In [1]:

```
!pip install plotly
!pip install numpy
!pip install matplotlib
!pip install statsmodels
!pip install pandas
!pip install sklearn
!pip install seaborn
!pip install yellowbrick
!pip install scipy
```

```
Python.framework/Versions/3.7/lib/python3.7/site-packages (from seabor
n) (3.4.1)
Requirement already satisfied: pandas>=0.23 in /Library/Frameworks/Pyt
hon.framework/Versions/3.7/lib/python3.7/site-packages (from seaborn)
(1.1.5)
Requirement already satisfied: pyparsing>=2.2.1 in /Library/Framework
s/Python.framework/Versions/3.7/lib/python3.7/site-packages (from matp
lotlib>=2.2->seaborn) (2.2.2)
Requirement already satisfied: pillow>=6.2.0 in /Library/Frameworks/Py
thon.framework/Versions/3.7/lib/python3.7/site-packages (from matplotl
ib>=2.2->seaborn) (8.2.0)

Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Pyt
hon.framework/Versions/3.7/lib/python3.7/site-packages (from matplotli
b>=2.2->seaborn) (0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in /Library/Framew
orks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from m
atplotlib>=2.2->seaborn) (2.7.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /Library/Framework
s/Python.framework/Versions/3.7/lib/python3.7/site-packages (from matp
lotlib>=2.2->seaborn) (1.0.1)
```

In [2]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import csv
import statsmodels.api as sm
import statsmodels.stats.diagnostic as smd
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.linear_model import Ridge
import seaborn as sns
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
from yellowbrick.regressor import ResidualsPlot
import scipy.stats as stats
import statsmodels.stats.diagnostic as diag
from scipy.stats import normaltest
```

In [3]:

```
trainSetGiven = pd.read_csv("data/train.csv")
trainSetGiven.columns
```

Out[3]:

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Stre
et',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgT
ype',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearR
emodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrT
ype',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQua
l',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heat
ing',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrS
F',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'F
ullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'Gar
ageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Gar
ageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQ
C',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleTyp
e',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [4]:

```
#We clean the data
trainSetGiven = trainSetGiven.drop(columns=['Id'])
```

In [5]:

```
#Get only numeric
dataSetCompleteNumeric = trainSetGiven._get_numeric_data().dropna(how='any')
lenP = len(dataSetCompleteNumeric['SalePrice'])
# dataSetCompleteNumeric['es_caro'] = np.random.randn(lenP)
# dataSetCompleteNumeric['es_barato'] = np.random.randn(lenP)
# dataSetCompleteNumeric['es_medio'] = np.random.randn(lenP)

dataSetCompleteNumeric
```

Out[5]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000
...	...	...	...	...	...	...	...
1455	60	62.0	7917	6	5	1999	2000
1456	20	85.0	13175	6	6	1978	1988
1457	70	66.0	9042	7	9	1941	2006
1458	20	68.0	9717	5	6	1950	1996
1459	20	75.0	9937	5	6	1965	1965

1121 rows × 37 columns

In [6]:

```
# We sperate to get valid sets
caros = dataSetCompleteNumeric[dataSetCompleteNumeric['SalePrice']>150000].copy()
# caros['cat'] = 'Caro'
caros['es_caro'] = 1
caros['es_barato'] = 0
caros['es_medio'] = 0
baratos = dataSetCompleteNumeric[dataSetCompleteNumeric['SalePrice']<80000].copy()
baratos['es_caro'] = 0
baratos['es_barato'] = 1
baratos['es_medio'] = 0
medio = dataSetCompleteNumeric[dataSetCompleteNumeric['SalePrice'] > 80000].copy()
medio = medio[medio['SalePrice'] < 150000]
medio['es_caro'] = 0
medio['es_barato'] = 0
medio['es_medio'] = 1
caros
```

Out[6]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
4	60	84.0	14260	8	5	2000	2000
6	20	75.0	10084	8	5	2004	2005
...	...	...	...	...	...	...	...
1451	20	78.0	9262	8	5	2008	2009
1454	20	62.0	7500	7	5	2004	2005
1455	60	62.0	7917	6	5	1999	2000
1456	20	85.0	13175	6	6	1978	1988
1457	70	66.0	9042	7	9	1941	2006

659 rows × 40 columns

In [7]:

```
#Proporciones caros
trainCaro = caros.sample(frac=0.65)
testCaro = caros.drop(trainCaro.index)

#Proporciones baratos
trainBarato = baratos.sample(frac=0.65)
testBarato = baratos.drop(trainBarato.index)

#Proporciones medio
trainMedio = medio.sample(frac=0.65)
testMedio = medio.drop(trainMedio.index)

# Proporcion 35, 65
train = trainCaro.append(trainMedio).append(trainBarato)
test  = testCaro.append(testMedio).append(testBarato)

#all data
datos = train.append(test)
salePrice = datos.pop("SalePrice")
```

In [8]:

