

Grupo 3. Entrega Final proyecto

ANA MARIA SOTO OROZCO, FELIPE DIEGO LOBATO DA SILVA, NICOLAS ALEJANDRO YEPES JOVEN y YOSSELIN NIETO GIL

1. RESUMEN SOBRE EL PROBLEMA

1.1 Contexto y Definición del Problema:

El proyecto aborda un desafío común en la industria de productos de consumo masivo: la necesidad de anticipar los precios de venta de los productos durante las distintas temporadas del año (otoño, invierno, primavera y verano). Estas variaciones estacionales afectan los hábitos de compra y el precio de los productos, lo que puede complicar la planificación estratégica de las empresas. Si no se prevén adecuadamente, estos cambios pueden provocar problemas de inventario, como exceso o falta de productos, pérdidas de ventas y costos adicionales. El objetivo de este proyecto es desarrollar un modelo que, utilizando datos históricos de ventas, estime un precio de venta óptimo por temporada, ayudando a las empresas a mejorar su planificación estratégica y eficiencia operativa.

1.2 Pregunta de Negocio y Alcance del Proyecto:

La pregunta central del proyecto es: ¿Cómo predecir el precio de venta estacional de productos de consumo masivo para optimizar la gestión de inventarios y mejorar la eficiencia operativa?

Este proyecto se limita a productos de consumo masivo vendidos a través de plataformas de e-commerce, en particular Olist Store, durante el año 2018. El modelo debe prever cuáles productos tendrán un mayor precio de venta en cada temporada, analizando patrones estacionales y de comportamiento del consumidor. La solución está dirigida a empresas que buscan ajustar sus inventarios y maximizar la disponibilidad de productos, considerando el precio estacional estimado y la demanda anticipada, para una mayor efectividad en sus estrategias comerciales.

1.3 Datos empleados y preparación inicial:

El proyecto empleará datos históricos de Olist Store, que incluyen información detallada sobre transacciones, productos, clientes, vendedores y reseñas. Las fuentes de datos principales se obtuvieron de Kaggle, complementadas con datos de códigos postales (CEP) adquiridos de un proveedor para enriquecer los análisis geoespaciales. La preparación de datos incluyó varios pasos:

- Estandarización y Transformación:** Se unificaron los formatos de los CEP y se convirtieron los archivos a Parquet para mejorar el rendimiento.
- Geolocalización:** Se integraron y ajustaron datos de geolocalización mediante un cruce de archivos y el uso de la API de Nominatim para obtener coordenadas precisas.
- Consolidación:** Los datos enriquecidos se consolidaron en un archivo optimizado para análisis espaciales, con las coordenadas ajustadas en formato de alta precisión.
- Transformación final:** Con los datos consolidados, se homologan los nombres de los campos al español para facilitar su interpretación. A continuación, utilizando el campo de fecha de compra del pedido, se filtra la base de datos para incluir únicamente el año más reciente disponible, en este caso, 2018. Posteriormente, se crea el campo temporada en función del mes de la fecha de compra, categorizando cada registro en una de las cuatro estaciones del año: otoño, invierno, verano o primavera. Esta transformación permitirá analizar y prever variaciones estacionales en los precios de venta de forma más precisa. Se obtiene una base con 63.215 registros y 33 columnas. En los anexos se detalla la base final en la Tabla de metadatos de base final.

2. MODELOS DESARROLLADOS Y SU EVALUACIÓN

Con la intención de determinar el mejor modelo para resolver las necesidades de la empresa Olist Store, se definió desarrollar un modelo que predice el precio de productos de acuerdo con la temporada en la que se compran, la ubicación del cliente, la categoría del producto y sus características físicas. Esta información resulta valiosa para la empresa como una herramienta que complementa la planeación contable, financiera y logística de la empresa, así como para la definición de estrategias de marketing ajustadas a los precios de venta de sus productos a lo largo del año.

Proceso de Entrenamiento:

- Cada modelo fue implementado dentro de un pipeline que integra preprocesamiento y entrenamiento, facilitando la escalabilidad y reproducibilidad del flujo de trabajo.
- Los datos se dividieron en conjuntos de entrenamiento y prueba utilizando una proporción de 80-20, con una semilla (`random_state=42`) para garantizar la consistencia en cada iteración.
- Se aplicó una búsqueda de hiperparámetros para cada modelo usando `GridSearchCV` o `RandomizedSearchCV`, asegurando así la selección óptima de parámetros para minimizar errores y mejorar la precisión.

Selección de Características:

- Las características utilizadas incluyen tanto atributos numéricos (como el peso, largo, altura y ancho del producto) como categóricos (temporada, categoría del producto, ciudad y estado del cliente, y el ID del producto).
- El preprocesamiento incluyó la estandarización de variables numéricas mediante `StandardScaler` y la codificación de variables categóricas con `OneHotEncoder`, facilitando una entrada balanceada y normalizada para los modelos.
- La selección de estas características se basó en su relevancia teórica para la predicción del precio del producto según factores como la estacionalidad y las propiedades físicas del producto.

Evaluación de Modelos:

- Los modelos fueron evaluados utilizando métricas de error y precisión, seleccionando la raíz del error cuadrático medio (RMSE), el error absoluto medio (MAE) y el coeficiente de determinación (R^2) para capturar diferentes aspectos del rendimiento.
- Se aplicó validación cruzada para obtener un RMSE promedio que refleje la robustez del modelo en diferentes particiones del conjunto de datos, disminuyendo así la posibilidad de overfitting o underfitting.
- Los modelos probados incluyen `XGBRegressor`, `RandomForestRegressor`, `LinearRegression`, `ElasticNet` y `KNeighborsRegressor`. En términos de rendimiento, el modelo de regresión lineal obtuvo el mejor RMSE (118.01) y el mayor R^2 (0.6336), lo cual indica que es el modelo más adecuado para este caso (Documentación de entren...)(modelos).

1. **Modelo con `XGBRegressor`:** Se implementa un pipeline que incluye preprocesamiento y el modelo `XGBRegressor`. Se realiza una búsqueda de hiperparámetros con `GridSearchCV`.
 - **Parámetros óptimos:** `{'colsample_bytree': 0.8, 'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 300, 'reg_alpha': 0.01, 'subsample': 0.8}`
 - **Métricas:**
 - **RMSE:** 132.67, **MAE:** 54.75, **R^2 :** 0.5368, **RMSE (validación cruzada):** 127.56
2. **Modelo con `RandomForestRegressor`:** Similar al anterior, este pipeline utiliza `RandomForestRegressor` y `RandomizedSearchCV` para ajustar hiperparámetros.
 - **Parámetros óptimos:** `{'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None, 'bootstrap': False}`
 - **Métricas:**
 - **RMSE:** 135.53, **MAE:** 43.09, **R^2 :** 0.5167, **RMSE (validación cruzada):** 132.04
3. **Modelo con `LinearRegression`:** Modelo lineal básico, sin ajuste de hiperparámetros. Incluye preprocesamiento mediante un pipeline.
 - **Métricas:**
 - **RMSE:** 118.01, **MAE:** 25.55, **R^2 :** 0.6336, **RMSE (validación cruzada):** 114.60

4. **Modelo con ElasticNet:** Modelo de regresión penalizada (ElasticNet) que también se entrena en pipeline.
 - **Métricas:**
 - **RMSE:** 183.53, **MAE:** 81.10, **R²:** 0.1137, **RMSE (validación cruzada):** 171.00
5. **Modelo con KNeighborsRegressor:** Modelo KNeighborsRegressor con ajuste de hiperparámetros mediante GridSearchCV.
 - **Parámetros óptimos:** {'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
 - **Métricas:**
 - **RMSE:** 142.54, **MAE:** 39.61, **R²:** 0.4654, **RMSE (validación cruzada):** 137.98

Modelo con Mejor Desempeño

Basado en el RMSE y el R², el modelo de **regresión lineal (LinearRegression)** tiene el **mejor desempeño** con un **RMSE de 118.01** y un **R² de 0.6336**. Este modelo proporciona las predicciones más precisas en este conjunto de pruebas y validación cruzada.

