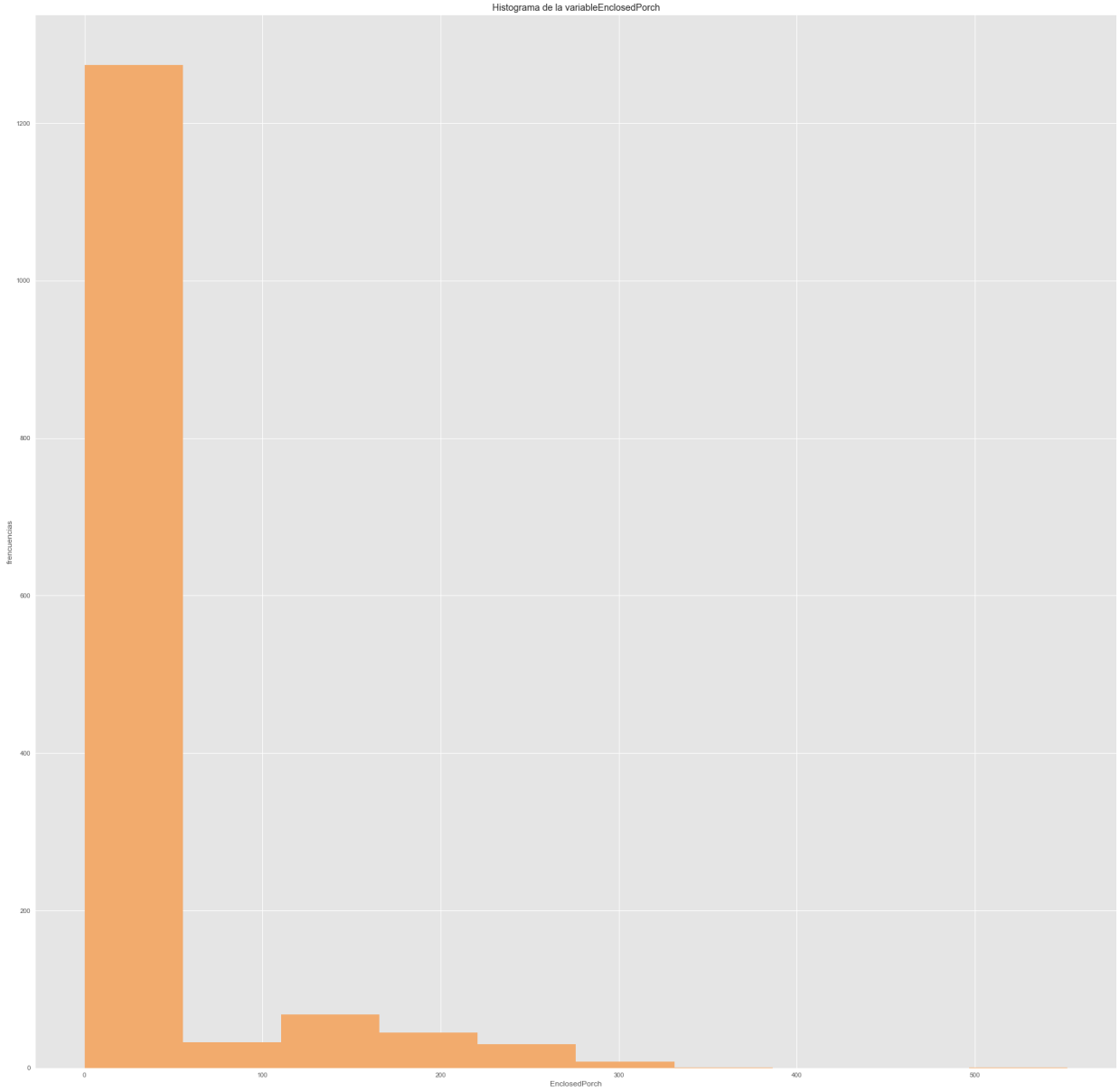

Histograma de la variableOpenPorchSF
frencuencias
OpenPorchSF
0
200
400
600
800
1000
0
100
200
300
400
500

