



Contenido

Resumen del problema	3
Desarrollo de Modelos.....	4
Observaciones y conclusiones.	6
Tablero Desarrollado	12
Repositorios y fuentes de los modelos y tablero.....	16
Reporte de trabajo en equipo.....	21
Anexos.....	22

Resumen del problema

1. Contexto del Problema

Colombia enfrenta importantes retos de desigualdad en el acceso y calidad de la educación básica y media, lo cual incide directamente en el desempeño de los estudiantes en pruebas nacionales como Saber 11. Administrada por el Instituto Colombiano para el Fomento de la Educación Superior (ICFES), esta prueba evalúa competencias en áreas como matemáticas, inglés y ciencias naturales, y se acompaña de un formulario sociodemográfico. Este registro permite analizar patrones de éxito y desigualdad en los resultados, y es una fuente clave para el Ministerio de Educación para la formulación de políticas que promuevan una educación más equitativa y de calidad.

2. Pregunta de Negocio y Alcance del Proyecto

La pregunta de negocio relevante sería, ¿Cómo podemos mejorar el desempeño de los estudiantes en las pruebas Saber 11 conociendo sus factores demográficos? De aquí se desprende el problema de analytics que sería ¿De qué manera podemos utilizar los datos sociodemográficos de los estudiantes para identificar patrones de éxito y desigualdad? Con este enfoque, el proyecto busca optimizar la predicción del puntaje mediante la selección y entrenamiento de varios modelos de Machine Learning, como Random Forest, Regresión Lineal y Gradient Boosting, con el objetivo de identificar el modelo que minimice el error en las predicciones. La métrica de rendimiento principal es el Error Cuadrático Medio (MSE), que permitirá evaluar la precisión de cada modelo y su adecuación para este contexto educativo. Así, los objetivos específicos son:

- Identificar y analizar variables sociodemográficas y académicas que afectan el puntaje global.
- Entrenar y evaluar múltiples modelos de Machine Learning, comparándolos según el MSE para seleccionar el más efectivo.
- Ofrecer al Ministerio de Educación insights prácticos sobre factores críticos en el desempeño estudiantil, que puedan informar políticas educativas orientadas a reducir desigualdades.

3. Conjunto de Datos

El análisis se basa en los datos de Saber 11 del segundo semestre de 2022, que contienen 53,279 registros y 81 variables, de las cuales se seleccionaron 24 de mayor relevancia (por ejemplo, nivel educativo de los padres, acceso a internet, y puntajes por área). Se ha realizado una depuración de variables redundantes para centrarse en aquellas que aportan valor predictivo y simplificar el análisis, optimizando así el rendimiento de los modelos.

Principales Cambios Respecto a la Primera Entrega:

1. **Refinamiento del enfoque en el modelo predictivo:** En esta versión, el objetivo incluye la optimización de la precisión del modelo predictivo, especificando la métrica de MSE y los modelos comparados.
2. **Selección y justificación de variables:** La selección de variables ahora destaca aquellas con mayor valor predictivo, descartando redundancias como el código de municipio, para fortalecer el análisis.
3. **Énfasis en el impacto práctico:** Se subraya cómo el proyecto busca no solo predecir resultados con precisión, sino también informar políticas para reducir desigualdades, proporcionando un beneficio social tangible.

Desarrollo de Modelos

Para el desarrollo de los modelos, se creó un nuevo archivo.py, el cual se le asignó el nombre de “modeloMLFlow.py”. Dentro de este archivo, se creó la lógica para entrar diferentes modelos de Machine Learning y registrar sus resultados en MLFlow. El archivo parte de la importación de las librerías necesarias para el entrenamiento de los modelos entre las cuales se tienen sklearn para el importe de los modelos y MLFlow para llevar el registro de los experimentos.

```
import mlflow
import mlflow.sklearn
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import mean_squared_error
import pandas as pd
```

Posteriormente, se importa el dataset a usar y se escogen las variables que se usarán en el entrenamiento de los modelos, tal como se muestra en la siguiente imagen:

```
# Cargar los datos
df_definitivo = pd.read_csv("df_definitivo.csv")
variables_finales = ["FAMI_ESTRATOVIVIENDA", "FAMI_PERSONASHOGAR", "FAMI_TIENEINTERNET",
                    "ESTU_HORASSEMANATRABAJA", "FAMI_COMECARNEPESCADOHUEVO", "COLE_NATURALEZA", "ESTU_DEPTO_RESIDE", "COLE_JORNADA", "COLE_GENERO"]
```

Posterior a ello, se crearon las funciones para dividir los datos (Crear el dataframe con la variable objetivo “PUNT_GLOBAL” y el dataframe con las variables independientes) y la función para realizar la partición de datos en Train y Test

```
# Función para dividir los datos en X (características) y y (variable objetivo)
def dividir_datos(dataframe):
    X = dataframe[variables_finales]
    y = dataframe["PUNT_GLOBAL"]
    return X, y

datos = dividir_datos(df_definitivo)

# Función para crear las particiones de train y test, y codificar las variables categóricas
def crear_train_test(tupla):
    X, y = tupla
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
    oe = OrdinalEncoder()
    oe.fit(X_train)
    X_train_enc = oe.transform(X_train)
    X_test_enc = oe.transform(X_test)
    return X_train_enc, X_test_enc, y_train, y_test

datos2 = crear_train_test(datos)
```

El siguiente paso fue crear las funciones para el entrenamiento de 3 modelos que elegimos, los cuales fueron:

- Random Forest Regressor
- LinearRegression
- Gradient Boosting

```
# Función para entrenar el modelo de RandomForestRegressor
def crear_modelo_random_forest(tupla, num_trees=100, max_depth=None, max_features='auto'):
    X_train, X_test, y_train, y_test = tupla
    rf_regressor = RandomForestRegressor(n_estimators=num_trees, max_depth=max_depth, max_features=max_features, random_state=0)
    modelo = rf_regressor.fit(X_train, y_train)
    return modelo

# Función para entrenar el modelo de LinearRegression
def crear_modelo_lineal(tupla):
    X_train, X_test, y_train, y_test = tupla
    linear_regressor = LinearRegression()
    modelo = linear_regressor.fit(X_train, y_train)
    return modelo

# Función para entrenar el modelo de GradientBoostingRegressor
def crear_modelo_gradient_boosting(tupla, n_estimators=100, learning_rate=0.1, max_depth=3):
    X_train, X_test, y_train, y_test = tupla
    gbr_regressor = GradientBoostingRegressor(n_estimators=n_estimators, learning_rate=learning_rate, max_depth=max_depth, random_state=0)
    modelo = gbr_regressor.fit(X_train, y_train)
    return modelo
```

El siguiente paso, fue crear la función para generar las predicciones y calcular la métrica de evaluación que para este caso es el mean_square_error (mse).

```
# Función para hacer predicciones
def predecir(modelo, tupla):
    X_train, X_test, y_train, y_test = tupla
    y_pred = modelo.predict(X_test)
    return y_pred

# Función para calcular el MSE (Error Cuadrático Medio)
def calcular_mse(modelo, X_test, y_test):
    y_pred = modelo.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    return np.float64(mse)

# Preparar los datos de prueba
datos_internos = crear_train_test(datos)
X_test = datos_internos[1]
y_test = datos_internos[3]
```

Observaciones y conclusiones.

Ya con esto, procedimos a crear el experimento en MLFlow al cual llamamos "ICFES_EXPERIMENTS"

```
# Establecer el nombre del experimento en MLflow
mlflow.set_experiment("ICFES_Experiment") # Nombre del experimento
```

Para este caso, se inició un experimento por cada modelo a entrenar. La estructura general del experimento parte por determinar los parámetros del modelo, entrenar el modelo con la función correspondiente creada anteriormente, calcular el "mse" a partir de las predicciones obtenidas, y posteriormente crear el log para cada uno de los parámetros del modelo y de la métrica de evaluación que para todos fue el mse. Además, se crea un Log para registrar el nombre del modelo que se ejecutó.

