main

August 11, 2024

0.1 Lab 1: Machine Learning Engineering

- Pablo Escobar 20936
- Diego Cordova 20212
- Paola Contreras 20213
- Paola de León 20361
- Jun Woo 20358
- Sebastian Gonzalez 18588

Repositorio: https://github.com/Diego-CB/Lab1-MLE

[ ]:
```python
import pandas as pd
```

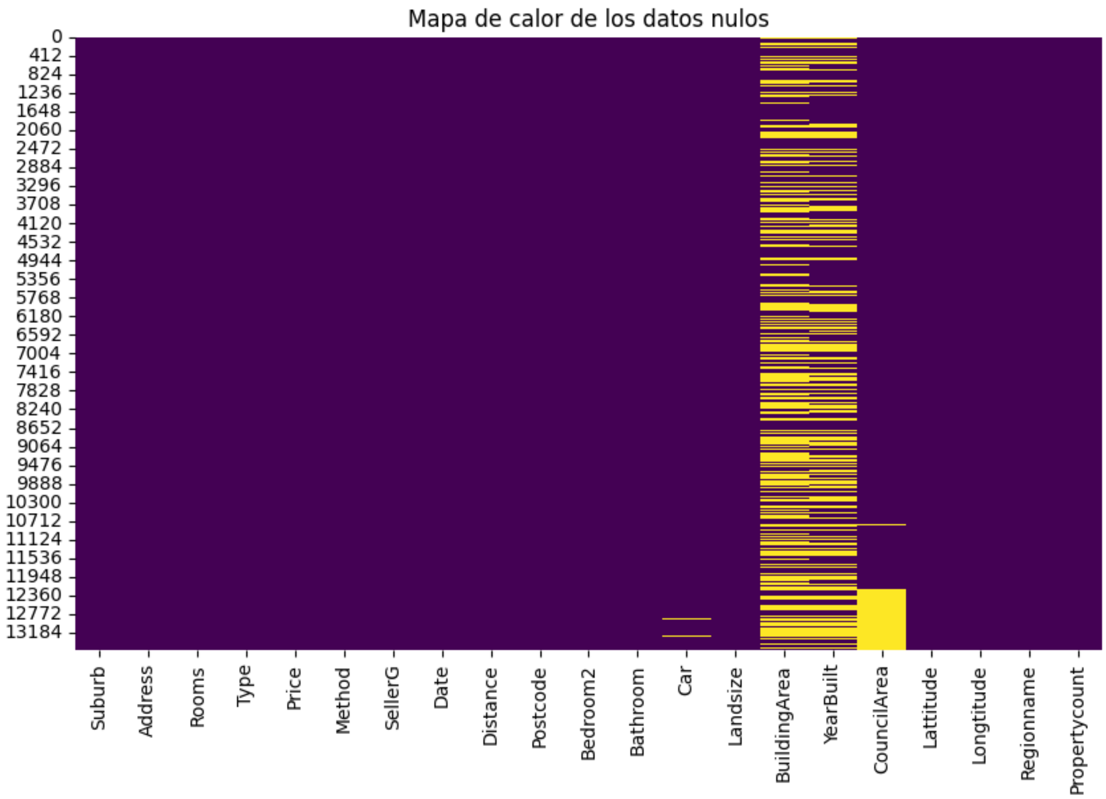
[ ]:
```python
# save filepath to variable for easier access
melbourne_file_path = './input/melbourne-housing-snapshot/melb_data.csv'
# read the data and store data in DataFrame titled melbourne_data
melbourne_data = pd.read_csv(melbourne_file_path)
# print a summary of the data in Melbourne data
melbourne_data.describe()
```

