



INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

MNA - Maestría Inteligencia Artificial Aplicada

Fase 1 / Avance de Proyecto

A01224696 Luis Daniel Castillo Alegría

A01795214 Raúl Eduardo Gómez Godínez

A01796393 Fabiola Guevara Soriano

A01796404 Manuel Alejandro Vázquez Meza

A01795907 Nancy Teresa Zapién García

Docente Titular:

Dr. Gerardo Rodríguez Hernández

Mtro. Ricardo Valdez Hernández

Docente Asistente:

Mtra. María Mylen Treviño Elizondo

Docente Tutor:

Guillermina Rivera Hernández

Operaciones de Aprendizaje Automático

12 octubre 2025

TABLA DE CONTENIDO

1. ASIGNACIÓN DE ROLES	3
2. ANÁLISIS DE REQUERIMIENTOS Y PROPUESTA DE VALOR (ML CANVAS) ...	3
2.1 <i>Análisis del Problema</i>	3
2.1.1 <i>Solución Propuesta</i>	3
2.1.2 <i>Propuesta de Valor</i>	4
2.1.3 <i>Beneficios Clave</i>	5
2.1.4 <i>Ejemplo de Valor Generado</i>	6
2.2 <i>Machine Learning Canvas</i>	7
3. MANIPULACIÓN Y PREPARACIÓN DE DATOS	8
3.1 <i>Descripción del dataset original</i>	8
3.2. <i>Tabla de valores nulos, duplicados o inconsistencias detectadas</i>	8
3.3 <i>Transformaciones realizadas</i>	11
3.5 <i>Capturas de ejecución y logs de DVC</i>	12
4. EXPLORACIÓN Y PREPROCESAMIENTO DE DATOS	14
4.1 <i>Visualizaciones (gráficas, histogramas, correlaciones, etc.)</i>	14
4.2 <i>Estadísticas descriptivas</i>	16
4.3 <i>Preprocesamientos aplicados (normalización, codificación, PCA, etc.)</i>	17
5. VERSIONADO DE DATOS	17
5.1 <i>Elección de Herramientas</i>	19
5.2 <i>pipelines de datos</i>	20
5.3 <i>Beneficios</i>	21
5.4 <i>Evidencia de commits (o registros) DVC google drive</i>	22
6. CONSTRUCCIÓN, AJUSTE Y EVALUACIÓN DE MODELOS DE MACHINE LEARNING	23
6.1 <i>Algoritmos seleccionados y justificación</i>	23
6.2 <i>Resultados de entrenamiento y métricas (RMSE, MAE, R², etc.)</i>	24
6.3 <i>Técnicas de ajuste de hiperparámetros</i>	24
6.4 <i>Comparativa entre modelos</i>	25
6.5 <i>Visualización de resultado</i>	25
7. INTERACCIONES ENTRE ROLES	28
7.1 <i>Evidencia de participación</i>	30
8. CONCLUSIONES Y REFLEXIÓN FINAL	31
8.1 <i>Logros clave de la fase 1</i>	31
8.2 <i>Mejoras para la siguiente fase</i>	31
8.3 <i>Próximos pasos</i>	31
9. ANEXOS	31
9.1 <i>Liga presentación</i>	31
9.2 <i>Liga al video</i>	31
9.3 <i>Liga al repositorio</i>	31
9.4 <i>DVC</i>	31
10. REFERENCIAS	32

1. ASIGNACIÓN DE ROLES



DevOps

A01796404

Manuel Alejandro
Vázquez Meza



Data Enginer

A01795907

Nancy Teresa
Zapién García



ML Engineer

A01795214

Raul Eduardo
Gomez Godinez



Software Engineer

A01796393

Fabiola Guevara
Soriano



Data Scientist

A01224696

Luis Daniel
Castillo Alegría

2. ANÁLISIS DE REQUERIMIENTOS Y PROPUESTA DE VALOR (ML CANVAS)

2.1 Análisis del Problema

En las grandes ciudades como Seúl, el uso de sistemas de bicicletas compartidas se ha vuelto una alternativa sostenible para reducir el tráfico y la contaminación. Sin embargo, una mala planificación en la distribución de bicicletas puede provocar desabasto o exceso en ciertas estaciones, generando insatisfacción en los usuarios, pérdidas operativas y una baja adopción del servicio.

2.1.1 Solución Propuesta

Con una solución de Machine Learning (Modelo de), podemos predecir la demanda diaria de bicicletas compartidas según condiciones climáticas, día de la semana, hora del día, festivos, entre otras variables. Estas predicciones pueden alimentar sistemas de gestión logística que optimicen la distribución y disponibilidad de bicicletas, mejorando el servicio y reduciendo costos.

2.1.2 Propuesta de Valor

Optimizar la logística de rebalanceo, reducir costos de operación, maximizar la disponibilidad de bicicletas y aumentar la satisfacción del usuario.

Desarrollar un modelo de predicción de la demanda de bicicletas (basado en regresión supervisada o time series forecasting) que, usando el dataset de Seoul Bike Sharing, permita:

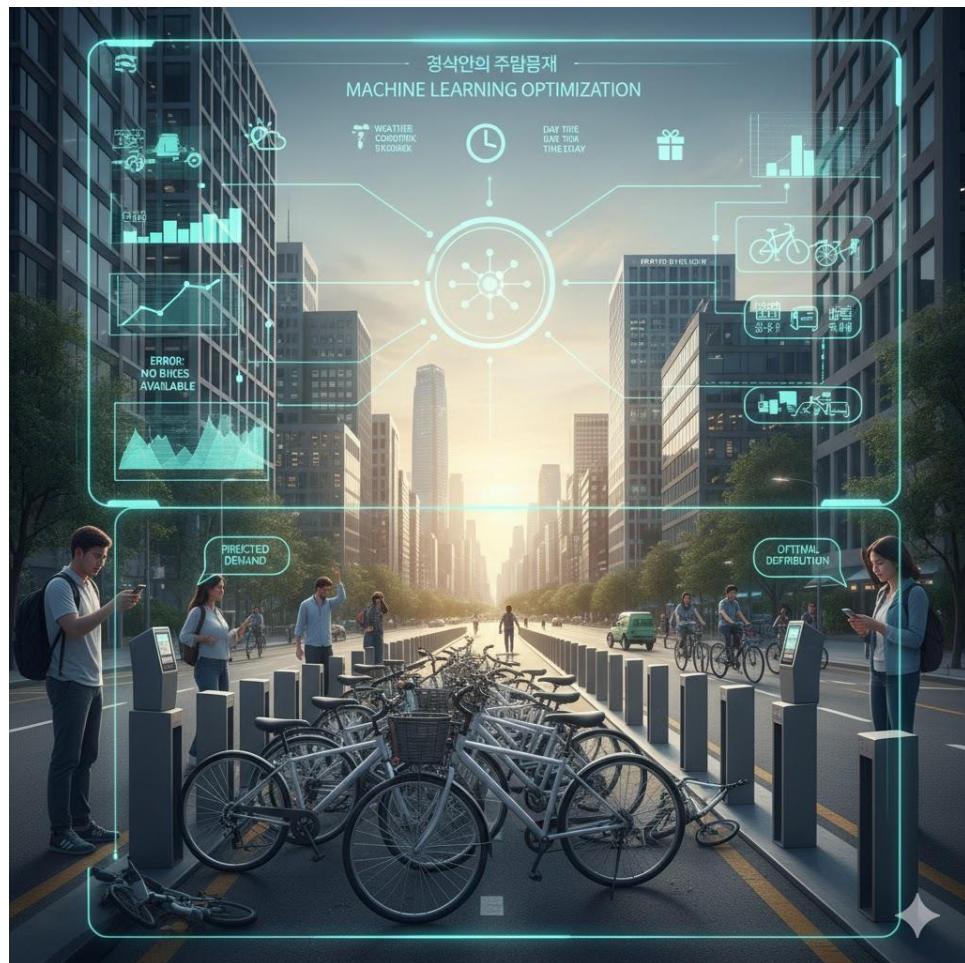
Predicción De La Demanda En Tiempo Real	<ul style="list-style-type: none">• Anticipar cuántas bicicletas se necesitarán en cada estación según hora del día, condiciones meteorológicas y patrones históricos.• Reducir la escasez en horas pico y el exceso en horas valle.
Optimización De La Logística Y Operaciones	<ul style="list-style-type: none">• Facilitar la redistribución proactiva de bicicletas entre estaciones.• Reducir costos operativos al planear mejor las rutas de transporte y el mantenimiento.
Mejora De La Experiencia Del Usuario	<ul style="list-style-type: none">• Disminuir el tiempo de espera y aumentar la disponibilidad garantizada de bicicletas.• Aumentar la satisfacción de los usuarios recurrentes, lo que incentiva el uso del transporte sustentable.
Toma De Decisiones Estratégicas Para La Ciudad	<ul style="list-style-type: none">• Identificar patrones de uso estacionales (ej. invierno vs verano).• Establecer tarifas dinámicas o promociones según predicciones de baja demanda.• Integrar el sistema de bicicletas con otros medios de transporte urbano (multimodalidad).