En la siguiente imagen, se muestra la estructura del experimento creado para el Random Forest Regressor.

```
# Iniciar un experimento en MLflow
with mlflow.start_run():
    # Parámetros del modelo RandomForestRegressor
    num_trees = 200 # Número de árboles en el Random Forest
    max_depth = 4 # Profundidad máxima de cada árbol
    max_features = 5 # Características a considerar para cada árbol

    # Entrenar el modelo de RandomForestRegressor
    modelo_rf = crear_modelo_random_forest(datos2, num_trees=num_trees, max_depth=max_depth, max_features=max_features)

    # Calcular el MSE para RandomForest
    mse_rf = calcular_mse(modelo_rf, x_test, y_test)

    # Log de los parámetros y métricas del modelo RandomForest
    mlflow.log_param("model_type", "RandomForestRegressor")
    mlflow.log_param("n_estimators", num_trees)
    mlflow.log_param("max_depth", max_depth)
    mlflow.log_param("max_features", max_features)
    mlflow.log_metric("mse", mse_rf)

    # Log del modelo RandomForest
    mlflow.sklearn.log_model(modelo_rf, "model_random_forest")

    # Mostrar los resultados del modelo RandomForest
    print("Random Forest - MSE es de: " + str(mse_rf))
    print("Parámetros del modelo RandomForest:")
    print("Número de Árboles: " + str(num_trees))
    print("Máxima Profundidad: " + str(max_depth))
    print("Máximas Características por Árbol: " + str(max_features))
```

Se realizó una primera ejecución en los que se entrenó cada modelo con los parámetros que se muestran en la siguiente imagen:

Parameters

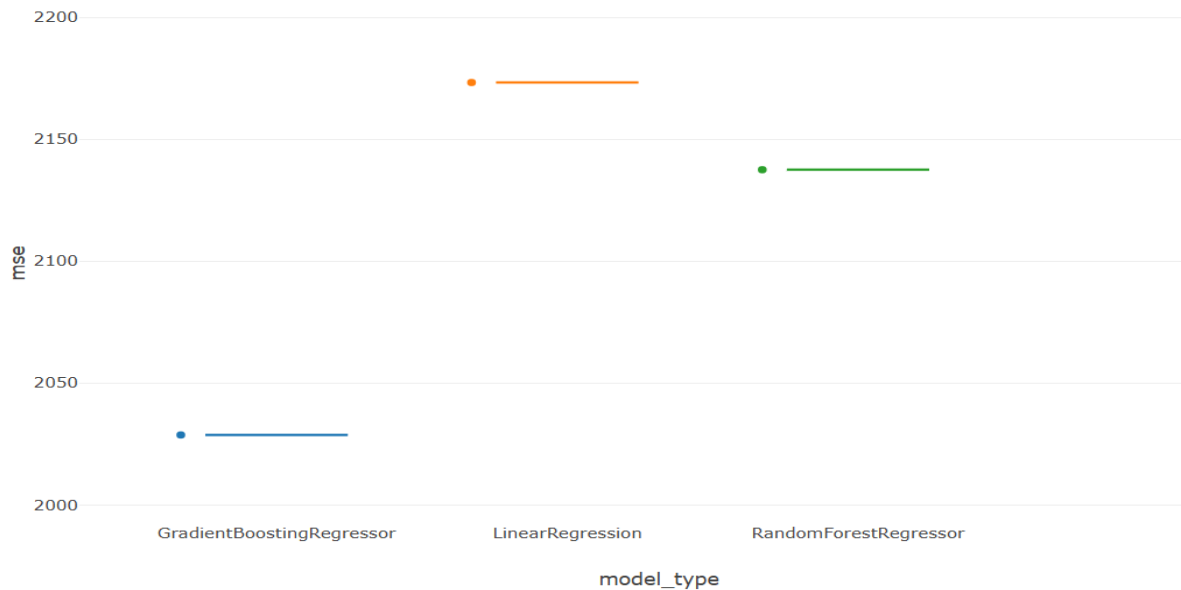
Show diff only

learning_rate	0.1		
max_depth	3		4
max_features			5
model_type	GradientBoostingRegressor	LinearRegression	RandomForestRegressor
n_estimators	100		200

Metrics

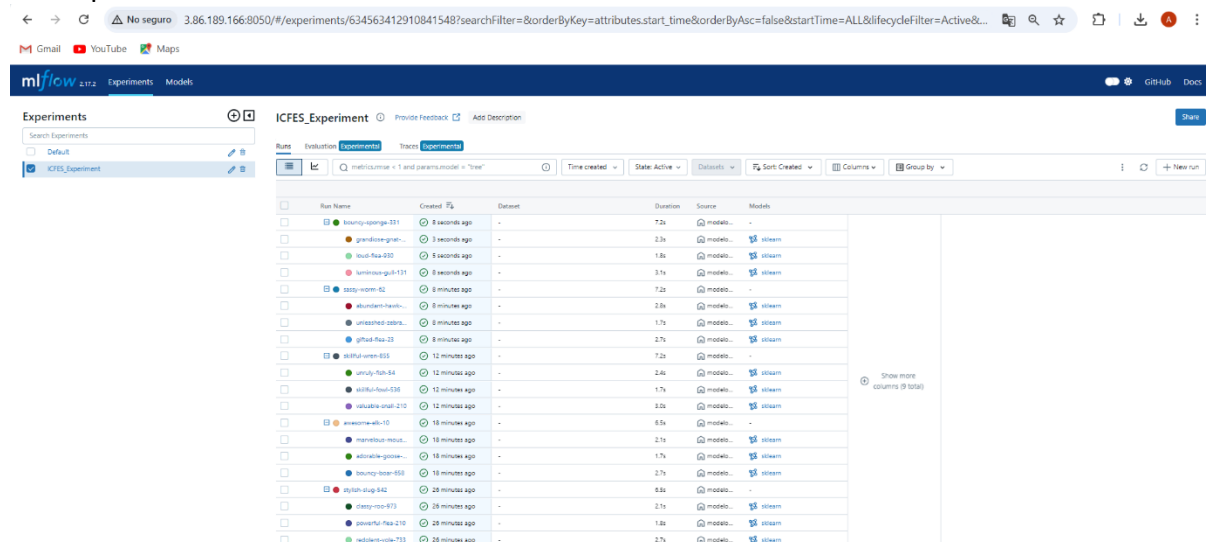
Show diff only

mse	2028.7	2173.3	2137.5
-----	--------	--------	--------



Lo que se puede concluir inicialmente es que el modelo que obtuvo un menor mse fue el GradientBoostingRegressor con un valor de 2028.7, seguido del RandomForestRegressor con 2137.5, y por último el modelo de regresión Lineal que obtuvo el resultado más alto. Con esta primera ejecución, nos centraremos en evaluar únicamente los modelos de GradientBoostingRegressor y el de RandomForestRegressor, modificando sus parámetros y comparando los resultados para saber que experimento obtuvo el menor mse, el cual sería finalmente el modelo que elegiremos.

Sin embargo, se realizaron más experimentos con el fin de buscar la versión del modelo con sus respectivos parámetros que permitiera disminuir el “mse”. En la siguiente imagen, se muestran los experimentos realizados y su respectivo registro en la interfaz de MLFlow desde la máquina virtual de AWS lanzada.



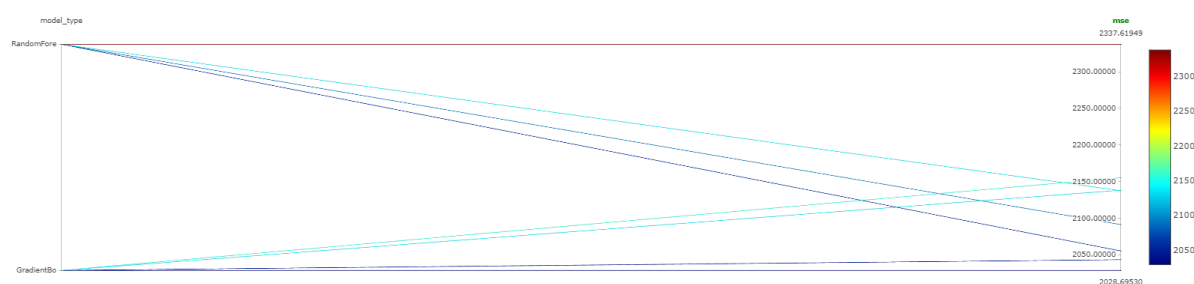
Posteriormente, empezamos a analizar cuál de todos los modelos de los experimentos obtuvieron el mejor resultado teniendo como preferencia los algoritmos de RandomForestRegressor y el de GradientBoostingRegressor ya que son los modelos con más parámetros ajustables y que mostraron el mejor performance en el primer experimento.

