



**Universidad
del Valle de México**

LAUREATE INTERNATIONAL UNIVERSITIES®

UNIVERSIDAD DEL VALLE DE MÉXICO

CAMPUS ONLINE

MATERIA: TALLER DE FORTALECIMIENTO AL EGRESO II - ICD

ACTIVIDAD: PROYECTO INTEGRADOR E1

ALUMNO: ANTONIO RENDÓN CÓRDOVA

PROFESOR: ELIZABETH CATAÑO VARGAS

FECHA DE ENTREGA: 26 MAYO 2025

Introducción:

Esta actividad consiste en aplicar los conocimientos adquiridos a lo largo del curso y retomar lo aprendido en cada una de las actividades realizadas, lo que garantiza la transversalidad de los contenidos revisados para fortalecer el desarrollo de competencias y lograr el fin de formación planteado.

Objetivo:

El objetivo del Proyecto Integrador es poner en práctica las técnicas y procedimientos de la ciencia de datos para la toma de decisiones y la extracción de conocimiento a partir de fuentes de datos diversas. Esto a través de la aplicación de la Minería de Datos y técnicas de Descubrimiento de Conocimiento en Bases de Datos (Knowledge Discovery in Databases- KDD).

1.1 Revisión de los materiales publicados de bases de datos

Se realizó una revisión de distintas plataformas de acceso a datos abiertos con el fin de identificar datasets relevantes para el proyecto. Entre las fuentes consultadas se encuentran:

- **Kaggle:** <https://www.kaggle.com/>
- **Plataforma Nacional de Datos Abiertos (México):** <https://www.datos.gob.mx/>
- **Google Dataset Search:** <https://datasetsearch.research.google.com/>
- **Datahub:** <https://www.datahub.com>

1.2 Selección de la base de datos

Durante la exploración en plataformas como Kaggle y Google Dataset Search, se identificó un conjunto de datos de interés titulado **Yellow Taxi Trip Data**, el cual destaca por su volumen considerable de registros y la relevancia del tema para el análisis. Inicialmente, este dataset contenía información correspondiente a los años 2015 y 2023, lo que motivó la búsqueda de versiones más recientes.

Como resultado, se localizó una fuente oficial más actualizada directamente en el portal de [Taxi & Limousine Commission](#) de la ciudad de Nueva York.

The screenshot shows the official website of the NYC Taxi & Limousine Commission. The header includes the NYC logo and navigation links. The main content area features a sidebar with links to 'Data', 'Pilot Programs', 'Reports', and 'Request Data'. The 'Data' section is expanded, showing 'TLC Trip Record Data' as a primary option. The main text describes the data fields captured in the trip records, including pickup and drop-off locations, distances, fares, and driver-reported passenger counts. It also mentions that the data is provided to the TLC by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP).

Allí se encontró información correspondiente al mes de marzo de 2025, lo cual representaba una mejora significativa en cuanto a actualidad. Por este motivo, se optó por utilizar este conjunto de datos actualizado para el desarrollo del proyecto.



1.3 Cuadro de metadatos

Proceso de negocio	Supervisión y análisis de los servicios de transporte por taxi en NYC, gestionados por la Taxi and Limousine Commission
Nombre del conjunto de datos	Yellow Taxi Trip Records
Descripción general	Datos de viajes realizados por taxis amarillos en NYC, con información sobre horarios, ubicaciones, tarifas y método de pago.
Número de archivos	1 archivo: yellow_tripdata_2025-03.parquet
Formato de almacenamiento	Parquet
Número de registros	4,145,257
Tamaño del archivo	66.7M

Diccionario de datos: ([version online](#))

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1 = Creative Mobile Technologies, LLC 2 = Curb Mobility, LLC 5 = Myle Technologies Inc 7 = Helix
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
passenger_count	The number of passengers in the vehicle.
trip_distance	The traveled distance in miles reported by the taximeter.
RatecodeID	The final rate code in effect at the end of the trip. 1 = Standard rate 2 = JFK 3 = Newark 4 = Nassau or Westchester 5 = Negotiated fare 6 = Group ride 99 = Null/Unknown
store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor Y = store and forward trip N = not a store and forward trip
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
payment_type	A numeric code signifying how the passenger paid for the trip. 1 = Credit card 2 = Cash 3 = No charge 4 = Dispute 5 = Unknown 6 = Voided trip
fare_amount	The time-and-distance fare calculated by the meter.
extra	Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.
mta_tax	Tax that is automatically triggered based on the metered rate in use.
tip_amount	This field is automatically populated for credit card trips. Cash tips are not included.
tolls_amount	Total amount of all tolls paid in trip.
improvement_surcharge	Total collected in improvement surcharge, which began in 2015.
total_amount	The total amount charged to passengers. Does not include cash tips.
congestion_surcharge	Total collected in congestion surcharge.
airport_fee	For pick up only at LaGuardia and John F. Kennedy Airports.
cbd_congestion_fee	Per-trip charge for MTA's Congestion Relief Zone starting Jan. 5, 2025.