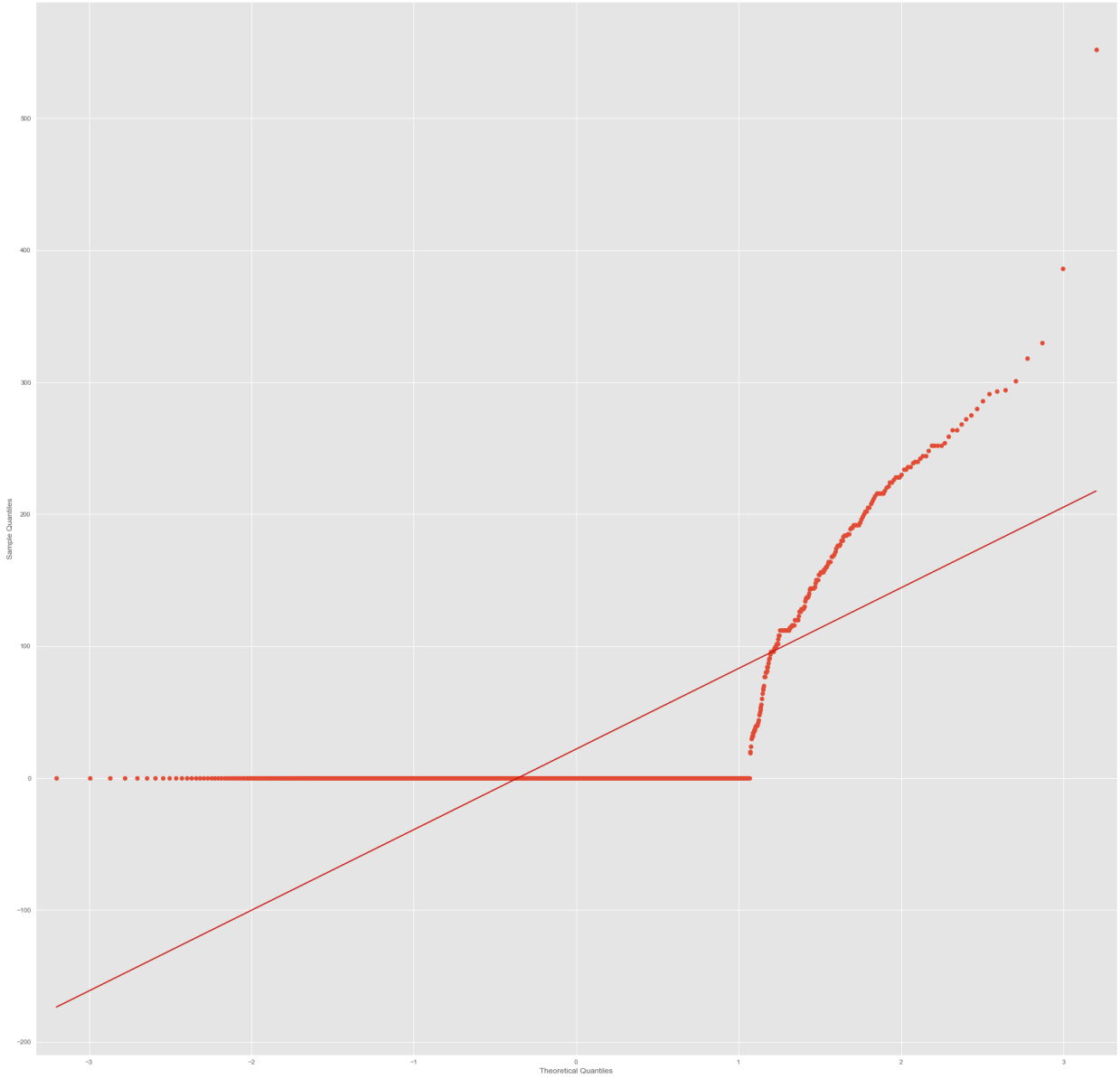


EnclosedPorch

Se puede determinar que la variable EnclosedPorch no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('EnclosedPorch')
```

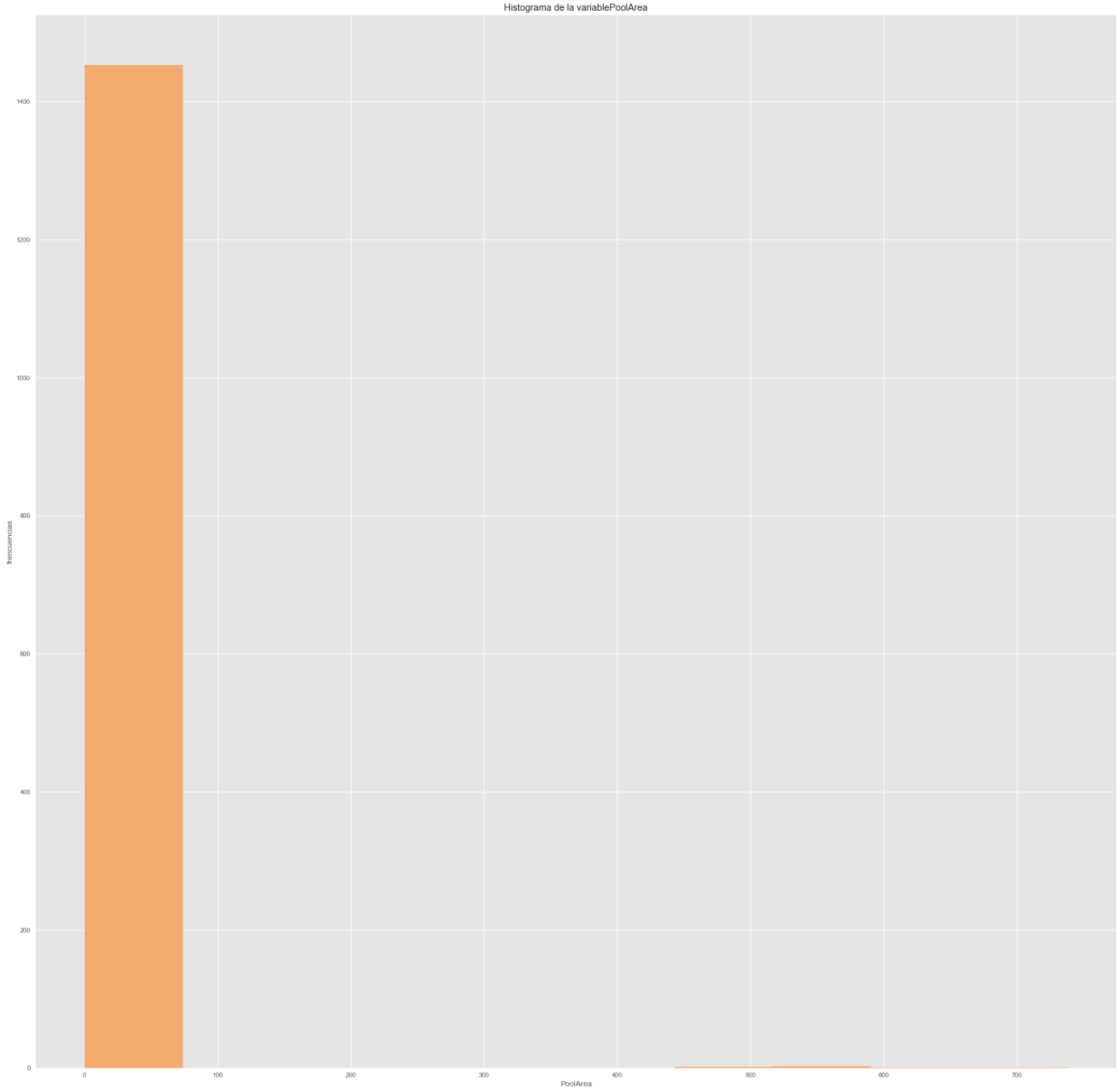

Histograma de la variableEnclosedPorch
frecuencias
EnclosedPorch
0
200
400
600
800
1000
1200
0
100
200
300
400
500

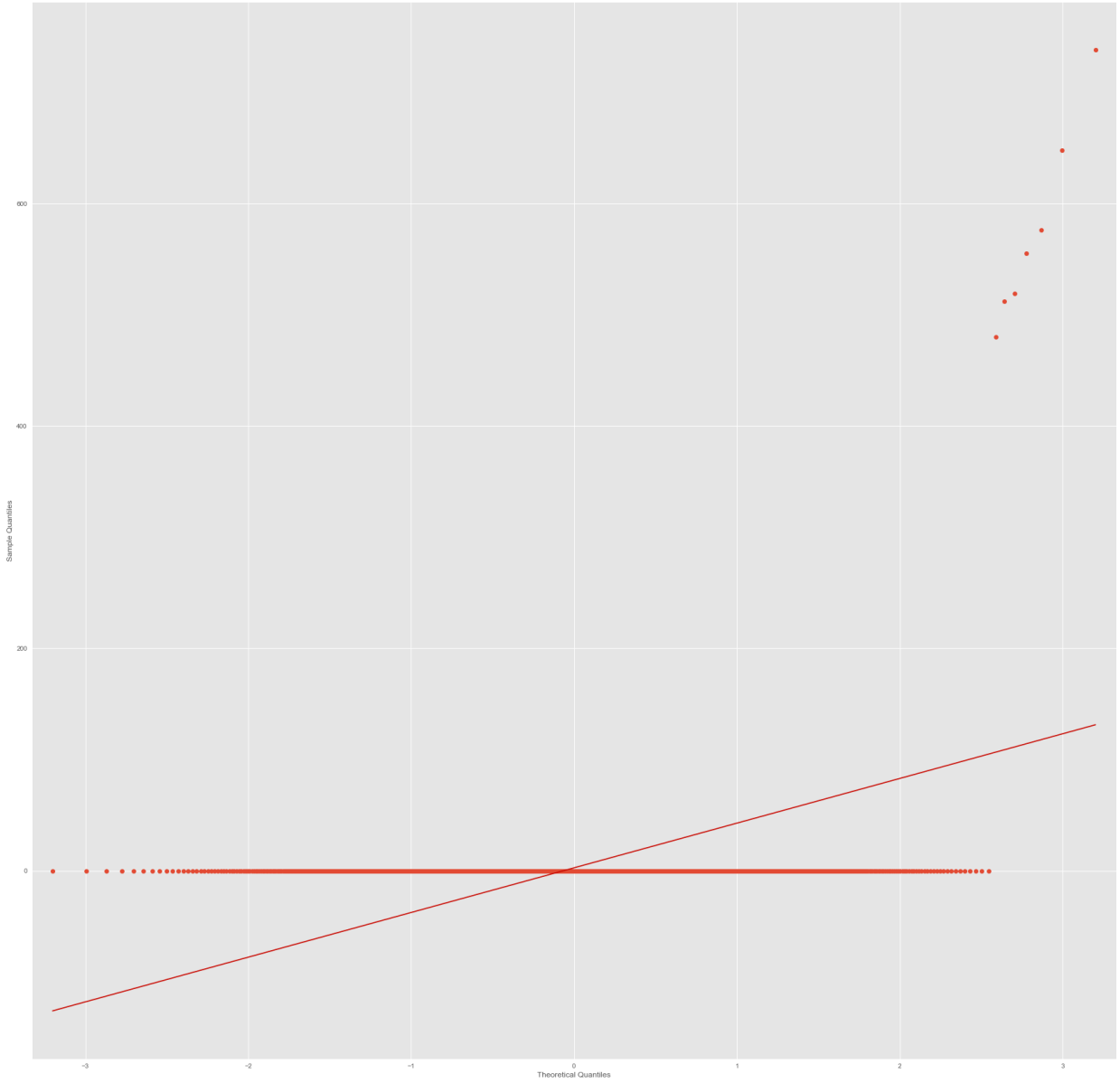


PoolArea

Se puede determinar que la variable PoolArea no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
get_histogram_qq('PoolArea')
```

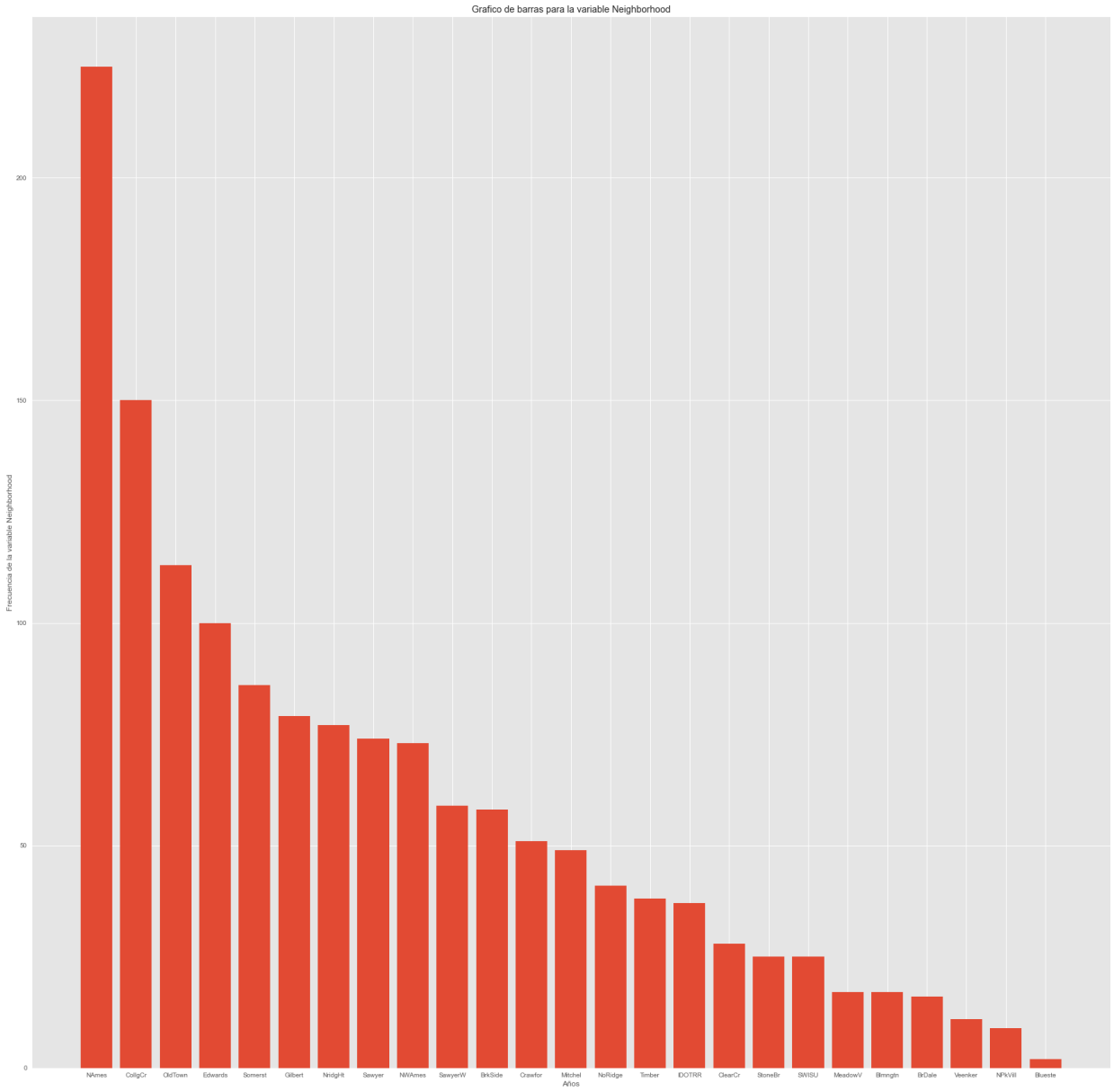

Histograma de la variablePoolArea
frencuencias
0
200
400
600
800
1000
1200
1400
0
100
200
300
400
500
600
700
PoolArea



