Tercer Parcial

Nicolás Gutiérrez Ramírez

Prof. John Jairo Corredor

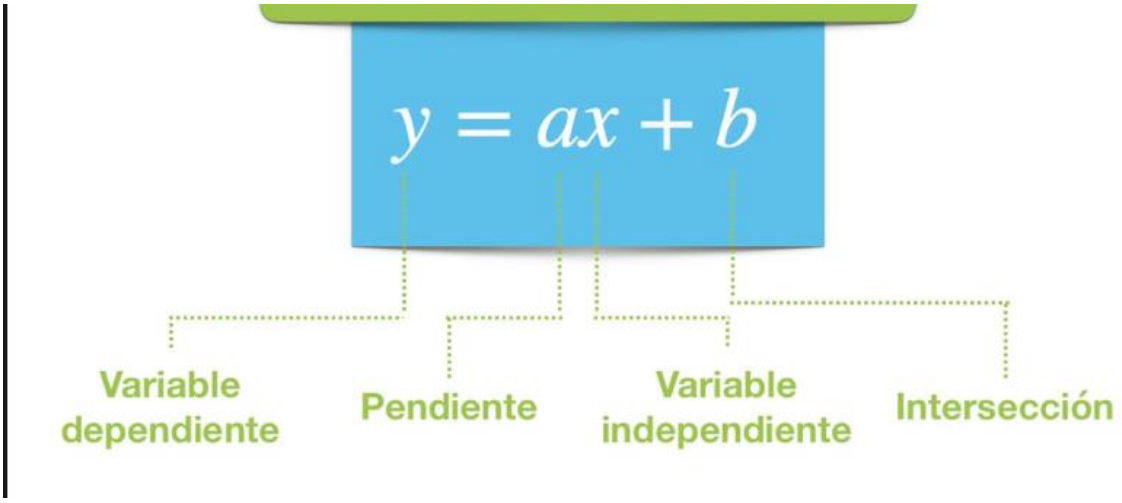
Métricas de Rendimiento
Escuela de ciencias exactas e ingeniería
Universidad Sergio Arboleda
23 de mayo de 2022

Introducción

El aprendizaje automatizado en los últimos años es un tema muy relevante para las ciencias computacionales, y no es de extrañar, ya que con todas las funciones que puede cumplir en diferentes ámbitos tanto laborales como comerciales. Pero en que consiste el aprendizaje automatizado.

El aprendizaje automatizado o machine Learning es una rama de la inteligencia artificial que permite a las maquinas aprender sin estar estrictamente programadas, esta funciona gracias a la gran capacidad de identificación de patrones dentro de los datos para hacer predicciones. Es decir que tomando como base datos pasados es posible tener una aproximación de lo que puede suceder. Esta tecnología esta presente en muchas aplicaciones actualmente como lo son google, Facebook, netflix, etc.

En el trabajo actual hicimos uso de la regresión línea el cual es un método estadístico que trata de modelar la relación entre una variable dependiente y una o más variables independientes mediante el ajuste de una ecuación lineal. Existen dos tipos de regresiones la simple y la múltiple, la regresión lineal simple consiste en que solo se estudia con una variable independiente y la regresión lineal múltiple cuando se trabaja con más de 1 variable independiente. El Principal objetivcco de esta regresión es hallar la educación de la recta la cual es:



Contexto

El archivo csv entregado para realizar el análisis respectivo es llamado californiaHousingHPC el cual contiene información acerca de las viviendas de la zona dividida por bloques o cuadras. Incluyendo lo que es la ubicación por medio de coordenadas indicadas por latitud y longitud, también cuenta con la información

de cuantas personas viven en cada bloque, que antigüedad tiene el predio, los ingresos medios de las personas, el total de cuartos, etc.

Este conjunto de datos posee bastante información por lo cual es un archivo perfecto para demostrar el como el aprendizaje automatizado puede predecir ciertos datos, basándose en los que ya están registrados.

