

# PROYECTO FINAL INDIVIDUAL

## Predicción de la edad de los abulones

---



### INTRODUCCIÓN

En el presente documento se describe una solución MLOps para un 'dataset' llamado 'Abalon.data' que es un conjunto de datos clásico utilizado en el aprendizaje automático, que contiene información sobre abulones, un tipo de molusco marino. Cada abulón tiene características asociadas, como la longitud, el diámetro, el peso de la carne, el peso de las vísceras, el peso del caparazón, etc.

El objetivo típico al trabajar con este conjunto de datos es predecir la edad de un abulón; ésta se puede determinar cortando la concha a través del cono, tiñendola y contando el número de anillos a través de un microscopio (como el tronco de un árbol) y sumándole el

---

---

factor 1.5, una tarea tediosa que consume mucho tiempo. Otra forma de hacerlo es mediante las medidas físicas mencionadas, que son más fáciles de obtener y con las que se puede estimar el número de anillos y de ahí la edad del abulón.

.

---

## A) DESARROLLO DE LA SOLUCIÓN PROPUESTA

### A1.-Análisis y comprensión del dataset proporcionado mediante un Análisis

#### Exploratorio de Datos

La información de las variables contenidas en el dataset "Abalon.data" es la siguiente:

NOMBRE VARIABLE	ROLE	TIPO	DESCRIPCIÓN	UNIDADES
Sex	Atributo	Categórica	M, F, I (Infante)	
Length	Atributo	Continua	Medida más larga del caparazón	mm
Diameter	Atributo	Continua	Diámetro, perpendicular a la longitud	mm
Height	Atributo	Continua	Altura de carne con caparazón	mm
Whole_weight	Atributo	Continua	Peso del abulón completo	grms
Shucked_weight	Atributo	Continua	Peso de la carne	grms
Viscera_weight	Atributo	Continua	Peso de las vísceras	grms
Shell_weight	Atributo	Continua	Peso del caparazón	grms
Rings	Objetivo	Entera	Anillos	

El problema puede ser abordado al menos de dos maneras, la primera a partir de una solución de tipo clasificación tomando a la variable objetivo 'rings' como una variable categórica ordinal y la segunda a partir de una solución de tipo regresión si se le toma como una variable no categórica continua (aunque es un tipo discreta); ésto último

---

implicará que en algún momento tengan que redondearse las predicciones, lo que pudiera introducir un pequeño error de precisión aunque haya efectos cancelatorios de redondear a veces hacia arriba y en otras ocasiones hacia abajo.

Se propone una solución MLOps de tipo regresión para resolver el problema de encontrar la edad de un abulón a partir de los atributos que se encuentran en el 'dataset'.

Específicamente se desea predecir la cantidad de anillos que tiene si se conocen sus otras características físicas. Obviamente se tendrá que explorar el 'dataset' para encontrar y seleccionar el menor número posible de éstas características sin perder información importante de las características no seleccionadas.

Las variables de entrada más importantes pueden variar según el enfoque del análisis y la aplicación específica del modelo. Sin embargo, la longitud y el diámetro suelen ser consideradas como dos de las variables más importantes para predecir la edad de los abulones. debido a que son medidas físicas directas del abulón muy relacionadas con su crecimiento y desarrollo. En general, se espera que abulones más grandes tengan más anillos y, por lo tanto, sean más viejos. Así que, esas variables son dos candidatas naturales para ser consideradas como importantes para predecir la edad de los abulones. Dependiendo del contexto el peso total o del caparazón también puede ser indicadores de la salud del abulón y por lo tanto de su edad.

Al realizar un análisis exploratorio de los datos se encontró lo siguiente:

- En la imagen (Fig 1: Histograma por variable atributo y variable objetivo) con los histogramas de las variables de entrada se nota que tanto la la variable de salida
-

---

como las variables de entrada con una distribución normal más evidente son:

'shell\_weigth', 'diameter' y 'lenght'

- En la figura (Fig 2: Mapa Calor Matriz Factores Correlación) de los mapas de calor se nota que los atributos más correlacionados con la variable objetivo 'ring' son nuevamente 'shell\_weigth', 'diameter', 'length' y 'height' siendo 'shell\_weigth' y 'diameter' las de coeficientes más altos. También son dos variables muy correlacionadas con los demás atributos (Fig 4: Correlaciones Pares Variables Tipo Atributo y Variable Objetivo), lo que implica que de éstas dos variables se pueden deducir las restantes y no considerar atributos redundantes sin perder información relevante
- En la imagen (Fig 3: Diagramas Caja Variables Tipo Atributo) con los gráficos de caja para las variables de atributos para 'shell\_weigth' y 'diameter' se nota que existen valores atípicos que se pueden eliminar, en la parte superior para 'shell\_weigth' de 0.95 a 1, y en la parte inferior para 'diameter' de 0 a 0.075) sin afectar a las demás variables
- En la imagen (Fig 4: Correlaciones Pares Variables Tipo Atributo y Variable Objetivo) se puede notar visualmente que hay correlación entre las variables de atributos 'shell\_weigth' y 'diameter' con las demás variables atributo, es decir se puede prescindir de los segundos porque se comportan de manera parecida a los primeros

Debido a que la variable objetivo 'age' no existe en el dataset 'Abalon.data', es posible crearla como una nueva columna a partir de la variable 'rings' y entonces se puede deducir a partir de ésta columna sumando el factor 1.5 y de algunas otras columnas de características, sin embargo, se desechó esa opción pues se consideró una redundancia

---

---

En conclusión, se determinó seleccionar al diámetro ('diameter') y al peso del caparazón del abulón ('shell\_weigth') como las dos características más relevantes para predecir el número de anillos de la variable objetivo. De manera implícita se obtendrá la edad del Abulón que es la pregunta que se debe responder en última instancia

## **A2.-Determinación de la pregunta que se desea contestar con un modelo de aprendizaje automático**

En la sección anterior se concluyó que las dos variables de tipo atributo más relevantes en el dataset 'Abalon.data' con las que se puede predecir la variable objetivo son 'shell\_weigth' y 'diameter'. Por lo tanto la pregunta que se desea contestar con un modelo de aprendizaje automático para predecir la variable objetivo es: ¿Cuántos anillos tendrá un abulón conocidos el peso de su caparazón y su diámetro?. De manera implícita se obtendrá la edad del abulón pues solo bastará sumar el factor 1.5 al número de anillos obtenido

---

### **A3.-Identificar por qué se necesita una estrategia de MLOps para éste dataset**

El objetivo principal del 'bootcamp' es aprender cómo aplicar un modelo con una estrategia MLOps a un 'dataset'. Sin embargo, específicamente para las predicciones del modelo de aprendizaje automático del dataset 'Abalon.data', mi punto de vista es la siguiente :

- El problema a resolver es relativamente simple, sin estructuras de datos altamente complejas y con muy pocos atributos. Considero que no se requiere una implementación sofisticada en producción con un flujo completo de trabajo de MLOps
  - El proyecto no tiene requisitos de producción exigentes, como escalabilidad, mantenibilidad, cumplimiento de regulaciones ni de seguridad. No se espera que el uso del modelo ni el tamaño de los datos tengan el potencial de crecer en el futuro ni que se vayan a desplegar muchos modelos que se deban mantener en producción a largo plazo
  - No requiere de un sistema en tiempo real ya que no maneja grandes volúmenes de datos ni que requiera de actualizaciones periódicas, pues la información parece formar parte de un proyecto de investigación que quedará estática. Al no esperar actualizaciones frecuentes implica que el modelo no necesitará reentrenamientos constantes. De hecho, en la página web donde se encuentra 'Abalon.data' se tiene la leyenda 'Expected update frequency' marcada con 'Never'
  - Sus predicciones no son de naturaleza crítica, pues no tiene impactos comerciales o legales. Tampoco incurre en costos financieros por errores en las predicciones. Sus datos tampoco son sensibles ni privados
-

**NOTA:**

Obviamente, por cuestiones de formación y de aprendizaje de un tema tan importante, sí se adoptará una estrategia MLOPs para las predicciones del dataset 'Abalon.data'

---



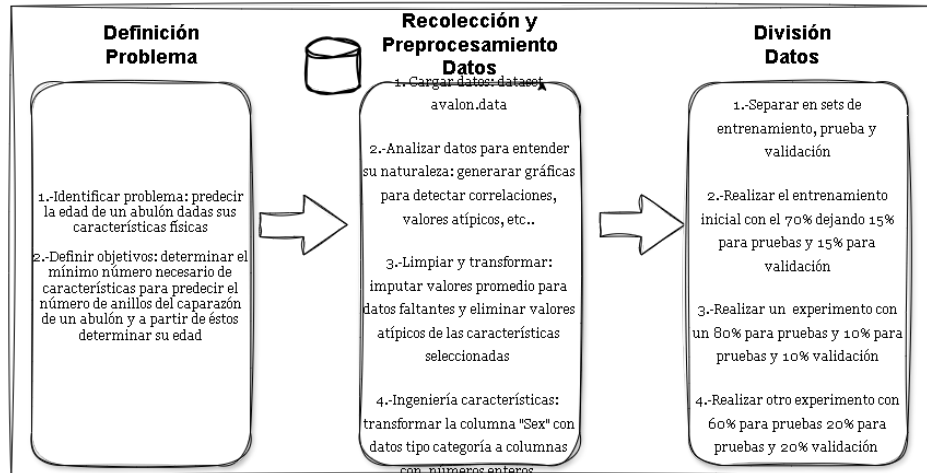
#### **A4.-Arquitectura del pipeline para ésta iniciativa de aprendizaje automático**

A4.1) Básicamente se separará en 3 etapas principales: preparación de datos, creación del modelo y el despliegue del mismo, como se ve enseguida:

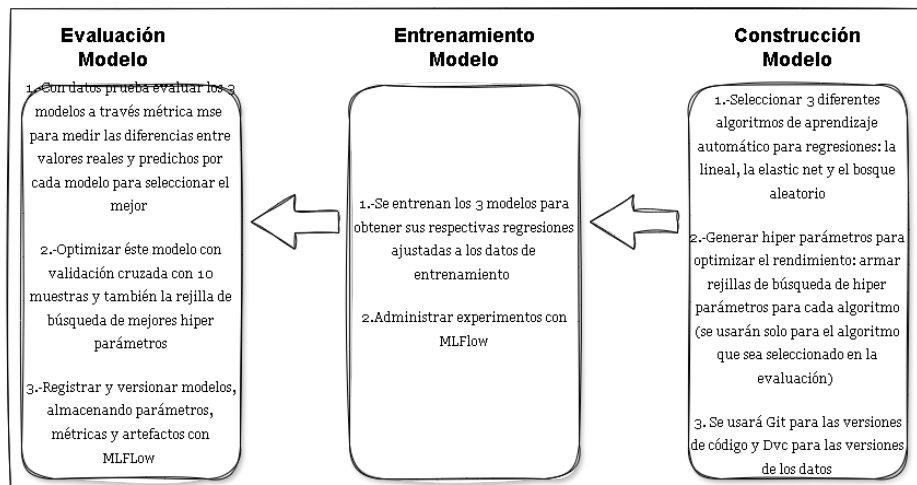
---

## MLOPS

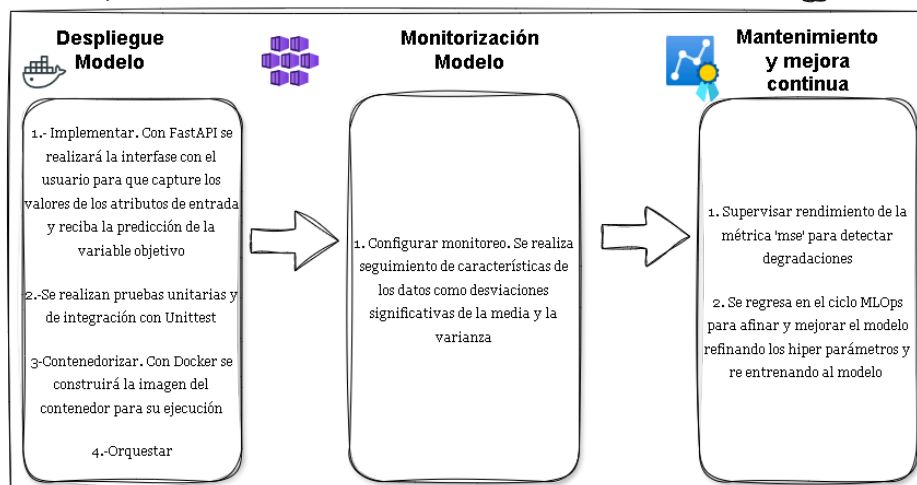
### PREPARACION DATOS



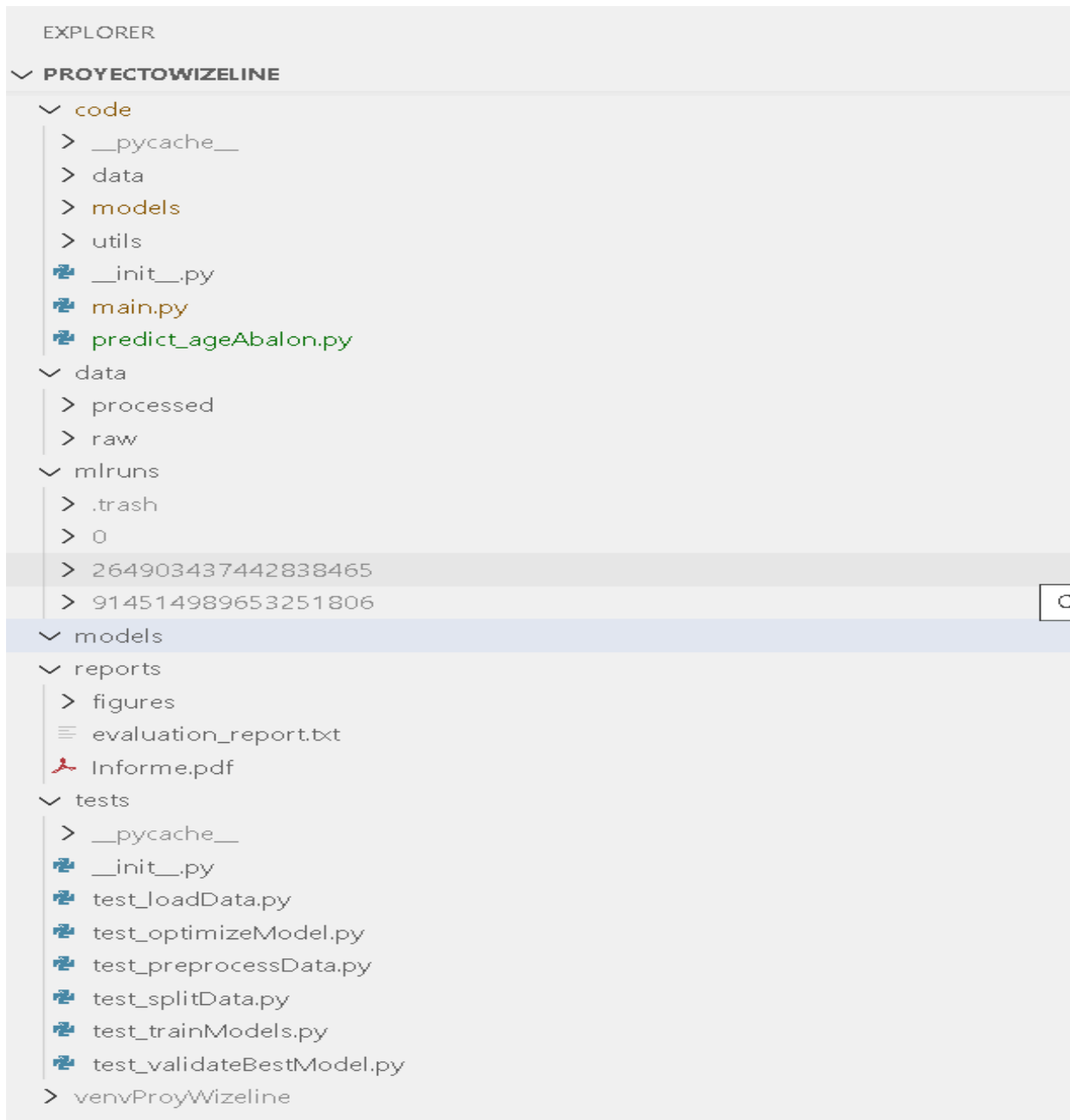
### CREACION MODELO



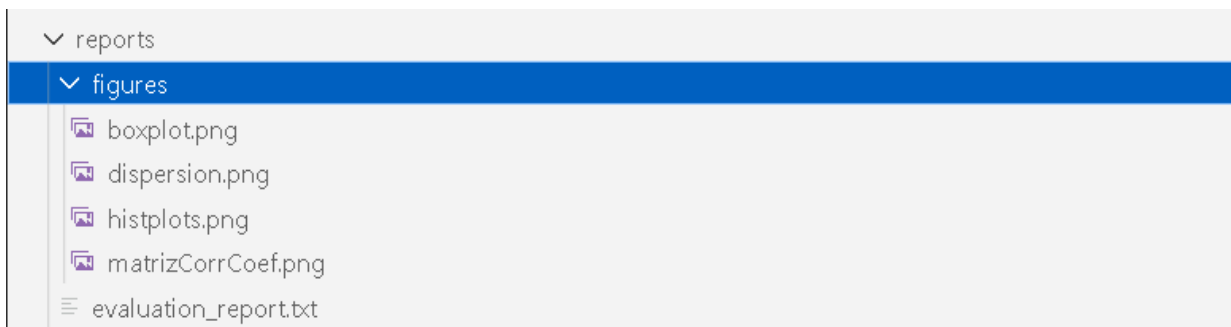
### DESPLIEQUE




A4.2) Una tarea importante en éste punto es que se realizó la refactorización del script original de Python separándolo en módulos y en carpetas para distinguir la preparación de datos, el modelo en sí y el despliegue



A4.3) Entre algunas otras cosas, las gráficas generadas dinámicamente en tiempo de ejecución se guardan en la carpeta 'reports\figures\' y en la carpeta 'reports\' se guarda 'evaluation\_report' con los resultados más importantes de la corrida, como se ve enseguida



 evaluation\_report: Bloc de notas

Archivo Edición Formato Ver Ayuda

Reporte de Validación de Modelos

Test mse

Linear Regression	5.629770992366412
Elastic Net	8.13740458015267
Random Forest	5.179389312977099

El mejor modelo optimizado es: RandomForestRegressor con mse: 5.0582959641255

**A5.-Crear un modelo base para abordar tareas de predicción relacionadas con la pregunta. Este modelo no necesita una alta precisión, recall o puntuación F1; el objetivo es crear un modelo rápido para iteración.**

#### A5.1) Creación modelo base

Se creó un modelo de 'machine learning' en el que se eligieron los siguientes algoritmos para responder a las tareas de predicción de los anillos y edad de una abulón a partir del peso del caparazón y su diámetro:

- 'linear regression',
- 'elastic net' y
- 'random forest'

En los 3 algoritmos se trataron de elegir hiper parámetros con valores default y los 3 se entrenaron con el mismo 'dataset' de entrenamiento ('train') del 70% del total de datos

#### A5.2) Etapas de prueba

Enseguida con los datos de prueba ('test') se prueban los 3 algoritmos realizando predicciones del número de anillos de los abalones y al compararlos contra los datos reales de ese set de prueba se obtuvo la métrica 'mse' para los 3, determinando que el algoritmo 'random forest' era el que minimizaba los errores

---

### A5.3) Etapas de validación

Para la etapa de validación del algoritmo 'random forest', se realizaron 2 experimentos con él, variando el tamaño de los datos de prueba y los hiper parámetros a mediante dos 'gridSearch'. Los dos experimentos optimizan la métrica 'mse' con lo que se tuvo la seguridad de que el modelo estaba generalizando adecuadamente

### A5.4) Registro de experimentos

Se decidió utilizar a 'MLFlow' para el registro de los experimentos, con sus corridas, sus hiper parámetros y sus métricas con la finalidad de versionalizar y reproducir al experimento elegido. En la carpeta 'mlruns' del proyecto, 'mlflow' guarda todo lo relacionado con la ejecución de los experimentos, incluidas corridas, hiper parámetros, métricas, artefactos y otros archivos de configuración como requerimientos y dependencias. Nótese:

---



#### A5.5) Predicciones

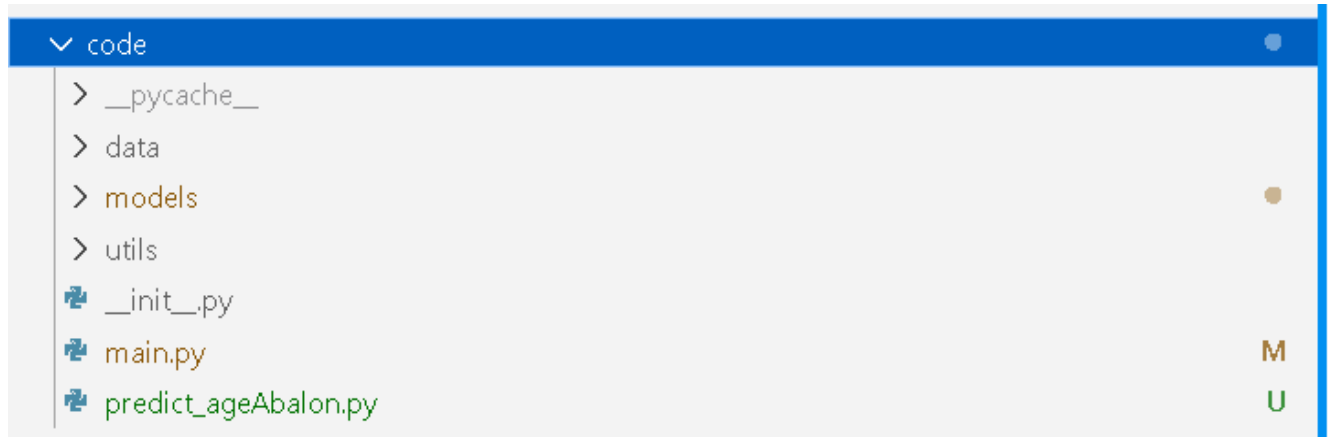
Se utilizará la librería 'FAstAPI' para realizar la interfaz entre el usuario y el modelo. Este tiene las características principales siguientes :

- rendimiento: Muy rápido, soporta programación asíncrona
- validación automática: Basada en las anotaciones de tipo de Python
- documentación: Genera automáticamente documentación interactiva
- facilidad de uso: Sintaxis intuitiva y limpia
- integración: compatible con herramientas como 'SQLAlchemy' y 'OAuth'

En resumen, 'FastAPI' permite desarrollar APIs de manera rápida y eficiente, con validación automática, documentación interactiva y un rendimiento excepcional

También se utilizará para el mismo objetivo a la librería 'Pydantic' que es una biblioteca de validación de datos y configuración para Python, que se enfoca en proporcionar una forma rápida y sencilla de definir y validar estructuras de datos mediante el uso de anotaciones de tipo. Es ampliamente utilizada junto con frameworks como 'FastAPI' para manejar la validación de datos de entrada de manera eficiente y segura