3. OBSERVACIONES Y CONCLUSIONES SOBRE LOS MODELOS

Para responder a la pregunta de negocio de predecir los precios estacionales de productos de consumo masivo, decidimos centrarnos en el modelo de regresión lineal, que había demostrado un buen rendimiento en la fase inicial descrita en el punto 2. Con este modelo, creamos un experimento en MLflow específicamente para monitorear y comparar varias configuraciones y ajustes, asegurando así que podríamos determinar la mejor versión para nuestro caso de uso. Este experimento fue fundamental, ya que nos permitió registrar y visualizar los resultados de cada variación en un entorno controlado y reproducible, proporcionando una comprensión clara de cómo cada ajuste afectaba el desempeño del modelo.

Dentro de este experimento, generamos tres versiones distintas del modelo, que correspondieron a tres códigos diferentes, cada uno enfocado en una configuración específica de preprocesamiento y ajuste de hiperparámetros para maximizar la precisión y la eficiencia. A continuación, detallamos cada uno de estos códigos y los resultados obtenidos:

Código 1: Este código representa la versión inicial y más básica del modelo de regresión lineal, que fue creada originalmente en la sección de modelos descrita anteriormente. Sin embargo, para este experimento, se adaptó para ejecutarse en el entorno de Databricks e integrarse con MLflow para un seguimiento sistemático. En esta configuración, utilizamos un pipeline simple que incluía la normalización de las variables numéricas (peso, dimensiones del producto) y la codificación one-hot para variables categóricas (temporada, categoría del producto, entre otras). Este enfoque buscaba capturar las relaciones principales entre las variables sin añadir complejidad excesiva al modelo. Los resultados obtenidos para esta versión fueron prometedores, con un RMSE de 118.01, MAE de 25.55 y un R² de 0.63, además de una validación cruzada que mostró un RMSE promedio de 114.60, indicando robustez y consistencia. Este rendimiento destacado convirtió a esta versión en una fuerte candidata para ser la solución final.

Código 2: La segunda versión del modelo buscó explorar un preprocesamiento más sofisticado de las variables categóricas, con el fin de mejorar la captura de patrones latentes. Aquí, decidimos combinar dos técnicas de codificación: OneHotEncoder para algunas variables categóricas con menor número de categorías, y TargetEncoder para variables como la ciudad del cliente y el identificador del producto, que tienen un mayor número de categorías y podrían beneficiarse de una codificación basada en la media del objetivo (precio). Este enfoque introdujo mayor flexibilidad al manejar variables de alta cardinalidad, permitiendo que el modelo capturara mejor las sutilezas de cada categoría. Aunque esta

configuración no trajo una mejora significativa en los resultados comparada con la primera versión, añadió un valor teórico y práctico importante, mostrando su utilidad especialmente en casos donde hay gran variabilidad en las categorías.

Código 3: La tercera versión fue un intento de optimizar el modelo mediante la selección de características. En esta versión, incorporamos el método SelectKBest con una función de puntuación basada en regresión (`f_regression`) para seleccionar las cinco variables más relevantes para la predicción del precio. La motivación detrás de este enfoque fue reducir el ruido de variables menos significativas, optimizando el modelo tanto en términos de rendimiento como de eficiencia computacional. Los resultados de esta versión fueron similares a los de las versiones anteriores en términos de RMSE y R^2 , pero la reducción de complejidad computacional la convierte en una opción interesante para aplicaciones que requieren un tiempo de procesamiento más rápido.

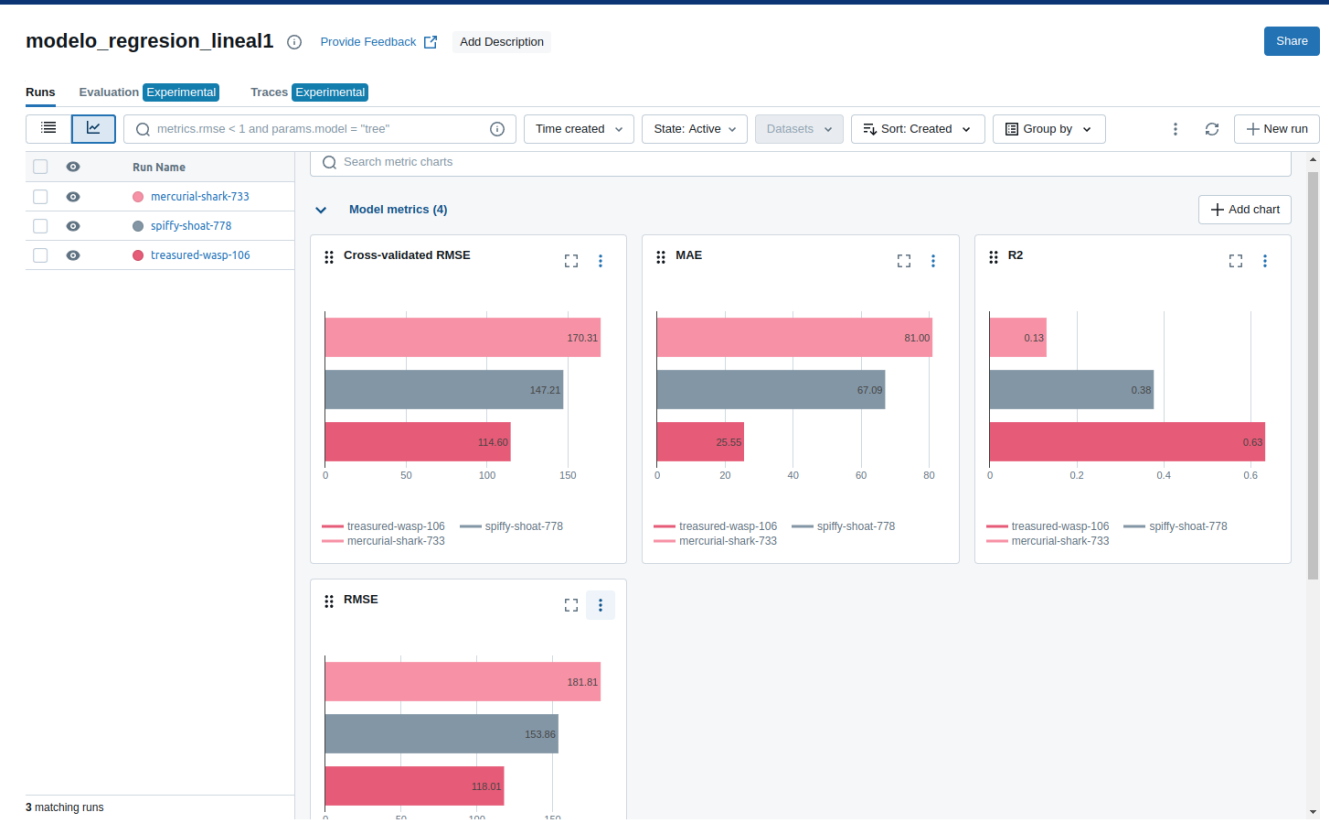


Figura 1 - Resultados del experimento de distintas configuraciones del modelo de regresión lineal en MLflow. La imagen muestra una comparación de rendimiento entre las tres versiones del modelo, destacando la configuración con mejor desempeño en términos de precisión y consistencia.