Los IDE utilizados para realizar este ejercicio fueron c++ y Python, donde ambos cumplirán el mismo objetivo, pero se guiarán por métodos o procesos distintos. En el caso de c++ es un proceso más modelado a mano y haciendo todos los pasos que requiere la regresión manualmente en el programa creando sus respectivos métodos y clases para diferenciar las funciones que cumplen. En el caso de Python este incluía una librería la cual posee todas las funciones y procesos de la regresión necesarios y ya están definidos.

Métricas

El modelo de regresión línea hace parte del aprendizaje automatizado, pero, ¿cómo tenemos la seguridad de que las predicciones resultantes si sean acertadas? La regresión lineal tiene medidas las cuales determinan que tan exacto es el modelo realizado las dos utilizadas para este trabajo fueron las MSE y el coeficiente R2.

El MSE o error cuadrático medio calcula el error entre el valor predicho y los valores resultantes. Es decir que al momento de realizar las predicciones esta medirá que tan lejos o cerca estuvo el valor predicho al valor real, mientras más grande sea el numero resultante más grande es el error.

$$\mathrm{MSE} = \frac{1}{N} \times \sum_{t=1}^{N} (y_t - f_t)^2 = \frac{\mathrm{SSE}}{N}$$

yt es el resultado actual en el tiempo t.

ft es el pronóstico de valor en el tiempo t.

SSE es la suma del error cuadrático.

EL coeficiente de determinación R2 describe la proporción de la varianza de la variable predicha por el modelo, este refleja que tan acertado esta el ajuste del modelo a la variable que pretende igualar, el coeficiente se mide de 1 a 0, mientras mas cerca este de 1 significa que el ajuste del modelo fue mas acertado. Y de igual manera cuanto mas cerca este de 0, menos acertado estuvo el modelo.

$$R^2 = \frac{\sum_{t=1}^{T} (\hat{Y}_t - \bar{Y})^2}{\sum_{t=1}^{T} (Y_t - \bar{Y})^2}$$

La ecuación anterior presenta como se calcula el coeficiente R2 teniendo como numerador el resultante de las variables predichas y se divide entre los valores reales del modelo.

EDA

```
[465] dataFrame.head()
```

	Unnamed: 0	median_house_value	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	0	66900.0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1493.6
1	1	80100.0	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1820.0
2	2	85700.0	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1650.9
3	3	73400.0	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3191.7
4	4	65500.0	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1925.0

Identificacion de las variables:

longitude : Ubicación Longitud

latitude : Ubicación Latitud

housing_median_age : Edad promedio de la casa

total_rooms: Número total de habitaciones en una cuadra

total_bedrooms: Número total de habitaciones dentro de una cuadra

population: Número total de personas que residen en una cuadra

households: Número total de hogares, grupo de personas que residen dentro de una unidad doméstica, para una cuadra

median_income: Mediana de los ingresos de los hogares de una cuadra de viviendas (medidos en decenas de miles de dólares)

median_house_value: Valor medio de la vivienda de los hogares de una manzana (medido en dólares estadounidenses)

Lo primero que observe fue que la variable a predecir escogida por el profesor que era median_house_value estaba ubicada en la segunda columna además tenemos una columna que es un índice por lo cual tampoco no será útil. Por cual se toma la decisión de reorganizar las columnas del dataset y no tener en cuenta la primera columna, esto se realizo con la siguiente línea de código:

```
dataFrame = dataFrame.reindex(columns=['longitude','latitude','housing_median_age','total_rooms','total_bedrooms','population','households','median_income','median_house_value'])
```

Gracias a esta se organiza de mejor manera el dataset y al momento de utilizar el csv en el programa de c++ será más sencilla la asignación de la variable dependiente o a predecir.

```
dataFrame['median_income'] = dataFrame['median_income'] * 1000
dataFrame
```

	median_house_value	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	66900.0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1493.6
1	80100.0	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1820.0
2	85700.0	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1650.9
3	73400.0	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3191.7
4	65500.0	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1925.0

