PROYECTO 1 – PROCESAMIENTO DEL LENGUAJE NATURAL

Por: Jaime Unriza, Roberto Bustamante e Iván Amarillo

Procesamiento de Datos y Análisis Exploratorio:

Para llevar a cabo el proceso de predicción de precio de los automóviles era necesario primero entender los datos disponibles dentro de cada uno de los datasets dispuestos ya que teníamos disponible un dataset para poder hacer todo el proceso de entrenamiento de los modelos a seleccionar y otro dataset de validación y predicción para poder probar el desempeño del modelo seleccionado y su predicción emplearla para la competencia.

Al explorar los datos encontramos que el dataset de entrenamiento tenía la columna Precio (variable dependiente), con la cual podríamos entrenar los modelos seleccionados en función a las variables de decisión que seleccionáramos posteriormente, mientras que el dataset de testing no contaba con esta variable o esta columna de Precio ya que iba a arrojar la predicción y la íbamos a someter a competencia dentro de la plataforma de Kaggle.

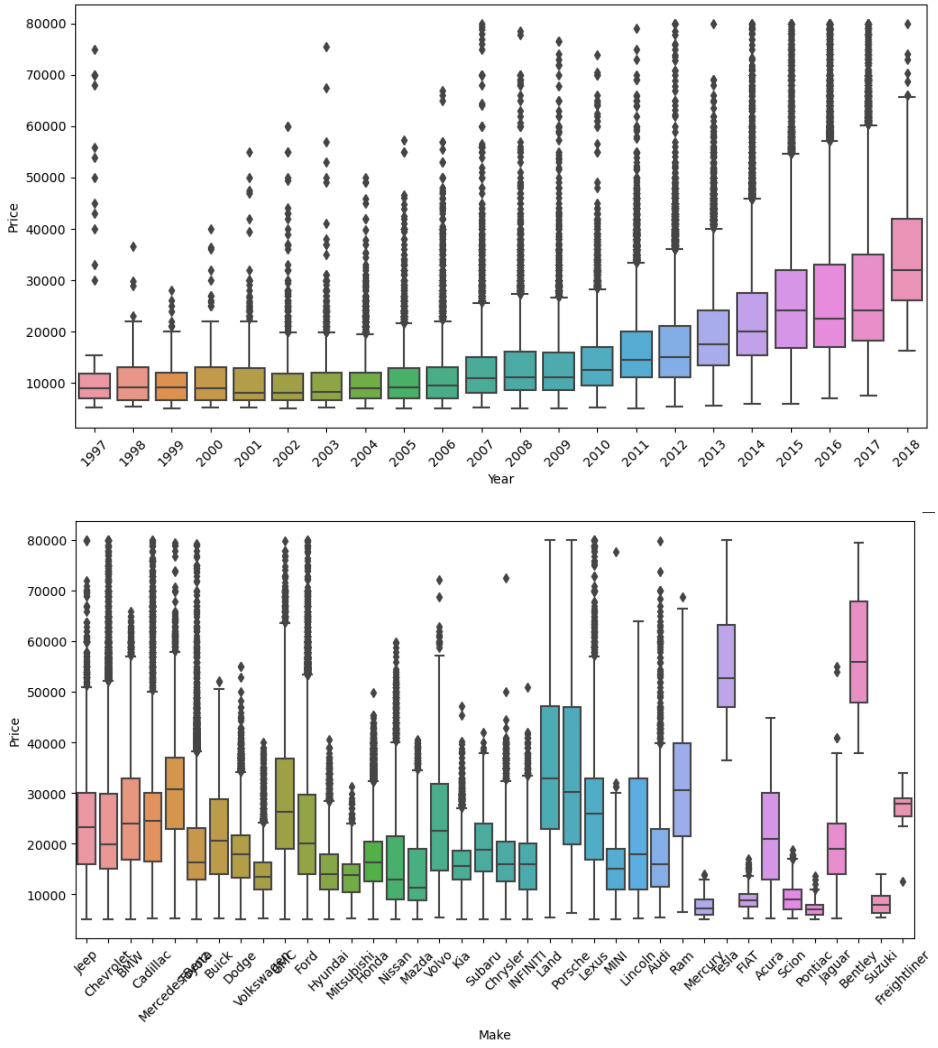
```
----------------------------------------
              Price           Year       Mileage
count  400000.000000  400000.000000  4.000000e+05
mean    21146.919312    2013.198125  5.507296e+04
std     10753.664940       3.292326  4.088102e+04
min      5001.000000    1997.000000  5.000000e+00
25%     13499.000000    2012.000000  2.584100e+04
50%     18450.000000    2014.000000  4.295500e+04
75%     26999.000000    2016.000000  7.743300e+04
max     79999.000000    2018.000000  2.457832e+06
----------------------------------------
Cantidad de Filas y Columnas:
(400000, 6)
```

