



Instituto Tecnológico y de Estudios Superiores de Monterrey



Campus: Querétaro

“Momento de Retroalimentación: Módulo 2 Implementación de una técnica de aprendizaje máquina sin el uso de un framework. (Portafolio Implementación)”

Materia:

“Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)”

Alumno:

Juan Pablo Cabrera Quiroga - A01661090

Fecha de entrega:
24 de Agosto del 2024

Leyenda:

Como alumnos del Tec y “Apegándose al código de Ética de los estudiantes del Tecnológico de Monterrey, nos comprometemos a que nuestra actuación en esta actividad de evaluación esté regida por la integridad académica. En congruencia con el mismo realizaremos esta actividad de forma honesta y personal, para reflejar a través de ella nuestros conocimientos y competencias”

ÍNDICE

ÍNDICE	1
Introducción.	2
ETL.	2
¿Qué es ETL y de qué nos sirve?	2
Descripción del Dataset utilizado.	3
Extracción de datos.	3
Transformación de datos.	3
Manejo de valores faltantes.	4
Imputación de datos.	4
1. Imputación por moda:	4
2. Imputación por media:	5
Conversión de datos categóricos a numéricos.	5
Escalamiento de datos.	6
Carga de datos.	6
Siguiendo pasos.	7
Modelo de regresión lineal.	8
Introducción.	8
Establecer función de hipótesis.	8
Función de costo o pérdida.	10
Gradiente descendiente.	11
Separación de datos de entrenamiento y prueba.	13
Resultados y evaluación del modelo.	14
Función de MSE.	14
Coeficiente de Determinación (R ²).	14
Evaluación del Bias.	15
Diagnóstico del modelo.	15
Visualización de Resultados	15
Datos de Entrenamiento: Reales vs. Predicción	15
Este gráfico muestra cómo las predicciones del modelo se alinean con los valores reales en el conjunto de entrenamiento.	15
Datos de Prueba: Reales vs. Predicción	16
Datos Reales vs. Predicción	16
Conclusión.	17
Siguiendo etapa.	17

Introducción.

En este ensayo detallaré el proceso completo que he llevado a cabo para desarrollar un modelo de regresión lineal, el cuál me permite realizar predicciones. Explicaré desde la extracción y transformación de datos (ETL) hasta la implementación y evaluación del modelo. A través de este documento, presentaré tanto los aspectos teóricos como prácticos, destacando las técnicas y herramientas vistas en clase y que son utilizadas en cada etapa.

Es importante destacar que el proceso que he llevado a cabo no es un proceso lineal. Como nos fuimos dando cuenta en las clases, durante la construcción de un modelo, es muy común que sea necesario retroceder a etapas previas de la construcción, ajustar parámetros, o incluso avanzar a etapas siguientes para poder validar decisiones tomadas anteriormente, todo esto con el fin de **analizar e interpretar los datos**. A lo largo de este ensayo, no solo se presentará los pasos exitosos, sino también los errores cometidos y el flujo de pensamiento detrás de cada decisión. Todo con el fin de ofrecer una visión completa del proceso de aprendizaje. Este ensayo busca precisamente demostrar que, más allá de seguir un conjunto rígido de pasos, lo esencial es comprender el fundamento de lo que se está haciendo y las bases detrás de cada decisión.

ETL.

¿Qué es ETL y de qué nos sirve?

ETL es un proceso fundamental en la gestión y análisis de datos. Significa **Extraer, Transformar y Cargar** (en inglés es Extract, Transform y Load, por eso ETL).

Extraer implica tomar datos de diversas fuentes, como bases de datos, archivos, o APIs.

En la fase de **Transformación**, estos datos se limpian, se ajustan, y se formatean para que sean consistentes y útiles para el análisis. Esto incluye tareas como corregir errores, llenar valores faltantes, o convertir datos categóricos en numéricos. Esto es algo que se verá de forma más detallada en las siguientes secciones.

Finalmente, en la etapa de **Carga**, los datos transformados se guardan y luego se suben o transfieren al modelo donde serán utilizados. Se pueden mandar como JSON, CSV, guardar en un servidor, etc.

El proceso ETL nos sirve porque permite que los datos que inicialmente pueden estar desorganizados, incompletos o que traigan información que no nos parezca relevante, se conviertan en información precisa y bien estructurada. Esto es crucial para poder realizar análisis confiables, construir nuestro modelo y tomar decisiones informadas basadas en datos. Es muy importante entender que sin un proceso ETL adecuado, es muy probable que los análisis resulten inexactos o que se pierda información valiosa en el camino.

