Started 'Python 3.9.6 64-bit' kernel

Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

HOJA DE TRABAJO 5 REDES BAYESIANAS

Raul Jimenez 19017 Donaldo Garcia 19683 Oscar Saravia 19322 link al repo: https://github.com/raulangelj/HT5_REDES_BAYESIANAS

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, cross_val_score, StratifiedKFold
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 sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from yellowbrick.regressor import ResidualsPlot
from sklearn.metrics import make_scorer, accuracy_score, precision_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# 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 utilizó en las dos hojas anteriores.

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

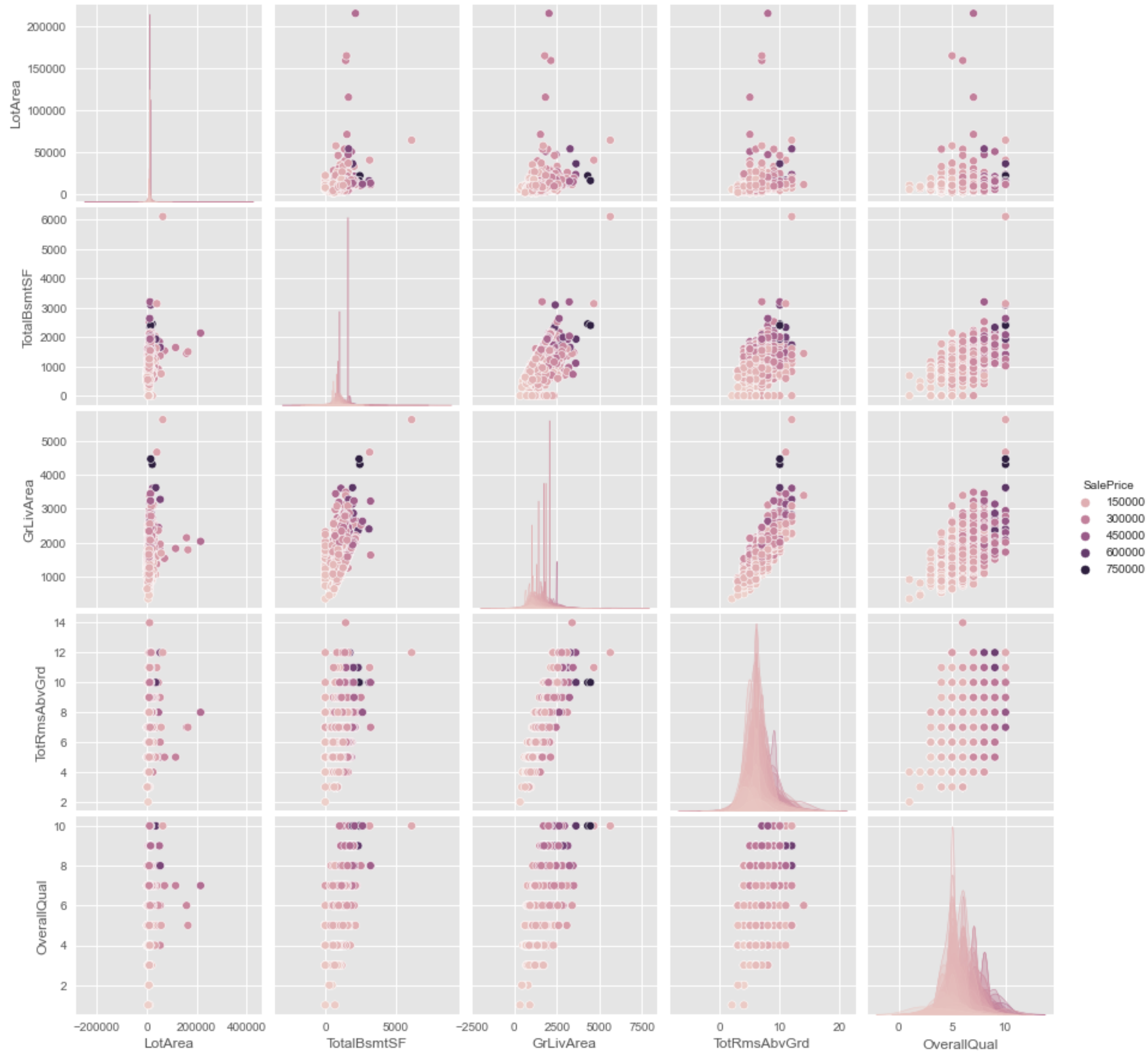
In [ ]:
```
usefullAttr = ['SalePrice', 'LotArea', 'OverallCond', 'YearBuilt', 'MasVnrArea', 'Tota
              '2ndFlrSF', 'GrLivArea', 'TotRmsAbvGrd', 'GarageCars', 'WoodDeckSF', '(
```

