Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Apellidos: Ruano Roca	23/08/2021
	Nombre: Miguel Angel Omar	

Actividad: Árboles y *random forest* para regresión y clasificación

- **Análisis descriptivo de los datos:**

Los datos que se proporcionan hacen referencia a los precios de venta de un inmueble y su rango de precio. Primero se leyó la data desde *housing_train.csv,* luego se describió la data, seguidamente se prosiguió a detectar los nulos para cada columna, de manera que podamos ver que tanta relevancia tiene la data, también se hizo un porcentaje de nulos.

```
'''
FUENTE DE DATOS
'''
import pandas as pd
import numpy as np
from statistics import mode
from pandas.api.types import CategoricalDtype
df = pd.read_csv("housing_train.csv", low_memory=False)
```

✓ 0.6s

```
df.info()
df.describe() # describe la data
df.isnull() # si hay nulos
dataNull=df.isnull().sum() # conteo de nulos para cada columna
print([dataNull[dataNull > 0]])
print([dataNull[dataNull > 0]/df.shape[0]])
```

✓ 0.1s

```
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
```

```
[LotFrontage    0.177397
Alley          0.937671
MasVnrType     0.005479
MasVnrArea     0.005479
BsmtQual       0.025342
BsmtCond       0.025342
BsmtExposure   0.026027
BsmtFinType1   0.025342
BsmtFinType2   0.026027
Electrical     0.000685
FireplaceQu    0.472603
GarageType     0.055479
GarageYrBlt    0.055479
GarageFinish   0.055479
GarageQual     0.055479
GarageCond     0.055479
PoolQC         0.995205
Fence          0.807534
MiscFeature    0.963014
dtype: float64]
```

- **De las variables numéricas, halla datos estadísticos.**

Para las variables numéricas primero se excluyeron los valores de tipo **object** de manera que solo quedaran **int** y **float**. Para estos datos se encontró la media, moda, mediana y el máximo y mínimo.

```
dfnum = df.select_dtypes(exclude=['object']) # excluir valores no numericos
print(dfnum.columns.tolist())
```

✓ 0.4s Python

```
['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice']
```

```
pd.set_option('display.max_columns', None)
dfnum.agg(['min', 'max','median', 'mean'],axis='rows')
```

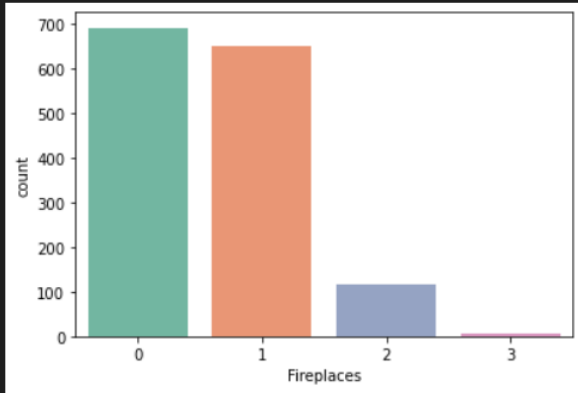
✓ 0.1s Python

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	Bsm
min	1.0	20.00000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0
max	1460.0	190.00000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474
median	730.5	50.00000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0
mean	730.5	56.89726	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46

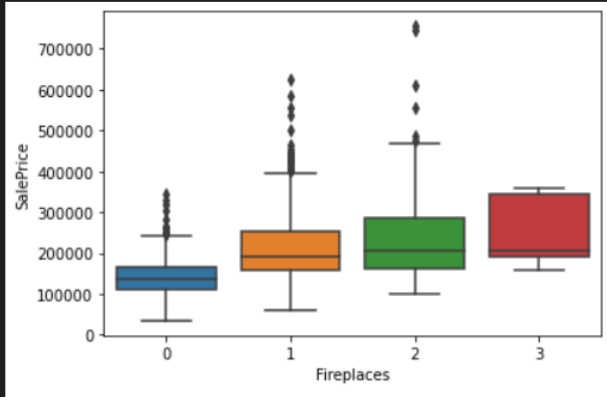
También se analizaron algunas gráficas para entender mejor el comportamiento de los datos.

```
plt.show()
sns.boxplot(x=df["Fireplaces"], y=df["SalePrice"])
```

✓ 0.6s




```
<AxesSubplot:xlabel='Fireplaces', ylabel='SalePrice'>
```

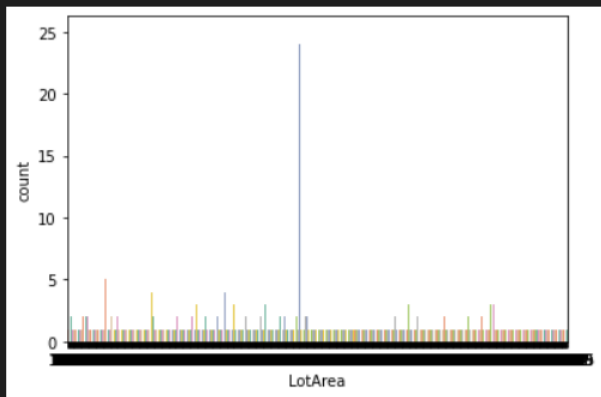


En el caso de las chimeneas se ve que la hay 0, 1, 2 o 3 chimeneas en cada casa. El numero más común de chimeneas en cada casa en 0, seguido de 1 chimenea.