1.4 Objetivo del análisis

Asumiendo el rol de directivo de la empresa de Taxis de NY, he definido lo siguiente:

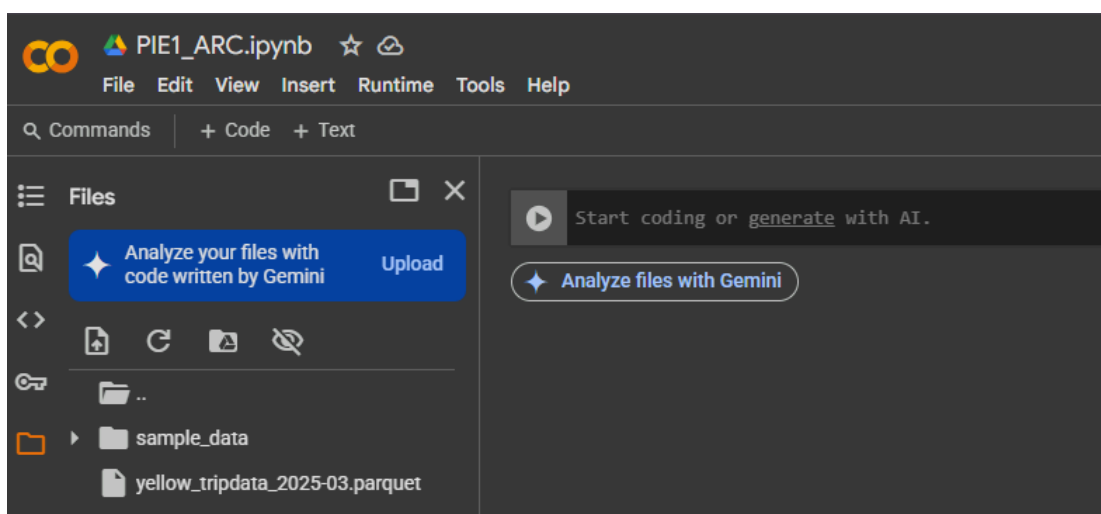
Objetivo: Reducir el uso de efectivo en taxis amarillos de Nueva York para optimizar la trazabilidad de transacciones, disminuir riesgos de seguridad e incrementar la eficiencia administrativa.

KPI	Meta	Objetivo o Criterio operado
% de viajes pagados con tarjeta	$\geq 70\%$	Adopción del pago digital
% de viajes pagados en efectivo	$\leq 30\%$	Reducción del uso de efectivo
Ingresos brutos	Crecimiento sostenido	Incrementar las ganancias de la empresa
% de viajes con status Dispute	$<2\%$	Mejorar los ingresos
Tendencia mensual del uso de tarjeta	Crecimiento sostenido	Éxito en implementación de estrategias digitales
Tip amount promedio	$>10\%$	Incrementar el incentivo al conductor

1.5 Higienización y Transformación de los Datos para su Análisis

En esta etapa se lleva a cabo la limpieza y transformación de los datos con el fin de prepararlos para su análisis. Para ello, se emplea el lenguaje de programación Python, ampliamente utilizado en ciencia de datos. El desarrollo de este proceso se realiza en el entorno colaborativo **Google Colab**.