```
#Plot config
plt.rcParams['figure.figsize'] = (15, 9)
plt.style.use('ggplot')
```

Separaremos valores y calcularemos con Los diferentes Y

Calcularemos los Caros

In [9]:

```
import random

#Medir si es caro o
c = datos.copy()
y_caro = c.pop("es_caro")
y_barato = c.pop("es_barato")
y_medio = c.pop("es_medio")
y = y_caro #La dicotómicas respuesta cara
X = c #El resto de los datos
random.seed(123)
y
```

Out[9]:

```
172     1
704     1
910     1
647     1
959     1
       ..
874     0
915     0
935     0
1279    0
1380    0
Name: es_caro, Length: 1116, dtype: int64
```

In [10]:

```
X_train, X_test,y_train, y_test = train_test_split(X, y,test_size=0.3,train_size=0.7
```

Haciendo regresion con Caros

In [11]:

```
from sklearn.metrics import confusion_matrix
logReg = LogisticRegression(solver='liblinear')
logReg.fit(X_train,y_train)
y_pred = logReg.predict(X_test)
y_proba = logReg.predict_proba(X)[:,1]
cm = confusion_matrix(y_test,y_pred)
```

Medir presicion Caros

In [12]:

```
from sklearn.metrics import accuracy_score ,precision_score,recall_score,f1_score
accuracy=accuracy_score(y_test,y_pred)
precision =precision_score(y_test, y_pred,average='micro')
recall =  recall_score(y_test, y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print('Matriz de confusión para detectar casas con valor cara\n',cm)
print('Accuracy: ',accuracy)
```

```
Matriz de confusión para detectar casas con valor cara
 [[121  11]
 [ 17 186]]
Accuracy:  0.9164179104477612
```

Resultados caros

Dentro de la matriz de confusión que nos mide los verdaderos correctos y los falsos correctos y viceversa podemos ver que tuvo una precision del 91.64%

Calcularemos los baratos

In [13]:

```
y = y_barato #La variable dicotómicas barata
X = c #El resto de los datos
```

In [14]:

```
X_train, X_test,y_train, y_test = train_test_split(X, y,test_size=0.3,train_size=0.7
```

Haciendo regresion con Baratos

In [15]:

```
logReg = LogisticRegression(solver='liblinear')
logReg.fit(X_train,y_train)
y_pred = logReg.predict(X_test)
y_proba = logReg.predict_proba(X)[:,1]
cm = confusion_matrix(y_test,y_pred)
```

In [16]:

```
accuracy=accuracy_score(y_test,y_pred)
precision =precision_score(y_test, y_pred,average='micro')
recall =  recall_score(y_test, y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print('Matriz de confusión para detectar casas con valor barato\n',cm)
print('Accuracy: ',accuracy)
```

```
Matriz de confusión para detectar casas con valor barato
 [[327   1]
 [  5   2]]
Accuracy:  0.982089552238806
```

Resultados baratos

En este modelo tuvimos una precision del 98.2% seindo excelente casi perfecto

Calcularemos los medios

In [17]:

```
y = y_medio #La variable dicotómicas medio
X = c #El resto de los datos
```

In [18]:

```
X_train, X_test,y_train, y_test = train_test_split(X, y,test_size=0.3,train_size=0.7
```

Haciendo regresion con medios

In [19]:

```
logReg = LogisticRegression(solver='liblinear')
logReg.fit(X_train,y_train)
y_pred = logReg.predict(X_test)
y_proba = logReg.predict_proba(X)[:,1]
cm = confusion_matrix(y_test,y_pred)
```

In [20]:

```
accuracy=accuracy_score(y_test,y_pred)
precision =precision_score(y_test, y_pred,average='micro')
recall =  recall_score(y_test, y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print('Matriz de confusión para detectar casas con valor medio\n',cm)
print('Accuracy: ',accuracy)
```

```
Matriz de confusión para detectar casas con valor medio
 [[186  24]
 [ 14 111]]
Accuracy:  0.8865671641791045
```

Resultados medios

En este modelo tuvimos una precision del 88.65% seindo excelente mas sin embargo no fue mejor que las casas baratas

Resumen

Se observaron buenos resultados en las 3 tanto las medias como las caras se corrieron vairas veces no se diria que una de esos modelos es mejor pero las baratas siempre sacaron ventaja.

Cual se tardo mas?

Ni una realmente todas parecieron correr igual de rapidos tal vez demoro un poco mas las medias

¿Cuál se equivocó más?

Se corrieron varias veces y tanto las caras como las medias rondaron entre 87% y 92% asi que no nos animariamos a decir cual de esas fallo menos. Pero tiene sentido ya que este grupo es mayor a las baratas.

¿Cuál se equivocó menos? y ¿por qué?

Se equivocaron menos las baratas porque su rango era mas delimitado reduciendo asi el margen de error y teniendo sets mas precisos. En otras palabras la variable categorica que se tomo fue mejor otorgada.

In [ ]:

```
```