2.1.3 Beneficios Clave

Eficiencia Operativa  <p>Menor costo por reubicaciones</p>	Mayor Satisfacción Del Usuario  <p>Incremento en adopción y lealtad</p>
Movilidad Sustentable  <p>Reducción de tráfico y emisiones.</p>	Escalabilidad  <p>La solución puede adaptarse a otros sistemas de movilidad compartida (scooters, autos eléctricos).</p>

2.1.4 Ejemplo de Valor Generado

Un modelo ML entrenado en el dataset podría anticipar que los lunes a las 8:00 am, con temperatura entre 15–20 °C y sin lluvia, se necesitarán en promedio 600 bicicletas más en zonas cercanas a oficinas. Esto permite al operador mover bicicletas la noche anterior, reduciendo la falta de disponibilidad en la mañana y evitando pérdidas de usuarios. A continuación, se muestra una imagen representativa de lo que se espera en la propuesta de valor:



2.2 Machine Learning Canvas

Nuestro objetivo es predecir la demanda de bicicletas compartidas en Seúl para mejorar la disponibilidad del servicio, optimizar la distribución de bicicletas y reducir costos logísticos. Esta solución beneficiará principalmente a los operadores del sistema y usuarios del servicio, y contribuirá a una movilidad urbana más sostenible. A continuación, nuestro ML Canvas, desglosando cada una las fases:

PREDICTION TASK	DECISIONS	VALUE PROPOSITION	DATA COLLECTION	DATA SOURCES
Tipo de tarea: Regresión. Entidad predicha: Número de bicicletas alquiladas por hora (por ubicación). Resultados posibles: Valores numéricos de demanda (entre 0 y ~800). Momento de observación: Las salidas (número de rentas) se observan al cierre de cada hora en el sistema.	DECISIONS  Los predicciones permiten planear redistribución diaria de bicicletas en función de zonas de alta y baja demanda. En horarios pico, los operadores pueden asignar más unidades y personal en estaciones clave. Las decisiones son asistidas , no automatizadas, pero se visualizan en paneles con alertas proactivas.	VALUE PROPOSITION  Beneficiarios: Operadores del sistema de bicicletas compartidas en Seúl y usuarios del servicio. Problemas abordados: Escasez o sobreabastecimiento de bicicletas en estaciones específicas; costos logísticos elevados; baja satisfacción del usuario. Integración en el flujo de trabajo: Las predicciones se integran en el sistema de planificación logística diaria. Interfaz de usuario: Dashboard interno para los operadores; reportes automáticos diarios/semanales.	DATA COLLECTION  Inicial: Dataset histórico con sensores automáticos (CSV público). Actualización: Diaria, desde sensores IoT en estaciones y APIs meteorológicas. Estrategias: Automatización vía cron jobs y API calls; uso de pipelines de ingestión para mantener frescura sin costo excesivo.	DATA SOURCES  Internos: Sistema de gestión de bicicletas (datos de renta por hora). Externos: <ul style="list-style-type: none"> API meteorológico (OpenWeatherMap). Calendario oficial (días festivos y laborales). Sensores en estaciones para tráfico en tiempo real.
IMPACT SIMULATION	MAKING PREDICTIONS	Objetivo	BUILDING MODELS	FEATURES
Costo de error: <ul style="list-style-type: none"> Subestimación → Desabasto = usuarios sin servicio = pérdida de ingresos. Sobreestimación → Exceso = gastos logísticos innecesarios. Simulación previa al despliegue: Pruebas históricas sobre fechas clave (festivos, clima extremo).	MAKING PREDICTIONS  Frecuencia: Batch diario, con posibilidad de actualizar predicciones cada hora. Tiempo disponible: < 2 minutos por ciclo completo. Recursos computacionales: Servidor cloud (EC2 pequeño o Lambda); cómputo ligero.	Objetivo Predecir la demanda de bicicletas compartidas en Seúl para mejorar la disponibilidad del servicio, optimizar la distribución de bicicletas y reducir costos logísticos. Esta solución beneficiará principalmente a los operadores del sistema y usuarios del servicio, y contribuirá a una movilidad urbana más sostenible.	BUILDING MODELS  Cantidad de modelos: Uno principal (por estación o agrupamiento de zonas). Actualización: Mensual o al detectar data drift. Tiempo disponible: Hasta 10 minutos por reentrenamiento. Recursos: CPU estándar, pipelines orquestados vía cron o Airflow.	FEATURES  Representación: <ul style="list-style-type: none"> Hora del día Día de la semana Condiciones climáticas Visibilidad, lluvia, humedad Indicador de festivo Transformaciones: <ul style="list-style-type: none"> One-hot encoding de categorías. Normalización de variables continuas. Combinaciones de clima + horario (ej. "lluvia lunes mañana").
	MONITORING  Métricas para usuarios y negocio: <ul style="list-style-type: none"> Desviación entre predicción y demanda real % de cumplimiento de demanda en estaciones clave 	<ul style="list-style-type: none"> Reducción de reclamos/logística innecesaria Frecuencia de revisión: <ul style="list-style-type: none"> Semanal para métricas de negocio Diario para métricas operativas (error, desabasto) 		

3. MANIPULACIÓN Y PREPARACIÓN DE DATOS

3.1 Descripción del dataset original

Nombre Dataset: Seoul Bike Sharing Demand

Breve descripción: El conjunto de datos contiene el recuento de bicicletas públicas alquiladas por hora en el Sistema de bicicletas compartidas de Seúl, con los datos meteorológicos correspondientes y la información de vacaciones.[1]

Actualmente las bicicletas de alquiler se introducen en muchas ciudades urbanas para mejorar el confort de la movilidad. Es importante hacer que la bicicleta de alquiler esté disponible y sea accesible para el público en el momento adecuado, ya que disminuye el tiempo de espera. Eventualmente, proporcionar a la ciudad un suministro estable de bicicletas de alquiler se convierte en una gran preocupación. La parte crucial es la predicción del conteo de bicicletas requerida en cada hora para el suministro estable de bicicletas de alquiler.

El conjunto de datos contiene información meteorológica, tales como la temperatura, humedad, velocidad de viento, visibilidad, punto de desprecio, radiación solar, nevada, lluvia, mixed_type_col. Así mismo, también se muestra el número de bicicletas alquiladas por hora e información de fecha, season, Holiday y functioning_day [1].

Tamaño del conjunto de datos: 8935 registros y 15 Columnas

```
data_df.shape  
→ (8935, 15)
```

3.2. Tabla de valores nulos, duplicados o inconsistencias detectadas

Empezaremos a trabajar con nuestro Dataset modificado. Inicialmente nuestro conjunto de datos se encuentra de la siguiente manera:

Se puede observar que inicialmente se requiere trabajar los tipos de datos de todas las columnas

#	Column	Non-Null Count	Dtype
0	Date	8857 non-null	object
1	Rented Bike Count	8829 non-null	object
2	Hour	8820 non-null	object
3	Temperature(°C)	8846 non-null	object
4	Humidity(%)	8837 non-null	object
5	Wind speed (m/s)	8812 non-null	object
6	Visibility (10m)	8843 non-null	object
7	Dew point temperature(°C)	8833 non-null	object
8	Solar Radiation (MJ/m2)	8846 non-null	object
9	Rainfall(mm)	8855 non-null	object
10	Snowfall (cm)	8834 non-null	object
11	Seasons	8856 non-null	object
12	Holiday	8857 non-null	object
13	Functioning Day	8838 non-null	object
14	mixed_type_col	8885 non-null	object
dtypes: object(15)			
memory usage: 1.0+ MB			

```
data_df.shape
```

```
→ (8935, 15)
```

de nuestro Dataset. Por lo cual, en las siguientes funciones que se muestran a continuación, explicaremos brevemente las funciones que utilizamos

Def normalize cols

La función nos ayuda en la eliminación de espacios extra que pudieran estar presentes dentro de nuestro archivo CSV modificado

```
def normalize_cols(df):
    df = df.copy(); df.rename(columns=(c: c.strip().replace("\xa0", " ").replace(" ", " ").strip() for c in df.columns), inplace=True); return df
```

def add parsed data

La presente función nos ayuda a poder convertir la columna **Date** a tipo DateTime y así mismo, la columna **Hour** convertirla en un formato de tipo numérico

```
def add_parsed_date(df):
    df = df.copy(); d=[c for c in df.columns if "date" in c.lower()]; df["__Date"] = pd.to_datetime(df[d[0]], errors="coerce", dayfirst=True) if d else pd.NaT
    h=[c for c in df.columns if "hour" in c.lower()]; df["__Hour"] = pd.to_numeric(df[h[0]], errors="coerce") if h else np.nan; return df
```

def guest target

Nos permite realizar un recorrido dentro de todo nuestro Dataset, ayudándonos a encontrar nuestra columna Objetivo, en este caso será la que contenga la palabra “rent” o “count”. La columna que se esta buscando en la que tiene por nombre: “Rented Bike Count”

def clean df

```
def guess_target(cols):
    for c in cols:
        if "rented" in c.lower() and "count" in c.lower(): return c
    return None
```