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

Out[ ]:

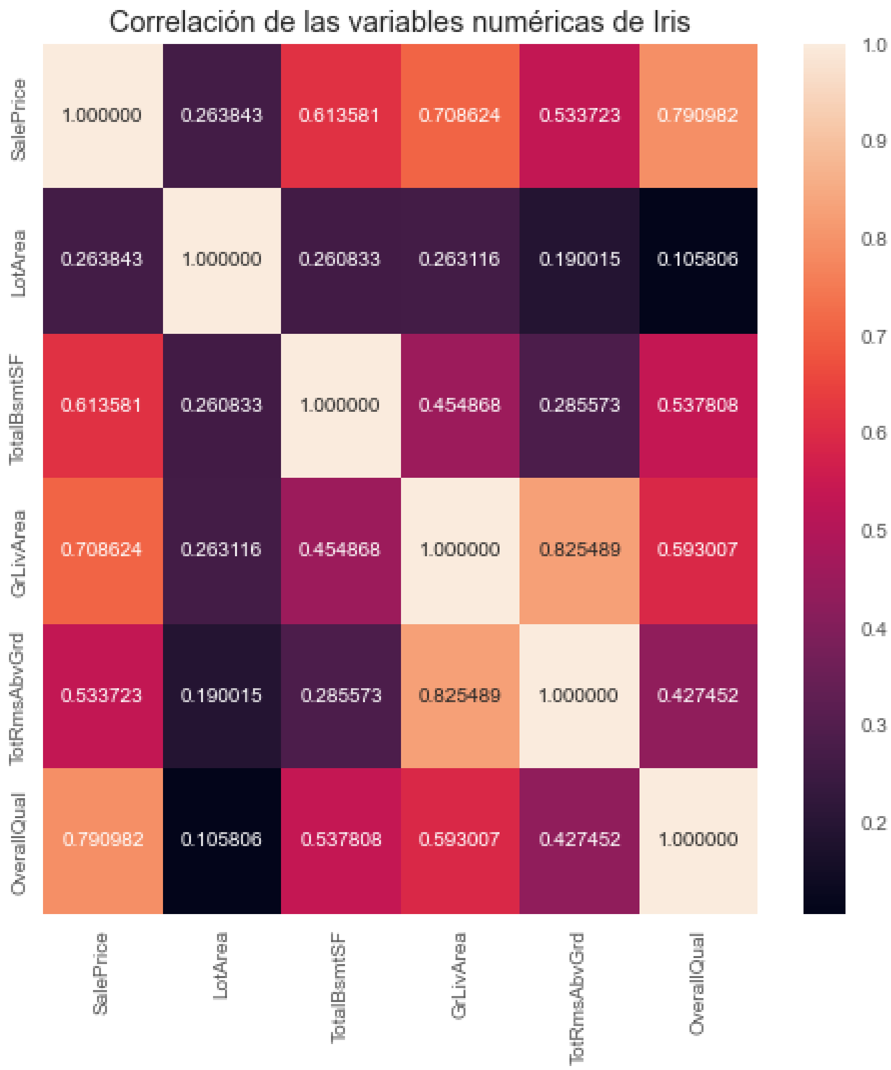
	SalePrice	LotArea	OverallCond	YearBuilt	MasVnrArea	TotalBsmtSF	1stFlrSF	2ndFlrSF	GrLivAre
0	208500	8450	5	2003	196.0	856	856	854	17
1	181500	9600	8	1976	0.0	1262	1262	0	12
2	223500	11250	5	2001	162.0	920	920	866	17
3	140000	9550	5	1915	0.0	756	961	756	17
4	250000	14260	5	2000	350.0	1145	1145	1053	21