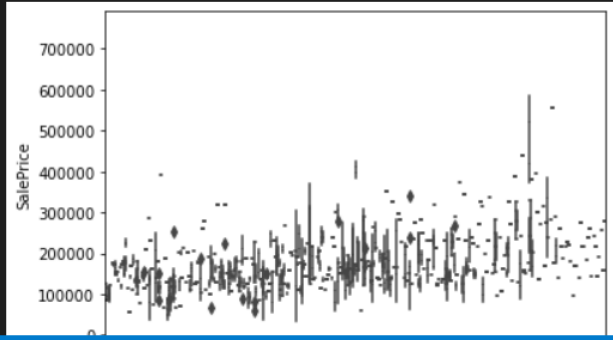
Sin embargo, para los tipos de tipo numérico, algunos tipos de gráfica funcionan mejor, en lugar de graficas de barras se puede realizar una curva de densidad o una gráfica de puntos. Estos dos casos se muestran en las siguientes figuras.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x="LotArea", data=df, palette="Set2")
plt.show()
sns.boxplot(x=df["LotArea"], y=df["SalePrice"])
```

[142] ✓ 104.5s



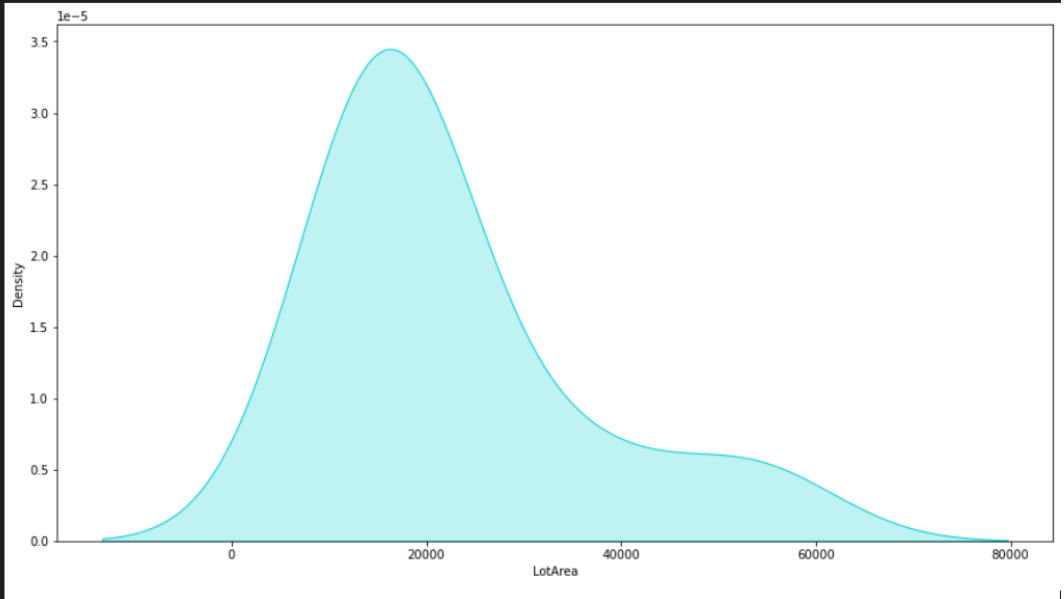

```
<AxesSubplot:xlabel='LotArea', ylabel='SalePrice'>
```




```
plt.figure(figsize=(15, 8))
sns.kdeplot(df["LotArea"][df.SalePrice>500000], color="darkturquoise", shade=True)
```

[8] ✓ 0.3s

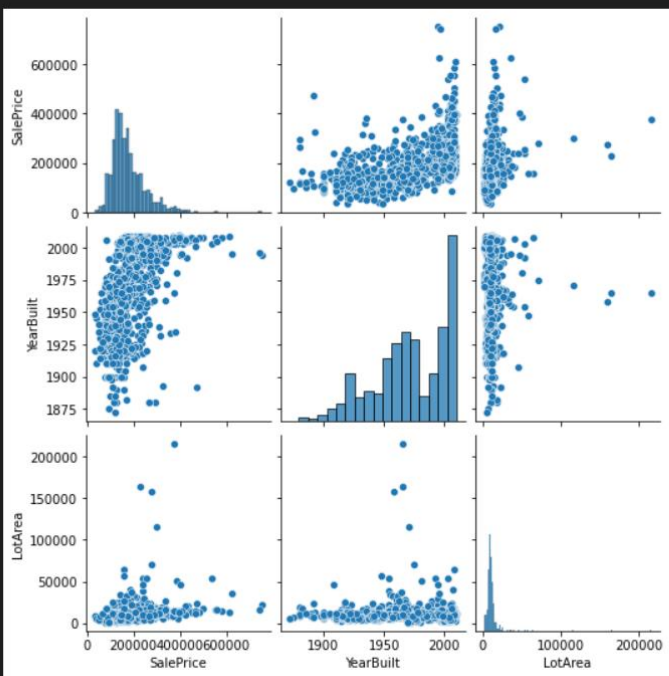
```
<AxesSubplot:xlabel='LotArea', ylabel='Density'>
```



```
cols=["SalePrice", "YearBuilt", "LotArea"]
sns.pairplot(df[cols])
```

[102] ✓ 3.8s Python

```
<seaborn.axisgrid.PairGrid at 0x2ab08109340>
```



- **De las variables categóricas, lista las diferentes categorías y halla la frecuencia de cada una de ellas.**

```
dfcat = df.select_dtypes(exclude=['int64', 'float64']) ## Excluye todos los valoes numericos
for col in dfcat.columns:
    print('-----\nCATEGORIA: ', col)
    print('FRECUENCIAS: ')
    print(dfcat[col].value_counts()) # para pintar el nombre de los elementos unicos de cada
    #print(dfcat[col].value_counts().reset_index(name='freq')) # lo mismo que el anterior per
```