Otra acción que tome fue pasar la columna de median_income que estaba medida en decenas de dólares a ser medida en dólares para que el dato quede en el mismo formato que la variable a predecir ya que median_house_value está medido en dólares y no en decenas.

```
#EDA del dataset
#1. informacion completa del dataset
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   median_house_value  17000 non-null  float64
 1   longitude           17000 non-null  float64
 2   latitude            16999 non-null  float64
 3   housing_median_age  16998 non-null  float64
 4   total_rooms         16998 non-null  float64
 5   total_bedrooms      16999 non-null  float64
 6   population          17000 non-null  float64
 7   households          17000 non-null  float64
 8   median_income       17000 non-null  float64
dtypes: float64(9)
memory usage: 1.2 MB
```

Como podemos observar en la imagen superior hay ciertas columnas que tiene datos vacíos, esto podría generar errores al momento de los cálculos ya que se calcularía con datos null y esto nos podría perjudicar, por lo cual se toma la decisión de eliminar los registros vacíos, como todos los datos son de tipo float64 no se modifican.

```
dataFrame = dataFrame.dropna()
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16996 entries, 0 to 16995
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          16996 non-null  int64  
 1   median_house_value  16996 non-null  float64
 2   longitude           16996 non-null  float64
 3   latitude            16996 non-null  float64
 4   housing_median_age  16996 non-null  float64
 5   total_rooms         16996 non-null  float64
 6   total_bedrooms      16996 non-null  float64
 7   population          16996 non-null  float64
 8   households          16996 non-null  float64
 9   median_income       16996 non-null  float64
dtypes: float64(9), int64(1)
memory usage: 1.4 MB
```

Como podemos observar el modelo ya no posee datos vacíos entonces ya al realizar las modificaciones pertinentes, se toma la decisión de exportar el csv modificado para usarlo en el modelo de c++

```
[471] dataFrame.to_csv('californiaHousingHPC2.csv', encoding='utf-8')
```

La anterior línea de código funciona para generar un csv con las modificaciones hechas

```
dataFrame.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1493600.0	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1820000.0	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1650900.0	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3191700.0	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1925000.0	65500.0

El data set resúltate quedaría asi.

```
# Se presenta resumen estadistico de los datos
dataFrame.describe()
```

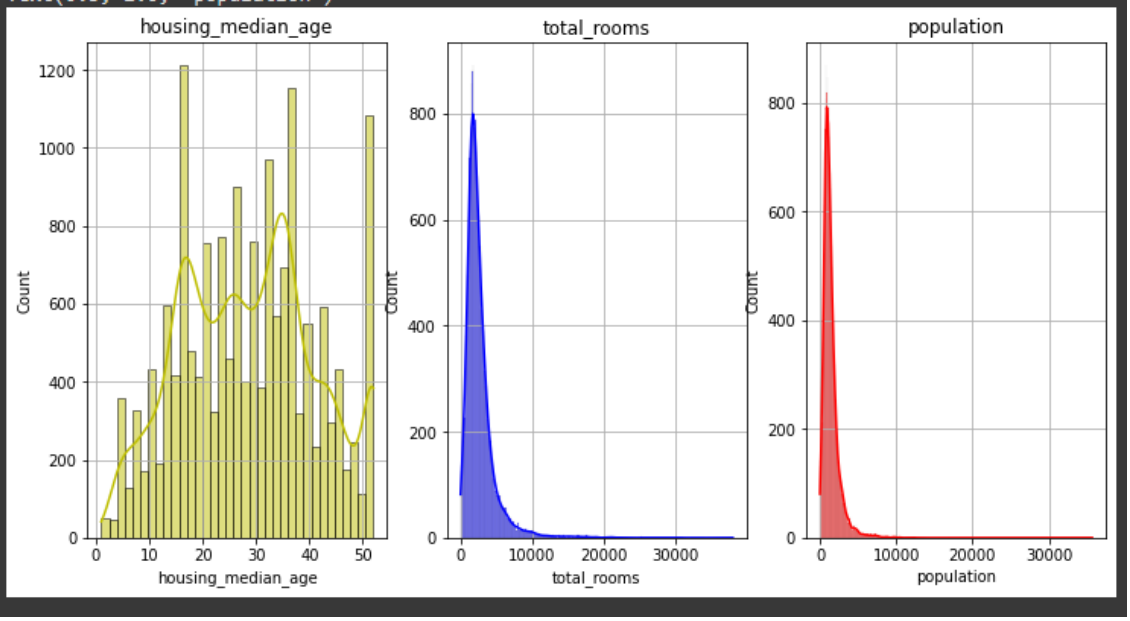
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	16996.000000	16996.000000	16996.000000	16996.000000	16996.000000	16996.000000	16996.000000	16996.000000	16996.000000
mean	-119.563018	35.625783	28.592257	2643.507472	539.360202	1429.522299	501.195811	3883.777736	207334.844081
std	2.004522	2.137265	12.586231	2179.265844	421.299896	1147.685508	384.411962	1908.227528	115976.220164
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	499.900000	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2566.700000	119475.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3545.050000	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4767.000000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15000.100000	500001.000000

El resumen de los datos nos muestra cuales datos son los máximos y mínimos por columna, su promedio, su raíz cuadrada, la cantidad total, etc.