Comparación y Resultados Consolidados en MLflow

La visualización de los resultados en MLflow nos permitió realizar un análisis comparativo claro entre las tres versiones del modelo. Observamos que, entre todas las versiones, el Código 1 mantuvo el mejor equilibrio entre precisión y simplicidad. Con un RMSE de 118.01, MAE de 25.55, y una validación cruzada con RMSE promedio de 114.60, destacó como el modelo más consistente y eficaz para el objetivo de predicción. El uso de MLflow no solo facilitó el registro detallado de las métricas, sino que también permitió un análisis visual y objetivo de los resultados, confirmando la robustez y eficiencia del Código 1 en comparación con las otras versiones.

Conclusión

Con base en los resultados de las tres versiones, concluimos que la primera configuración, representada por el Código 1, es la más adecuada para el proyecto. Este modelo de regresión lineal, ahora adaptado al entorno de Databricks y gestionado a través de MLflow, cumple con los requisitos de precisión y simplicidad, además de permitir un fácil seguimiento de versiones y métricas para futuras mejoras. La experiencia con MLflow fue fundamental para el éxito del experimento, permitiéndonos documentar y rastrear cada etapa del proceso de manera eficiente y clara. Así, el Código 1 se posiciona como la elección final para implementación, atendiendo a los objetivos de predicción de precios estacionales de forma precisa y práctica.

4. TABLERO DESARROLLADO

El tablero desarrollado busca ofrecer la solución propuesta, a través de 2 páginas.

La página 1, vinculada a la base de datos final (base_dash.csv), permite al usuario explorar los datos relacionados con precios de venta y demanda de categorías de productos mediante los diferentes gráficos. Esta página contiene, seleccionadores donde se puede escoger la fecha, la temporada (invierno, verano, otoño y primavera) y también una categoría de producto, al realizar las selecciones solicitadas (ver manual de usuario Anexo 4 de este documento) se obtendrán datos sobre la demanda por temporada para cada una de las categorías a través de la tabla interactiva, también se obtendrá el top 10 de categorías más vendidas con sus precios estimados correspondientes. En la parte inferior de esta página el usuario podrá observar la ubicación geográfica de los clientes que generan mayor demanda de la categoría seleccionada para la temporada y al lado derecho del mapa, un rango de los precios históricos para esa misma selección.



Ilustración 1 página 1 del tablero parte superior A.

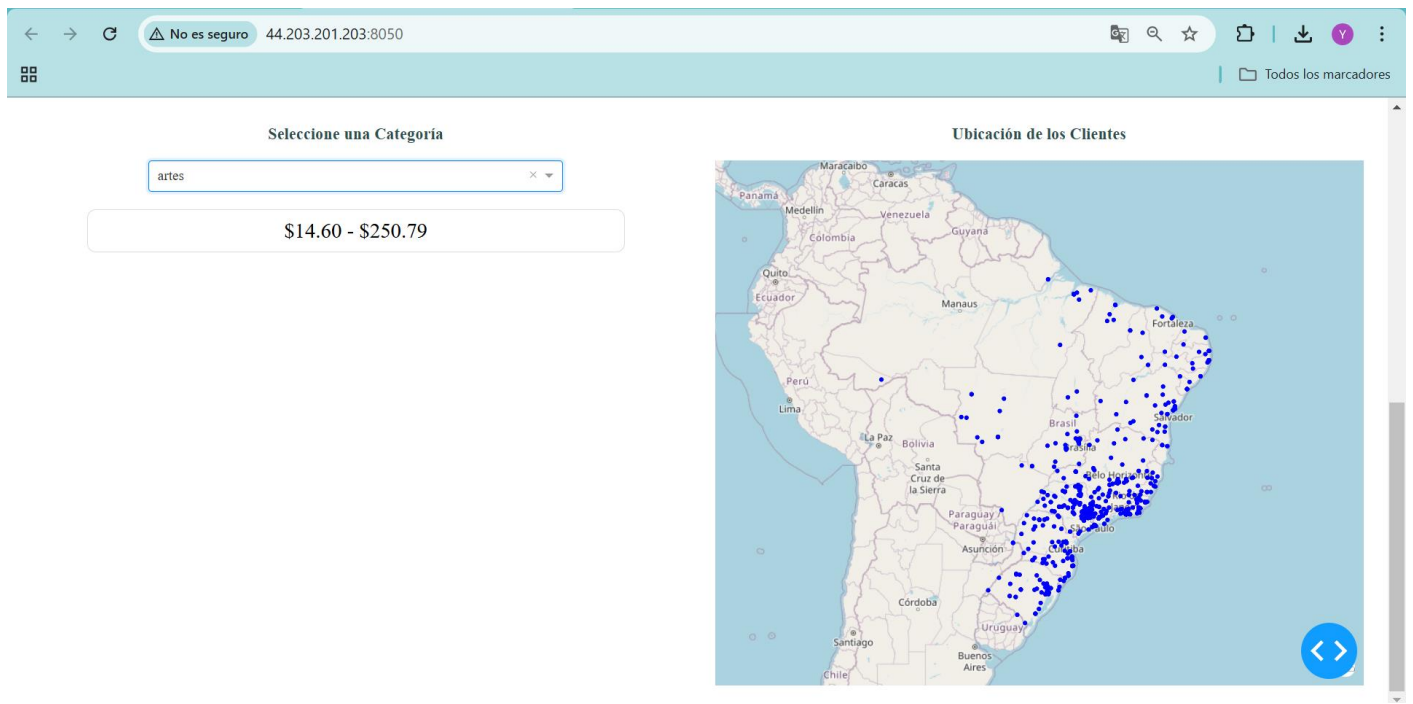


Ilustración 2 Parte inferior de la página 1 del tablero. B.

En la página 2 se encuentra la sección de predicciones, utilizando el modelo seleccionado como el mejor según lo indicado en el ítem 3. El modelo fue empaquetado en un formato PKL (modelo_regresion_lineal.pkl) para integrarlo al tablero. Esta página le permite al usuario seleccionar características específicas de su interés y así ver el precio estimado del producto, de acuerdo con la predicción del modelo. El usuario debe seleccionar la temporada, la ciudad del cliente, la categoría y el código de identificación del producto en la plataforma de Olist para obtener las predicciones.

