# INSTITUTO SUPERIOR POLITÉCNICO DE CÓRDOBA

## TECNICATURA SUPERIOR EN CIENCIA DE DATOS E INTELIGENCIA ARTIFICIAL

### Módulo de Analista de Datos: Evidencia 2


Docentes:

- Pratta, Nahuel
- Ugarte, Marcos

Integrantes:

- López, Erick
- Nüesch, Christian
- Zurita, Débora

# INDICE

# INTRODUCCIÓN

## Descripción del proyecto

El proyecto consiste en el análisis a través de diferentes herramientas profesionales, de los datos recolectados a través de un peinado web (o web scrapping) con la información de varias características de automóviles ordenadas en doce diferentes columnas.

Este procesamiento y limpieza de los datos para realizar un posterior análisis se realizó con la finalidad de articular lo visto en los diferentes espacios curriculares de la carrera Ciencia de Datos e Inteligencia Artificial. Estos espacios curriculares son "Estadística y Exploración de Datos", "Procesamiento de Datos" y "Ciencia de Datos".

## Objetivos generales del proyecto

Desarrollar la capacidad de organizar y planificar eficazmente, aplicando los conocimientos para identificar problemas relevantes en el contexto profesional. Se deberá comunicar los hallazgos de manera clara, tanto de forma oral como escrita. Asimismo se busca el razonamiento crítico y desarrollar actitudes importantes como la precisión, la revisión crítica, la tolerancia y el compromiso ético con la igualdad de oportunidades, sin discriminación por sexo, raza o religión, y con atención a la diversidad. Por último, se busca mejorar la capacidad para tomar decisiones y trabajar en equipo, colaborando en entornos multidisciplinares para alcanzar los objetivos propuestos.

## Objetivos específicos del proyecto

Desarrollar la capacidad de limpiar y transformar los datos crudos para poder eliminar errores, valores atípicos, valores ausentes y demás inconsistencias; para luego aplicar análisis gráficos y descriptivos, y relacionar diferentes variables.

# DESARROLLO

## Origen de los datos:

https://www.kaggle.com/datasets/ahmettalhabektas/argentina-car-prices

Este dataset comprende varias características de automóviles ordenadas en diferentes columnas, obtenidas mediante peinado web (web scraping) de un sitio web en Argentina a principios de enero de 2023.

## Descripción de cada columna:

(Los precios son para enero del 2023)

- Precio:
    - El precio del auto.
    - Tipo de dato: entero.
    - Valores posibles de la columna: múltiples valores numéricos enteros.
- Marca:
    - La marca del auto.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "Audi", "Baic", "BMW", "Chery", "Chevrolet", "Citroën", "Dodge", "DS", "Fiat", "Ford", "Honda", "Hyundai", "Jeep", "Kia", "Mercedes-Benz", "Mini", "Mitsubishi", "Nissan", "Peugeot", "Porsche", "RAM", "Renault", "Subaru", "Suzuki", "Toyota", "Volkswagen", "Volvo".
- Modelo:
    - El modelo del auto.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: múltiples valores de cadena de caracteres.
- Año:
    - El año del auto.

    - Tipo de dato: entero.
    - Valores posibles de la columna: "1995", "1996", "1997", "2000", "2007", "2008", "2009", "2010", "2011", "2012", "2013", "2014", "2015", "2016", "2017", "2018", "2019", "2020", "2021", "2022".
- Color:
    - El color del auto.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "Azul", "Beige", "Blanco", "Celeste", "Dorado", "Gris", "Gris oscuro", "Marrón", "Naranja", "Negro", "Plateado", "Rojo", "Verde", "Violeta".
- Combustible:
    - El tipo de combustible que utiliza.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "Diésel", "Híbrido/Nafta", "Nafta", "Nafta/GNC".
- Puertas:
    - La cantidad de puertas del auto.
    - Tipo de dato: entero.
    - Valores posibles de la columna: "2", "4", "5".
- Caja:
    - El tipo de caja de cambios.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "Automática", "Manual".
- Motor:
    - El tamaño del motor (en litros).
    - Tipo de dato:
    - Valores posibles de la columna: múltiples valores numéricos con un decimal.
- Carrocería:
    - El tipo de carrocería.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "Coupé", "Crossover", "Furgón", "Hatchback", "Minivan", "Monovolumen", "Pick-Up", "Rural", "Sedán", "SUV".