La siguiente imagen, muestra los valores del “mse” para cada uno de los modelos ejecutados al variar sus parámetros:

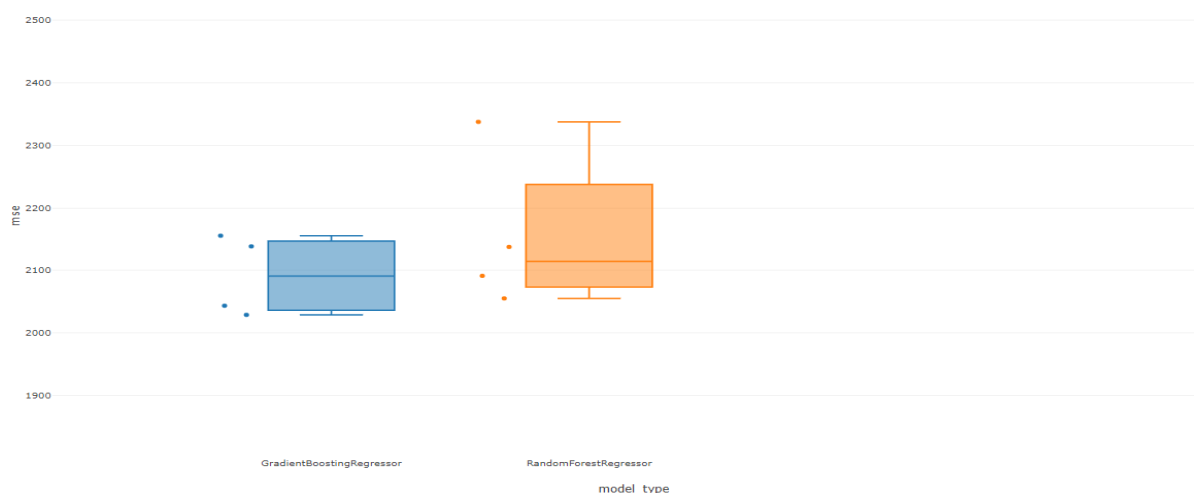
learning_rate	0.01		0.05		0.05		0.1	
max_depth	6	None	7	5	5	6	3	4
max_features		6		5		7		5
model_type	GradientBoostingRegressor	RandomForestRegressor	GradientBoostingRegressor	RandomForestRegressor	GradientBoostingRegressor	RandomForestRegressor	GradientBoostingRegressor	RandomForestRegressor
n_estimators	100	100	200	150	150	250	100	200
Metrics								
Show diff only								
mse	2138.3	2337.6	2155.4	2091.2	2043.3	2055.1	2028.7	2137.5

Nuevamente observamos como el modelo que obtiene los menores valores del MSE es el modelo de GradientBoostingRegressor, obteniendo un MSE de 2028.3. Para verificar la información anterior realizamos dos gráficos distintos, uno de pararell cordinates plot y otro de boxplot.

A continuación, se muestra el grafico de coordenadas paralelas para comparar el tipo del modelo vs el mse obtenido.



De anterior gráfico, podemos ver como para dos de los experimentos de Gradient Boosting se obtuvieron los menores valores del MSE, seguidos por los experimentos de RandomForestRegressor. Para evidenciar esa misma tendencia, graficamos un boxplot para ver el comportamiento del MSE por cada tipo de modelo:



En este gráfico, se puede observar el comportamiento del MSE para todos los experimentos desarrollados de cada modelo. Como se puede observar para el modelo de GradientBoosting el MSE tiende a tener una menor varianza en los resultados de sus experimentos y su rango Inter cuartil es más pequeño comparado con el modelo de RandomForestRegressor. Como conclusión, nos quedaremos inicialmente con el modelo de GradientBoostingRegressor con los siguientes parámetros:

- Learning_rate = 0.1
- Max_depth = 3
- Num_estimator = 100

Como paso a seguir probaremos nuevos experimentos, para ver si logramos encontrar una configuración de un modelo que permita reducir aún más el MSE.

Empaquetamiento del Modelo y Despliegue de API

Luego de que definimos el modelo que obtuvo el menor valor para el MSE, el cual corresponde al modelo de GradientBoostingRegressor cuyos parámetros especificamos en la sección anterior, se procedió a empaquetar dicho modelo, para que después pueda ser usado en una API que vamos a desplegar. Para empaquetar ese modelo, utilizamos una librería de python llamada “joblib” la cual permite guardar el modelo entrenado, y poder cargarlo cuando sea requerido para hacer predicciones.

Para ello, se creó un script de python al cual llamamos “train_model.py” que contiene el código que permite tanto entrenar el modelo de GradientBoosting seleccionado y empaquetarlo utilizando la librería de “Joblib”. La siguiente imagen muestra dicho proceso:

```
# Cargar el dataset
df = pd.read_csv(DATA_PATH)

# Dividir los datos
X = df[VARIABLES_FINALES]
y = df["PUNT_GLOBAL"]

# Codificar variables categóricas
from sklearn.preprocessing import OrdinalEncoder

# Configurar el encoder
encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
X_encoded = encoder.fit_transform(X)

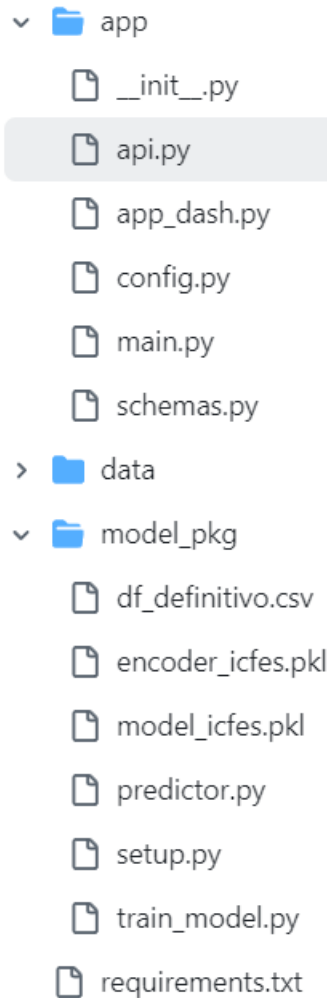
# Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Entrenar el modelo
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

# Guardar el modelo y el encoder
joblib.dump(model, "model_pkg/model_icfes.pkl")
joblib.dump(encoder, "model_pkg/encoder_icfes.pkl")
```

Al ejecutar este script se generan dos archivos de extensión .pkl, el primero llamado “model_icfes.pkl” encargado de almacenar el modelo de GradientBoostingRegressor con los parámetros definidos y el otro llamado “encoder_icfes.pkl” el cual almacena el objeto Ordinal Encoder que se utilizó para transformar las variables categóricas en valores numéricos. Ya con esto teníamos el modelo empaquetado, listo para poder usarlo en la API que creamos.

La siguiente imagen, muestra la organización de las carpetas del proyecto para desplegar tanto la API como el Tablero desarrollado.



```

  app
  ├── __init__.py
  ├── api.py
  ├── app_dash.py
  ├── config.py
  ├── main.py
  ├── schemas.py
  ├── data
  └── model_pkg
      ├── df_definitivo.csv
      ├── encoder_icfes.pkl
      ├── model_icfes.pkl
      ├── predictor.py
      ├── setup.py
      ├── train_model.py
      └── requirements.txt

```

En la carpeta `model_pkg`, se encuentra el script que mencionamos anteriormente para entrenar el modelo y empaquetarlo. Aparte de eso se tienen otros scripts, destacando el script llamado “`predictor.py`” el cual básicamente tiene la lógica para realizar las predicciones usando el modelo empaquetado. Dicha lógica está dentro de una función llamada “`make_prediction`” que recibe como parámetro los valores de las variables de entrada y retorna las predicciones de dichos datos de entrada. Cabe destacar que en esta misma carpeta se encuentran los archivos `.pkl` generados en el script “`train_model.py`”.