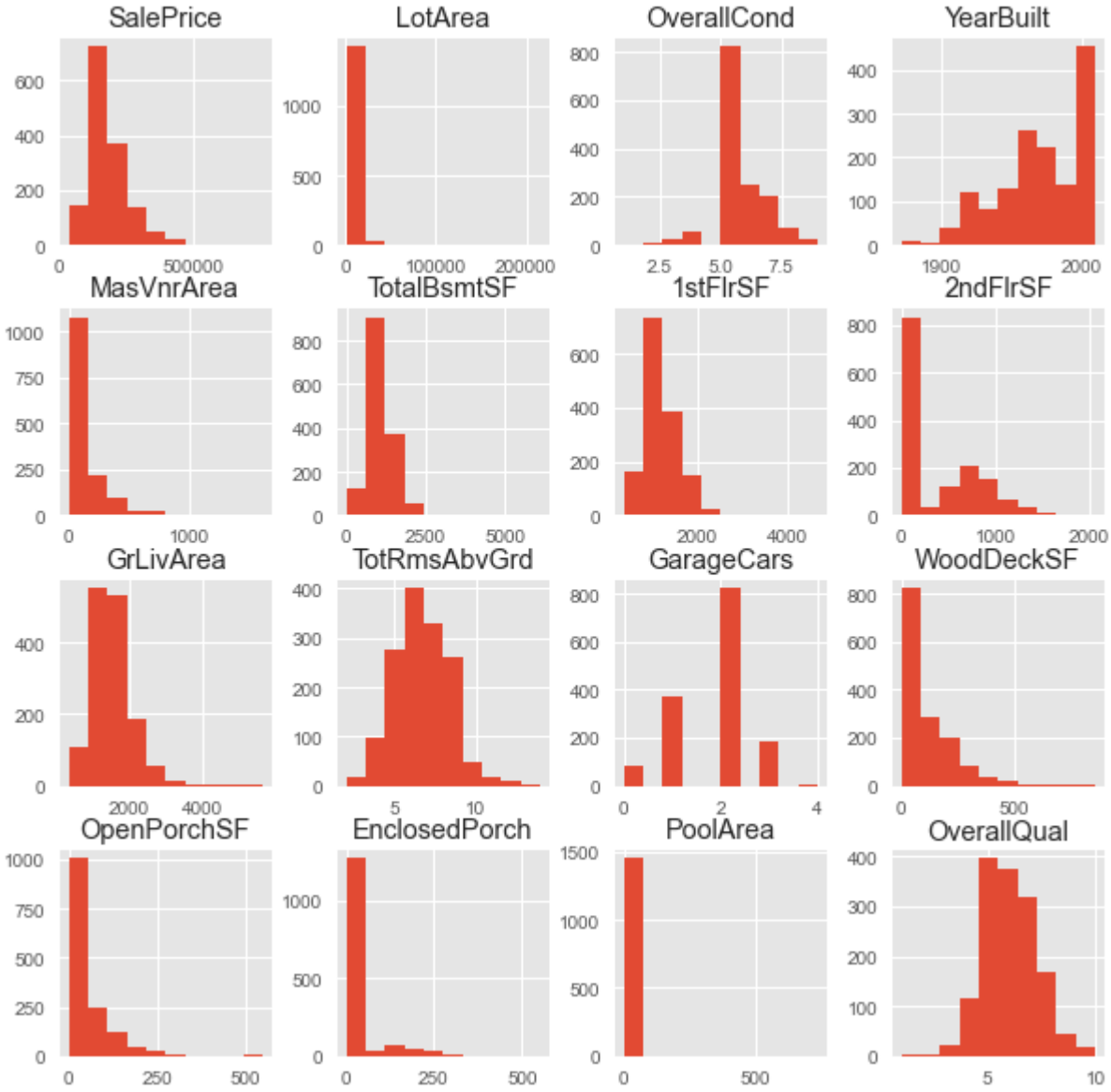
Neighborhood

Se puede determinar que la variable Neighborhood no sigue una disctribucion normal debido a que el histograma no sigue una forma de campana y el diagrama QQ nos muestra que los datos son muy distintos.

In [ ]:
```
eje_x = np.array(pd.value_counts(data['Neighborhood']).keys())
eje_y = pd.value_counts(data['Neighborhood'])

plt.bar(eje_x, eje_y)
plt.rcParams['figure.figsize'] = (10, 10)
plt.ylabel('Frecuencia de la variable Neighborhood')
plt.xlabel('Años')
plt.title('Grafico de barras para la variable Neighborhood')
plt.show()
```



In [ ]:
```
data.hist()
plt.show()
```



In [ ]:
```
# NORMALIZAMOS DATOS
if 'Neighborhood' in data.columns:
    usefullAttr.remove('Neighborhood')
data = train[usefullAttr]
X = []
for column in data.columns:
    try:
        column
        if column != 'Neighborhood' or column != 'SalePrice':
            data[column] = (data[column]-data[column].mean()) / \
                data[column].std()
            X.append(data[column])
    except:
        continue
data_clean = data.dropna(subset=usefullAttr, inplace=True)
X_Scale = np.array(data)
X_Scale
```

```
<ipython-input-23-ba92df615b8e>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[column] = (data[column]-data[column].mean()) / \
C:\Users\ALIEWARE\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas\util\_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return func(*args, **kwargs)
```

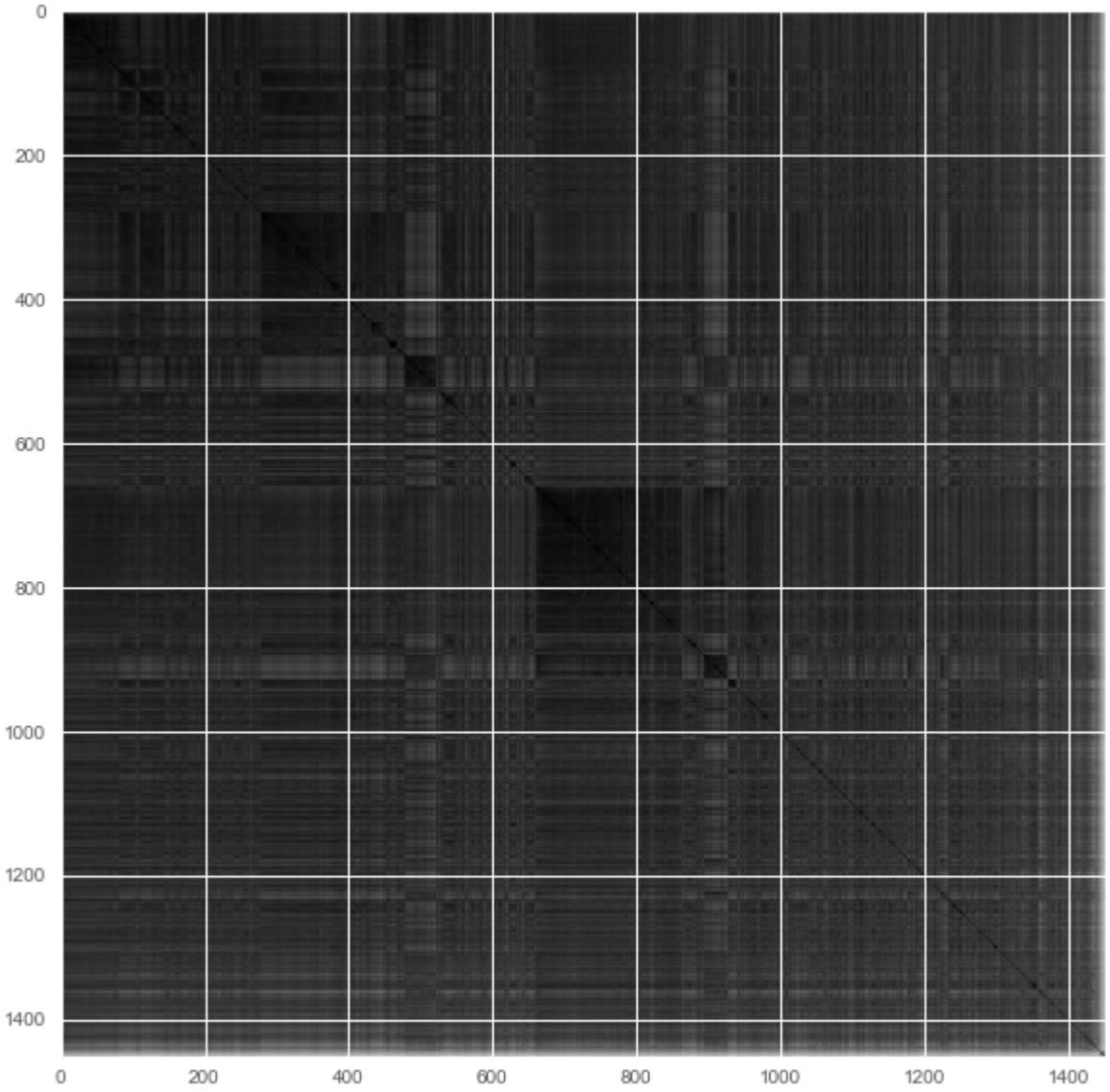
Out[ ]:
```
array([[ 0.34715427, -0.20707076, -0.51702265, ..., -0.35920182,
        -0.06866822,  0.6512561 ],
       [ 0.00728582, -0.0918549 ,  2.17888118, ..., -0.35920182,
        -0.06866822, -0.07181151],
       [ 0.53597007,  0.07345481, -0.51702265, ..., -0.35920182,
        -0.06866822,  0.6512561 ],
       ...,
       [ 1.07724204, -0.14775964,  3.07751579, ..., -0.35920182,
        -0.06866822,  0.6512561 ],
       [-0.48835566, -0.08013294,  0.38161196, ...,  1.47328444,
        -0.06866822, -0.79487911],
       [-0.42069666, -0.05809164,  0.38161196, ..., -0.35920182,
        -0.06866822, -0.79487911]])
```

In [ ]:
```
# HOPKINGS
X_scale = sklearn.preprocessing.scale(X_Scale)
# X = X_scale
pyclustertend.hopkins(X_scale, len(X_scale))
```