- Kilómetros:
    - El kilometraje del auto.
    - Tipo de dato: entero.
    - Valores posibles de la columna: múltiples valores numéricos enteros.
- Moneda:
    - La moneda en que se cotiza.
    - Tipo de dato: cadena de caracteres.
    - Valores posibles de la columna: "dólares", "pesos".

## Estado de los datos

Los datos están en RAW (o datos crudos), lo que significa que están en su forma original, sin ningún tipo de procesamiento, limpieza o transformación. Estos datos fueron recogidos directamente de la fuente y no han sido manipulados para eliminar errores, duplicados, valores faltantes, etc.
No se le realizaron operaciones de limpieza o transformación, por lo tanto contienen errores, valores atípicos, valores ausentes y algunas inconsistencias.
Los datos están en distintos formatos (texto y diferentes tipos de números) y contienen toda la información capturada de la fuente, lo que los hace útiles para la práctica y el aprendizaje.

# Desarrollo de la primera parte

Para empezar a trabajar con el dataset lo primero que hicimos fue importar las librerías de Python que íbamos a utilizar (numpy, pandas y matplotlib.pyplot)
Luego leímos los datos desde el archivo que los contiene, en este caso un archivo de valores separados por comas (csv) e imprimimos los primeros diez registros del dataset para poder hacernos una idea de qué se trataba. Siempre es posible imprimir las últimas filas del dataset (tanto para *head* como para *tail* por defecto siempre se imprimen cinco filas).

También es posible imprimir todo el dataset, aunque *se debe tener sumo cuidado al realizar esta operación* si el dataset es muy grande, ya que todo deberá ser cargado en memoria.

Con el método *info()* pudimos consultar múltiples datos de la filas y lasa columnas:

- el tipo de datos de cada columna
- el número de valores no nulos de cada columna
- el uso de memoria del dataset

Otros datos específicos que se consultaron fueron:

- los nombres de las columnas
- la cantidad total de celdas o elementos del dataset
- una lista con los nombres de las filas del dataset (que en nuestro caso no tuvo mucho sentido esta información, pero sí puede servir para otros casos)
- si faltaban datos en alguna fila
- si faltaban datos en alguna columna. En nuestro caso se pude ver que en las columnas 'Color', 'Caja', 'Motor' y 'Carrocería' había datos ausentes (en todos los casos el total debía ser 510)
- qué cantidad hay de cada elemento en cada columna, simplemente ajustando o indicando la columna a la que nos referimos

Otra cosa muy importante a la hora de hacer procesamiento de datos es calcular los valores válidos sobre el total. Esta instancia es un poco más compleja, ya que se debe utilizar una combinación de varios varios métodos. Además, como la operación se debe realizar varias veces, decidimos empaquetar todo el código en una función, y así lo reutilizamos al código.
Un análisis extra fue el siguiente: como trabajamos con información sobre autos puestos a la venta, fue bueno saber cuántos modelos había en total disponibles en la plaza, y su marca comercial correspondiente.
Como la información devuelta era mucha, ajustamos la configuración de "Pandas" para mostrar todas las filas, pero igual que en el ejemplo anterior, esto hizo que se muestren todas las filas obtenidas, por lo que en determinados casos podría hacer que se consuman muchos recursos del sistema. Si lo que se deseaba era cambiar esto, se comentaba esta instrucción y se debía volver a ejecutar el notebook (ya que "Pandas" habrá quedado precargado para mostrar todas las filas).

## Análisis estadísticos del dataset

Una parte muy importante de este trabajo es poder poner en práctica los conocimientos adquiridos en el espacio curricular “Estadística y exploración de datos”, por tanto a partir de este punto buscamos diferentes datos e informaciones de tipo estadístico del dataset con el que estábamos trabajando (siempre para las columnas con datos numéricos):

- menor de los datos
- para la columna "Motor" no podemos obtener de manera inmediata el valor mínimo porque había valores mal registrados (y lo mismo va a ocurrir para todos los análisis que impliquen valores numéricos)
- mayor de los datos
- media de los datos
- la varianza de los datos
- la desviación estándar de los datos

Luego volvimos a utilizar el método *describe()* pero indicándole de forma específica las columnas sobre las que queremos trabajar, y así pudimos obtener la misma información que hasta ahora, pero con el añadido de los cuartiles.