Por otro lado, tenemos la carpeta `data`, y la carpeta `app`. La primera carpeta tiene como contenido el dataset usado para el entrenamiento del modelo, y la segunda carpeta tiene tanto los scripts encargados de desplegar la API como el tablero.

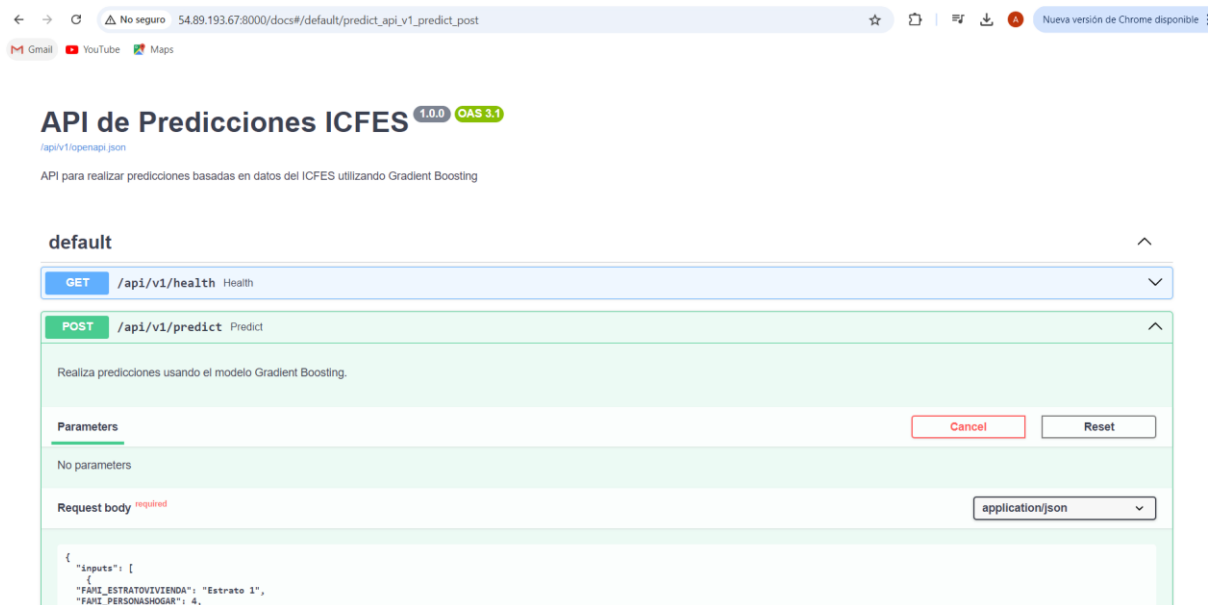
Los archivos relacionados con el despliegue de la API son:

- `Api.py`: Este script define una API utilizando Fast API para exponer el modelo de predicción basado en Gradient Boosting. Se destaca la ruta “`predict`”, la cual tiene la lógica para generar las predicciones según los datos de entrada usando la función `make_prediction` del script “`predictor.py`” de la carpeta `model_pkg`.

- Config.py: Configura las variables globales y la gestión de logs para la aplicación de FastAPI.
- Schemas.py: Este script permite definir modelos de datos para estructurarlos. Por ejemplo, define la estructura de las variables de entrada como lo son los tipos de datos de cada variable.
- Main.py: Este script configura la API para realizar las predicciones usando datos del ICFES.

Finalmente, el script “main.py” es el encargado de desplegar la API. Este despliega el servidor en el puerto 8000 tal como se muestra en el siguiente fragmento de código: “uvicorn.run(app, host=“0.0.0.0”, port=8000, log_level=“debug”)”

La siguiente imagen muestra la API desplegada:



Para mayores detalles, se debe revisar el manual de instalación del tablero en el cual se encuentra la explicación de como ejecutar el proyecto, para desplegar tanto la API, como el tablero.

Tablero Desarrollado

Se crearon dos versiones la primera con una estética mas cuidadosa y una segunda versión mas lite pero más funcional la cual es la que finalmente desplegamos. Con la primera versión teníamos lo siguiente:

Versión 1

Selección de Variables

Estrato: Estrato 1

Personas Hogar: 1 a 2

Tiene Internet:

No
Si

Horas Trabaja: 0

Come Carne:

1 o 2 veces por se...
3 a 5 veces por se...
Nunca o rara vez c...

Tiene Internet:

NO OFICIAL
OFICIAL

Depto Reside: BOGOTÁ

Predecir Puntaje

Resultados de Predicción

Random Forest
Regresión Lineal
Gradient Boosting
Estrato
Personas Hogar
Tiene Internet
Horas Trabaja
Come Carne
Naturaleza Cole
Depto Reside

Resultados de Predicción

	Random Forest	Regresión Lineal	Gradient Boosting	Estrato	Personas Hogar	Tiene Internet	Horas Trabaja	Come Carne	Naturaleza Cole	Depto Reside
0	245.715564	263.106372	215.364215	Estrato 1	1 a 2	No	0	1 o 2 veces por semana	NO OFICIAL	ANTIOQUIA
1	232.533134	261.225387	242.680879	Estrato 1	1 a 2	Si	0	1 o 2 veces por semana	OFICIAL	ANTIOQUIA
2	243.856668	261.363189	251.294682	Estrato 1	1 a 2	Si	0	1 o 2 veces por semana	OFICIAL	BOGOTÁ

Para el despliegue por problemas en el rendimiento se creó una versión más lite que es la que se ve a continuación:

Versión 2

Predicción del Puntaje Global - ICFES

Estrato de Vivienda: Personas en el Hogar: Tiene Internet?:

Horas Semanales que Trabaja: Frecuencia de Consumo de Carne, Pescado o Huevo: Naturaleza del Colegio: Departamento donde Reside:

Jornada del Colegio: Género del Colegio:

Predicción del Puntaje Global: 214.69

Descripción del tablero desarrollado y funcionalidad

Este dashboard interactivo realiza predicciones sobre un puntaje global un modelo de Gradient Boosting. La interfaz de usuario permite seleccionar varias variables y, tras hacer una predicción, este tablero se conecta a la api del modelo y le pasa los input registrados por el usuario y devuelve la predicción del puntaje.

Descripción del dashboard:

- Selección de Variables:
 - Estrato: Campo abierto para seleccionar el estrato socioeconómico (de 1 a 6).
 - Personas en el hogar: Un campo para recibir datos numéricos para seleccionar el número de personas en el hogar (1-2, 3-4, etc.).
 - Tiene Internet: Un menu para indicar si la persona tiene internet (Sí o No).
 - Frecuencia de consumo de carne: Un campo abierto para indicar la frecuencia con la que se consume carne (de 1-2 veces por semana a todos los días).
 - Naturaleza del colegio: Un campo abierto para determinar si el colegio es oficial o no oficial.
 - Departamento de residencia: Un campo abierto para seleccionar el departamento de residencia de la persona (por ejemplo, Bogotá, Antioquia, etc.).
 - Horas de trabajo semanales: Un campo para registrar el número de horas trabajadas por semana (menos de 10 horas a más de 30 horas).
- Botón de predicción:

Un botón titulado "Predecir" que, cuando se hace clic, realiza la predicción del puntaje utilizando los modelos entrenados y muestra el resultado.

- Resultados de predicción:

Una vez que se realiza la predicción, el dashboard muestra los puntajes predichos por el modelo (Gradient Boosting).

