Predicción de precios de vehículos usados

Grupo 05 – Machine Learning y Procesamiento del Lenguaje Natural

Lina Herrera Bayona - Sandra Tamayo Duque - Wilson Gómez Álvarez - Daniela Castillo Martínez
Universidad de los Andes
Maestría en Inteligencia analítica de datos

El presente informe tiene como fin mostrar el trabajo realizado para predecir el precio de vehículos usados en Estados Unidos, el informe se divide en cinco partes. En la primera parte se revisará el preprocesamiento y caracterización de los datos dispuestos, posteriormente se mostrará el método de calibración del modelo, para entonces proceder al entrenamiento de este. Una vez entrenado se hará la disposición del modelo en AWS a través de una API y, por último, se procederá a realizar las conclusiones respectivas.

1. Caracterización y preprocesamiento de datos

```
# Carga de datos de archivo .csv
dataTraining = pd.read_csv('https://raw.githubusercontent.com/albahnsen/MIAD_ML_and_NLP/main/datasets/dataTrain_carListings.zip')
dataTesting = pd.read_csv('https://raw.githubusercontent.com/albahnsen/MIAD_ML_and_NLP/main/datasets/dataTest_carListings.zip', index_col=0)
```

Ilustración 1: carga de datos

Antes de iniciar el preprocesamiento se va a realizar una breve caracterización de los datos. Se inicia realizando la respectiva carga con las líneas de la Ilustración 1**¡Error! No se encuentra el origen de la referencia.**. En general, se denota que la base de datos cuenta con 400 mil registros y seis columnas.

Tabla 1: Variables

Variable	Descripción	Tipo
Price	Precio de venta del vehículo. Es la variable objetivo en el problema de regresión.	Numérica
Year	Año de fabricación del vehículo. Ayuda a determinar la antigüedad y cómo influye en su precio.	Numérica
Mileage	Kilometraje del vehículo (distancia total recorrida en millas). Indica desgaste y puede afectar el precio.	Numérica
State	Estado de EE. UU. donde se encuentra el vehículo. Puede afectar el precio debido a impuestos y demanda local.	Categórica
Make	Marca del vehículo (ej., Ford, Toyota, BMW). Afecta el precio debido a calidad, rendimiento y percepción.	Categórica
Model	Modelo específico del vehículo dentro de la marca. Afecta el precio debido a características y demanda.	Categórica

Así pues, las variables con su respectivo tipo y dirección se pueden ver en la Tabla 1, la primera variable siendo el precio se considera la variable dependiente dentro del modelo mientras las otras cinco son predictoras. De estás cinco, el año (Year) y millaje (Mileage) son variables continuas mientras que el estado (State), la marca (Make) y el modelo (Model) son categóricas.

```
#DATOS DE ENTRENAMIENTO
print('Dimensiones de la data DE ENTRENAMIENTO\n')
print(dataTraining.shape)

print ("\nSe tiene un total de " +str(dataTraining.isna().sum().sum()) + " datos pérdidos o nulos")
print('\r\nCantidad de datos null or nan por columna:\r\n')
percent_missing = dataTraining.isna().sum() * 100 / len(dataTraining)
missing_value_df = pd.DataFrame({'#_faltantes': dataTraining.isna().sum(),
                                 '%_faltantes': percent_missing})
print(missing_value_df)


print('\r\nTipo de los datos:\r\n')
print(dataTraining.dtypes)
```

Ilustración 2: código datos faltantes

```
Dimensiones de la data DE ENTRENAMIENTO

(400000, 6)

Se tiene un total de 0 datos pérdidos o nulos

Cantidad de datos null or nan por columna:

         #_faltantes  %_faltantes
Price              0          0.0
Year               0          0.0
Mileage            0          0.0
State              0          0.0
Make               0          0.0
Model              0          0.0

Tipo de los datos:

Price       int64
Year        int64
Mileage     int64
State      object
Make       object
```

Ilustración 3: datos faltantes

Ahora, para el preprocesamiento se procedió a revisar si existen datos faltantes, por tanto, se corre el código que se observa en la Ilustración 2 que genera los resultados que se observan en la Ilustración 3 donde se concluye que no existen datos faltantes dentro de la tabla por tanto no es necesaria la imputación.