```
-----
CATEGORIA:  MSZoning
FRECUENCIAS:
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64
-----
CATEGORIA:  Street
FRECUENCIAS:
Pave    1454
Grvl       6
Name: Street, dtype: int64
-----
CATEGORIA:  Alley
FRECUENCIAS:
Grvl    50
Pave    41
Name: Alley, dtype: int64
-----
CATEGORIA:  LotShape
FRECUENCIAS:
Reg    925
```

Para cada categoría, se encuentran varios datos y cada dato posee una frecuencia. Por ejemplo, para la columna *MSZoning*, tenemos una frecuencia de 1151 para RL, 218 para EM, 65 para FV, 16 para RH y 10 para C(all).

Estos datos serán luego transformados a un número único para cada opción dentro del rango.

- **Crea matriz de correlaciones existentes entre las variables numéricas del conjunto de datos y analiza los resultados.**

```
dfnum = df.select_dtypes(exclude=['object']) # excluir valores no num
corrTrain=dfnum.corr()
print(corrTrain['SalePrice'].sort_values(ascending=False)[:], '\n')
```

44] ✓ 0.9s

```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
```

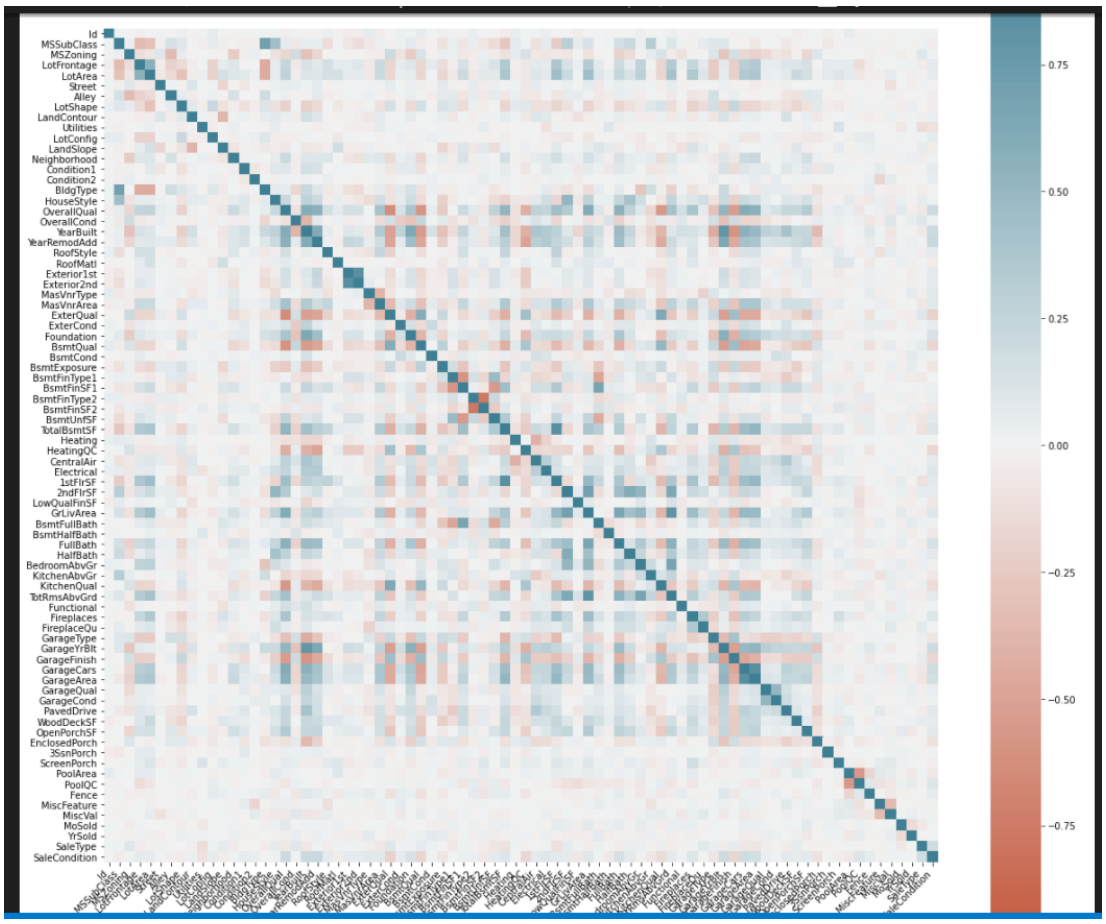
Para los datos de entrenamiento se tiene que respecto a ***SalePrice***, la variable mas relevante para determinar el precio es ***OverallQual,*** la cual asumo que es la calidad de la casa como un todo, o la calidad de materiales y acabados. Luego se observa que es también muy importante la calidad del ***LivingArea***, que sería la sala.

```
dfnumcat2 = dfnumcat.copy()
dfnumcat2 = dfnumcat2.drop(['SalePrice'], axis=1)
dfnumcat2.head()

dfnumcatCorrMtx = dfnumcat2.corr()
print(dfnumcatCorrMtx)
dfnumcatCorrMtx.info()
```

[33] ✓ 0.1s

```
                     Id  MSSubClass  MSZoning  LotFrontage   LotArea  \
Id             1.000000    0.018745 -0.006096    -0.024333 -0.004765
MSSubClass     0.018745    1.000000  0.037785    -0.332077 -0.298528
MSZoning      -0.006096    0.037785  1.000000    -0.117522 -0.080135
LotFrontage   -0.024333   -0.332077 -0.117522     1.000000  0.573462
LotArea       -0.004765   -0.298528 -0.080135     0.573462  1.000000
...                 ...         ...       ...          ...       ...
MiscVal       -0.008427   -0.012910  0.018630     0.005867  0.062162
MoSold         0.021172    0.007052 -0.031496     0.012533 -0.002017
YrSold         0.000712   -0.022100 -0.020628     0.007927 -0.031183
SaleType       0.019773    0.020195  0.097437    -0.035355 -0.021014
SaleCondition -0.005806   -0.033097  0.009494     0.062591  0.062024