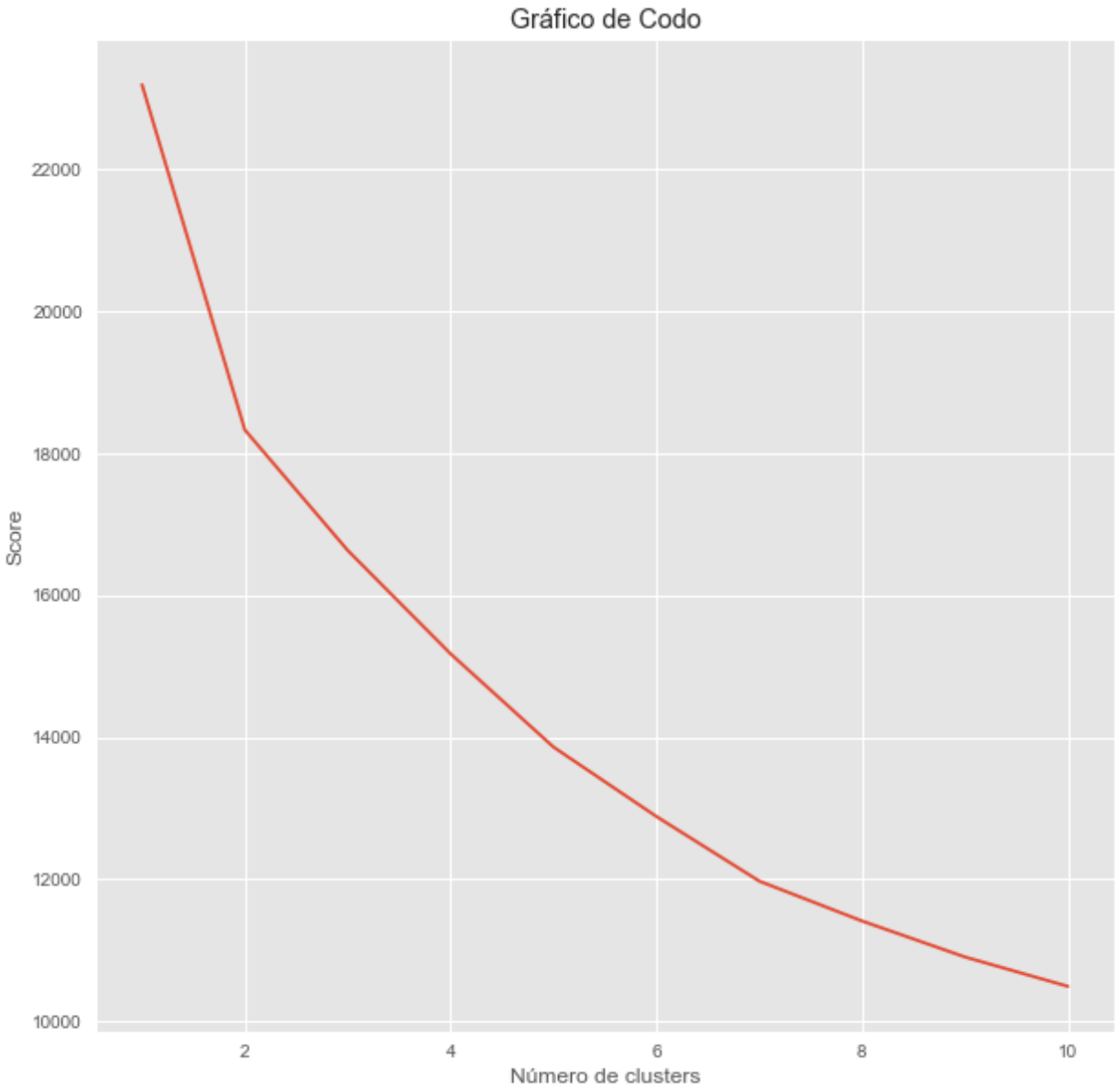
Out[ ]:
```
0.09216656314987669
```

In [ ]:
```
# VAT
pyclustertend.vat(X_Scale)

# devolvemos el SalePrice a su valor original
data['SalePrice'] = train['SalePrice']
```

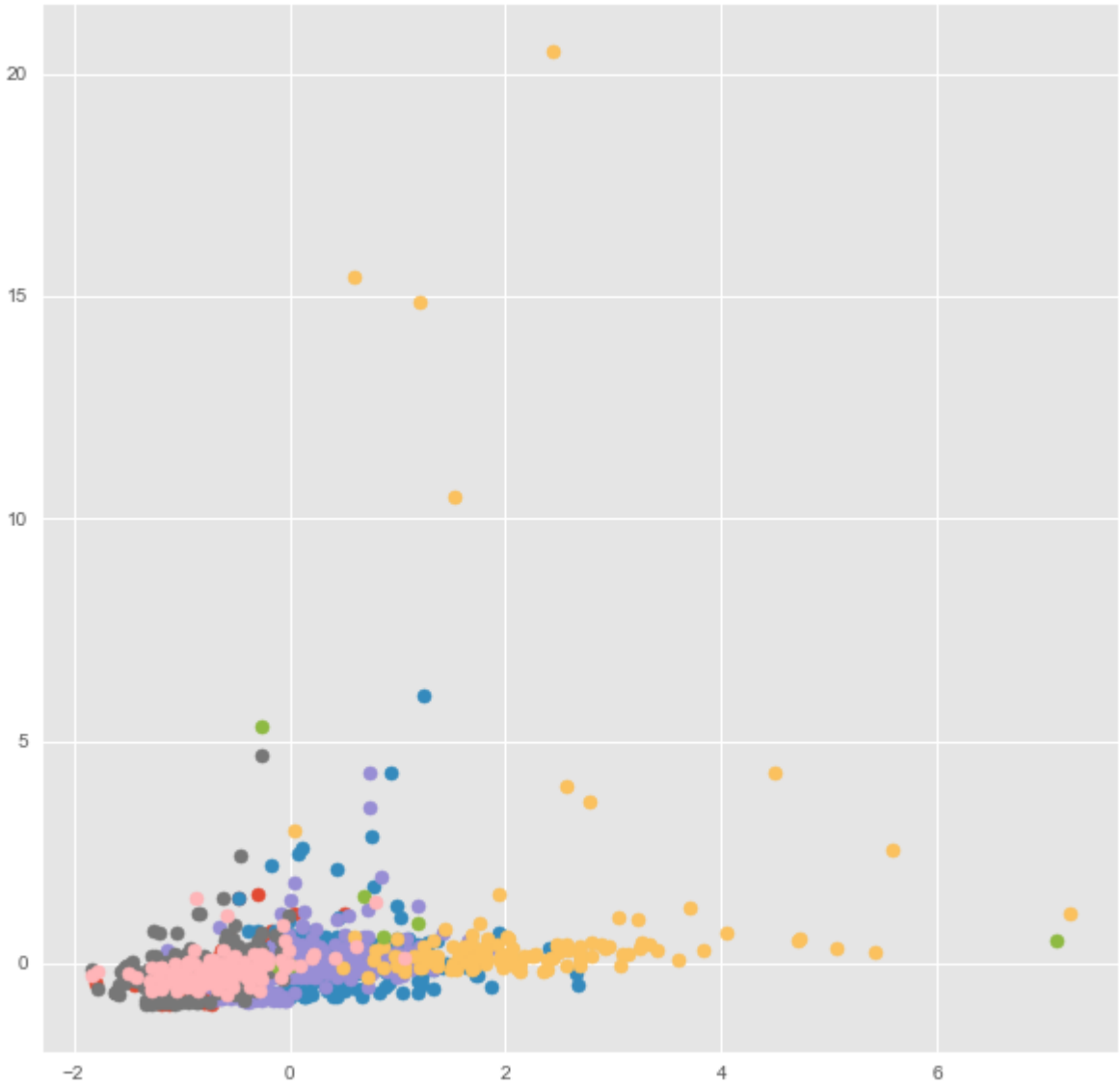
```
<ipython-input-25-9d4f744f424f>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['SalePrice'] = train['SalePrice']
```



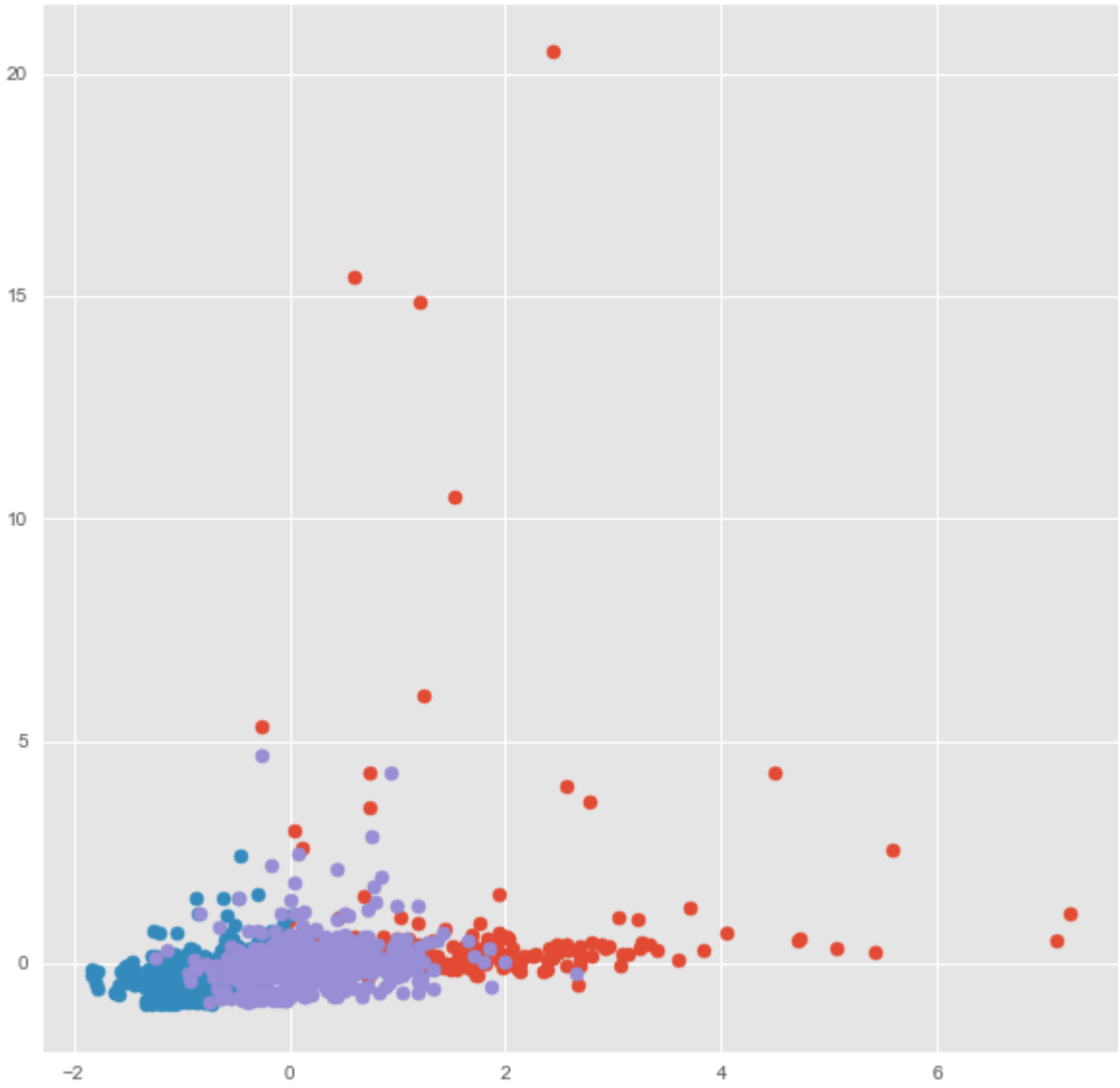
In [ ]:
```
numeroClusters = range(1, 11)
wcss = []
for i in numeroClusters:
    kmeans = cluster.KMeans(n_clusters=i)
    kmeans.fit(X_Scale)
    wcss.append(kmeans.inertia_)

plt.plot(numeroClusters, wcss)
plt.xlabel("Número de clusters")
plt.ylabel("Score")
plt.title("Gráfico de Codo")
plt.show()
```