The screenshot shows a web browser window with the address bar displaying "44.203.201.203:8050/predicciones". The page has a light blue header with navigation icons and a "Todos los marcadores" (All bookmarks) link. Below the header, there are two links: "Análisis de Ventas" and "Predicciones de Precio Medio Promedio". The main content area is titled "Predicciones de Precio Medio Promedio". Below this title, there is a heading "Selecciona las características para el cálculo del Precio Promedio". There are four dropdown menus arranged in a 2x2 grid. The top-left dropdown is labeled "Selecciona una Temporada:" and shows "Primavera". The top-right dropdown is labeled "Selecciona una Categoría de Producto:" and shows "alimentos". The bottom-left dropdown is labeled "Selecciona una Ciudad Cliente:" and shows "abelardo luz". The bottom-right dropdown is labeled "Selecciona un ID Producto:" and shows "001b237c0e9bb435f2e54071129237e9". Below these dropdowns, there is a large white box with a black border containing the text "El Precio Promedio Predicho es: \$88.38". A blue double-arrow button is located at the bottom right of the page.

Ilustración 3 vista de la Página 2 del tablero

Finalmente, se puede decir que el tablero logra las metas inicialmente propuestas en el proyecto, brindando al usuario insumos para predecir el precio de venta estacional de productos de consumo masivo. Con esto, el usuario puede optimizar la gestión de inventarios y mejorar la eficiencia operativa.

En el manual de usuario correspondiente al anexo 4, se encuentran los detalles de funcionalidad y uso del tablero con las imágenes de este. El enlace para acceder al tablero es <http://44.203.201.203:8050/> , también relacionado con los demás enlaces de interés en el anexo 2.

5. Responsabilidades y Contribuciones de los Integrantes

Ana María Soto Orozco:

Asumió el rol de coordinadora del equipo, organizando las tareas y centralizando cada una de las entregas del proyecto. Su trabajo incluyó la realización del análisis exploratorio de los datos, lo que permitió comprender la estructura y características principales de la información. También fue responsable de la limpieza y procesamiento de los datos, asegurando que estuvieran en condiciones óptimas para su uso en el modelado. También realizó el cruce entre las diferentes bases de datos para generar una base final con la información requerida, que sirvió como insumo para las etapas siguientes. Adicionalmente, colaboró en las actividades de modelación junto con Nicolás, aportó en la versión final del tablero.

Nicolás Alejandro Yepes Joven:

Fue el encargado de la fase de modelación del proyecto. Su trabajo incluyó la creación de modelos de aprendizaje automático, la evaluación de métricas de desempeño, y la comparación entre distintos enfoques para seleccionar el más adecuado. Realizó una exhaustiva búsqueda de hiperparámetros para optimizar el mejor modelo y llevó a cabo pruebas fuera de tiempo para asegurar la generalización del modelo seleccionado. Gracias a sus aportes, se logró definir una solución analítica robusta y alineada con los objetivos del proyecto. Adicionalmente aportó en la versión final del proyecto.

Felipe Diego Lobato da Silva:

Fue el responsable de establecer la infraestructura para el despliegue de los modelos. Implementó la estructura en MLflow para facilitar el seguimiento de los experimentos y creó la instancia en AWS necesaria para el despliegue del proyecto juntamente con readecuación de los scripts en el formato exigido por el mlflow para la creación de varios modelos adentro del experimento. Adicionalmente trabajó en la construcción y el despliegue de la versión final del tablero.

Yoselin Nieto Gil:

colaboró en la etapa de exploración de la base de datos, liderando la fase de visualización y también documentación del tablero en los manuales. Se aseguró de que el tablero cumpliera con todos los requisitos planteados y de que presentara los resultados de forma clara y útil para los usuarios finales. Asimismo, fue la revisora final de la documentación, verificando la coherencia y completitud del reporte para garantizar que reflejara adecuadamente el trabajo del equipo y los hallazgos del proyecto.

Coordinación y Reuniones de Equipo

El equipo mantuvo reuniones periódicas para controlar el avance de cada tarea, analizar los resultados intermedios, y asegurar la cohesión en cada fase del proyecto. En estas sesiones, se compartieron avances, se resolvieron dudas, y se sacaron conclusiones sobre el desarrollo del trabajo. Todos los integrantes documentaron sus tareas para facilitar la trazabilidad del proyecto y enriquecer la documentación final.

6. Anexos

Anexo 1. Pantalla Inicial del repositorio General

The screenshot shows the GitHub repository page for 'despliegue_analytica' by 'felipelobatodasilva'. The repository is public and has 77 commits. The file list includes:

File	Description	Last Commit
.dvc	Confirmación de todos los cambios	last month
.ipynb_checkpoints	Añadir modelos	3 weeks ago
app-dash-olist-prediction	Agregando requirements.txt	yesterday
documentacion	app olist-dash_main	yesterday
mlflow-modelos	Agregando archivos para el mlflow databricks en la carpeta ...	2 weeks ago
.dvcignore	Inicializando DVC en el repositorio	last month
.gitignore	Eliminando carpeta venv desde adentro del repo	last week
01 - Conversión de CSV a Parquet.ipynb	Añadiendo los archivos restantes al control de versiones en ...	last month
02 - Transformación de Datos y Unión de Geol...	Añadiendo los archivos restantes al control de versiones en ...	last month
03 - Recuperación de Coordenadas de Geoloc...	Creando carpeta para la documentacion del proyecto e inici...	last month
04 - Enriquecimiento de Geolocalización con C...	Actualizando el DVC	last month
05 - Exploración de datos.ipynb	Actualizando el archivo 05	3 weeks ago
06 - Transformacion_Final.ipynb	Agregando transformacion finales para tablero y requiremne...	last week
07 - modelos.ipynb	Agregando base_dash.csv al repo	last week

The screenshot shows the GitHub repository page for 'despliegue_analytica' by 'felipelobatodasilva'. The repository is public and has 77 commits. The file list includes:

File	Description	Last Commit
app-dash-olist-prediction	Agregando requirements.txt	yesterday
documentacion	app olist-dash_main	yesterday
mlflow-modelos	Agregando archivos para el mlflow databricks en la carpeta ...	2 weeks ago
.dvcignore	Inicializando DVC en el repositorio	last month
.gitignore	Eliminando carpeta venv desde adentro del repo	last week
01 - Conversión de CSV a Parquet.ipynb	Añadiendo los archivos restantes al control de versiones en ...	last month
02 - Transformación de Datos y Unión de Geol...	Añadiendo los archivos restantes al control de versiones en ...	last month
03 - Recuperación de Coordenadas de Geoloc...	Creando carpeta para la documentacion del proyecto e inici...	last month
04 - Enriquecimiento de Geolocalización con C...	Actualizando el DVC	last month
05 - Exploración de datos.ipynb	Actualizando el archivo 05	3 weeks ago
06 - Transformacion_Final.ipynb	Agregando transformacion finales para tablero y requiremne...	last week
07 - modelos.ipynb	Agregando base_dash.csv al repo	last week
README.md	Añadiendo los archivos restantes al control de versiones en ...	last month
archivos_base.dvc	Actualizando el DVC después de mover archivos a la carpeta...	last month
files_csv.dvc	Agregando el base_dash.csv	last week
files_parquet.dvc	Adiciona df_basefinal.parquet ao DVC	2 weeks ago
olist.db	Añadir modelos	3 weeks ago

The README section shows the repository name 'despliegue_analytica'. The suggested workflows section includes:

- Django: Build and Test a Django Project
- Python Package using Anaconda

Anexo 2. Enlaces relacionados con el proyecto:

Aplicación mlflow - <http://3.84.19.167:8050/>

Ruta de los scripts del modelo: [GitHub – felipelobatodasilva/despliegue_analytica/mlflow-modelos](https://github.com/felipelobatodasilva/despliegue_analytica/mlflow-modelos)

Ruta del repo general con las fuentes del tablero y API desarrollados –:

https://github.com/felipelobatodasilva/despliegue_analytica

https://github.com/felipelobatodasilva/despliegue_analytica

Ruta del tablero en línea: <http://44.203.201.203:8050/>

Ruta al video de presentación del proyecto: <https://www.youtube.com/watch?v=qH2BSQh8zAU>

Anexo 3. Tabla de metadatos de base final

Nombre de la Columna	Descripción	Tipo de Dato	Ejemplo del Dato	Clave
id_pedido	Identificador único del pedido	string	"995392413cee616c1..."	Clave primaria
id_cliente	Identificador único del cliente	string	"4bf2490c4245cdb25a..."	Clave foránea (customers.customer_id)
estado_pedido	Estado del pedido (ej. entregado, cancelado)	string	"entregado"	
fecha_pedido	Fecha en que se realizó la compra	timestamp	"2017-09-04"	
hora_compra_pedido	hora en que se realizó la compra	string	"22:43:54"	
fecha_aprobacion_pedido	Fecha y hora en que se aprobó el pedido	timestamp	"2017-09-04 22:43:54"	
fecha_entrega_estimada	Fecha estimada de entrega del pedido	timestamp	"2017-09-27 00:00:00"	
id_item_pedido	Identificador del ítem dentro del pedido	integer	1	Clave primaria compuesta junto a id_pedido
id_producto	Identificador único del producto	string	"4lebbrb7a41c44632..."	Clave foránea (products.product_id)
id_vendedor	Identificador único del vendedor	string	"7a67c85e85bbc2e85..."	Clave foránea (sellers.seller_id)

fecha_limite_envio	Fecha límite para el envío	timestamp	"2017-05-22 16:05:44"	
precio	Precio del producto	doble	109.99	
valor_flete	Valor del envío	doble	18.02	
secuencia_pago	Secuencia del pago dentro del pedido	integer	1	Clave primaria compuesta junto a id_pedido
tipo_pago	Tipo de pago utilizado	string	"tarjeta_credito"	
cuotas_pago	Número de cuotas del pago	integer	8	
valor_pago	Valor del pago	doble	99.33	
id_unico_cliente	Identificador único y persistente del cliente	string	"15ee900ec703c9a10"	
codigo_postal_cliente	Prefijo del código postal del cliente	string	"68590"	Clave foránea (geo.cep_prefix)
ciudad_cliente	Ciudad del cliente	string	"jacunda"	
estado_cliente	Estado del cliente	string	"PA"	
nombre_categoria_producto	Categoría del producto	string	"perfumaria"	
longitud_nombre_producto	Longitud del nombre del producto	integer	40	
longitud_descripcion_producto	Longitud de la descripción del producto	integer	287	
cantidad_fotos_producto	Cantidad de fotos del producto	integer	1	
peso_producto_g	Peso del producto en gramos	integer	225	

largo_producto_cm	Longitud del producto en centímetros	integer	16	
altura_producto_cm	Altura del producto en centímetros	integer	10	
ancho_producto_cm	Ancho del producto en centímetros	integer	14	
codigo_postal_vendedor	Prefijo del código postal del vendedor	string	"13023"	Clave foránea (geo.cep_prefix)
ciudad_vendedor	Ciudad del vendedor	string	"campinas"	
estado_vendedor	Estado del vendedor	string	"SP"	
temporada	Estación del año de acuerdo al mes de compra del pedido (Otoño, Invierno, Verano, Primavera)	string	"Verano"	

Anexo 4. Manual de usuario

El tablero contiene dos páginas de interacción: Análisis de ventas y Predicciones precio promedio



Ilustración 4 vista inicial del Dash con las dos ventanas a seleccionar

Página 1 Análisis de ventas

Contiene análisis de datos históricos con la base de datos-Exploración. Se selecciona una temporada y el rango de fechas en las barras de selección



Ilustración 5 área superior de la ventana analisis de ventas

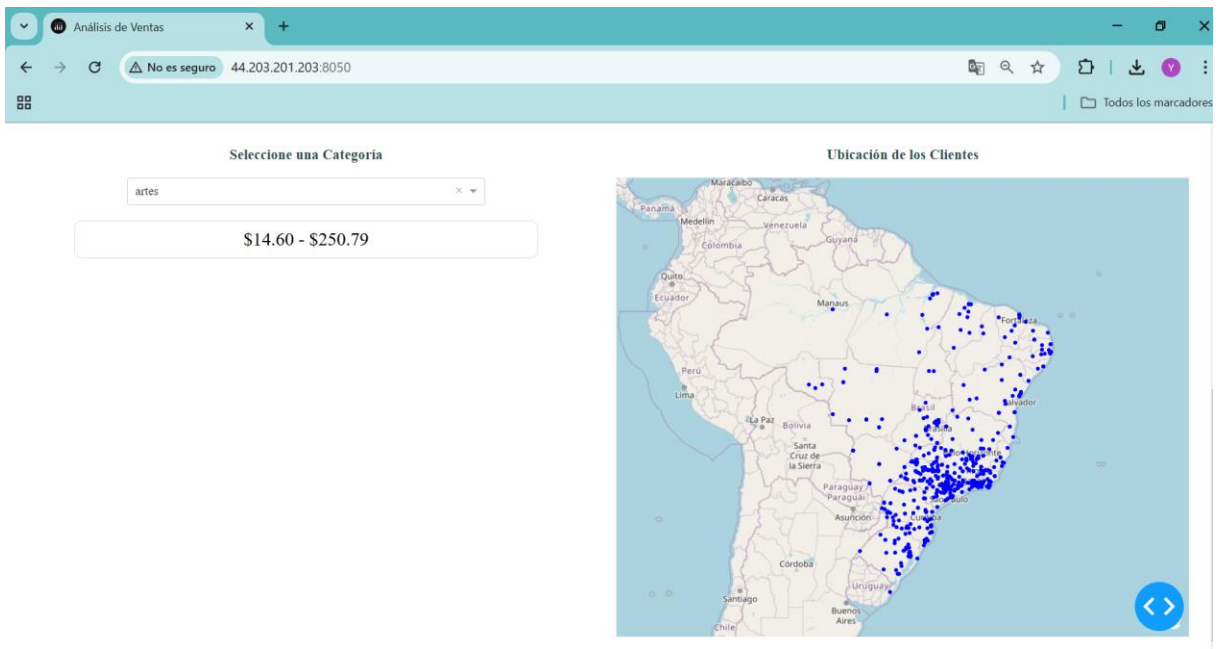


Ilustración 6 Parte inferior de la ventana análisis de ventas.

Paso 1. El usuario debe seleccionar la temporada

The screenshot shows a web browser window with the title 'Análisis de Ventas'. The address bar displays 'No es seguro' and the IP address '44.203.201.203:8050'. The main content area has a header 'Análisis de Ventas Olist Ecommerce'. Below the header, there is a section for date selection. On the left, there is a dropdown menu currently showing 'Invierno'. On the right, there is a date range selector with the text 'Seleccione un Intervalo de Fechas' and the dates '27/08/2018' and '03/09/2018'.