```
# Se grafica la distribucion de la variable
fig, axes=plt.subplots(1,3,figsize=(12,6))
axes[0].grid()
axes[1].grid()
axes[2].grid()

sns.histplot(dataFrame['housing_median_age'], ax=axes[0], kde=True, color="y")
axes[0].set_title('housing_median_age')
sns.histplot(dataFrame['total_rooms'], ax=axes[1], kde=True, color="b")
axes[1].set_title('total_rooms')
sns.histplot(dataFrame['population'], ax=axes[2], kde=True, color="r")
axes[2].set_title('population')
```

```
Text(0.5, 1.0, 'population')
```



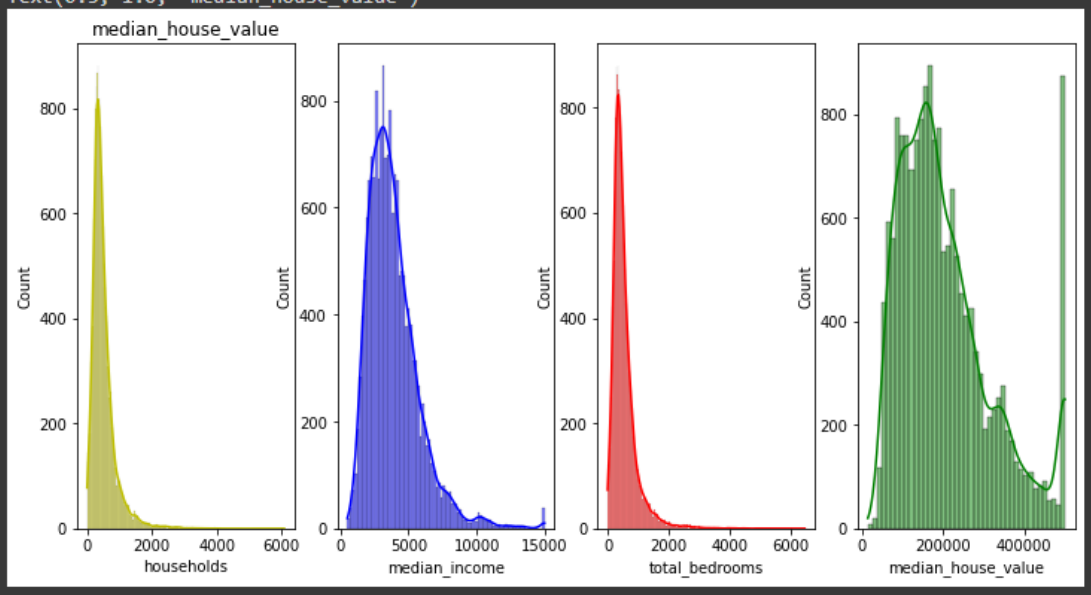
Aquí graficamos a distribución de algunas variables independientes, podemos observar que en la gráfica de house_median_age podemos concluir que la mayoría de casas tienen entre 30 a 40 años, en la gráfica de total_room concluimos que la

