California House Prices
Regresión Lineal

Jorge Fernandez
David Elvira

El objetivo de el siguiente informe es detallar los pasos seguidos y las decisiones tomadas a la hora de hacer el trabajo de regresión lineal propuesto.

Empezamos cargando el dataset y viendo que variables tenemos.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200

```
data.dtypes
```

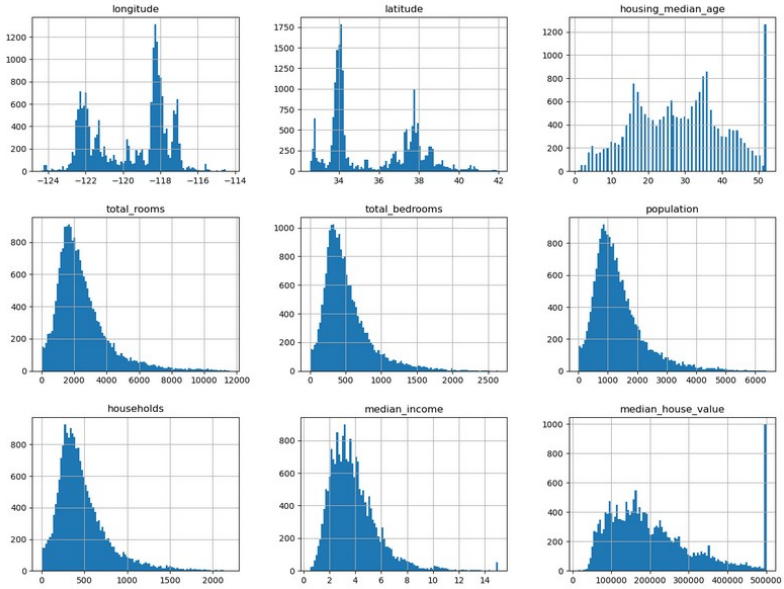
```
longitude             float64
latitude              float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
ocean_proximity        object
median_house_value      int64
dtype: object
```

Tenemos variables geográficas como la latitud y longitud. El resto de variables pertenecen a los hogares de california, pero no están individualizados, es decir, cada observación pertenece a un edificio, no a un único hogar. Tenemos variables como la mediana de los años que llevan construidos los hogares en el bloque, el numero total de habitaciones y de dormitorios en el bloque, la población del bloque, el número de hogares en el bloque, la mediana de los ingresos de los residentes en el bloque. También tenemos una variable categórica que nos indica lo proximo que se encuentra el bloque al oceano. Por último tenemos la variable objetivo que será la que intentaremos predecir, que es el precio medio de un hogar en un bloque.

Veamos almo más en detalle estas variables. Empecemos con las variables numéricas:

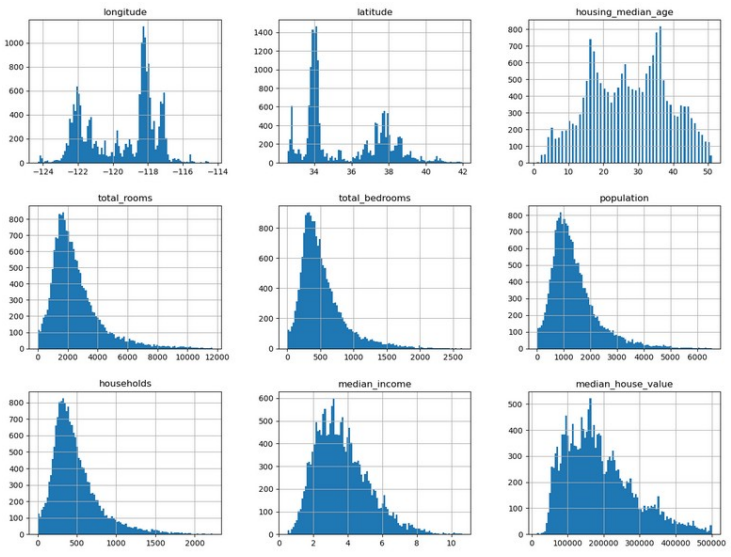
	count	mean	std	min	25%	50%	75%	max
longitude	20640.0	-119.569704	2.003532	-124.3500	-121.8000	-118.4900	-118.01000	-114.3100
latitude	20640.0	35.631861	2.135952	32.5400	33.9300	34.2600	37.71000	41.9500
housing_median_age	20640.0	28.639486	12.585558	1.0000	18.0000	29.0000	37.00000	52.0000
total_rooms	20640.0	2635.763081	2181.615252	2.0000	1447.7500	2127.0000	3148.00000	39320.0000
total_bedrooms	20433.0	537.870553	421.385070	1.0000	296.0000	435.0000	647.00000	6445.0000
population	20640.0	1425.476744	1132.462122	3.0000	787.0000	1166.0000	1725.00000	35682.0000
households	20640.0	499.539680	382.329753	1.0000	280.0000	409.0000	605.00000	6082.0000
median_income	20640.0	3.870671	1.899822	0.4999	2.5634	3.5348	4.74325	15.0001
median_house_value	20640.0	206855.816909	115395.615874	14999.0000	119600.0000	179700.0000	264725.00000	500001.0000