In [ ]:
```
kmeans = cluster.KMeans(n_clusters=7)
kmeans.fit(X_Scale)
kmeans_result = kmeans.predict(X_Scale)
kmeans_clusters = np.unique(kmeans_result)
for kmeans_cluster in kmeans_clusters:
    # get data points that fall in this cluster
    index = np.where(kmeans_result == kmeans_cluster)
    # make the plot
    plt.scatter(X_Scale[index, 0], X_Scale[index, 1])
plt.show()
```



In [ ]:
```
kmeans = cluster.KMeans(n_clusters=3)
kmeans.fit(X_Scale)
kmeans_result = kmeans.predict(X_Scale)
kmeans_clusters = np.unique(kmeans_result)
for kmeans_cluster in kmeans_clusters:
    # get data points that fall in this cluster
    index = np.where(kmeans_result == kmeans_cluster)
    # make the plot
    plt.scatter(X_Scale[index, 0], X_Scale[index, 1])
plt.show()
```



In [ ]:
```
data['cluster'] = kmeans.labels_
print(data[data['cluster'] == 0].describe().transpose())
print(data[data['cluster'] == 1].describe().transpose())
print(data[data['cluster'] == 2].describe().transpose())
```

	count	mean	std	min
SalePrice	187.0	325915.711230	91124.858614	147000.000000
LotArea	187.0	0.749226	2.407136	-0.493908
OverallCond	187.0	-0.223885	0.856842	-3.212926
YearBuilt	187.0	0.702365	0.834545	-3.021822
MasVnrArea	187.0	1.341101	1.563777	-0.572637
TotalBsmtSF	187.0	1.333063	1.236907	-0.821575
1stFlrSF	187.0	1.364880	1.150799	-0.485341
2ndFlrSF	187.0	0.563273	1.404467	-0.794891
GrLivArea	187.0	1.480014	1.160952	0.004827
TotRmsAbvGrd	187.0	1.194840	1.023190	-0.933810
GarageCars	187.0	1.191774	0.708179	-2.364630
WoodDeckSF	187.0	0.775877	1.244999	-0.751918
OpenPorchSF	187.0	0.702148	1.302112	-0.704242
EnclosedPorch	187.0	-0.139590	1.025960	-0.359202
PoolArea	187.0	0.304544	2.288559	-0.068668
OverallQual	187.0	1.436190	0.750968	-0.794879
cluster	187.0	0.000000	0.000000	0.000000

	25%	50%	75%	max
SalePrice	271450.000000	312500.000000	368597.000000	755000.000000
LotArea	0.012941	0.186266	0.428320	20.511245
OverallCond	-0.517023	-0.517023	-0.517023	3.077516
YearBuilt	0.702985	1.050634	1.149962	1.282400
MasVnrArea	0.374530	1.084215	2.023098	8.263909
TotalBsmtSF	0.607630	1.291460	1.790657	11.517003
1stFlrSF	0.651529	1.325374	1.907389	9.129553
2ndFlrSF	-0.794891	0.520029	1.871602	3.935614
GrLivArea	0.589054	1.304590	2.084828	7.852884
TotRmsAbvGrd	0.296662	0.911897	2.142369	4.603312
GarageCars	0.311618	1.649742	1.649742	2.987865
WoodDeckSF	-0.001951	0.724081	1.274589	6.085550
OpenPorchSF	-0.032605	0.382452	1.137100	7.551611
EnclosedPorch	-0.359202	-0.359202	-0.359202	8.672338
PoolArea	-0.068668	-0.068668	-0.068668	18.299910
OverallQual	1.374324	1.374324	2.097391	2.820459
cluster	0.000000	0.000000	0.000000	0.000000

	count	mean	std	min
SalePrice	650.0	125140.146154	28148.752670	34900.000000
LotArea	650.0	-0.193614	0.348891	-0.923413
OverallCond	650.0	0.304191	1.166288	-4.111561
YearBuilt	650.0	-0.694282	0.807482	-3.286697
MasVnrArea	650.0	-0.375268	0.470206	-0.572637
TotalBsmtSF	650.0	-0.515661	0.636160	-2.410341
1stFlrSF	650.0	-0.527210	0.554765	-2.143438
2ndFlrSF	650.0	-0.340121	0.695040	-0.794891
GrLivArea	650.0	-0.665423	0.608632	-2.248350
TotRmsAbvGrd	650.0	-0.532487	0.761943	-2.779517
GarageCars	650.0	-0.703298	0.899220	-2.364630
WoodDeckSF	650.0	-0.281194	0.809980	-0.751918
OpenPorchSF	650.0	-0.381276	0.754493	-0.704242
EnclosedPorch	650.0	0.256015	1.197400	-0.359202
PoolArea	650.0	-0.046612	0.562322	-0.068668
OverallQual	650.0	-0.735921	0.658751	-3.687150
cluster	650.0	1.000000	0.000000	1.000000

	25%	50%	75%	max
SalePrice	109002.000000	128000.000000	142500.000000	230000.000000
LotArea	-0.365968	-0.204065	-0.050603	2.417847
OverallCond	-0.517023	0.381612	1.280247	3.077516

YearBuilt	-1.366352	-0.538617	-0.108195	1.183071
MasVnrArea	-0.572637	-0.572637	-0.572637	3.017210
TotalBsmtSF	-0.819296	-0.440910	-0.112671	1.619699
1stFlrSF	-0.855244	-0.555182	-0.215027	1.677170
2ndFlrSF	-0.794891	-0.794891	0.327602	2.895590
GrLivArea	-1.117956	-0.774460	-0.321066	2.065798
TotRmsAbvGrd	-0.933810	-0.318574	-0.318574	2.757604
GarageCars	-1.026506	-1.026506	0.311618	2.987865
WoodDeckSF	-0.751918	-0.751918	0.089800	2.966005
OpenPorchSF	-0.704242	-0.704242	-0.391063	7.189379
EnclosedPorch	-0.359202	-0.359202	0.107100	5.040088
PoolArea	-0.068668	-0.068668	-0.068668	14.267783
OverallQual	-0.794879	-0.794879	-0.071812	1.374324
cluster	1.000000	1.000000	1.000000	1.000000

	count	mean	std	min	\
SalePrice	615.0	195066.242276	40068.264642	82500.000000	
LotArea	615.0	-0.025440	0.521432	-0.861296	
OverallCond	615.0	-0.245240	0.728708	-2.314292	
YearBuilt	615.0	0.508393	0.736359	-2.624509	
MasVnrArea	615.0	-0.011157	0.831704	-0.572637	
TotalBsmtSF	615.0	0.131154	0.791030	-2.410341	
1stFlrSF	615.0	0.133920	0.859073	-1.726973	
2ndFlrSF	615.0	0.187945	0.999425	-0.794891	
GrLivArea	615.0	0.247107	0.606348	-1.060865	
TotRmsAbvGrd	615.0	0.198624	0.812320	-1.549046	
GarageCars	615.0	0.374716	0.516936	-2.364630	
WoodDeckSF	615.0	0.064523	0.964901	-0.751918	
OpenPorchSF	615.0	0.179862	0.946932	-0.704242	
EnclosedPorch	615.0	-0.228789	0.631522	-0.359202	
PoolArea	615.0	-0.042443	0.650364	-0.068668	
OverallQual	615.0	0.330285	0.645819	-1.517947	
cluster	615.0	2.000000	0.000000	2.000000	

	25%	50%	75%	max
SalePrice	168500.000000	188000.000000	220500.000000	392000.000000
LotArea	-0.246545	-0.072819	0.135120	4.677080
OverallCond	-0.517023	-0.517023	-0.517023	3.077516
YearBuilt	0.090461	0.818868	1.083743	1.249290
MasVnrArea	-0.572637	-0.572637	0.355200	5.662651
TotalBsmtSF	-0.494476	0.124390	0.742117	2.458531
1stFlrSF	-0.575876	0.143236	0.815787	2.574767
2ndFlrSF	-0.794891	0.419234	1.111056	2.474083
GrLivArea	-0.168348	0.181808	0.625211	2.988763
TotRmsAbvGrd	-0.318574	0.296662	0.911897	3.372840
GarageCars	0.311618	0.311618	0.311618	2.987865
WoodDeckSF	-0.751918	0.029963	0.588449	5.120166
OpenPorchSF	-0.704242	-0.070337	0.563567	5.604618
EnclosedPorch	-0.359202	-0.359202	-0.359202	4.843750
PoolArea	-0.068668	-0.068668	-0.068668	16.059839
OverallQual	-0.071812	0.651256	0.651256	2.097391
cluster	2.000000	2.000000	2.000000	2.000000

```
<ipython-input-29-aaea25b7312c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  data['cluster'] = kmeans.labels_
```

Variable clasificacion

In [ ]:
```
# Clasificacion de casas en: Economias, Intermedias o Caras.
data.fillna(0)
limit1 = data.query('cluster == 0')['SalePrice'].mean()
limit2 = data.query('cluster == 1')['SalePrice'].mean()
data['Clasificacion'] = data['LotArea']
data.loc[data['SalePrice'] < limit1, 'Clasificacion'] = 'Economica'
data.loc[(data['SalePrice'] >= limit1) & (
    data['SalePrice'] < limit2), 'Clasificacion'] = 'Intermedia'
data.loc[data['SalePrice'] >= limit2, 'Clasificacion'] = 'Caras'
```

```
<ipython-input-30-cb51636d69dd>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Clasificacion'] = data['LotArea']
C:\Users\ALIEWARE\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas\core\indexing.py:1817: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value, pi)
```

Contamos la cantidad de casas por clasificacion

In [ ]:
```
# Obtener cuantos datos hay por cada clasificacion
print(data['Clasificacion'].value_counts())
```

```
Caras        1141
Economica     311
Name: Clasificacion, dtype: int64
```

Dividmos en entrenamiento y prueba

Estableciendo los conjuntos de Entrenamiento y Prueba

In [ ]:
```
y = data['SalePrice']
X = data.drop(['Clasificacion', 'SalePrice', 'cluster', 'OverallQual'], axis=1)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, train_size=0.7)
y_train
```

Out[ ]:
```
69      225000
394     109000
1027    293077
1263    180500
1306    202500
         ...
1206    107000
29       68500
792     269790
576     145000
732     222500
Name: SalePrice, Length: 1016, dtype: int64
```

70% de entrenamiento y 30% prueba

Inicia de HT4

2. Elabore un modelo de regresión lineal utilizando el conjunto de entrenamiento que hizo para predecir los precios de las casas. Explique los resultados a los que llega. Muestre el modelo gráficamente. El experimento debe ser reproducible por lo que debe fijar que los conjuntos de entrenamiento y prueba sean los mismos siempre que se ejecute el código.

