Proyecto #1
Aprendizaje Supervisado

Badillo L. Antonio[#1], Bello M. Daniel[*2], Di Pietro Z. José[#3]

[#]*Ingeniería Informática, Universidad Católica Andrés Bello, Montalbán, Caracas, Venezuela*

[1]aabadillo.18@est.ucab.edu.ve, [2]dabello.18@est.ucab.edu.ve,
[3]jadipietro.19@est.ucab.edu.ve

INTRODUCCIÓN

Este informe detalla el proceso del desarrollo de varios modelos de Aprendizaje Automático con el uso de la librería ScikitLearn en Python, donde analizamos, estudiamos, clasificamos y limpiamos, según la necesidad existente, los datos de entrada para dar respuesta a dos (2) problemáticas propuestas a resolver.

Para llegar al modelo de Aprendizaje Automático que nos sea de mayor utilidad debemos realizar ciertos estudios a la data de prueba que tenemos para llegar al modelo indicado que resuelva nuestra problemática.

El Aprendizaje Automático Supervisado es una rama del Aprendizaje Automático en el cual el sistema aprende mediante datos clasificados y etiquetados para predecir resultados, existen dos tipos de aprendizaje supervisado, clasificados y de regresión.

En los modelos de regresión es casi imposible predecir el valor exacto, en vez de esto buscamos encontrar al valor más cercano al valor real, por lo que en nuestro modelo nos centramos en medir lo cerca o lejos que está nuestra predicción al valor real, dado por los datos de prueba suministrados para el modelo.

DESARROLLO

1. Washington Homes for Sale

Se desarrolló un sistema de aprendizaje automático para predecir el precio de las viviendas de Washington en base a una serie de parámetros de entrada.

Luego de importar las librerías a utilizar e importar el dataset proporcionado por la empresa, se realizó una exploración inicial de los datos. Es de suma importancia conocer la naturaleza de los datos suministrados ya que son la “materia prima” del desarrollo.

OBTENIENDO Y VISUALIZANDO LOS DATOS

```
homesOriginal = pd.read_csv("homes.csv")
homes = homesOriginal
```

```
homes.head()
```

7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	

5 rows × 21 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Se debe resaltar que el único atributo que no es numérico es el atributo “date”. Con el método value_counts() podemos examinar mejor dicha columna.

```
homes["date"].value_counts()
```

```
20140623T000000    142
20140626T000000    131
20140625T000000    131
20140708T000000    127
20150427T000000    126
                  ...
20141102T000000      1
20150131T000000      1
20150524T000000      1
20140517T000000      1
20140727T000000      1
Name: date, Length: 372, dtype: int64
```

```
from datetime import datetime
def custom_string_to_year(item):
    str_date = item.split("T")[0]
    return datetime.strptime(str_date, '%Y%m%d').year

def custom_string_to_month(item):
    str_date = item.split("T")[0]
    return datetime.strptime(str_date, '%Y%m%d').month
```

```
homes['dateYEAR'] = homes['date'].transform(custom_string_to_year)
homes['dateMONTH'] = homes['date'].transform(custom_string_to_month)
```

Se puede observar que la fecha de venta de la casa está representada como un string con el siguiente formato: “YmdT000000”, donde:

- Y representa el año de la venta
- M representa el mes de la venta
- d representa el día de la venta
- T000000 no representa ninguna característica significativa.

Se puede observar que se definieron dos funciones para convertir dichos strings que representan una fecha al año y mes correspondientes.

Esto se realiza debido a que si se deja de esta manera no se podrá medir la correlación de la fecha de venta de la casa ni usar este atributo para modelos predictivos.

Se crearon dos columnas adicionales correspondientes al año y mes de venta de la casa (int).