Respecto a la forma en cómo el usuario puede utilizar el modelo para predicciones, se desarrolló el script de python 'predict\_ageAbalon.py' que hace uso de las dos librerías mencionadas para ofrecer al usuario una interfaz sencilla para enviar al modelo los valores de las variables de entrada (los atributos 'Shell\_weight' y 'Diameter' de un abulón), cargar y ejecutar el modelo ya entrenado y regresar el resultado de la variable objetivo ('Rings' y por deducción 'Age' del abulón)



Basta ejecutar en la terminal el comando:

**uvicorn code.predict\_ageAbalon:app --reload**

Y en el navegador de internet revisar la 'FastAPI' en la dirección:



<http://127.0.0.1:8000/docs>



default



**POST** /predict\_abalon Predict Abalon



**GET** / Read Root



La ruta GET solo da la bienvenida a la API. La ruta POST es la que permite enviarle al modelo la características del abulón del cual deseamos predecir su número de anillos y por deducción su edad. Solo debemos abrir la ruta del POST y ejecutarla con 'Try it out' para indicar los valores de entrada (en nuestro ejemplo será 0.114 grs de peso del caparazón y 0.295 mm de diámetro del caparazón) y dar clic sobre el botón 'Execute' para que el modelo se ejecute:

Request body required

application/json

```
{  
  "Shell weight": 0.114,  
  "Diameter": 0.295  
}
```

Execute

Lo anterior mostrará una sección 'responses' con el resultado de la predicción, en éste ejemplo el número de anillos es 9 y por lo tanto la edad del abulón es de 10.5 años:

**Responses**


**Curl**

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict_abalon' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "Shell_weight": 0.114,
    "Diameter": 0.295
  }'
```

**Request URL**

```
http://127.0.0.1:8000/predict_abalon
```

**Server response**

Code	Details
200	<div><b>Response body</b><pre>{   "predicted_rings": 9,   "predicted_age": 10.5 }</pre><div> <b>Download</b></div></div>

## A5.6) Contenedores

En relación a la contenedorización, se instaló la 'Docker Community Edition (Docker CE) que se utiliza en plataformas de Windows, macOS y Linux y que es una edición gratuita y de código abierto de Docker Engine, diseñada para desarrolladores individuales y equipos pequeños que buscan experimentar con contenedores y crear aplicaciones

---

contenedorizadas. Ofrece actualizaciones frecuentes y es ideal para entornos de desarrollo y prueba

Una vez que se ha construido y ejecutado la imagen Docker utilizando el Dockerfile que se proporcione, se creará una instancia de contenedor que contiene todos los componentes del proyecto de ML, incluyendo por ejemplo la aplicación FastAPI, el servidor MLFlow y las pruebas unitarias. Esta instancia de contenedor se ejecutará en la máquina local o en el entorno donde se utilice Docker.

En cuanto a subir o no subir la carpeta `docker` a GitHub, se sugiere subir por varias causas:

- Asegura que todos los desarrolladores y sistemas CI/CD usen la misma configuración de Docker
  - Facilita la colaboración entre desarrolladores, quienes podrán revisar y mejorar la configuración de Docker
  - Permite la integración con servicios de CI/CD que pueden construir y desplegar automáticamente la imagen de Docker
  - La configuración de Docker y los archivos relacionados son parte de la documentación del proyecto, proporcionando contexto sobre cómo construir y ejecutar el proyecto en un entorno contenedorizado
-

Básicamente debajo de la carpeta principal del proyecto, en nuestro caso 'ProyectoWizeLine', se creó una carpeta llamada 'docker' donde que básicamente contiene dos archivos:

- dockerfile, que es el archivo donde se define como construir la imagen de docker para el proyecto. A continuación se muestra las instrucciones que se agregaron:

```
# Base de la imagen para el proyecto ML
```

```
FROM python:3.12.1-slim
```

```
# Establecer el directorio de trabajo
```

```
WORKDIR /app
```

```
# Copiar el archivo de requerimientos antes de instalar las dependencias
```

```
COPY requirements.txt .
```

```
# Crear un entorno virtual en el contenedor
```

```
RUN python -m venv venvProyWizeline
```

```
# Instalar las dependencias dentro del entorno virtual
```

```
RUN venvProyWizeline/bin/pip install --upgrade pip && \
```

```
    venvProyWizeline/bin/pip install -r requirements.txt
```

```
# Copiar todos los archivos necesarios al directorio de trabajo
```

```
COPY . .
```

```
# Exponer los puertos necesarios para FastAPI
```

EXPOSE 80

# Instalar MLFlow dentro del entorno virtual

RUN venvProyWizeline/bin/pip install mlflow

# Configurar la variable de entorno MLFLOW\_TRACKING\_URI para almacenar los registros en un directorio dentro del contenedor

ENV MLFLOW\_TRACKING\_URI file:///mlflow

# Indicar un volumen Docker para persistir los registros de experimentos

VOLUME /mlflow

# Instalar DVC en el entorno virtual

RUN venvProyWizeline/bin/pip install dvc

# Inicializar DVC

RUN . venvProyWizeline/bin/activate && dvc init -f

# Configurar el remoto de DVC

RUN . venvProyWizeline/bin/activate && dvc remote add -d myremote /mlflow

# Añadir archivos de datos a DVC

RUN . venvProyWizeline/bin/activate && dvc add /app/data/raw/abalone.data

RUN . venvProyWizeline/bin/activate && dvc add  
/app/data/raw/abaloneNuevo.data

# Hacer push de los datos a DVC

---

---

```
RUN . venvProyWizeline/bin/activate && dvc push
```

```
# Ejecutar `dvc pull` para obtener los datos
```

```
RUN . venvProyWizeline/bin/activate && dvc pull
```

```
# Establecer el directorio de trabajo para la aplicación
```

```
WORKDIR /app
```

```
# Establecer PYTHONPATH
```

```
ENV PYTHONPATH=/app
```

```
# Establecer el comando CMD para ejecutar el comando "fastapi"
```

```
CMD ["venvProyWizeline/bin/python", "/app/code/main.py",  
"/app/data/raw/abalone.data.dvc", "/app"]
```

- .dockerignore, que es el archivo para especificar los archivos y directorios que no deben copiarse en la imagen de Docker. Esto ayuda a mantener la imagen ligera y evitar incluir archivos innecesarios. Enseguida se muestra lo que se ignoró en éste proceso específico

```
# Archivos y directorios específicos del sistema operativo
```

```
.DS_Store
```

```
Thumbs.db
```

```
# Control de versiones
```

---

.git

.gitignore

.gitattributes

# Dependencias locales y archivos compilados

\_\_pycache\_\_

\*.pyc

\*.pyo

\*.pyd

# Entornos virtuales

venv/

env/

# Archivos de configuración locales

.env

.env.\*

# Logs y temporales

\*.log

\*.tmp

# Archivos y directorios específicos de IDE

.vscode/

---



```
.idea/  
  
# Datos grandes  
  
data/  
  
datasets/  
  
*.h5  
  
*.csv  
  
*.tsv  
  
*.pkl  
  
# Directorios de salida de compilación  
  
build/  
  
dist/  
  
*.egg-info
```

Se levanta a la aplicación 'Docker Desktop' para revisar las imágenes de docker que se van creando

Después que ambos archivos se crearon, se ejecutó el siguiente comando en la consola para crear la imagen :

```
docker build -t proywizelineimg -f docker/dockerfile .
```

---

PROBLEMS OUTPUT TERMINAL JUPYTER GITLENS

powershell + v

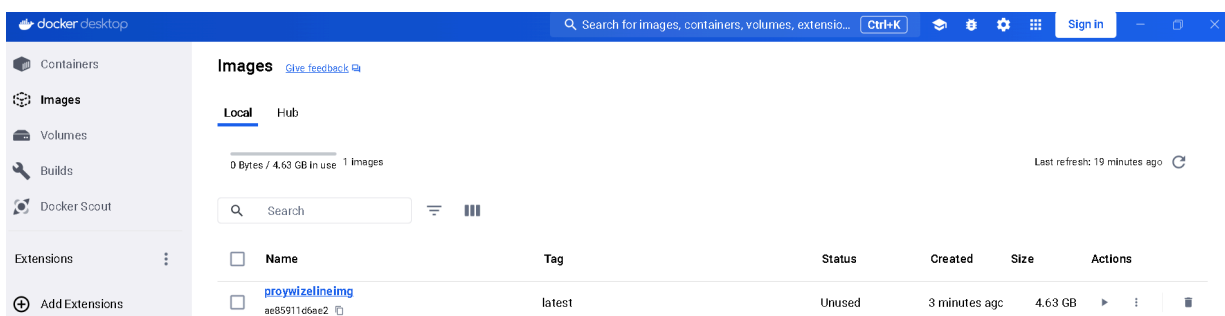
```
(venvProyWizeline) PS C:\Users\52477\Documents\ProyectoWizeline> docker build -t proywizelineimg -f docker/dockerfile .
[+] Building 0.0s (0/0) docker:default
[+] Building 4.0s (20/20) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 1.93kB
=> [internal] load metadata for docker.io/library/python:3.12.1-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [ 1/15] FROM docker.io/library/python:3.12.1-slim@sha256:a64ac5be6928c6a94f00b16e09cdf3ba3edd44452d10ffa4516a58004873573e
=> [internal] load build context
=> => transferring context: 3.22MB
=> CACHED [ 2/15] WORKDIR /app
=> CACHED [ 3/15] COPY requirements.txt .
=> CACHED [ 4/15] RUN python -m venv venvProyWizeline
=> CACHED [ 5/15] RUN venvProyWizeline/bin/pip install --upgrade pip && venvProyWizeline/bin/pip install -r requirements.txt
=> CACHED [ 6/15] COPY . .
=> CACHED [ 7/15] RUN venvProyWizeline/bin/pip install mlflow
=> CACHED [ 8/15] RUN venvProyWizeline/bin/pip install dvc
=> CACHED [ 9/15] RUN . venvProyWizeline/bin/activate && dvc init -f
=> CACHED [10/15] RUN . venvProyWizeline/bin/activate && dvc remote add -d myremote /mlflow
=> CACHED [11/15] RUN . venvProyWizeline/bin/activate && dvc add /app/data/raw/abalone.data
=> CACHED [12/15] RUN . venvProyWizeline/bin/activate && dvc add /app/data/raw/abaloneNuevo.data
=> CACHED [13/15] RUN . venvProyWizeline/bin/activate && dvc push
=> CACHED [14/15] RUN . venvProyWizeline/bin/activate && dvc pull
=> CACHED [15/15] WORKDIR /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:86cdcc6f957357158e56cfb056de6a4f7d04a5d619d3a81c9603f6e06dae00c4
=> => naming to docker.io/library/proywizelineimg
```

What's Next?

View a summary of image vulnerabilities and recommendations → [docker scout quickview](#)

```
(venvProyWizeline) PS C:\Users\52477\Documents\ProyectoWizeline>
```

Entonces es posible consultar las imágenes existentes en el 'Docker Desktop'



Desde ésta interfaz se puede ejecutar el contenedor o con el comando siguiente:

```
docker run --rm -p 80:80 proywizelineimg
```

---

Nota: es muy importante que el nombre de la imagen sea en letras minúsculas

---

## **B) OTROS ASPECTOS DE LA SOLUCIÓN PROPUESTA**

### **B1.-¿Qué elementos matemáticos se consideraron en esta decisión?**

- En la parte de la exploración de datos básicamente se consideraron análisis de entre pares de variables como los coeficientes de correlación y las dispersiones de los datos y también estadísticas básicas como la media y la varianza
  - En la parte de la selección del algoritmo de aprendizaje se consideró el error cuadrático medio (mse), que es una medida comúnmente utilizada para evaluar la precisión de un modelo del tipo abordado. Es una métrica útil para comparar diferentes modelos o ajustes de un mismo modelo y que suma los cuadrados de las diferencias entre los valores reales y los predichos
  - En la parte de la evaluación del desempeño del modelo se utilizó 'Cross-Validation' (Validación Cruzada), que es una técnica de dividir los datos en múltiples subconjuntos y entrenar y validar el modelo en diferentes combinaciones de estos subconjuntos
-

## B2.-¿Cómo se integrarán nuevos datos?

Dada la naturaleza crítica del versionamiento, trazabilidad y reproducibilidad de los experimentos que se realizan en un modelo MLOps, en modelo que nos atañe se utiliza “git” para versionar el código del modelo y “dvc” para versionar los datos asociados a una versión del código. El flujo básico entre “git” y “dvc” es como sigue:

Datos con DVC:

- DVC proporciona un sistema de control de versiones para datos, permitiendo rastrear y versionar conjuntos de datos
- Cada archivo de datos, como ``abalone.data`` o ``abaloneNuevo.data``, es gestionado por DVC. Sin embargo, en el repositorio de Git solo se incluyen archivos ``.dvc``: ``abalone.data.dvc``, ``abaloneNuevo.data.dvc`` etc., pero no a los archivos con los datos: ``abalone.data``, ``abaloneNuevo.data``, etc.
- Los archivos ``.dvc`` actúan como metadatos que apuntan a la ubicación o versión específica de los datos. Estos archivos son ligeros y contienen información sobre cómo acceder a la versión de los datos correspondiente

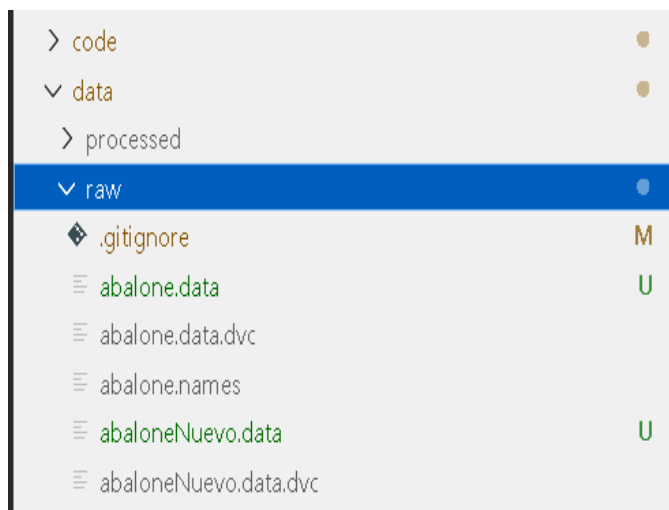
Código con Git:

- Git gestiona las versiones de los scripts de código, lo que incluye todo el código relacionado con el preprocesamiento de datos, entrenamiento de modelos, evaluación, etc.
- En el repositorio Git, solo se incluyen archivos ``.dvc`` que hacen referencia a los datos versionados en DVC. Los datos reales no se almacenan en Git

- Esta separación mantiene el repositorio Git más liviano y enfocado en el código, evitando la carga de datos grandes en el historial de Git

#### Relación entre datos y código:

- Cuando el código necesita utilizar nuevos datos o se modifican los datos existentes, se actualizan los archivos '.dvc'
- Estas actualizaciones en los archivos '.dvc', que apuntan a versiones específicas de los datos, se registran junto con los cambios en el código en un 'commit' de Git
- Esta relación explícita entre los datos y el código permite una trazabilidad clara de qué versión del código está asociada con qué versión de los datos, lo que facilita la reproducción de experimentos y la colaboración en equipo



Cuando se desea ejecutar el modelo se realiza de la manera siguiente:

PROBLEMS OUTPUT TERMINAL JUPYTER GITLENS

```
(venvProyWizeline) PS C:\Users\52477\Documents\ProyectoWizeLine> python code\main.py data\raw\abalone.data.dvc .\
```

lo que lanzará la ejecución del modelo para una versión de datos 'abalone.data.dvc' y el modelo buscará en el repositorio de dvc donde está el archivo de datos correspondiente

PROBLEMS OUTPUT TERMINAL JUPYTER GITLENS

```
0 Sex 4176 non-null object
1 Length 4176 non-null float64
2 Diameter 4176 non-null float64
3 Height 4176 non-null float64
4 Whole_weight 4176 non-null float64
5 Shucked_weight 4176 non-null float64
6 Viscera_weight 4176 non-null float64
7 Shell_weight 4176 non-null float64
8 Rings 4176 non-null int32
dtypes: float64(7), int32(1), object(1)
memory usage: 277.4+ KB
```

```
data_Fr.shape: (3492, 11)
test_size: 0.15
Inicia entrenamiento ...
Inician pruebas ...
```

Test mse

```
-----
Linear Regression 5.629771
Elastic Net 8.137405
Random Forest 5.179389
```