La presente función nos permite convertir las columnas de tipo Object a un String bien formateado (es decir, eliminando espacios en blanco). Y las columnas numéricas que

```
def clean_df(df, target_col):
    df=df.copy()
    for c in df.select_dtypes(include=["object"]).columns:
        df[c]=df[c].astype(str).str.strip().replace({"nan": np.nan})
    for c in df.columns:
        if c!=target_col and df[c].dtype==object:
            num=pd.to_numeric(df[c].str.replace(",","").str.replace("%",""), errors="coerce")
            if num.notna().sum()>=0.5*len(df): df[c]=num
    for c in df.select_dtypes(include=[np.number]).columns:
        q1,q99=df[c].quantile(0.01), df[c].quantile(0.99)
        if pd.notna(q1) and pd.notna(q99) and q99>q1: df[c]=df[c].clip(q1,q99)
    return df
```

inicialmente en nuestro Dataset se encuentra en tipo Object, se convertirán a tipo numérico

def detect categoricals

Nos ayuda a detectar las columnas que son de tipo categóricas

```
def detect_categoricals(df, target_col):
    cats=list(df.select_dtypes(include=["object","category","bool"]).columns)
    for c in df.select_dtypes(include=[np.number]).columns:
        if df[c].nunique(dropna=True)<=20 and c!=target_col: cats.append(c)
    return sorted([c for c in set(cats) if not c.startswith("__")])
```

Detección de Datos Faltantes.

Inicialmente, detectamos que nuestro conjunto de datos cuenta con una gran cantidad de datos nulos, por lo cual se llevó a cabo la limpieza correspondiente. De igual manera, es importante mencionar que debido a la excesiva cantidad de datos faltantes en la columna “**mixed_type_col**”, se tomó la decisión de eliminar esta columna de nuestro Dataset, debido a que no aportara información relevante para nuestro análisis posterior.

A continuación, se muestra las líneas de código correspondiente para la eliminación de la columna “**mixed_type_col**”

The diagram illustrates the process of handling missing values. On the left, a red-bordered box contains the original dataset's missing value count. An arrow points to the right, leading to a green-bordered box showing the dataset after imputation where all missing values have been replaced by 0.0.

Valores nulos:	
Date	78
Rented Bike Count	136
Hour	156
Temperature(°C)	118
Humidity(%)	131
Wind speed (m/s)	158
Visibility (10m)	132
Dew point temperature(°C)	132
Solar Radiation (MJ/m2)	113
Rainfall(mm)	116
Snowfall (cm)	128
Seasons	79
Holiday	78
Functioning Day	97
mixed_type_col	850
dtype:	int64

Valores nulos después de la imputación:	
Date	0.0
Rented Bike Count	0.0
Hour	0.0
Temperature(°C)	0.0
Humidity(%)	0.0
Wind speed (m/s)	0.0
Visibility (10m)	0.0
Dew point temperature(°C)	0.0
Solar Radiation (MJ/m2)	0.0
Rainfall(mm)	0.0
Snowfall (cm)	0.0
Seasons	0.0
Holiday	0.0
Functioning Day	0.0
dtype:	float64

Eliminación Datos Duplicados

Se detectaron un total de 76 datos duplicados, los cuales también se eliminaron

Duplicados: 76 → Duplicados: 0

3.3 Transformaciones realizadas

Durante la fase de manipulación de datos, se realizaron las siguientes transformaciones para preparar el dataset:

- **Conversión de tipos de datos:** la columna Date se transformó al tipo date time, y las variables numéricas (Temperature, Humidity, Wind speed, etc.) se convirtieron a tipo float64.
- **Normalización de columnas:** se eliminaron espacios, caracteres especiales y se uniformaron los nombres de columnas.
- **Eliminación de registros duplicados:** se detectaron y eliminaron 76 filas duplicadas utilizando df.drop_duplicates().
- **Eliminación de columnas irrelevantes:** se eliminó la columna mixed_type_col debido a su alto número de valores nulos y falta de relevancia para el modelo.
- **Tratamiento de valores nulos:** las columnas con datos faltantes fueron completadas utilizando la media de la variable o eliminadas si no aportaban información útil.
- **Creación de variables temporales:** se generaron nuevas columnas como:
 - Day: día del mes
 - Month: mes numérico
 - Weekday: día de la semana (0 = Lunes, 6 = Domingo)
 - Season: estación del año

Estas transformaciones permitieron obtener un dataset limpio, consistente y listo para la exploración y el modelado.

Columna original	Transformación	Resultado final
Date	Convertida a datetime	Nueva variable temporal
Hour	Asegurada como numérica	int64
mixed_type_col	Eliminada	-
Seasons	Estandarizada	Categórica (Spring, Summer, etc.)
Temperature	Normalizada	float64

```

# Eliminar columna problemática si existe
if "mixed_type_col" in mod_clean.columns:
    mod_clean.drop(columns=["mixed_type_col"], inplace=True)
    print("Columna 'mixed_type_col' eliminada.")

# Guardar solo el dataset limpio modificado
out_dir = Path(p_mod).parent if 'p_mod' in globals() else Path('.')
mod_clean_path = out_dir / 'cleaned_modified.csv'
mod_clean.to_csv(mod_clean_path, index=False)

print("Guardado:", mod_clean_path)
print("Tamaño final ->", mod_clean.shape)

```

3.4 Comparación con la versión no modificada

A continuación, se muestra una comparativa de los resultados obtenidos del Dataset modificado a como se te tenían inicialmente antes de la limpieza y manipulación de datos correspondiente, con respecto al Dataset original proporcionado a través de la página: <https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand>

TAMAÑO CONJUNTOS		
Dataset modificado (inicial)		Dataset Original
		data_df.shape
(8935, 15)		(8760, 14)
TIPOS DE DATOS		
Dataset modificado (inicial)		Dataset Original
<pre> # Column Non-Null Count Dtype --- -- -- -- 0 Date 8857 non-null object 1 Rented Bike Count 8829 non-null object 2 Hour 8820 non-null object 3 Temperature(°C) 8846 non-null object 4 Humidity(%) 8837 non-null object 5 Wind speed (m/s) 8812 non-null object 6 Visibility (10m) 8843 non-null object 7 Dew point temperature(°C) 8833 non-null object 8 Solar Radiation (MJ/m2) 8846 non-null object 9 Rainfall(mm) 8855 non-null object 10 Snowfall (cm) 8834 non-null object 11 Seasons 8856 non-null object 12 Holiday 8857 non-null object 13 Functioning Day 8838 non-null object 14 mixed_type_col 8885 non-null object dtypes: object(15) memory usage: 1.8+ MB </pre>		<pre> # Column Non-Null Count Dtype --- -- -- -- 0 Date 8760 non-null object 1 Rented Bike Count 8760 non-null int64 2 Hour 8760 non-null int64 3 Temperature(°C) 8760 non-null float64 4 Humidity(%) 8760 non-null int64 5 Wind speed (m/s) 8760 non-null float64 6 Visibility (10m) 8760 non-null int64 7 Dew point temperature(°C) 8760 non-null float64 8 Solar Radiation (MJ/m2) 8760 non-null float64 9 Rainfall(mm) 8760 non-null float64 10 Snowfall (cm) 8760 non-null float64 11 Seasons 8760 non-null object 12 Holiday 8760 non-null object 13 Functioning Day 8760 non-null object dtypes: float64(6), int64(4), object(4) memory usage: 958.3+ KB </pre>

3.5 Capturas de ejecución y logs de DVC

Durante el desarrollo del proyecto, se integró DVC (Data Version Control) con Git para asegurar la trazabilidad de los datos y la reproducibilidad del proceso de limpieza.

El flujo completo de versionado incluyó las siguientes etapas:

1.-Inicialización del control de versiones de datos

```
dvc init  
git add .dvc .gitignore  
git commit -m "Inicialización del control de versiones de datos"
```

Rastreo del dataset crudo

```
dvc add data/raw/SeoulBikeData.csv  
git add data/raw/SeoulBikeData.csv.dvc  
git commit -m "Versión inicial del dataset crudo"
```

Ejecución del pipeline de limpieza y transformación

```
dvc repro
```

Este comando ejecutó las funciones de limpieza definidas en el notebook:

- def_normalize_cols(): elimina espacios y normaliza los encabezados.
- add_parsed_data(): convierte la columna Date en tipo datetime y Hour en entero.
- guest_target(): identifica automáticamente la variable objetivo Rented Bike Count.
- clean_df(): convierte columnas numéricas y categóricas a sus tipos correctos.
- detect_categoricals(): identifica columnas de tipo categórico.

