Análisis exploratorio de datos para decisiones comerciales

I. Sobre la ejecución

1. IDA: Carga, limpieza y transformación de datos

Lo primero a realizar es importar las librerías numpy, pandas, matplotlib.pyplot y seaborn para usarlas posteriormente.

Luego, en mi caso, decidí importar una base de datos ya disponible en Kaggle sobre la empresa brasilera Olist. Estos datos venían en formato csv y en distintos datasets, así que los cargaremos todos.

```
# Carga de Archivos CVS
df_clientes = pd.read_csv('/content/olist_customers_dataset.csv')
df_localidades = pd.read_csv('/content/olist_geolocation_dataset.csv')
df_orden_items = pd.read_csv('/content/olist_order_items_dataset.csv')
df_pagos = pd.read_csv('/content/olist_order_payments_dataset.csv')
df_criticas = pd.read_csv('/content/olist_order_reviews_dataset.csv')
df_orden_dataset = pd.read_csv('/content/olist_orders_dataset.csv')
df_productos = pd.read_csv('/content/olist_products_dataset.csv')
df_vendedores = pd.read_csv('/content/olist_sellers_dataset.csv')
df_categorias = pd.read_csv('/content/product_category_name_translation
```

Una vez cargados los datos y asignados a un DataFrame con Pandas, procedemos a ver el tipo de dato que contienen y los nombres de las columnas e identificar cuales de estas nos permitirán unir los DataFrames, para ello utilizamos .info().

```
# Revisamos los nombres de las columnas para unirlas
df_clientes.info()
print('-----')
df_localidades.info()
print('-----')
df_orden_items.info()
print('-----')
df_pagos.info()
print('-----')
df_criticas.info()
print('-----')
df_orden_dataset.info()
print('-----')
df_productos.info()
print('-----')
df_vendedores.info()
print('-----')
df_categorias.info()
```

Para hacer más práctico no pondré el detalle de cada DF, pero con esta información podemos considerar las columnas ‘customer_id’, ‘order_id’, ‘product_id’,

‘product_category_name’ y ‘order_id’ para la fusión de los dataframes, para esto utilizaremos la función .merge().

```
# Coincidencias: order_id, customer_id y product_id
df_consolidado = df_clientes.merge(df_orden_dataset, on='customer_id', how= 'left')
df_consolidado = df_consolidado.merge(df_orden_items, on='order_id', how= 'left') #
df_consolidado = df_consolidado.merge(df_productos, on='product_id', how= 'left') #
df_consolidado = df_consolidado.merge(df_categorias, on='product_category_name', how
df_consolidado = df_consolidado.merge(df_pagos, on='order_id', how= 'left') # DF sem
df_consolidado = df_consolidado.merge(df_criticas, on='order_id', how= 'left') # DF
```

Si bien, se podría haber hecho en una sola línea, decidí ir uniendo uno por uno y en cada línea para que pudiese verse más ordenado. Con el DataFrame consolidado debemos corroborar que se hayan unido sin problema las columnas, usaré .info() y no .head() debido a que son muchas columnas.

```
# Revisamos el DF
df_consolidado.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119143 entries, 0 to 119142
Data columns (total 37 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   customer_id                    119143 non-null  object
 1   customer_unique_id             119143 non-null  object
 2   customer_zip_code_prefix       119143 non-null  int64
 3   customer_city                  119143 non-null  object
 4   customer_state                 119143 non-null  object
 5   order_id                       119143 non-null  object
 6   order_status                   119143 non-null  object
```

Son 37 columnas, de las cuales catorce contienen datos float, una contiene dato integer y veintidos contienen datos object. Es debido a esto que solo consideraremos los más relevantes para nuestro IDA y EDA. Guardamos en un DF solo las trece columnas más relevantes.

```
# Priorizaremos mantener solo los datos relevantes y numéricos
columnas_relevantes=['customer_id','customer_city','order_id','order_stat
df=df_consolidado[columnas_relevantes]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119143 entries, 0 to 119142
Data columns (total 14 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   customer_id                    119143 non-null  object
 1   customer_city                  119143 non-null  object
 2   order_id                       119143 non-null  object
 3   order_status                   119143 non-null  object
 4   order_purchase_timestamp       119143 non-null  object
 5   order_item_id                  118310 non-null  float64
 6   product_id                     118310 non-null  object
 7   seller_id                      118310 non-null  object
```

Ahora, procedemos a la limpieza de datos. Eliminamos valores nulos con .dropna(inplace=True) y lo visualizamos para corroborar que se haya realizado correctamente la eliminación.

```
df.dropna(inplace=True)
print(df.isnull().sum())
```

```
customer_id                     0
customer_city                   0
order_id                        0
order_status                    0
order_purchase_timestamp        0
order_item_id                   0
product_id                      0
seller_id                       0
price                           0
freight_value                   0
product_category_name_english   0
payment_value                   0
review_id                       0
review_score                    0
```

Lo mismo hacemos con los valores duplicados.

```
# Eliminamos valores duplicados y corroboramos que estén eliminadas
df.drop_duplicates(inplace=True)
print(df.duplicated().sum())
```

```
/tmp/ipython-input-3787325925.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/panda
  df.drop_duplicates(inplace=True)
0
```

Luego, transformamos el tipo de dato de la fecha, de objeto a fecha.

```
ımos tipo de dato de fecha objeto a datetime
ırchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])
```

```
/tmp/ipython-input-3434635853.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_times
```

Pasamos la fecha a un formato mensual por año para facilitar las futuras conclusiones, análisis y modelamiento en las otras etapas.

```
df['mes_año'] = df['order_purchase_timestamp'].dt.to_period('M')
```

```
/tmp/ipython-input-2798275462.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
  df['mes_año'] = df['order_purchase_timestamp'].dt.to_period('M')
```

