Tecnológico de Monterrey

Bitácora de decisiones

Paulina Leal Mosqueda, A01659576
Santiago Nava Figueroa A01174557
Uziel Heredia Estrada, A01667072

4 de diciembre de 2025

Aplicación de métodos multivariados en Ciencias de Datos (Gpo 601)

Profesores: Juliho Davida Castillo Colmenares y Jonathan Montalvo Urquizo

Instituto Tecnológico y de Estudios Superiores de Monterrey CCM

¿Qué se transformó?	¿Por qué?	¿Cómo?
Reemplazar commas en Precio Unitario por puntos y convertir a entero.	Para poder generar cálculos y análisis numéricos, se necesita que la variable también sea numérica.	*productos['Precio_Unitario'].str.replace(",", ".").astype(float)*
Cambiar typo en columna de Fecha de registro.	Evita errores tipográficos y mejora presentación.	clientes.rename(columns={'Fecha_Resgistro': 'Fecha_Registro'})
Juntar ventas con clientes	Se usa para identificar las compras de cada uno de los clientes.	Juntar columnas mediante *ID_Cliente. df = pd.merge(ventas, clientes, on ='ID_Cliente')*
Juntar df con productos	Se usa para identificar las compras de cada uno de los clientes según sus productos.	Juntar columnas mediante *ID_Producto. df.merge(productos, on ='ID_Producto')*
Renombrar Fecha como Fecha_Venta	Evita confusiones con la fecha de registro del cliente	df.rename(columns={'Fecha':'Fecha_Venta'})
Ordenar columnas de df	Tener mejor claridad en el orden de los datos.	*df[['ID_Venta', 'ID_Cliente', 'Nombre', 'Apellido', 'Email', 'Fecha_Registro', 'Fecha_Venta', 'Región', 'Cantidad', 'Método_Pago', 'Estado', 'Nombre_producto', 'Categoría', 'Precio_Unitario', 'Stock']]*
Cambiar etiquetas para Método de Pago	Hacer que las visualizaciones sean más entendibles al darle la etiqueta de cada método de pago.	*df['Método_Pago'] = df['Método_Pago'].map({ 1: 'Efectivo', 2: 'Tarjeta de Crédito', 3: 'Tarjeta de Débito', 4: 'Mercado Pago', 5: 'Transferencia' })*
Eliminar duplicados	Evitar hacer un análisis con compras duplicadas, ya que esto altera los resultados.	*df = df.drop_duplicates(subset=cols_clave, keep='first')*
Generar nueva variable df_encoder y utilizar OneHotEncoder para las variables categóricas	Esta variable será utilizada para realizar modelos predictivos. Es necesario tener las variables	Se utiliza OneHotEncoder, ya que se considera que todas estas variables son nominales: método de pago, estado y categoría.

	categóricas como números, para que puedan ser aplicadas en el modelo.	*encoded_variable = encoder.fit_transform(df_encoder[["Variable"]])* *variable_df = pd.DataFrame(encoded_nueva_variable, columns=encoder.get_feature_names_out(['Variable']))*
Estandarizar variables numéricas	Para poder realizar el modelo predictivo, se necesita tener escalas ya que las variables tienen dimensiones diferentes. Se eligió Standard Scaler para hacer que su desviación estándar sea 1 y su media 0.	*numeric_var = ["Cantidad", "Precio_Unitario", "Stock"]* *scaler = StandardScaler()* *df_encoder[numeric_var] = scaler.fit_transform(df_encoder[numeric_var])*
Variables como fechas	Las fechas, no están puestas con datetime. Para poder generar visualizaciones de series de tiempo se necesita de esto.	*df['Fecha_Venta'] = pd.to_datetime(df['Fecha_Venta'])* *df['Fecha_Registro'] = pd.to_datetime(df['Fecha_Registro'])*
Variables de temporalidad: Mes, día, año	Sirve para entender el comportamiento dependiendo de la temporalidad.	*df['Año_Venta'] = df['Fecha_Venta'].dt.year* *df['Dia_Venta'] = df['Fecha_Venta'].dt.day* *df['Mes_Venta'] = df['Fecha_Venta'].dt.month*
Estacionalidad	Entender las ventas por estación del año en Argentina	*df['Temporada'] = df['Mes_Venta'].map({* *1: 'Verano', 2: 'Verano', 3: 'Verano',* *4: 'Otoño', 5: 'Otoño',* *6: 'Otoño', 7: 'Invierno',8: 'Invierno',* *9: 'Invierno', 10: 'Primavera', 11: 'Primavera',* *12: 'Primavera'* *})*
Ingreso generado	Sirve para entender la	*df['Ingreso'] = df['Cantidad'] **

	cantidad de ingreso que la empresa está generando	*df['Precio_Unitario']*
Generar variables RFM	Sirve para entender el comportamiento de los clientes y encontrar hallazgos importantes durante el análisis	*snapshot_date = df['Fecha_Venta'].max() + pd.Timedelta(days=1)* *df['Recencia'] = (* *snapshot_date - df.groupby('ID_Cliente')['Fecha_Venta'].transform('max')* *).dt.days* *df['Frecuencia'] = (* *df.groupby('ID_Cliente')['ID_Venta'].transform('count')* *)* *df['Valor_Monetario'] = (* *df.groupby('ID_Cliente')['Ingreso'].transform('sum')* *)*
Tabla para clusters por producto	Sirve para generar clusters de los productos y entender el comportamiento de cada uno.	*df_prod = df.groupby("Nombre_producto").agg(* *{* *"Cantidad": ["sum", "mean"], # Promedio y media de cantidad vendida* *"Ingreso": "sum", # Ingreso total por producto* *"Precio_Unitario": "mean", # Precio promedio por producto* *"Stock": "mean", # Stock promedio por producto* *"Mes_Venta": "mean", # Mes promedio de venta* *"Categoría": "first" # Categoría del producto* *})* *df_prod.columns = [* *"Cantidad_total",* *"Cantidad_promedio",* *"Ingreso_total",* *"Precio_promedio",* *"Stock_promedio",* *"Mes_promedio",* *"Categoria"*

		]
Utilizar escala logarítmica para variables categóricas en clustering de productos	Hay un outlier de asado, que tiene valores más altos que el resto. Sin este valor solo haya dos clusters. Por eso, se optó por usar una escala logarítmica y mantener el dato	*log_vars = [* *"Cantidad_total",* *"Cantidad_promedio",* *"Ingreso_total",* *"Precio_promedio",* *"Stock_promedio",* *# columnas de región:* *]*
Utilizar Standard Scaler para df_prod_log	Estandarizar para K-Means las variables continuas	*scaler.fit_transform(df_prod_log[log_vars])*
Generar tabla para clustering por clientes	Para entender los patrones y grupos específicos de cada cliente, se genera una tabla con la información necesaria	Agrupar df, por recencia (min), frecuencia (máx), valor monetario (sum), cantidad (sum), precio unitario (mean), región (moda), temporada (moda), mes y día (moda), categoría (moda) y estado (moda).
Usar OneHotEncoder y Standard Scaler para tabla de clientes	K-Means necesita que las variables estén estandarizadas y que estén codificadas.	preprocessor = ColumnTransformer(transformers=[("num", StandardScaler(), num_cols), # estandarizar numéricas ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), cat_cols) # one hot encoding], remainder="drop") .reset_index())