```
Inicia experimento 1 ...
```

PROBLEMS OUTPUT TERMINAL JUPYTER GITLENS

```
random_state=42)
Corrida 3 model RandomForestRegressor
```

	Test mse	Val mse
RandomForestRegressor	5.179389	5.058296

Inicia validación del modelo ...

experiment_id	run_id	model	me_val
264903437442838465	bd61ab3df9994567827c897e7f15b661	RandomForestRegressor	5.058295964125561

	Shell_weight	Diameter	Rings_Predicted	Rings
0	0.0700	0.260	8.0	9.0
1	0.2040	0.385	11.0	14.0
2	0.2740	0.425	12.0	11.0
3	0.1130	0.315	8.0	10.0
4	0.1250	0.350	9.0	13.0
..	...	...	...	...
441	0.3145	0.470	11.0	12.0
442	0.0600	0.255	8.0	7.0
443	0.2650	0.470	10.0	10.0
444	0.1620	0.375	9.0	8.0
445	0.3250	0.505	11.0	11.0

[446 rows x 4 columns]

F I N A L I Z A C I O N

El modelo será entrenado y validado con el dataset de datos '.data' asociado al archivo '.data.dvc' que se especificó en los parámetros de ejecución. Además dicha ejecución almacenará con ayuda de Dataflow artefactos del modelo, hiper parámetros, métricas, etc. del mejor modelo que pudo configurar



### **B3.-¿Cómo se medirá el drifting?**

El 'drifting' en 'Machine Learning' puede definirse cómo 'desplazando' o 'desviando'.

Existen al menos dos tipos de 'drifting':

- 'Data drift', se refiere a cambios en la distribución de las características (inputs) que se usan para entrenar el modelo. Esto puede suceder debido a cambios en el entorno o en el proceso de recolección de datos. Por ejemplo, si un modelo se entrena con datos de una región y luego se usa en otra región con condiciones diferentes, puede experimentar data drift. Lo anterior puede llevar a la degradación del rendimiento del modelo por no ser capaz de generalizar correctamente con los nuevos datos
- 'Model drift', se refiere a cambios en la relación entre las características y la variable objetivo. Esto significa que la estructura subyacente que el modelo está tratando de aprender ha cambiado. Por ejemplo, en un modelo los métodos del tópico que se analiza pueden evolucionar con el tiempo, lo que cambia la relación entre las características observadas y la etiqueta predecida

Para mitigar el 'drifting' existen varias técnicas como 'El monitoreo continuo', 'Re entrenamiento regular', 'Validación cruzada por ventanas de tiempo', 'Actualización de características y modelo adaptativo', 'Registro de experimentos y versionado de modelos', etc...

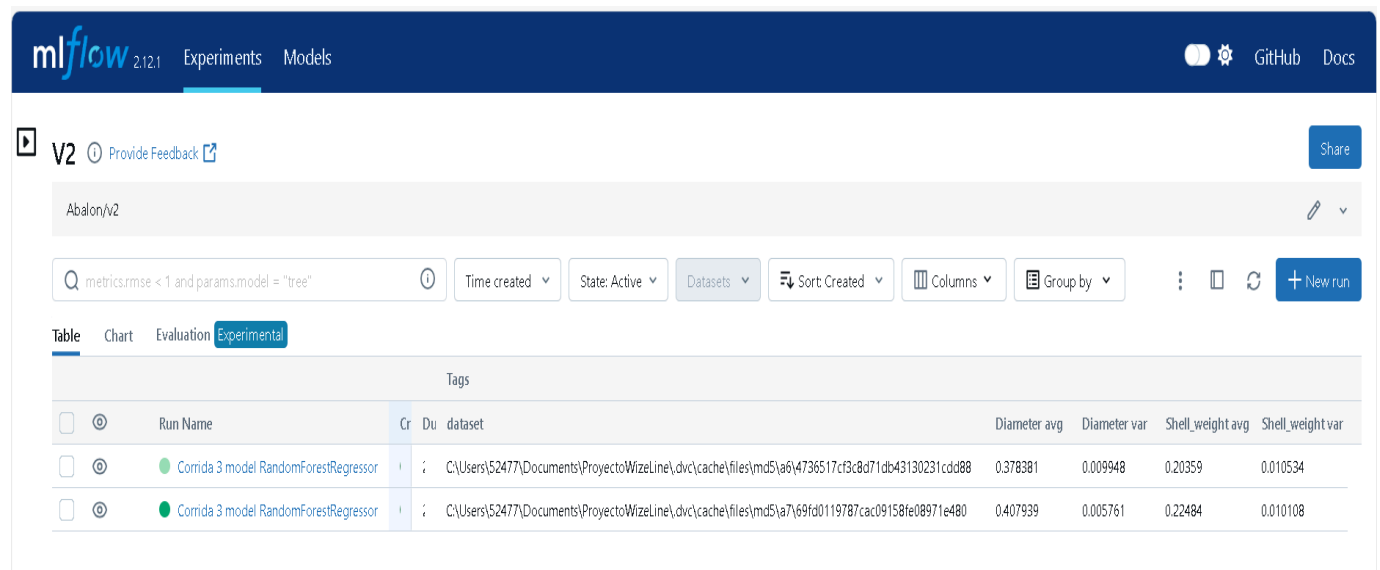
---

En el presente proyecto, con ayuda de la herramienta 'MLFlow' se realizó la mitigación del drifting, utilizando la técnica 'Registro de experimentos y versionado de modelos', para posibilitar el monitoreo tanto de los datos como del rendimiento del modelo. Cada vez que se ejecuta el modelo para encontrar al mejor modelo disponible que generalice con los nuevos datos se guarda junto con el experimento información de los dos tipos:

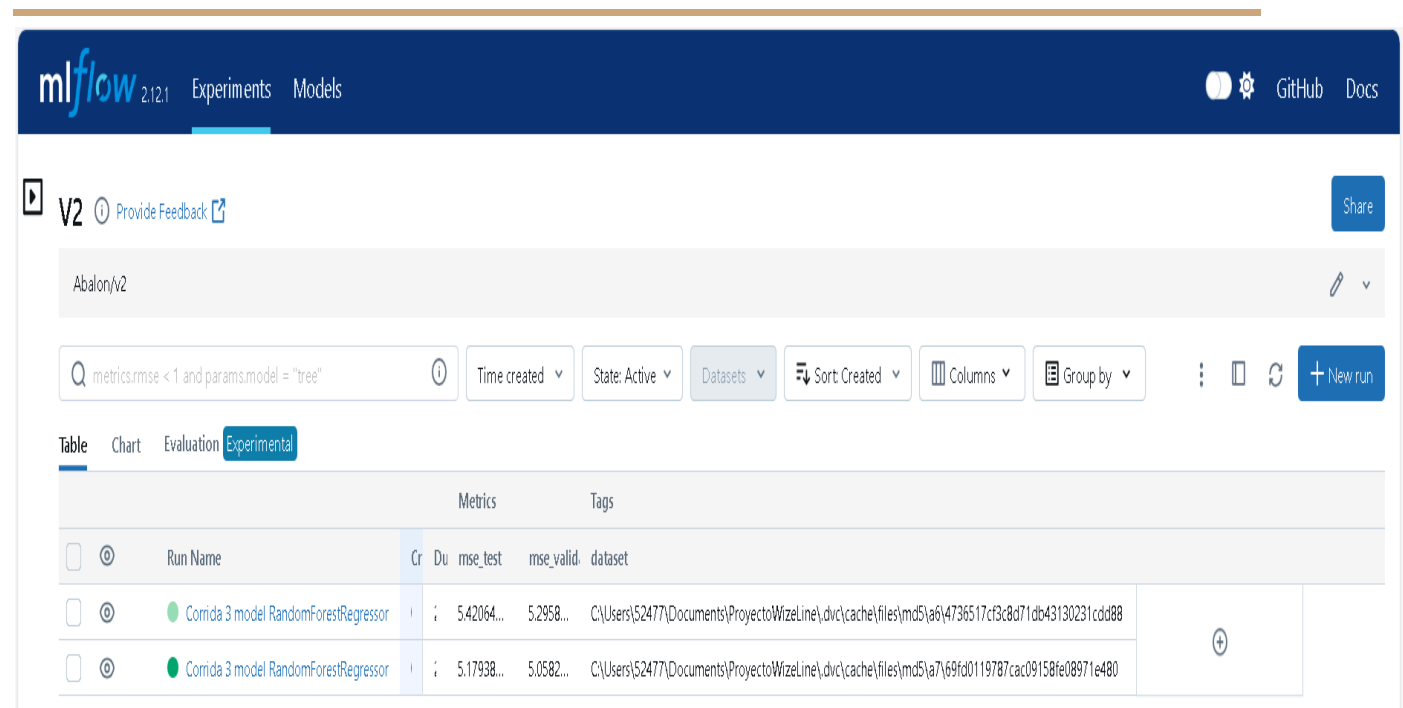
1. estadísticas de los datos de prueba, en éste caso la media y la varianza para validar qué tanto han cambiado los datos ('data drift')
2. las métricas mse tanto para los datos de prueba como de validación, para observar si el modelo mantiene su rendimiento ('model drift')

En la siguiente imagen se mostrarán las estadísticas 'media' y 'varianza' de cada una de las dos características de entrada seleccionadas para la regresión: 'shell\_weight' y 'diameter' correspondientes a la versión 2 del experimento, así como el archivo 'data.dvc' que apunta a los últimos datos

---



Por otro lado también se puede observar la métrica 'mse' que se almacena tanto para el entrenamiento como para la validación y es sencillo observar o monitorear los valores que toma en la imagen siguiente:



The screenshot shows the mlflow Experiments interface. At the top, there's a navigation bar with 'mlflow 2.12.1', 'Experiments', and 'Models'. Below this, the experiment name 'AbaloneV2' is displayed. A search bar contains the query 'metrics.rmse < 1 and params.model = "tree"'. Below the search bar, there are filters for 'Time created', 'State: Active', 'Datasets', 'Sort: Created', 'Columns', and 'Group by'. The 'Table' tab is selected, showing a table of runs. The table has columns for 'Run Name', 'Created', 'Duration', 'mse\_test', 'mse\_valid', and 'dataset'. Two runs are listed, both using the 'RandomForestRegressor' model.

Run Name		Cr	Du	mse_test	mse_valid	dataset
<input type="checkbox"/>	Corrida 3 model RandomForestRegressor	5.42064...	5.2958...	C:\Users\52477\Documents\ProyectoWizeline\div\cache\files\md5\64736517cf3c8d71db43130231cdd88		
<input type="checkbox"/>	Corrida 3 model RandomForestRegressor	5.17938...	5.0582...	C:\Users\52477\Documents\ProyectoWizeline\div\cache\files\md5\769fd0119787cac09158fe08971e480		

#### B4.-¿Se considera la prueba en el desarrollo del pipeline?

En el contexto de machine learning, existen varias clases de pruebas automáticas que se pueden realizar para asegurar la calidad y el rendimiento del modelo, para detectar sesgos

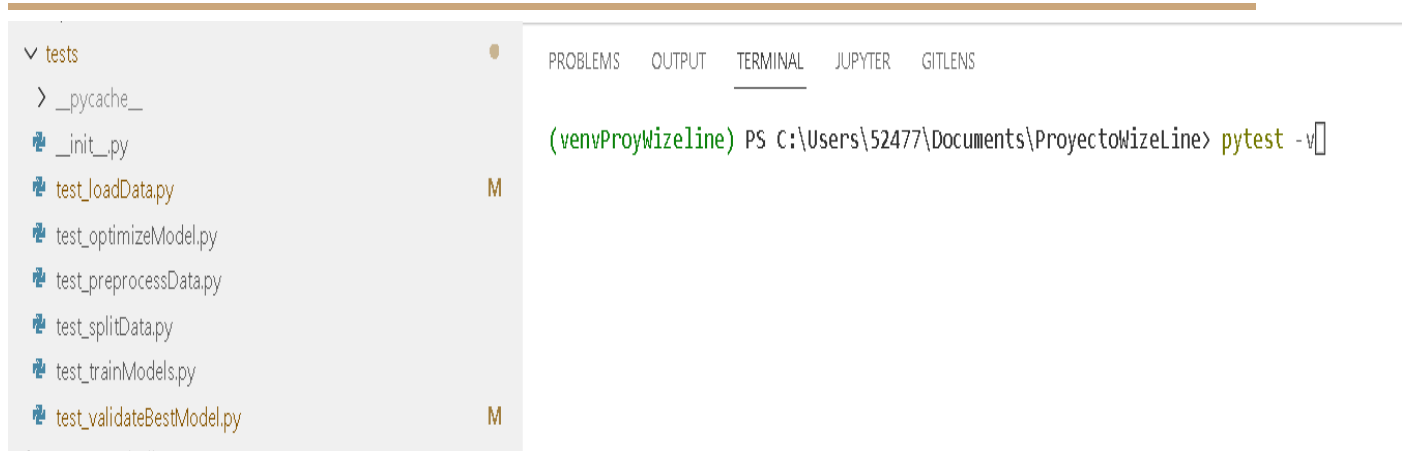
y monitorear la evolución del modelo ante nuevos datos, etc. Entre los tipos de pruebas que pueden realizarse están las pruebas unitarias, las pruebas de integración, las pruebas de validación del rendimiento del modelo, etc...

También existen muchas herramientas que ayudan al desarrollo de dichas pruebas automatizadas como unittest, pytest, nose2, hypothesis, testify, etc...

Esencialmente por simplicidad y facilidad de desarrollo, en el presente modelo solo se desarrollaron 6 scripts con un total de 12 pruebas unitarias con la librería unittest a funciones de diferentes módulos de python desarrollados. Se enlistan a continuación:



Es posible ejecutar en la terminal todas las pruebas con el comando `pytest -v`



```
(venvProyWizeline) PS C:\Users\52477\Documents\ProyectoWizeline> pytest -v
```

De ésta forma y de manera automática se llevan a cabo 12 pruebas que serán verificadas y se mostrará el resultado de éstas, como se ve a continuación:



```
ms\Python\Python310\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\52477\Documents\ProyectoWizeline
configfile: pytest.ini
collected 12 items