Finalmente, renombramos las columnas para que sean claras de entender.

```python
# Renombramos las columnas
df.rename(columns={'customer_id': 'id_cliente',
                   'customer_city':'ciudad_cliente',
                   'order_status':'estado_orden',
                   'order_purchase_timestamp':'fecha_compra',
                   'order_item_id':'id_item_orden',
                   'order_id':'id_orden',
                   'product_id':'id_producto',
                   'seller_id':'id_vendedor',
                   'price':'precio',
                   'freight_value':'valor_envio',
                   'product_category_name_english':'categoria_producto',
                   'review_score':'puntuacion_critica',
                   'payment_value':'valor_pago'}, inplace=True)
```

2. EDA

2.1. Análisis Exploratorio de Datos

El Dataset ya está generado, así que, lo primero que haremos es usar un .info() para identificar el tipo de dato para saber qué tipo de gráfico utilizaremos.

```python
df.info() # para ver tipos de variables
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 114541 entries, 0 to 119142
Data columns (total 14 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   id_cliente          114541 non-null  object
 1   ciudad_cliente      114541 non-null  object
 2   id_orden            114541 non-null  object
 3   estado_orden        114541 non-null  object
 4   fecha_compra        114541 non-null  datetime64[ns]
 5   id_item_orden       114541 non-null  float64
 6   id_producto         114541 non-null  object
 7   id_vendedor         114541 non-null  object
 8   precio              114541 non-null  float64
 9   valor_envio         114541 non-null  float64
 10  categoria_producto  114541 non-null  object
 11  valor_pago          114541 non-null  float64
 12  puntuacion_critica  114541 non-null  float64
 13  mes_año             114541 non-null  period[M]
dtypes: datetime64[ns](1), float64(5), object(7), period[M](1)
memory usage: 13.1+ MB
```

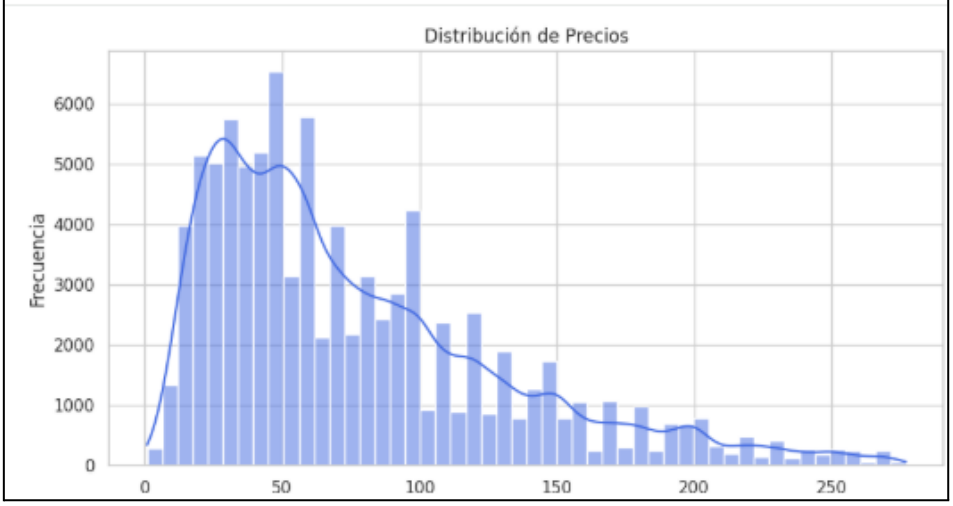
Haremos una pequeña tabla resumen estableciendo el tipo de variable:

Variable	Tipo
ID cliente	Categórica - Nominal
Ciudad	Categórica - Nominal
ID orden	Categórica - Ordinal
Estado Orden	Categórica - Ordinal
Fecha	Categórica - Ordinal
ID item	Categórica - Nominal

ID Producto	Categórica - Nominal
ID Vendedor	Categórica - Nominal
Precio	Cuantitativa - Continua
Valor Transporte	Cuantitativa - Continua
Categoría Producto	Categórica - Nominal
Valor del Pago	Cuantitativa - Continua
Puntuación Crítica	Cuantitativa - Continua

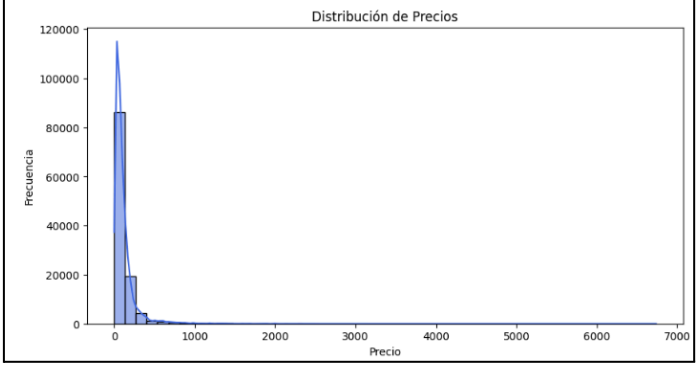
Creamos un histograma sobre la distribución de precios y frecuencia de compra con los datos con menor cantidad de outliers.

```
plt.figure(figsize=(10, 5))
sns.histplot(df['precio'],bins=50,kde=True, color='royalblue')
plt.title('Distribución de Precios')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')
plt.show()
```



Una vez definidas las variables, generamos un histograma para visualizar la distribución de los precios en base a los precios que hay y la frecuencia de compra. Junto con esto, también podremos identificar si es que hay outliers.

```
# Hallazgos en Visual Studio Core
plt.figure(figsize=(10, 5))
sns.histplot(df['precio'],bins=50,kde=True, color='royalblue')
plt.title('Distribución de Precios')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')
plt.show()
```