Descripción del Dataset utilizado.

El dataset que utilicé para este proyecto fue extraído de la página de Kaggle. El dataset que elegí se centra en las especificaciones de diferentes automóviles de 1985, incluyendo características como el tipo de combustible, marca, el número de cilindros, la potencia del motor y el precio del vehículo. Después de haber corroborado con mi profesor, concordamos en que este conjunto de datos me sirve para trabajar en el proyecto, ya que incluye varios atributos (columnas) categóricos como numéricos. Además de tener una cantidad de instancias (filas) suficiente como para poder manipularlo y realizar análisis.

Extracción de datos.

Previamente mencioné que el proceso de ETL comienza con la extracción de datos. En mi caso, utilicé un archivo .data (que contiene el dataset) que descargué de la página de Kaggle. Como ya expliqué antes, este dataset contenía las especificaciones de automóviles de 1985. Durante este proceso de extracción, me centré principalmente en escoger un dataset que fuera relevante para mí y lo que quería lograr con el modelo, que es tener predicciones de datos.

De la página de Kaggle, además de poder descargar el set de datos, también pude descargar un archivo .names. Este es un archivo esencial para entender el significado de cada atributo y tratar de asegurar un correcto análisis de los datos. La información contenida en el archivo .names me permitió identificar y etiquetar adecuadamente las columnas al momento de cargar los datos, identificar también para que se había usado el anteriormente el dataset y sobre todo una parte muy importante al final de este archivo que me ayudaba a identificar los valores nulos en el dataset.

Ahora bien, centrándome en lo que está en mi código, utilicé la biblioteca de **Pandas** para cargar el dataset en un DataFrame, con los nombres de los features. Esto me permitió que en los siguientes pasos pudiera trabajar de manera eficiente con los datos.

Transformación de datos.

Esta etapa para mí es la más importante de todo el proceso de ETL. Ya que en esta etapa es en la que se realiza la limpieza, estructuración y preparación de los datos para su posterior análisis. En mi proyecto, implementé varias técnicas de preprocesamiento para asegurar que los datos estuvieran en el mejor estado posible para el modelado predictivo. Las presento a continuación.

Manejo de valores faltantes.

La primera técnica que apliqué tiene que ver con el cálculo de valores faltantes. En el dataset original, los valores faltantes estaban representados por el símbolo "?", esto nos lo menciona el archivo .name que mencioné anteriormente y después pude sacar la cantidad de datos faltantes gracias a una función que hice con la librería de Pandas. Ahora bien, para facilitar el procesamiento posterior, reemplacé estos símbolos por NaN utilizando la función replace de pandas:

```
Python  
df.fillna(df.mean(), inplace=True)
```

Con esto lo que logré fue que esos datos faltantes se convirtieran en Nan y entonces sí decidir qué hacer con esos Nan. Hasta este momento del proceso, no me había encontrado con ningún problema y pude continuar con la siguiente fase.

Imputación de datos.

La segunda técnica que apliqué tiene que ver con cómo manejo esos valores faltantes. A esto se le conoce como "Imputación de datos". En este proyecto, apliqué dos métodos diferentes de imputación:

1. Imputación por moda:

Para todas las columnas excepto 'Price', utilicé la imputación por moda. La moda es el valor más frecuente en cada columna, y decidí usar este método porque es muy útil cuando se trabaja con datos categóricos o cuando hay un valor que se repite con mayor frecuencia en una columna. En mi caso, muchas de las columnas contenían datos categóricos, como el número de puertas o el tipo de combustible, y la imputación por moda asegura que los valores faltantes se reemplacen de una manera que respete la distribución original de los datos. Este es el código que utilicé para lograrlo:

```
Python  
mode_imputer = other_columns.mode().iloc[0]  
other_columns_imputed =  
other_columns.fillna(mode_imputer)
```

2. Imputación por media:

Para la columna 'Price', utilicé la imputación por media. Esto significa que reemplacé los valores faltantes con el promedio de los valores existentes en esa columna. Decidí hacer esto porque 'Price' es un dato numérico, y la media es una buena forma de representar el valor típico en este tipo de datos.

Usando la media, pude completar los valores faltantes sin alterar demasiado la distribución general de los precios en el dataset.