tests/test_loadData.py::TestDataLoading::test_column_conversion PASSED [ 8%]
tests/test_loadData.py::TestDataLoading::test_column_names PASSED [ 16%]
tests/test_loadData.py::TestDataLoading::test_data_loading PASSED [ 25%]
tests/test_loadData.py::TestDataLoading::test_data_types PASSED [ 33%]
tests/test_optimizeModel.py::TestYourFunctions::test_gen_gridSearchHiperParam PASSED [ 41%]
tests/test_preprocessData.py::TestPreprocessData::test_preprocess_data_imputation PASSED [ 50%]
tests/test_preprocessData.py::TestPreprocessData::test_preprocess_data_output PASSED [ 58%]
tests/test_splitData.py::TestSplitDataset::test_split_dataset_proportions PASSED [ 66%]
tests/test_splitData.py::TestSplitDataset::test_split_dataset_sizes PASSED [ 75%]
tests/test_trainModels.py::TestInitialTrainModels::test_initialTrainModels PASSED [ 83%]
tests/test_validateBestModel.py::TestValidation::test_output_format PASSED [ 91%]
tests/test_validateBestModel.py::TestValidation::test_output_type PASSED [100%]
```

## C) GRÁFICAS DEL EDA

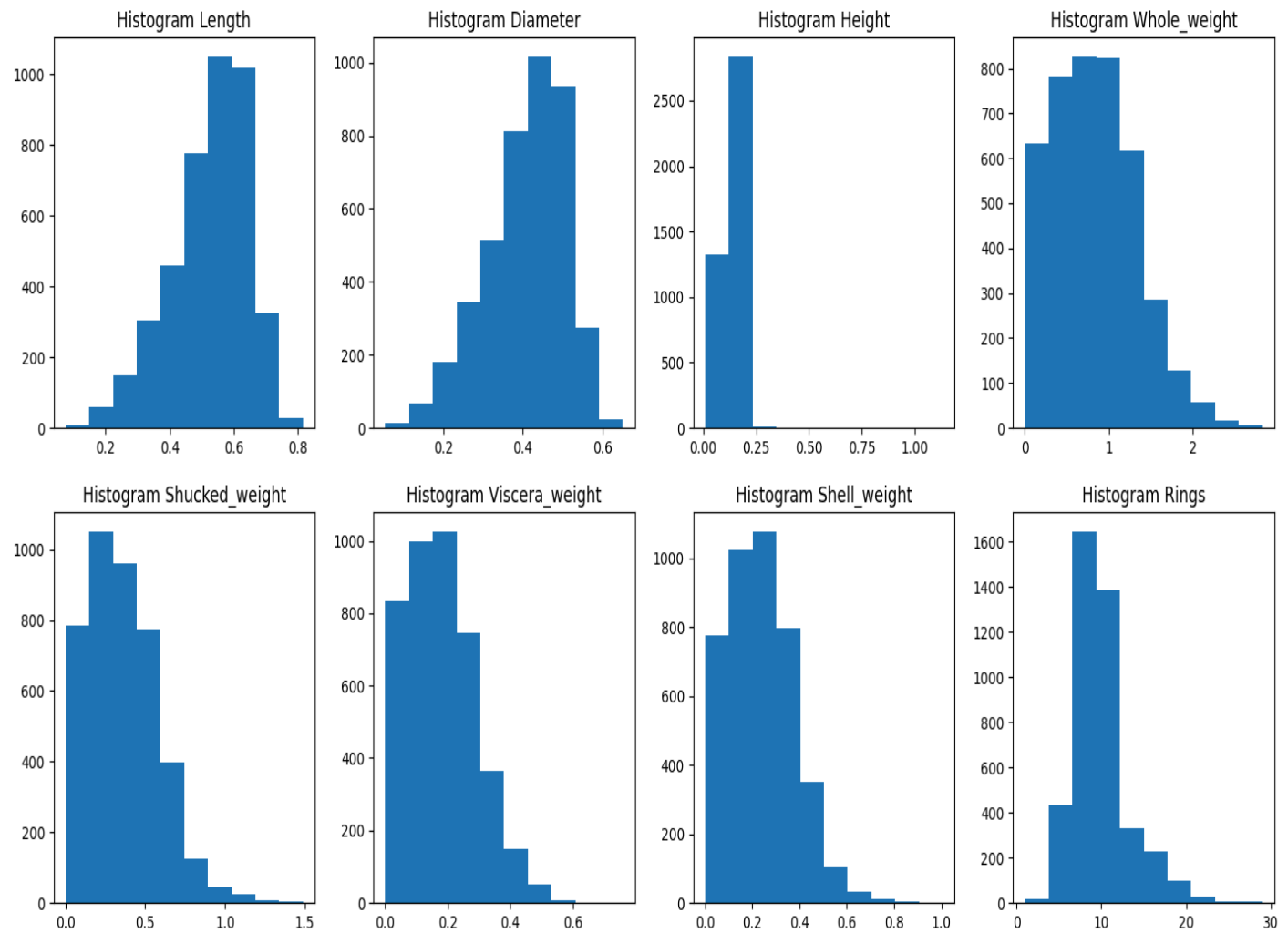


Fig 1: Histograma por variable atributo y variable objetivo

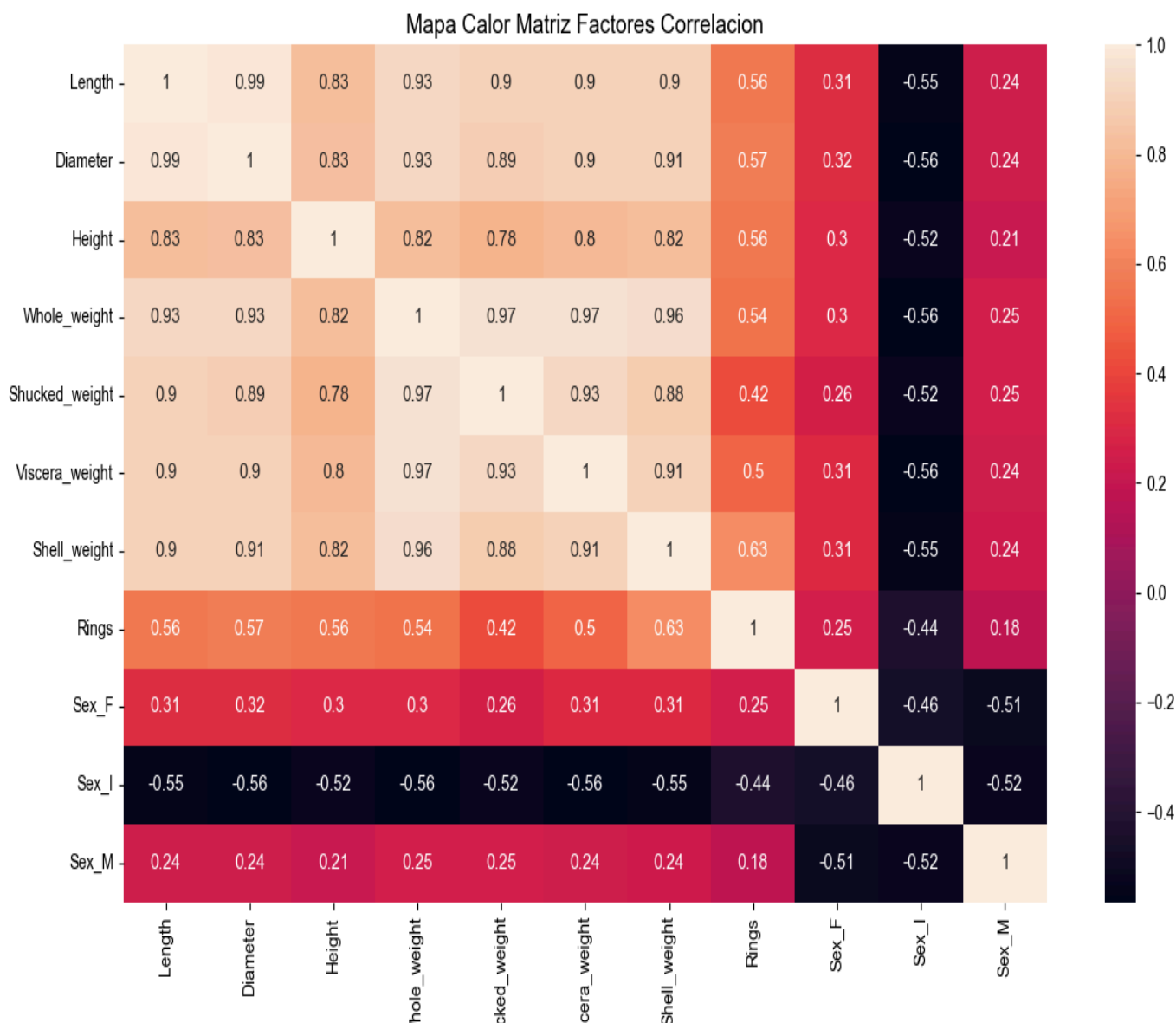


Fig 2: Mapa Calor Matriz Factores Correlación



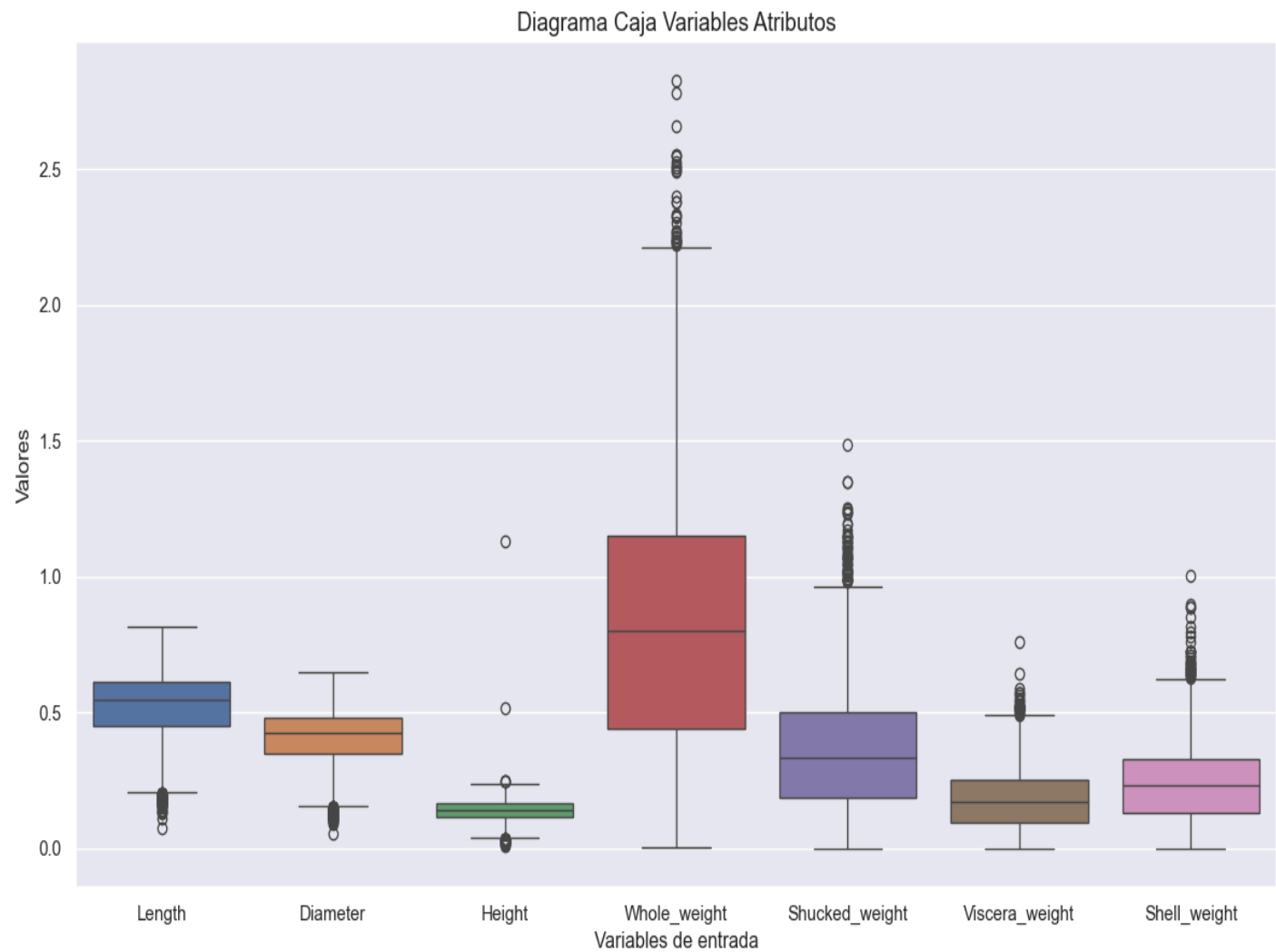


Fig 3: Diagramas Caja Variables Tipo Atributo

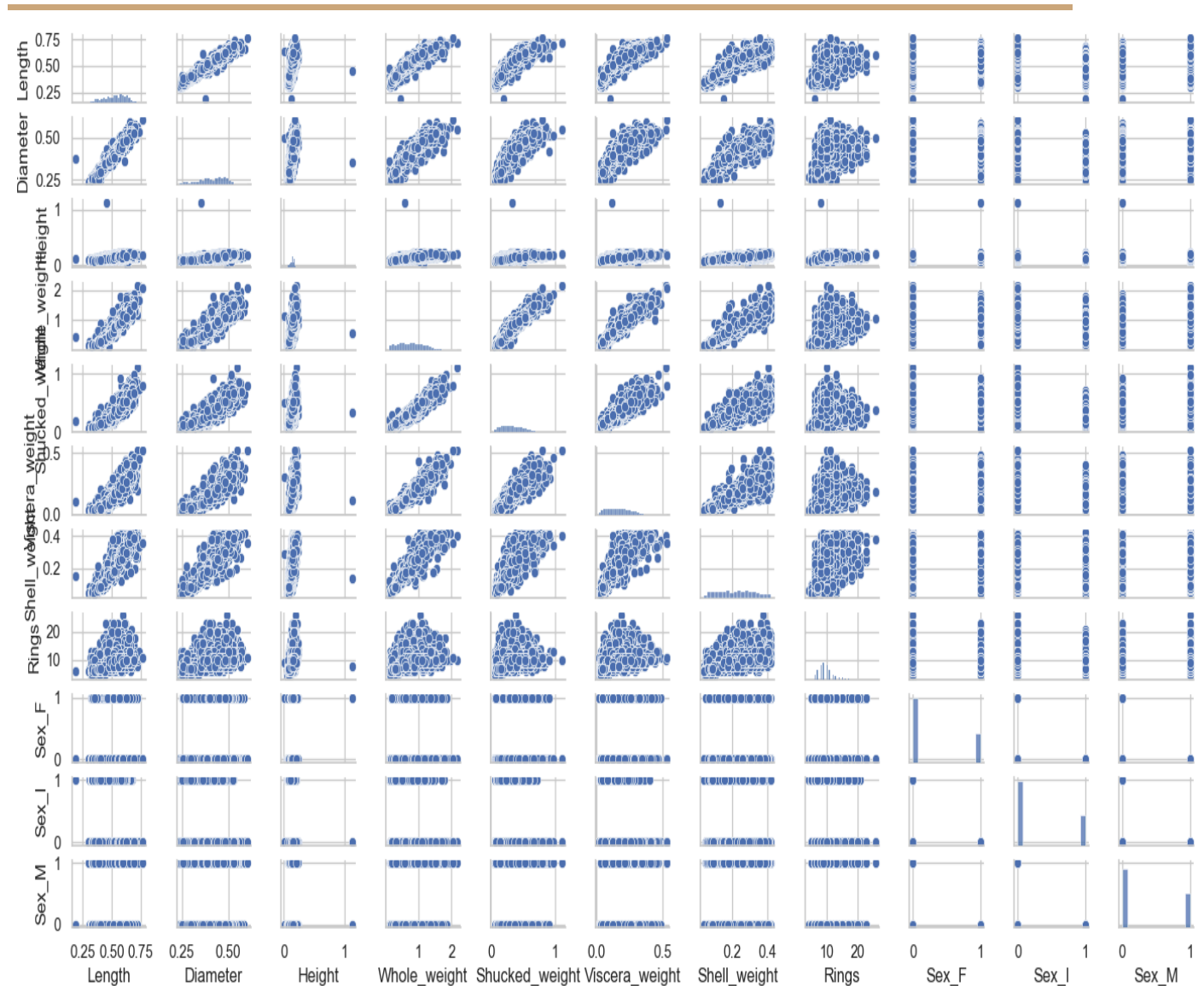


Fig 4: Correlaciones Pares Variables Tipo Atributo y Variable Objetivo

## D.-IMPLEMENTACIONES DE CÓDIGO

### Script code/main.py