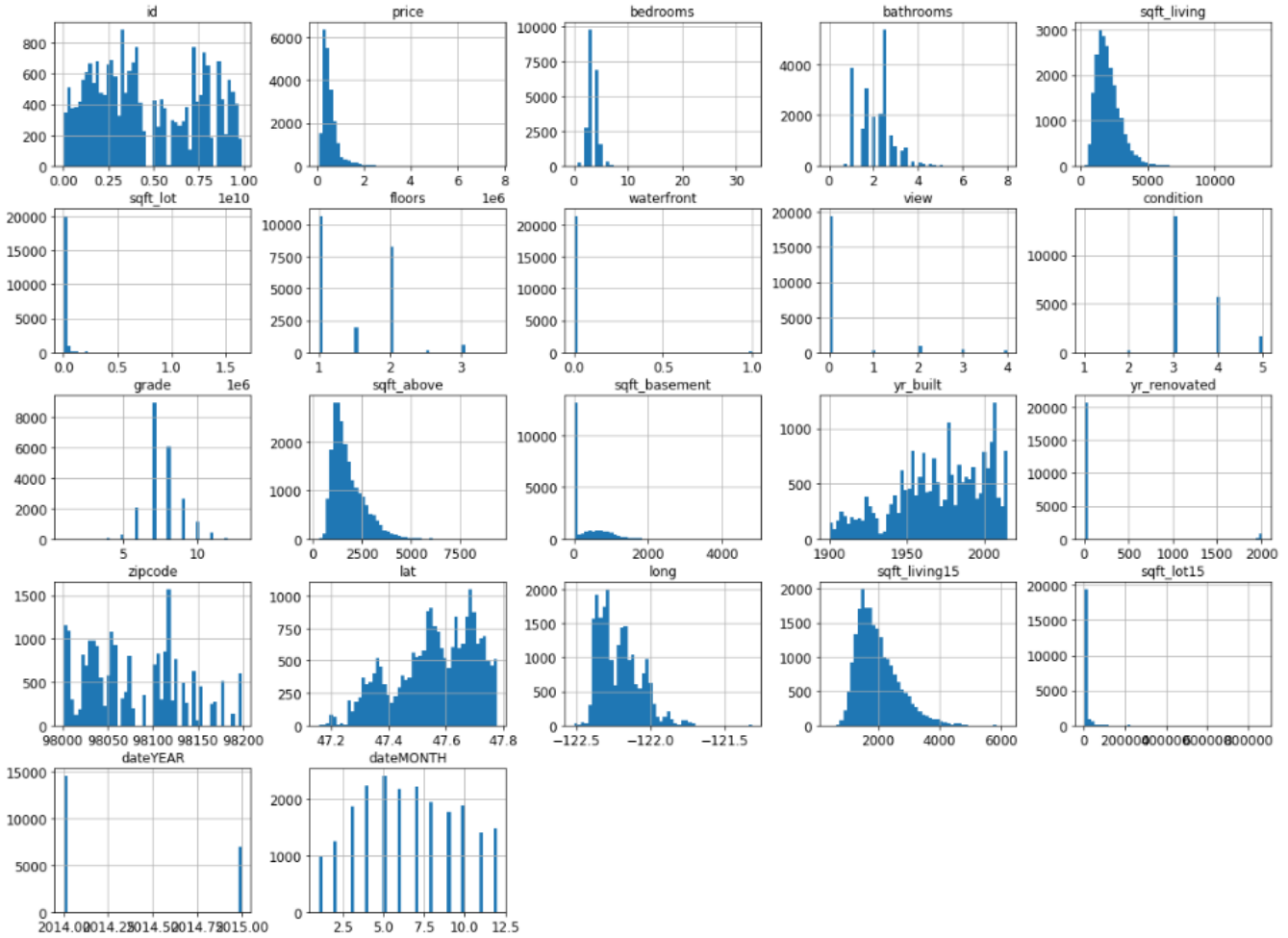
Luego, se usó el método describe() para visualizar y analizar medidas resúmenes de los atributos. En la imágen a continuación se hace énfasis en las dos últimas columnas correspondientes a las creadas en los pasos previos.

```
homes.describe()
```

ıdition	...	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15	dateYEAR	dateMONTH
)00000	...	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
l09430	...	291.509045	1971.005136	84.402258	98077.939805	47.560053	-122.213896	1986.552492	12768.455652	2014.322954	6.574423
350743	...	442.575043	29.373411	401.679240	53.505026	0.138564	0.140828	685.391304	27304.179631	0.467616	3.115308
)00000	...	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000	2014.000000	1.000000
)00000	...	0.000000	1951.000000	0.000000	98033.000000	47.471000	-122.328000	1490.000000	5100.000000	2014.000000	4.000000
)00000	...	0.000000	1975.000000	0.000000	98065.000000	47.571800	-122.230000	1840.000000	7620.000000	2014.000000	6.000000
)00000	...	560.000000	1997.000000	0.000000	98118.000000	47.678000	-122.125000	2360.000000	10083.000000	2015.000000	9.000000
)00000	...	4820.000000	2015.000000	2015.000000	98199.000000	47.777600	-121.315000	6210.000000	871200.000000	2015.000000	12.000000

Se visualiza y analiza la distribución mediante histogramas de frecuencia.

```
homes.hist(bins=50, figsize=(20,15))
plt.show()
```



Luego se corrobora que no existen atributos NULL. Es importante esta verificación ya que los algoritmos a continuación no admiten registros incompletos.