De esta forma es como lo realicé en el código:

```
Python  
price_imputed = price_column.fillna(price_column.mean())
```

Este paso de realizar la imputación de los datos, ya se me complicó bastante. Lo que sucedió es que en mis columnas con datos faltantes, tenía yo datos que no eran numéricos (strings). Entonces realmente no podía obtener valores. Esto me lleva a mi tercera técnica de preprocesamiento.

Conversión de datos categóricos a numéricos.

En mi caso, por el dataset que elegí, este contenía muchos valores que no eran numéricos. Como lo vimos en clase, la mejor forma para solucionar esto es hacer diccionarios de datos.

La forma en como yo lo hice fue que asigné un valor numérico a cada categoría dentro de las columnas correspondientes. Por ejemplo, en la columna "Num-of-doors", los valores "two" y "four" fueron transformados en 2 y 4, respectivamente.

Esta técnica de conversión no sólo me ayudó a poder resolver el problema de los datos faltantes en columnas categóricas (y poder aplicar la imputación de datos), sino que también permitió que todos los datos estuvieran en el mismo formato (numérico). Este paso fue fundamental para asegurar que mi modelo pudiera procesar y analizar correctamente todas las variables involucradas.

Ahora bien, después de aplicar las técnicas de imputación y conversión de datos, me di cuenta de que los datos que guardé en mi data frame eran demasiado variables. Por ejemplo, en una columna tenía valores de peso que no bajaban de los 1,500 o RPM que llegaba hasta los 5,000. En contraste, otra columna, como la del número de puertas, solo contenía valores de 2 o 4.

Esta gran diferencia en los rangos de valores entre las columnas me empezó a generar problemas en mi modelo de regresión lineal, ya que algunas características podrían influir más en el modelo solo porque sus números son mayores. Como lo vimos en clase, los números más pequeños tienden a ser más irrelevantes, a desaparecer.

Escalamiento de datos.

Es por esta razón que utilicé otra técnica que se conoce como escalamiento de datos. Específicamente, implementé el “StandardScaler” de la biblioteca sklearn. Esta técnica transforma los datos de manera que cada característica tenga una media de 0 y una desviación estándar de 1. Esta es la única ocasión en la que hago uso de una biblioteca en toda mi implementación y esto lo hago porque sklearn está diseñado para manejar este tipo de transformaciones con precisión y sin riesgo de pérdida de datos o errores de cálculo.

La verdad es que al realizar el escalamiento manualmente, existe la posibilidad de introducir errores que podrían alterar la integridad de los datos o afectar negativamente el rendimiento del modelo. Es por eso que en un paso tan importante decidí hacerlo con un framework que me asegurara que el proceso no iba a salir mal.

Después de aplicar el escalamiento, todas mis características ahora están en una escala comparable, lo que debería mejorar significativamente el rendimiento y la estabilidad de mi modelo de regresión lineal. Llevándome al último paso que realicé de este proceso.

Carga de datos.

Una vez completado todo el proceso de limpieza, transformación y escalamiento de los datos, el último paso de mi ETL fue cargar estos datos procesados en un formato que pudiera ser fácilmente utilizado para el modelado posterior.

Para esto, utilicé la función `to_csv` de pandas para guardar mis datos procesados en un nuevo archivo CSV. Elegí el formato CSV porque a mi gusto es el más fácil de manipular con las diferentes herramientas y lenguajes de programación. Además, al guardar los datos procesados, puedo evitar tener que repetir todo el proceso de ETL cada vez que quiera trabajar con estos datos.

Siguientes pasos.

Es importante decir que, aunque hemos completado los pasos principales de ETL, los datos aún experimentarán una transformación adicional **muy importante** antes de ser utilizados en el modelo. Con esto me refiero a que se realizará una división entre datos de entrenamiento y prueba.

Esta división es clave para poder evaluar que tan bien funciona el modelo con datos que no ha visto durante el entrenamiento. Esto nos ayuda a entender si el modelo puede hacer buenas predicciones en datos nuevos. Los detalles de esta división y cómo se implementa se explicarán en la sección del modelo de regresión lineal.

Modelo de regresión lineal.

Introducción.

La regresión lineal es un método estadístico que nos permite entender y predecir la relación entre una variable dependiente (la que queremos predecir) y una o más variables independientes (las que usamos para hacer la predicción).