[ ]:
```
              Rooms         Price      Distance      Postcode      Bedroom2  \
count  13580.000000  1.358000e+04  13580.000000  13580.000000  13580.000000
mean       2.937997  1.075684e+06     10.137776   3105.301915      2.914728
std        0.955748  6.393107e+05      5.868725     90.676964      0.965921
min        1.000000  8.500000e+04      0.000000   3000.000000      0.000000
25%        2.000000  6.500000e+05      6.100000   3044.000000      2.000000
50%        3.000000  9.030000e+05      9.200000   3084.000000      3.000000
75%        3.000000  1.330000e+06     13.000000   3148.000000      3.000000
max       10.000000  9.000000e+06     48.100000   3977.000000     20.000000

           Bathroom           Car       Landsize  BuildingArea    YearBuilt  \
count  13580.000000  13518.000000   13580.000000   7130.000000  8205.000000
mean       1.534242      1.610075     558.416127    151.967650  1964.684217
std        0.691712      0.962634    3990.669241    541.014538    37.273762
min        0.000000      0.000000       0.000000      0.000000  1196.000000
25%        1.000000      1.000000     177.000000     93.000000  1940.000000
50%        1.000000      2.000000     440.000000    126.000000  1970.000000
75%        2.000000      2.000000     651.000000    174.000000  1999.000000
max        8.000000     10.000000  433014.000000  44515.000000  2018.000000
```

```
        Lattitude    Longtitude  Propertycount
count  13580.000000  13580.000000  13580.000000
mean     -37.809203    144.995216   7454.417378
std        0.079260      0.103916   4378.581772
min      -38.182550    144.431810    249.000000
25%      -37.856822    144.929600   4380.000000
50%      -37.802355    145.000100   6555.000000
75%      -37.756400    145.058305  10331.000000
max      -37.408530    145.526350  21650.000000
```

Aqui estamos realizando un analisis exploratorio de los datos numericos para poder observar como se comportan los datos. Cone stos datos podemos afirmar cosas como que en promedio, las propiedades tienen aproximadamente 2.94 habitaciones y un precio medio de alrededor de 1,075,684, con una desviación estándar considerable de $639,311, lo que indica una dispersión significativa en los precios.