                 Street     Alley  LotShape  LandContour  Utilities  ...  \
Id             0.008916 -0.001658  0.032122    -0.019659   0.013324  ...
MSSubClass    -0.025737  0.149647  0.090920    -0.007658  -0.026217  ...
MSZoning       0.087654 -0.329278  0.061887    -0.017854  -0.001192  ...
LotFrontage   -0.043002 -0.192289 -0.154484    -0.032146   0.001029  ...
LotArea       -0.058066 -0.155431 -0.308136    -0.038541   0.038049  ...
...                 ...       ...       ...          ...        ...  ...
MiscVal       -0.097106 -0.027540 -0.021865     0.028295  -0.004242  ...
MoSold         0.003690 -0.021780 -0.033455    -0.011599  -0.051552  ...
YrSold        -0.025043 -0.001392  0.036449     0.020507   0.023353  ...
```

En el mapa de calor se observa que variables como **OverallQul y OverrallCond** tienen mas peso en el precio de venta



- **Tratamiento de missing. Si existen valores faltantes, decide si eliminar los registros o llenarlos con valores como la media, la mediana o la moda, y justifica tu respuesta.**

Para los datos numéricos faltantes considero que la **media** es la medida que se debe aplicar porque, por ejemplo, las medidas de un patio trasero pueden variar unas de otras por milésimas o centésimas y el comprador no notaría la diferencia. Siguiendo esta línea de pensamiento, una moda no determinaría apropiadamente la tendencia central de las medidas.

En cuanto a los datos categóricos considero que la **moda** si funciona porque daría un dato exacto en su representación numérica, si se aplica mediana o moda el resultado se tendrían que ajustar para que coincidiera con algún dato del dominio.

```
## Llenando datos numericos con la media
dfnum = dfnum.fillna(dfnum.mean())
dfnum.head()
dfnum.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 38 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   LotFrontage    1460 non-null   float64
 3   LotArea        1460 non-null   int64
 4   OverallQual    1460 non-null   int64
 5   OverallCond    1460 non-null   int64
 6   YearBuilt      1460 non-null   int64
 7   YearRemodAdd   1460 non-null   int64
 8   MasVnrArea     1460 non-null   float64
 9   BsmtFinSF1     1460 non-null   int64
 10  BsmtFinSF2     1460 non-null   int64
 11  BsmtUnfSF      1460 non-null   int64
 12  TotalBsmtSF    1460 non-null   int64
 13  1stFlrSF       1460 non-null   int64
 14  2ndFlrSF       1460 non-null   int64
 15  LowQualFinSF   1460 non-null   int64
 16  GrLivArea      1460 non-null   int64
 17  BsmtFullBath   1460 non-null   int64
 18  BsmtHalfBath   1460 non-null   int64
```

```
## Completando datos categóricos con la moda
for col in dfcat.columns:
    dfcat[col].fillna(dfcat[col].mode()[0], inplace=True)
dfcat.head()
dfcat.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 43 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSZoning       1460 non-null   object
 1   Street         1460 non-null   object
 2   Alley          1460 non-null   object
 3   LotShape       1460 non-null   object
 4   LandContour    1460 non-null   object
 5   Utilities      1460 non-null   object
 6   LotConfig      1460 non-null   object
 7   LandSlope      1460 non-null   object
 8   Neighborhood   1460 non-null   object
 9   Condition1     1460 non-null   object
 10  Condition2     1460 non-null   object
 11  BldgType       1460 non-null   object
 12  HouseStyle     1460 non-null   object
 13  RoofStyle      1460 non-null   object
 14  RoofMatl       1460 non-null   object
 15  Exterior1st    1460 non-null   object
 16  Exterior2nd    1460 non-null   object
 17  MasVnrType     1460 non-null   object
 18  ExterQual      1460 non-null   object
 19  ExterCond      1460 non-null   object
```

Para las potenciales variables que se iban a quitar utilicé feature_selection de sklearn, que me retornó una lista de variables óptimas. Al listado original le quité los elementos óptimos y me quedó la lista de características a quitar.

```
'''
Seleccion de caracteristicas: DesitionTreeRegressor
'''
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.feature_selection import RFECV

dfnumcat2 = dfnumcat.copy()
dfnumcat2 = dfnumcat2.drop(['SalePrice'], axis=1)

rfecv = RFECV(estimator=DecisionTreeRegressor(), step=1, cv=10, scoring='r2')
rfecv.fit(dfnumcat2, df['SalePrice'])

print("Optimal number of features: %d" % rfecv.n_features_)
listaOptima = list(dfnumcat2.columns[rfecv.support_])

print("Selected features: %s" % listaOptima)
```

✓ 15.9s

```
Optimal number of features: 67
Selected features: ['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'LotShape', 'LandCon
'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle', 'OverallQual',
'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'Exterior1st', 'Exterior2nd', 'MasVnrType
'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC
'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBath', 'FullBath', 'Half
'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces',
'FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageCon
'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'MiscVal',
'MoSold', 'YrSold', 'SaleType', 'SaleCondition']
```

Variables para eliminar:

```
cols = dfnumcat2.columns.tolist()
quitarDeLista = [x for x in cols if x not in listaOptima]
print(quitarDeLista)
```

✓ 0.8s

```
['Street', 'Alley', 'Utilities', 'Condition2', 'RoofMatl', 'Heating', 'LowQualFinSF', 'BsmtHalfBath',
'GarageQual', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature']
```

Procedí a juntar los datos categóricos con los datos numéricos y a quitar las características que no eran óptimas de los datos de entrenamiento, por lo que me quedaron solo 67 columnas.