# Desarrollo de la segunda parte

Para esta segunda parte se procedió a trabajar con las librerías de Python *pandas* y *matplotlib*.

En el primer bloque de la segunda parte manipulamos el dataset con el que veníamos trabajando con el objetivo de "curarlo" o dejar los datos tratados para poder operar con ellos, por ejemplo generando gráficos  para entenderlos de una manera visual, y por tanto más intuitiva. Este tratamiento de los datos siempre se debe realizar, más aún cuando se trabaja con grandes paquetes de datos, los cuales en la mayoría de los casos incluirán inconsistencias.

***Nota:*** *en el caso de nuestro dateset, se presentaron todas las inconsistencias estudiadas en el espacio curricular* ***Procesamiento de Datos****, excepto que no tenía filas duplicadas, por tanto se tuvo que manipular el dataset original para generar algunos duplicados y poder aplicar lo aprendido.*
En el segundo bloque de esta entrega, como se dijo, generamos una serie de gráficos usando librerías específicas, para poder tener un abordaje visual de los datos trabajados, y entenderlos mejor.

# Primer bloque

Para empezar a trabajar con esta segunda parte, comenzaremos por leer nuevamente el dataset para poder partir de cero con lo que íbamos a hacer.

## Datos ausentes

Lo primero que hicimos fue abordar el tratamiento de datos ausentes, lo cual se presenta cuando hay celdas vacías. En este caso las soluciones podían ser de dos tipos:

- rellenar la celda vacía con alguna de las siguientes opciones (entre muchas):
    - usando la media, la mediana o la moda de los valores de esa columna (estas opciones excepto la moda, son válidas si en la columna tratada los valores son de tipo numérico)
    - reemplazar directamente los valores vacíos
    - eliminar la fila completa donde se encontraba la celda vacía. Esta solución se utiliza cuando se dispone de grandes volúmenes de datos, por tanto la eliminación de alguna celda vacía no comprometerá la cantidad o la calidad de la información disponible. Es la opción más sencilla y la que utilizamos a lo último, para tener la oportunidad de generar alguna transformación antes

En nuestro caso todas las columnas estaban completas, excepto:

- **Color** que le faltaban once datos: *para no perder muchos datos, se completaron todos los ausentes con el color “Blanco”*

- **Caja** que le faltaba sólo un dato: *aquí reemplazamos usando la moda*
- **Motor** que le faltaban diez datos: *aquí eliminamos todas las filas*
- **Carrocería** que le faltaba sólo un dato: *completamos de manera directa*

Comenzamos reemplazando el único dato ausente en la columna *Caja* usando la moda: primero calculamos la moda de esta columna y luego rellenamos los valores faltantes en la columna 'Caja' con dicha moda.
Luego reemplazamos el dato ausente en *Carrocería* de manera directa usando el método de Pandas *loc*.
Ahora completamos todos los ausentes en la columna *Color* con el color *Blanco*, que sería lo más neutro, usando el método *.fillna()*.
Una vez que ya agregamos algunos datos en las celdas vacías, procedimos a eliminar todas las filas que tenían algún valor ausente usando el método de Pandas *.dropna()*. A medida que íbamos haciendo todo esto, modificamos el dataframe original. Las celdas que le faltaban algunos datos eran de la columna *Motor*.

## Datos en formato incorrecto

Este quizás fue el punto más complicado que tuvimos al momento de darle tratamiento al dataset con el que trabajamos, ya que implicó pensar una estrategia para dejar los datos limpios.
Lo primero que hicimos fue comprobar el tipo de dato de algunas celdas específicas para averiguar con qué estábamos trabajando.
Haciendo diferentes pruebas vimos que la columna *Motor* era del tipo de dato *object*, que en nuestro caso estaba siendo utilizado para representar cadenas de texto.
Lo que hicimos entonces fue pasar todo directamente a *str* para poder manipular el contenido de las celdas, y luego pasar nuevamente todo a formato de *float* que es el que nos interesaba. Luego buscamos en la columna *Motor* y en todas las celdas donde había un punto modificamos la celda para dejar únicamente el carácter que estaba antes del punto, el punto, y el carácter siguiente. Para lograr esto usamos una combinación de expresiones regulares y la función *.apply()* de Pandas (primero debimos importar el módulo *re* para trabajar con expresiones regulares) y luego definimos la función que realizó la operación.