mayoría de bloques tiene entre 0 y 10000 cuartos. Y por ultimo la population se concluye que en la mayoría de bloques viven entre 0 y 50000 personas.

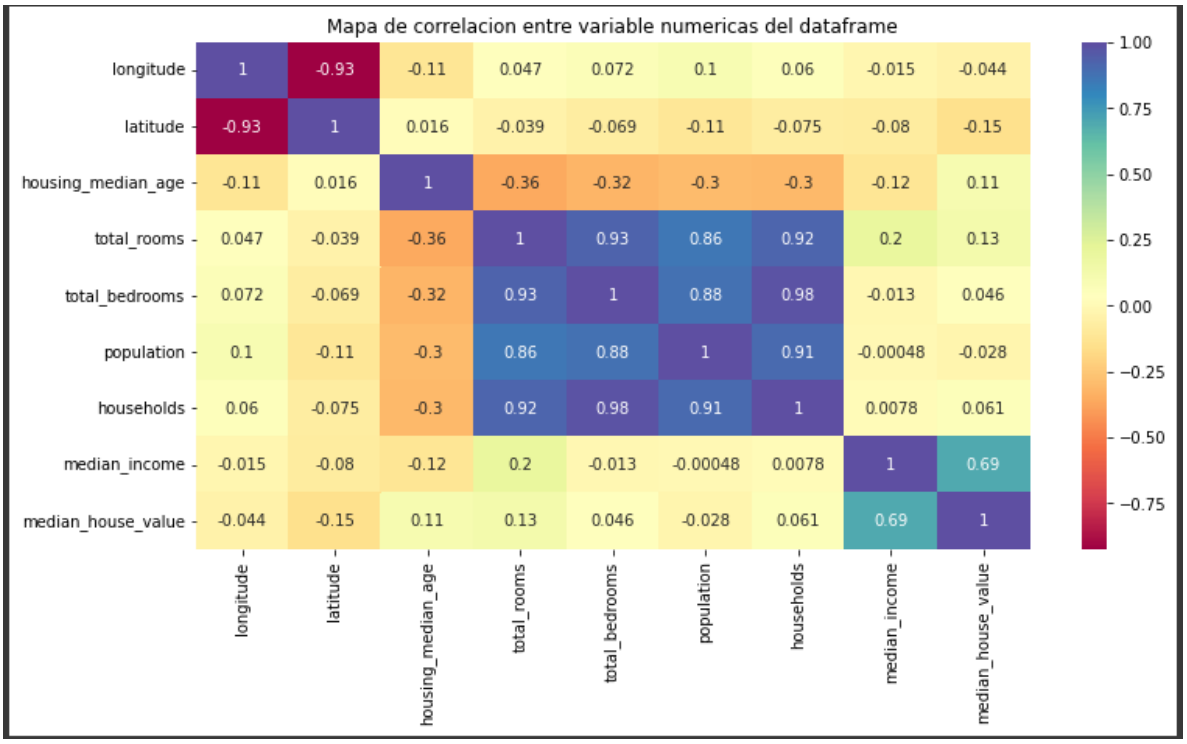
```
fig, axes=plt.subplots(1,4,figsize=(12,6))

sns.histplot(dataFrame['households'], ax=axes[0], kde=True, color="y")
axes[0].set_title('households')
sns.histplot(dataFrame['median_income'], ax=axes[1], kde=True, color="b")
axes[0].set_title('median_income')
sns.histplot(dataFrame['total_bedrooms'], ax=axes[2], kde=True, color="r")
axes[0].set_title('total_bedrooms')
sns.histplot(dataFrame['median_house_value'], kde=True, color="g")
axes[0].set_title('median_house_value')
```