Corroborando que no exiten atributos NULL

```
incomplete_colums = homes.isnull().any()
incomplete_colums
```

```
5]: id              False
    date            False
    price           False
    bedrooms        False
    bathrooms       False
    sqft_living     False
    sqft_lot        False
    floors          False
    waterfront      False
    view            False
    condition       False
    grade           False
    sqft_above      False
    sqft_basement   False
    yr_built        False
    yr_renovated    False
    zipcode         False
    lat             False
    long            False
    sqft_living15   False
    sqft_lot15      False
    dateYEAR        False
    dateMONTH       False
    dtype: bool
```

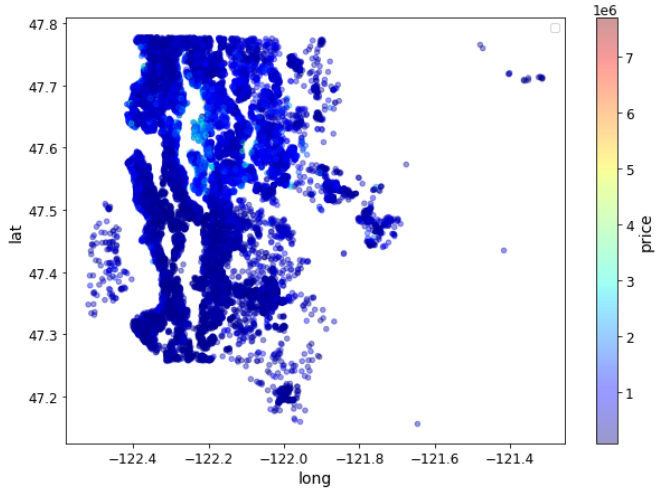
En este caso, no existen atributos NULL.

Se visualiza mediante la longitud y latitud los registros de propiedades.

```
homes.plot(kind="scatter", x="long", y="lat", alpha=0.4,
             figsize=(10,7),
            c="price", cmap=plt.get_cmap("jet"), colorbar=True,
            sharex=False)
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when lege
nd() is called with no argument.
```

```
<matplotlib.legend.Legend at 0x184b6f38400>
```



Luego se verifica la correlación. Esta es una de las partes más importantes, ya que se puede verificar aquellos atributos con correlación positiva o negativa más fuerte con respecto al precio. Se verifica correlación con respecto al precio debido a que esta es la variable que se desea predecir.

VERIFICANDO CORRELACIÓN

```
corr_matrix = homes.corr()
```

```
corr_matrix["price"].sort_values(ascending=False)
```