In [ ]:
```
p_length = y_train.values.reshape(-1, 1)
p_length_t = y_test.values.reshape(-1, 1)
p_width = X_train['GrLivArea'].values.reshape(-1, 1)
p_width_t = X_test['GrLivArea'].values.reshape(-1, 1)
lm = LinearRegression()
lm.fit(p_width, p_length)
p_length_pred = lm.predict(p_width_t)
```

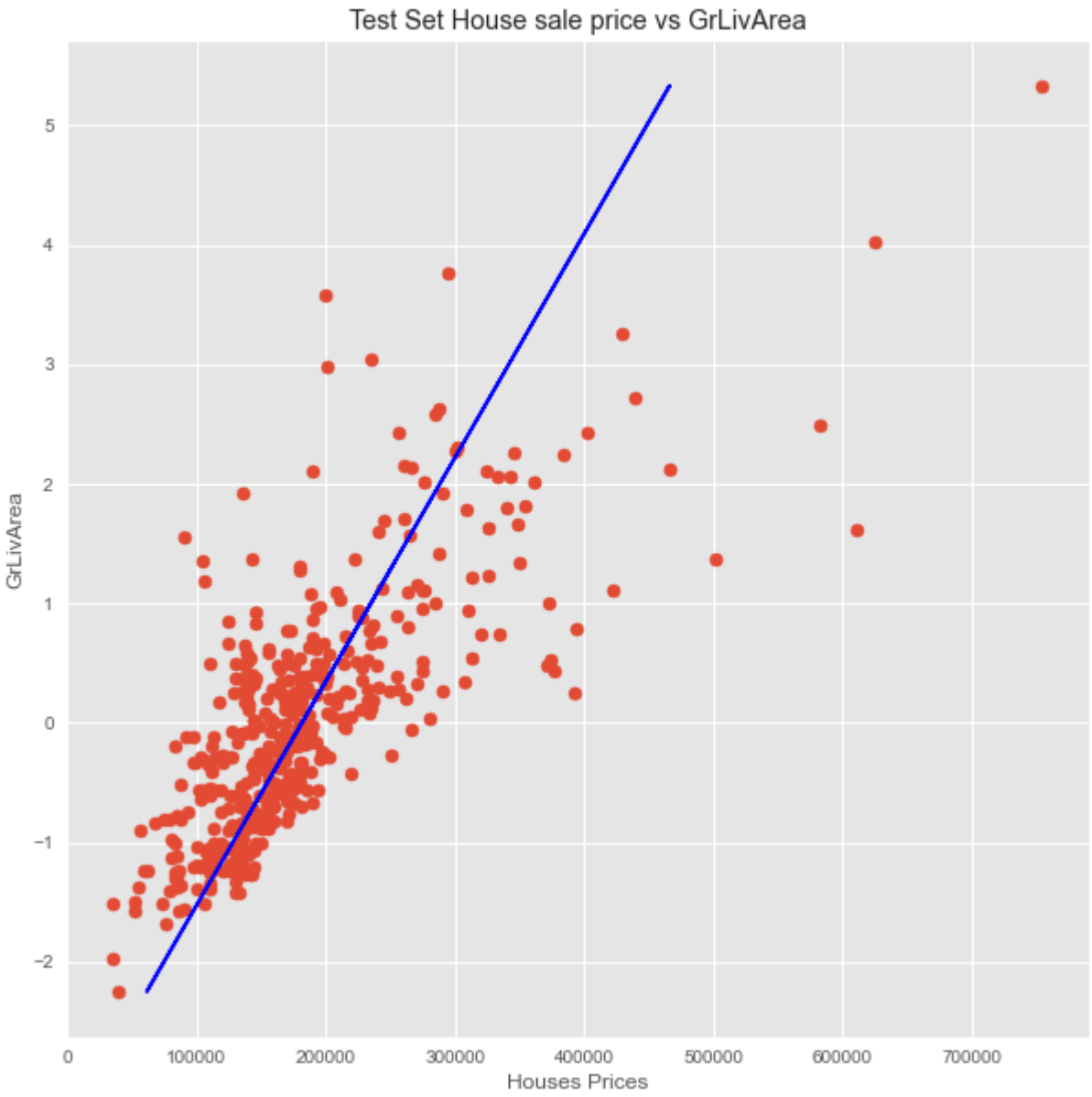
Haciendo la ecuación

In [ ]:
```
#y = mx + c
m = lm.coef_[0][0]
c = lm.intercept_[0]


label = r'$p_length = %0.4f*p_width %+0.4f$' % (m, c)
print(label)
```

$p_length = 53452.5544*p_width +181300.1822$

In [ ]:
```
fig = plt.figure()
plt.scatter(p_length_t, p_width_t)
plt.plot(p_length_pred, p_width_t, color="blue")
plt.xlabel("Houses Prices")
plt.ylabel("GrLivArea")
plt.title("Test Set House sale price vs GrLivArea")
```

Out[ ]: Text(0.5, 1.0, 'Test Set House sale price vs GrLivArea')



In [ ]:
```
print("Mean Squared Error: %.2f" %
      mean_squared_error(p_length_t, p_length_pred))
print("R squared: %.2f" % r2_score(p_length_t, p_length_pred))
```

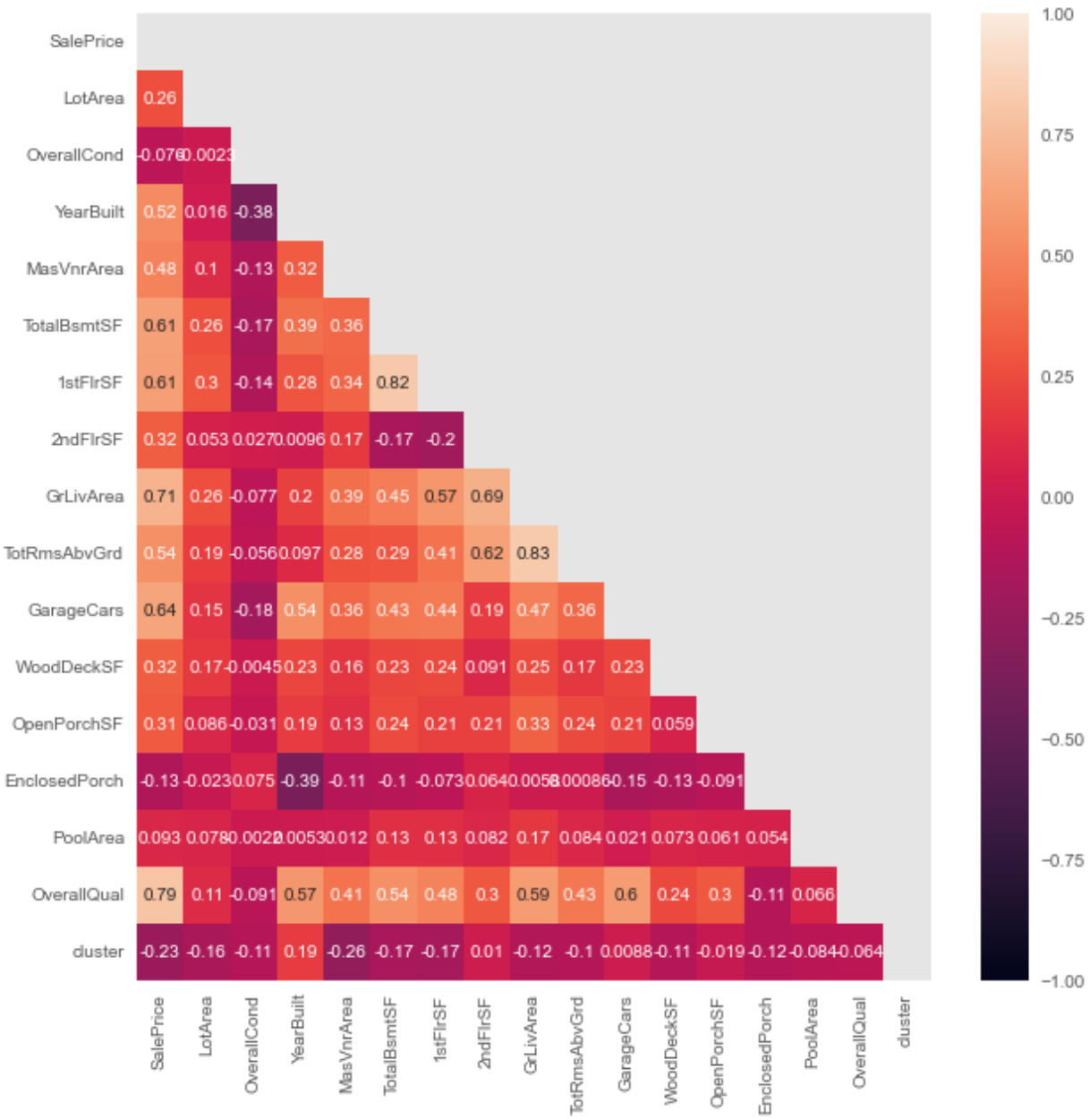
```
Mean Squared Error: 3315529683.89
R squared: 0.55
```

3. Analice el modelo. Determine si hay multicolinealidad en las variables, y cuáles son las que aportan al modelo, por su valor de significación. Haga un análisis de correlación de las variables del modelo y especifique si el modelo se adapta bien a los datos. Explique si hay sobreajuste (overfitting) o no. En caso de existir sobreajuste, haga otro modelo que lo corrija.