Un ejemplo de la vida real para entender mejor cuándo usar regresión lineal, podría ser cuando se usa para predecir precios de casas basándose en características como el tamaño, la ubicación, y el número de habitaciones. Otro ejemplo es el uso en negocios para prever ventas futuras a partir de datos históricos.

Existen dos tipos principales de regresión lineal: la **regresión lineal simple**, que usa una sola variable independiente, y la **regresión lineal múltiple**, que usa dos o más variables independientes para hacer predicciones. En mi proyecto yo utilicé la regresión lineal múltiple, porque estoy trabajando con varias variables independientes (como el tipo de combustible, el número de cilindros, la potencia del motor, etc.) para predecir una variable dependiente, que es el precio del vehículo.

Me gustaría dar un poco más de contexto sobre lo que va a ser el modelo que estoy construyendo.

El tipo de modelo que estoy construyendo es de tipo “Aprendizaje supervisado”. Con esto me refiero a que vamos a comparar lo que el modelo predice con los valores reales que ya conocemos. Durante el proceso de entrenamiento, el modelo aprende ajustando sus parámetros para que sus predicciones se acerquen lo más posible a estos valores conocidos. Este enfoque es fundamental porque nos permite medir y mejorar la precisión del modelo antes de aplicarlo a nuevos datos.

Establecer función de hipótesis.

La hipótesis en la regresión lineal es una fórmula matemática que describe cómo las variables independientes se relacionan con la variable dependiente que queremos predecir. La fórmula general para la hipótesis en una regresión lineal múltiple es:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Esta fórmula parece muy rebuscada, pero realmente es algo muy sencillo. Esta es la explicación:

- $h(x)$ representa la predicción que queremos lograr con el modelo. En mi caso ejemplo, esta representa el precio del carro.
- θ_0 es el intercepto, es decir, el valor de $h(x)$ cuando todas las variables independientes son cero. En la clase nosotros lo estamos manejando como “Bias”.
- Por último, $\theta_1, \theta_2, \dots, \theta_n$ son los coeficientes que multiplican a cada una de las variables independientes x_1, x_2, \dots, x_n . Estos coeficientes indican cuánto influye cada variable independiente en la predicción.

Para entender esto de una mejor manera, voy a poner un ejemplo con mi proyecto. Es importante que previo a esto, hayan leído la parte de ETL. Sobre todo para entender las columnas (atributos) y filas (features).

Si estoy usando atributos como “el tipo de combustible”, “el número de cilindros”, y “la potencia del motor” para predecir el precio del vehículo, la hipótesis podría verse así:

$$h(\text{precio}) = \theta_0 + \theta_1(\text{tipo de combustible}) + \theta_2(\text{número de cilindros}) + \theta_3(\text{potencia del motor}) + \dots$$

En este caso, $\theta_1, \theta_2, \theta_3$ son los parámetros que el modelo ajusta durante el entrenamiento para que la predicción de $h(x)$ se acerque lo más posible al precio real del vehículo.

Visualmente, si fuera una regresión lineal simple con solo una variable independiente, la hipótesis se representaría como una línea recta en un gráfico. En la regresión lineal múltiple, estamos ajustando un plano o un hiperplano en un espacio de varias dimensiones, dependiendo de cuántas variables independientes estemos utilizando. Me gusta ejemplificar esto porque es una pregunta que he realizado múltiples veces en clase y el profesor Benjamin me ha ayudado a entender.

Ahora bien, en relación a cómo manejo esto en mi código: Definí una función de hipótesis que realiza este cálculo de una manera simple y directa. La función hypothesis que creé toma como entradas los parámetros (params), los valores de las variables independientes (sample), y el Bias (bias). Luego, calcula la predicción usando el producto punto entre los parámetros y las variables, sumándole el Bias al final. Este enfoque me pareció más claro que incluir el Bias dentro de la matriz de datos desde el principio.

```
Python
def hypothesis(params, sample, bias):
    return np.dot(params, sample) + bias
```

Como podemos observar, hago la misma fórmula que explique al inicio pero como una multiplicación de vectores.

Función de costo o pérdida.

La función de costo es lo siguiente que implementé en mi modelo. Esta función mide qué tan bien el modelo está realizando sus predicciones en comparación con los valores reales. (Recordemos que estamos trabajando con un modelo de aprendizaje supervisado).

