## Universidad de Los Andes

## Inteligencia de Negocios

## Informe Laboratorio 4

**Integrantes:**

1. Julián David Saenz (jd.saenz1519)
2. Juan Diego Barrios (jd.barriosc)
3. Alejandro Ahogado Prieto (a.ahogado)

## Contenido



### 1. Construcción de pipelines

### 1.1 Selección de modelo.

Para la selección del modelo se realizó una prospección inicial con el data set de prueba. Se consideraron dos tipos de modelado diferentes para comparar las métricas e implementar el que tuviera mejores resultados.

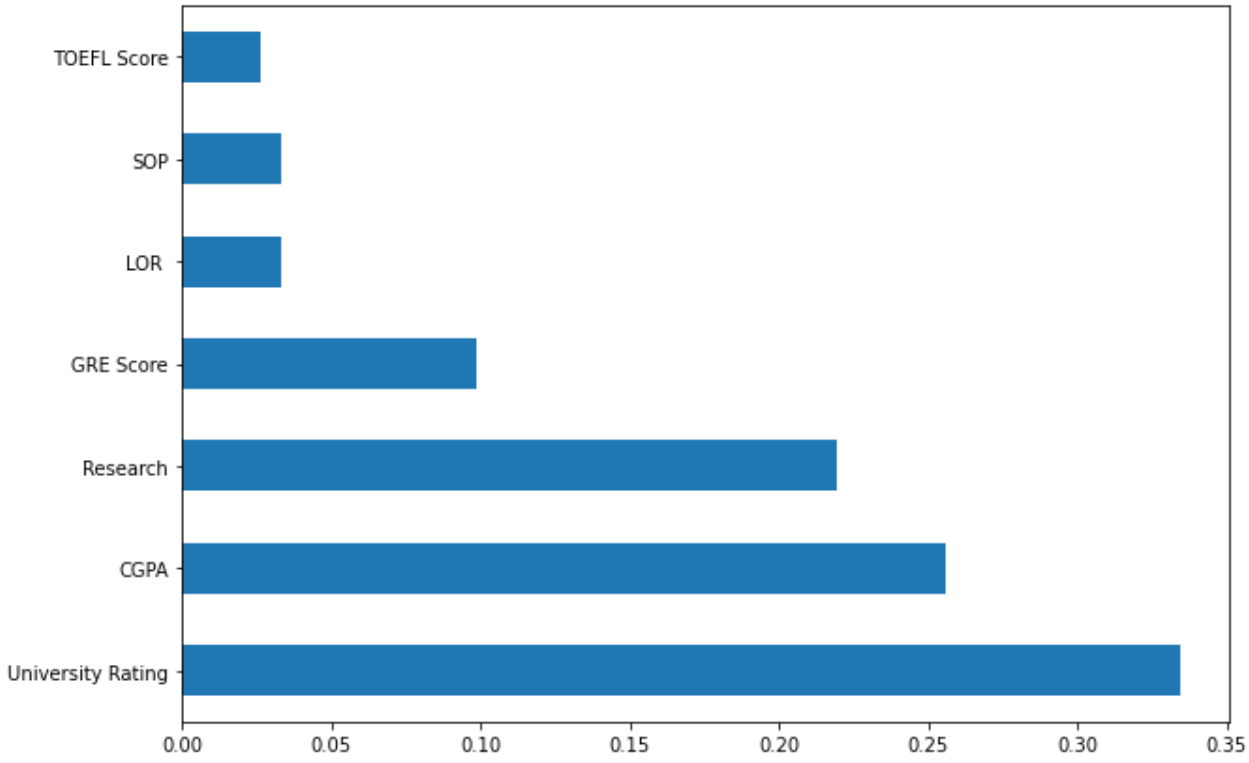
Resultados preliminares Regresión lineal:

```
RMSE on train data:  10.259749354151626
RMSE on test data:  10.931010255142235
```

Resultados preliminares Random Forest:

```
RMSE on train data:  3.99381831280534
RMSE on test data:  8.894300334132067
```

Adicionalmente se realizó una prospección de las variables para determinar el peso relativo que cada una tienen en la implementación del modelo. En este caso se determina que eliminar las variables menos relevantes ( TOEFL, SOP, LOR) no mejoraba considerablemente la calidad del modelo, pero si dificultaba la construcción de un pipeline. Para este caso se decidió utilizar la totalidad de las variables provistas.