A continuación, se hará la caracterización de cada una de las variables:

Año:

Esta variable contiene el año de fabricación del respectivo vehículo. Dentro de la Ilustración 4 se observa que la mayor concentración de vehículos se encuentra entre 2013 y 2015, del mismo modo se puede ver que distribución de los años está sesgada hacia épocas más recientes.

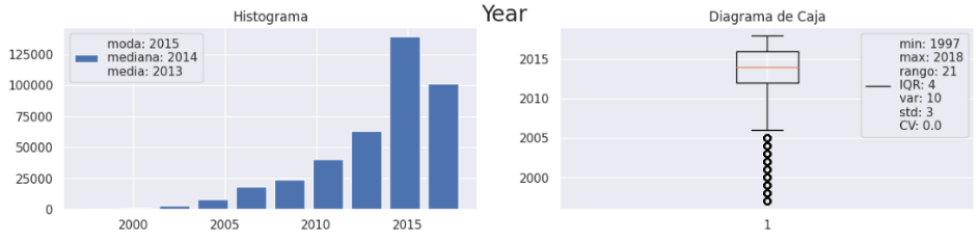

Ilustración 4: caracterización variable año.

El rango de la variable abarca desde 1997 hasta 2018, lo que equivale a 21 años. Es importante señalar que el modelo entrenado con estos datos no considera el impacto de los vehículos clásicos.

En cuanto al IQR, se puede concluir que la mitad de los vehículos están dentro de un rango de cuatro años. Además, la desviación estándar (3) indica que la mayoría de los datos se encuentran a tres unidades de medida por encima o debajo de la media.

Con un coeficiente de variación (CV) cercano a 0, se puede inferir que hay poca variabilidad en los años de fabricación en relación con la media. Es decir, la distribución es bastante homogénea.

Cabe destacar que la distribución de los años de fabricación presenta un sesgo leve hacia vehículos más recientes, aunque también se incluyen algunos vehículos más antiguos en el conjunto de datos.

Millaje:

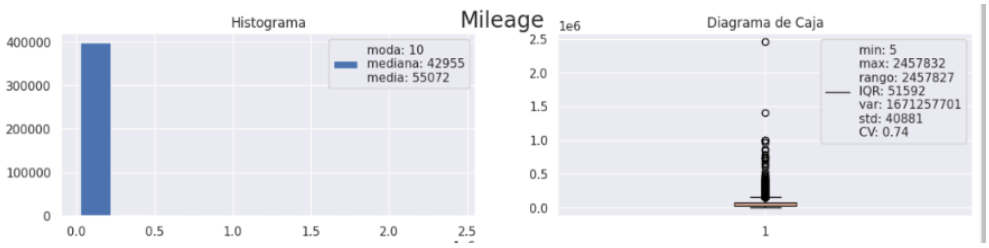

Ilustración 5: caracterización variable Millaje

Para está variable las estadísticas descriptivas muestran que el conjunto de datos contiene vehículos con una amplia gama de millaje. Al igual que la anterior la distribución, esta variable esta sesgada, en este caso, hacia vehículos con menor millaje, es decir, existen datos atípicos muy altos presentes en el conjunto de datos. Esto también se evidencia observando que la media es mayor que la mediana. Además, el CV de 0.74 indica que hay una variabilidad relativamente alta en relación con la media. Esto muestra que los datos están dispersos e incluyen vehículos con niveles de uso muy diferentes.

Estado:

Esta variable tiene un total de 51 Estados donde se encuentra ubicado el vehículo. La Ilustración 6 muestra los 10 Estados con mayor cantidad de vehículos. En general, el análisis muestra una alta concentración en los 10 estados principales, ya que representan aproximadamente el 79.9% de todos los vehículos en la base. Los tres estados con más vehículos, a continuación, se listan las tres clases más frecuentes y se da una caracterización de estas:

Agrupado por State:

	State	Count
43	TX	45918
4	CA	36534
9	FL	33759
10	GA	18182
27	NC	17930
14	IL	16793
45	VA	15894
38	PA	13039
34	NY	12447
31	NJ	12132

Ilustración 6: Top 10 clases del estado.