```
Text(0.5, 1.0, 'median_house_value')
```



Igual como en las anteriores graficas las variables tienen una tendencia hacia la izquierda commo se puede observar en la imagen de arriba. Por lo cual los media_income o ingresos medios de las casa está entre 0 y 5000, también tenemos la información de que la gran mayoría de bloques posee entre 1000 cuartos.



Esta imagen representa el coeficiente de relación que tienen las variables entre sí, en este caso nuestra variable dependiente es median_house_value, por lo cual es en la principal que debemos fijarnos, al observar el grafico podemos concluir que la mayor correlación que tiene nuestra variable es de 0.69 y es con la variable median_income. Aunque en general no posee una alta correlación con las demás variables estas pueden ayudar a las predicciones que va a realizar el modelo.

Observando más allá de nuestra variable podemos vero como hay una fuerte correlación entre total_rooms, total_bedrooms, population y housenols, quizás esto tenga que ver mayormente porque estas son totalmente dependientes de las personas que residen allí, por lo cual pueden tener altos índices de variación.

```
[586] # A continuacion se separa las variables dependientes de las varialbes independientes
      X = dataFrame.drop(['median_house_value'], axis=1)
      y = dataFrame['median_house_value']
```

Separamos las variables independientes y dependiente para posteriormente realizar el entrenamiento

```
[587] # Se hacen los grupos para entrenamiento y prueba
      X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,shuffle=False)

[588] # A continuacion se hace un pipeline para entrenar el modelo usando la normalizacion z (similar a lo elaborado en c++)
      pipe = Pipeline([('scaler', StandardScaler()),('LR', LinearRegression())])
      pipe.fit(X_train, y_train)

      Pipeline(steps=[('scaler', StandardScaler()), ('LR', LinearRegression())])
```

Se dividen en dos grupos unos de prueba y otro de entrenamiento, los datos de train van a tomar el 80% de los datos y el otro 20% es para los datos de test. Y después se entrena el modelo con pipe.fit, se hace por medio de este método ya que en la mayoría de casos al trabajar con dataset puede que algunos datos no sean compatibles es por esto que se recomienda estandarizar los datos o normalizar para que asi los cálculos no generen inconsistencias en el modelo.

```
[589] # Variable de prediccion de prueba por sklearn
      y_hat_test_sk = pipe.predict(X_test)

[590] # Variable de prediccion de entrenamiento por sklearn
      y_hat_train_sk = pipe.predict(X_train)
```

Se definen dos variables que almacenan los datos de la predicciones del modelo, una con los datos de prueba(test) y otro con los datos de entrenamiento(train)

```
pd.DataFrame({'Actual': y_test, 'Predicted': y_hat_test_sk})
```

	Actual	Predicted
13600	204100.0	192585.927354
13601	126700.0	199574.979331
13602	265900.0	292477.438541
13603	267400.0	277947.286809
13604	351300.0	352347.055703
...	...	...
16995	111400.0	153656.523820
16996	79000.0	147957.448201
16997	103600.0	99380.815531
16998	85800.0	62351.105546
16999	94600.0	177667.034112

3400 rows × 2 columns

En esta pequeña tabla podemos observar una pequeña comparación de los datos predichos y los datos reales, a simple vista se concluye que el MSE en algunos casos es más grande que otro, pero el modelo depende de la puntuación R2 para determinar si es aconsejable o no implementarlo.

```
pipe.score(X_test, y_test)
```

```
0.6556193950554592
```

```
[593] pipe.score(X_train, y_train)
```

```
0.6202294643662776
```

Como resultado en la prueba de scores tenemos como resultado con los datos de test un 65% de precisión en los datos y en los datos de train tenemos una precision del 62% si bien, está por encima del 50% no están lo suficientemente altos para

asegurar al 100% que es modelo que puede servir por lo cual no lo recomendaría. Para mí sería recomendable si supera el 75%.