Pero ahora teníamos un problema: en algunas celdas se encuentra un valor de tipo *char* con un dígito, pero sin el *.0*, por lo tanto necesitábamos una función que nos "empareje" esos datos. Para ello volvimos a usar la función *.apply()* y generamos una función que tomaba un texto como entrada y con *.isdigit()* verificaba si el texto contenía solo dígitos y si era así, devolvía el texto original con más *.0* agregado al final, y si no, devolvía el texto original sin modificaciones.
Pero hasta aquí seguíamos teniendo los valores dentro de las celdas de la columna *Motor* como cadenas de caracteres.
Para pasar todo a *float* primero extrajimos sólo los números y luego los convertimos a punto flotante, otra vez utilizando las expresiones regulares. Por último con *astype(float, errors='ignore')* convertimos la columna a *float*, y de esta manera obtuvimos la columna *Motor* sólo con valores numéricos de punto flotante.

## Datos erróneos

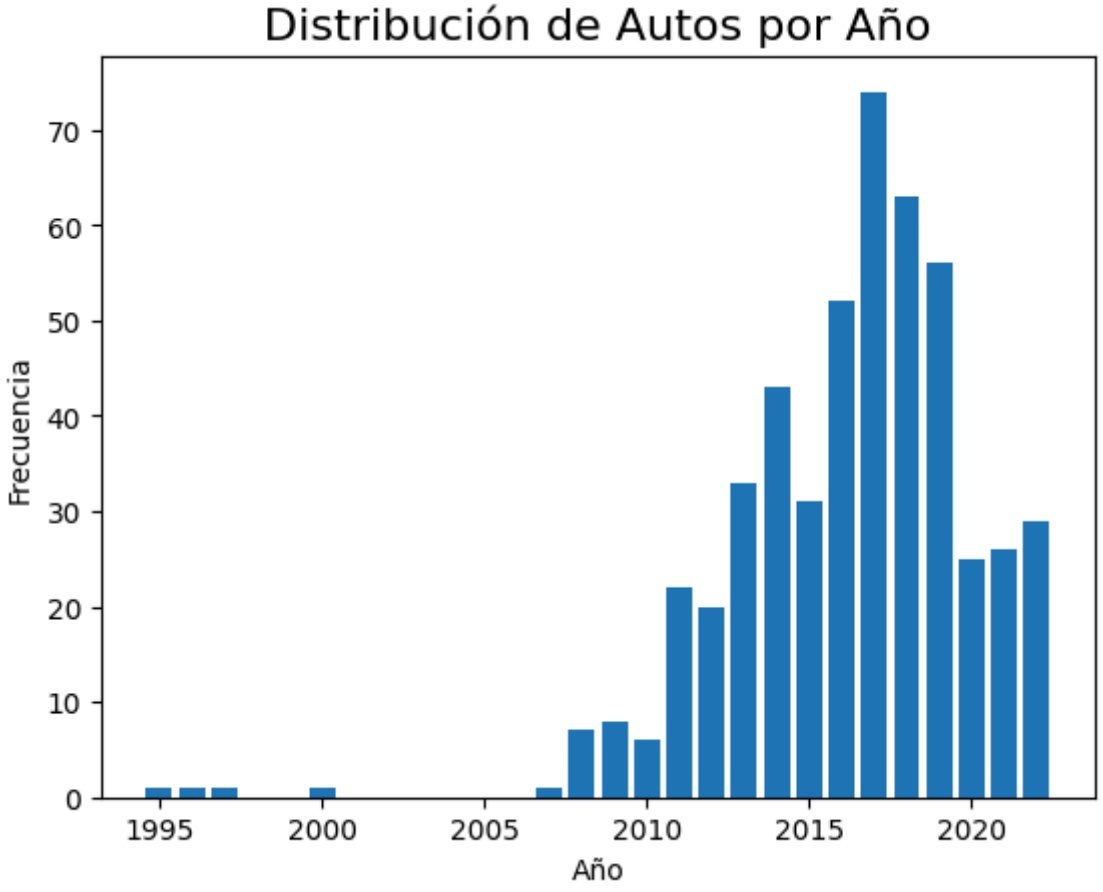
Este apartado y el siguiente fueron los más sencillos de realizar.
Lo que hicimos fue centrarnos en la columna *Año*, en la cual existían datos (de tipo entero o *int64*) que habían sido cargados erróneamente, con ceros de más.
Lo que hicimos fue sencillo, comprobamos si algún valor es mayor que "3000" y procedimos entonces a quitarle ceros hasta dejarlo de cuatro dígitos. Esto lo logramos tomando un valor de año como entrada y si el año era mayor que 3000, entramos en un bucle *while* que se ejecutaba mientras el año sea mayor que 9999 (es decir, tenga más de cuatro dígitos). Dentro del bucle, con *año* hicimos una división entera entre 10, eliminando el último dígito del año, hasta que finalmente obtuvimos el año corregido.