Como paso previo debemos importar nuestro dataset en nuestro entorno de Google Colab, para poder trabajar con nuestros datos



1.5.1 Carga del dataset

El siguiente paso es cargar las librerías que vamos a necesitar (pandas y pyarrow) procedemos a leer el dataset. utilizamos la función `read_parquet()` de la biblioteca pandas para leer el archivo en formato Parquet. Una vez cargado, aplicamos el método `head()` para visualizar las primeras cinco filas del DataFrame, lo cual nos permite obtener una vista preliminar de la estructura de los datos.

```
[1] !pip install pyarrow
import pandas as pd

Requirement already satisfied: pyarrow in /usr/local/lib/python3.11/dist-packages (18.1.0)

# 1. Cargar el archivo Parquet
df = pd.read_parquet('yellow_tripdata_2025-03.parquet')
df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
0	1	2025-03-01 00:17:16	2025-03-01 00:25:52	1.0	0.90	1.0	N	140	140
1	1	2025-03-01 00:37:38	2025-03-01 00:43:51	1.0	0.60	1.0	N	140	140
2	2	2025-03-01 00:24:35	2025-03-01 00:39:49	1.0	1.94	1.0	N	161	161
3	2	2025-03-01 00:56:16	2025-03-01 01:01:35	2.0	0.95	1.0	N	231	231
4	1	2025-03-01 00:01:44	2025-03-01 00:10:00	1.0	1.50	1.0	N	163	163

1.5.3 Exploración inicial del tamaño del dataset

Una vez visualizadas las primeras filas, el siguiente paso es conocer la dimensión del dataset. Para ello, utilizamos el atributo `.shape` del DataFrame, el cual nos devuelve una tupla con el número de filas y columnas. Esta información es clave para dimensionar el volumen de datos con el que trabajaremos.

```
print("Filas y columnas:", df.shape)
```

```
Filas y columnas: (4145257, 20)
```

Observamos que el conjunto de datos contiene 20 columnas y más de 4 millones de registros, este resultado nos permite entender el tamaño total del conjunto de datos y anticipar decisiones relacionadas con memoria, limpieza o muestreo.

1.5.4 Exploración preliminar del tipo de datos

Para asegurar una correcta interpretación de las columnas, examinamos los tipos de datos contenidos en el DataFrame para identificar posibles conversiones en las columnas

```
# Ver tipos de datos y columnas
print("\n--- Tipos de datos ---")
print(df.dtypes)
```

```
--- Tipos de datos ---
VendorID                int32
tpep_pickup_datetime    datetime64[us]
tpep_dropoff_datetime   datetime64[us]
passenger_count         float64
trip_distance           float64
RatecodeID             float64
store_and_fwd_flag      object
PULocationID            int32
DOLocationID            int32
payment_type            int64
fare_amount             float64
extra                  float64
mta_tax                 float64
tip_amount              float64
tolls_amount            float64
improvement_surcharge   float64
total_amount            float64
congestion_surcharge    float64
Airport_fee             float64
cbd_congestion_fee      float64
dtype: object
```

Se observa que las fechas están correctamente identificadas como datetime64[us], mientras que las variables numéricas como fare_amount, tip_amount, y trip_distance se encuentran en formato float64. La columna store_and_fwd_flag se presenta como tipo object, por lo que podría requerir transformación a tipo categórico para optimizar su análisis.

1.5.5 Detección de valores nulos

Como siguiente paso en el proceso de limpieza, verificamos la presencia de valores nulos en cada columna del DataFrame. Para ello, utilizamos el método `isnull().sum()` que nos permite identificar cuántos valores faltantes hay por columna:

```
# Verificar nulos por columna
print("\n--- Valores nulos por columna ---")
print(df.isnull().sum())
```

```
--- Valores nulos por columna ---
VendorID                0
tpep_pickup_datetime    0
tpep_dropoff_datetime   0
passenger_count        916663
trip_distance           0
RatecodeID             916663
store_and_fwd_flag      916663
PULocationID           0
DOLocationID           0
payment_type            0
fare_amount             0
extra                   0
mta_tax                 0
tip_amount              0
tolls_amount            0
improvement_surcharge   0
total_amount            0
congestion_surcharge    916663
Airport_fee             916663
cbd_congestion_fee      0
dtype: int64
```

Se observa una cantidad significativa de valores nulos en varias columnas, principalmente:

- passenger_count
- RatecodeID
- store_and_fwd_flag
- congestion_surcharge
- Airport_fee

La presencia de estos valores nulos requiere atención especial.

1.5.6 Limpieza y transformación de datos

1.5.6.1 Eliminación de columnas con valores nulos irrelevantes

Dado que el objetivo principal de este análisis es evaluar los pagos realizados por cada viaje, se decidió eliminar aquellas columnas que no aportan valor directo a este propósito. En particular, se identificaron cinco columnas con valores nulos:

[passenger_count], [RatecodeID], [store_and_fwd_flag], [congestion_surcharge], [Airport_fee]

La columna passenger_count sí se conserva, ya que puede influir en el monto pagado (por ejemplo, en viajes individuales). Los valores nulos se imputan con **1**, asumiendo un pasajero como caso más frecuente.

También se eliminan columnas adicionales sin impacto directo en el análisis de pagos:

[VendorID], [extra], [mta_tax], [tolls_amount], [improvement_surcharge], [cbd_congestion_fee]