Podemos notar que no están distribuidos uniformemente los datos, esto nos indica que habrá outliers que tratar. Excepcionalmente, eliminaremos los outliers porque están muy dispersos y lejanos. Utilizaremos solo columnas financieras ‘precio’, ‘valor_envio’ y ‘valor_pagado’, no consideramos la evaluación (puntuación crítica), ya que los valores de esta no presentan outliers. Utilizamos for col in las columnas definidas para que calcule el IQR de cada columna y elimine el lower_bound y el upper bound de cada columna.

```python
# Columnas financieras para la eliminación de outliers
columnas_financieras = ['precio', 'valor_envio', 'valor_pagado']

for col in columnas_financieras:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
```

2.2. Conceptos Básicos de Estadística Descriptiva

Para calcular la estadística descriptiva básica utilizaremos la función .describe() y la aplicaremos al precio, el valor de envío, valor pagado y evaluación.

```python
df[['precio','valor_envio','valor_pagado','evaluacion']].describe()
```

	precio	valor_envio	valor_pagado
count	91364.000000	91364.000000	91364.000000
mean	76.917766	15.715410	104.607238
std	54.751912	5.068852	64.809865
min	0.850000	2.140000	0.000000
25%	34.990000	12.650000	53.850000
50%	59.900000	15.310000	89.480000
75%	104.000000	18.310000	143.822500
max	277.000000	30.630000	302.100000

el 50% corresponde a la mediana de cada variable, mean corresponde al promedio, min y max corresponden al valor mínimo y máximo dentro de esa variable, y std como la desviación estándar.

```python
df[['evaluacion','precio', 'valor_envio', 'valor_pagado']].var()
```

	0
evaluacion	1.816730
precio	2843.405569
valor_envio	25.568428
valor_pagado	3858.645825

```python
df[['evaluacion','precio', 'valor_envio', 'valor_pagado']].std()
```

	0
evaluacion	1.350837
precio	54.751912
valor_envio	5.068852
valor_pagado	64.809865

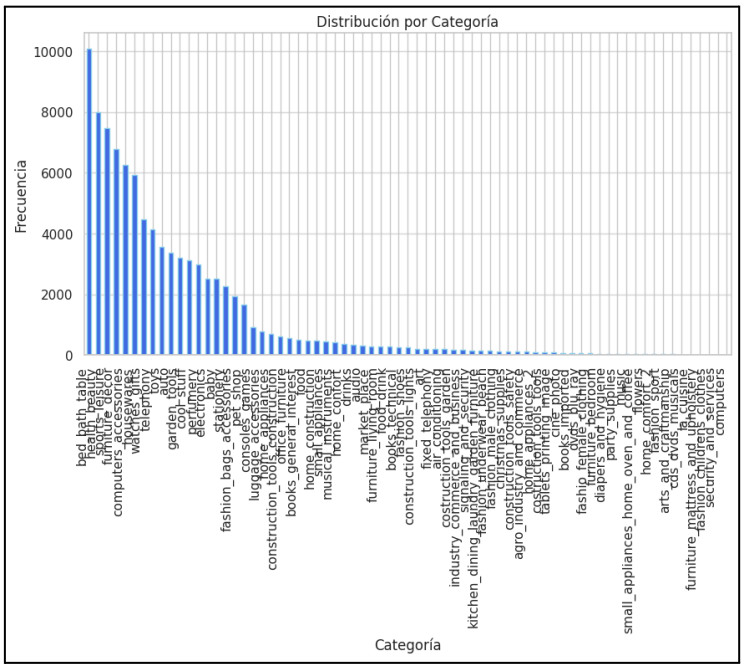
Calculamos igualmente la varianza con .var() y la desviación estándar con std(). También vemos la moda con .mode() de las variables categóricas.

```python
category_mode = df['categoria_producto_ing'].mode()[0]
print("Moda de la categoría:", category_mode)
vendedor_mode = df['id_vendedor'].mode()[0]
print("Moda del vendedor:", vendedor_mode)
time_mode = df['fecha_orden'].mode()[0]
print("Moda del tiempo:", time_mode)
```

```
Moda de la categoría: bed_bath_table
Moda del vendedor: 6560211a19b47992c3666cc44a7e94c0
Moda del tiempo: 2017-09-23 14:56:45
```

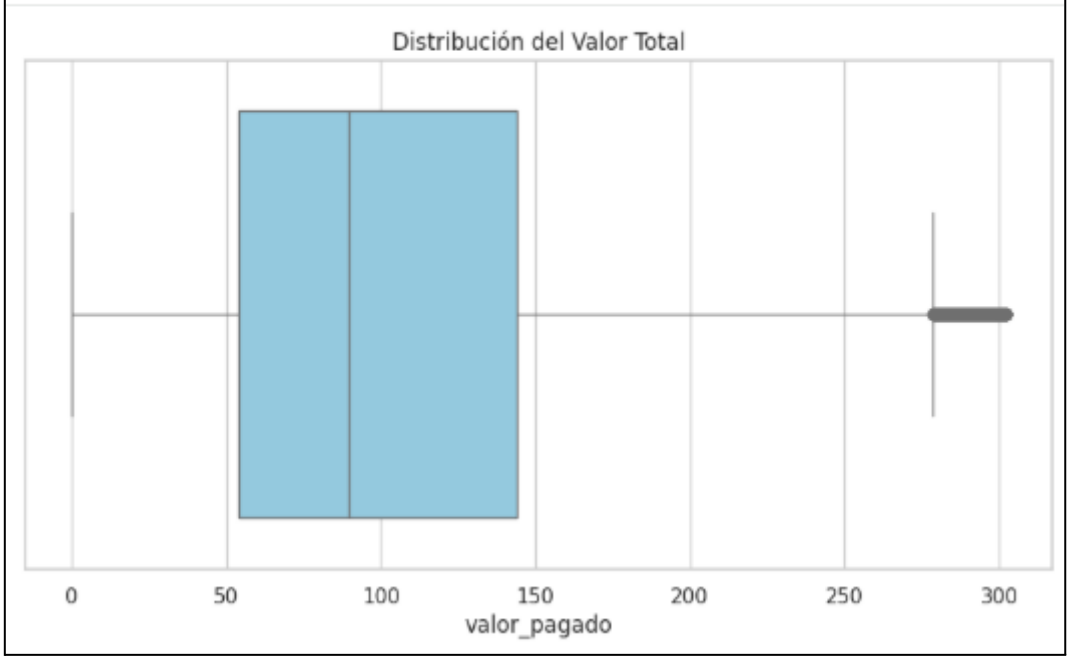
Generamos un histograma de las variables categóricas en base a la frecuencia de datos.

```
plt.figure(figsize=(10, 5))
df['categoria_producto_ing'].value_counts().plot(kind='bar', edgecolor='skyblue'
plt.title('Distribución por Categoría')
plt.xlabel('Categoría')
plt.ylabel('Frecuencia')
plt.grid(axis='y', alpha=0.75)
plt.xticks(rotation=90, ha='right')
plt.show()
```