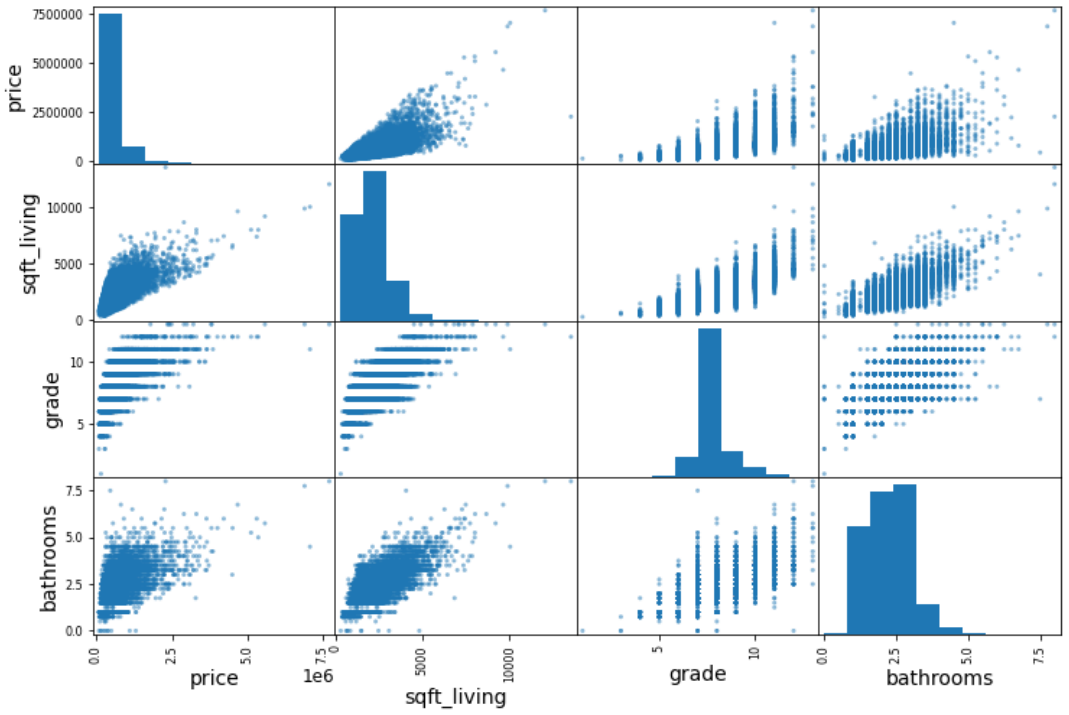
```
9]: price            1.000000
    sqft_living      0.702035
    grade            0.667434
    sqft_above       0.605567
    sqft_living15    0.585379
    bathrooms        0.525138
    view             0.397293
    sqft_basement    0.323816
    bedrooms         0.308350
    lat              0.307003
    waterfront       0.266369
    floors           0.256794
    yr_renovated     0.126434
    sqft_lot         0.089661
    sqft_lot15       0.082447
    yr_built         0.054012
    condition        0.036362
    long             0.021626
    dateYEAR         0.003576
    dateMONTH       -0.010081
    id              -0.016762
    zipcode         -0.053203
    Name: price, dtype: float64
```

Podemos observar que los atributos sqft_living , grade , sqft_above , sqft_living15 y bathrooms son las más correlacionadas al precio.

Existen atributos que analizando la naturaleza de los mismos se podía saber que no se correlacionaba, como el zipcode y el id.

Podemos observar que los atributos dateYEAR y dateMONTH (creados anteriormente) tienen una correlación débil con respecto al precio.

A continuación se puede observar una gráfica de las correlaciones entre los atributos más destacados.



Luego se procedió a excluir los atributos con correlación débil con respecto al precio.

Excluyendo atributos no correlacionados al precio

In [191]:
```
homesDeleted = homes
```

In [192]:
```
homesDeleted.drop('zipcode' , inplace=True , axis=1)
homesDeleted.drop('id' , inplace=True , axis=1)
homesDeleted.drop('dateMONTH' , inplace=True , axis=1)
homesDeleted.drop('dateYEAR' , inplace=True , axis=1)
homesDeleted.drop('date' , inplace=True , axis=1)
```

In [193]:
```
homesDeleted.head()
```

Es importante mencionar que los modelos que a continuación se van a explicar se probaron excluyendo mayor y menor cantidad de atributos no correlacionados. El mejor resultado fue el mostrado en la imágen superior.

Luego se generaron los subconjuntos de entrenamiento y prueba (80% para entrenamiento y 20% para prueba).

GENERANDO LOS CONJUNTOS DE ENTRNAMIENTO Y PRUEBA

In [194]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(homesDeleted, test_size=0.2, random_state=42)
```

Se comenzaron a probar modelos: se comenzó con regresión lineal y árbol de decisión. Para ambos se calculó el RMSE.

In [197]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(homes_train_X, homes_train_Y)
```

Out[197]:
```
LinearRegression()
```

In [198]:
```
some_labels = homes_test_Y[:5]
some_data_prepared = homes_test_X[:5]

print("Predictions:", lin_reg.predict(homes_test_X))
```

```
Predictions: [ 466542.01642822  776114.26499263 1212668.57336035 ...  390487.32120099
  593654.26179958  409770.27218117]
```

In [199]:
```
print("Labels:", list(some_labels))
```

```
Labels: [365000.0, 865000.0, 1038000.0, 1490000.0, 711000.0]
```

In [200]:
```
from sklearn.metrics import mean_squared_error

homes_predictions = lin_reg.predict(homes_test_X)
lin_mse = mean_squared_error(homes_test_Y, homes_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[200]:
```
214472.75555898252
```

Árbol de desición

In [201]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(homes_train_X, homes_train_Y)
```