- Flujo del dashboard:

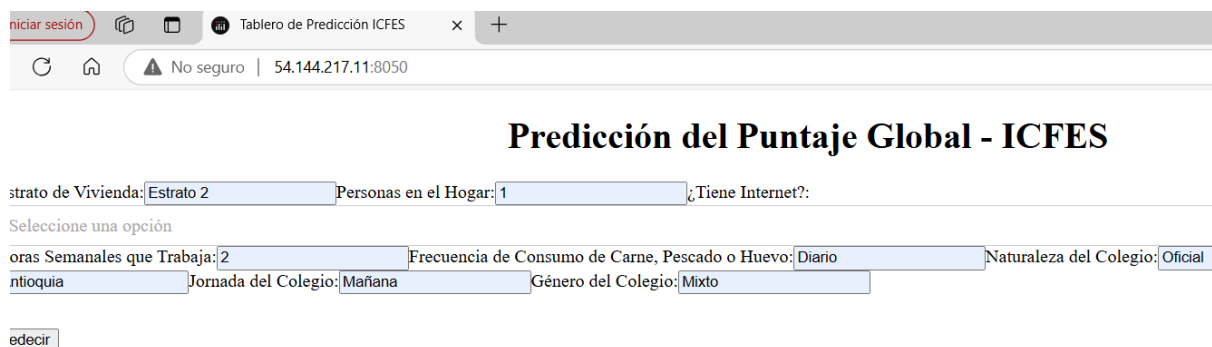
El usuario selecciona las opciones de las variables a través de los widgets (menús desplegables y botones).

Al hacer clic en el botón "Predecir Puntaje", se realiza la predicción usando los modelos

Para la versión final del despliegue solo se usa el modelo de gradient boosting ya que fue el que mejor desempeño tuvo en los experimentos de mlflow

Interacción con la API:

El tablero interactúa con una API externa para obtener la predicción del puntaje. A través de una solicitud POST, el tablero envía los datos ingresados por el usuario en un formato semi estructurado tipo json con las variables predictoras que son transformadas por el modelo encoder empaquetado en la api que permite la predicción del modelo y retorna la predicción en pantalla como se ve a continuación:



Inicio sesión

Tablero de Predicción ICFES

No seguro | 54.144.217.11:8050

Predicción del Puntaje Global - ICFES

strato de Vivienda: Estrato 2 Personas en el Hogar: 1 ¿Tiene Internet?:

Seleccione una opción

oras Semanales que Trabaja: 2 Frecuencia de Consumo de Carne, Pescado o Huevo: Diario Naturaleza del Colegio: Oficial

ntioquia Jornada del Colegio: Mañana Género del Colegio: Mixto

edecir

Predicción del Puntaje Global: 214.69

- Predicción del Puntaje Global: Se muestra el puntaje estimado con dos decimales y en color verde si la predicción es exitosa.
- Manejo de Errores: En caso de que la API no esté disponible o si ocurre un error en el proceso, el tablero muestra un mensaje de error en color rojo.

Este dashboard es útil para visualizar cómo el modelo predice el puntaje global en función de las variables socioeconómicas y de comportamiento seleccionadas por el usuario. Además, su integración con una API hace que el proceso de predicción sea eficiente, mientras que su diseño en Dash proporciona una experiencia de usuario clara y amigable.

Esta son las salidas de consola:

- Api modelo

```
2024-11-24 22:19:32.380 | INFO | app.api.predict:32 - Recibiendo datos de entrada para predicción.
2024-11-24 22:19:32.384 | INFO | app.api.predict:36 - Datos de entrada: [{'FAMI_ESTRATOVIVIENDA': 'Estrato 2', 'FAMI_PERSONASHOGAR': 1, 'FAMI_TIENEINTERNET': None, 'ESTU_HORASSEMANATRABAJA': 2, 'FAMI_COMECARNEPESCADOHUEVO': 'Diario', 'COLE_NATURALEZA': 'Oficial', 'ESTU_DEPTO_RESIDE': 'Antioquia', 'COLE_JORNADA': 'Mañana', 'COLE_GENERO': 'Mixto'}]
2024-11-24 22:19:32.384 | INFO | model_pkg.predictor:make_prediction:42 - Validando las columnas en los datos de entrada...
2024-11-24 22:19:32.384 | INFO | model_pkg.predictor:make_prediction:50 - Datos originales:
0 FAMI_ESTRATOVIVIENDA FAMI_PERSONASHOGAR FAMI_TIENEINTERNET ESTU_HORASSEMANATRABAJA ... COLE_NATURALEZA ESTU_DEPTO_RESIDE COLE_JORNADA COLE_GENERO
Estrato 2 1 None 2 ... Oficial Antioquia Mañana Mixto

[1 rows x 9 columns]
2024-11-24 22:19:32.314 | INFO | model_pkg.predictor:make_prediction:51 - Tipos de datos originales:
FAMI_ESTRATOVIVIENDA object
FAMI_PERSONASHOGAR int64
FAMI_TIENEINTERNET object
ESTU_HORASSEMANATRABAJA int64
FAMI_COMECARNEPESCADOHUEVO object
COLE_NATURALEZA object
ESTU_DEPTO_RESIDE object
COLE_JORNADA object
COLE_GENERO object
dtype: object
2024-11-24 22:19:32.322 | INFO | model_pkg.predictor:make_prediction:57 - Datos después de transformar a tipo 'str':
0 FAMI_ESTRATOVIVIENDA FAMI_PERSONASHOGAR FAMI_TIENEINTERNET ESTU_HORASSEMANATRABAJA ... COLE_NATURALEZA ESTU_DEPTO_RESIDE COLE_JORNADA COLE_GENERO
Estrato 2 1 None 2 ... Oficial Antioquia Mañana Mixto

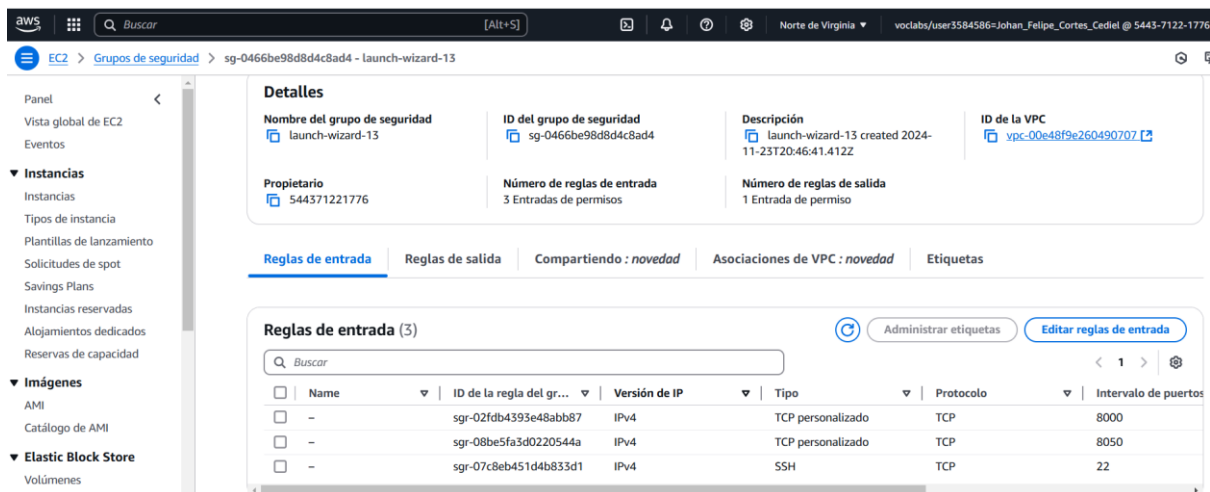
[1 rows x 9 columns]
2024-11-24 22:19:32.323 | INFO | model_pkg.predictor:make_prediction:58 - Tipos de datos después de transformar:
FAMI_ESTRATOVIVIENDA object
FAMI_PERSONASHOGAR object
FAMI_TIENEINTERNET object
ESTU_HORASSEMANATRABAJA object
FAMI_COMECARNEPESCADOHUEVO object
COLE_NATURALEZA object
ESTU_DEPTO_RESIDE object
COLE_JORNADA object
COLE_GENERO object
dtype: object
2024-11-24 22:19:32.327 | INFO | model_pkg.predictor:make_prediction:62 - Datos codificados:
[[ 1. -1. -1. -1. -1. -1. -1. -1. -1.]]
2024-11-24 22:19:32.327 | INFO | model_pkg.predictor:make_prediction:65 - Realizando predicción con el modelo...
2024-11-24 22:19:32.327 | INFO | app.api.predict:45 - Predicciones generadas: [214.689715813579]
```