```
# Imputar valores nulos en passenger_count con 1
df['passenger_count'] = df['passenger_count'].fillna(1)

# Eliminar columnas irrelevantes
df.drop(columns=[
    'RatecodeID', 'store_and_fwd_flag', 'congestion_surcharge', 'Airport_fee',
    'VendorID', 'extra', 'mta_tax', 'tolls_amount',
    'improvement_surcharge', 'cbd_congestion_fee'
], inplace=True)
```

1.5.6.2 Limpieza y transformación de columnas de fecha

a) Limpieza

El dataset incluye dos columnas clave relacionadas con el tiempo del viaje:

- tpep_pickup_datetime: fecha y hora de inicio del viaje
- tpep_dropoff_datetime: fecha y hora de finalización

Para validar la coherencia temporal, verificamos si existen registros donde la hora de finalización sea anterior a la de inicio, lo cual no tiene sentido en el contexto del servicio.

Utilizamos el siguiente código para identificar estos casos:

```
# Contar registros con fechas inconsistentes
inconsistentes = df[df['tpep_dropoff_datetime'] < df['tpep_pickup_datetime']]
print(f"Registros con fechas inconsistentes: {len(inconsistentes)}")
```

→ Registros con fechas inconsistentes: 81

El resultado indica que existen **81 registros** con esta inconsistencia. Dado que representan solo el **0.002%** del total, los eliminamos:

```
# Eliminar registros con fechas inconsistentes
df = df[df['tpep_dropoff_datetime'] >= df['tpep_pickup_datetime']]
```

b) Transformación

Para facilitar el análisis posterior, transformamos las columnas de fecha y hora en nuevas variables más manejables:

- pickup_date: fecha del viaje (sin hora)
- pickup_hour: hora de inicio (0 a 23)
- pickup_day_of_week: día de la semana
- trip_duration_min: duración del viaje en minutos

Código usado:

```
# Crear nuevas variables derivadas de la fecha de inicio
df['pickup_date'] = df['tpep_pickup_datetime'].dt.date
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day_of_week'] = df['tpep_pickup_datetime'].dt.day_name()

# Calcular la duración del viaje en minutos
df['trip_duration_min'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime']).dt.total_seconds() / 60

# Eliminar las columnas originales de fecha y hora
df.drop(columns=['tpep_pickup_datetime', 'tpep_dropoff_datetime'], inplace=True)
```

Antes de continuar con el análisis, es importante validar que la variable `trip_duration_min`, que representa la duración total del viaje en minutos, no contenga valores negativos. Una duración negativa sería ilógica y podría deberse a errores de registro en las fechas.

Para detectar posibles casos, aplicamos el siguiente filtro:

```
# Detectar viajes con duración negativa
viajes_negativos = df[df['trip_duration_min'] < 0]
print(f"Cantidad de viajes con duración negativa: {len(viajes_negativos)}")

# Calcular el porcentaje que representan
porcentaje_negativos = (len(viajes_negativos) / len(df)) * 100
print(f"Porcentaje del total: {porcentaje_negativos:.4f}%")
```

→ Cantidad de viajes con duración negativa: 0
Porcentaje del total: 0.0000%

Arrojo 0 registros, por lo tanto ese dato ya no requiere mayor manipulación.

1.5.6.3 Limpieza y transformación de columnas de ubicación

Las columnas `PULocationID` y `DOLocationID` indican las zonas donde inicia y termina cada viaje mediante identificadores numéricos. Sin embargo, estos valores carecen de significado práctico y pueden ser malinterpretados por los algoritmos como si tuvieran un orden o relación numérica real.

Para mejorar la interpretabilidad y la calidad del análisis, transformamos estos identificadores en variables categóricas descriptivas utilizando el archivo oficial de zonas publicado por la NYC Taxi & Limousine Commission. Este archivo contiene el nombre de cada zona (Zone) y su respectivo distrito (Borough).

Primero, descargamos el archivo `taxi_zone_lookup.csv` desde la fuente oficial:

```
# Cargar archivo de zonas
zone_lookup = pd.read_csv("https://d37ci6vzurychx.cloudfront.net/misc/taxi_zone_lookup.csv")[['LocationID', 'Zone', 'Borough']]
```

Luego, realizamos las uniones para enriquecer el dataset con las nuevas columnas categóricas:

```
# Unir zona de inicio (pickup)
df = df.merge(zone_lookup, left_on='PULocationID', right_on='LocationID', how='left')
df.rename(columns={'Zone': 'PU_Zone', 'Borough': 'PU_Borough'}, inplace=True)
df.drop(columns=['LocationID'], inplace=True)

# Unir zona de destino (dropoff)
df = df.merge(zone_lookup, left_on='DOLocationID', right_on='LocationID', how='left')
df.rename(columns={'Zone': 'DO_Zone', 'Borough': 'DO_Borough'}, inplace=True)
df.drop(columns=['LocationID'], inplace=True)
```

Y finalmente procedemos a eliminar las columnas originales que ya no son necesarias

```
df.drop(columns=['PULocationID', 'DOLocationID'], inplace=True)
```

Como resultado, el conjunto de datos ahora incluye las variables categóricas PU_Zone, DO_Zone, PU_Borough y DO_Borough, lo cual mejora significativamente la interpretación de los datos y evita el uso de identificadores arbitrarios sin contexto.

Después de incorporar las zonas geográficas (PU_Zone, DO_Zone, PU_Borough, DO_Borough) mediante una unión con el archivo de zonas, se realizó una verificación para detectar valores nulos que indicarían fallas en la correspondencia.

```
# Verificar valores nulos en las columnas de zona y distrito
zonas_nulas = df[['PU_Zone', 'DO_Zone', 'PU_Borough', 'DO_Borough']].isnull().sum()
print("--- Valores nulos en columnas de ubicación ---")
print(zonas_nulas)

# Calcular el porcentaje de valores nulos por columna
porcentaje_nulos = (zonas_nulas / len(df)) * 100
print("\n--- Porcentaje de valores nulos ---")
print(porcentaje_nulos.round(4))
```

```
--- Valores nulos en columnas de ubicación ---
PU_Zone      7687
DO_Zone      9987
PU_Borough   1548
DO_Borough   14203
dtype: int64

--- Porcentaje de valores nulos ---
PU_Zone      0.1854
DO_Zone      0.2409
PU_Borough   0.0373
DO_Borough   0.3426
dtype: float64
```

Estos valores nulos no se detectaron en el análisis inicial, ya que los identificadores de ubicación (PULocationID y DOLocationID) estaban completos. Sin embargo, algunos de estos IDs no tienen correspondencia en el archivo oficial de zonas, lo que genera valores faltantes tras el proceso de unión.

Como se puede observar porcentajes son muy bajos y no representan un impacto significativo sobre el análisis general. Por lo tanto, se opta por eliminar estos registros para mantener la integridad de los datos sin complicar el modelo con imputaciones innecesarias.

```
# Eliminar registros con valores nulos en zonas geográficas
df.dropna(subset=['PU_Zone', 'DO_Zone', 'PU_Borough', 'DO_Borough'], inplace=True)
```

1.5.6.4 Transformación de la variable de método de pago

La columna payment_type contiene códigos numéricos que indican el método de pago utilizado. Para mejorar su interpretación, se asignaron etiquetas descriptivas según la documentación oficial.

Cualquier valor no reconocido se clasificó como 'Unknown', manteniendo una estructura categórica coherente. Esta transformación facilita el análisis y la visualización de los métodos de pago.

```
# Mapear códigos de método de pago a valores descriptivos
payment_map = {
    0: 'Flex Fare',
    1: 'Credit Card',
    2: 'Cash',
    3: 'No Charge',
    4: 'Dispute',
    5: 'Unknown',
    6: 'Voided Trip'
}
df['payment_type'] = df['payment_type'].map(payment_map).fillna('Unknown')
```

Después de aplicar la transformación, es importante verificar que:

- Todos los valores de payment_type sean válidos (esperados)
- No existan valores extraños, nulos o mal mapeados

```
# Verificar los valores únicos en payment_type
print("--- Valores únicos en 'payment_type' ---")
print(df['payment_type'].value_counts())