```
# #####  
  
# Proyecto de MLOps para Bootcamp con Wizeline  
  
# Autor: Edgar Gonzalez Ambriz  
  
# Fecha : Mayo 2024  
  
# #####  
  
# Para correr modelo, ejecutar en consola:  
  
# python code\main.py data\raw\abalone.data.dvc .\  
  
# Para ejecutar pruebas  
  
# Correr en consola: pytest -v  
  
# Para revisar experimentos, comando en consola: mlflow ui  
  
# Correr en browser de internet: http://127.0.0.1:5000  
  
# Para ejecutar predicciones, comando en consola: uvicorn  
code.predict_ageAbalon:app --reload
```

---

```
# Correr en browser de internet: http://localhost:8000/docs
```

```
# Importar librerías
```

```
import numpy as np
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
import mlflow
```

```
import json
```

```
import sys
```

```
import dvc.api
```

```
import os
```

```
from utils.visualization import histplots_figure, matrixCorrCoef_figure,  
boxplot_figure, dispersión_figure
```

```
from utils.mseMetric import find_minMseMetric, print_mse_tests,  
metrics_report
```

```
from data.preprocessData import remove_outliers, preprocess_data
```

```
from data.loadData import load_data
```

---

---

```
from data.splitData import split_dataset

from models.experiments import do_experiment

from models.validateBestModel import
validateBestModel_withValuationDataset

from models.optimizeModel import gen_gridSearchHiperParam

from models.adminWithMLFlow import get_metadataOfBestModel

from models.trainModels import initialTrainModels

from models.trialModels import testModels_gettingMSE

from models.trialModels import testModels_gettingStatistics
```

```
#
=====

def predictAbalonAge (data_file) :

#
=====

#
-----
-----
```

---

---

```
# P R E P R O C E S O D A T O S

#

-----

# "Rings" es una variable categórica discreta que representa el número
de anillos

# Es posible su predicción mediante clasificación pero también
mediante regresión

# si se considera como una variable numérica continua

# Obtención de datos y generación de la información que contiene

data_Mtrx, names_Arr = load_data(data_file)

# Conversiones e imputaciones de datos por datos faltantes

data_Mtrx, cols_names_Arr, cols_ImputerNames_Arr, data_Fr =
preprocess_data (data_Mtrx)

# Gráficas de exploración de datos

# Desplegar histogramas de los principales atributos

# Se notará que prácticamente todos son aproximaciones de
distribuciones normales con excepción de 'Height' que está muy cargado 2
dos rangos
```

---

---

```
histplots_figure (data_Fr, cols_ImputerNames_Arr, nrows=2, ncols=4)

# Se deducirá que las variables de entrada mas correlacionadas con la
variable de salida son:

# 'Shell_weight', 'Diameter', 'Length' y 'Height', de las que se
elegirán para las predicciones las 2 primeras por cumplir con los
criterios

matrixCorrCoef_figure (data_Fr, cols_names_Arr)

# Graficar diagramas de caja para variables de entrada de manera que
se visualicen los valores atípicos

# De las dos características seleccionadas anteriormente, se notará
que :

# 'Shell_weight' tiene valores atípicos en la parte alta de la gráfica

# 'Diameter' tiene valores atípicos en la parte baja de la gráfica

boxplot_figure (data_Fr,
['Length', 'Diameter', 'Height', 'Whole_weight', 'Shucked_weight', 'Viscera_weight', 'Shell_weight'])

# Remover valores atípicos de las variables de atributos
'Shell_weight' y 'Diameter'
```

---

---

```
data_Fr = remove_outliers(data_Fr, column='Shell_weight',
lower_percentile=0.0, upper_percentile=0.90)

data_Fr = remove_outliers(data_Fr, column='Diameter',
lower_percentile=0.075, upper_percentile=1)

print ('data_Fr.shape: ', data_Fr.shape)


# Diagramas de dispersión para pares de variables

# Se confirma que las características 'Shell_weight' y 'Diameter'
están muy relacionadas con las demás variables de entrada

dispersión_figure (data_Fr, cols_names_Arr)


# -----
#   S   E   P   A   R   A   C   I   O   N           D   A   T   O   S
# -----
```

---



```
# -----  
  
#           E           N           T           R           E           N           A           M           I           E           N           T           O  
  
# -----  
  
test_size = 0.15  
  
print ('test_size: ', test_size)  
  
X, y, X_train, y_train, X_test, y_test, X_val, y_val = split_dataset  
(test_size, data_Fr [['Shell_weight', 'Diameter']], data_Fr ['Rings'])  
  
linear_model, rf_model, en_model = initialTrainModels (X_train,  
y_train)  
  
  
  
  
  
  
  
# -----  
  
#           P           R           U           E           B           A           S  
  
# -----  
  
# Calcular las desviaciones entre los datos reales del conjunto de  
prueba contra los datos predichos por los modelos  
  
mse_linear_test, mse_en_test, mse_rf_test = testModels_gettingMSE  
(linear_model, en_model, rf_model, X_test, y_test)
```

---

```
# Se elige el modelo con la mejor métrica

models = ['LinearRegression', 'ElasticNet', 'RandomForestRegressor']

test_min_mse, min_idx = find_minMseMetric (mse_linear_test,
mse_en_test, mse_rf_test)

test_best_model = models [min_idx]

print_mse_tests (mse_linear_test, mse_en_test, mse_rf_test)

# Calcular media y varianza de las variables de entrada (atributos)

stats_X_test = testModels_gettingStatistics (X_test)

print ('Shell_weight avg: ',
float("{:.6f}".format(stats_X_test['mean_X_test'][0])), 'Shell_weight var:
', float("{:.6f}".format(stats_X_test['var_X_test'][0])))

print ('Diameter avg: ',
float("{:.6f}".format(stats_X_test['mean_X_test'][1])), 'Diameter var: ',
float("{:.6f}".format(stats_X_test['var_X_test'][1])))

print
("-----
-----")
```

---

---

```
#
-----
-----

#                               G   E   N   E   R   A   R               E   X   P   E   R   I
M   E   N   T   O   S

#
-----
-----

# Generar rejilla de hiper parámetros que se utilizarán en cada
experimento con el modelo seleccionado en las pruebas

param_grid = gen_gridSearchHiperParam (test_best_model)

# Para guardar los rendimientos de cada corrida de cada experimentos

performanceRunningLst = []

print ('Inicia experimento 1 ...')

experiment_name = "Abalon/V1"                                # Iniciar experimento
versión 1

mlflow.set_experiment(experiment_name)

test_size = 0.10

if 'LinearRegression' in test_best_model :

    y_pred_val, mse_val1 = do_experiment (test_best_model,
test_min_mse, stats_X_test, param_grid, test_size, performanceRunningLst,
X_train, y_train, X_val, y_val, data_file)
```

---

---

```
elif 'ElasticNet' in test_best_model or 'RandomForestRegressor' in
test_best_model :

    y_pred_val, mse_val1 = do_experiment (test_best_model,
test_min_mse, stats_X_test, param_grid[0], test_size,
performanceRunningLst, X_train, y_train, X_val, y_val, data_file)


print ('Inicia experimento 2 ...')

experiment_name = "Abalon/V2"                # Iniciar experimento
versión 2

mlflow.set_experiment(experiment_name)

test_size = 0.20

if test_best_model == 'LinearRegression' :

    y_pred_val, mse_val2 = do_experiment (test_best_model,
test_min_mse, stats_X_test, param_grid, test_size, performanceRunningLst,
X_train, y_train, X_val, y_val, data_file)

elif test_best_model == 'ElasticNet' or test_best_model ==
'RandomForestRegressor' :

    y_pred_val, mse_val2 = do_experiment (test_best_model,
test_min_mse, stats_X_test, param_grid[1], test_size,
performanceRunningLst, X_train, y_train, X_val, y_val, data_file)
```

---

---

```
#  
-----  
-----  
  
#   V   A   L   I   D   A   C   I   O   N           M   O   D   E   L   O  
  
#  
-----  
-----  
  
# Predecir número de anillos de los abulones con el conjunto de datos  
de validación con el modelo con el mejor rendimiento  
  
print ('Inicia validación del modelo ...')  
  
# Se obtienen algunos metadatos del modelo con el mejor rendimiento  
info_entry = get_metadataOfBestModel (performanceRunningLst)  
  
flg, predictions_df = validateBestModel_withValuationDataset (X_val,  
y_pred_val, y_val, info_entry['model'])  
  
# Imprimir el DataFrame predictions_df  
print(predictions_df)  
  
# Se escribe el reporte de las métricas  
  
metrics_report (mse_linear_test, mse_en_test, mse_rf_test, mse_val1,  
mse_val2, test_best_model)
```

---

---

```
# Suponiendo que 'info_entry' es tu diccionario y 'flg' es tu string
data = { "info_entry": info_entry, "flg": flg }

# Guardar el diccionario en un archivo JSON
with open('predict_vars.json', 'w') as file:
    json.dump(data, file, indent=4) # Especificar indent=4
    para una mejor legibilidad del archivo JSON

#
=====
=====

#                               ENTRADA PRINCIPAL AL CODIGO

#
=====
=====

if __name__ == "__main__":
    # Verificar que se proporcionen los argumentos esperados
    if len(sys.argv) != 3:
        print("Uso: python main.py <data_file_dvc> <path_repo_dvc>")
```

---

---

```
sys.exit(1)
```

```
# Obtener los nombres de archivo y el repositorio DVC de los  
argumentos de la línea de comandos
```

```
data_file_dvc = sys.argv[1]
```

```
path_repo_dvc = sys.argv[2]
```

```
print ('*****')
```

```
print ('data_file_dvc: ', data_file_dvc)
```

```
print ('path_repo_dvc: ', path_repo_dvc)
```

```
print ('*****')
```

```
# quitar extensión '.dvc'
```

```
data_file_name = data_file_dvc[:-4]
```

```
print ('*****')
```

```
print ('data_file_name: ', data_file_name)
```

```
print ('*****')
```

```
fullPath_data_file_name = dvc.api.get_url(path_repo_dvc +  
data_file_name)
```

```
print ('*****')
```

---

---

```
print ('fullPath_data_file_name: ', fullPath_data_file_name)

print ('*****')

predictAbalonAge(fullPath_data_file_name)

print ('F      I      N      A      L      I      Z      A      C      I      O      N')
```

### Script code/predict\_ageAbalon.py

```
# Para predicción, correr en consola: uvicorn code.predict_ageAbalon:app
--reload

# Correr en browser de internet:  http://localhost:8000/docs
```

```
import json

import pandas as pd
```

---



---

```
from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from pydantic.types import confloat

from code.models.predictAbalonRings import
predict_abalonRings_withUserData


# Leer el archivo JSON

with open('predict_vars.json', 'r') as file:

    data = json.load(file)


# Recuperar el diccionario y el string

info_entry = data["info_entry"]

flg = data["flg"]