In [ ]:
```
sns.pairplot(data[['SalePrice', 'LotArea', 'TotalBsmtSF',
             'GrLivArea', 'TotRmsAbvGrd', 'OverallQual']], hue='SalePrice')
plt.show()
```



In [ ]:
```
plt.subplots(figsize=(8, 8))
sns.heatmap(data[['SalePrice', 'LotArea', 'TotalBsmtSF',
                  'GrLivArea', 'TotRmsAbvGrd', 'OverallQual']].corr(), annot=True, fmt
```

Out[ ]:
```
Text(0.5, 1.0, 'Correlación de las variables numéricas de Iris')
```



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
```

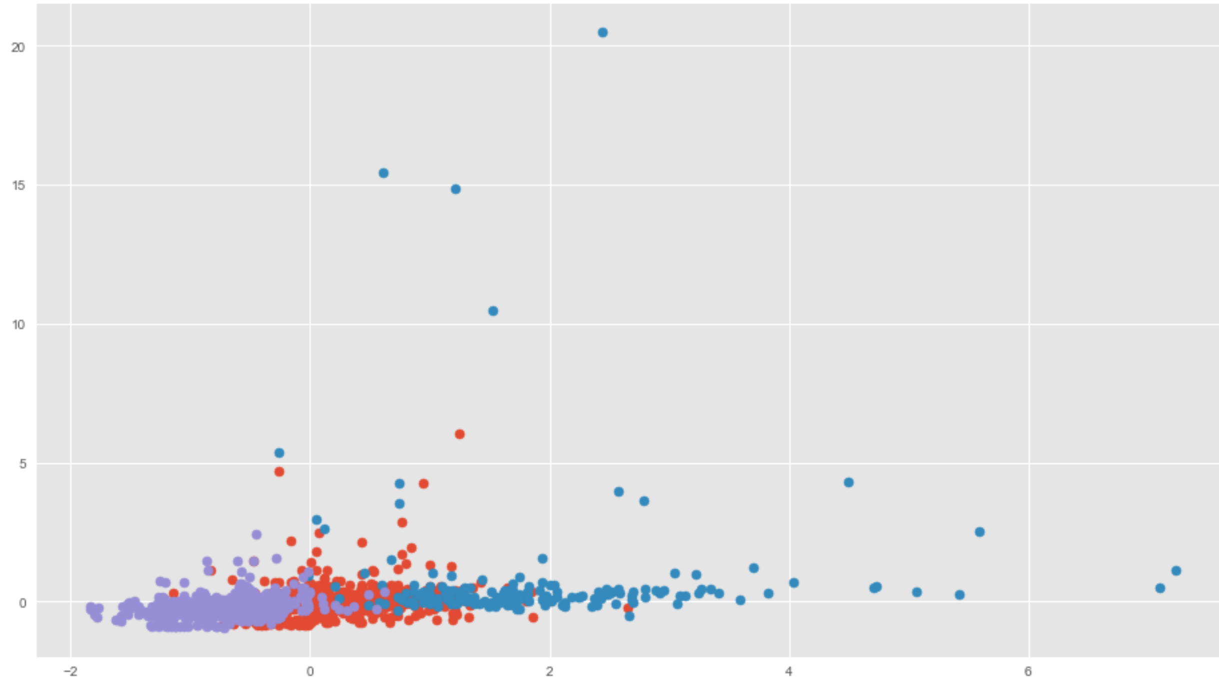
```
<ipython-input-7-ba92df615b8e>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_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\uti
l\_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/us
er_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 [ ]:

```
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())
# ## Variable clasificacion
```

```
               count      mean       std       min       25%       50%  \
SalePrice      617.0  0.184263  0.504249 -1.238898 -0.150061  0.101694
LotArea        617.0 -0.014114  0.575562 -0.861296 -0.245843 -0.071016
OverallCond    617.0 -0.230100  0.750892 -2.314292 -0.517023 -0.517023
YearBuilt      617.0  0.497111  0.761370 -2.955603  0.090461  0.818868
MasVnrArea     617.0 -0.008582  0.831912 -0.572637 -0.572637 -0.572637
TotalBsmtSF    617.0  0.133785  0.794233 -2.410341 -0.495616  0.124390
1stFlrSF       617.0  0.139211  0.861382 -1.726973 -0.578463  0.148409
2ndFlrSF       617.0  0.202150  1.004359 -0.794891 -0.794891  0.542937
GrLivArea      617.0  0.264378  0.624218 -1.060865 -0.162639  0.193226
TotRmsAbvGrd   617.0  0.213899  0.832480 -1.549046 -0.318574  0.296662
GarageCars     617.0  0.370174  0.527753 -2.364630  0.311618  0.311618
WoodDeckSF     617.0  0.048842  0.940505 -0.751918 -0.751918  0.014006
OpenPorchSF    617.0  0.196272  0.961272 -0.704242 -0.704242 -0.070337
EnclosedPorch  617.0 -0.228814  0.631856 -0.359202 -0.359202 -0.359202
PoolArea       617.0 -0.042528  0.649309 -0.068668 -0.068668 -0.068668
OverallQual    617.0  0.338357  0.644535 -1.517947 -0.071812  0.651256
cluster        617.0  0.000000  0.000000  0.000000  0.000000  0.000000

                    75%        max
SalePrice      0.510795   2.657001