- Texas (TX) tiene la mayor cantidad de vehículos, con un total de 45.918, que representa el 21.6% de todos los vehículos en la lista. Esto indica que aproximadamente 1 de cada 5 vehículos en la base proviene de Texas
- California (CA) ocupa el segundo lugar con 36.534 vehículos, que representa aproximadamente el 17.2% del total de vehículos. Juntos, Texas y California conglomeran casi el 40% de todos los vehículos en la lista
- Florida (FL) sigue en tercer lugar con 33.759 vehículos, que representa aproximadamente el 15.9% del total. Los tres estados principales, Texas, California y Florida, representan más de la mitad (54.7%) de todos los vehículos en la lista

Marca:

Esta variable tiene un total de 38 clases que representan las marcas de los vehículos. La Ilustración 7 muestra las 10 marcas de vehículos con mayor frecuentas. Del mismo modo se realiza una caracterización de la participación porcentual del top 5:

Agrupado por Make:

	Make	Count
10	Ford	62899
6	Chevrolet	58383
35	Toyota	45941
13	Honda	33191
17	Jeep	24369
12	GMC	20834
18	Kia	16352
8	Dodge	16159
14	Hyundai	15057
20	Lexus	13664

Ilustración 7: top 10 de la variable marca

- Ford: 62.899 vehículos, representando aproximadamente el 18.1% del total.
- Chevrolet: 53.383 vehículos, representando aproximadamente el 16.8% del total.
- Toyota: 45.941 vehículos, representando aproximadamente el 13.2% del total.
- Honda: 33.191 vehículos, representando aproximadamente el 9.5% del total.
- Jeep: 24.369 vehículos, representando aproximadamente el 7.0% del total.

Estas cinco marcas representan el 64.6% del total de vehículos listados en la base.

Modelo:

Esta variable tiene un total de 538 clases que representan los modelos de vehículo. La Ilustración 8 muestra los 10 modelos de vehículos con mayor frecuencia. De mismo modo, se enlistan el top 5 para mostrar su participación porcentual:

- Silverado: 18.085 unidades, que representan el 12.91%
- Grand: 12.344 unidades, que representan el 8.83%
- Sierra: 8.409 unidades, que representan el 6.01%
- Accord: 7.357 unidades, que representan el 5.26%
- F-1504WD: 6.684 unidades, que representan el 4.78%

Agrupado por Model:

	Model	Count
417	Silverado	18085
248	Grand	12344
416	Sierra	8409
40	Accord	7357
187	F-1504WD	6684
489	Wrangler	5914
104	Civic	5766
11	3	5516
264	Jetta	5131
233	FusionSE	4986

Ilustración 8: agrupación por Modelo

Preparación de los datos

Una vez realizado el contexto de los datos a tratar, se procede a realizar el preprocesamiento, las primeras líneas que se corren separan las variables independientes y la variable dependiente como se pueden ver a continuación.

```
# Separar variables independientes (X) y dependiente (y)
X_train = train.drop('Price', axis=1)
y_train = train['Price']
```

Ilustración 9: script separación de variables

Para iniciar, se separa la base para dejar el 77% de los datos para entrenar y el 33% para evaluar el desempeño, como se puede ver en la Ilustración 10.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.33, random_state=42)
```

Ilustración 10: script separación de test y train.

Posteriormente se crea un preprocesador para los datos que se observa en la Ilustración 11 . Lo primero que se realiza es la creación de dos listas: *num_features* y *cat_features*. La primera lista contiene los nombres de las características numéricas mencionadas anteriormente. Mientras que la segunda lista contiene los nombres de las características categóricas.