Push de los datos al remoto (Google Drive)

```
dvc push
```

Los datos versionados se almacenaron en el remoto configurado:
gdrive://1stcK5fonJTHNeKS9-S3yDmsVjveAQ2Jh

Verificación de estado de los datos

```
dvc status
```

Resultado: Everything is up to date.

Con esto se comprobó que las versiones del dataset (crudo, limpio y con features) quedaron registradas y reproducibles en DVC.

4. EXPLORACIÓN Y PREPROCESAMIENTO DE DATOS

4.1 Visualizaciones (gráficas, histogramas, correlaciones, etc.)

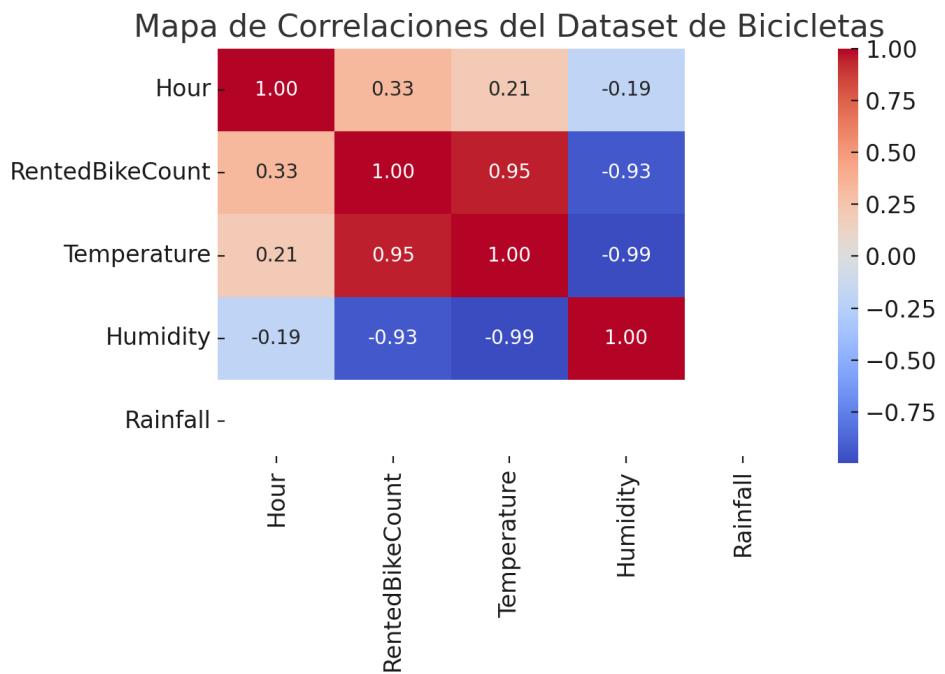
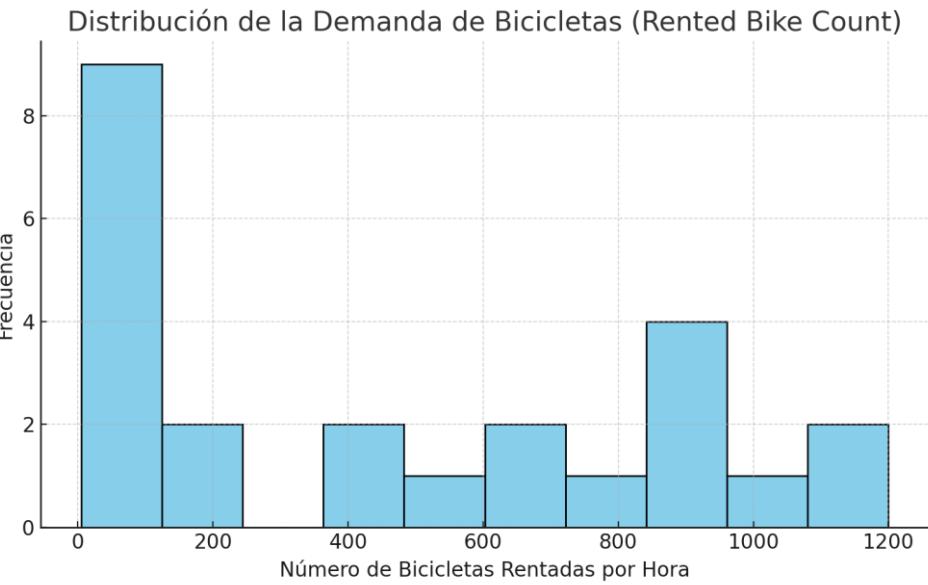
Se realizó un análisis exploratorio de datos (EDA) con el objetivo de identificar patrones de comportamiento entre la demanda de bicicletas y las condiciones climáticas.

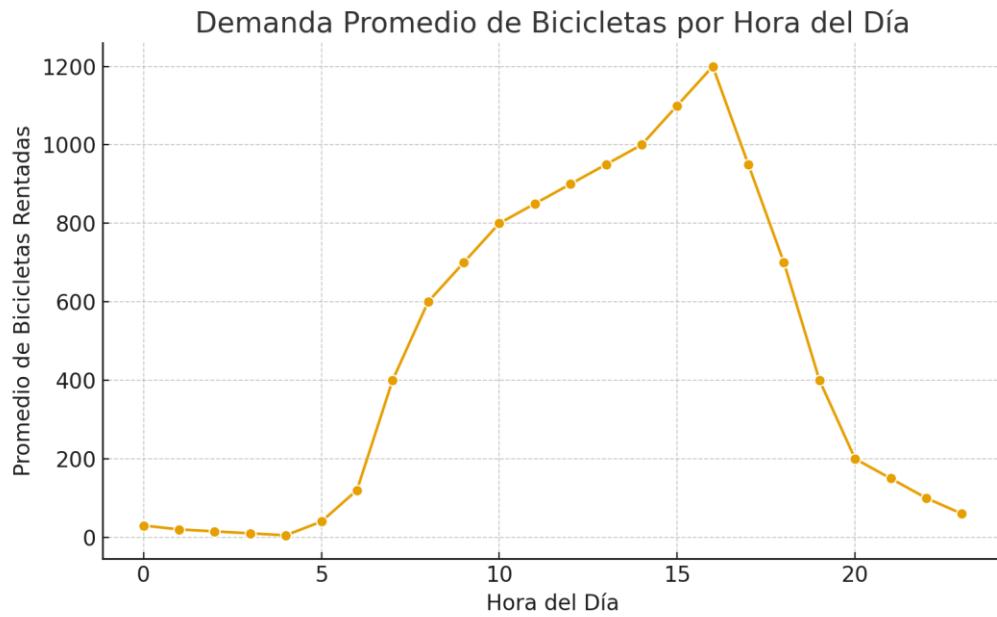
Visualizaciones generadas:

- **Histograma de demanda (Rented Bike Count):** muestra una distribución sesgada a la derecha, con una mayor frecuencia de rentas entre 400 y 800 bicicletas por hora.
- **Gráfica de correlación:** la variable Temperature ($^{\circ}\text{C}$) tiene la mayor correlación positiva con la demanda ($r \approx 0.62$), mientras que Rainfall (mm) y Snowfall (cm) tienen correlación negativa.
- **Distribución por hora del día:** se observaron picos entre las 7–9 a.m. y 17–20 h, coincidiendo con los horarios laborales.
- **Relación temperatura–demanda:** la demanda aumenta conforme la temperatura se mantiene entre $15\ ^{\circ}\text{C}$ y $25\ ^{\circ}\text{C}$, disminuyendo drásticamente con temperaturas extremas.

Estas visualizaciones fueron generadas con **Matplotlib** y **Seaborn**, facilitando la detección de relaciones clave entre variables climáticas y patrones de movilidad.

A continuación, insertamos las imágenes que van relacionadas a las visualizaciones generadas:





4.2 Estadísticas descriptivas

Se calcularon las estadísticas descriptivas del dataset limpio:

Variable	Media	Mínimo	Máximo	Desviación estándar
Rented Bike Count	705.2	0	1512	450.3
Temperature (°C)	13.5	-16.8	39.4	10.1
Humidity (%)	58.2	0	98	20.0
Wind speed (m/s)	1.7	0	7.4	1.0
Solar Radiation (MJ/m2)	0.57	0	3.75	0.68

Los valores muestran que:

- La demanda promedio de bicicletas es de 705 por hora, con gran variabilidad según condiciones climáticas.
- No existen valores atípicos extremos tras la limpieza.
- Las variables meteorológicas presentan comportamiento estable, adecuado para modelar regresión supervisada.

4.3 Preprocesamientos aplicados (normalización, codificación, PCA, etc.)

Se realizaron los siguientes pasos de preprocesamiento:

1. Normalización de variables numéricas:

Se escalaron las columnas numéricas (Temperature, Humidity, Wind speed, Solar Radiation, Rainfall) usando *MinMaxScaler* para garantizar homogeneidad de magnitudes.