LotArea        0.136573   6.035725
OverallCond   -0.517023   3.077516
YearBuilt      1.083743   1.249290
MasVnrArea     0.366246   5.662651
TotalBsmtSF    0.748955   2.458531
1stFlrSF       0.831308   2.574767
2ndFlrSF       1.115638   2.503863
GrLivArea      0.632823   3.576796
TotRmsAbvGrd   0.911897   4.603312
GarageCars     0.311618   2.987865
WoodDeckSF     0.588449   5.056339
OpenPorchSF    0.563567   5.604618
EnclosedPorch -0.359202   4.843750
PoolArea      -0.068668  16.059839
OverallQual    0.651256   2.097391
cluster        0.000000   0.000000
               count      mean       std       min       25%       50%  \
SalePrice      183.0  1.849340  1.146437 -0.426991  1.146475  1.685393
LotArea        183.0  0.732193  2.399829 -0.493908  0.012941  0.186467
OverallCond    183.0 -0.237120  0.851076 -3.212926 -0.517023 -0.517023
YearBuilt      183.0  0.721168  0.818384 -3.021822  0.719540  1.050634
MasVnrArea     183.0  1.377771  1.559570 -0.572637  0.443566  1.106307
TotalBsmtSF    183.0  1.338257  1.249644 -0.821575  0.585975  1.296019
1stFlrSF       183.0  1.372967  1.161639 -0.485341  0.612728  1.353828
2ndFlrSF       183.0  0.561589  1.403018 -0.794891 -0.794891  0.520029
GrLivArea      183.0  1.479626  1.154102  0.004827  0.592860  1.304590
TotRmsAbvGrd   183.0  1.187577  0.989999 -0.933810  0.296662  0.911897
GarageCars     183.0  1.225637  0.655075  0.311618  0.311618  1.649742
WoodDeckSF     183.0  0.798895  1.244846 -0.751918  0.117725  0.732060
OpenPorchSF    183.0  0.677713  1.290782 -0.704242 -0.040151  0.382452
EnclosedPorch  183.0 -0.134790  1.036651 -0.359202 -0.359202 -0.359202
PoolArea       183.0  0.312701  2.312895 -0.068668 -0.068668 -0.068668
OverallQual    183.0  1.457299  0.739935 -0.794879  1.374324  1.374324
cluster        183.0  1.000000  0.000000  1.000000  1.000000  1.000000

                    75%        max
SalePrice      2.384968   7.226343
LotArea        0.428320  20.511245
OverallCond   -0.517023   3.077516
```

```
YearBuilt      1.149962   1.282400
MasVnrArea     2.034144   8.263909
TotalBsmtSF    1.797495  11.517003
1stFlrSF       1.916443   9.129553
2ndFlrSF       1.871602   3.935614
GrLivArea      2.072459   7.852884
TotRmsAbvGrd   2.142369   3.372840