```
dfnumcat = pd.concat([dfnum, dfcat], axis=1)
columnas = df.columns.tolist()
dfnumcat = dfnumcat[columnas] # Reorden original

dfnumcat = dfnumcat.drop(quitarDeLista, axis=1)

dfnumcat.head()
dfnumcat.info()
```

✓ 0.1s

```
 16  YearRemodAdd   1460 non-null   int64
 17  RoofStyle      1460 non-null   object
 18  Exterior1st    1460 non-null   object
 19  Exterior2nd    1460 non-null   object
show more (open the raw output data in a text

 65  SaleType       1460 non-null   object
 66  SaleCondition  1460 non-null   object
 67  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(32), object(33)
memory usage: 775.8+ KB
```

Sin embargo, luego de ejecutar la regresión y el RandomTree me di cuenta de que las predicciones no eran muy buenas, por lo que al final decidí no quitar ninguna columna del set de entrenamiento.

```
DESEMPEÑO DEL ENTRENAMIENTO... con
[208500.]  vs  208500
[181500.]  vs  181500
[223500.]  vs  223500
[140000.]  vs  140000
[250000.]  vs  250000
[143000.]  vs  143000
[307000.]  vs  307000
[200000.]  vs  200000
[129900.]  vs  129900
[118000.]  vs  118000
[129500.]  vs  129500
[345000.]  vs  345000
[144000.]  vs  144000
[279500.]  vs  279500
[157000.]  vs  157000
[132000.]  vs  132000
```

```
DESEMPEÑO DE LA PRUEBA... con
[158900.]  vs  235000
[755000.]  vs  625000
[155000.]  vs  171000
[165000.]  vs  163000
[147400.]  vs  171900
[214500.]  vs  200500
[210000.]  vs  239000
[290000.]  vs  285000
[143000.]  vs  119500
[62383.]  vs  115000
[66500.]  vs  154900
[85500.]  vs  93000
[335000.]  vs  250000
[235000.]  vs  392500
[755000.]  vs  745000
[125000.]  vs  120000
```

- **Aplica árboles y *random forest* al problema de regresión.**

  - Árboles

Primero se transforman los datos categóricos a una representación numérica

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
class MultiColumnLabelEncoder:
    def __init__(self, columns = None):
        self.columns = columns # Arreglo de nombres de columnas a codificar

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        '''
        Transforma las columnas de X especificadas en self.colums usando LabelEncoder(). Si no h
        '''
        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                output[col] = LabelEncoder().fit_transform(output[col])
        else:
            for colname, col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output

    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

✓ 0.3s

```
cols = dfnumcat.columns.tolist()
dfnumcat = MultiColumnLabelEncoder(columns = cols).fit_transform(dfnumcat)
```

```
dfnumcat = dfnumcat.drop(['SalePrice'], axis=1)
```

✓ 0.2s

Luego se aplica el algoritmo de regresión

```
'''
REGRESSION
'''
from sklearn import metrics
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

#dfnumcat.info()
y = df.SalePrice
Xtrain, Xtest, ytrain, ytest = train_test_split(dfnumcat, y, test_size=0.2, random_state=12345, shuff

dtr = DecisionTreeRegressor(random_state=12345)

dtr = dtr.fit(Xtrain, ytrain)

print(r2_score(list(ytrain), list(dtr.predict(Xtrain))))
print(r2_score(list(ytest), list(dtr.predict(Xtest))))

ypred = dtr.predict(Xtest)
print("sklearn metrics MSE:", mean_squared_error(ytest, ypred))

actNp = np.array(ytest)
actFc = np.array(ypred)

# Mean Squared Error
mse=1/(actNp.size)*(np.sum((actNp-actFc)**2))
print(f'Calculated mse: {mse:.5f} \n')
# Root Mean Squared Error
rmse=np.sqrt(mse)
print(f'rmse: {rmse:.5f} \n')

print(mean_absolute_error(ytest, ypred))
```

Tenemos que el error de la media es de 2983785869.6232877, el error de la raíz es de 54624.04113 y el error absoluto es 29761.705479452055

```
1.0
0.5550154454603338
sklearn metrics MSE: 2983785869.6232877
Calculated mse: 2983785869.62329

rmse: 54624.04113

29761.705479452055
```

- Random Forest

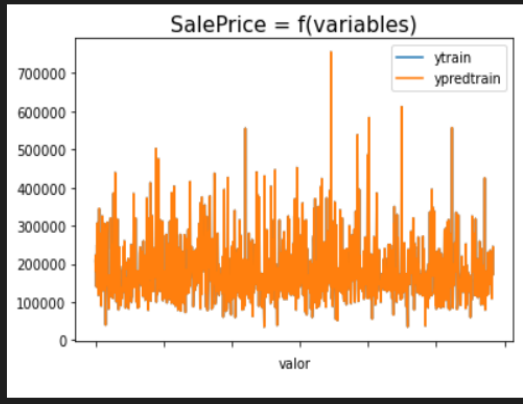
```
from sklearn.ensemble import RandomForestRegressor
dtr = RandomForestRegressor(n_estimators=100)
Xtrain, Xtest, ytrain, ytest = train_test_split(dfnumcat, y, test_size=0.2,
dtr = dtr.fit(Xtrain, ytrain)

from sklearn.metrics import r2_score
print(r2_score(list(ytrain), list(dtr.predict(Xtrain))))
print(r2_score(list(ytest), list(dtr.predict(Xtest))))

ypred = dtr.predict(Xtest)