### 1.2 Implementación del pipeline

La construcción del pipeline se realizo en tres pasos. Dado que los datos de entrada no contenían ninguna variable categórica fue posible omitir los pasos asociados a la transformación y el manejo de estas. Inicialmente se definen los features numéricos (que en este caso corresponde a la totalidad de los features) y se escalan.

```
[103] numeric_features = ['Serial No.','GRE Score',
                          'TOEFL Score','University Rating',
                          'SOP','LOR','CGPA', 'Research']
      numeric_transformer = Pipeline(steps=[
                                          ('poly',PolynomialFeatures(degree =3)),
                                          ('scaler', StandardScaler())])
```

Este preprocesamiento se almacena en una función que posteriormente se va a ejecutar en el pipeline

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)])
```

Finalmente se ejecuta el pipeline incluyendo los parámetros de transformación y preprocesamiento definidps anteriormente

```
X = df.drop('Admission Points', axis = 1)
y = df['Admission Points']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
pipeline.fit(X_train, y_train)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('poly',
                                                                   PolynomialFeatures(degree=3)),
                                                                  ('scaler',
                                                                   StandardScaler())]),
                                                  ['Serial No.', 'GRE Score',
                                                   'TOEFL Score',
                                                   'University Rating', 'SOP',
                                                   'LOR', 'CGPA',
                                                   'Research'])])),
                ('classifier',
                 RandomForestRegressor(max_depth=10, random_state=0))])
```

El resultado de este modelo se almacena en un archivo .joblib que posteriormente se va a usar para la construcción del API

## 2. Construcción del API

Para la construcción del API se decidió utilizar fastAPI como framework.

Inicialmente se definió cuál sería el endpoint que realizaría la predicción.

```python
import pandas as pd
from typing import List
from fastapi import FastAPI
from DataModel import DataModel
from PredictionModel import Model
from DataModelWithLabel import DataModelWithLabel

app = FastAPI()

@app.post("/predict")
def make_predictions(data: DataModel | List[DataModel] ):
    model = Model()
    if isinstance(data, list):
        return model.make_predictions(data)
    else:
        return model.make_prediction(data)
```

Este recibe objetos de la forma de un modelo definido previamente, el cual cuenta con los atributos del dataset sin tener en cuenta la variable objetivo (Admission score), dejando una clase con los demás atributos.

```python
from pydantic import BaseModel

class DataModel(BaseModel):

# Estas varibles permiten que la librería pydantic haga el parseo entre el Json recibido y el modelo declarado.
    serial_no: float
    gre_score: float
    toefl_score: float
    university_rating: float
    sop: float
    lor: float
    cgpa: float
    research: float