- Tablero dash

```
Already up to date.
(venv) ubuntu@ip-172-31-18-182:~/Prueba2/app$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 2), reused 4 (delta 2), pack-reused 0 (from 0)
Unpacking objects: 100% (4/4), 440 bytes | 48.00 KiB/s, done.
From https://github.com/felipemiad/Prueba2
 8d41b2a..2cb64f2  main      -> origin/main
Updating 8d41b2a..2cb64f2
Fast-forward
 app/app_dash.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
(venv) ubuntu@ip-172-31-18-182:~/Prueba2/app$ python app_dash.py
Dash is running on http://0.0.0.0:8050/

* Serving Flask app 'app_dash'
* Debug mode: on
client_loop: send disconnect: Connection reset
```

- Grupos de seguridad



The screenshot shows the AWS Management Console interface. On the left is a navigation menu with options like 'Panel', 'Vista global de EC2', 'Eventos', 'Instancias', 'Tipos de instancia', 'Plantillas de lanzamiento', 'Solicitudes de spot', 'Savings Plans', 'Instancias reservadas', 'Alojamientos dedicados', 'Reservas de capacidad', 'Imágenes', 'AMI', 'Catálogo de AMI', and 'Elastic Block Store'. The main content area displays the details for a security group named 'launch-wizard-13'. It includes fields for 'Nombre del grupo de seguridad', 'ID del grupo de seguridad', 'Descripción', 'ID de la VPC', 'Propietario', 'Número de reglas de entrada', and 'Número de reglas de salida'. Below this, there are tabs for 'Reglas de entrada', 'Reglas de salida', 'Compartiendo : novedad', 'Asociaciones de VPC : novedad', and 'Etiquetas'. The 'Reglas de entrada' tab is active, showing a table with 3 rules.

	Name	ID de la regla del gr...	Versión de IP	Tipo	Protocolo	Intervalo de puertos
<input type="checkbox"/>	-	sgr-02fdb4393e48abb87	IPv4	TCP personalizado	TCP	8000
<input type="checkbox"/>	-	sgr-08be5fa3d0220544a	IPv4	TCP personalizado	TCP	8050
<input type="checkbox"/>	-	sgr-07c8eb451d4b833d1	IPv4	SSH	TCP	22

Repositorios y fuentes de los modelos y tablero

Debido a la complejidad del proyecto, fue necesario crear dos repositorios para su correcta gestión, el primer repositorio, [Proyecto Analítica](https://github.com/felipemiad/Proyecto_Analitica) (https://github.com/felipemiad/Proyecto_Analitica) fue el inicial y en él se desarrolló la primera etapa del proyecto. Este repositorio contiene el versionamiento de datos, el proyecto completo, así como los distintos commits y experimentos realizados con los modelos. Dentro de este repositorio, en la carpeta notebooks, se encuentran los modelos y el código de preprocesamiento utilizados.

Sin embargo, para el despliegue, surgieron complicaciones debido a los controles establecidos en el primer repositorio, los cuales requerían la aprobación de una persona para realizar *push*. Esto generó conflictos durante el proceso de despliegue. Como solución, se creó un segundo repositorio dedicado exclusivamente al despliegue: [Prueba2](https://github.com/felipemiad/Prueba2.git) (<https://github.com/felipemiad/Prueba2.git>).

De esta manera, se logró separar las etapas de desarrollo y despliegue, permitiendo un flujo de trabajo más ágil y organizado.

Repositorio 1:

En la carpeta notebooks se encuentran los modelos y código de preprocesamiento.

Proyecto_Analitica / notebooks /

Name	Last commit message	Last commit date
..		
.ipynb_checkpoints	Imputación valores faltantes y actualización base	yesterday
data	Imputación valores faltantes y actualización base	yesterday
modeloMLFlow.py	Se ajusta el codigo con la ip de s3	yesterday
modelos_mlflow.ipynb	Añadir notebooks para análisis y MLflow, y archivo .py para Dash	yesterday
proyecto_analitica.ipynb	Imputación valores faltantes y actualización base	yesterday

En la carpeta dashboard se encuentra el código de análisis descriptivo y el del tablero, el cual puede correrse para visualizar el tablero tal como se mostró en los snapshots más arriba:

Proyecto_Analitica / dashboard /

Name	Last commit message	Last commit date
..		
Tablero.py	Añadir notebooks para análisis y MLflow, y archivo .py para Dash	yesterday

Además, se crearon distintas ramas por integrante:

felipemad / Proyecto_Analitica

5 Branches 0 Tags

Switch branches/tags

Find or create a branch...

Branches

- ✓ master (default)
- andres
- jorge
- jorge2
- laura

View all branches

README

Commit history:

- 465bfa4 · 38 minutes ago · 25 Commits
- Inicialización del proyecto limpio con configuración de DVC ... 8 hours ago
- Añadir notebooks para análisis y MLflow, y archivo .py para ... 7 hours ago
- Se ajusta el codigo con la ip de s3 38 minutes ago
- Se ajusta el codigo con la ip de s3 38 minutes ago
- Inicialización del proyecto limpio con configuración de DVC ... 8 hours ago
- Actualizar .gitignore y limpiar archivos untracked 6 hours ago
- Inicialización del proyecto limpio con configuración de DVC ... 8 hours ago

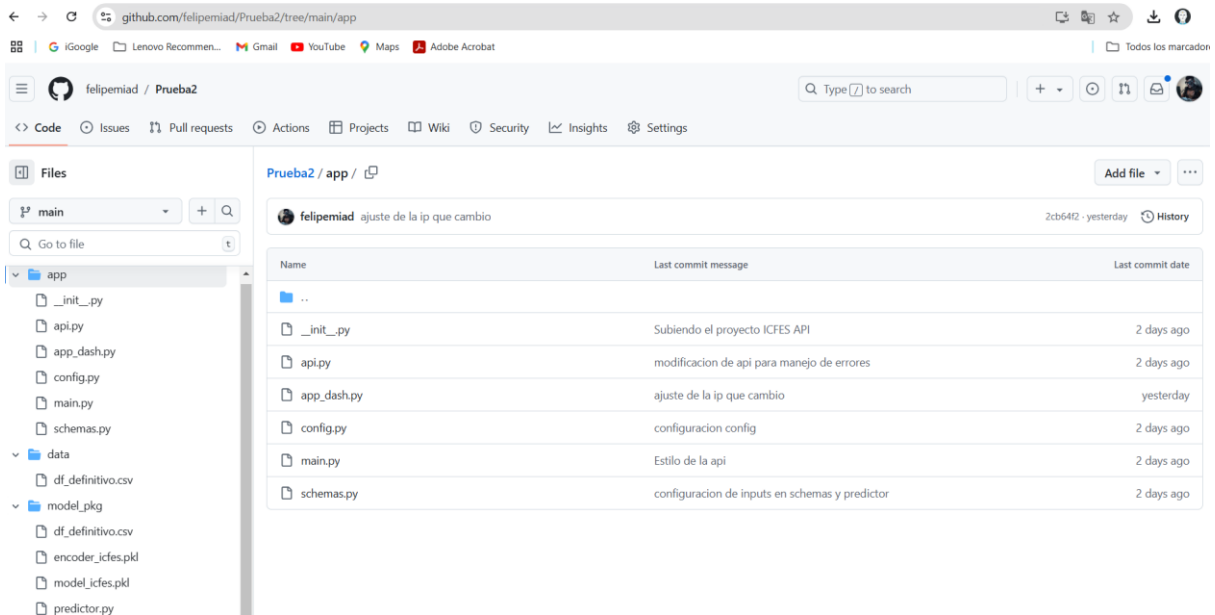
Cada uno hizo commit con las tareas que le correspondían como se ve a continuación:

Commits

master	All users	All time
Commits on Nov 10, 2024		
Se ajusta el código con la ip de s3 felipemiad committed 40 minutes ago	465bfa4	<>
Merge pull request #14 from felipemiad/jorge2 felipemiad authored 1 hour ago	Verified 893efcb	<>
Imputación valores faltantes y actualización base josorioh committed 1 hour ago	37bf441	<>
Merge pull request #13 from felipemiad/jorge2 felipemiad authored 1 hour ago	Verified 25f9532	<>
Merge pull request #12 from felipemiad/master felipemiad authored 1 hour ago	Verified 4279b50	<>
Merge pull request #11 from felipemiad/jorge felipemiad authored 1 hour ago	Verified 20994b3	<>
Merge pull request #10 from felipemiad/master felipemiad authored 1 hour ago	Verified 7450697	<>
Merge pull request #9 from felipemiad/andres felipemiad authored 2 hours ago	Verified f0249f0	<>
Incluir nueva versión de los datos Ruizfierro9 committed 3 hours ago	52c489b	<>
Añadir archivo Modelos con MLFlow Ruizfierro9 committed 3 hours ago	26a2e0e	<>
Merge pull request #8 from felipemiad/master felipemiad authored 3 hours ago	Verified ee71877	<>
Merge pull request #7 from felipemiad/laura felipemiad authored 3 hours ago	Verified 4c975cb	<>
Merge pull request #6 from felipemiad/master felipemiad authored 3 hours ago	Verified 01766e0	<>
Merge pull request #5 from felipemiad/andres felipemiad authored 3 hours ago	Verified e3ef0f9	<>
Merge pull request #4 from felipemiad/master felipemiad authored 3 hours ago	Verified 335ecf3	<>

Merge pull request #3 from felipemiad/jorge	Verified	469b65d	<>
felipemiad authored 3 hours ago			
Merge pull request #2 from felipemiad/master	Verified	beff3b9	<>
felipemiad authored 3 hours ago			
Merge pull request #1 from felipemiad/laura	Verified	47f7a45	<>
felipemiad authored 4 hours ago			
importar librerías y traer dataframe		0ff54db	<>
LauraCMartinezCruz committed 4 hours ago			
Agregar la carpeta notebooks con los notebooks del proyecto		9b5128f	<>
felipemiad committed 5 hours ago			
Actualizar .gitignore y limpiar archivos untracked		28c14fb	<>
felipemiad committed 6 hours ago			
Añadir base de datos depurada para modelos al control de versiones de DVC		ca90451	<>
felipemiad committed 7 hours ago			
Añadir notebooks para análisis y MLflow, y archivo .py para Dash		d383db8	<>
felipemiad committed 7 hours ago			
Añadir base de datos original al control de versiones de DVC		0d66c94	<>
felipemiad committed 7 hours ago			
Inicialización del proyecto limpio con configuración de DVC y estructura de carpetas		b5c8d92	<>
felipemiad committed 8 hours ago			

En el segundo repositorio ya es como tal el despliegue de la api para el consumo del servicio y el despliegue del tablero en dash. Como se ve a continuación



github.com/felipemiad/Prueba2/tree/main/app

felipemiad / Prueba2

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Files

main

Go to file

app

- _init_.py
- api.py
- app_dash.py
- config.py
- main.py
- schemas.py

data

- df_definitivo.csv

model_pkg

- df_definitivo.csv
- encoder_icfes.pkl
- model_icfes.pkl
- predictor.py

Prueba2 / app

felipemiad ajuste de la ip que cambio 2cb64f2 · yesterday History

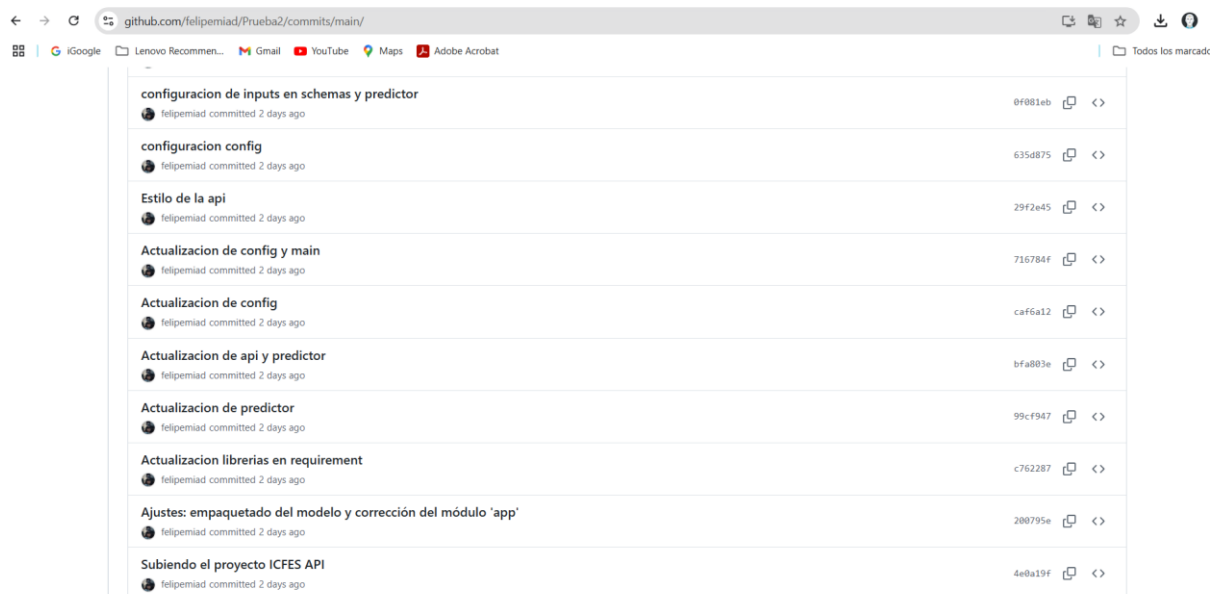
Name	Last commit message	Last commit date
..		
init.py	Subiendo el proyecto ICSES API	2 days ago
api.py	modificacion de api para manejo de errores	2 days ago
app_dash.py	ajuste de la ip que cambio	yesterday
config.py	configuracion config	2 days ago
main.py	Estilo de la api	2 days ago
schemas.py	configuracion de inputs en schemas y predictor	2 days ago

Commits:

Commits

main	All users	All time
Commits on Nov 24, 2024		
ajuste de la ip que cambio felipemiad committed yesterday	2cb64f2	📄 <>
ajuste al tablero felipemiad committed yesterday	8d41b2a	📄 <>
se elimina version especifica de pandas que genera problema felipemiad committed yesterday	ba1e7fc	📄 <>
se crea el tablero y se ajusta requirements felipemiad committed yesterday	6c6424f	📄 <>
Commits on Nov 23, 2024		
se ajusta encoder felipemiad committed 2 days ago	ff66376	📄 <>
agregando log para evidenciar errores felipemiad committed 2 days ago	78448ee	📄 <>
se pone ruta relativa y se incluye datos a otra carpeta por rutas felipemiad committed 2 days ago	7a96587	📄 <>

github.com/felipemiad/Prueba2/commits/main/ 🔍 📄 ☆ ⬇️ ⓘ		
🔍 iGoogle 📁 Lenovo Recommen... 📧 Gmail 📺 YouTube 📍 Maps 📄 Adobe Acrobat 📁 Todos los marcados		
se pone ruta relativa y se incluye datos a otra carpeta por rutas felipemiad committed 2 days ago	18bf2d8	📄 <>
se pone ruta relativa felipemiad committed 2 days ago	6045ae5	📄 <>
se ajusta ruta felipemiad committed 2 days ago	2d55eb5	📄 <>
se quita pandas felipemiad committed 2 days ago	b34f612	📄 <>
reentrenando el modelo felipemiad committed 2 days ago	4f08e7b	📄 <>
eliminado prueba felipemiad committed 2 days ago	baa4220	📄 <>
ajustando encoder felipemiad committed 2 days ago	ef91e74	📄 <>
Sobrescribir model_icfes.pkl con la versión actual Ubuntu committed 2 days ago	66eca48	📄 <>
se reentrena el modelo por errores y se actualiza felipemiad committed 2 days ago	ff7c923	📄 <>
modificacion de variables finales felipemiad committed 2 days ago	a2f1d8a	📄 <>
modificacion de api para manejo de errores	93ef19b	📄 <>