```
regr = LinearRegression()
regr.fit(X_train, y_train)
print('Coeficientes: \n', regr.coef_)
print('Corte eje Y: \n', regr.intercept_)
print("Error cuadratico medio: %.2f" % mean_squared_error(y_train, y_hat_train_sk))
```

```
Coeficientes:
 [-4.01598995e+04 -4.09906075e+04  9.81707925e+02 -8.38072087e+00
  1.17868267e+02 -3.56089213e+01  3.36467028e+01  3.99079812e+01]
Corte eje Y:
 -3326308.026347383
Error cuadratico medio: 4655835338.14
```

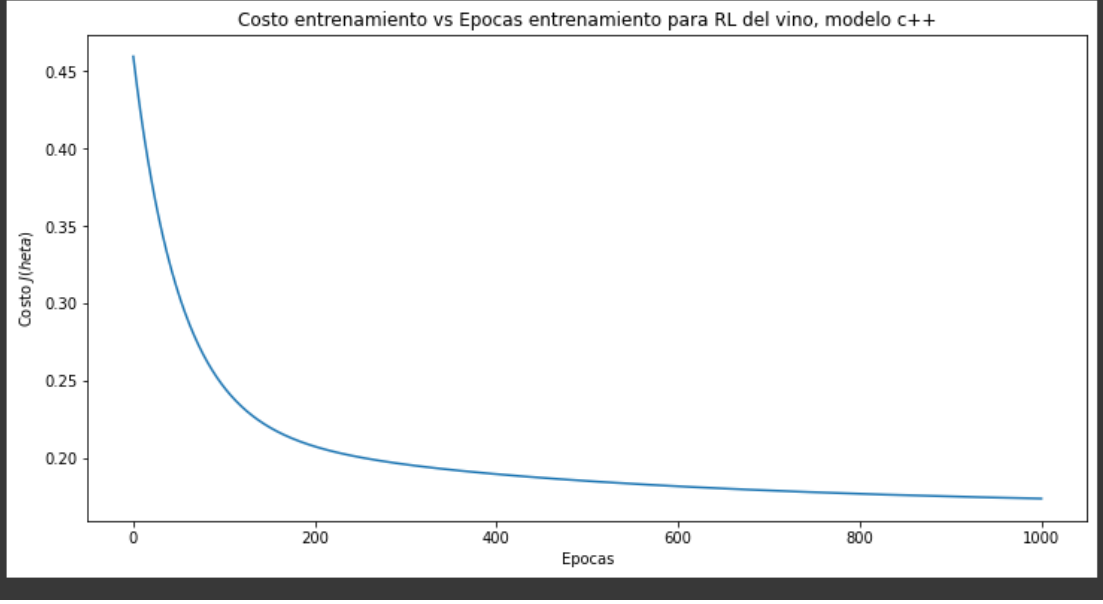
Al final del modelo, tenemos los datos necesarios para generar una ecuación de regresión línea, teniendo como resultado que b es igual -3326308.02, además tenemos como resultado un error cuadrático medio bastante alto, por lo cual algunas predicciones están demasiado alejadas del resultado real. Además tenemos diferentes coeficientes ya que al realizar la regresión con 8 variables independientes se tiene 8 pendientes distintas, una para cada variable.

```
pd.DataFrame({'C ++': prom, 'SkLearn': dataFrame.mean()})
```

	C ++	SkLearn
longitude	-119.556	-119.563018
latitude	35.6237	35.625783
housing_median_age	28.5906	28.592257
total_rooms	2643.35	2643.507472
total_bedrooms	539.328	539.360202
population	1429.44	1429.522299
households	501.166	501.195811
median_income	3883.55	3.883778
median_house_value	207323	207334.844081

En esta imagen podemos observar la comparación del promedio de las columnas en ambos modelos tanto Python como c++.