En principio no parece haber cosas muy raras en cuanto a los máximos o mínimos de los datos. Lo único que quizás destaca es que los ingresos medianos por bloque son muy pequeños, esto se debe a que la escala está en decenas de miles de dolares. Más tarde se solucionará, normalizando las variables. Podemos aplicar también coeficientes como el de asimetria o el de curtosis para ver mejor la distribución de las variables. En este caso, vamos a verlo de forma gráfica.



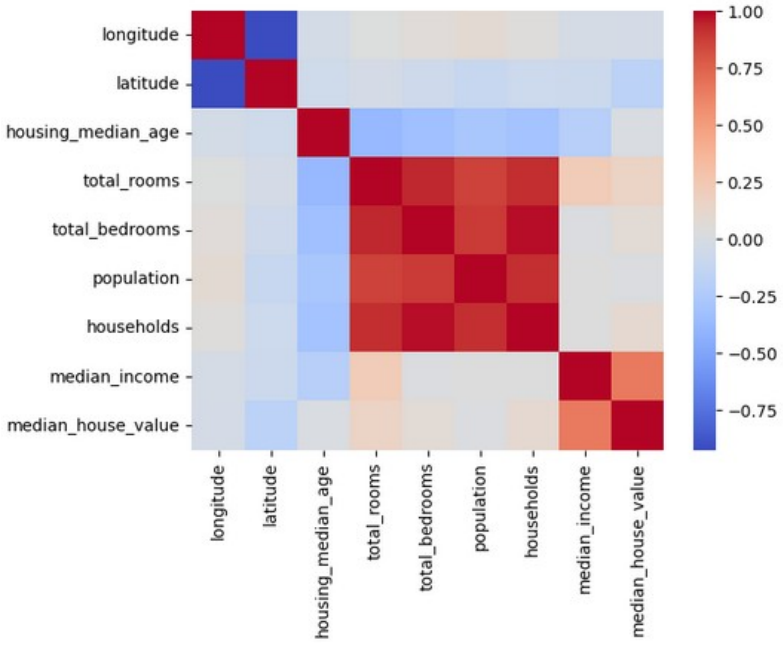
De esta visualización podemos fijarnos en 2 cosas.
La primera es que (sin contar la latitud y longitud) casi todas las variables parecen seguir una distribución similar. Se distribuyen de forma asimétrica, parecido a una F de Snedecor.
La segunda cosa que podemos ver es que tanto para la variable objetivo como para la edad mediana de los hogares hay una gran cantidad de valores en el máximo, más de lo habitual. Esto se puede deber a dos cosas. En primer lugar, es posible que haya un máximo en el que se han agrupado todas las observaciones que tenian más de ese número. Otra posibilidad es que esto sean datos faltantes que están codificados de esta forma debido a como se recogieron los datos. Si observamos cuantos datos corresponden, en este caso son 952 observaciones con el valor máximo.
En este caso, voy a considerar que son valores faltantes o valores que no están bien tomados. De estar completamente seguros, podríamos utilizar estas observaciones como conjunto de datos test, pero como no estamos seguros, si hacemos esto los datos podrían estar muy sesgados y sería erroneo. Por tanto, en este caso, vamos a eliminar estas observaciones ya que tenemos suficientes datos como para permitirnos elimirarlas y trabajar con las que parecen estar distribuidas de una forma “normal”.

```
data = data[data['median_house_value'] != 500001.0000]
data = data[data['housing_median_age'] != 52]
data.hist(bins=100, figsize=(16, 12))
```



De esta forma eliminamos estos valores extremos y nos quedamos con una distribución de datos más amortiguada.

Vemos a ver ahora la correlación entre las variables independientes y la variable objetivo.