Ilustración 7 Barra de selección

Al seleccionar la temporada, en la tabla “predicciones de Demanda por categoría” aparecen las categorías de los productos presentes en la base de datos y se muestra la demanda (baja, media y alta) de los consumidores para esa temporada

nombre_categoria_producto	demanda
cama_mesa_banho	Alta
beleza_saude	Alta
informatica_acessorios	Alta
esporte_lazer	Alta
moveis_decoracao	Alta
utilidades_domesticas	Alta
relogios_presentes	Alta

Ilustración 8 tabla de demanda del producto

Simultáneamente se genera el gráfico “Top 10 Categorías de Mayor demanda” para la misma temporada y fechas seleccionadas en las barras de la parte superior del tablero.

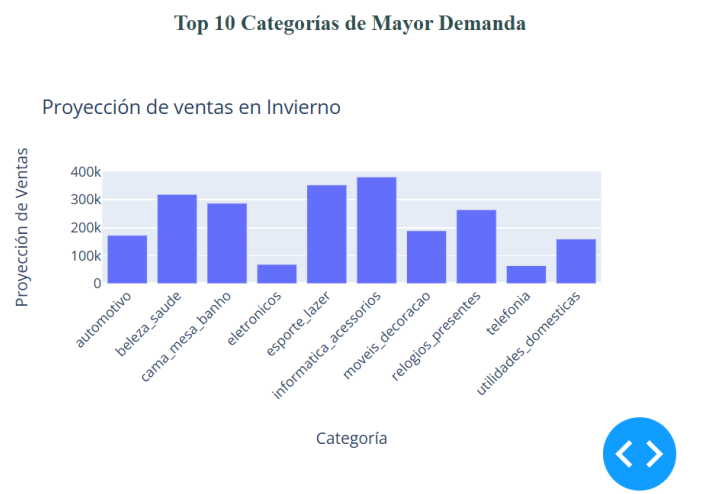


Ilustración 9 gráfico top 10 categorías

Debajo del gráfico, se indica la ubicación espacial de los compradores, donde el usuario del tablero podrá observar donde se concentran espacialmente las ventas en el mapa “Ubicación de los clientes”

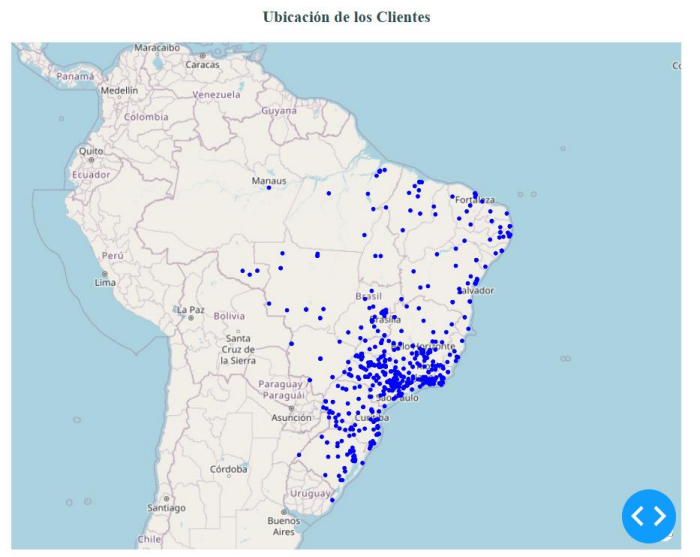


Ilustración 10 mapa ubicación de los clientes

Paso 2. Seleccione la categoría para el rango de precio

En la parte inferior izquierda se encuentra otro seleccionador “seleccione una categoría” el usuario debe dar clic en la flecha para desplegar las opciones de categoría del producto, seleccionar la de su interés y obtendrá un intervalo de precio estimado para esa categoría en la temporada.

Seleccione una Categoría

agro_industria_e_comercio

\$12.99 - \$1899.01

Ilustración 11 seleccionador e intervalo de precios de la categoría.

Página 2. Predicciones precio promedio

Dar clic en predicciones de Precio promedio para pasar a la página 2

En esta página el usuario selecciona una opción para cada una de las características específicas de su interés en 4 listas desplegables para: Temporada, Categoría del producto, ciudad del cliente y la identificación del producto.

Ilustración 12 seleccionadores de características.

Seleccione la Temporada: se refiere a la temporada del año relacionada (primavera, verano, otoño e invierno). Debe seleccionar una.

Seleccione la Categoría del producto: se debe seleccionar una de acuerdo con las categorías disponibles en la base de datos, por ejemplo: "perfumaria"

Seleccione la Ciudad cliente: Se refiere a la ubicación del cliente de acuerdo con las ciudades disponibles en la base de datos, por ejemplo: Sao Paulo.

Seleccione el ID del producto: Se refiere a la identificación única del producto, un código de letras y números. Por ejemplo: "4lebbrb7a41c44632...". El usuario debe seleccionar la correspondiente a su producto de interés de acuerdo con la base de datos.

Al terminar de seleccionar las 4 características, el modelo determinará un precio predicho que se mostrará en el tablero como se puede ver a continuación:



Ilustración 13 Predicción del precio con el modelo.

Anexo 5. Manual de Instalación del tablero

Paso 1. Crear la EC2

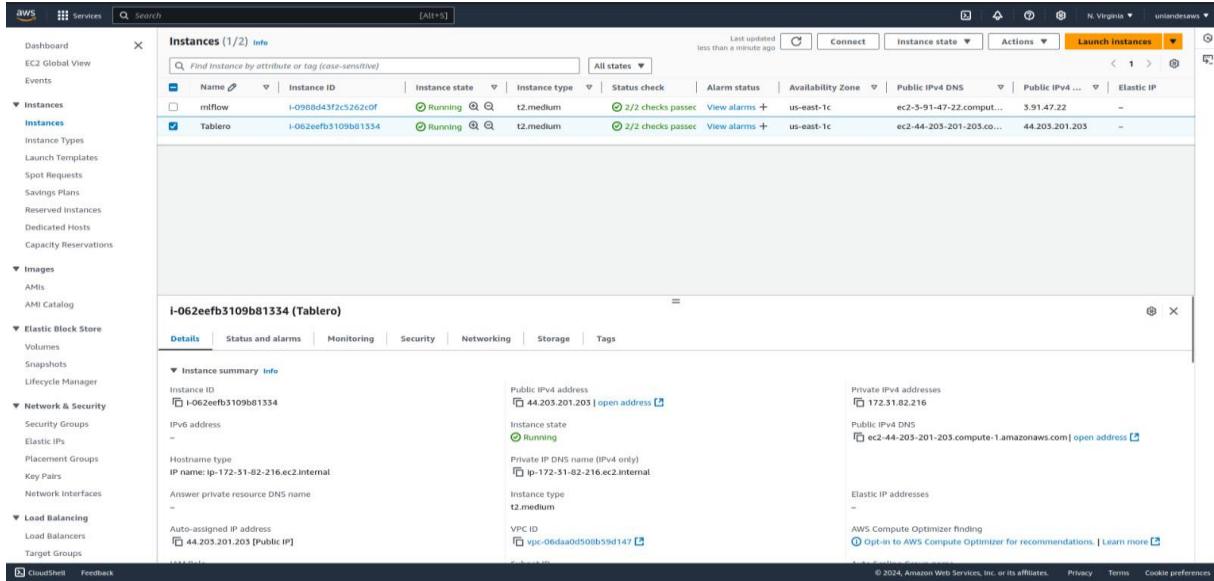


Ilustración 14 Datos de la instancia creada.