```
# A continuacion se grafica la funcion de costo de nuestro modelo artesanal C++
fig, axes = plt.subplots(figsize=(12,6))
axes.plot(costo)
axes.set_title('Costo entrenamiento vs Epocas entrenamiento para RL del vino, modelo c++')
axes.set_xlabel('Epocas')
axes.set_ylabel('Costo $J(\theta)$')
plt.show()
```

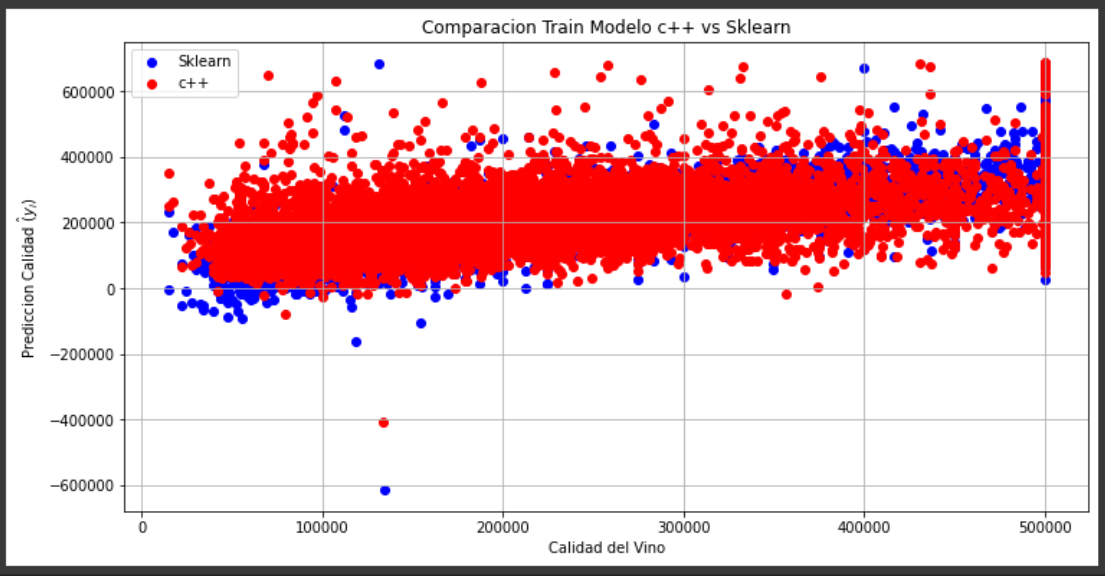


La grafica del vector costo realizada en el modelo de c++, como podemos observar sus datos disminuyen cada vez que avanza el modelo.

```
[635] y_hat = pd.read_csv("/content/y_train_hat.txt", header = None)
      y_hat = y_hat.drop(13597, axis=0)
      y_hat = y_hat.drop(13596, axis=0)

[636] # Se compara modelo en c++ vs python SKlearn
      fig, axes = plt.subplots(figsize=(12,6))

      plt.scatter(y_train, y_hat_train_sk, c='b', label='Sklearn')
      plt.scatter(y_train, y_hat, c='r', label='c++')
      axes.set_title('Comparacion Train Modelo c++ vs Sklearn')
      axes.set_xlabel('Calidad del Vino')
      axes.set_ylabel('Prediccion Calidad $\hat(y_i)$')
      plt.legend(loc=2)
      plt.grid()
      plt.show()
```



Al momento de realizar la comparación de ambos modelos tenemos como conclusión que ambos modelos tiene un comportamiento similar, y la mayoría de predicciones las lograron hallar en ambos casos, por lo cual se puede decir que no existe mucha diferencia al momento de realizar las operaciones.

Conclusiones

En conclusión, el proyecto es bastante instructivo además de que muestra la capacidad que tienen las máquinas de aprender basándose en información pasada. Además, tenemos que el modelo en ambos IDE tiene un resultado similar incluso en la métrica de rendimiento R2 tuvieron porcentajes similares, por lo cual ambos procesos resultan tonalmente funcionales, pero en mi opinión es más efectivo realizarlo con Python ya que posee todas las herramientas ya definidas, en vez en c++ toca hacer el proceso casi todo a mano.

Respecto al modelo resultante se concluye que no es un modelo muy preciso ya que el MSE es bastante elevado y la distancia que tiene algunas predicciones con los datos reales no muestra para nada una buena aproximación. Igualmente, esto se puede concluir viendo la gráfica, ya que al ver tantos puntos distribuidos en el eje y es normal que el error sea así de grande.

Bibliografia

Regresión lineal con python (cienciadedatos.net)