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))  # Ajusta el tamaño de la figura
sns.heatmap(melbourne_data.isnull(), cmap='viridis', cbar=False)
plt.title('Mapa de calor de los datos nulos')
plt.show()
```



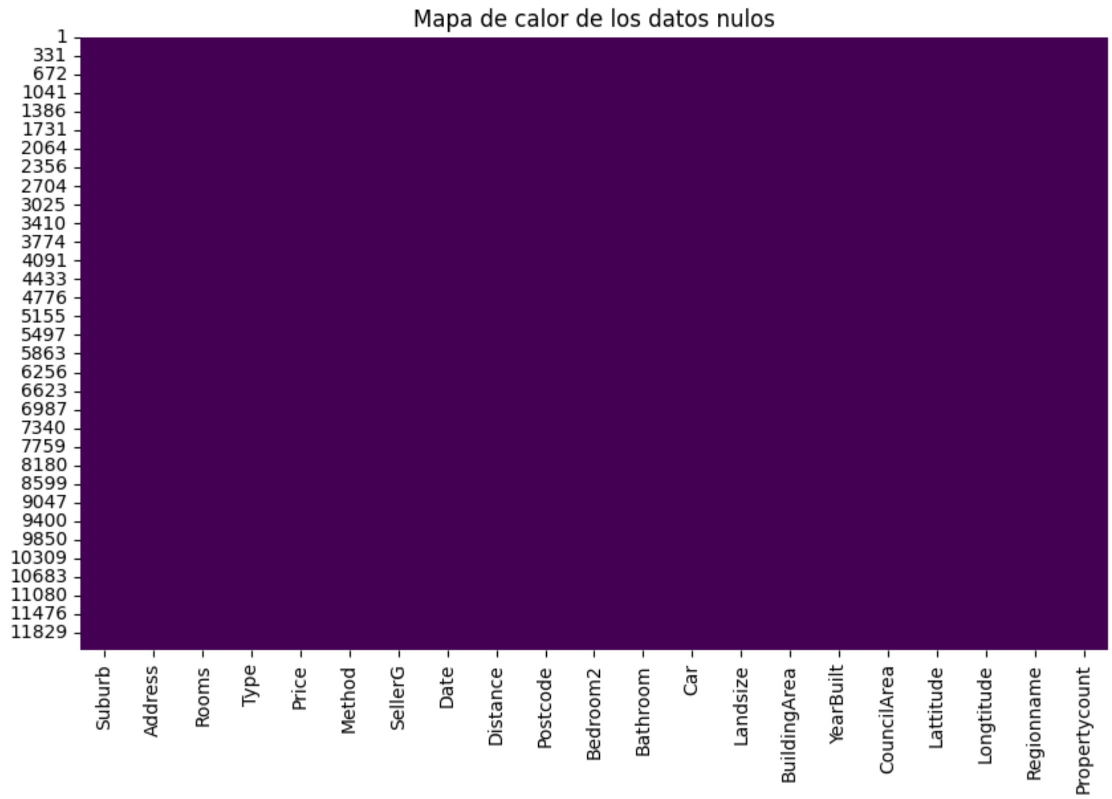
Con este grafico podemos observar en que columnas existen mayor cantidad de datos nulos, lo que nos puede ayudar al momento de que limpiemos la data para entrenar al modelo

[ ]:
```
melbourne_data.columns
```

[ ]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

[ ]:
```
melbourne_data = melbourne_data.dropna(axis=0)
```

[ ]:
```
plt.figure(figsize=(10, 6))  # Ajusta el tamaño de la figura
sns.heatmap(melbourne_data.isnull(), cmap='viridis', cbar=False)
plt.title('Mapa de calor de los datos nulos')
plt.show()
```



Limpiamos los datos nulos que existían en nuestro dataset

[ ]:
```
y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude',␣
↪'Longtitude']
X = melbourne_data[melbourne_features]
X.describe()
```

[ ]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

Escogemos los features con los cuales entrenaremos nuestro modelo.

0.1.1 Train Split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,␣
↪test_size=0.25)
```

0.1.2 Elección de Modelo

En este caso se utilizó un GridSearch para encontrar para comparar varios modelos con varios parámetros con el finde evaluarlos con el dataset y encontrar el que brinda mejor rendimiento. Para esto se compararon los siguientes modelos: - Regresión Lineal - Árbol de Decisión - Random Forest - Support Vector Machines

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn import tree
from sklearn.svm import LinearSVR
from sklearn.preprocessing import FunctionTransformer
from sklearn.ensemble import RandomForestRegressor

pipeline = Pipeline([
    ('regr', LinearRegression())  # Default model, will be overridden by␣
↪GridSearch
])

search_space = [
    {'regr': [LinearRegression()], 'regr__fit_intercept': [True, False]},
    {'regr': [tree.DecisionTreeRegressor()],
    'regr__max_depth': [None, 10, 20]},

    {'regr': [RandomForestRegressor()],
    'regr__n_estimators': [10, 50, 100],
    'regr__max_depth': [None, 10, 20],
    'regr__max_features': ['auto', 'sqrt', 'log2', None, 0.2, 0.5]},

    {'regr': [LinearSVR()],
    'regr__epsilon': [0.01, 0.1, 1, 10, 100]}
]

gs = GridSearchCV(pipeline, param_grid = search_space, scoring =␣
↪'neg_mean_squared_error', cv = 5)
gs.fit(X_train, y_train)
```

```
c:\Python312\Lib\site-packages\sklearn\model_selection\_validation.py:540:
FitFailedWarning:
45 fits failed out of a total of 320.
```