In [ ]:
```
print('La multicolinealidad ocurre cuando hay dos o más variables independientes en ur
```

```
La multicolinealidad ocurre cuando hay dos o más variables independientes en un model
o de regresión múltiple, en el heatmap podemos observar la relacion entre varias vari
ables, esto por medio de los indices de correlacion, podemos ver que los cuadros con
colores mas claros estan mas correlacionados que aquellas variables con un indice men
or, entonces a traves de la grafica podemos sacar conclusioones de que variables infl
uyen sobre cuales en el contexto de las casas. Otro de los indicadores es el VIF, lo
cual nos muestra tambien alto grado de correlacion
```

In [ ]:
```
hm = sns.heatmap(data.corr(), annot=True, mask=np.triu(
    np.ones_like(data.corr(), dtype=bool)), vmin=-1, vmax=1)
plt.show()
```



In [ ]:
```
# Extraido de: https://towardsdatascience.com/statistics-in-python-collinearity-and-mu


def calculate_vif(df, features):
    vif, tolerance = {}, {}
    # all the features that you want to examine
```

```
    for feature in features:
        # extract all the other features you will regress against
        X = [f for f in features if f != feature]
        X, y = df[X], df[feature]
        # extract r-squared from the fit
        r2 = LinearRegression().fit(X, y).score(X, y)

        # calculate tolerance
        tolerance[feature] = 1 - r2
        # calculate VIF
        vif[feature] = 1/(tolerance[feature])
    # return VIF DataFrame
    return pd.DataFrame({'VIF': vif, 'Tolerance': tolerance})
```

In [ ]:
```
calculate_vif(df=data, features=['SalePrice',
              'GrLivArea', 'LotArea', 'OverallQual'])
```

Out[ ]:

	VIF	Tolerance
SalePrice	3.648074	0.274117
GrLivArea	2.060637	0.485287
LotArea	1.126017	0.888086
OverallQual	2.769891	0.361025

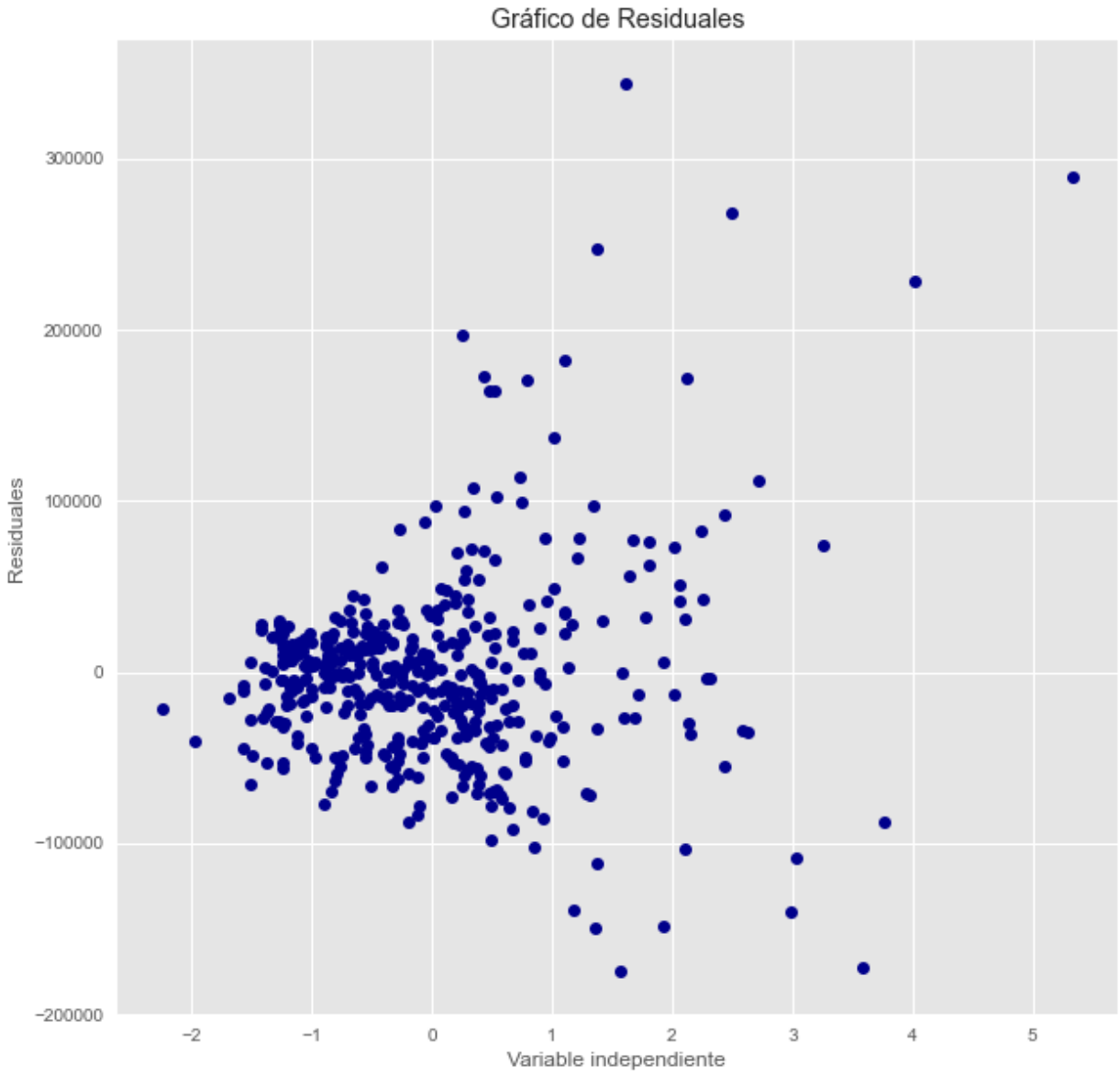
4. Determine la calidad del modelo realizando un análisis de los residuos.

In [ ]:
```
residuales = p_length_t - p_length_pred
len(residuales)
```

Out[ ]:
```
436
```

In [ ]:
```
plt.plot(p_width_t, residuales, 'o', color='darkblue')
plt.title("Gráfico de Residuales")
plt.xlabel("Variable independiente")
plt.ylabel("Residuales")
```

Out[ ]:
```
Text(0, 0.5, 'Residuales')
```

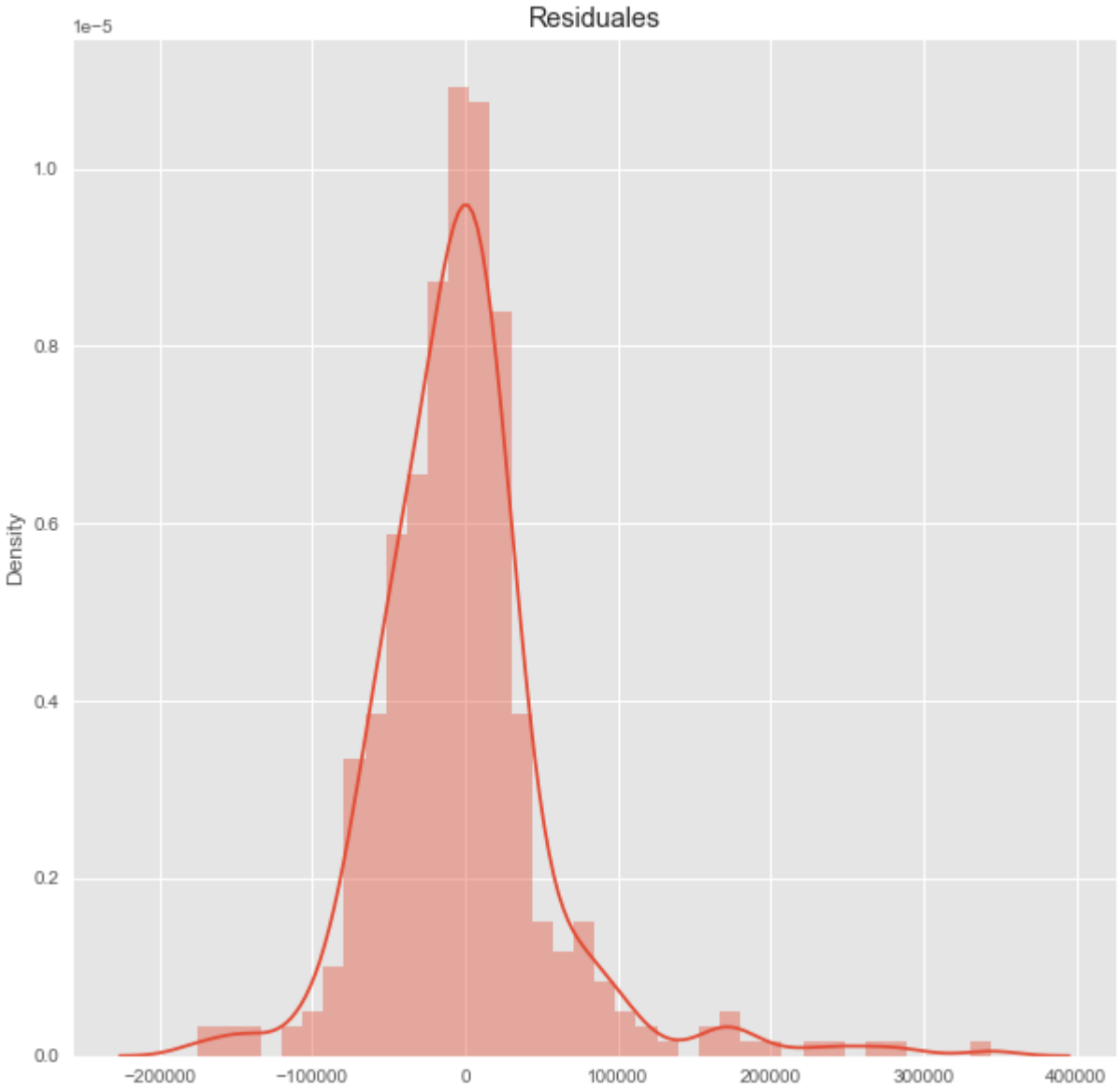


Según el gráfico de los residuos se puede observar que parecen estar aleatoriamente distribuidos alrededor de 0