Reporte de trabajo en equipo

Johan Felipe Cortés Cediel	Laura Cristina Martínez Cruz	Jorge Andrés Osorio Henao	Andrés David Ruiz Fierro
Encargado de la migración del proyecto al nuevo repositorio en github manejando ramas para cada integrante y aprobación central en la rama master, junto con todo el despliegue técnico en AWS que incluyo el bucket, el servicio IAM y la configuración de todo el	Colaboración en el desarrollo del dashboard en Python, siguiendo en general el diseño conceptual previamente elaborado en la entrega anterior. Realicé una descripción detallada del dashboard, de su estructura, funcionalidades, y de la utilidad que cada sección. En mi commit realicé la importación de las librerías	Ajuste e implementación del código en Python necesario para la corrida de los modelos predictivos y para el dashboard de predicción. Commit en github actualizando la base de datos e imputando los valores faltantes en la base.	Creación del código para el uso de MLFlow y configuración de los experimentos de los modelos. Lanzamiento de instancia en AWS, ejecución del server de MLFlow y creación de experimentos de MLFlow. Creación del manual de instalación del tablero, y revisión de los procesos de despliegue tanto del tablero como de la API.

ambiente para los commits.	necesarias para visualización y manejo de datos y realicé la carga de la data asegurando que las fuentes estuvieran integradas adecuadamente al entorno de desarrollo.		
Creación del segundo repositorio y el despliegue de la api y el tablero que se comunica con la api del modelo			

Anexos

Características de la instancia de AWS

Clonación del repositorio remoto dentro de la instancia

```
ubuntu@ip-172-31-91-104:~$ git clone https://github.com/felipemiad/Proyecto_Analitica.git
Cloning into 'Proyecto_Analitica'...
remote: Enumerating objects: 72, done.
remote: Counting objects: 100% (72/72), done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 72 (delta 22), reused 53 (delta 17), pack-reused 0 (from 0)
Receiving objects: 100% (72/72), 8.94 MiB | 20.43 MiB/s, done.
Resolving deltas: 100% (22/22), done.
ubuntu@ip-172-31-91-104:~$ cd Proyecto_Analitica/
ubuntu@ip-172-31-91-104:~/Proyecto_Analitica$ cd notebooks/
ubuntu@ip-172-31-91-104:~/Proyecto_Analitica/notebooks$ cp modeloMLFlow.py /home/ubuntu/
ubuntu@ip-172-31-91-104:~/Proyecto_Analitica/notebooks$ cd ..
ubuntu@ip-172-31-91-104:~/Proyecto_Analitica$ cd ..
ubuntu@ip-172-31-91-104:~$ ls
Proyecto_Analitica  env-mlflow  modeloMLFlow.py
ubuntu@ip-172-31-91-104:~$
```

Activación del ambiente virtual, y ejecución del server

```
ubuntu@ip-172-31-91-104:~$ source env-mlflow/bin/activate
(env-mlflow) ubuntu@ip-172-31-91-104:~$ mlflow server -h 0.0.0.0 -p 8050
[2024-11-11 04:58:10 +0000] [3657] [INFO] Starting gunicorn 23.0.0
[2024-11-11 04:58:10 +0000] [3657] [INFO] Listening at: http://0.0.0.0:8050 (3657)
[2024-11-11 04:58:10 +0000] [3657] [INFO] Using worker: sync
[2024-11-11 04:58:10 +0000] [3658] [INFO] Booting worker with pid: 3658
[2024-11-11 04:58:10 +0000] [3659] [INFO] Booting worker with pid: 3659
[2024-11-11 04:58:10 +0000] [3660] [INFO] Booting worker with pid: 3660
[2024-11-11 04:58:10 +0000] [3661] [INFO] Booting worker with pid: 3661
[2024-11-11 04:58:20 +0000] [3657] [INFO] Handling signal: winch
[2024-11-11 04:58:24 +0000] [3657] [INFO] Handling signal: winch
```

Ejecución del primer experimento, al ejecutar el archivo modeloMLFlow.py dentro de la máquina virtual

```
ubuntu@ip-172-31-91-104:~$ source env-mlflow/bin/activate
(env-mlflow) ubuntu@ip-172-31-91-104:~$ python3 modeloMLFlow.py
2024/11/11 05:13:08 INFO mlflow.tracking.fluent: Experiment with name 'ICFES_Experiment' does not exist. Creating a new experiment.
2024/11/11 05:13:11 WARNING mlflow.models.model: Model logged without a signature and input example. Please set 'input_example' parameter when logging the model to auto infer the model signature.
Random Forest - MSE: 2137.4997643113675
2024/11/11 05:13:13 WARNING mlflow.models.model: Model logged without a signature and input example. Please set 'input_example' parameter when logging the model to auto infer the model signature.
Linear Regression - MSE: 2173.277183815201
2024/11/11 05:13:15 WARNING mlflow.models.model: Model logged without a signature and input example. Please set 'input_example' parameter when logging the model to auto infer the model signature.
Gradient Boosting - MSE: 2028.6952993901064
(env-mlflow) ubuntu@ip-172-31-91-104:~$
```

Interfaz de MLflow en la máquina virtual

The screenshot shows the MLflow web interface in a browser. The address bar indicates the URL is http://3.86.189.166:8050/experiments/634563412910841548?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&startTime=ALL&lifecycleFilter=Active&model.... The interface has a sidebar on the left with 'Experiments' and 'Models' tabs. The main content area shows the 'ICFES_Experiment' page. At the top, there are tabs for 'Runs', 'Evaluation', and 'Traces'. Below these, there is a search bar and a table of runs. The table has columns: Run Name, Created, Status, Dataset, Duration, Source, and Models. The runs are listed in descending order of creation time, with the most recent run at the top. The table shows 20 runs, each with a unique name, creation time, status, and source.

Run Name	Created	Status	Dataset	Duration	Source	Models
louny-sponge-021	35 minutes ago	Completed	-	7.2s	models...	-
grandiose-gran...	35 minutes ago	Completed	-	2.3s	models...	sklearn
loud-flea-932	35 minutes ago	Completed	-	1.8s	models...	sklearn
luminous-gid-131	35 minutes ago	Completed	-	3.1s	models...	sklearn
leaky-worm-62	41 minutes ago	Completed	-	7.2s	models...	-
abundant-hawk...	41 minutes ago	Completed	-	2.9s	models...	sklearn
unfashioned-cobra...	42 minutes ago	Completed	-	1.7s	models...	sklearn
gifted-flea-23	42 minutes ago	Completed	-	2.7s	models...	sklearn
scuffy-worm-655	45 minutes ago	Completed	-	7.2s	models...	-
smelly-fish-34	48 minutes ago	Completed	-	2.4s	models...	sklearn
valuable-stall-516	48 minutes ago	Completed	-	1.7s	models...	sklearn
valuable-stall-210	48 minutes ago	Completed	-	3.0s	models...	sklearn
awesome-elf-10	52 minutes ago	Completed	-	6.8s	models...	-
marvelous-moose...	52 minutes ago	Completed	-	2.1s	models...	sklearn
adorable-goose...	52 minutes ago	Completed	-	1.7s	models...	sklearn
louny-dear-498	52 minutes ago	Completed	-	2.7s	models...	sklearn
stylish-slug-542	1 hour ago	Completed	-	6.8s	models...	-
sleepy-roo-473	1 hour ago	Completed	-	2.1s	models...	sklearn
powerful-flea-215	1 hour ago	Completed	-	1.8s	models...	sklearn
relaxed-roo-733	1 hour ago	Completed	-	2.7s	models...	sklearn