```
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
45 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Python312\Lib\site-packages\sklearn\model_selection\_validation.py",
line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Python312\Lib\site-packages\sklearn\base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Python312\Lib\site-packages\sklearn\pipeline.py", line 473, in fit
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "c:\Python312\Lib\site-packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "c:\Python312\Lib\site-packages\sklearn\base.py", line 666, in
_validate_params
    validate_parameter_constraints(
  File "c:\Python312\Lib\site-packages\sklearn\utils\_param_validation.py", line
95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestRegressor must be an int in the range [1, inf), a float
in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
c:\Python312\Lib\site-packages\sklearn\model_selection\_search.py:1102:
UserWarning: One or more of the test scores are non-finite: [-2.63645131e+11
-2.71940960e+11 -1.86980923e+11 -1.56915786e+11
 -1.83516465e+11             nan             nan             nan
 -1.10271121e+11 -9.71854290e+10 -9.68563635e+10 -1.03981652e+11
 -9.87730781e+10 -9.78414880e+10 -1.03807060e+11 -9.68233659e+10
 -9.44102216e+10 -1.16676006e+11 -1.03567380e+11 -1.01876483e+11
 -1.08577062e+11 -9.72828617e+10 -9.68469380e+10             nan
             nan             nan -1.20180788e+11 -1.14742034e+11
 -1.13320692e+11 -1.18367553e+11 -1.13452271e+11 -1.13321374e+11
 -1.09309178e+11 -1.04706986e+11 -1.04837444e+11 -1.30807001e+11
 -1.24729677e+11 -1.21986191e+11 -1.19618133e+11 -1.15867907e+11
 -1.12596427e+11             nan             nan             nan
 -1.05995603e+11 -9.63035394e+10 -9.66227418e+10 -1.06708449e+11
 -9.80739847e+10 -9.73163107e+10 -1.03555941e+11 -9.74941946e+10
 -9.76832232e+10 -1.14593861e+11 -1.04225766e+11 -1.01262825e+11
 -1.10525504e+11 -9.73605509e+10 -9.72291900e+10 -4.69852556e+11
```

```
 -4.68253962e+11 -4.69176087e+11 -4.69432661e+11 -4.68342953e+11]
  warnings.warn(
```

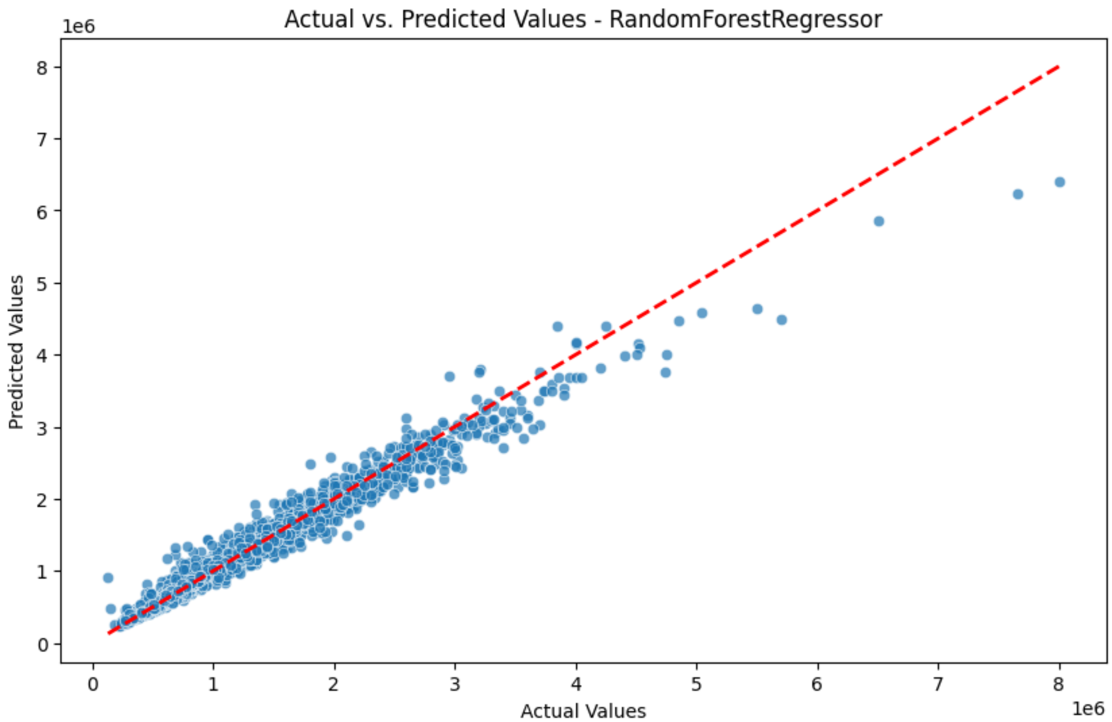
[ ]:
```
GridSearchCV(cv=5, estimator=Pipeline(steps=[('regr', LinearRegression())]),
             param_grid=[{'regr': [LinearRegression()],
                          'regr__fit_intercept': [True, False]},
                         {'regr': [DecisionTreeRegressor()],
                          'regr__max_depth': [None, 10, 20]},
                         {'regr': [RandomForestRegressor()],
                          'regr__max_depth': [None, 10, 20],
                          'regr__max_features': ['auto', 'sqrt', 'log2', None,
                                                 0.2, 0.5],
                          'regr__n_estimators': [10, 50, 100]},
                         {'regr': [LinearSVR()],
                          'regr__epsilon': [0.01, 0.1, 1, 10, 100]}],
             scoring='neg_mean_squared_error')
```

[ ]:
```python
best_pipeline = gs.best_estimator_
best_regression_model = best_pipeline.named_steps['regr']
best_model_hyperparameters = best_regression_model.get_params()