```
# Crear preprocesador
num_features = ['Year', 'Mileage']
cat_features = ['State', 'Make', 'Model']

preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), num_features),
    ('cat', OneHotEncoder(), cat_features)
])
```

Ilustración 11: script preprocesador.

Siguiente a esto, se define el preprocesador utilizando *ColumnTransformer*. Esta permite aplicar diferentes transformaciones a diferentes columnas del conjunto de datos. En este caso, se aplican dos:

- La primera transformación *'num* utiliza *StandardScaler()*, esta convierte los valores en una escala común, de manera que tengan una media de cero y una desviación estándar de uno.
- La segunda transformación *'cat'* utiliza *OneHotEncoder()*que convierte cada valor de la característica en una columna binaria separada.

2. Calibración del modelo

Una vez realizado el preprocesamiento, se inició la iteración de diferentes modelos para evaluar el poder de predicción, se debe tener en cuenta que se usan exclusivamente modelos para problemas de regresión. En la Ilustración 12, Ilustración 13 e Ilustración 14 se muestran tres ejemplo de lo modelos que se entrenaron.

```
model_bagreg = BaggingRegressor(DecisionTreeRegressor(), n_estimators=50
                                bootstrap=True, oob_score=True, random_state=1

#Entrenar el modelo
model_bagreg.fit(X_train, y_train)
```

Ilustración 12: Modelo Bagging

```
# Crear el modelo de árbol de decisiones
dt_reg = DecisionTreeRegressor(random_state=42)

# Crear el pipeline del modelo
model_pipeline_dt = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', dt_reg)
])


# Entrenar el modelo con GridSearchCV para optimizar los parámetros
grid_search = GridSearchCV(model_pipeline_dt, param_grid, cv=5, n_jobs=-
grid_search.fit(X_train, y_train)
```

Ilustración 13: Árboles de decisión.

```
# Definir el modelo final con hiperparámetros ajustados

final_model = RandomForestRegressor(n_estimators=200, max_depth=12, min_

# Crear modelo de ensamble

model = StackingRegressor(estimators=base_models, final_estimator=final_
pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', mod
pipeline.fit(X_train, y_train)
```

Ilustración 14: Random Forest.

Entre otros modelos se evaluaron Xgboost, Ridge, Lasso y Regresión Lineal. Sin embargo, se pudo evidenciar que el poder predictivo mejoraba cuando se realizaba ensamblaje de modelos, por lo tanto, se inició a entrenar estos modelos, variando los modelos base usados, en un ejercicio parecido al anterior. Al final, los modelos bases con el mejor poder predictivo dentro de un *StackingRegressor* fueron los mostrados en la Ilustración 15.

```
base_models = [
    ('linear_regression', LinearRegression()),
    ('decision_tree', DecisionTreeRegressor(max_depth=10, min_samples_split=20, random_state=42)),
    ('gradient_boosting', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
    ('xgboost', XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
    ('lightgbm', LGBMRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
    ('catboost', CatBoostRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42, verbose=0))
]
```

Ilustración 15: modelos base escogidos.

Con el fin de mejorar el poder predictivo del modelo general se realizó la calibración de los hiperparámetros de cada uno de los modelos base por separado, en la Ilustración 16 se observa el ajuste del modelo Gradient Boosting usando GridSearch con validación cruzada ya que permite explorar una serie de combinaciones de hiperparámetros posibles para determinar la combinación óptima que mejor se adapte a los datos.

```
# Definir los parámetros a optimizar en el GridSearchCV
param_grid_bg = {
    'gb__n_estimators': [50,100,200],
    'gb__max_depth': [1, 2, 4],
    'gb__learning_rate': [0.1, 0.05, 0.01],
}

# Crear el modelo de árbol de decisiones
dt = GradientBoostingRegressor()

# Crear el pipeline del modelo
model_pipeline_gb = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('gb', dt)
])

# Entrenar el modelo con GridSearchCV para optimizar los parámetros
grid_search_gb = GridSearchCV(model_pipeline_gb, param_grid=param_grid_b
grid_search_gb.fit(X_train, y_train)
print(grid_search_gb.best_params_)
```