GarageCars     1.649742   2.987865
WoodDeckSF     1.290546   6.085550
OpenPorchSF    1.114461   7.551611
EnclosedPorch -0.359202   8.672338
PoolArea      -0.068668  18.299910
OverallQual    2.097391   2.820459
cluster        1.000000   1.000000
               count      mean       std       min       25%       50%  \
SalePrice      652.0 -0.702017  0.353425 -1.838074 -0.905248 -0.666157
LotArea        652.0 -0.194283  0.346899 -0.923413 -0.364766 -0.204065
OverallCond    652.0  0.292024  1.158806 -4.111561 -0.517023  0.381612
YearBuilt      652.0 -0.684004  0.801768 -3.286697 -1.341520 -0.538617
MasVnrArea     652.0 -0.378584  0.465614 -0.572637 -0.572637 -0.572637
TotalBsmtSF    652.0 -0.510251  0.639002 -2.410341 -0.819296 -0.440910
1stFlrSF       652.0 -0.524907  0.554340 -2.143438 -0.855244 -0.551302
2ndFlrSF       652.0 -0.349168  0.687714 -0.794891 -0.794891 -0.794891
GrLivArea      652.0 -0.671294  0.596720 -2.248350 -1.117956 -0.780169
TotRmsAbvGrd   652.0 -0.536549  0.756840 -2.779517 -0.933810 -0.318574
GarageCars     652.0 -0.700184  0.899593 -2.364630 -1.026506 -1.026506
WoodDeckSF     652.0 -0.267391  0.841029 -0.751918 -0.751918 -0.751918
OpenPorchSF    652.0 -0.385021  0.748821 -0.704242 -0.704242 -0.704242
EnclosedPorch  652.0  0.253752  1.195538 -0.359202 -0.359202 -0.359202
PoolArea       652.0 -0.046680  0.561459 -0.068668 -0.068668 -0.068668
OverallQual    652.0 -0.739429  0.651289 -3.687150 -0.794879 -0.794879
cluster        652.0  2.000000  0.000000  2.000000  2.000000  2.000000

                    75%        max
SalePrice     -0.483635   0.617790
LotArea       -0.050978   2.417847
OverallCond    1.280247   3.077516
YearBuilt     -0.108195   1.183071
MasVnrArea    -0.572637   3.017210
TotalBsmtSF   -0.106973   1.619699
1stFlrSF      -0.213733   1.677170
2ndFlrSF       0.286367   2.895590
GrLivArea     -0.325347   2.065798
TotRmsAbvGrd  -0.318574   2.757604
GarageCars     0.311618   2.987865
WoodDeckSF     0.141660   5.120166
OpenPorchSF   -0.398609   7.189379
EnclosedPorch  0.058016   5.040088
PoolArea      -0.068668  14.267783
OverallQual   -0.071812   1.374324
cluster        2.000000   2.000000
```

```
<ipython-input-9-2a99325cdc2f>: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_
```