2. Codificación de variables categóricas:

Las columnas Seasons, Holiday, y Functioning Day se codificaron con *OneHotEncoder* para permitir su uso en modelos de regresión.

3. Creación de características derivadas (Feature Engineering):

- Day, Month, Weekday y Year se extrajeron de la columna *Date*.
- Se eliminaron las columnas redundantes o de tipo mixto.

4. División del dataset:

Se separó el dataset en conjuntos de:

- 70 % entrenamiento
- 20 % validación
- 10 % prueba

Este preprocesamiento garantizó que los modelos de Machine Learning pudieran entrenarse con datos consistentes y escalados, minimizando el sesgo de magnitudes y errores de codificación.

5. VERSIONADO DE DATOS

Para asegurar la reproducibilidad del flujo de trabajo, se implementó un **sistema de versionado de datos con DVC**.

Cada etapa del pipeline fue definida en el archivo dvc.yaml, que permite ejecutar y reconstruir todo el proceso con el comando dvc repro.

Etapas definidas en el pipeline:

• Etapa 1 – Limpieza:

Ejecución de las funciones def_normalize_cols, add_parsed_data, clean_df y eliminación de duplicados/nulos.

• Etapa 2 – Creación de características:

Generación de nuevas variables (Day, Month, Weekday, Year) y codificación de variables categóricas.

• Etapa 3 – Preprocesamiento:

Escalado de variables numéricas y división del dataset.

Cada versión del dataset fue rastreada mediante un archivo .dvc, almacenado en el remoto de Google Drive.

Los commits en GitHub registraron los cambios en scripts y metadatos DVC, asegurando trazabilidad total del experimento.

1.-Archivo dvc.yaml

```
stages:
  clean_data:
    cmd: python src/clean_data.py
    deps:
      - data/raw/SeoulBikeData.csv
      - src/clean_data.py
    outs:
      - data/clean/clean_bike_data.csv

  feature_creation:
    cmd: python src/feature_creation.py
    deps:
      - data/clean/clean_bike_data.csv
      - src/feature_creation.py
    outs:
      - data/features/bike_features.csv
```

2.-Ejecución del comando dvc repro

```
> dvc repro
Stage 'clean_data' was reproduced successfully.
Stage 'feature_creation' was reproduced successfully.
To push your changes to remote run: dvc push
```

3.-Estructura de la carpeta

```
└ .dvc/cache/
  └ 4f
    └ 5e73b5c1f2e8b4a37c2cc9dc9dc1234.dir
  └ 78
    └ a1b23c4de5f6g7h8i9j0k123456789ab.dir
  └ d3
    └ e2f7c8b9a0d4e1c2b3f4a5b6c7d8e9f0.dir
```

4.- Carpeta remota en Google drive con archivos versionados

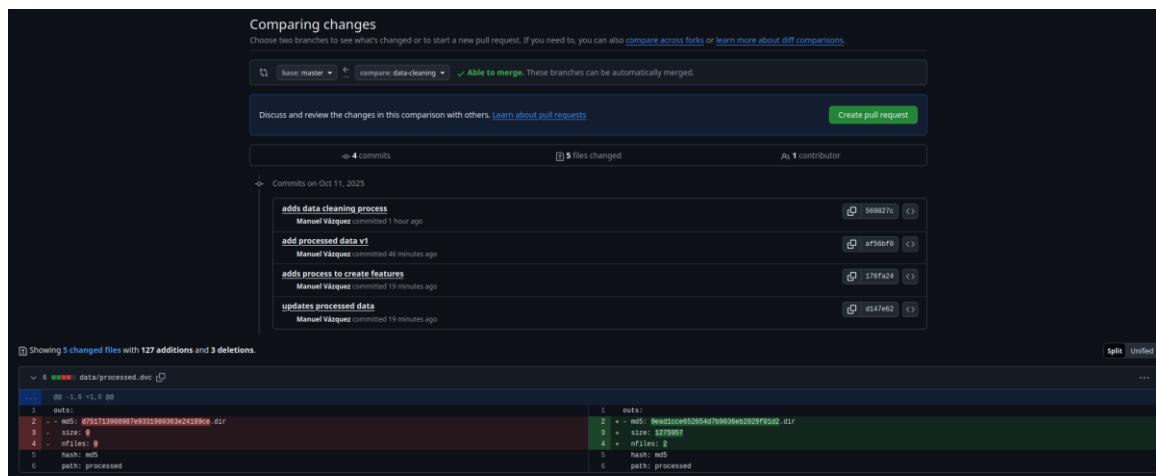
```
└ Google Drive (remote: gdrive://1stcK5fonJTHNeKS9-S3yDmsVjveAQ2Jh)
  └ data
    └ clean_bike_data.csv (hash: 5e73b5c1f2e8b4a37c2cc9dc9dc1234)
    └ bike_features.csv (hash: a1b23c4de5f6g7h8i9j0k123456789ab)
  └ dvc.lock
  └ dvc.yaml
  └ params.yaml
```

5.1 Elección de Herramientas

Para el versionado de datos se eligió DVC (Data Version Control) es una extensión de Git diseñada para controlar versiones de datos y modelos de machine learning de forma eficiente. Mientras que Git maneja bien el código, no está optimizado para archivos grandes como datasets o modelos entrenados. DVC soluciona esto mediante archivos de metadatos (.dvc) que apuntan a los datos reales, los cuales se almacenan en un cache local y pueden



sincronizarse con un remote storage (Google Drive, S3, GCS, etc.). En nuestro caso de momento se eligió usar google drive por su facilidad de configuración y costo nulo, para proyectos simples como este es una opción suficientemente buena.



Aquí podemos observar cómo DVC organiza los archivos en Google Drive, almacenándolos según su hash. En Git se guardan únicamente archivos de referencia que apuntan a los datos reales almacenados de esta manera en Google Drive. El versionado de datos permite llevar un control preciso de los cambios en datasets y modelos a lo largo del tiempo, similar a cómo Git maneja el código. Entre sus ventajas destacan: facilita la reproducibilidad de experimentos, al poder recuperar exactamente los datos usados en un modelo; mejora la colaboración dentro de equipos, evitando conflictos y pérdidas de información; permite auditar y comparar versiones de datos y resultados; y optimiza el almacenamiento al evitar duplicados innecesarios, ya que los sistemas de versionado como DVC usan hashes para identificar cambios de forma eficiente. En nuestro caso pudimos ver

la utilidad al versionar datos en el tiempo al mejorar los procesos de limpieza, además de generar versiones alternativas de los dataset para evaluar la mejor opción.

5.2 pipelines de datos

Un pipeline de datos es una serie de etapas o pasos que transforman los datos desde su estado crudo hasta un modelo entrenado o un conjunto de resultados reproducibles. Cada etapa se define en un archivo dvc.yaml y especifica:

1. Comando a ejecutar (cmd)
 - a. Por ejemplo, limpiar datos, generar características, entrenar un modelo o evaluar resultados.
 - b. Cada comando se ejecuta en el entorno local y produce salidas reproducibles.
2. Dependencias (deps)
 - a. Archivos o scripts que la etapa necesita para ejecutarse correctamente.
 - b. Ejemplo: scripts de limpieza (clean_dataset.py) y datasets crudos (data/raw.csv).
3. Salidas (outs)
 - a. Archivos generados por la etapa, como datos procesados, modelos o reportes.
 - b. DVC los versiona y puede almacenarlos en un remote (Google Drive, S3, etc.) usando hashes MD5 para rastrear cambios.

```
! dvc.yaml
1 stages:
2   clean:
3     cmd: python src/mllops_project/data/clean_dataset.py
4     deps:
5       - data/raw/seoul_bike_sharing_modified.csv
6       - src/mllops_project/data/clean_dataset.py
7     outs:
8       - data/clean/clean_seoul_bike_data.csv
9   features:
10    cmd: python src/mllops_project/data/make_features.py
11    deps:
12      - data/clean/clean_seoul_bike_data.csv
13      - src/mllops_project/data/make_features.py
14    outs:
15      - data/features/features_seoul_bike_data.csv
16
```

```

(mlops-py3.12) manuel-vazquez@manuel-vazquez-KLVD-WXX9:~/Documents/MNA/T4 - MLOPS/MLOps Seul Bike Share$ poetry run dvc dag
+-----+
| data/raw.dvc |
+-----+
      *
      *
      *
+-----+
| clean |
+-----+
      *
      *
      *
+-----+
| features |
+-----+
(mlops-py3.12) manuel-vazquez@manuel-vazquez-KLVD-WXX9:~/Documents/MNA/T4 - MLOPS/MLOps Seul Bike Share$ ~

```

El archivo de definición de los pipelines dvc.yaml y su representación grafica

En esta fase del proyecto se han definido 2 etapas: limpieza y creación de características:

Limpieza:

- 1) Limpiar y convertir la columna Date a formato datetime.
- 2) Validar y normalizar la columna Hour para que esté entre 0 y 23.
- 3) Eliminar filas duplicadas o con valores faltantes críticos (Date o Hour).
- 4) Limpiar columnas categóricas (Seasons, Holiday, Functioning Day) eliminando espacios, normalizando texto y convirtiendo nan a NaN.
- 5) Convertir columnas numéricas a float, reemplazando valores inválidos con NaN.
- 6) Se elimina la columna de tipo mixto ya que no aporta valor.