# Verificar si hay valores nulos o vacíos
print("\n--- Valores nulos en 'payment_type' ---")
print(df['payment_type'].isnull().sum())
```

```
--- Valores únicos en 'payment_type' ---
payment_type
Credit Card    2683721
Flex Fare       915288
Cash            400863
Dispute         90423
No Charge       26466
Name: count, dtype: int64

--- Valores nulos en 'payment_type' ---
0
```

Los resultados obtenidos nos confirman que la transformación de payment_type fue exitosa:

- Solo hay 5 valores válidos y esperados, sin datos extraños ni nulos.
- Todos los métodos de pago fueron correctamente mapeados.

1.5.6.5 Análisis de viajes con método de pago "Dispute"

Para investigar posibles patrones anómalos en los viajes registrados como Dispute, se filtraron estos registros y se ordenaron por tres variables clave:

- pickup_date: fecha del viaje
- pickup_hour: hora de inicio
- trip_duration_min: duración total en minutos

Veamos un ejemplo de estos datos para ver si logramos detectar patrones..

```
# Ordenar por fecha, hora y duración de viaje
df_dispute = df[df['payment_type'] == 'Dispute'].copy()

df_dispute = df_dispute.sort_values(
    by=['pickup_date', 'pickup_hour', 'trip_duration_min'],
    ascending=[True, True, True]
)

# Ver los primeros registros ordenados
df_dispute[['pickup_date', 'pickup_hour', 'trip_duration_min', 'total_amount']].head(10)
```

	pickup_date	pickup_hour	trip_duration_min	total_amount
2696	2025-03-01	0	0.100000	-23.50
2697	2025-03-01	0	0.100000	23.50
885	2025-03-01	0	0.183333	-74.75
886	2025-03-01	0	0.183333	74.75
2231	2025-03-01	0	0.216667	-5.50
2232	2025-03-01	0	0.216667	5.50
2750	2025-03-01	0	0.400000	-74.75
2751	2025-03-01	0	0.400000	74.75
4379	2025-03-01	0	0.466667	8.75
3556	2025-03-01	0	0.516667	8.75

Interesante: al ordenar los viajes con método de pago **Dispute** por fecha, hora y duración, se identificaron pares de registros con los mismos valores pero montos opuestos. Esto sugiere que estos viajes fueron **anulados o revertidos**, probablemente reflejando una cancelación seguida de un reembolso.

Para facilitar su detección y limpieza:

1. Se convirtieron los valores de total_amount y tip_amount a su valor absoluto.
2. Luego se eliminaron los duplicados exactos conservando solo la primera ocurrencia.

```
# Normalizar montos a valor absoluto
df['total_amount'] = df['total_amount'].abs()
df['tip_amount'] = df['tip_amount'].abs()

# Eliminar registros duplicados exactos
df = df.drop_duplicates(keep='first')
```

1.5.6.7 Verificación de valores negativos en variables numéricas

Como paso subsecuente del proceso de limpieza, se revisaron las columnas numéricas clave para asegurar que no quedaran valores negativos, los cuales serían inválidos en el contexto del análisis.

Se inspeccionaron particularmente las variables:

- trip_distance: no puede ser negativa
- trip_duration_min: una duración negativa carece de sentido
- total_amount y tip_amount: ya fueron convertidas a valores absolutos

```
# Verificar si existen valores negativos en las columnas numéricas relevantes
print("Distancias negativas:", (df['trip_distance'] < 0).sum())
print("Duraciones negativas:", (df['trip_duration_min'] < 0).sum())
print("Montos totales negativos:", (df['total_amount'] < 0).sum())
print("Propinas negativas:", (df['tip_amount'] < 0).sum())
```

```
Distancias negativas: 0
Duraciones negativas: 0
Montos totales negativos: 0
Propinas negativas: 0
```

El resultado confirmó que no existen valores negativos residuales, por lo que se concluye satisfactoriamente la etapa de limpieza y transformación de datos.

1.5.6.8 Análisis de valores atípicos (outliers)

Se realizó un análisis exploratorio para identificar valores atípicos en las variables numéricas trip_distance y total_amount mediante el método del rango intercuartílico (IQR). Este análisis reveló lo siguiente:

- 11.63% de los registros tienen una distancia mayor a 6.94 millas
- 9.54% tienen un monto total superior a 49.71 USD

Aunque estos valores podrían considerarse extremos y ser excluidos en una etapa de modelado, no se eliminarán en esta fase de limpieza, ya que el objetivo actual es conservar la integridad del dataset para exploración posterior.

Estos casos pueden aportar información valiosa en una etapa de análisis, como la detección de viajes inusuales, tarifas especiales o eventos excepcionales. Por lo tanto, se conservarán para su evaluación futura.