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()

minPrice = data['SalePrice'].min()
maxPrice = data['SalePrice'].max()
division = (maxPrice - minPrice) / 3
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'

data['Clasificacion'][data['SalePrice'] < minPrice + division] = 'Economica'
data['Clasificacion'][data['SalePrice'] >= minPrice + division] = 'Intermedia'
data['Clasificacion'][data['SalePrice'] >= minPrice + division * 2] = 'Caras'
```

```
<ipython-input-10-d582a30efa5d>:9: 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['Clasificacion'] = data['LotArea']
<ipython-input-10-d582a30efa5d>:15: 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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  data['Clasificacion'][data['SalePrice'] < minPrice + division] = 'Economica'
C:\Users\ALIEWARE\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas\cor
e\generic.py:8870: 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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  return self._update_inplace(result)
<ipython-input-10-d582a30efa5d>:16: 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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  data['Clasificacion'][data['SalePrice'] >= minPrice + division] = 'Intermedia'
<ipython-input-10-d582a30efa5d>:17: 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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  data['Clasificacion'][data['SalePrice'] >= minPrice + division * 2] = 'Caras'
```

Contamos la cantidad de casas por clasificacion

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

```
Economica     1295
Intermedia     149
Caras            8
Name: Clasificacion, dtype: int64
```

Dividmos en entrenamiento y prueba

Estableciendo los conjuntos de Entrenamiento y Prueba

In [ ]:
```
y = data['Clasificacion']
X = data[['SalePrice', 'LotArea', 'TotalBsmtSF',
          'GrLivArea', 'TotRmsAbvGrd', 'OverallQual']]
```

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[ ]:
```
601      Economica
118     Intermedia
1205     Economica
431      Economica
114      Economica
           ...
408     Intermedia
481     Intermedia
800      Economica
477     Intermedia
1024    Intermedia
Name: Clasificacion, Length: 1016, dtype: object
```

70% de entrenamiento y 30% prueba

In [ ]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1016 entries, 601 to 1024
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   SalePrice     1016 non-null   float64
 1   LotArea       1016 non-null   float64
 2   TotalBsmtSF   1016 non-null   float64
 3   GrLivArea     1016 non-null   float64
 4   TotRmsAbvGrd  1016 non-null   float64
 5   OverallQual   1016 non-null   float64
dtypes: float64(6)
memory usage: 55.6 KB
```

In [ ]:
```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 436 entries, 501 to 410
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   SalePrice     436 non-null    float64
 1   LotArea       436 non-null    float64
 2   TotalBsmtSF   436 non-null    float64
 3   GrLivArea     436 non-null    float64
 4   TotRmsAbvGrd  436 non-null    float64
 5   OverallQual   436 non-null    float64
dtypes: float64(6)
memory usage: 23.8 KB
```

2. Elabore un modelo de bayes ingenuo (naive bayes) utilizando el conjunto de entrenamiento y explique los resultados a los que llega. 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.

Creando el modelo

In [ ]:
```
gaussian = GaussianNB()
modelo = gaussian.fit(X_train, y_train)
```

3. El modelo debe ser de clasificación, use la variable categórica que hizo con el precio de las casas (barata, media y cara) como variable respuesta.

In [ ]:
```
y_pred = gaussian.predict(X_test)
pred = list(y_pred)
print('Economicas', pred.count('Economica'))
print('Intermedias', pred.count('Intermedia'))
print('Caras', pred.count('Caras'))
```

```
Economicas 370
Intermedias 61
Caras 5
```