Existen varios tipos de funciones de costo, cada una con sus propias características y aplicaciones específicas. Algunas de las más comunes incluyen:

- Error Cuadrático Medio (Mean Squared Error - MSE)
- Error Absoluto Medio (Mean Absolute Error - MAE)
- Error Cuadrático Logarítmico (Mean Squared Logarithmic Error - MSLE)
- Entropía Cruzada (Cross-Entropy)

Para el modelo de regresión lineal, he optado por utilizar el Error Cuadrático Medio (MSE) como función de costo. Esto se debe a que es la función que más hemos visto en clase, porque es la que más he practicado y me he dado cuenta de que es particularmente útil con modelos de regresión lineal.

El MSE se calcula de la siguiente manera:

$$MSE = (1/n) * \sum (y_i - \hat{y}_i)^2$$

Donde:

- n es el número total de muestras
- y_i es el valor real
- \hat{y}_i es el valor predicho por el modelo

En otras palabras, el MSE calcula el promedio de las diferencias al cuadrado entre los valores que predice el modelo y los valores reales. Al elevar esas diferencias al cuadrado, el MSE hace que los errores grandes tengan un mayor impacto en el resultado final, lo que es útil para asegurarse de que el modelo no se aleje demasiado de los valores correctos.

En mi implementación, codifiqué la función de costo MSE de la siguiente manera:

Python

```
def cost_function(params, bias, samples, y):  
    y_prima = np.dot(samples, params) + bias  
    mse = np.mean(np.square(y_prima - y)) / 2  
    return mse
```

Esta función toma los parámetros del modelo (params), el sesgo (bias), las muestras de entrada (samples) y los valores reales (y). Calcula las predicciones (y_prima) utilizando los parámetros actuales y luego calcula el MSE.

Gradiente descendiente.

Una vez que hemos definido nuestra función de costo, el siguiente paso es ajustar los parámetros del modelo para que haga las mejores predicciones posibles. Para hacer esto, necesitamos encontrar los valores de los parámetros que minimicen la función de costo, es decir, que hagan que el error entre las predicciones del modelo y los valores reales sea lo más pequeño posible. Aquí es donde entra el Gradiente Descendente.

El Gradiente Descendente es un algoritmo que nos ayuda a encontrar esos valores óptimos de los parámetros. Básicamente, lo que hace es empezar con valores iniciales para los parámetros y luego, en cada paso, ajustarlos un poco en la dirección que reduce el error (lo hace a través de las derivadas). Este proceso se repite muchas veces hasta que el modelo llega a un punto donde los cambios en los parámetros no mejoran mucho más el modelo. De esta forma, logramos que el modelo sea lo más preciso posible en sus predicciones.

La fórmula que estoy utilizando es la siguiente:

$$\theta_j = \theta_j - \frac{\alpha}{m} \sum_{i=1}^m [(h_{\theta}(x_i) - y_i)x_i]$$

Donde:

- θ_j : Es el parámetro que se está ajustando en cada paso del Gradiente Descendente.
- α : Es la tasa de aprendizaje, que controla el tamaño de los pasos que damos en la dirección del mínimo. Si es muy grande, podemos saltarnos el mínimo; si es muy pequeño, el algoritmo puede ser muy lento.
- m : Es el número total de muestras en el dataset. Este valor lo utilizamos para calcular el promedio de los gradientes.
- $h_{\theta}(x_i)$: Es la predicción del modelo para la i-ésima muestra, calculada usando la hipótesis.
- y_i : Es el valor real para la i-ésima muestra, que estamos tratando de predecir.
- x_i : Es el valor de la variable independiente para la i-ésima muestra.

Ahora pasemos esto a mi código:

```
Python
def gradient_descent(params, bias, samples, y, alpha, iterations):
    m = len(y)

    for _ in range(iterations):
        predictions = np.dot(samples.T, params) + bias
        errors = predictions - y

        # Calcular los gradientes
        gradient_params = (1/m) * np.dot(samples, errors)
        gradient_bias = (1/m) * np.sum(errors)

        # Actualizar los parámetros y el bias
        params -= alpha * gradient_params
        bias -= alpha * gradient_bias

    return params, bias
```

Este código es muy lineal y fácil de entender. Básicamente, lo que hacemos es aplicar paso a paso el algoritmo de Gradiente Descendente que mostré antes, integrándolo ahora con las funciones de MSE e Hipótesis que ya definimos:

1. Inicializamos los parámetros (incluyendo el bias) con ceros. (esto lo hago más abajo donde defino los parámetros del modelo).
2. Cálculo de las predicciones: Utilizando los parámetros actuales, calculamos las predicciones del modelo. Esto se hace con el producto punto entre las variables independientes y los parámetros, sumando el bias al final. (El bias en la primera iteración es 0 pero la magia viene que después ese bias va a ser modificado con cada iteración).
3. Luego, calculamos la diferencia entre las predicciones del modelo y los valores reales (es decir, el error).
4. Con base en el error, calculamos los gradientes de la función de costo con respecto a cada parámetro. Estos gradientes nos indican en qué dirección debemos ajustar los parámetros para reducir el error.
5. Después actualizamos los parámetros. Ajustamos los parámetros (incluyendo el bias) en la dirección opuesta al gradiente, es decir, hacia donde la función de costo disminuye. Esto nos permite acercarnos cada vez más a la solución óptima.

Ahora, de conceptos que son importantes para poder entender este código:

- α es la tasa de aprendizaje, que controla el tamaño de los pasos que damos en la dirección del gradiente.

- "iterations" es el número máximo de iteraciones que realizaremos. (En clase lo hemos visto como EPOCHS).

Los pasos 2-5 los repetimos hasta que el cambio en la función de costo sea muy pequeño o alcancemos un número máximo de iteraciones (EPOCHS, en este momento lo tengo en 3000).

Separación de datos de entrenamiento y prueba.

Como mencioné anteriormente en la última etapa de ETL, todavía me faltaba un paso más de la manipulación de datos. Este paso es la separación entre datos de entrenamiento y prueba. Esta división nos permite evaluar el rendimiento del modelo en datos que no ha visto durante el entrenamiento, proporcionando así una medida más realista de su capacidad de generalización.

Esto lo realicé de la siguiente manera:

```
Python
def train_test_split(X, y, test_size=0.2, random_state=None):
    if random_state is not None:
        np.random.seed(random_state)

    n_samples = len(X)
    n_test = int(n_samples * test_size)

    indices = np.random.permutation(n_samples)

    test_indices = indices[-n_test:]
    train_indices = indices[:-n_test]

    X_train = X[train_indices]
    X_test = X[test_indices]
    y_train = y[train_indices]
    y_test = y[test_indices]

    return X_train, X_test, y_train, y_test
```

Como se puede observar, lo que hago es dividir el total de mi dataset en un 80% la cantidad de datos para entrenar y el 20% de los datos restantes los utilizo para después comparar esos datos.

Resultados y evaluación del modelo.

Es momento de poner en práctica todos los conceptos que mencioné anteriormente. En mi caso, como lo mencioné anteriormente, voy a estar tratando de predecir el costo del carro basándose en sus características (diferentes atributos del dataset).

Estos fueron los resultados que obtuve al correr mi modelo de regresión lineal:

Función de MSE.

- **Entrenamiento:** El error final en el conjunto de entrenamiento (recordemos que esto abarca el 80% de mi dataset) fue de **0.0533**. Este valor representa que tan bien se ajusta la función a los datos de entrenamiento; entre menor sea, quiere decir que se ajusta de mejor manera.
- **Prueba:** El error final en el conjunto de prueba fue de **0.1379**. Este valor muestra el MSE del modelo al predecir sobre datos no vistos durante el entrenamiento (me refiero al 20% restante de los datos del dataset).

Al comparar los resultados de los dos, puedo inferir que mi modelo no está prediciendo con la misma precisión en los datos de prueba que en los de entrenamiento, lo que sugiere un posible overfitting. Pero no me gustaría decir nada hasta no ver el Bias y en ese momento dar una buena conclusión.

Coeficiente de Determinación (R^2).

En mi caso, la R^2 es una métrica que utilizo para evaluar qué tan bien mi modelo de regresión lineal predice el valor de la variable dependiente en función (precio) de las variables independientes (otras features).

Los valores que me dieron para mi R^2 fueron los siguientes:

- **Entrenamiento:** En el conjunto de entrenamiento fue de **0.8890**. Este valor es un valor que yo considero como bueno ya que nos indica que aproximadamente el 88.90% de la variabilidad en el costo del carro está explicada por el modelo en el conjunto de entrenamiento.
- **Prueba:** Por otro lado, en el conjunto de prueba el R^2 fue de **0.7587**. Esto significa que el modelo explica alrededor del 75.87% de la variación en el costo del carro cuando se utiliza en datos nuevos, no vistos durante el entrenamiento. Aunque sigue siendo un valor relativamente alto, es menor que en el conjunto de entrenamiento, lo que sugiere que el modelo podría estar perdiendo algo de precisión al generalizar a nuevos datos.