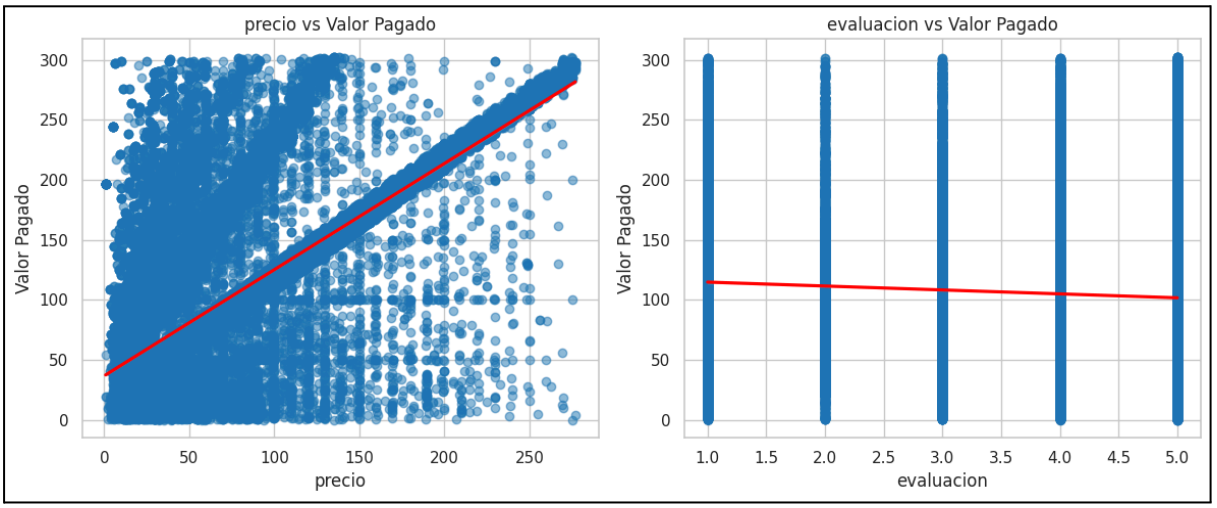
Luego graficamos con boxplot la distribución del valor total/ valor pagado para visualizar outliers, visualizar la mediana, el intervalo de confianza y similares.

```
plt.figure(figsize=(10, 5))
sns.boxplot(x=df['valor_pagado'], color = 'skyblue')
plt.title('Distribución del Valor Total')
plt.show()
```



Luego graficamos con scatterplots, el precio y evaluación con relación al valor pagado..

```
# Scatterplots
variables_predictoras= ['precio','evaluacion']
fig, axes = plt.subplots(1, len(variables_predictoras), figsize=(6 * len(variables_predictoras), 5))
axes = np.array(axes).flatten()
scatter_kws={'alpha': 0.05, 's': 2}

for i, var in enumerate(variables_predictoras):
    sns.regplot(ax=axes[i], x=df[var], y=df['valor_pagado'], scatter_kws={'alpha': 0.5}, line_kws={'color': 'red'})
    axes[i].set_title(f'{var} vs Valor Pagado')
    axes[i].set_xlabel(var)
    axes[i].set_ylabel('Valor Pagado')

plt.tight_layout()
plt.show()
```



En el caso de evaluación no se logra percibir tan bien, pero en el caso del precio vs valor pagado muestra una alta presencia de outliers.

2.3. Correlación

La correlación entre las variables las observaremos con la matriz de correlación y Pearson, solo sobre las variables evaluación, precio, valor de envío y valor pagado.

```
# Matriz de correlación
columnas_numericas = ['evaluacion', 'precio', 'valor_envio', 'valor_pagado']
correlaciones = df[columnas_numericas].corr()
correlaciones
```

	evaluacion	precio	valor_envio	valor_pagado
evaluacion	1.000000	0.029608	-0.024573	-0.068249
precio	0.029608	1.000000	0.301774	0.747239
valor_envio	-0.024573	0.301774	1.000000	0.284358
valor_pagado	-0.068249	0.747239	0.284358	1.000000