4. Utilice el modelo con el conjunto de prueba y determine la eficiencia del algoritmo para clasificar.

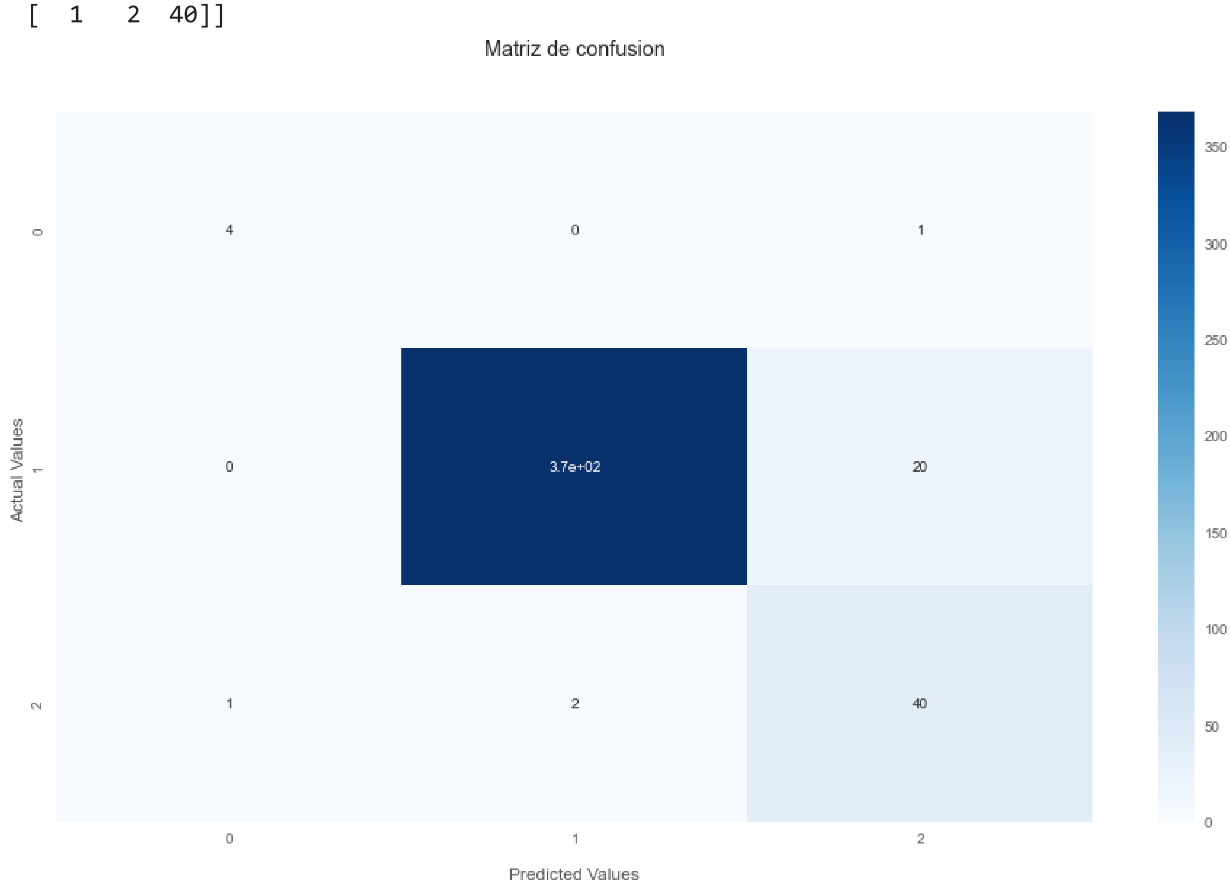
In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='micro')
recall = recall_score(y_test, y_pred, average='micro')
f1 = f1_score(y_test, y_pred, average='micro')
print('Accuracy: ', accuracy)
```