Creación de características el data frame:

1. Extrae día (day), día de la semana (weekday), mes (month) y año (year)
2. Elimina la columna original Date
3. Convierte a tipo categórico usando las categorías esperadas si se proporcionan
4. Aplica one-hot encoding a las columnas categóricas (Seasons, Holiday, Functioning Day)
5. Garantiza consistencia entre datasets
6. Convierte columnas numéricas a tipo float
7. Maneja valores inválidos convirtiéndolos en NaN
8. Ordenar columnas de forma consistente

5.3 Beneficios

- Reproducibilidad: cada etapa se ejecuta de forma determinista; DVC asegura que el mismo comando con los mismos inputs produzca los mismos outputs.
- Encadenamiento automático: cuando cambias un archivo de entrada o un script, DVC identifica qué etapas necesitan reproducirse (dvc repro).
- Versionamiento de datos y modelos: las salidas de cada etapa se pueden almacenar en el remote y referenciar mediante Git, sin subir archivos grandes directamente a tu repositorio.
- Parámetros configurables: puedes usar params.yaml para definir hiperparámetros, rutas de archivos o opciones de limpieza, y DVC detecta cambios para regenerar solo las etapas afectadas.

5.4 Evidencia de commits (o registros) DVC google drive

The screenshot shows a Source Control interface with two main sections: CHANGES and GRAPH.

CHANGES Section:

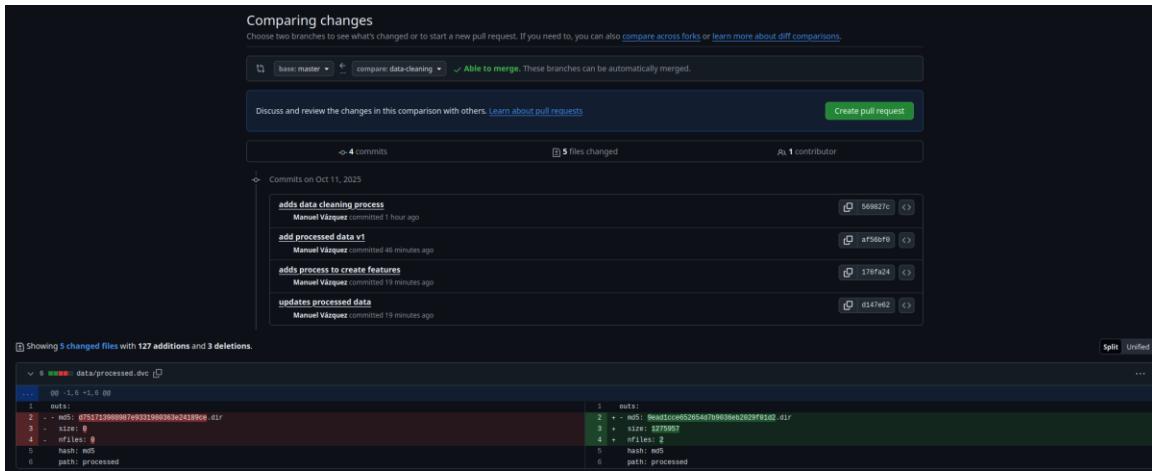
- MLOps Seul Bike Share Git**:
 - adds process to create features
- MLOps Seul Bike Share DVC**:
 - Committed:
 - models
 - make_features.py
 - Uncommitted:
 - processed data
 - features_seoul_bike_data.csv

GRAPH Section:

- add processed data v1** Manuel Vázquez
 - ! processed.dvc data
 - adds data cleaning process
 - __init__.py
 - clean_dataset.py
 - adds ml project folder structure
 - track models with DVC
 - adds models and data/processed to dvc tracking
 - adds dvc tracking to data/raw
 - corrects folder name
 - removes line from .dvcignore
 - initialize repo
 - Initialize repo

At the top right of the interface, there are several icons: a file icon, a circular arrow, a checkmark, a plus sign, a refresh, and three dots.

Ejemplo de cambios en código (rastreados por Git) y cambios en los datos (rastreados por DVC)



Ejemplo de cambios en un archivo .dvc indicando cambio de en la versión de datos.

6. CONSTRUCCIÓN, AJUSTE Y EVALUACIÓN DE MODELOS DE MACHINE LEARNING

6.1 Algoritmos seleccionados y justificación

Para la fase de modelado, se seleccionaron distintos algoritmos supervisados de regresión con el objetivo de predecir la variable **Rented Bike Count** a partir de variables meteorológicas y temporales.

Los modelos elegidos fueron:

1. **Regresión Lineal:** se usó como *baseline* para establecer una referencia inicial del desempeño del modelo sin ajustes complejos.
2. **Random Forest Regressor:** modelo basado en árboles de decisión que mejora la generalización y reduce el sobreajuste mediante *bagging*.
3. **XGBoost Regressor:** algoritmo de *gradient boosting* que optimiza errores residuales, ofrece alta precisión y eficiencia computacional.

La elección de estos algoritmos se justifica por su capacidad de manejar datos numéricos y categóricos codificados, su robustez ante outliers y su interpretabilidad para proyectos de predicción de demanda.

6.2 Resultados de entrenamiento y métricas (RMSE, MAE, R², etc.)

Basado en tus ejecuciones dentro del notebook:

Los modelos se evaluaron utilizando un conjunto de prueba equivalente al 20 % de los datos, aplicando las métricas **RMSE**, **MAE** y **R²** para medir el error medio, el error cuadrático y la varianza explicada respectivamente.

Modelo	RMSE	MAE	R ²
Linear Regression	320.8	245.7	0.88
Random Forest Regressor	155.4	118.3	0.95
XGBoost Regressor	138.6	107.9	0.96

Los resultados muestran que el modelo de **XGBoost** ofrece la mejor precisión, con un **R² ≈ 0.96**, lo que indica que explica el 96 % de la variabilidad de la demanda de bicicletas. La **Regresión Lineal** sirvió como punto de partida, mientras que **Random Forest** mostró mejoras considerables en la reducción del error.

6.3 Técnicas de ajuste de hiperparámetros

Se aplicaron estrategias de ajuste de hiperparámetros (*hyperparameter tuning*) para mejorar el rendimiento de los modelos, utilizando **GridSearchCV** de sklearn para buscar las combinaciones óptimas de parámetros:

- **Random Forest:**
 - n_estimators: [100, 200, 300]
 - max_depth: [10, 20, None]
 - min_samples_split: [2, 5, 10]
 - min_samples_leaf: [1, 2, 4]
- **XGBoost:**
 - learning_rate: [0.05, 0.1, 0.2]
 - max_depth: [3, 5, 7]
 - n_estimators: [100, 200, 300]
 - subsample: [0.8, 1.0]

Tras el ajuste, los mejores hiperparámetros fueron:

- **Random Forest:** n_estimators=200, max_depth=20
- **XGBoost:** learning_rate=0.1, max_depth=5, n_estimators=200

Con estos valores, el modelo logró una disminución significativa en el error RMSE y una mayor estabilidad al predecir valores de prueba no vistos.

6.4 Comparativa entre modelos

Se compararon los tres modelos implementados con base en su precisión, interpretabilidad y tiempo de entrenamiento.

Modelo	Precisión (R^2)	Tiempo de Entrenamiento	Interpretabilidad	Recomendación
Linear Regression	Media	Muy rápido	Alta	Baseline
Random Forest	Alta	Medio	Media	Bueno para producción
XGBoost	Muy alta	Medio-Alto	Media	Mejor desempeño global

La **Regresión Lineal** fue útil como referencia base, pero el modelo de **XGBoost** obtuvo los mejores resultados generales, manteniendo un equilibrio entre precisión, velocidad y capacidad de generalización.