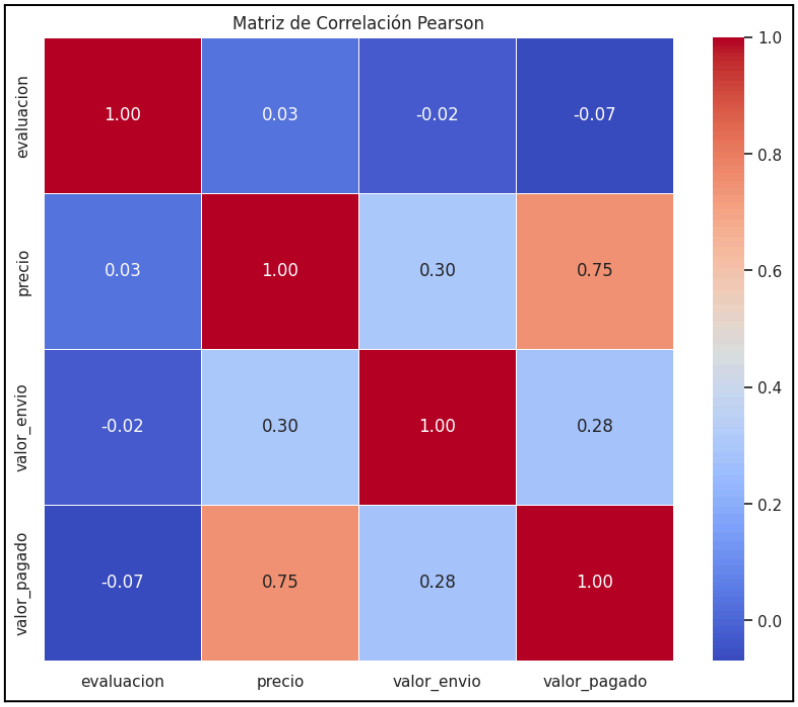
Podemos notar que solo los que tienen mayor correlación entre sí, son el precio y valor pagado. Ahora calculamos el coeficiente Pearson con la librería pingouin.

```
import pingouin as pg
# Calculamos el coeficiente de Pearson (R)
parcial = pg.partial_corr(data=df, x='precio', y='valor_pagado', covar='evaluacion')
parcial
```

	n	r	CI95	p_val
pearson	91364	0.75134	[0.75, 0.75]	0.0

El coeficiente lo calculamos considerando el precio como variable dependiente, valor pagado independiente y evaluación como covarianza. Luego realizamos la matriz de correlación Pearson con seabourn

```
plt.figure(figsize=(10,8))
sns.heatmap(correlaciones, annot=True,cmap='coolwarm',fmt=".2f",linewidths=0.5)
plt.title('Matriz de Correlación Pearson')
plt.show()
```



2.4. Regresiones Lineales

Aplicamos el modelo de regresión lineal con statsmodels. agregando una constante.

```
import statsmodels.api as sm

x=sm.add_constant(df['precio'])
y=df['valor_pagado']
model = sm.OLS(y, x).fit()
print(model.summary())
```

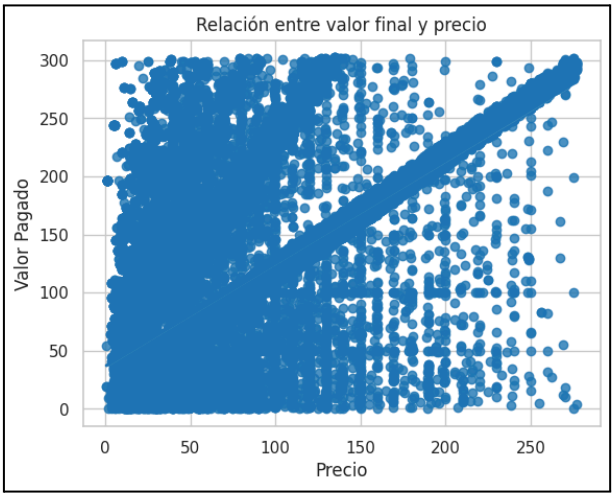
```
                            OLS Regression Results
==============================================================================
Dep. Variable:           valor_pagado   R-squared:                       0.558
Model:                            OLS   Adj. R-squared:                  0.558
Method:                 Least Squares   F-statistic:                 1.155e+05
Date:                Wed, 25 Feb 2026   Prob (F-statistic):               0.00
Time:                        01:33:17   Log-Likelihood:            -4.7343e+05
No. Observations:               91364   AIC:                         9.469e+05
Df Residuals:                   91362   BIC:                         9.469e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         36.5729      0.246    148.844      0.000      36.091      37.055
precio         0.8845      0.003    339.868      0.000       0.879       0.890
==============================================================================
Omnibus:                    32949.190   Durbin-Watson:                   0.915
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           190619.598
Skew:                           1.630   Prob(JB):                         0.00
Kurtosis:                       9.280   Cond. No.                         163.
==============================================================================
```

Con este resultado podemos obtener el R2 y R2 ajustado. El coeficiente y desviación estándar del error, el valor t y el intervalo de confianza. Luego ahora calculamos el MSE, MAE y RMSE.

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Precciones del modelo
y_pred = model.predict(x)