```
{'gb__learning_rate': 0.1, 'gb__max_depth': 4, 'gb__n_estimators': 200}
```

Ilustración 16: calibración hiperparametros Gradient Boosting

Una vez realizada la parametrización de todos los modelos, se pudo evidenciar que mejoraba significativamente el poder predictivo del modelo, confirmándolo con las medidas de desempeño expuestas en el numeral 3.

3. Entrenamiento del modelo

Una vez se encontró el modelo calibrado, se procedió a realizar el entrenamiento de los datos procesados. En primer lugar, se define un modelo final, que en este caso es un *RandomForestRegressor*. Este modelo

se utiliza para combinar las predicciones de los modelos base vistos en el punto anterior y generar una única predicción final.

```
# Definir el modelo final con hiperparámetros ajustados
final_model = RandomForestRegressor(n_estimators=200, max_depth=12, min_samples_split=15, random_state=42)
```

Ilustración 17: modelo final

A continuación, se crea un modelo de ensamblaje utilizando la función *StackingRegressor* como se puede ver en la Ilustración 18. La función toma como entrada la lista de modelos base, el modelo final y el número de divisiones para la validación cruzada, en este caso se usan 5 divisiones. Se escogió este valor debido a que el tiempo de procesamiento no excedía el tiempo disponible con el que se contaba.

```
# Crear modelo de ensamble
model = StackingRegressor(estimators=base_models, final_estimator=final_model, cv=KFold(n_splits=5, shuffle=True, random_state=42))
pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
pipeline.fit(X_train,  y_train)
y_pred = pipeline.predict(X_test)
```

Ilustración 18: modelo de ensamblaje

Después de crear el modelo de ensamblaje, se utiliza un Pipeline para aplicar el preprocesamiento visto en el primer punto y entrenar el modelo completo. Los datos de entrenamiento se ajustan al Pipeline y luego se utilizan para generar predicciones en los datos de prueba. Luego se generaron las medidas de desempeño como se puede ver a continuación.

```
#Métricas
#MAE (error absoluto medio): Es el promedio de la diferencia absoluta entre el valor observado y los valores predichos (debe acercarse a 0)
MAE = mean_absolute_error(y_test, y_pred)
#Arreglo para acumular las medidas para comparar al final
print('MAE:',MAE)

#RMSE devuelve la fiabilidad del modelo
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
#Arreglo para acumular las medidas para comparar al final
print('RMSE:',RMSE)

#Otras métricas
# calcular R cuadrado -> calidad del ajuste del modelo a los datos observados
R2 = r2_score(y_test, y_pred)
#Arreglo para acumular las medidas para comparar al final
# imprimir los resultados
print('R2:',R2)

# calcular MAPE Error Porcentual Absoluto Medio ->se expresa como un porcentaje, y cuanto más cercano a cero sea el valor del MAPE, mayor será la precisión
MAPE = np.mean(np.abs((y_test - y_pred)/y_test))*100
# imprimir los resultados
print('MAPE:',MAPE)
```

Ilustración 19: Código de métricas de desempeño

Por último, después de entrenar el modelo completo, generar predicciones en lo
s datos de prueba y las medidas de desempeño, se evalúan las métricas para determinar la capacidad del modelo para generalizar a nuevos datos en la Ilustración 20 se pueden ver los valores de estas.