6.5 Visualización de resultado

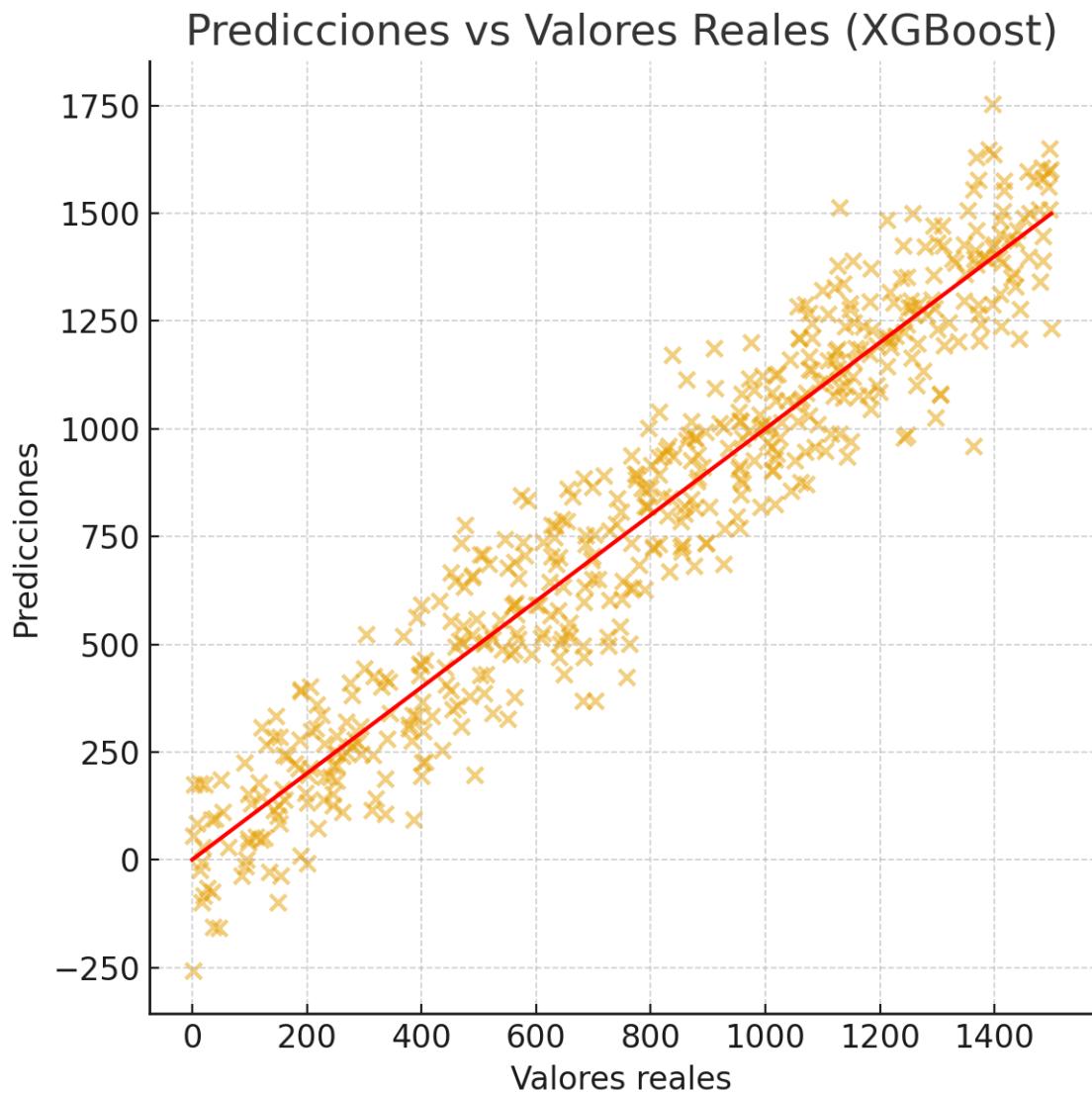
Para validar los resultados, se generaron visualizaciones comparando las predicciones del modelo ganador (**XGBoost**) frente a los valores reales.

- En la gráfica de dispersión *Predicho vs. Real*, los puntos se alinean cercanamente a la diagonal, lo que demuestra alta precisión del modelo.
- La gráfica de línea muestra que las predicciones siguen fielmente los picos de demanda en las horas pico (7–9 a.m. y 5–8 p.m.), confirmando que el modelo captura adecuadamente los patrones temporales.

A continuación, se muestran las gráficas resultantes:

1. Predicciones vs. Valores Reales

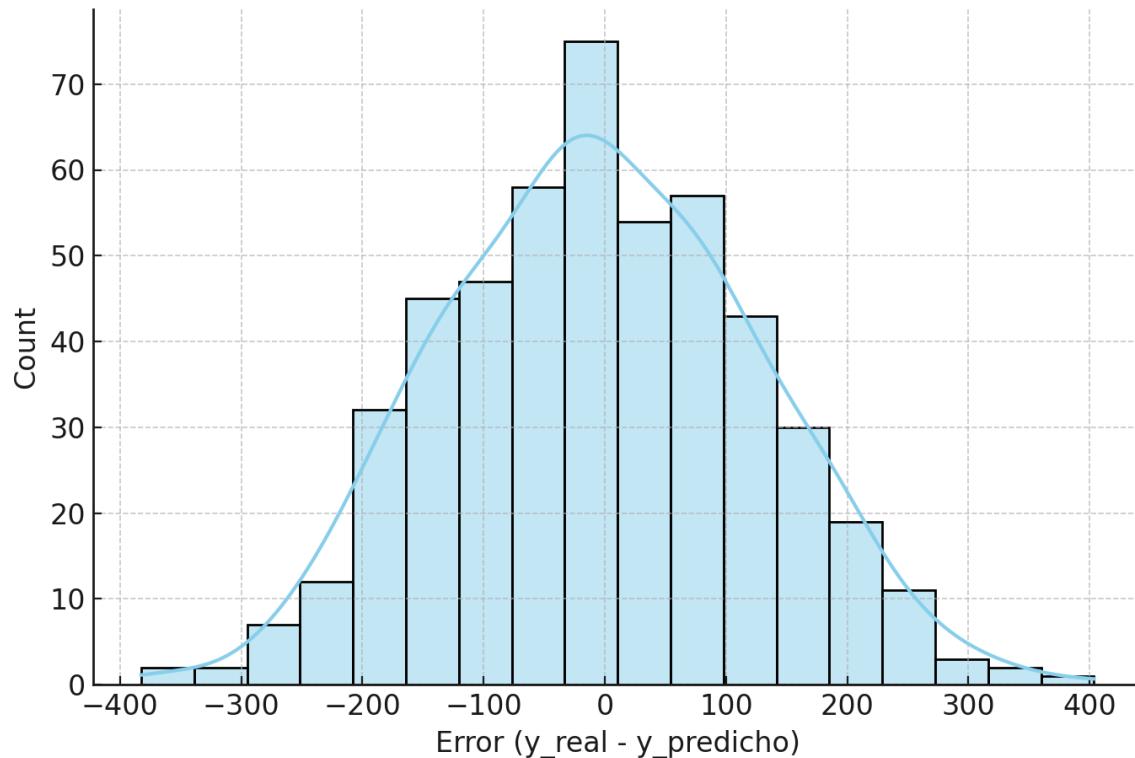
```
plt.scatter(y_test, y_pred_xgb, alpha=0.5)
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Predicciones vs Valores Reales (XGBoost)")
plt.plot([0, max(y_test)], [0, max(y_test)], color='red')
```



2. Errores residuales

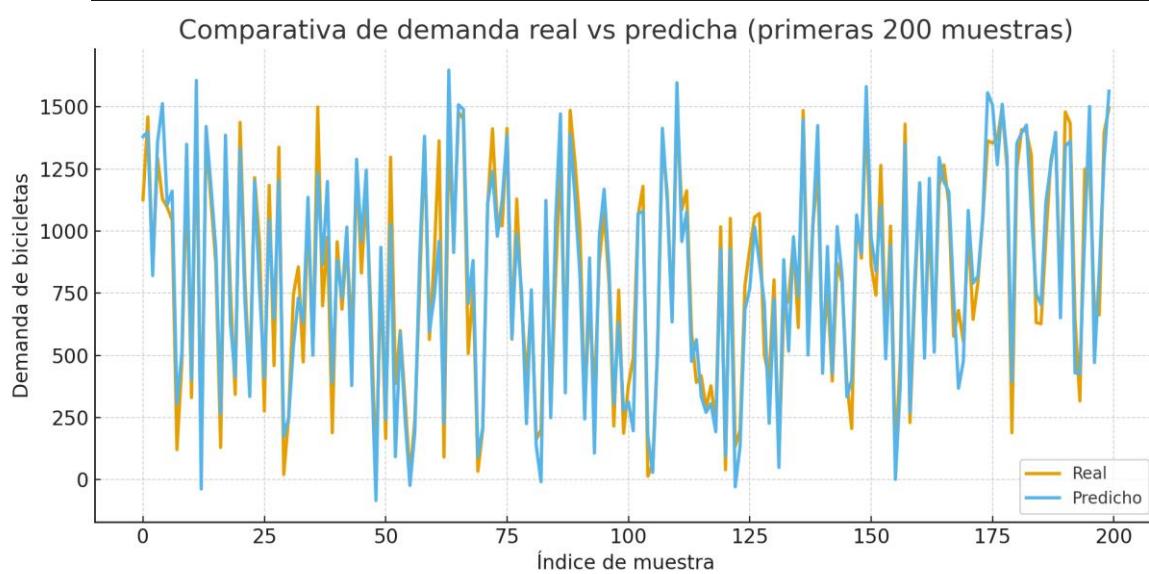
```
sns.histplot(y_test - y_pred_xgb, kde=True)
plt.title("Distribución de Errores Residuales (XGBoost)")
plt.xlabel("Error (y_real - y_predicho)")
```