Evaluación del Bias.

Para el Bias, obtuve el siguiente resultado:

- Con un valor final de 0.0170, considero que mi valor de Bias es bajo. Recordemos que un Bias bajo indica que el modelo es capaz de analizar o captar bien los patrones en los datos de entrenamiento (algo que hemos visto en la función de MSE y R^2).

Con esta información, ahora sí me gustaría dar el diagnóstico de mi modelo.

Diagnóstico del modelo.

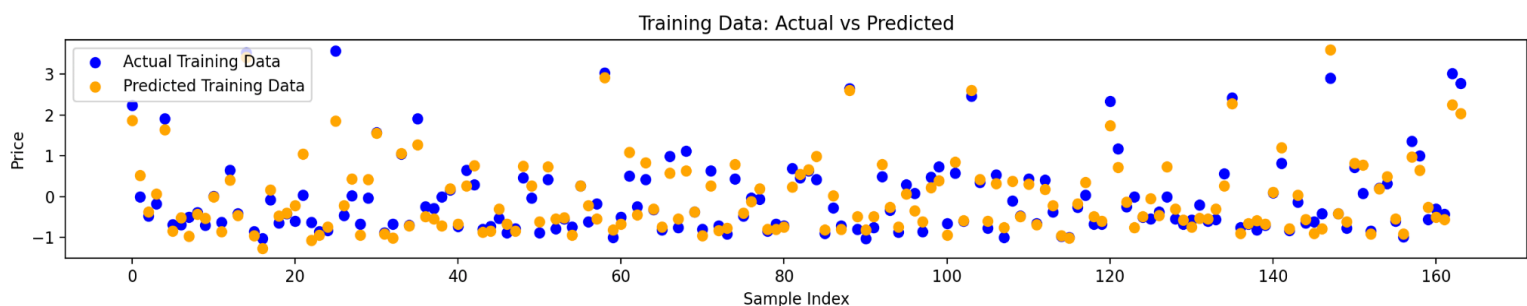
Tomando en cuenta todos los valores que mostré con anterioridad:

Tomando en cuenta lo que vimos en clase, mi modelo está presentando overfitting (un bias bajo y MSE con los datos de prueba alto). Esto significa que el modelo se ajusta muy bien a los datos de entrenamiento, pero tiene dificultades con datos nuevos no vistos durante el entrenamiento.

Visualización de Resultados

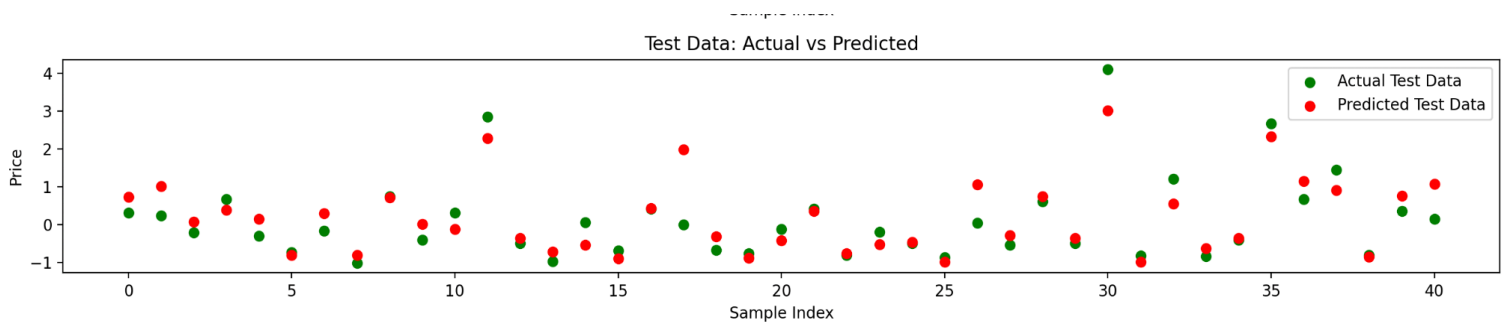
Para entender mejor cómo el modelo predice el costo de los automóviles, comparé los valores reales con los predichos tanto en el conjunto de entrenamiento como en el de prueba. Esto se ve de esta forma.