# Cálculo de métricas
mae= mean_absolute_error(y,y_pred)
mse= mean_squared_error(y,y_pred)
rmse= np.sqrt(mse)
r2= r2_score(y,y_pred)
```

Visualizamos información con librería seaborn.

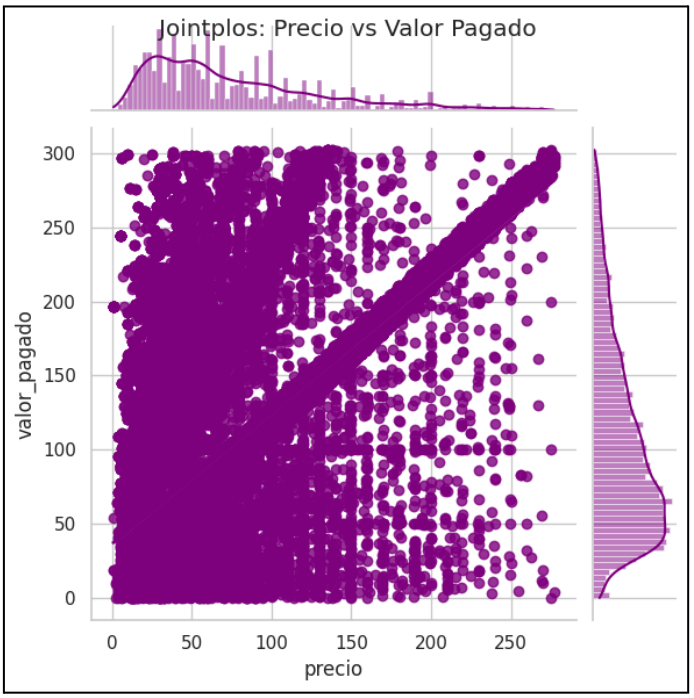
```python
sns.regplot(x='precio', y='valor_pagado', data=df)
plt.title('Relación entre valor final y precio')
plt.xlabel('Precio')
plt.ylabel('Valor Pagado')
plt.show()
```



2.5. Análisis Visual de datos

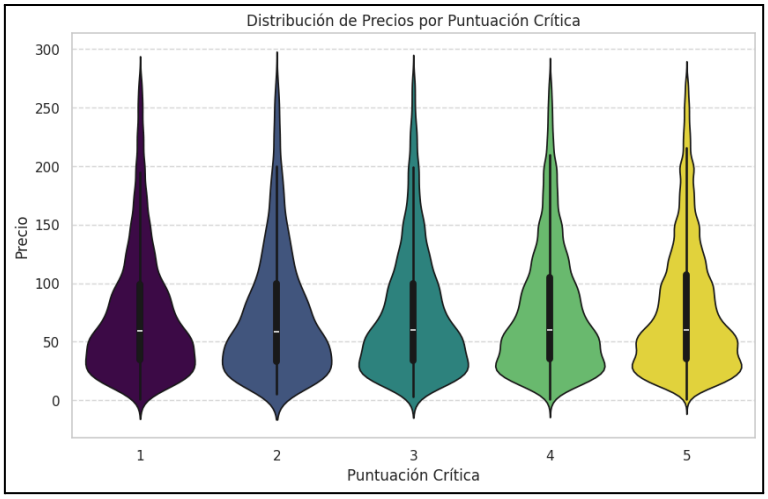
Como ya utilizamos heatmap, graficamos con joinplot utilizando variables precio y valor pagado.

```
sns.jointplot(x='precio', y='valor_pagado', data=df, kind='reg',color='purple')
plt.suptitle('Jointplos: Precio vs Valor Pagado')
plt.show()
```



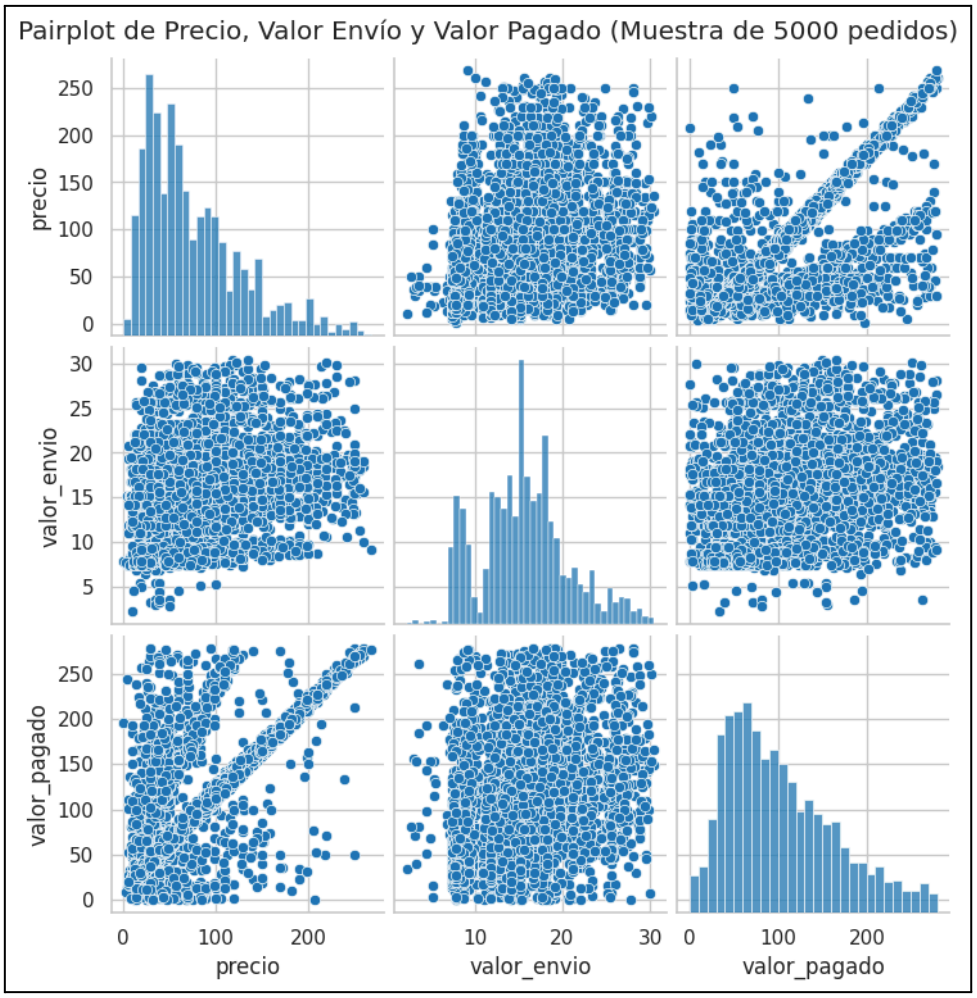
Utilizamos violinplot para visualizar la distribución de precios según la evaluación/ puntuación crítica.

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='evaluacion', y='precio', data=df, hue='evaluacion', palette='viridis'
plt.title('Distribución de Precios por Puntuación Crítica')
plt.xlabel('Puntuación Crítica')
plt.ylabel('Precio')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



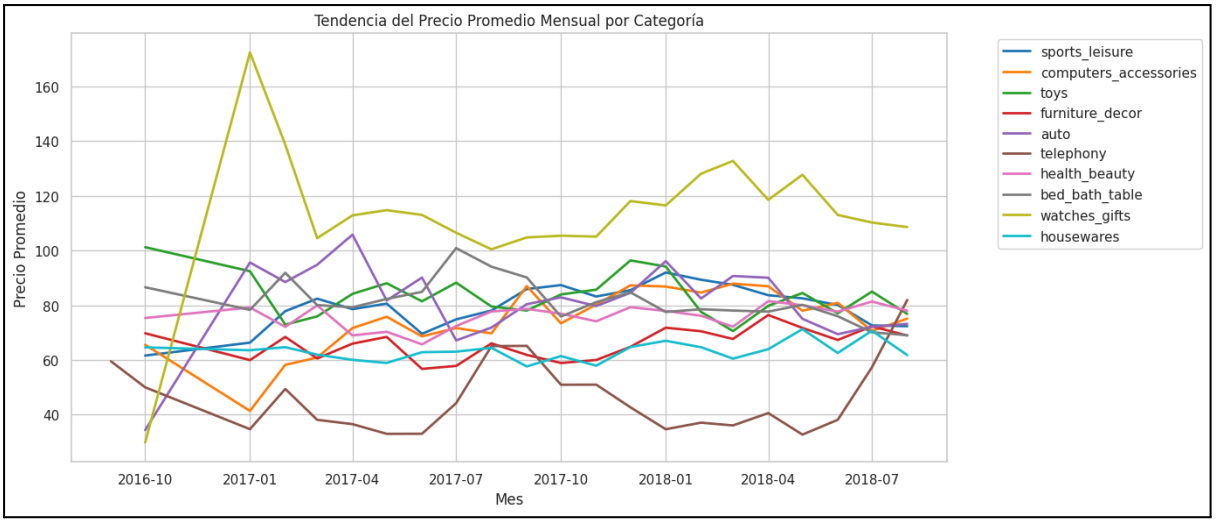
Para graficar con pairplot sobre las variables precio, valor de envío y valor pagado consideraremos sólo una muestra de 5.000 datos