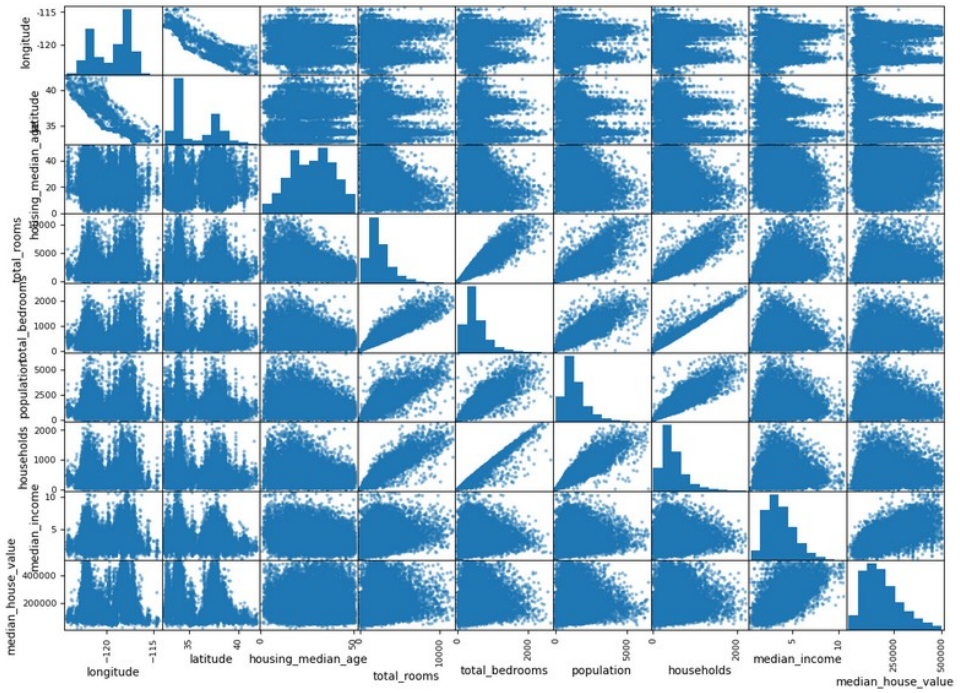


Podemos ver, que casi todas las variables carecen de correlación con la variable objetivo. Las únicas variables con correlación son median_income y latitud. La correlación en este caso es muy importante, ya que vamos a realizar una regresión lineal para predecir la variable objetivo, es decir un modelo lineal. Los modelos lineales dependen de que haya relaciones lineales entre las variables para predecir bien, y la correlación es el indicativo de que existe esta relación. Por tanto, de este mapa de calor podemos deducir ya que las variable que van a tener mayor importancia para predecir la variable objetivo son las 2 mencionadas anteriormente. Podemos ver también que existen correlaciones entre algunas de las variables independientes. Esto nos indica que existen varias variables que aumentan de forma lineal cuando aumentan las otras, esto no indica necesariamente causalidad. Estas variables son el total de habitaciones, dormitorios, la poblacion y los hogares.

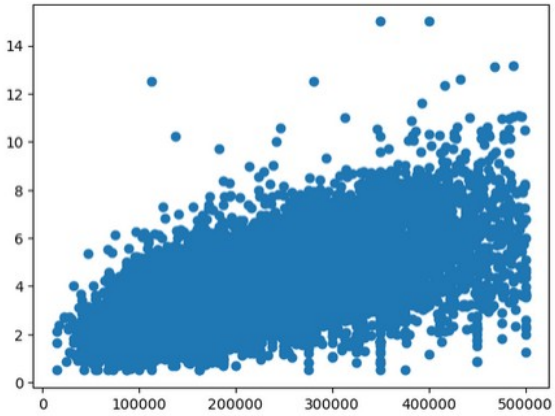
Algo que podemos hacer con estas variables es relativizarlas con respecto a la población o al número de hogares en el bloque. De esta forma, probablemente podamos prescindir de algunas de estas variables y simplificar el modelo y reducir “ruido”. También podríamos realizar un análisis de componentes principales para agrupar estas variables. En este caso por simplicidad, unicamente vamos a crear algunas variables nuevas relativas sin eliminar ninguna de las existentes.

Viendo que parece que solo hay una variable con una correlación relevanto, vamos a verlo de forma más gráfica mostrando un scatter plot de todas las variables.

```
pd.plotting.scatter_matrix(data, figsize=(14,10))
plt.show()
```



Si vemos más en detalle la variable de median_income:



Podemos ver como efectivamente parece aumentar median_income cuando aumenta median_house_value y viceversa.