Datos de Entrenamiento: Reales vs. Predicción



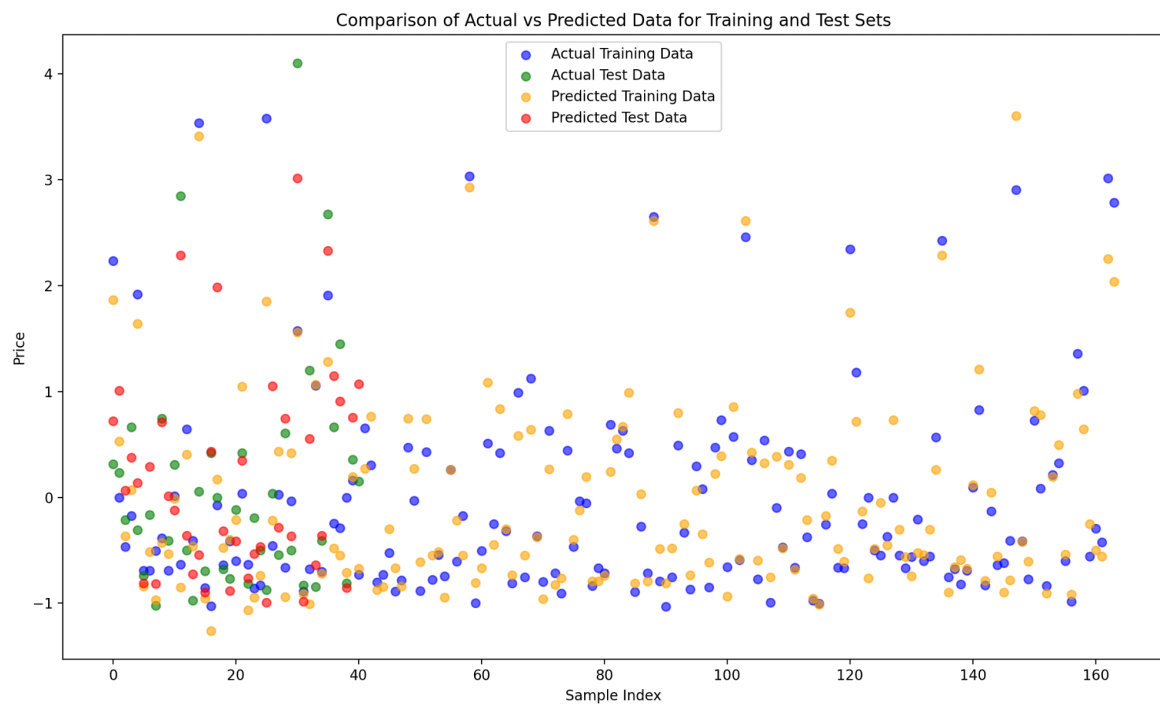
Este gráfico muestra cómo las predicciones del modelo se alinean con los valores reales en el conjunto de entrenamiento.

Datos de Prueba: Reales vs. Predicción



Este gráfico muestra la capacidad del modelo para predecir con datos no vistos.

Datos Reales vs. Predicción



Por último, este gráfico muestra tanto el comportamiento de los datos de predicción de train y test contra los valores reales.

Conclusión.

A lo largo de todo este ensayo, me gustaría pensar que demostré todos los conocimientos que he obtenido en este corto periodo de tiempo respecto a los modelos predictivos y el proceso de ETL. Este ensayo es algo de lo que me enorgullezco porque pude plasmar paso por paso como fue que fui desarrollando mi modelo desde 0, entendiendo las bases y el porqué de las cosas.

Desde la selección y preparación de los datos, hasta la implementación y evaluación del modelo de regresión lineal, cada etapa representó un desafío y una oportunidad para aplicar lo aprendido en clase. Esta primera entrega me hace darme cuenta de lo mucho que estoy disfrutando la concentración porque realmente estoy aprendiendo y comprendiendo lo que estoy haciendo.

Siguiente etapa.

Como siguiente paso, me gustaría que mi siguiente entrega fuera sobre redes neuronales, pudiendo implementar y explicar una desde 0. De igual manera, se que este modelo presenta overfitting. Lo siguiente para mejorar el aprendizaje, es implementar una técnica de regularización. En mi caso estaré implementando la función de Ridge ya que es la que se ajusta de mejor manera a lo que ya tengo codificado.