```
columnas_pairplot = ['precio', 'valor_envio', 'valor_pagado']
df_sample = df[columnas_pairplot].sample(n=5000, random_state=42)
sns.pairplot(df_sample)
plt.suptitle('Pairplot de Precio, Valor Envío y Valor Pagado (Muestra de 5000 pedidos)', y=1.02)
plt.show()
```



Agrupamos por mes y graficamos la tendencia promedio mensual por categoría.

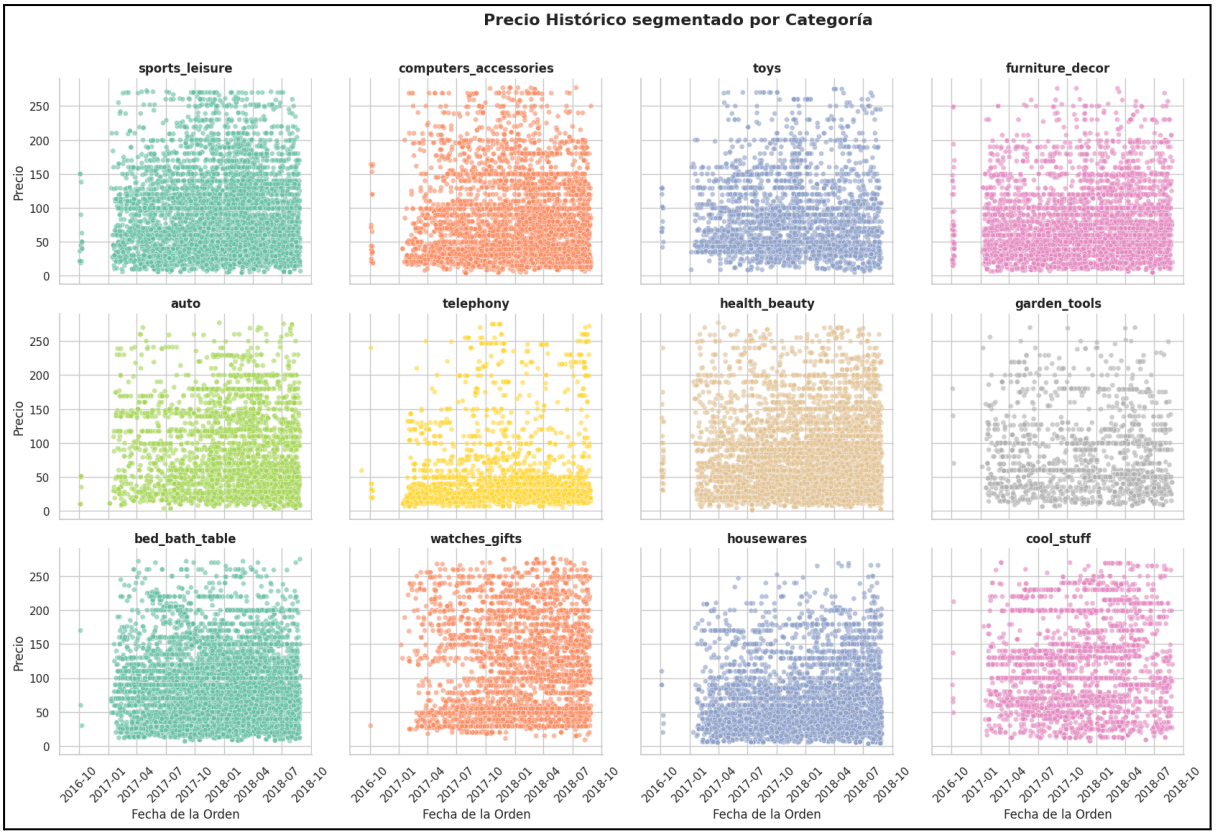
```
df_top = df_top.copy()
df_top['mes_orden'] = df_top['fecha_orden'].dt.to_period('M').dt.to_timestamp()