Vamos a observar ahora si existen valores nulos en el conjunto de datos.

```
data.isna().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      193
population            0
households            0
median_income         0
ocean_proximity       0
median_house_value    0
dtype: int64
```

Podemos ver que la única variable con valores nulos es total_bedrooms. Al ser una variable muy correlacionada con otras, podriamos eliminarla sin perder mucha información. En este caso, como la cantidad de datos faltantes es muy pequeña, vamos simplemente a eliminar estas observaciones del conjunto de datos. Hay que destacar que para datos que nos lleguen de fuera y queramos predecir, no podemos eliminar la observación, es por esto, que para datos que tengamos que predecir que tengan valores nulos, se insertará la mediana de todos los valores.

Por último, vamos a explorar la variable categórica y ver que valores puede tener.

```
funcionesmineria.analizar_variables_categoricas(data)
```

```
{'ocean_proximity':                    n         %
 ocean_proximity
 <1H OCEAN        8331  0.448579
 INLAND           6344  0.341589
 NEAR OCEAN       2328  0.125350
 NEAR BAY         1567  0.084374
 ISLAND              2  0.000108}
```

Podemos ver que puede tener 5 valores diferentes. Cabe destacar que hay algunas categorías que están muy poco representadas. Transformaremos esta variable más adelante para que esté mejor distribuida.

Transformación de datos.

Vamos por tanto a realizar la transformación de los datos para ayudar a crear un mejor modelo. En primer lugar vamos a fijarnos en si existen valores atípicos.

```
num_vars = data.select_dtypes(['int', 'float'])
[funcionesmineria.atipicosAmissing(data[x])[1] / len(data[x]) for x in num_vars]
```

```
[0.0,
 0.0,
 0.0,
 0.009799698470816283,
 0.0,
 0.007861296575489986,
 0.007753607581305191,
 0.0009153564505707517,
 0.0]
```

El resultado de la función anterior, nos muestra el porcentaje de valores atípicos de cada variable. Como vemos, es muy pequeño y simplemente los convertiremos a valores nulos y los eliminamos junto a los valores nulos vistos anteriormente.

Antes de seguir con el proceso, vamos a dividir el conjunto de datos para evitar hacer sobreajuste en la predicción.

```
y = data['median_house_value']
X = data.drop('median_house_value', axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=31254)
```

Ahora que hemos dividido la muestra, vamos a transformar los datos.
En primer lugar, vamos a crear algunas nuevas variables relativas que hemos mencionado anteriormente y añadirlas a las ya existentes. En este caso, vamos a crear una para habitaciones por casa, dormitorios por casa y población entre habitaciones totales. Es posible que existan variables más optimas pero por simplificar nos quedaremos solo con estas.

```
rooms_house = X_train['total_rooms'] / X_train['households']
bedrooms_house = X_train['total_bedrooms'] / X_train['households']
population_rooms = X_train['population'] / X_train['total_rooms']

df_rooms_house = pd.DataFrame(rooms_house, columns=['rooms_house'])
df_bedrooms_house = pd.DataFrame(bedrooms_house, columns=['bedrooms_house'])
df_population_rooms = pd.DataFrame(population_rooms, columns=['population_rooms'])
X_train = pd.concat([X_train, df_rooms_house, df_bedrooms_house, df_population_rooms], axis=1)
```

A continuación vamos a escalar los datos para estandarizarlos.

```
num_vars = X.select_dtypes(['int', 'float']).columns
transformed_data = pd.concat([X, funcionesmineria.Transf_Auto(X[num_vars], y)], axis = 1)
```

```
num_vars = X_train.select_dtypes(['int', 'float']).columns
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X_train[num_vars])
scaled_data
```

Para la variable categórica, en primer lugar, vamos a ver como se comporta ante la variable objetivo.

```
funcionesmineria.analizar_variables_categoricas(data)
```

```
{'ocean_proximity':                    n         %
 ocean_proximity
 <1H OCEAN        8331  0.448579
 INLAND           6344  0.341589
 NEAR OCEAN       2328  0.125350
 NEAR BAY         1567  0.084374
 ISLAND              2  0.000108}
```

```
ocean_prox = data.groupby('ocean_proximity')['median_house_value'].mean()
ocean_prox
```

```
ocean_proximity
<1H OCEAN     223167.038771
INLAND        123044.829288
ISLAND        368750.000000
NEAR BAY      224600.382897
NEAR OCEAN    227192.955326
Name: median_house_value, dtype: float64
```