Para ejecutar el dashboard, es necesario crear una instancia EC2 en AWS. En este caso, se configuró una instancia del tipo **t2. medium**, que ofrece un buen equilibrio entre costo y rendimiento. La instancia se lanzó en la región **us-east-1**, lo que garantiza proximidad geográfica y menor latencia. Además, se asignó una dirección IPv4 pública, como **44.203.201.203**, para permitir el acceso externo al dashboard. Durante la configuración, también se habilitó el puerto **8050** en el grupo de seguridad para permitir el tráfico necesario para la aplicación.

Una vez creada y configurada la instancia EC2, conéctese a ella mediante SSH utilizando el siguiente comando en la terminal:

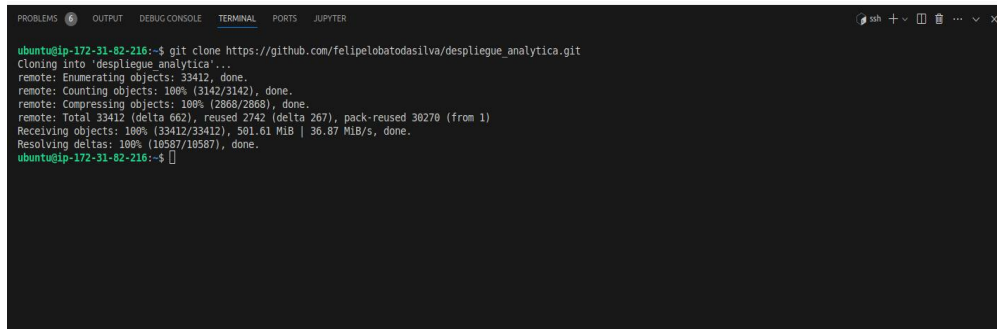
```
ssh -i [tu-clave.pem] ubuntu@44.203.201.203
```

Paso 2. Clonar el repo

Una vez conectados a la instancia EC2 vía SSH, es necesario clonar el repositorio que contiene el código del dashboard. Este repositorio incluye todos los archivos y scripts necesarios para la implementación de la aplicación.

Para clonar el repositorio, ejecute el siguiente comando en la terminal:

```
git clone https://github.com/felipelobatodasilva/despliegue_analytica.git
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
ubuntu@ip-172-31-82-216:~$ git clone https://github.com/felipelobotodasilva/despliegue_analytica.git
Cloning into 'despliegue_analytica'...
remote: Enumerating objects: 33412, done.
remote: Counting objects: 100% (3142/3142), done.
remote: Compressing objects: 100% (2868/2868), done.
remote: Total 33412 (delta 662), reused 2742 (delta 267), pack-reused 38278 (from 1)
Receiving objects: 100% (33412/33412), 301.61 MiB | 36.87 MiB/s, done.
Resolving deltas: 100% (10587/10587), done.
ubuntu@ip-172-31-82-216:~$
```

Ilustración 15 Clonar el repositorio en la máquina virtual

Paso 3. Alistar paquetes y bibliotecas que se exige para la aplicación

Una vez clonado el repositorio en la instancia EC2, es necesario instalar las dependencias especificadas en el archivo requirements.txt. Esto asegura que el dashboard cuente con todas las bibliotecas necesarias para su correcto funcionamiento.

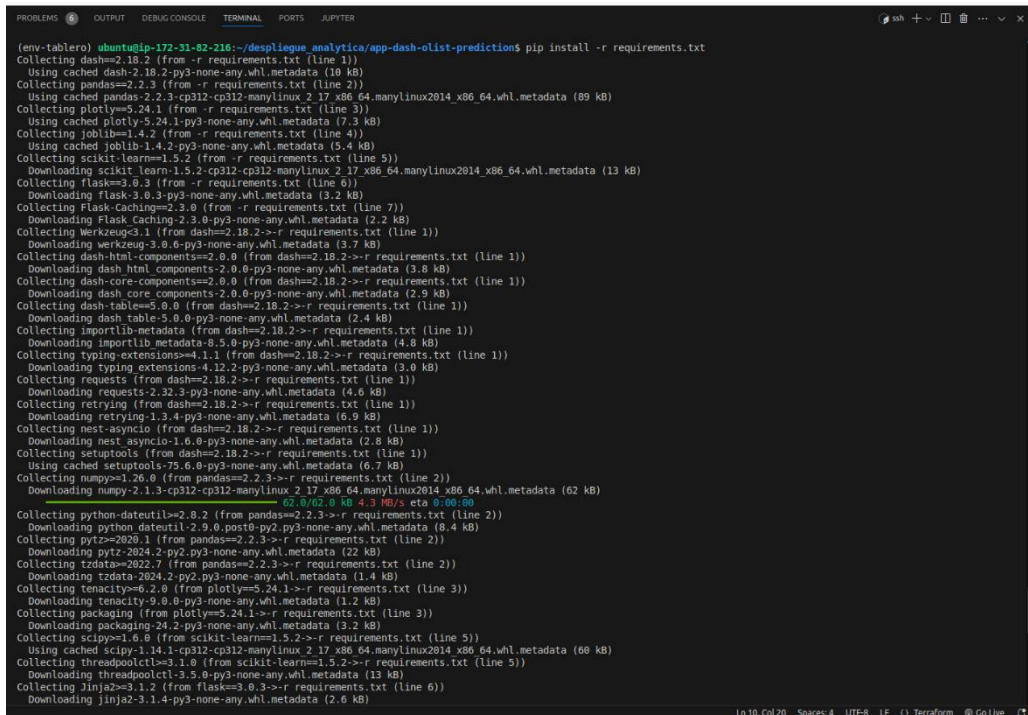
Navegue al directorio del proyecto:

cd despliegue_analytica/app-dash-olist-prediction

Ejecute el siguiente comando para instalar las dependencias:

pip install -r requirements.txt

Durante la instalación, se descargan e instalan automáticamente las bibliotecas listadas en el archivo, como dash, pandas, y numpy. El proceso muestra en la terminal los detalles de cada paquete instalado y la velocidad de descarga.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
(env:tablero) ubuntu@ip-172-31-82-216:~/despliegue_analytica/app-dash-olist-prediction$ pip install -r requirements.txt
Collecting dash==2.18.2 (from -r requirements.txt (line 1))
Using cached dash-2.18.2-py3-none-any.whl.metadata (10 kB)
Collecting pandas==2.2.3 (from -r requirements.txt (line 2))
Using cached pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (89 kB)
Collecting plotly==5.24.1 (from -r requirements.txt (line 3))
Using cached plotly-5.24.1-py3-none-any.whl.metadata (7.3 kB)
Collecting joblib==1.4.2 (from -r requirements.txt (line 4))
Using cached joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting scikit-learn==1.5.2 (from -r requirements.txt (line 5))
Downloading scikit-learn-1.5.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Collecting flask==3.0.3 (from -r requirements.txt (line 6))
Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Flask-Caching==2.3.0 (from -r requirements.txt (line 7))
Downloading Flask-Caching-2.3.0-py3-none-any.whl.metadata (2.2 kB)
Collecting Werkzeug==3.1 (from dash==2.18.2->-r requirements.txt (line 1))
Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Collecting dash-html-components==2.0.0 (from dash==2.18.2->-r requirements.txt (line 1))
Downloading dash-html-components-2.0.0-py3-none-any.whl.metadata (3.8 kB)
Collecting dash-core-components==2.0.0 (from dash==2.18.2->-r requirements.txt (line 1))
Downloading dash-core-components-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Collecting dash-table==5.0.0 (from dash==2.18.2->-r requirements.txt (line 1))
Downloading dash-table-5.0.0-py3-none-any.whl.metadata (2.4 kB)
Collecting importlib-metadata (from dash==2.18.2->-r requirements.txt (line 1))
Downloading importlib-metadata-8.5.0-py3-none-any.whl.metadata (4.8 kB)
Collecting typing-extensions==4.1.1 (from dash==2.18.2->-r requirements.txt (line 1))
Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting requests (from dash==2.18.2->-r requirements.txt (line 1))
Downloading requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting retrying (from dash==2.18.2->-r requirements.txt (line 1))
Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Collecting nest-asyncio (from dash==2.18.2->-r requirements.txt (line 1))
Downloading nest_asyncio-1.6.0-py3-none-any.whl.metadata (2.8 kB)
Collecting setuptools (from dash==2.18.2->-r requirements.txt (line 1))
Using cached setuptools-75.6.0-py3-none-any.whl.metadata (6.7 kB)
Collecting numpy==1.26.8 (from pandas==2.2.3->-r requirements.txt (line 2))
Downloading numpy-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
62.0/62.0 KB 4.3 MB/s eta 0:00:00
Collecting python-dateutil==2.0.2 (from pandas==2.2.3->-r requirements.txt (line 2))
Downloading python_dateutil-2.0.2-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting pytz==2020.1 (from pandas==2.2.3->-r requirements.txt (line 2))
Downloading pytz-2024.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata==2022.7 (from pandas==2.2.3->-r requirements.txt (line 2))
Downloading tzdata-2024.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting tenacity==6.2.0 (from plotly==5.24.1->-r requirements.txt (line 3))
Downloading tenacity-9.0.0-py3-none-any.whl.metadata (1.2 kB)
Collecting packaging (from plotly==5.24.1->-r requirements.txt (line 3))
Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting scipy==1.6.0 (from scikit-learn==1.5.2->-r requirements.txt (line 5))
Using cached scipy-1.14.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
Collecting threadpoolctl==3.1.0 (from scikit-learn==1.5.2->-r requirements.txt (line 5))
Downloading threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Collecting Jinja2==3.1.2 (from flask==3.0.3->-r requirements.txt (line 6))
Downloading Jinja2-3.1.4-py3-none-any.whl.metadata (2.6 kB)
```

Ilustración 16 Alistamiento de paquetes y bibliotecas.

Paso 4. Ejecutar el archivo principal de la aplicación

Ejecutar main.py dentro de un screen, una herramienta de Linux que mantiene procesos activos en segundo plano, incluso después de desconexiones.

Con las dependencias instaladas correctamente, el siguiente paso es ejecutar el archivo principal de la aplicación para iniciar el servidor del dashboard. Asegúrese de estar en el directorio donde se encuentra el archivo main.py y ejecute el siguiente comando:

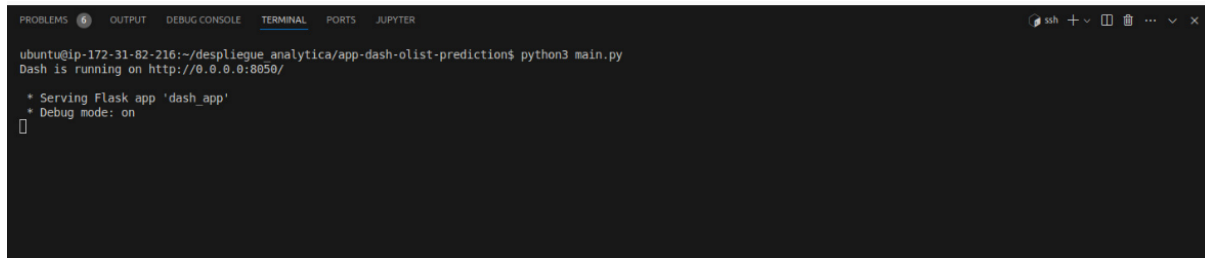
python3 main.py

Al ejecutar este comando, el servidor Flask/Dash se inicializa y comienza a escuchar en la dirección IP y puerto configurados. En este caso, la salida en la terminal confirmará que el dashboard está en ejecución:

Dash is running on http://0.0.0.0:8050/

* Serving Flask app 'dash_app'

* Debug mode: on



```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
ubuntu@ip-172-31-82-216:~/despliegue_analytica/app-dash-olist-prediction$ python3 main.py
Dash is running on http://0.0.0.0:8050/
* Serving Flask app 'dash_app'
* Debug mode: on
```

Ilustración 17 Ejecucion de main.py

Acceso al Dashboard:

Para visualizar el dashboard en un navegador, utilice la dirección pública de la instancia EC2 junto con el puerto especificado. Para este ejemplo:

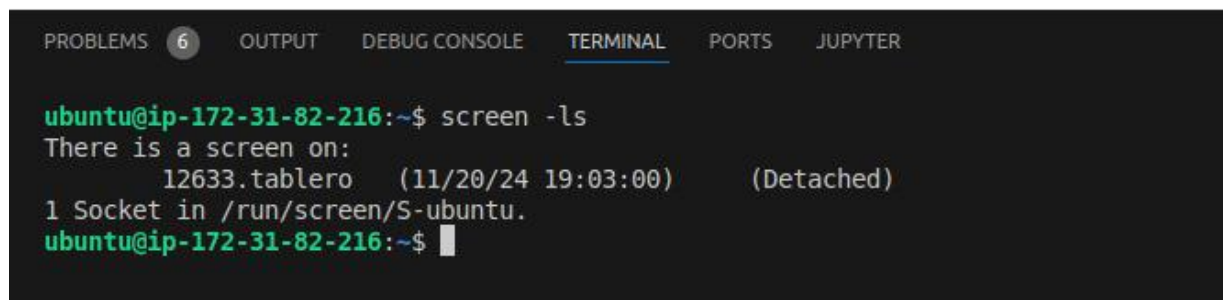
<http://44.203.201.203:8050/>

Paso 5. comprobar una sesión de screen activa

En este caso identificada como 12633.tablero, actualmente en estado Detached (desanexada y ejecutándose en segundo plano)

Para comprobar las sesiones activas en **screen**, se ejecuta:

screen -ls



```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
ubuntu@ip-172-31-82-216:~$ screen -ls
There is a screen on:
      12633.tablero  (11/20/24 19:03:00)      (Detached)
1 Socket in /run/screen/S-ubuntu.
ubuntu@ip-172-31-82-216:~$
```

Ilustración 18 Proceso activo

Esta salida debe mostrar que existe una sesión llamada tablero, en estado **Detached**, lo que indica que el proceso sigue activo.