Dentro de las variables de entrenamiento encontramos 400.000 registros con una media en precio de 21.146 USD y una desviación estándar de 10.753 USD con un mínimo de 5.001 USD un percentil del 25% de 13.499 USD la mediana en 18.450 USD, un percentil 75% en 26.999 USD y un máximo de 79.999 USD.

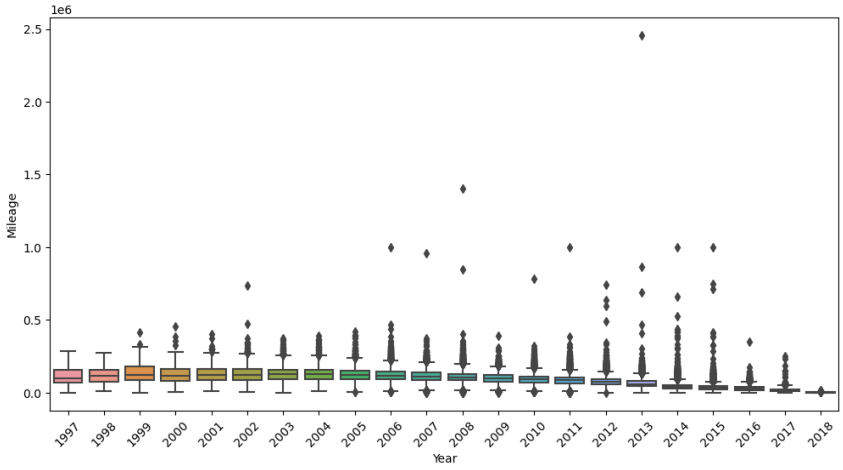
Para el parámetro correspondiente al año, encontramos un promedio de registros del año 2013, una desviación estándar de 3 años, siendo el menor de ellos o el más antiguo de 1997 y el más moderno de 2018 encontrando la mediana en 2014 y los percentiles 25% y 75% en 2012 y 2016 respectivamente.

Por último, en la columna de millas recorridas por vehículo encontramos un promedio dentro de los datos de 55.072 millas con una desviación estándar de 4088 millas, un mínimo de 5 millas recorridas y un máximo de 2.457.832 millas. Esto cubriría nuestro análisis de exploratorio de las variables numéricas proporcionadas en el data set de entrenamiento.

De este punto en adelante, decidimos realizar el contraste entre el precio versus el año usando gráficos de caja que nos permitirían identificar como a través de los años se pueden identificar tendencias de aumento de los precios y concentraciones medias cada vez más altas al realizar la revisión entre los precios y los años al igual que los precios y las marcas y las millas y los años como se muestran en los gráficos a continuación.