## Datos duplicados

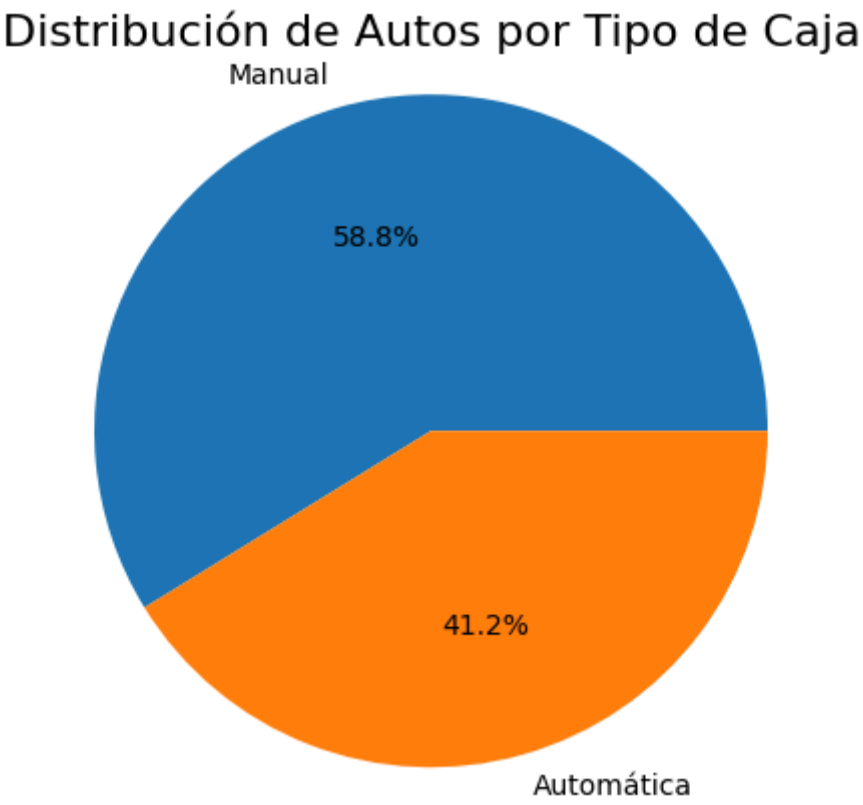
Para el apartado de trabajar con datos duplicados, lo primero que hicimos fue buscar si había datos duplicados en el dataset, y lo que más nos interesaba, cuántos datos duplicados había (en el caso afirmativo).
Con las filas duplicadas no es mucho lo que se podía hacer, simplemente procedimos a eliminarlas.

## Segundo bloque

En esta sección se desplegaron algunos gráficos usando la librería MatPlotLib... realmente lo que se puede hacer aquí sólo está limitado por la imaginación. Empezamos por lo más sencillo, algunos gráficos de barra tomando como dato las columnas *Año*, *Color* y *Carrocería*. Por ejemplo:




Para las columnas donde hay almacenados datos bivariantes generamos gráficos de torta (pie chart), ya que representan mejor esta información. Nosotros tenemos datos bivariantes en las columnas *Caja* y *Moneda*.