```
def detectar_outliers_iqr(df, columna):
    Q1 = df[columna].quantile(0.25)
    Q3 = df[columna].quantile(0.75)
    IQR = Q3 - Q1
    limite_superior = Q3 + 1.5 * IQR
    outliers = df[df[columna] > limite_superior]
    return outliers, limite_superior

# Detectar outliers en trip_distance
outliers_dist, max_dist = detectar_outliers_iqr(df, 'trip_distance')
print(f"Outliers en trip_distance: {len(outliers_dist)} (límite > {max_dist:.2f})")

# Detectar outliers en total_amount
outliers_total, max_total = detectar_outliers_iqr(df, 'total_amount')
print(f"Outliers en total_amount: {len(outliers_total)} (límite > {max_total:.2f})")

Outliers en trip_distance: 471821 (límite > 6.94)
Outliers en total_amount: 387160 (límite > 49.71)

[46] print(f"% de outliers en trip_distance: {(len(outliers_dist) / len(df) * 100):.2f}%")
print(f"% de outliers en total_amount: {(len(outliers_total) / len(df) * 100):.2f}%")

% de outliers en trip_distance: 11.63%
% de outliers en total_amount: 9.54%
```

1.5.6.9 Verificación final de limpieza y preparación para análisis en Weka

```
# Valores nulos por columna
print("\n--- Valores nulos por columna ---")
print(df.isnull().sum())

# Muestra aleatoria de registros
print("\n--- Muestra de registros ---")

from IPython.display import display

# Mostrar tabla bonita en notebook sin cortes
display(df.sample(20, random_state=42))

--- Valores nulos por columna ---
passenger_count    0
trip_distance       0
payment_type        0
tip_amount          0
total_amount        0
pickup_date         0
pickup_hour         0
pickup_day_of_week  0
trip_duration_min   0
PU_Zone             0
PU_Borough          0
DO_Zone             0
DO_Borough          0
dtype: int64

--- Muestra de registros ---
```

	passenger_count	trip_distance	payment_type	tip_amount	total_amount	pickup_date	pickup_hour	pickup_day_of_week	trip_duration_min	PU_Zone	PU_Borough	DO_Zone	DO_Borough
777564	1.0	0.80	Credit Card	2.35	14.30	2025-03-08	13	Saturday	5.966667	Greenwich Village North	Manhattan	Penn Station/Madison Sq West	Manhattan
585515	2.0	0.92	Cash	0.00	16.55	2025-03-06	17	Thursday	8.583333	Times Sq/Theatre District	Manhattan	Central Park	Manhattan
2852104	1.0	0.40	Credit Card	1.95	11.80	2025-03-28	12	Friday	3.100000	Penn Station/Madison Sq West	Manhattan	Midtown South	Manhattan
1396347	1.0	1.30	Credit Card	2.20	16.90	2025-03-14	11	Friday	12.116667	Upper East Side North	Manhattan	Upper East Side South	Manhattan
2310819	1.0	2.79	Credit Card	4.27	25.62	2025-03-22	23	Saturday	15.250000	Greenwich Village North	Manhattan	Midtown East	Manhattan
2717503	1.0	2.35	Credit Card	3.64	21.84	2025-03-27	8	Thursday	0.000000	Upper West Side South	Manhattan	Morningside Heights	Manhattan
3885093	1.0	0.01	Flex Fare	0.00	24.63	2025-03-27	16	Thursday	15.666667	West Village	Manhattan	Gramercy	Manhattan
521854	1.0	21.18	Credit Card	16.54	99.23	2025-03-06	6	Thursday	51.350000	JFK Airport	Queens	Upper West Side South	Manhattan

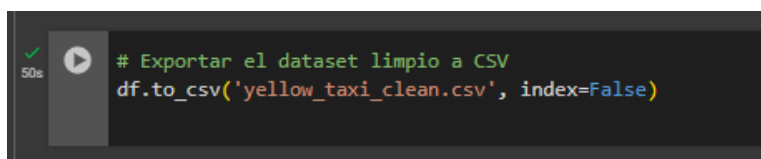
Para concluir la etapa de limpieza, se realizó una verificación adicional de dos aspectos:

1. **Valores nulos por columna**
2. **Muestra aleatoria de registros**

Con esto, se valida que el dataset está completamente limpio, estructurado y preparado para exportación.




1.5.6.12 Exportación del dataset limpio

Una vez completado el proceso de limpieza, transformación y verificación, el dataset final se exporta en formato CSV, el cual es compatible con Weka y otras herramientas de análisis de datos.