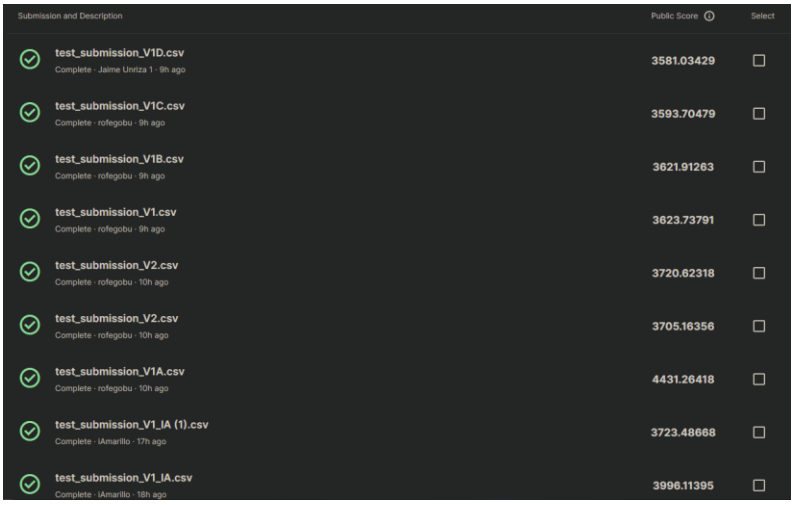
Por medio de estos gráficos fue posible identificar la dispersión de los datos a pesar de ser agrupados esto se puede explicar ya que una marca al tener diferentes modelos puede sacar modelos de gama alta media y baja con precios bastante diferentes entre sí en relación al kilometraje.



Es de resaltar que, visualmente se pueden identificar datos outliers (por fuera del rango esperado y dispersión según los boxplots), primera aproximación para decidir eliminar estos datos y hacer el proceso de modelamiento mucho más preciso.

Así mismo, decidimos crear unos gráficos de caja empleando widgets para poder hacer filtros dinámicos mediante la selección por lista de atributos como el fabricante y el modelo y poder hacer una evaluación de los datos de forma transversal y dinámica que nos diera mucha más información de la dispersión de estos.

Posterior a este ejercicio decidimos eliminar valores atípicos o outlets haciendo un método de filtrado empleando como métrica principal un corte dentro del dataset mediante desviaciones estándar determinadas que finalmente, después de varias iteraciones y corridas (sometidas a competencia en Kaggle), decidimos mantener en 1.8 para los cortes finales que nos daban mejor desempeño en la competencia. A continuación mostramos un screenshot de los diferentes modelos que subimos a la competencia con diferentes métricas en cuanto al corte según desviaciones estándar oscilando entre una y dos desviaciones estándar para hacer el corte y filtrado de outliers.



Con este método diseñamos dos metodologías de filtrado (metodología1 y metodología2, ver imagen a continuación) tanto en metodología 1 como metodología 2 encontramos que el método de filtrado era más ajustado mediante la metodología 2 metodología que decidimos conservar para poder hacer el entrenamiento del modelo.

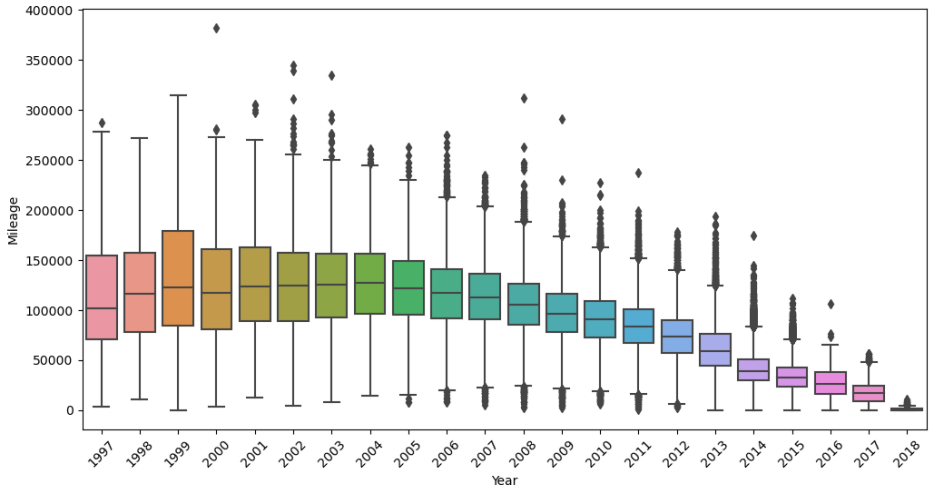
Identificar y Eliminar Valores Atípicos (Outliers) - Filtrado 1

```
[ ]
# Identificar y eliminar valores atípicos de 'Price' y 'Milage' para cada 'Model' y 'Year',

# Agrupar por 'Year' y 'Model' y calcular la media y la desviación estándar para 'Price' y 'Mileage'
grouped = dataTraining.groupby(['Year', 'Model']).agg({
    'Price': ['mean', 'std'],
    'Mileage': ['mean', 'std']
}).reset_index()

# Aplanar encabezado de las columnas multi-índice
grouped.columns = ['Year', 'Model', 'Price_mean', 'Price_std', 'Mileage_mean', 'Mileage_std']

# Unir los cálculos anteriores al DataFrame original para tener los valores de referencia por año y modelo
dataTraining = dataTraining.merge(grouped, on=['Year', 'Model'], how='left')

# Identificar valores atípicos usando la media y la desviación estándar por año y modelo
outliers = dataTraining[
    ((dataTraining['Price'] < (dataTraining['Price_mean'] - 1.8 * dataTraining['Price_std'])) |
     (dataTraining['Price'] > (dataTraining['Price_mean'] + 1.8 * dataTraining['Price_std']))) |
    ((dataTraining['Mileage'] < (dataTraining['Mileage_mean'] - 1.8 * dataTraining['Mileage_std'])) |
     (dataTraining['Mileage'] > (dataTraining['Mileage_mean'] + 1.8 * dataTraining['Mileage_std'])))
]

# Eliminar filas con valores atípicos
dataTraining_filtrado1 = dataTraining.drop(outliers.index)

# Eliminar columnas auxiliares usadas para los cálculos
dataTraining_filtrado1 = dataTraining_filtrado1.drop(columns=['Price_mean', 'Price_std', 'Mileage_mean', 'Mileage_std'])

# Mostrar DataFrame actualizado
print(dataTraining_filtrado1)
```