#Esta función retorna los nombres de las columnas correspondientes con el modelo esxportado en joblib.
    def columns(self):
        return ["Serial No.","GRE Score","TOEFL Score","University Rating","SOP","LOR" ,"CGPA","Research"]
```

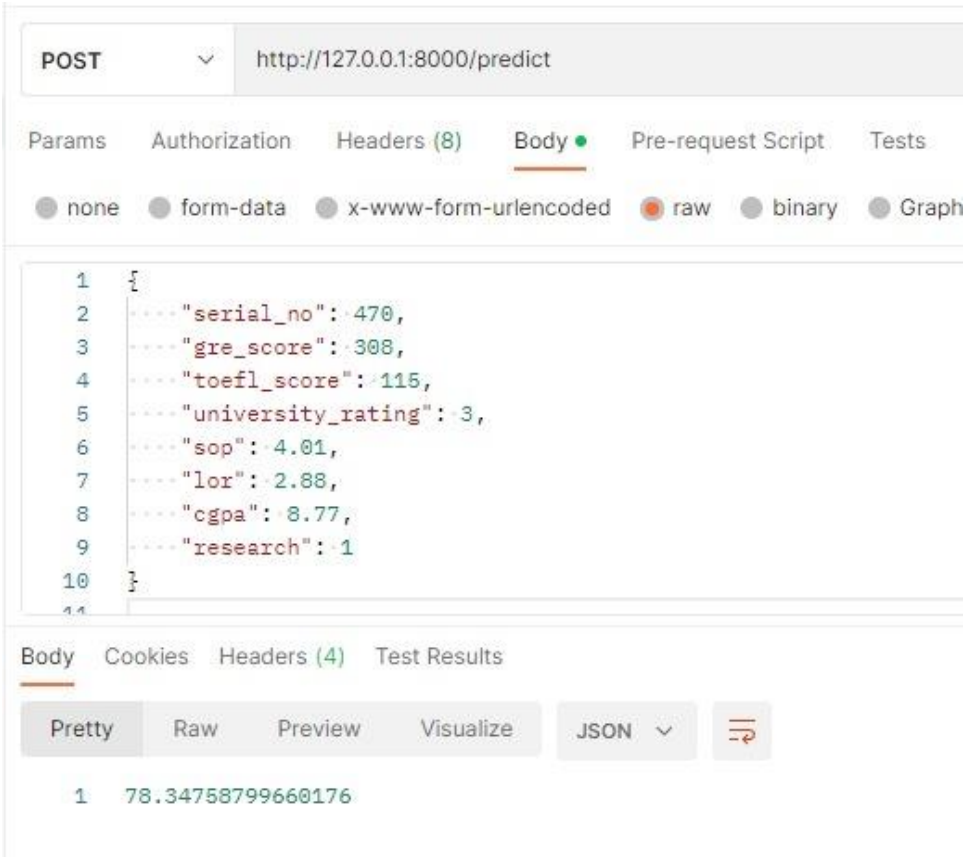
Después este toma dependiendo de si se envió un solo documento o un arreglo de documentos y los preprocesa haciendo uso de la librería joblib como se puede ver en la siguiente imagen para posteriormente retornárselo al usuario.

```python
import pandas as pd
from joblib import load

class Model:

    def __init__(self):
        self.model = load("assets/pipeline.joblib")

    def make_prediction(self, dataModel):
        df = pd.DataFrame(dataModel.dict(), columns=dataModel.dict().keys(), index=[0])
        df.columns = dataModel.columns()
        result = self.model.predict(df)[0]
        return result

    def make_predictions(self, dataModels):
        results = []
        for dataModel in dataModels:
            result = self.make_prediction(dataModel)
            results.append(result)
        return results
```

## 3. Escenarios de prueba y conclusiones

**Escenario1.**

**En este escenario se prueba el modelo con valores correctos en su debido formato, es por esta razón que se obtiene un resultado acorde a lo esperado.**

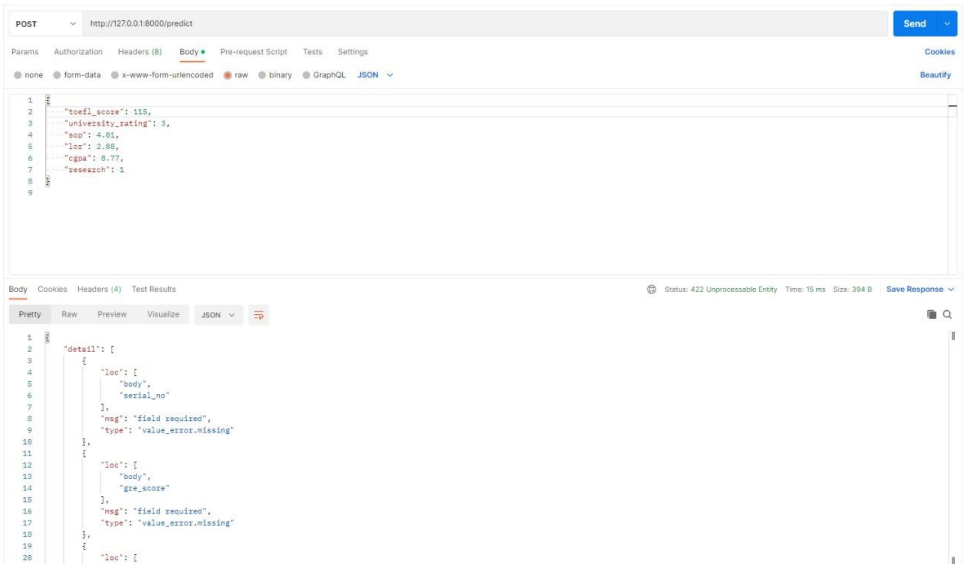



**Datos de prueba:**