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

print("sklearn metrics MSE:", mean_squared_error(ytest, ypred))

actNp=np.array(ytest)
actFc=np.array(ypred)

mse=1/(actNp.size)*(np.sum((actNp-actFc)**2))
print(f'Calculated mse: {mse:.5f} \n')

# Root Mean Squared Error
rmse=np.sqrt(mse)
print(f'rmse: {rmse:.5f} \n')

print(mean_absolute_error(ytest, ypred))
```

Se tienen los siguientes resultados

```
0.9799280240706635
0.8095236964116166
sklearn metrics MSE: 1277214000.6815262
Calculated mse: 1277214000.68153

rmse: 35738.13091

19044.170410958905
```

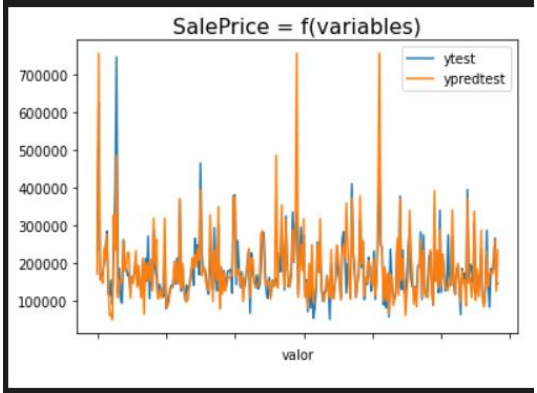
- **Compara, mediante las medidas que te parezcan adecuadas, la capacidad predictiva de ambos métodos.**
  - Árboles

```
PLOTEO DE DATOS DE ENTRENAMIENTO
<ipython-input-171-a3534ef52a11>:17: UserWarning: 
FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotatio
```




```
Train MSE: 0.0
```

```
PLOTEO DE DATOS DE PRUEBA
<ipython-input-171-a3534ef52a11>:28: UserWarning: F
FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation
```



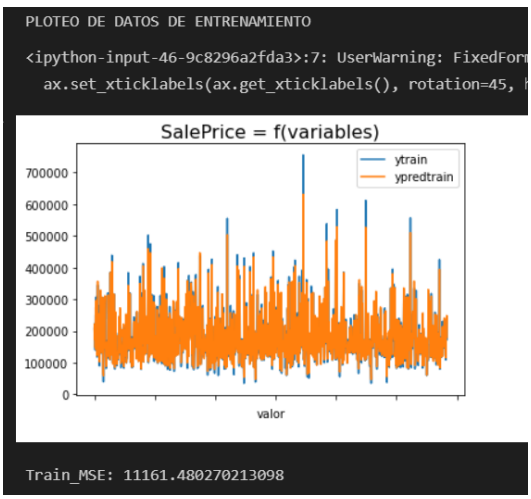