Identificar y Eliminar Valores Atípicos (Outliers) - Filtrado 2

```
# Identificar y eliminar valores atípicos de 'Price' para cada 'Model', 'Year' y 'Mileage_range'

# Crear la columna 'Mileage_range'
dataTraining_filtrado1['Mileage_range'] = (np.ceil(dataTraining_filtrado1['Mileage'] / 5000) * 5000).astype(int)

# Agrupar por 'Year', 'Model', y 'Mileage_range' y calcular la media y la desviación estándar para 'Price'
grouped = dataTraining_filtrado1.groupby(['Year', 'Model', 'Mileage_range']).agg({
    'Price': ['mean', 'std']
}).reset_index()

# Aplanar encabezado de las columnas multi-índice
grouped.columns = ['Year', 'Model', 'Mileage_range', 'Price_mean', 'Price_std']

# Unir los cálculos anteriores al DataFrame original para tener los valores de referencia
dataTraining_filtrado1 = dataTraining_filtrado1.merge(grouped, on=['Year', 'Model', 'Mileage_range'], how='left')

# Identificar los valores atípicos usando la media y la desviación estándar para 'Price'
outliers = dataTraining_filtrado1[
    ((dataTraining_filtrado1['Price'] < (dataTraining_filtrado1['Price_mean'] - 1.7 * dataTraining_filtrado1['Price_std'])) |
     (dataTraining_filtrado1['Price'] > (dataTraining_filtrado1['Price_mean'] + 1.7 * dataTraining_filtrado1['Price_std'])))
]

# Eliminar filas con valores atípicos
dataTraining_filtrado2 = dataTraining_filtrado1.drop(outliers.index)

# Eliminar columnas auxiliares usadas para los cálculos
dataTraining_filtrado2 = dataTraining_filtrado2.drop(columns=['Price_mean', 'Price_std', 'Mileage_range'])

# Mostrar DataFrame actualizado
print(dataTraining_filtrado2)
```

A continuación, presentamos los gráficos posteriores al filtrado y con los cuales procedimos a entrenar los modelos.