```
{
   "serial_no": 470,
   "gre_score": 308,
   "toefl_score": 115,
   "university_rating": 3,
   "sop": 4.01,
   "lor": 2.88,
   "cgpa": 8.77,
   "research": 1
}
```


**Escenario 2.**

**En este escenario se prueba el modelo sin dos atributos correspondientes, por lo que el modelo no logra ejecutarse al no contar con todos los datos necesarios (serial_ no y gre_score).**

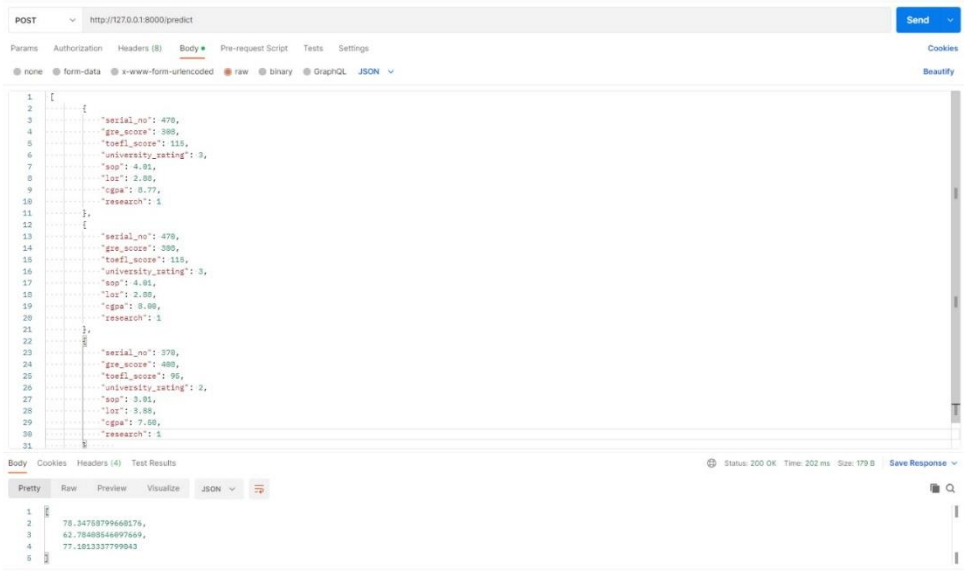



**Datos de prueba:**