Out[201]:
```
DecisionTreeRegressor(random_state=42)
```

In [202]:
```
homes_predictions = tree_reg.predict(homes_test_X)
tree_mse = mean_squared_error(homes_test_Y, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[202]:
```
3693.924683758535
```

In [203]:
```
homes_predictions[:5]
```

Out[203]:
```
array([ 435000.,  825000., 1355000., 1815000.,  712000.])
```

In [204]:
```
homes_test_Y[:5]
```

Out[204]:
```
735       365000.0
2830      865000.0
4106     1038000.0
16218    1490000.0
19964     711000.0
Name: price, dtype: float64
```

Luego, se aplicó cross validation a los modelos anteriores con CV=10. Se imprimieron en pantalla los scores, así como su promedio y su desviación estándar.

Cross validation

In [205]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

Cross validation en regresión lineal

In [206]:
```
lin_scores = cross_val_score(lin_reg, homes_train_X, homes_train_Y,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [190642.78319582 250244.2153095  188685.90803984 224085.37580099
 177001.25239054 212662.06946543 186323.95153198 190092.19104418
 184527.89053866 186965.81558721]
Mean: 199123.1452904144
Standard deviation: 21670.73937403698
```

Cross validation en árbol de desición

In [207]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, homes_train_X, homes_train_Y,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
display_scores(tree_rmse_scores)
```

```
Scores: [170060.51182581 227254.6275121  168666.19698988 169388.98772151
 163988.62730234 187829.88575147 167016.52308467 202127.30142018
 195557.76502708 169982.22328273]
Mean: 182187.26499177795
Standard deviation: 19604.23384824147
```

Luego se implementó un modelo de ensamblaje. Se eligió usar RandomForestRegressor. De igual manera se aplicó cross validation. Se notó una disminución en el RMSE.

Modelos de ensamblaje

In [208]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=20, random_state=42)
forest_reg.fit(homes_train_X, homes_train_Y)
```

Out[208]:
```
RandomForestRegressor(n_estimators=20, random_state=42)
```

In [209]:
```
homes_predictions = forest_reg.predict(homes_test_X)
forest_mse = mean_squared_error(homes_test_Y, homes_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[209]:
```
151327.30223316693
```

In [210]:
```
homes_predictions[:5]
```

Out[210]:
```
array([ 368957.5,  908608. , 1039412.5, 1891750. ,  715395. ])
```

In [211]:
```
homes_test_Y[:5]
```

Out[211]:
```
735       365000.0
2830      865000.0
4106     1038000.0
16218    1490000.0
19964     711000.0
Name: price, dtype: float64
```

In [212]:
```
forest_scores = cross_val_score(forest_reg, homes_train_X, homes_train_Y,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [128374.31996434 161394.67161798 125570.2177859  141551.81296563
 126459.69757837 138759.45694246 119881.08052899 125894.58254023
 129919.32310763 121600.24747426]
Mean: 131940.54105057853
Standard deviation: 11759.038061610729
```

En vista a que este es el mejor modelo, se afinaron sus hiper parámetros usando grid search. Se obtuvo que la mejor combinación de hiper parámetros fué {'max_features': 8, 'n_estimators': 100}.

Observación: En base a que probar con hiper parámetros más elevados, se dejó como mayor n_estimators: 100.

AFINANDO LOS MODELOS: Grid Search

Por razones de poder de cómputo, no se evaluarán mas hiperparámetros ni un numero muy elevado de n_estimators