```
Test_MSE:54624.04113230078
```

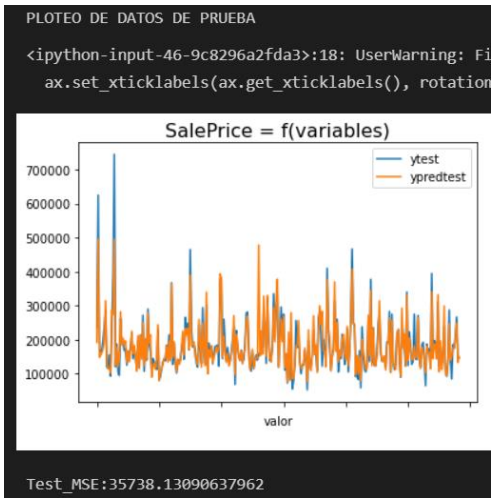
```
DESEMPEÑO DEL ENTRENAMIENTO...
[208500.]  vs  208500
[181500.]  vs  181500
[223500.]  vs  223500
[140000.]  vs  140000
[250000.]  vs  250000
[143000.]  vs  143000
[307000.]  vs  307000
[200000.]  vs  200000
[129900.]  vs  129900
[118000.]  vs  118000
[129500.]  vs  129500
[345000.]  vs  345000
[144000.]  vs  144000
[279500.]  vs  279500
[157000.]  vs  157000
[132000.]  vs  132000
```

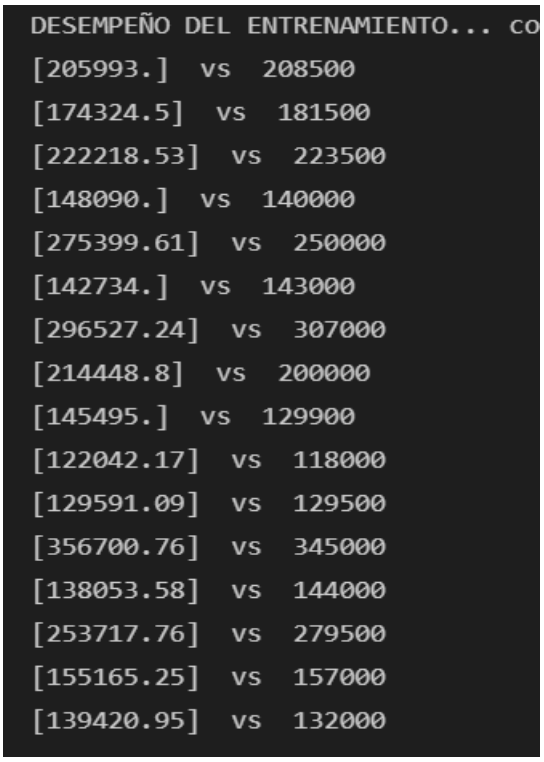
```
DESEMPEÑO DE LA PRUEBA...
[172500.]  vs  235000
[755000.]  vs  625000
[157000.]  vs  171000
[161500.]  vs  163000
[147400.]  vs  171900
[240000.]  vs  200500
[210000.]  vs  239000
[278000.]  vs  285000
[143000.]  vs  119500
[62383.]  vs  115000
[66500.]  vs  154900
[52000.]  vs  93000
[328000.]  vs  250000
[235000.]  vs  392500
[485000.]  vs  745000
[110500.]  vs  120000
```

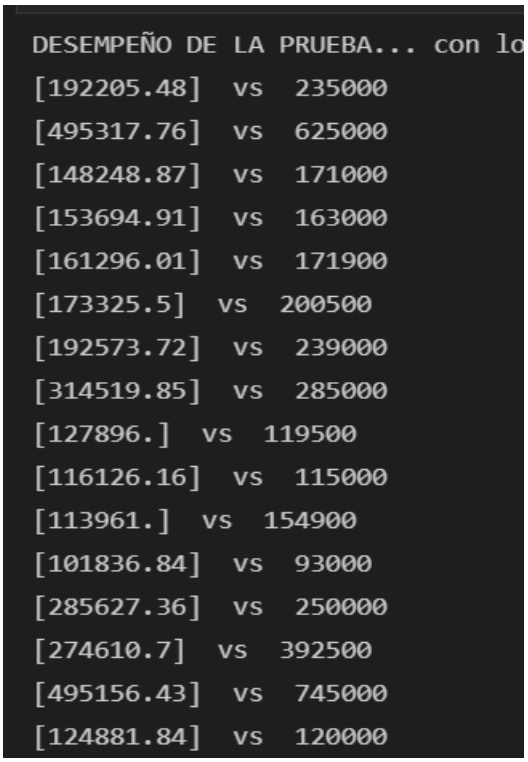
Como podemos apreciar los datos de entrenamiento son iguales a los valores reales, mientras los datos de la prueba son bastante diferentes de lo que deberían, esto nos muestra que estos datos están **sobreentrenados**.

- Random Forest









A diferencia de los datos anteriores, estos se acercan un poco mas a los datos que deberían ser, por lo que están mejores entrenados y ofrecerán mejores resultados en aplicaciones reales

- **Comenta las ventajas y desventajas de cada modelo. De acuerdo con los resultados, ¿son realmente útiles los modelos creados para el conjunto de datos propuesto?**

El árbol es muy útil si se encuentra el numero correcto de parámetros de entrada, el método Random Forest arroja mejores resultados incluso si hay sobrepoblación o si

no se eligen bien los datos de entrada. Los modelos creados pueden ser de utilidad si se desarrollan mas a profundidad, por ahora arrojan resultados muy pobres.

- **Para el ejercicio de clasificación, tanto para árboles como para *random forest*, se crean los siguientes grupos: grupo 1 (SalePrice menor o igual a 100 000), grupo 2 (SalePrice entre 100 001 y 500 000) y grupo 3 SalePrice (mayor o igual a 500 001).**
  - Árboles

```
'''
CLASIFICACION
'''
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import metrics

df['SalePrice'].agg(["min", "max", "median", "mean"], axis="rows")
bins=[34900-1, 100000, 500000, 755000+1]
names=["bajo", "medio", "alto"]
yclass = pd.cut(df["SalePrice"], bins, labels=names)
yclass.head()
yclass.tail()

Xclasstrain, Xclasstest, yclasstrain, yclasstest = train_test_split(dfnumcat, yclass, test_
✓ 0.5s
```

```
max_depth = []
acc_gini = []
acc_entropy = []

for i in range(1, 8):
    dtree = DecisionTreeClassifier(criterion='gini', max_depth=i, random_state=12345)
    dtree.fit(Xclasstrain, yclasstrain)
    yclasspred = dtree.predict(Xclasstest)
    acc_gini.append(metrics.accuracy_score(yclasstest, yclasspred))
    print("i=", i, "gini  :", metrics.accuracy_score(yclasstest, yclasspred) )
    ####
    dtree = DecisionTreeClassifier(criterion="entropy", max_depth=i, random_state=12345)
    dtree.fit(Xclasstrain, yclasstrain)
```

```
for i in range(1, 8):
    dtree = DecisionTreeClassifier(criterion='gini', max_depth=i, random_
    dtree.fit(Xclasstrain, yclasstrain)
    yclasspred = dtree.predict(Xclasstest)
    acc_gini.append(metrics.accuracy_score(yclasstest, yclasspred))
    print("i=", i, "gini  :", metrics.accuracy_score(yclasstest, yclasspr
    ####
    dtree = DecisionTreeClassifier(criterion="entropy", max_depth=i, rand
    dtree.fit(Xclasstrain, yclasstrain)
    yclasspred = dtree.predict(Xclasstest)
    acc_entropy.append(metrics.accuracy_score(yclasstest, yclasspred))
    ####
    max_depth.append(i)
    print("i=", i, "entropy: ", metrics.accuracy_score(yclasstest, yclass
✓ 0.2s

i= 1 gini  : 0.928082191780822
i= 1 entropy:  0.928082191780822
i= 2 gini  : 0.9315068493150684
i= 2 entropy:  0.9417808219178082
i= 3 gini  : 0.9315068493150684
i= 3 entropy:  0.934931506849315
i= 4 gini  : 0.934931506849315
i= 4 entropy:  0.9486301369863014
i= 5 gini  : 0.9315068493150684
i= 5 entropy:  0.958904109589041
i= 6 gini  : 0.928082191780822
i= 6 entropy:  0.9623287671232876
i= 7 gini  : 0.9246575342465754
i= 7 entropy:  0.9452054794520548
```

Se observan resultados similares para el índice de Gini y la entropía para ambos métodos por lo que se puede decir que son bastante precisos.

- Random Forest