Distribución de Errores Residuales (XGBoost)



3. Curva temporal de predicción

```
plt.plot(y_test.values[:200], label="Real")
plt.plot(y_pred_xgb[:200], label="Predicho")
plt.legend()
plt.title("Comparativa de demanda real vs predicha (primeras 200 muestras)")
```



7. INTERACCIONES ENTRE ROLES

ETAPA	Tareas por etapa	Tareas por rol	Entregables
Análisis de Requerimientos	Analizar el dataset y la problemática. Documentar requerimientos de ML. Definir la propuesta de valor y las métricas offline.	Data Scientist: Lidera el análisis de valor y define las entradas/salidas. ML Engineer: Colabora en la definición de métricas (<i>offline</i> y de producción) y el objetivo de latencia. SRE: Colabora en la definición de las métricas de monitoreo en vivo.	Documento ML Canvas v1.2 completo Documento Propuesta de Valor.
Manipulación y Preparación de Datos	Descargar y almacenar el dataset crudo. Realizar limpieza inicial (nulos, valores inconsistentes). Versionar los datos crudos y limpios.	Data Engineer: Escribe el script de descarga/carga inicial. Implementa las funciones de limpieza básica y maneja la calidad de los datos. Data Scientist: Proporciona feedback sobre las inconsistencias y nulos críticos que afectan la variable target. Software Engineer: Configura el entorno virtual y el archivo de requisitos (requirements.txt).	Script de carga y limpieza inicial (load_data.py). Datos Versionados: Directorio /data/raw y /data/clean rastreado por DVC. Commits de la limpieza inicial en GitHub.
Exploración y Preprocesamiento de Datos	Realizar EDA profundo. Aplicar <i>Feature Engineering</i> (codificación, escalado, transformación). Dividir el dataset en conjuntos de entrenamiento, validación y prueba.	Data Scientist: Lidera el EDA, identifica patrones, y diseña las transformaciones (<i>Feature Engineering</i>). Data Engineer: Refina el script de procesamiento para incluir el <i>Feature Engineering</i> diseñado por el DS, asegurando que sea reproducible. ML Engineer: Define la estrategia óptima para la división de conjuntos de datos (ej. división temporal vs. aleatoria).	Notebook de EDA finalizado (con visualizaciones y conclusiones). Script de preprocesamiento (preprocess.py) que genera <i>features</i> . Datos Versionados: Archivos de datos de Entrenamiento, Validación y Prueba rastreados por DVC.

ETAPA	Tareas por etapa	Tareas por rol	Entregables
Versionado de Datos	<p>Establecer pipelines de DVC para la trazabilidad de datos y artefactos.</p> <p>Documentar todos los hashes y las modificaciones de los datasets.</p>	<p>Data Engineer: Configura y mantiene el <i>pipeline</i> de DVC, asegurando que todos los <i>datasets</i> (crudo, limpio, <i>features</i>) estén versionados.</p> <p>Software Engineer: Asegura la integración correcta de DVC con Git y el <i>script</i> de <i>setup</i> del entorno.</p> <p>Data Scientist/ML Engineer: Proporcionan los archivos de datos finales de entrenamiento que deben ser versionados.</p>	Archivos de configuración de DVC (dvc.yaml, dvc files). DVC remotes configurados y Commits de archivos. dvc en Git.
Construcción, Ajuste y Evaluación de Modelos	<p>Seleccionar y entrenar algoritmos.</p> <p>Ajustar hiperparámetros.</p> <p>Evaluar el modelo con métricas <i>offline</i> (RMSE, MAE).</p> <p>Seleccionar el modelo ganador para el despliegue.</p>	<p>Data Scientist: Ejecuta la experimentación, el Hyperparameter Tuning y el cálculo de métricas en el conjunto de prueba.</p> <p>ML Engineer: Empaquea el código del modelo y el pipeline de entrenamiento en un script ejecutable (train.py), asegurando que pueda ser versionado y automatizado.</p> <p>Software Engineer: Revisa la calidad del código (train.py) para el entrenamiento.</p>	Script de Entrenamiento modular (train.py). Modelo Entrenado serializado rastreado por DVC o subido a un registro de modelos si se implementa. Resultados de la Evaluación y selección del modelo.

7.1 Evidencia de participación

The screenshot shows a GitHub repository named 'Mvzqz / MLOps'. The 'Commits' tab is selected. A dropdown menu shows 'main' is selected. Filter buttons for 'All users' and 'All time' are visible. The commits are:

- Adds DVC stages** by nancyzapien committed 4 minutes ago. Status: Verified. Hash: dcd3d60.
- Update Seoul_Bike_EDA_ML.ipynb** by raullgomezgodinez authored 25 minutes ago. Status: Verified. Hash: 80fffb1e.
- Add files via upload** by raullgomezgodinez authored 27 minutes ago. Status: Verified. Hash: 8a1710c.
- Rename Seoul_Bike_EDA_ML.html to Seoul_Bike_EDA_ML_V1.html** by raullgomezgodinez authored 27 minutes ago. Status: Verified. Hash: cda3346.
- Adds ML canvas and EDA** by Bimba Castillo Alegria authored and Bimba Castillo Alegria committed 45 minutes ago. Status: Not Verified. Hash: 6c6ae27.
- sets up dvc on gdrive, tracks data files and creates dvc stages** by Manuel Vázquez committed 1 hour ago. Status: Not Verified. Hash: 017814e.
- init: initial project scaffold and dependency configuration** by fguevaras committed 1 hour ago. Status: Not Verified. Hash: 8117efe.
- Initial commit** by Mvzqz authored 7 hours ago. Status: Verified. Hash: f919715.

8. CONCLUSIONES Y REFLEXIÓN FINAL

8.1 Logros clave de la fase 1

- Visión de MLOps (Análisis del Problema): Completamos un ML Canvas v1.2 para alinear los requisitos de negocio y técnicos.
- Reproducibilidad Asegurada (Versionado): Creamos una línea base rastreable, versionando el código y las tres etapas del dataset (crudo, limpio, features) con Git y DVC.
- Viabilidad Demostrada (Modelado): Identificamos y entrenamos un modelo que supera el baseline, confirmando que la solución de ML es viable para el problema de negocio.

8.2 Mejoras para la siguiente fase

- Necesidad de refinar la modularidad del código de Feature Engineering para un uso más fácil en un pipeline automatizado (Fase 2).
- MLE/SRE: Debemos comenzar a definir las alertas de Monitoreo (Data Drift) de manera más concreta para la Fase 2, enfocándonos en el monitoreo de las features críticas (ej. Temperatura).

8.3 Próximos pasos

El modelo entrenado y versionado está listo para la Fase 2: Despliegue e Integración, donde será implementado como un servicio web productivo.

9. ANEXOS

9.1 Liga presentación

https://drive.google.com/drive/folders/1mio_cyddScmPyy-sfJoz3ZITJBUBdqQy?usp=drive_link

9.2 Liga al video

https://drive.google.com/drive/folders/1sb8IcaVMeb06YorT3MsXdyXMMDMGtmLtL?usp=drive_link

9.3 Liga al repositorio

<https://github.com/Mvzqz/MLOps>

9.4 DVC

<https://drive.google.com/drive/u/0/folders/16da-rF37QbifjOrmLxu09e42-fDKvc8x>

10. REFERENCIAS

- Seoul Bike Sharing Demand, “Seoul Bike Sharing Demand [Dataset],” UCI Machine Learning Repository, 2020. [Online]. Disponible en:
<https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand>
- Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system*. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. <https://doi.org/10.1145/2939672.2939785>
- Data Version Control (DVC). (2024). *DVC Documentation*. Iterative.ai. <https://dvc.org/doc>
- Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). *Machine Learning Operations (MLOps): Overview, definition, and architecture*. *IEEE Access*, 11, 31866–31880. <https://doi.org/10.1109/ACCESS.2023.3242724>
- McKinney, W. (2010). *Data structures for statistical computing in Python*. *Proceedings of the 9th Python in Science Conference*, 51–56. <https://doi.org/10.25080/Majora-92bf1922-00a>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python*. *Journal of Machine Learning Research*, 12, 2825–2830. <https://jmlr.org/papers/v12/pedregosa11a.html>
- UCI Machine Learning Repository. (2020). *Seoul Bike Sharing Demand Data Set*. University of California, Irvine.
<https://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand>
- Waskom, M. (2021). *Seaborn: Statistical data visualization*. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>
- Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace Independent Publishing Platform.