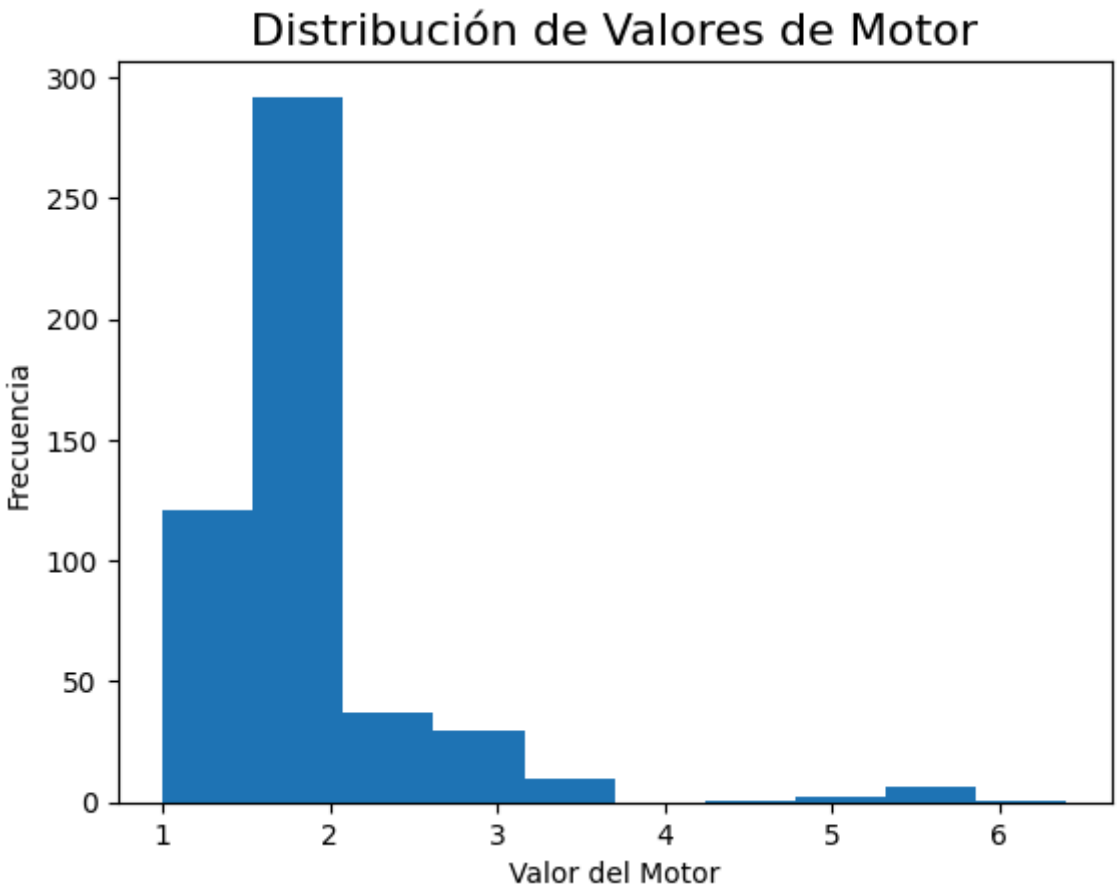



*Fue sorprendente descubrir la gran proporción de autos que todavía usan caja de cambios manual.*

Una opción de gráfico muy interesante para mostrar es el gráfico de "Treemap", también llamado gráfico de mosaico o gráfico de área rectangular. Es interesante sobre todo cuando se tienen valores muy variados, porque se pueden representar visualmente a todos de una sola vez.
Hay que tener en cuenta que no se puede generar un gráfico de Treemap con *pyplot* directamente ya que esta librería no tiene una función específica para crear gráficos de Treemap. Sin embargo, se puede utilizar la biblioteca *squarify* para crear un gráfico de Treemap en Python. *squarify* proporciona una función llamada *squarify.plot* que se puede utilizar para crear Treemaps.

Por último, para determinar si los datos son simétricos o asimétricos, utilizamos gráficos de distribución de frecuencias, en nuestro caso histogramas. Este gráfico muestra la frecuencia de los valores de una variable en función de su distancia a la media de la distribución. Si la distribución es simétrica, el gráfico tendrá una forma