```
dfTrain = dfnumcat # no tiene SalePrice
dfTrain.shape
dfnumcat.shape
conditions = [
    (df['SalePrice'] <= 100000),
    (df['SalePrice'] > 100000) & (df['SalePrice'] <= 500000),
    (df['SalePrice'] > 500000)
]

values = ['Plow', 'PMedium', 'PHigh']

yclass = np.select(conditions, values)

Xclasstrain, Xclasstest, yclasstrain, yclasstest = train_test_split(dfnumcat, yclass,
```

✓ 0.3s

```
acc_entropy = []

for i in range(1, 8):
    dtree = RandomForestClassifier(criterion='gini', max_depth=i, random_state=12345)
    dtree.fit(Xclasstrain, yclasstrain)
    yclasspred = dtree.predict(Xclasstest)
    acc_gini.append(metrics.accuracy_score(yclasstest, yclasspred))
    print("i=", i, "gini  :", metrics.accuracy_score(yclasstest, yclasspred) )
    ####
    dtree = RandomForestClassifier(criterion="entropy", max_depth=i, random_state=12345)
    dtree.fit(Xclasstrain, yclasstrain)
    yclasspred = dtree.predict(Xclasstest)
    acc_entropy.append(metrics.accuracy_score(yclasstest, yclasspred))
    ####
    max_depth.append(i)
    print("i=", i, "entropy: ", metrics.accuracy_score(yclasstest, yclasspred))
```

✓ 3.3s

```
i= 1 gini  : 0.928082191780822
i= 1 entropy:  0.928082191780822
i= 2 gini  : 0.9315068493150684
i= 2 entropy:  0.928082191780822
i= 3 gini  : 0.9486301369863014
i= 3 entropy:  0.9383561643835616
i= 4 gini  : 0.952054794520548
i= 4 entropy:  0.952054794520548
i= 5 gini  : 0.952054794520548
i= 5 entropy:  0.9554794520547946
i= 6 gini  : 0.9554794520547946
i= 6 entropy:  0.952054794520548
i= 7 gini  : 0.958904109589041
i= 7 entropy:  0.952054794520548
```

- **Compara los resultados de los dos clasificadores y comenta los resultados.**

  - Árboles

```
dtreeg = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state=12345)
dtreeg = dtreeg.fit(Xclasstrain, yclasstrain)
yclasspred = dtreeg.predict(Xclasstest)
print("Desempeno gini=", metrics.accuracy_score(yclasstest, yclasspred))
```

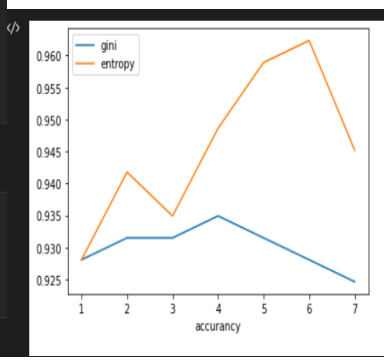
[49] ✓ 0.4s

```
Desempeno gini= 0.928082191780822
```

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=6, random_state=12345)
clf = clf.fit(Xclasstrain, yclasstrain)
yclasspred = clf.predict(Xclasstest)
print("Accurancy:", metrics.accuracy_score(yclasstest, yclasspred))
```

[50] ✓ 0.4s

```
Accurancy: 0.9623287671232876
```



  - Random Forest

```
dtreeg = RandomForestClassifier(criterion='gini', max_de
dtreeg = dtreeg.fit(Xclasstrain, yclasstrain)
yclasspred = dtreeg.predict(Xclasstest)
print("Desempeno gini=", metrics.accuracy_score(yclasste
```

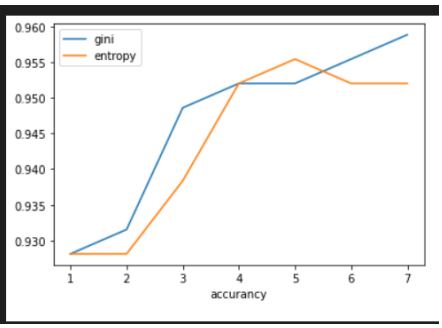
[63] ✓ 0.3s

```
Desempeno gini= 0.9554794520547946
```

```
clf = RandomForestClassifier(criterion="entropy", max_de
clf = clf.fit(Xclasstrain, yclasstrain)
yclasspred = clf.predict(Xclasstest)
print("Accurancy:", metrics.accuracy_score(yclasstest, y
```

[64] ✓ 0.2s

```
Accurancy: 0.952054794520548
```



En este caso tuvo un mejor desempeño el Random Forest y lo podemos ver en su gráfica. Los valores de Gini están mas apegados a los valores de entropía. Una entropía alta significa más información y por lo tanto una mejor predicción de los datos.

- **Otros comentarios que consideres adecuados.**

Existen diferencias entre las metodologías, pero desde mi punto de vista siempre funciona mejor el Random Forest, debido a que da mejores resultados y es más fácil de implementar. Se deben escoger los datos que proporcionen más información al modelo.