```
# Exportar el dataset limpio a CSV
df.to_csv('yellow_taxi_clean.csv', index=False)
```

El archivo generado, yellow_taxi_clean.csv, contiene todos los registros depurados, sin valores nulos, con columnas renombradas y variables transformadas listas para análisis exploratorio, modelado o minería de datos.

Name	Size	Date modified
 PI E1 ARC.gdoc	1 KB	5/24/2025 11:33 PM
 yellow_taxi_clean.csv	475,550 KB	5/24/2025 11:30 PM
 yellow_tripdata_2025-03.parquet	68,325 KB	5/24/2025 2:56 PM

Como podemos observar el archivo limpio en formato CSV es mucho mas grande, esto se debe a que el formato parquet es un formato binario optimizado para almacenamiento eficiente.

Archivos

Se creo un repositorio para este proyecto, en el cual se puede descargar:

- Este archivo en formato PDF [descargar]
- El script completo en python de limpieza y transformacion [Descargar]
- El dataset original en formato parquet [Descargar]
- El dataset limpio en formato CSV [Descargar]

Referencias:

- DataHub Collections. (s. f.). <https://datahub.io/collections>
- Dataquest. (2024, 9 diciembre). *Complete Guide to Data Cleaning in Python – Dataquest*. <https://www.dataquest.io/guide/data-cleaning-in-python-tutorial/>
- Dataset search. (s. f.). Google Dataset Search. <https://datasetsearch.research.google.com/>
- Find Open Datasets and Machine Learning Projects | Kaggle. (s. f.). <https://www.kaggle.com/datasets?search=yellowcab>
- GeeksforGeeks. (2025, 28 abril). *Read a Parquet File Using Pandas*. GeeksforGeeks. <https://www.geeksforgeeks.org/read-a-parquet-file-using-pandas/>
- Google Colab. (s. f.). *Colab.google*. colab.google. <https://colab.google/>
- Keldenich, T. (2023, 14 noviembre). 3 Ways to Open a Parquet File in Python - Easy. *Inside Machine Learning*. <https://inside-machinelearning.com/en/open-parquet-python/>
- Margono, M. A. (2023, 3 noviembre). NYC Taxi Trip Data Analysis - M. Aris Margono - Medium. *Medium*. <https://medium.com/@muhammadaris10/nyc-taxi-trip-data-analysis-45ecfdb6f91>
- Ngo, H. (2025, 5 marzo). *How to Clean Your Data in Python*. Towards Data Science. <https://towardsdatascience.com/how-to-clean-your-data-in-python-8f178638b98d/>
- Norberte. (s. f.). *DS_professional_development/DS interviews/NYC Yellow Taxi.ipynb at master · norberte/DS_professional_development*. GitHub. https://github.com/norberte/DS_professional_development/blob/master/DS%20interviews/NYC%20Yellow%20Taxi.ipynb
- Plataforma Nacional de Datos Abiertos. (s. f.). datos.gob.mx. <https://www.datos.gob.mx/>
- Samuel, O. (2024, 9 septiembre). *How to Use Pandas for Data Cleaning and Preprocessing*. freeCodeCamp.org. <https://www.freecodecamp.org/news/data-cleaning-and-preprocessing-with-pandasbdvhi/>
- TLC Trip Record Data - TLC. (s. f.). <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>
- Trevino, M. V., & Trevino, M. V. (2022, 6 agosto). NYC Yellow Taxi Data Usage and Profitability. *Data Science Blog*. <https://nycdatascience.com/blog/r/nyc-yellow-taxi-data-usage-and-profitability/>
- W3Schools.com. (s. f.). https://www.w3schools.com/python/pandas/pandas_cleaning.asp
- Zhangqi. (s. f.). *GitHub - zhangqi0210/Yellow_Cab: Yellow Cab Case Study: Data-Driven Business Insights*. GitHub. https://github.com/zhangqi0210/Yellow_Cab