```
{
   "toefl_score": 115,
   "university_rating": 3,
   "sop": 4.01,
   "lor": 2.88,
   "cgpa": 8.77,
   "research": 1
}
```


**Escenario 3.**

**En este escenario se prueba el modelo con 3 conjuntos de datos correctos. Al contar todos estos con el formato se evidencia que se obtienen los resultados esperados y se logra predecir el puntaje espetado correctamente.**



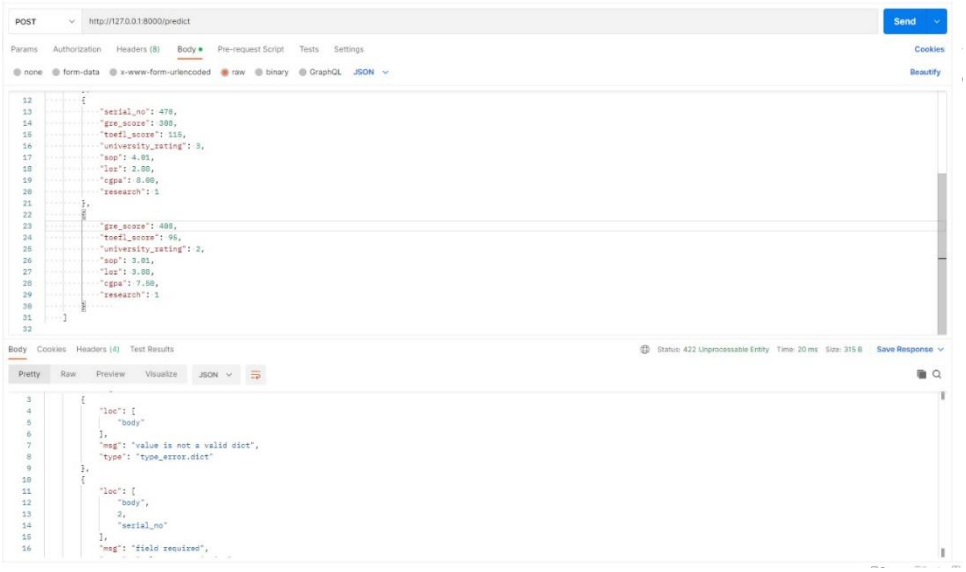

**Datos de prueba:**


```
[
    {
      "serial_no": 470,
      "gre_score": 308,
      "toefl_score": 115,
      "university_rating": 3,
      "sop": 4.01,
      "lor": 2.88,
      "cgpa": 8.77,
      "research": 1
    },
    {
      "serial_no": 470,
      "gre_score": 308,
      "toefl_score": 115,
      "university_rating": 3,
      "sop": 4.01,
      "lor": 2.88,
      "cgpa": 8.00,
      "research": 1
    },
    {
      "serial_no": 370,
      "gre_score": 408,
      "toefl_score": 95,
      "university_rating": 2,
      "sop": 3.01,
      "lor": 3.88,
```

```
    "cgpa": 7.50,
    "research": 1
  }
]
```

**Escenario 4.**

**En este escenario se prueba el modelo con 3 conjuntos de datos, de los cuáles 2 se encuentran en el formato correcto pero al último conjunto le hace falta un parámetro necesario (serial_no), por lo que el modelo no logra ejecutarse de forma correcta.**



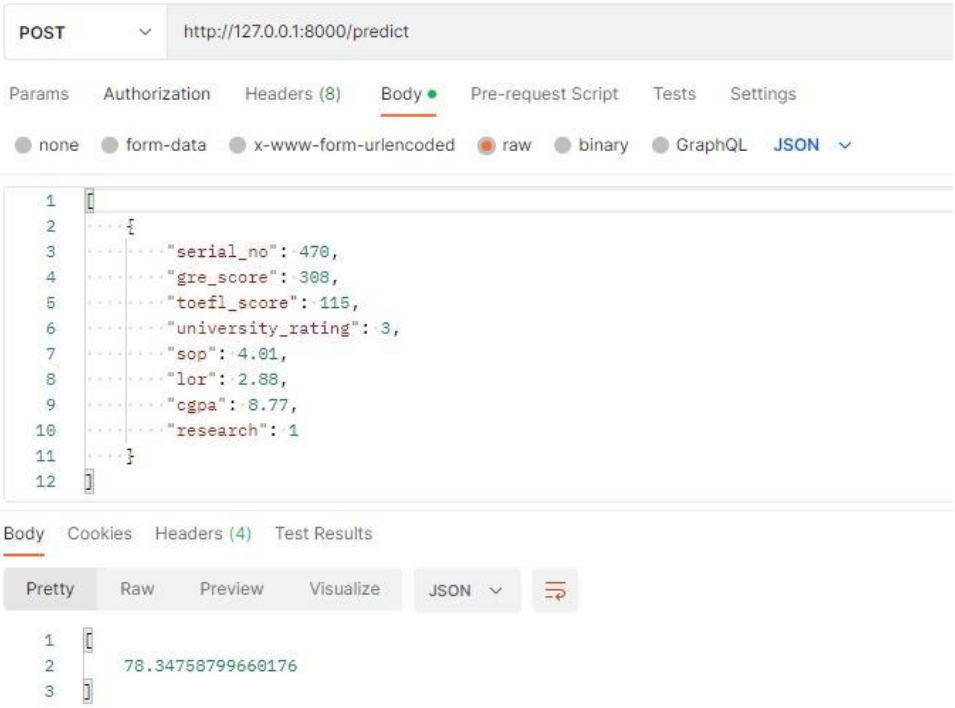

**Datos de prueba:**