```
MAE: 2459.2524528028785
RMSE: 3758.3641970379695
R2: 0.8772239066264903
MAPE: 12.340828807765462
```

Ilustración 20: desempeño del modelo

De acuerdo con estas métricas se puede concluir que:

- En promedio, el modelo tiene una diferencia absoluta de 2.459 dólares entre las predicciones y los precios reales de los vehículos usados en el conjunto de prueba.
- Teniendo en cuenta que el RMSE es mayor que el MAE y que a diferencia de este penaliza los errores más grandes. Se sugiere que el modelo esta cometiendo errores importantes.

- El modelo explica el 87% de la varianza en los datos de prueba.
- en promedio, las predicciones del modelo difieren del valor real en un 12.3%

En general, el modelo tiene un buen desempeño. Adicional a esto, un modelo con *Stacking* tiene muchas ventajas en comparación con otras opciones, lo cual también representó una parte importante a la hora de escogerlo, a continuación, se presentan:

- los modelos de ensamble combinan las predicciones de varios modelos más simples para mejorar la precisión de las predicciones finales, esto reduce el riesgo de sobreajuste.
- Estos modelos son flexibles y pueden trabajar con diferentes tipos de modelos base, lo que permitió combinar los modelos que se evaluaron en la primera etapa de experimentación.
- En este caso teniendo en cuenta que existen clases poco representadas en marcas y modelos, el modelo escogido es bueno manejando datos desbalanceados.

En resumen, la elección de este modelo de ensamble con *Stacking* se basa en su capacidad para combinar las ventajas de varios modelos más simples y mejorar la precisión de las predicciones finales.

4. Disponibilización del modelo

Una vez se entrenó el modelo, se procede a realizar la disponibilización de los datos en una API alojada en AWS. La IP Pública de AWS es http://3.93.241.63:5000/ donde se puede ver la ventana mostrada en la Ilustración 21 donde se ingresan los valores de las 5 variables usadas para entrenar el modelo en un formato tipo diccionario como se puede ver en la misma ilustración.

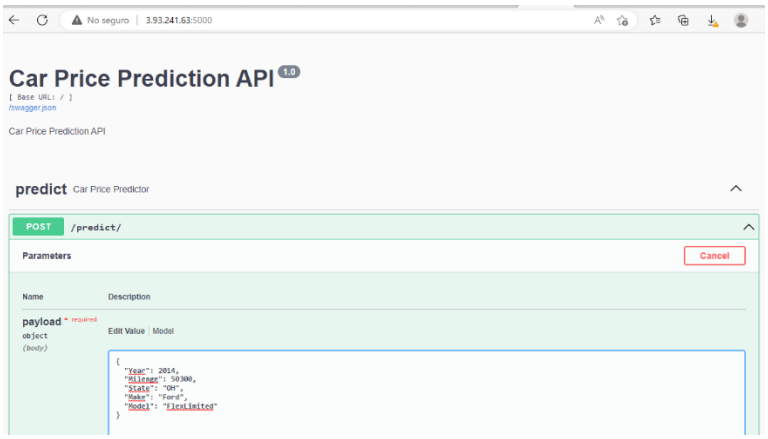

Ilustración 21: ventana Car Price Prediction API

Una vez se añaden los parámetros se procede a ejecutar la API, en este punto se genera una pantalla como la mostrada en la Ilustración 22 donde en el cuerpo de la respuesta se observa el precio predicho por el modelo en un formato tipo diccionario.

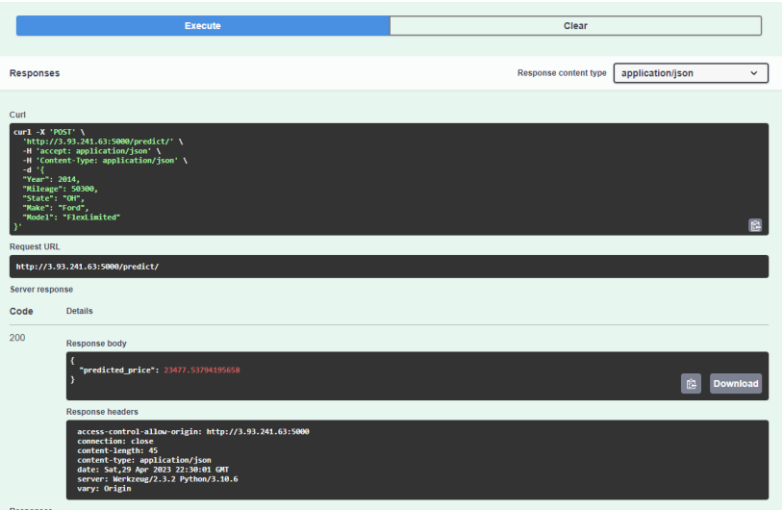

Ilustración 22: ventana resultados API

Con el fin de mostrar el uso de la API y su poder predictivo a continuación se muestran dos ejemplos obtenidos del set de validación.

En el primero se tiene un vehículo Cadillac SRXLuxury del 2014, con un millaje de 28729 y State OH. El ingreso de los parámetros se puede ver en la parte izquierda de la UKFAO y en a derecha el resultado obtenido. Ya que la observación la extrajimos del Dataset se sabe que el valor real es de 24.681 y el valor predicho es de 26.990, podemos evidenciar un error de 2.309

	Price	Year	Mileage	State	Make	Model
0	34995	2017	9913	FL	Jeep	Wrangler
1	37895	2015	20578	OH	Chevrolet	Tahoe4WD
2	18430	2012	83716	TX	BMW	X5AWD
3	24681	2014	28729	OH	Cadillac	SRXLuxury

Ilustración 23: valores primer ejemplo de implementación.

Ilustración 24: implementación ejemplos API primer ejemplo.

Mientras que para el segundo ejemplo, al tener un vehículo FL Jeep Wrangler modelo 2014, el error de predicción aumenta a 5.454.