Calibración de Modelos:

Empezamos el ejercicio de calibración de los modelos con el preprocesamiento de los datos como se muestra en la imagen a continuación, esto para garantizar que el ingreso de variables numéricas y categóricas en los modelos fuera de forma adecuada.

```
# Selección de características
numeric_features = ['Year', 'Mileage']
categorical_features = ['State', 'Make', 'Model']

# Pipeline para características numéricas
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

# Pipeline para características categóricas
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Combinar transformadores en un preprocesador
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Separación de variables predictoras (X) y variable de interés (y)
y = dataTraining_filtrado2['Price']
X = dataTraining_filtrado2.drop('Price', axis=1)

# División en conjuntos de entrenamiento y validación
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.33, random_state=42)

# Aplicar el preprocesador
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_valid_preprocessed = preprocessor.transform(X_valid)
```

```
preprocessor
```

ColumnTransformer
num
SimpleImputer
StandardScaler
cat
SimpleImputer
OneHotEncoder

Como mencionamos anteriormente, decidimos calibrar parámetros pertinentes del modelo alrededor de cuántas desviaciones estándar y así eliminar outliers que no cumplieran con la dispersión máxima dispuesta para el grupo de datos. Posteriormente, empezamos a entrenar diferentes modelos sobre el dataset de entrenamiento empezando con el entrenamiento de un modelo de bagging calibrando entre 10 y 20 estimadores y calculando su RMSE.

Así mismo, entrenamos un modelo xgboost y un random forest encontrando para sus resultados un RMSE como se expone a continuación.

```
RMSE Bagging: 10305.59
MAE Bagging: 8134.39
------------------------------------------------------------
RMSE XGBoost: 3569.18
MAE XGBoost: 2662.65
------------------------------------------------------------
RMSE RandomForest: 2852.09
MAE RandomForest: 1874.07
```

Siendo hasta este punto el random forest el que decidimos mantener durante el ejercicio, calibramos en torno a diferentes parámetros como el número de estimadores y encontramos que la mejor distribución para el número de estimadores en caso del random forest era mantener 100 estimadores, razón por la cual decidimos quedarnos con estos para hacer el comparativo que expusimos anteriormente.

Entrenamiento del modelo:

Random Forest con calibración de parámetros:

Para el entrenamiento de modelos definitivos damos por entrenar un random forest con calibración de parámetros y un XGBoost con calibración de parámetros respectivos. Para el random forest con calibración, identificamos que nuestro mejor número de árboles oscila entre 100 y 500 la profundidad máxima entre 10, 30 y 50, el número mínimo de muestras necesarias para dividir un nodo entre 2 y 16, dándonos como resultado un RMSE de 2,627.6457 para el random forest calibrado.

```python
# Creación del espacio de parámetros para Random Search
param_dist = {
    'n_estimators': randint(100, 501),  # Número de árboles
    'max_depth': [None, 10, 30, 50],  # Profundidad máxima de cada árbol
    'min_samples_split': randint(2, 16),  # Número mínimo de muestras necesarias para dividir un nodo
    'min_samples_leaf': randint(1, 5),  # Número mínimo de muestras en una hoja
    'max_features': ['auto', 'sqrt', 'log2']  # Número de características para considerar en cada split
}
```

```
Mejor score (RMSE): 2627.6457197972777
```

XGBoost con calibración de parámetros:

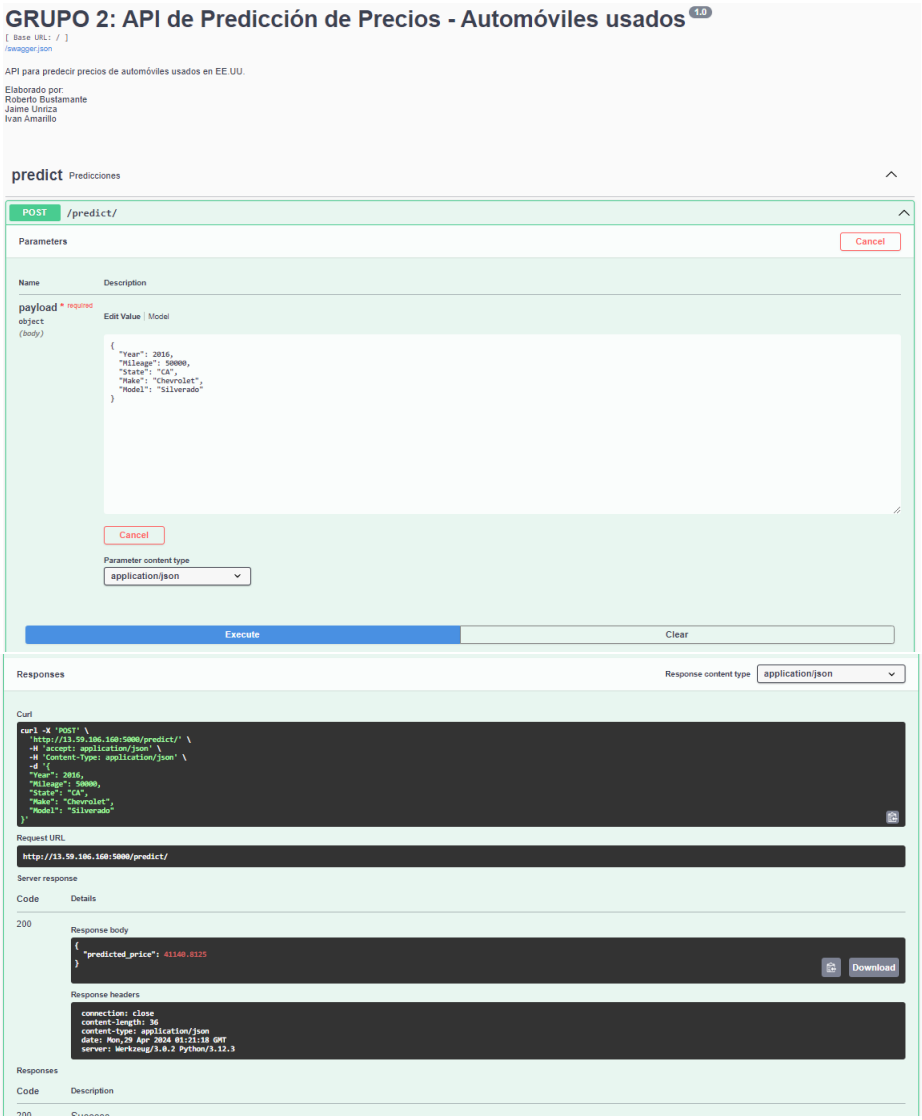
Asimismo, para el XGBoost calibrado hicimos iteración entre 100 y 500 estimadores, profundidad entre 3 y 10 una tasa de aprendizaje uniforme entre 0.05 y 0.25 y un gamma uniforme entre 0 y 1 logrando con este un RMSE 2,577.65 siendo este el mejor modelo entrenado y por tanto el seleccionado para someter a la competencia.

```
# Espacio de parámetros para Random Search
param_dist = {
    'n_estimators': randint(100, 501),  # 100 a 500
    'max_depth': randint(3, 11),  # 3 a 10
    'learning_rate': uniform(0.05, 0.25),  # desde 0.05 hasta 0.30
    'subsample': uniform(0.5, 0.5),  # desde 0.5 hasta 1.0
    'colsample_bytree': uniform(0.5, 0.5),  # desde 0.5 hasta 1.0
    'gamma': uniform(0, 1)  # desde 0 hasta 1
}
```

```
RMSE XGBoost post-calibración: 2577.65
```

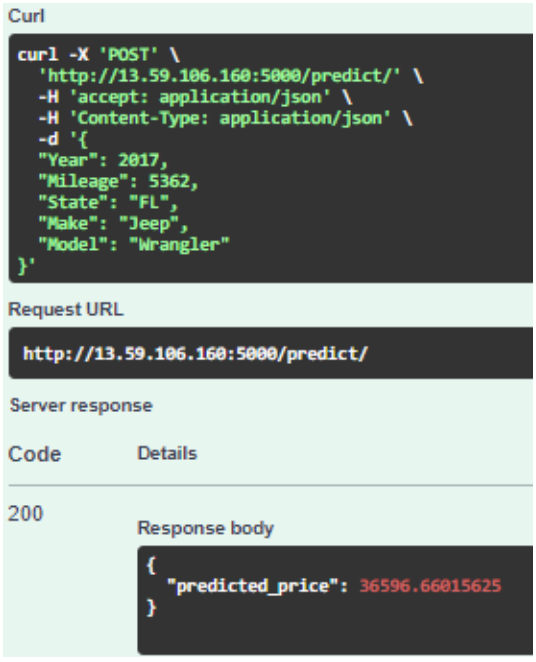
Disponibilización del modelo:

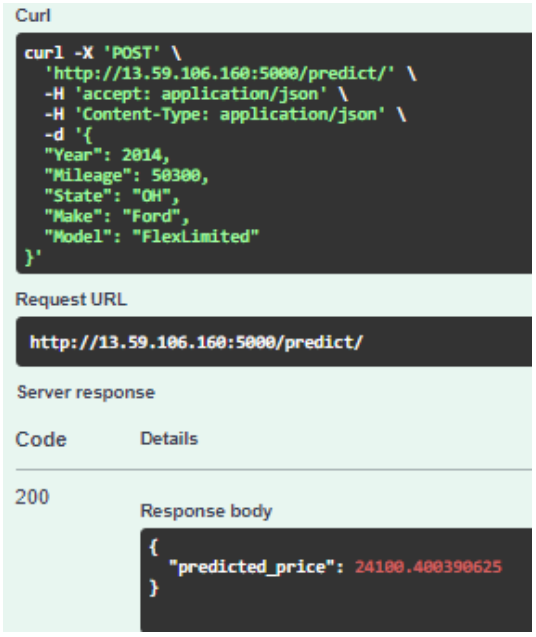
Igualmente, diseñamos una API para la predicción de precios de automóviles usados la cual pusimos a disposición a través de Amazon Web Service (AWS) y se encuentra activa en este momento a través del link que exponemos a continuación y los pantallazos expuestos muestran la estructura que tiene en su front end para interacción con la API y los parámetros necesarios a incluir para poder obtener un precio a manera de predicción sobre un vehículo usado.

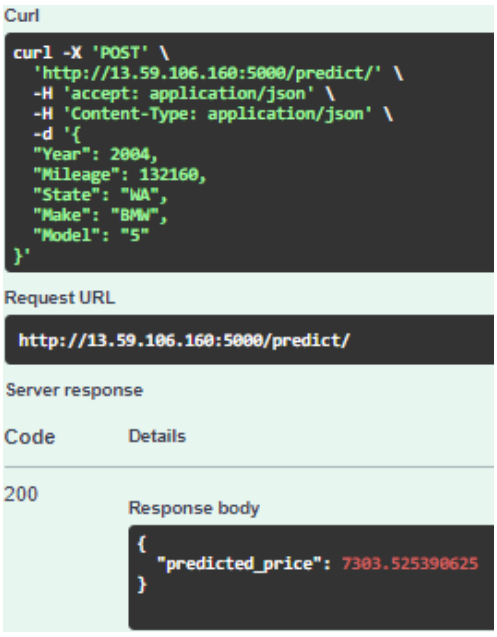


http://13.59.106.160:5000/

A continuación, mostramos también un ejemplo del uso de la API sobre cuatro ejemplares de la base de datos de test. Para cada uno de ellos, se predice un precio en función del año, el kilometraje, el estado al que pertenece, el fabricante y el modelo respectivo.







Conclusiones:

A manera de conclusión quisiéramos exponer como a lo largo de este estudio se empleó un reconocimiento generalizado de los datos tanto de variables categóricas como cuantitativas, se preprocesaron los datos para garantizar la entrada adecuada de estas variables en el entrenamiento y validación de los diferentes modelos propuestos y se desarrolló un portafolio de modelos entrenados sometidos a calificación mediante el RMSE con el cual seleccionamos el XGBoost calibrado como modelo a proponer tanto para el desarrollo de este documento como para la competencia.

Asimismo, podemos ver la implementación de este modelo y sus predicciones en acción a través de la interacción con la API dispuesta en Amazon Web Service (AWS) y los resultados para diferentes combinaciones de parámetros correspondientes a vehículos usados.

Concluimos entonces, que un modelo XGBoost calibrado presenta una mejor bondad de ajuste y capacidad predictiva frente a los otros modelos como bagging, random forest y random forest calibrado y consideramos es una buena aproximación para poder predecir precios basándonos en su comparativa relativa del RMSE.