print("> Best Regresion Model:", best_regression_model)
print("> Best Hyper-parameters:", best_model_hyperparameters)
```

```
> Best Regresion Model: RandomForestRegressor(max_features=None)
> Best Hyper-parameters: {'bootstrap': True, 'ccp_alpha': 0.0, 'criterion':
'squared_error', 'max_depth': None, 'max_features': None, 'max_leaf_nodes':
None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1,
'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None,
'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None,
'verbose': 0, 'warm_start': False}
```

En este caso el modelo con mejor rendimiento fue el RandomForest, además se logró encontrar los hiperparámetros más adecuados con el mismo gridSearch

[ ]:
```python
model = best_regression_model
y_pred = model.predict(X_train)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_train, y=y_pred, alpha=0.7)
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], '--r',␣
 ↪linewidth=2)  # Line of perfect fit
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values - RandomForestRegressor')
plt.show()
```




```
[ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Calcular las métricas
mse = mean_squared_error(y_train, y_pred)
mae = mean_absolute_error(y_train, y_pred)
r2 = r2_score(y_train, y_pred)

# Importancia de las variables
feature_importances = model.feature_importances_

featuresArray = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
# Mostrar los resultados
print(f"Error Cuadrático Medio (MSE): {mse}")
print(f"Error Absoluto Medio (MAE): {mae}")
print(f"R² (Coeficiente de Determinación): {r2}")

print("\nImportancia de las Variables:")
for i, importance in enumerate(feature_importances):
    print(f"{featuresArray[i]}: {importance}")
```

```
Error Cuadrático Medio (MSE): 13257277967.425127
Error Absoluto Medio (MAE): 69361.45243731234
R² (Coeficiente de Determinación): 0.9688784502677918
```

```
Importancia de las Variables:
Rooms: 0.2739266406476306
Bathroom: 0.05575731536004612
Landsize: 0.16477917881511578
Lattitude: 0.26150031760293596
Longtitude: 0.24403654757427154
```

Estos resultados nos indican que nuestros valores mas importantes son la cantidad de cuartos y la latitud y longitud de la propiedad. El siguiente feature más importante es el tamaño de la propiedad el feature menos importante para nuestro modelo es la cantidad de baños.