In [213]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [10, 25, 100], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [10, 25], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(homes_train_X, homes_train_Y)
```

Out[213]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [10, 25, 100]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [10, 25]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [214]:
```
grid_search.best_params_
```

Out[214]:
```
{'max_features': 8, 'n_estimators': 100}
```

In [215]:
```
grid_search.best_estimator_
```

Out[215]:
```
RandomForestRegressor(max_features=8, random_state=42)
```

Luego se eligió la mejor combinación de hiper parámetros y se implementó como modelo final.

MODELO FINAL

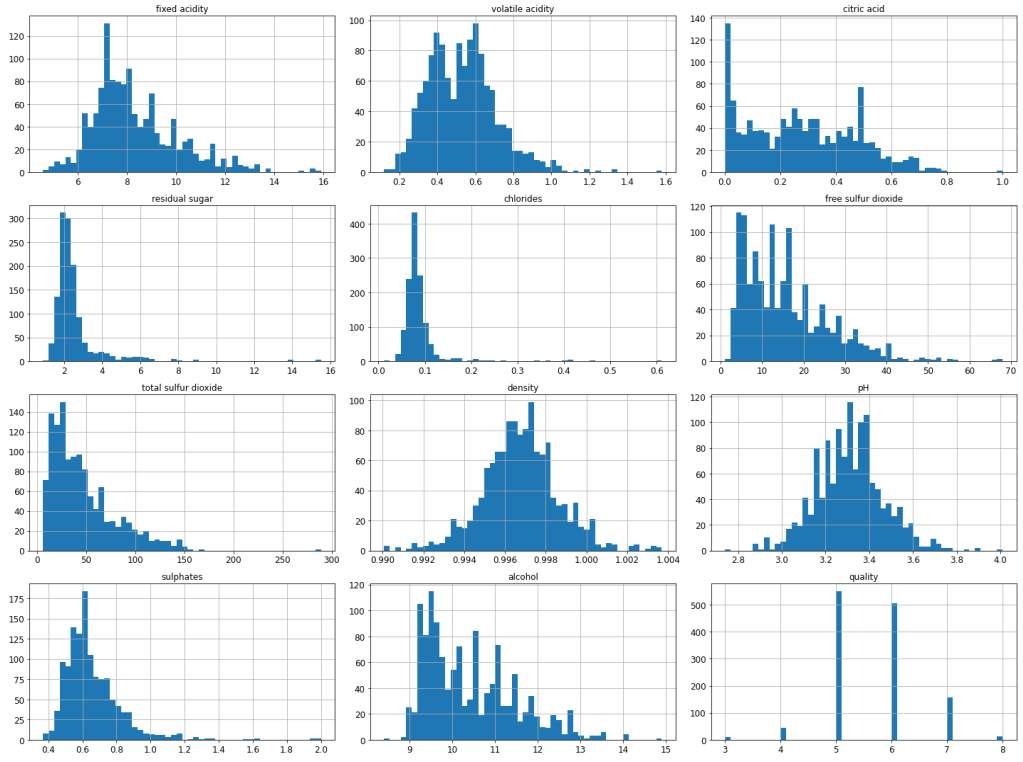
```
In [217]: final_model = grid_search.best_estimator_

          final_predictions = final_model.predict(homes_test_X)
          final_mse = mean_squared_error((homes_test_Y), final_predictions)
          final_rmse = np.sqrt(final_mse)

In [218]: final_rmse

Out[218]: 152836.10239549784
```

2. Santa Clara Winery

El objetivo del problema Santa Clara Winery es predecir la calidad de un vino, analizando una serie de variables fisicoquímicas como la acidez, el ácido cítrico, el azúcar residual, los cloruros, etc. El dataset dado tiene casi 1600 registros, cada uno con las columnas/variables que se muestran a continuación.

```
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
```

La variable que nos interesa predecir es *quality* o calidad del vino, todas las demás son las variables fisicoquímicas que componen al vino. La variable de calidad se encuentra entre los valores de 0 y 10, siendo 0 el peor vino, y 10 el mejor. Pero, como se observa a continuación, ninguno de los registros cumplen con ese rango.



Como se puede ver, la variable de calidad del vino se encuentra principalmente entre los valores 5 y 6, además, no hay ningún vino con valor menor a 3, ni mayor a 6. Inclusive, los valores de calidad son solamente enteros. Con esto se puede concluir que el dataset no está bien balanceado, y el valor de calidad no tiene una buena especificidad o granularidad.

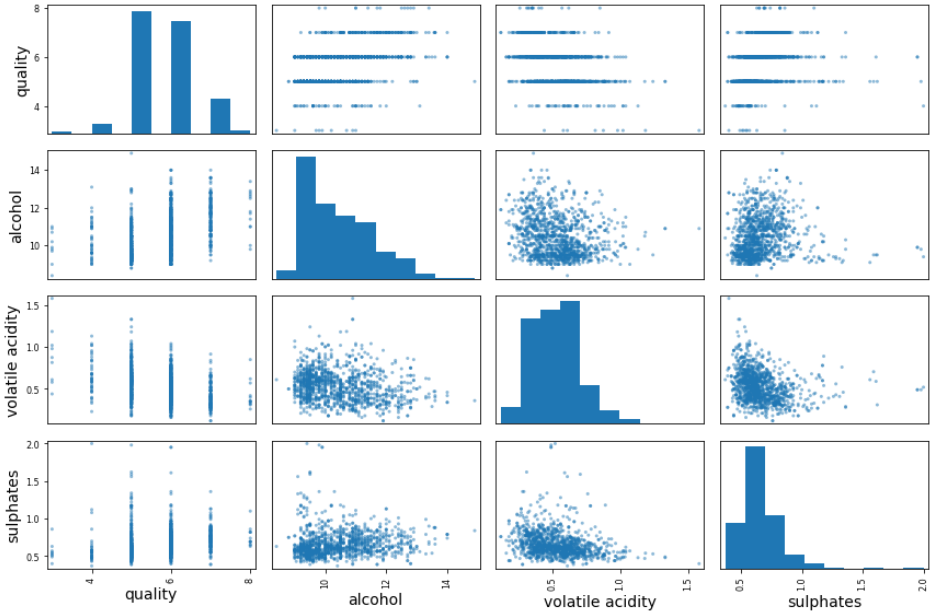
Finalmente, se procedió a visualizar la correlación entre los atributos del dataset.