```
dataTraining
```

	Price	Year	Mileage	State	Make	Model
0	34995	2017	9913	FL	Jeep	Wrangler

Ilustración 25: Valores segundo ejemplo de implementación.

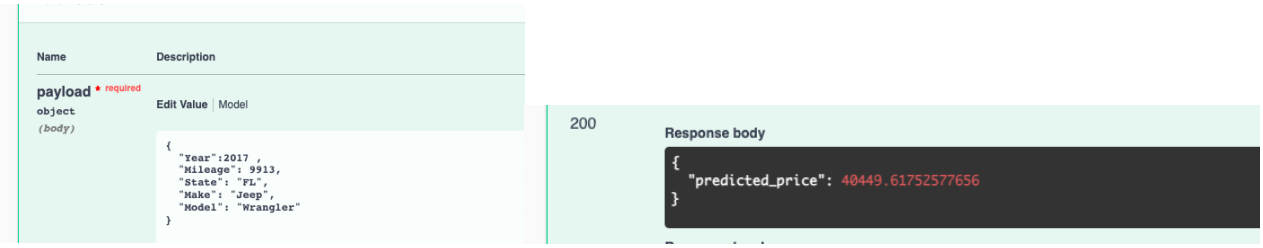

Ilustración 26: implementación ejemplos API segundo ejemplo.

5. Conclusiones

En general, durante el desarrollo de este proyecto se pudieron observar en la práctica las ventajas y desventajas de los modelos aprendidos durante la clase.

- ✓ Se entrenaron más de 50 modelos diferentes para finalmente llegar al modelo expuesto en este informe. Se ejecutaron múltiples instancias de calibración de hiperparámetros y, finalmente, se logró crear una aplicación que hace uso de tecnologías en la nube.
- ✓ En cuanto al alcance del modelo, en principio se puede deducir que no será bueno para predecir valores de vehículos con un valor adicional al servicio de transporte, como, por ejemplo, vehículos de colección. Del mismo modo, los datos disponibles no permiten segmentar los mercados más allá de solo el estado, como conocer si es un vehículo que se encuentra en un área rural o urbana. Además, el modelo está limitado a Estados Unidos, lo que representa un problema de escalabilidad.
- ✓ En cuanto a los resultados, en general, se puede decir que el modelo está haciendo predicciones bastante buenas basándose en las medidas de evaluación descritas en el punto tres. Sin embargo, el modelo puede estar cometiendo errores graves en casos específicos, por lo que sería importante que un experto verifique la lógica del resultado dado por el modelo antes de generar resultados a partir de este.
- ✓ Por último, en cuanto a la implementación del proyecto, se presentaron importantes retos a la hora de evaluar los hiperparámetros y los diferentes modelos, ya que los tiempos de procesamiento eran altos y limitaban la generación de resultados. Del mismo modo, el ejercicio de disponibilización en la nube de AWS fue altamente constructivo, ya que se presentaron retos que no se habían considerado en la clase y se pudieron aclarar diferentes dudas, logrando desarrollar el ejercicio práctico.