In [ ]:
```
sns.distplot(residuales)
plt.title("Residuales")
```

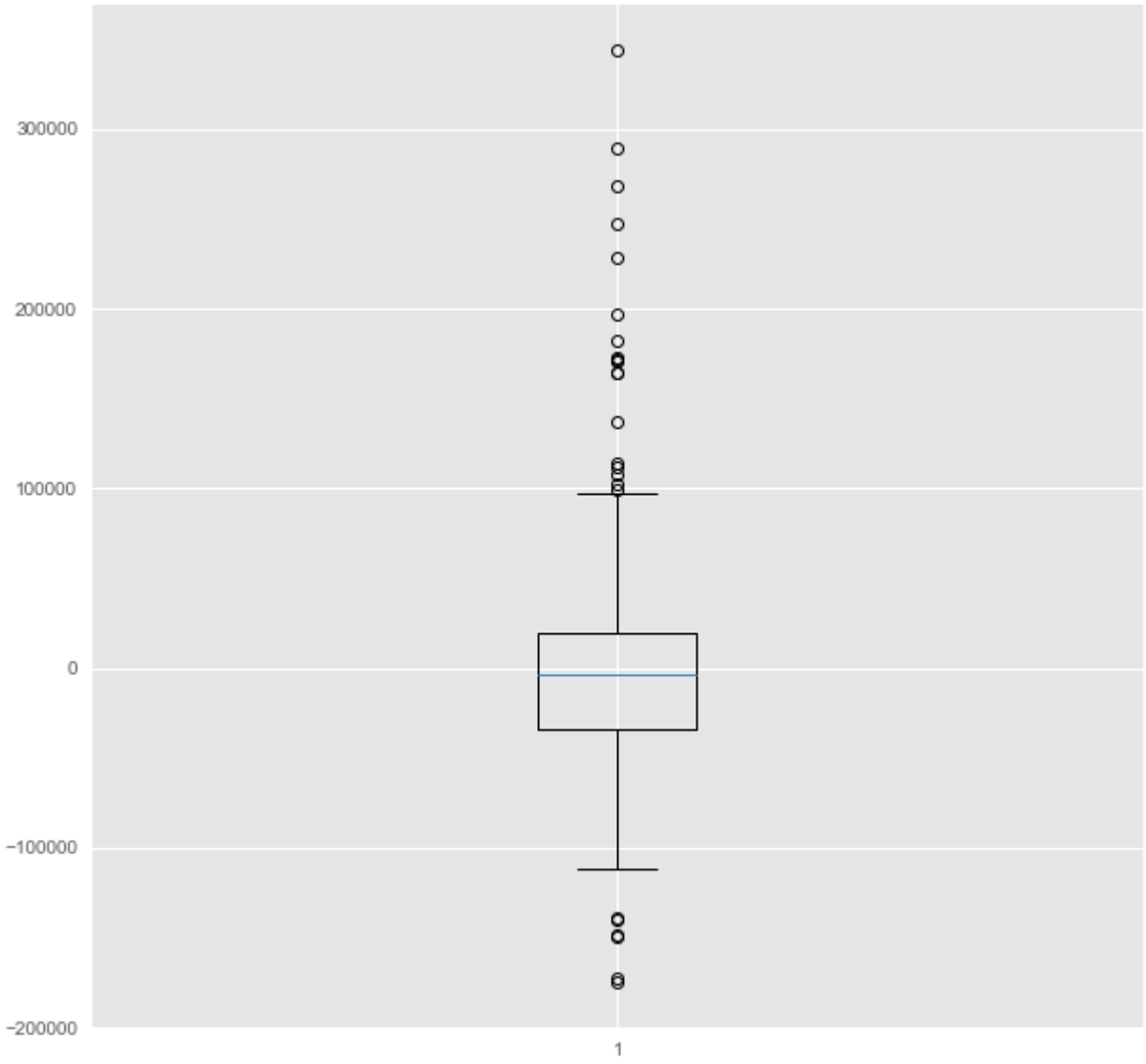
```
C:\Users\ALIEWARE\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:
```
Text(0.5, 1.0, 'Residuales')
```



In [ ]:
```
plt.boxplot(residuales)
```

Out[ ]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x255d65ba280>,
  <matplotlib.lines.Line2D at 0x255d65ba0d0>],
 'caps': [<matplotlib.lines.Line2D at 0x255d65bac70>,
  <matplotlib.lines.Line2D at 0x255d65baca0>],
 'boxes': [<matplotlib.lines.Line2D at 0x255d66aae80>],
 'medians': [<matplotlib.lines.Line2D at 0x255d65ba250>],
 'fliers': [<matplotlib.lines.Line2D at 0x255d664cfa0>],
 'means': []}
```



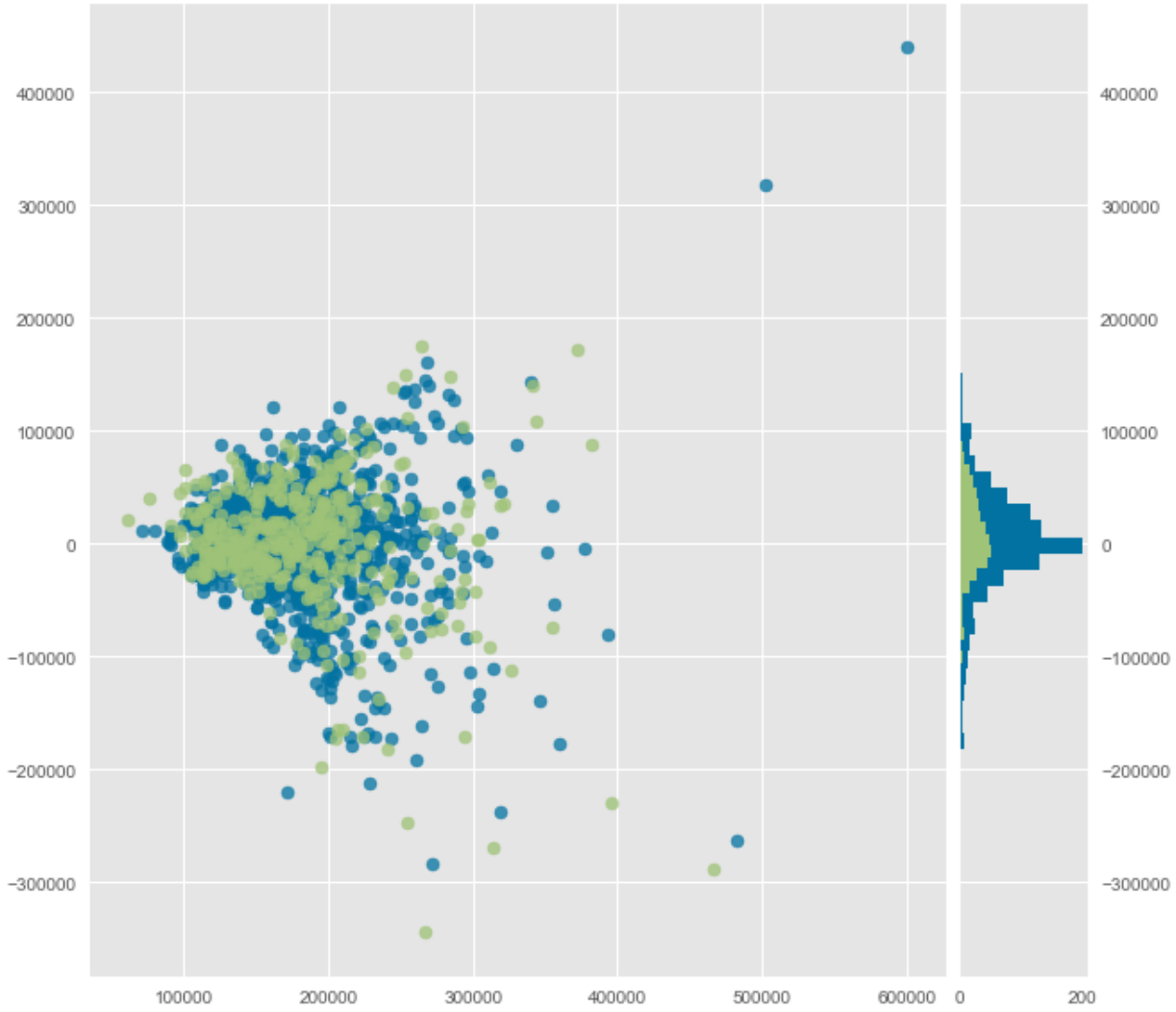
In [ ]:
```
normaltest(residuales)
```

Out[ ]:
```
NormaltestResult(statistic=array([171.75860113]), pvalue=array([5.04770137e-38]))
```

Podemos ver que los residuos siguen una distribución normal puesto que no se puede rechazar la hipotesis nula de normalidad porque el valor de p es mayor a 0.05

In [ ]:
```
model = Ridge()
visualizer = ResidualsPlot(model)
visualizer.fit(p_width, p_length)
visualizer.score(p_width_t, p_length_t)
```

Out[ ]:
```
0.5521137266458225
```



5. Utilice el modelo con el conjunto de prueba y determine la eficiencia del algoritmo para predecir el precio de las casas.

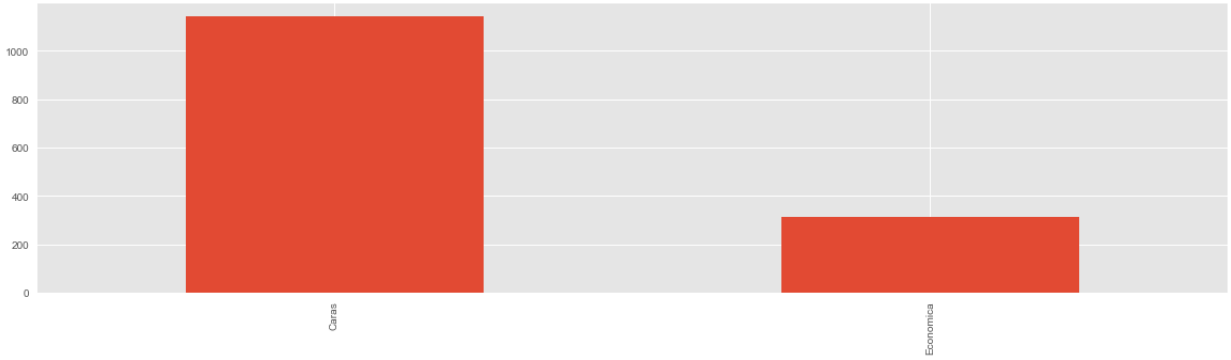
In [ ]:
```
y = data['Clasificacion']
X = data.drop(['Clasificacion', 'SalePrice'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, train_size=0.7)
y_train
arbol = DecisionTreeClassifier(max_depth=4, random_state=42)
arbol = arbol.fit(X_train, y_train)
y_pred = arbol.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(
    y_test, y_pred, average='weighted'))
print('Se determino que el conjunto de prueba tiene una alta eficiencia, si observamos
```

```
Accuracy: 0.8990825688073395
Precision: 0.8973767186586632
Se determino que el conjunto de prueba tiene una alta eficiencia, si observamos la pr
ecision y la exactitud, podemos ver valores cercanos a uno, lo cual indica que el alg
oritmo para determinar el precio de las casas si es eficiente
```

6. Discuta sobre la efectividad del modelo. Haga los gráficos que crea que le pueden ayudar en la discusión.

In [ ]:
```
print("Se puede observar que el arbol de decisión tuvo un accuracy de 0.84 y una preci
plt.figure(figsize=(20, 5))
data['Clasificacion'].value_counts().plot(kind='bar')
plt.show()
```

```
Se puede observar que el arbol de decisión tuvo un accuracy de 0.84 y una precisión d
e 0.84. dado esto se puede concluir que el modelo es efectivo en un 84%. También se p
uede observar que los puntos residuales se encuentran en un radio de 2 y por su conce
ntración en 0 se puede concluir que es efectivo.
```



7. Compare la eficiencia del algoritmo con el resultado obtenido con el árbol de decisión (el de regresión). ¿Cuál es mejor para predecir? ¿Cuál se demoró más en procesar?

In [ ]:
```
print("El coeficiente de R nos dice como el algoritmo de arbol de decisión es mucho má
```

```
El coeficiente de R nos dice como el algoritmo de arbol de decisión es mucho más efic
az y mejor para predecir. Con los valores de AIC y BIC concluimos que el arbol de dec
isión es el más eficiente.
```