Como vemos, y hemos dicho antes, hay categorías como ISLAND que apenas tienen representación. Si vemos como se comportan las categorías ante la variable objetivo, hay 3 que se comportan practicamente de la misma forma, que son: NEAR OCEAN, NEAR BAY y <1H OCEAN. Vemos también que 2 de estas están muy poco representadas, con alrededor del 10% de los valores cada una. Podemos agrupar estas 3 variables en una, ya que se comportan practicamente de la misma forma y asi aumentamos la representación de categorias poco representadas. Asi mismo, tambien añadiremos ISLAND a esta categoria ya que es la que más se le acerca y tiene unicamente 2 observaciones.

```
data['ocean_proximity'] = data['ocean_proximity'].replace({"<1H OCEAN" : "NEAR_OCEAN", "INLAND" : "INLAND", "NEAR OCE
```

```
funcionesmineria.analizar_variables_categoricas(data)
```

```
{'ocean_proximity':                      n         %
 ocean_proximity
 NEAR_OCEAN       12228  0.658411
 INLAND            6344  0.341589}
```

```
ocean_prox = data.groupby('ocean_proximity')['median_house_value'].mean()
ocean_prox
```

```
ocean_proximity
INLAND        123044.829288
NEAR_OCEAN    224140.996075
Name: median_house_value, dtype: float64
```

Vemos que tras juntar las categorias mencionadas, quedan agrupadas en 2 categorias, ambas con suficiente representación.
Por último, aplicamos One Hot Encoding para transformar esta variable categórica en 2 columnas numéricas binarias.

```
encoder = OneHotEncoder()
encoded_array = encoder.fit_transform(X_train[['ocean_proximity']]).toarray()
encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(['ocean_proximity']))
encoded_df
```

Por último, queda aplicar estas transformaciones para nuevos datos que reciba el modelo para predecir, o en este caso los datos de test.

```
def transform_test(X_test):

    X_test = X_test.reset_index()
    # Create new variables
    rooms_house = X_test['total_rooms'] / X_test['households']
    bedrooms_house = X_test['total_bedrooms'] / X_test['households']
    population_rooms = X_test['population'] / X_test['total_rooms']

    df_rooms_house = pd.DataFrame(rooms_house, columns=['rooms_house'])
    df_bedrooms_house = pd.DataFrame(bedrooms_house, columns=['bedrooms_house'])
    df_population_rooms = pd.DataFrame(population_rooms, columns=['population_rooms'])

    X_test = pd.concat([X_test, df_rooms_house, df_bedrooms_house, df_population_rooms], axis=1)

    # Scale numerical data
    num_vars = X_test.select_dtypes(['int', 'float']).columns
    scaled_data = scaler.fit_transform(X_test[num_vars])
    num_df = pd.DataFrame(scaled_data, columns=["s_" + x for x in num_vars])

    # replace scaled data
    X_test = pd.concat([num_df, pd.DataFrame(X_test['ocean_proximity'])], axis=1)

    # OneHotEncoding
    encoded_array = encoder.fit_transform(X_test[['ocean_proximity']]).toarray()
    encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(['ocean_proximity']))
    X_test = pd.concat([X_test, encoded_df], axis=1)
    X_test = X_test.drop(['ocean_proximity', 's_index'], axis=1)
    return X_test
X_test = transform_test(X_test)
```

Vamos a realizar el último paso que es el de la predicción. Para ello, primero tenemos que elegir una métrica a evaluar. En este caso, nuestro objetivo es realizar predicciones lo más cercanas posible a los valores reales. Es por esto que utilizaremos como métricas el error absoluto medio y el error medio al cuadrado para evaluar el modelo. También veremos que valores toman el R2. Este último explica el porcentaje de varianza explicada por el modelo. Existe también un R2 ajustado que penaliza el numero de variables del modelo, pero como nuestro objetivo no es interpretar mejor los resultados, no nos fijaremos en el.

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
prediction = lin_reg.predict(X_test)
prediction
```

```
mse = mean_absolute_error(y_test, prediction)
me = mean_squared_error(y_test, prediction, squared=False)
r2 = r2_score(y_test, prediction)
print(mse)
print(me)
print(r2)
```

```
42864.60885659402
57398.25669293369
0.6169638291945768
```

El resultado de las predicciones es un error medio de 42864 dolares, un error medio al cuadrado de 57398 y un R2 de 0.616.