plt.figure(figsize=(14, 6))
sns.lineplot(
    data=df_top,
    x='mes_orden',
    y='precio',
    hue='categoria_producto_ing',
    errorbar=None,
    linewidth=2
)

plt.title('Tendencia del Precio Promedio Mensual por Categoría')
plt.xlabel('Mes')
plt.ylabel('Precio Promedio')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Ya que vimos el comportamiento de las categorías en base a la fecha y el precio. Ahora, para poder notar el comportamiento de las categorías en base a dos variables, fecha de la orden y el precio.

```
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1.0)

top_categorias = df['categoria_producto_ing'].value_counts().nlargest(12).index
df_filtrado = df[df['categoria_producto_ing'].isin(top_categorias)]

g = sns.relplot(
    data=df_filtrado,
    x='fecha_orden',
    y='precio',
    col='categoria_producto_ing',
    col_wrap=4,
    hue='categoria_producto_ing',
    kind='scatter',
    palette='Set2',
    alpha=0.6,
    s=25,
    height=3.5,
    aspect=1.2,
)

g.set_titles(col_template="{col_name}", fontweight='bold')
g.set_axis_labels("Fecha de la Orden", "Precio")

for ax in g.axes.flat:
    for label in ax.get_xticklabels():
        label.set_rotation(45)
```

```
g.fig.suptitle('Precio Histórico segmentado por Categoría', y=1.05, fontsize=16, fontweight='bold')
g._legend.remove()

plt.show()
```



Finalmente, guardamos el gráfico en formato .png.

```
g.savefig("precio_por_categoria.png", dpi=300, bbox_inches='tight')
plt.show()
```

II. Informe y Conclusiones

Para garantizar la integridad del análisis, realizamos el análisis inicial y preparación de los datos (IDA), el cual está enfocado en la limpieza y transformación de datos, y sobre la estructuración del dataset. En esta etapa se eliminaron los registros duplicados y valores nulos perjudiquen o hiciesen más engorroso el cálculo e interpretación de datos. En esta etapa clasificamos los tipos de datos según si son numéricos o categóricos, esto es fundamental para escoger el tipo de gráfico con el cual visualizamos la información.

En cuanto a la distribución de los precios mediante histogramas, detectamos una marcada concentración de ventas en rangos de precios bajos, la cual es seguida por valores atípicos (outliers). Dado que estos valores son tan extremos aplicamos el IQR para aislar y limpiar estos valores, esto con motivo que la muestra sea mucho más representativa de una compra regular.

En cuanto a los datos estadísticos podemos afirmar que tanto el precio como el valor pagado presentan una varianza y desviación estándar sumamente altas. Esto se puede deber a que 'ComercioYa' tenga productos mucho más caros que los de bajo gasto o más económicos, este motivo explicaría la gran dispersión sobre los valores.

Si nos enfocamos en la satisfacción del cliente, muestra una desviación estándar de 1.35, lo cual se puede interpretar que hay una alta variación sobre la opinión de la calidad del servicio.

El análisis y visualización de las variables categóricas y de precios y valores, podemos afirmar:

- La categoría de producto con mayor ventas es "Bed, bath and table", lo cual nos podría indicar que es el fuerte del negocio, el que concentra gran parte de las ventas.
- El peak de las ventas fue el 23 de septiembre de 2017 a las 14:56, es un dato sobre la temporada que puede generar mayores ventas, por esto debe ser considerado a nivel comercial y publicitario.

Complementando con lo relacionado a la visualización estratégica (EDA), podemos mencionar que se corrobora con el histograma/ gráfico de barras que Bed, bath & table es la categoría con mayores ventas. Además, los diagramas de caja evidencian la magnitud de los valores atípicos, incluso después de 'normalizar' más los datos. De esto podemos deducir que, a pesar de que la mayoría compra un monto bajo, existe un porcentaje de gente que compra un monto mucho más alto.

Sobre el análisis de correlaciones utilizamos el coeficiente de Pearson y visualizamos mediante un heatmap (mapa de color) y un pairplot. De esto pudimos descubrir:

- La correlación entre el precio y la reseña es cercana a cero, lo cual quiere decir que esas notas bajas no provienen necesariamente por los precios de los productos.
- Con respecto al precio y valor de envío hay una correlación positiva moderada, esto se puede interpretar como que no hay una tarifa fija, sino más bien que a mientras más costoso el producto de la tienda mayor costo de envío tendrá.

Con respecto al análisis multivariado y distribución de densidad aplicamos visualizaciones de densidad, por un lado, la concentración del ticket, el cual fue visualizado mediante un jointplot, la mayoría del gráfico se concentra en productos menores a $100 con envíos de menos $20, y por otro lado, la distribución cualitativa, a través de un gráfico con violinplot para observar el comportamiento de la distribución de precios respecto a la reseña, de esto podemos deducir que hay una leve extensión hacia precios más altos.