```
winesData = train_set.copy()
```

```
corr_matrix = winesData.corr()
corr_matrix["quality"].sort_values(ascending=False)
```

```
quality                1.000000
alcohol                0.472676
sulphates              0.242596
citric acid            0.216115
fixed acidity          0.122488
residual sugar         0.005425
pH                    -0.045185
free sulfur dioxide   -0.055860
chlorides             -0.126541
density               -0.167091
total sulfur dioxide  -0.200067
volatile acidity      -0.378372
Name: quality, dtype: float64
```

Se observa que el nivel de alcohol, además de la acidez volátil, está altamente relacionado con la calidad del vino.



Luego de visualizar la data, se procedió a limpiar los campos, en busca de tipos de datos no numéricos y valores nulos. En este caso, no hizo falta la limpieza de datos ya que el dataset solo contiene atributos numéricos.

Se entrenan y evalúan varios modelos de aprendizaje con el conjunto de entrenamiento. Se decidió entrenar y probar con 4 tipos de modelos diferentes: Regresión lineal, Árbol de decisión, Random Forest y SVM. Para cada modelo los resultados del **error cuadrático medio** fueron:

- ➔ Regresion Lineal: 0.6513
- ➔ Random Forest: 0.2183
- ➔ SVM: 0.6567
- ➔ Decision Tree (Árbol de decisión): 0.0

Como se observa, el resultado con el árbol de decisión es 0.0, lo cual indica que el modelo pudiese estar sobre ajustado. Para la evaluación final con los datos de prueba se usaron los modelos de SVM y Random Forest. Además, se usó Random Search para entonar los hiper parámetros. Los resultados finales del **error cuadrático medio** fueron:

- ➔ SVM: 0.62982 , con un intervalo de confianza de [0.57824851, 0.67748118]
- ➔ Random Forest: 0.54637 , con un intervalo de confianza de [0.49074965, 0.59682896]

Conclusión

A la hora de realizar algoritmos para modelos de Aprendizaje Automático Supervisado debemos realizar un estudio de los datos para verificar y detallar la correlación existente entre los atributos de nuestro conjunto de datos, verificar si están balanceados o no, filtrar los datos relevantes a nuestra problemática para así limpiar el ruido que puedan generar atributos no correlacionados, entre otras características.

En el primer problema desarrollamos un sistema de aprendizaje automático el cual intenta predecir el precio de las viviendas en Washington, a la hora de realizar el análisis de la data podemos observar que los indicadores más relacionados a este valor son los que indican las dimensiones, calidad de diseño y construcción de la vivienda, gracias a esto podemos limpiar nuestro conjunto de datos para generar un modelo más preciso.

Para el segundo problema desarrollamos un sistema de aprendizaje automático que intente predecir la calidad del vino, al realizar el estudio del conjunto de datos pudimos observar que los datos no se encuentran muy balanceados, ya que contamos con más volumen de datos en vinos con puntuación entre 5 y 6.

Para este problema dado que el indicador del error cuadrático medio (RMSE) resultó ser bajo podemos concluir que nuestro modelo tiene un buen ajuste para la predicción de nuestro valor, en este caso, la calidad del vino.

Referencias

[1] Géron Aurélien. (2019) *"Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow"*. O'Reilly Media, Inc.