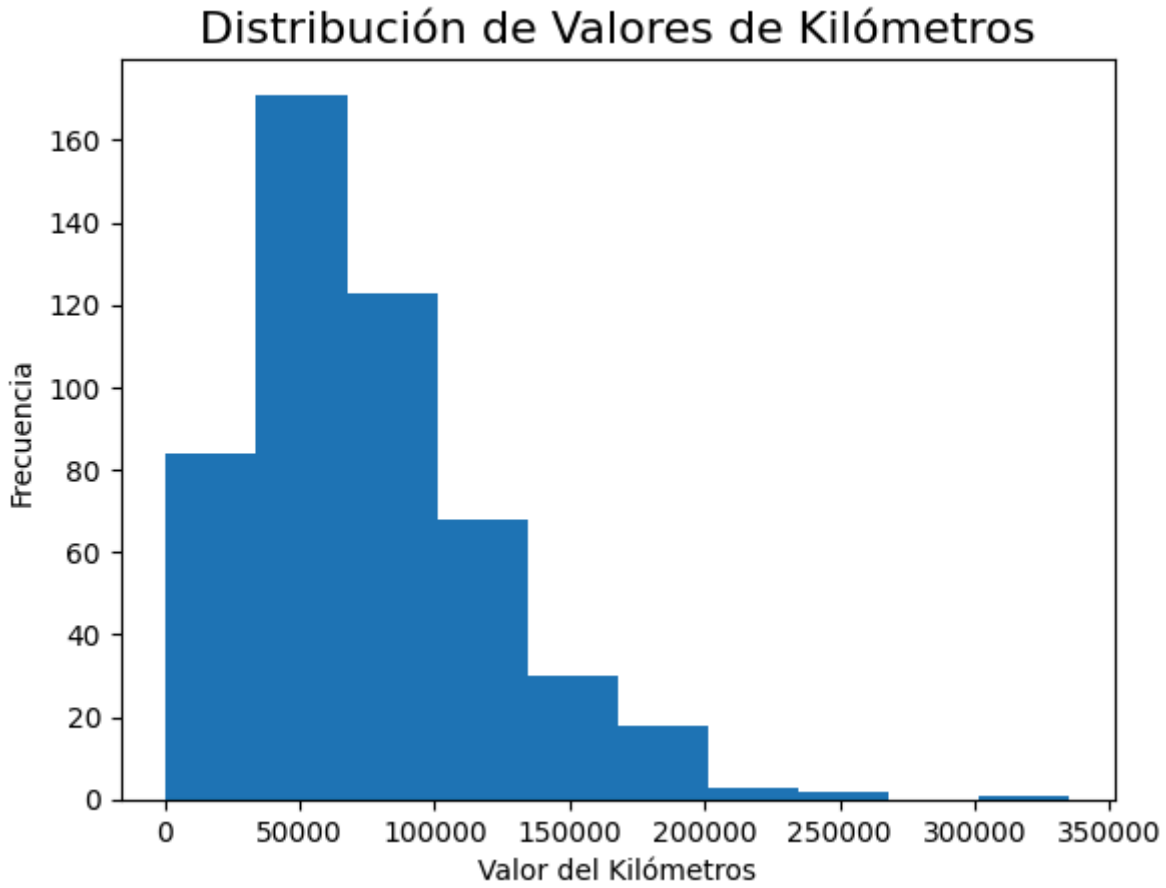
de campana, con la mayoría de los valores concentrados en torno a la media y una cola que se extiende hacia ambos lados de manera simétrica.
En el caso de una distribución asimétrica, el gráfico mostrará una cola más larga hacia un lado que hacia el otro, lo que indica que los valores están más dispersos en una dirección específica. La asimetría puede ser positiva (a la derecha) o negativa (a la izquierda).
En este proyecto analizamos la distribución de frecuencias del tamaño del *Motor* y la cantidad de *Kilómetros* que tiene el vehículo.




**Aclaración:** en un histograma, los *bins* (o "contenedores") son los intervalos en los que se divide el rango total de valores de la variable numérica que se está analizando. Cada bin representa un rango específico de valores, y la altura de la barra correspondiente a ese bin indica la frecuencia (cantidad de observaciones) que caen dentro de ese rango.
Por ejemplo, si se está analizando la columna Motor y se establece *bins=10*, el histograma dividirá el rango de valores de *Motor* en 10 intervalos iguales. Cada

barra del histograma mostrará cuántos autos tienen un valor de *Motor* dentro de cada uno de esos intervalos.

*Como se puede ver, en nuestro conjunto de valores para el tamaño del `Motor`, tenemos una distribución asimétrica positiva.*




*Y la misma forma de distribución asimétrica que antes, pero ahora se puede observar con más claridad.*