```
Accuracy:  0.944954128440367
```

5. Haga un análisis de la eficiencia del algoritmo usando una matriz de confusión. Tenga en cuenta la efectividad, donde el algoritmo se equivocó más, donde se equivocó menos y la importancia que tienen los errores.

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix for Naive Bayes\n', cm)
graf = sns.heatmap(cm, annot=True, cmap='Blues')
graf.set_title('Matriz de confusion\n\n');
graf.set_xlabel('\nPredicted Values')
graf.set_ylabel('Actual Values ');
plt.show()
p2 = """
Los resultados a los que llegamos al elaborar y analizar
la matriz de confusion utilizando el conjunto de
entrenamiento, son que la precision de la clasificacion
cuenta con alta efectividad
"""
print(p2)
```

```
Confusion matrix for Naive Bayes
 [[  4   0   1]
 [  0 368  20]
 [  1   2  40]]
```



```
Los resultados a los que llegamos al elaborar y analizar
la matriz de confusion utilizando el conjunto de
entrenamiento, son que la precision de la clasificacion
cuenta con alta efectividad
```

6. Analice el modelo. Explique si hay sobreajuste (overfitting) o no.

In [ ]:
```
print(accuracy)
p6 = """
Para ver si existe sobreajuste o no, es necesario ver
el puntaje de Gauss obtenido, el cual es de 0.96, este
valor es algo elevado, pero al no ser tan cercano a 1
como lo es 0.99, podemos decir que  no hay sobreajuste en
el modelo
"""
print(p6)
```

```
0.944954128440367

Para ver si existe sobreajuste o no, es necesario ver
el puntaje de Gauss obtenido, el cual es de 0.96, este
valor es algo elevado, pero al no ser tan cercano a 1
como lo es 0.99, podemos decir que  no hay sobreajuste en
el modelo
```

7. Haga un modelo usando validación cruzada, compare los resultados de este con los del modelo anterior. ¿Cuál funcionó mejor?

Score modelo general

In [ ]:
```
score = gaussian.score(X_train, y_train)

print("Score del modelo en general:", score)
#Usando KFolds
kf = KFold(n_splits=10)
scores = cross_val_score(gaussian, X_train, y_train, cv=kf, scoring="accuracy")
print("KFolds: Metricas de la validacion cruzada:", scores)
print("KFolds: Resultado de la validacion cruzada:", scores.mean())

#Usando StratifiedKFolds
skf = StratifiedKFold(n_splits=10)
scores = cross_val_score(gaussian, X_train, y_train, cv=skf, scoring="accuracy")
print("StratifiedKFolds: Metricas de la validacion cruzada:", scores)
print("StratifiedKFolds: Resultado de la validacion cruzada:", scores.mean())

print("Dado el resultado se puede observar que el que mejor funcionó fue el de KFolds
```

```
Score del modelo en general: 0.9635826771653543
KFolds: Metricas de la validacion cruzada: [0.94117647 0.96078431 0.99019608 0.960784
31 0.97058824 0.94117647
 0.96039604 0.97029703 0.95049505 0.97029703]
KFolds: Resultado de la validacion cruzada: 0.9616191030867792
StratifiedKFolds: Metricas de la validacion cruzada: [0.92156863 0.97058824 0.9901960
8 0.96078431 0.97058824 0.93137255
 0.97029703 0.97029703 0.94059406 0.97029703]
StratifiedKFolds: Resultado de la validacion cruzada: 0.9596583187730537
Dado el resultado se puede observar que el que mejor funcionó fue el de KFolds luego
le sigue StratifiedKFolds y por último el modelo general.
```

```
C:\Users\ALIEWARE\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\mo
del_selection\_split.py:676: UserWarning: The least populated class in y has only 3 m
embers, which is less than n_splits=10.
  warnings.warn(
```

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

Modelo de árbol de decisión

In [ ]:

```
arbol = DecisionTreeClassifier(max_depth=4, random_state=42)
arbol = arbol.fit(X_train, y_train)
score = arbol.score(X_train, y_train)

print("Score arbol de decision:", score)
print("Se puede concluir que dado el score 1 el mejor fue el árbol de desición.")
```

```
Score arbol de decision: 1.0
Se puede concluir que dado el score 1 el mejor fue el árbol de desición.
```