```
[
    {
      "serial_no": 470,
      "gre_score": 308,
      "toefl_score": 115,
      "university_rating": 3,
      "sop": 4.01,
      "lor": 2.88,
      "cgpa": 8.77,
      "research": 1
    },
    {
      "serial_no": 470,
      "gre_score": 308,
      "toefl_score": 115,
      "university_rating": 3,
      "sop": 4.01,
      "lor": 2.88,
      "cgpa": 8.00,
```

```
    "research": 1
  },
  {
    "gre_score": 408,
    "toefl_score": 95,
    "university_rating": 2,
    "sop": 3.01,
    "lor": 3.88,
    "cgpa": 7.50,
    "research": 1
  }
]
```


**Escenario 5.**

**En este escenario se prueba el modelo con nuevos datos, para poder verificar el correcto funcionamiento de este al brindar resultados diferentes, ya que estos deben ser acorde a los parámetros recibidos.**




**Datos de prueba:**


```
[
  {
    "serial_no": 470,
    "gre_score": 308,
    "toefl_score": 115,
    "university_rating": 3,
    "sop": 4.01,
```

```
      "lor": 2.88,
      "cgpa": 8.77,
      "research": 1
    }
]
```


**3.2 Conclusiones/ observaciones generales.**

En este caso la producción de una API para la producción de predicciones permite que los modelos generados con técnicas de ML sean más utilizables en un contexto de negocio. Desplegar los resultados por medio de una API permite que estos sean utilizados por el cliente y que sean mas accesibles por un rango amplio de personas sin necesidad de tener conocimientos sobre implementaciones especificas

**4. Estrategia para mitigación de incoherencias y errores**

De acuerdo al contexto en el que se desarrolla el problema, existe una gran posibilidad de que se generen incoherencias o predicciones erróneas en los resultados del modelo por la inconsistencia de los datos recibidos. De igual manera, pueden llegar a presentarse errores que impidan la ejecución del algoritmo si no se cuenta con la suficiente claridad de los atributos necesarios para su correcto funcionamiento. Es por esta razón que deben implementarse estrategias que permitan mitigar la existencia de todas estas inconsistencias, un ejemplo de esto es el proceso que debe seguirse tras recibir el archivo de datos, sobre el cuál debería verificarse que por cada información recibida, asociada a una variable, esta cuente con el tipo de dato y formato esperado (por ejemplo, si se está esperando una variable binaria, que esta solo tenga los valores 0 o 1). Así mismo, es importante verificar la existencia de todas las variables, para el funcionamiento óptimo del modelo y en caso de que no se encuentren todas las variables, poder manejar el problema e informar sobre este.