# Preparacion de la aplicación que llamará a FastAPI

app = FastAPI()                                     # Se crea instancia


# Clase con las variables de entrada (atributos)

class AbalonInput(BaseModel):
```

---

---

```
    Shell_weight: confloat (ge=0.001, le=1)           # Solo recibirá
valores en rango [0.001, 1]

    Diameter: confloat (ge=0.001, le=1.5)           # Solo recibirá
valores en rango [0.001, 1.5]


# Clase con la variable de salida (objetivo) y una segunda variable
derivada de la primera

class AbalonOutput(BaseModel):

    predicted_rings: int

    predicted_age: float


# -----

@app.post("/predict_abalon")

async def predict_abalon (input_data: AbalonInput) :

# -----

    Shell_weight = input_data.Shell_weight

    Diameter = input_data.Diameter


    X_new = [[Shell_weight, Diameter]]           # Inicializar la
matriz de datos nuevos de entrada


    # Llamar a la función predict_abalonRings_withUserData
```

---

---

```
y_pred_new_rounded = predict_abalonRings_withUserData(X_new,
info_entry, flg)

abalon_age = y_pred_new_rounded + 1.5           # Se suma el factor
1.5 para obtener la edad del abalón

# Devolver los resultados

return AbalonOutput(predicted_rings=y_pred_new_rounded,
predicted_age=abalon_age)

# -----

@app.get("/")

async def read_root () :

# -----

    return {"message": "Bienvenido a la API de predicción de edad de
abulón."}
```

### Script code/data/loadData.py

```
# Importar librerías

import numpy as np
```

---

---

```
import pandas as pd

import os


def load_data (data_file) :

    '''

    Obtención de los datos desde un dataset en un archivo separado por
    comas

    Impresion de la información y descripcion de los datos que contiene el
    dataset

    '''

    # Cargar los datos y los nombres de las columnas del dataset a
    analizar y sobre el que se realizarán predicciones

    data_Mtrx = pd.read_csv(data_file)

    names_Arr = ['Sex', 'Length', 'Diameter', 'Height', 'Whole_weight',
'Shucked_weight', 'Viscera_weight', 'Shell_weight', 'Rings']

    # Asignar nombres de columnas

    data_Mtrx.columns = names_Arr

    # Convertir la columna 'Rings' a tipo int

    data_Mtrx['Rings'] = data_Mtrx['Rings'].astype(int)
```

---

---

```
# Informacion y descripcion de los datos

print ('Información y descripción de las características del dataset')

print
('-----')

data_Mtrx.info()

print ("\n")

data_Mtrx.describe()

print ("\n")

return data_Mtrx, names_Arr
```

**Script code/data/preprocessData.py**

---

---

```
# Importar librerías

import numpy as np

import pandas as pd

from sklearn.impute import SimpleImputer


#
=====

def preprocess_data (data_Mtrx) :

#
=====

'''

    Conversiones e imputaciones de datos

'''

# El atributo "Sex" que es categorico se convierte a 3 columnas de
tipo "int"

data_Mtrx = pd.get_dummies(data_Mtrx, columns=["Sex"], dtype=int)
```

---

---

```
cols_names_Arr =
['Length', 'Diameter', 'Height', 'Whole_weight', 'Shucked_weight', 'Viscera_weight', 'Shell_weight', 'Rings', 'Sex_F', 'Sex_I', 'Sex_M']

cols_ImputerNames_Arr =
['Length', 'Diameter', 'Height', 'Whole_weight', 'Shucked_weight', 'Viscera_weight', 'Shell_weight', 'Rings']

# Se crea una instancia de un imputer para aplicar la media a los
valores "nan" en la matriz con los datos y después guardarlos en un data
frame

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

data_Fr =
pd.DataFrame(imputer.fit_transform(data_Mtrx[cols_names_Arr]))

data_Fr.columns = cols_names_Arr

# Se aplica la media a los valores 0 en el data frame para todos los
atributos excepto el de "Sex"

column_means = data_Fr[cols_ImputerNames_Arr].mean()

for col in cols_ImputerNames_Arr:

    for index, row in data_Fr.iterrows():

        if row[col] == 0:

            data_Fr.at[index, col] = column_means[col]

return data_Mtrx, cols_names_Arr, cols_ImputerNames_Arr, data_Fr
```

---

```
#
=====

def remove_outliers(df, column, lower_percentile, upper_percentile):

#
=====

    """

    Esta función elimina los valores atípicos de un DataFrame en la
    columna especificada,

    utilizando percentiles como límites inferior y superior. Los
    percentiles son medidas estadísticas

    utilizadas para dividir un conjunto de datos ordenados en cien partes
    iguales.

    En otras palabras, un percentil indica el valor por debajo del cual
    cae un cierto porcentaje

    de los datos en un conjunto de datos ordenado

    """

    # Calcula los percentiles definidos por "lower_percentile" y
    "upper_percentile" para la columna especificada

    # y devuelve los valores correspondientes para esos percentiles

    lower_value = df[column].quantile(lower_percentile)

    upper_value = df[column].quantile(upper_percentile)
```



---

```
# Filtra el DataFrame para mantener solo las filas donde los valores  
estén dentro de los límites
```

```
filtered_df = df[(df[column] >= lower_value) & (df[column] <=  
upper_value)]
```

```
# Devuelve el DataFrame filtrado
```

```
return filtered_df
```

---

## Script code/data/splitData.py

```
# Importar librerías

from sklearn.model_selection import train_test_split


#
=====

def split_dataset (test_size, data_inputAtributes, data_outputTarget) :

#
=====

    '''

    Separación del dataset en datos para entrenamiento, prueba y
    validacion

    '''

    # Separar las dos características de entrada de la variable objetivo
    de salida 'Rings'

    X = data_inputAtributes

    y = data_outputTarget
```

```
# Se repartirán los datos para entrenamiento, pruebas y validacion

X_remain, X_test, y_remain, y_test = train_test_split (X, y,
test_size=test_size, random_state=42)

X_train, X_val, y_train, y_val = train_test_split (X_remain, y_remain,
test_size=test_size, random_state=42)

return X, y, X_train, y_train, X_test, y_test, X_val, y_val
```

---

### Script code/models/adminWithMLFlow.py

```
import os

import pickle

import mlflow.sklearn

import numpy as np

import datetime


from sklearn.metrics import mean_squared_error

from code.utils.mseMetric import print_mseMetrics


#
=====
=====
=====

def do_runningsModels (validation_best_model, test_best_model,
test_min_mse, stats_X_test, X_train, X_val, y_train, y_val, test_size,
performanceRunningsLst, data_file) :

#
=====
=====
=====
```

---

```
'''  
  
    Se realizan las corridas del modelo seleccionado como el mejor de las  
    pruebas  
  
    '''  
  
print ('validation_best_model: ', validation_best_model)  
  
if (test_best_model == 'LinearRegression') :  
    i = 0  
  
elif (test_best_model == 'ElasticNet') :  
    i = 1  
  
elif (test_best_model == 'RandomForestRegressor') :  
    i = 2  
  
# Realizar corrida del modelo del experimento  
  
print("Corrida " + str(i + 1) + " model " +  
validation_best_model.__class__.__name__)  
  
with mlflow.start_run(run_name="Corrida " + str(i + 1) + " model " +  
validation_best_model.__class__.__name__):
```

---

```
# guarda en MLFlow

mlflow.log_param("test_size", test_size)

mlflow.log_param("modelo",
validation_best_model.__class__.__name__)

if test_best_model == 'LinearRegression':

    mlflow.log_param("fit_intercept",
validation_best_model.fit_intercept)

    mlflow.log_param("n_jobs", validation_best_model.n_jobs)

    mlflow.log_param("positive", validation_best_model.positive)

elif test_best_model == 'ElasticNet':

    mlflow.log_param("alpha", validation_best_model.alpha)

    mlflow.log_param("l1_ratio", validation_best_model.l1_ratio)

    mlflow.log_param("fit_intercept",
validation_best_model.fit_intercept)

    mlflow.log_param("precompute",
validation_best_model.precompute)

    mlflow.log_param("max_iter", validation_best_model.max_iter)

    mlflow.log_param("tolerance", validation_best_model.tol)

    mlflow.log_param("random_state",
validation_best_model.random_state)

elif test_best_model == 'RandomForestRegressor':
```

---

---

```
        mlflow.log_param("n_estimators",
validation_best_model.n_estimators)

        mlflow.log_param("max_depth", validation_best_model.max_depth)

        mlflow.log_param("max_features",
validation_best_model.max_features)
```

```
# Predicciones con Regresión Lineal en conjunto de datos de
validación
```

```
y_pred_val = np.round(validation_best_model.predict(X_val))
```

```
# Métrica error cuadrático medio (mse)
```

```
mlflow.log_metric("mse_test", test_min_mse)
```

```
mse_val = mean_squared_error(y_val, y_pred_val)
```

```
mlflow.log_metric("mse_validation", mse_val)
```

```
# se guarda el modelo
```

```
mlflow.sklearn.log_model(validation_best_model,
f"modelo_regresion_{i + 1}")
```

```
# Obtener el experiment_id y run_id dentro del contexto del run
actual
```

```
experiment_id = mlflow.active_run().info.experiment_id
```

```
run_id = mlflow.active_run().info.run_id
```

---

---

```
mlflow.set_tag ("dataset", data_file)

mlflow.set_tag ('Shell_weight avg',
float("{:.6f}".format(stats_X_test['mean_X_test'][0])))

mlflow.set_tag ('Shell_weight var',
float("{:.6f}".format(stats_X_test['var_X_test'][0])))

mlflow.set_tag ('Diameter avg',
float("{:.6f}".format(stats_X_test['mean_X_test'][1])))

mlflow.set_tag ('Diameter var',
float("{:.6f}".format(stats_X_test['var_X_test'][1])))

# Se agrega la corrida a la lista de experimentos

performanceRunningsLst.append ({'experiment_id': experiment_id,
'run_id': run_id, 'model': test_best_model, 'mse_val': mse_val} )

# Finaliza la corrida

mlflow.end_run()

print_mseMetrics (test_best_model, test_min_mse, mse_val)

return y_pred_val, mse_val
```

---



---

```
#
-----
--

def load_bestModelFromMLFlow (info_entry, flg) :

#
-----
--

    '''

    Como los modelos están registrado en MLFlow, entonces para hacer
    predicciones se carga desde ahí el pickle del modelo óptimo

    '''

    # Configurar la conexión a MLflow

    experiment_id = info_entry ['experiment_id']

    run_id = info_entry ['run_id']

    # Obtener el cliente de MLflow

    client = mlflow.tracking.MlflowClient()

    # Obtener la información del run

    run_info = client.get_run(run_id)

    # Ruta al directorio de artefactos y nombre del archivo .pickle
    buscado
```

---

---

```
artifacts_dir =
"C:\\Users\\52477\\Documents\\ProyectoWizeLine\\mlruns\\" + experiment_id
+ "\\" + run_id + "\\artifacts\\modelo_regresion_" + flg + "\\"

pickle_file_name = "model.pkl"

# Verificar si el directorio de artefactos existe

if os.path.exists(artifacts_dir):

    files_in_artifacts = os.listdir(artifacts_dir)                #
    Obtener la lista de archivos en el directorio de artefactos

    if pickle_file_name in files_in_artifacts:                    #
        Buscar el archivo .pickle por su nombre

        model_path = artifacts_dir + pickle_file_name            #
        Construir la ruta completa al archivo .pickle

        try:

            with open(model_path, "rb") as f:

                loaded_model = pickle.load(f)                      #
                Cargar el modelo desde el archivo .pickle

        except Exception as e:

            print ("Ocurrió un error al cargar el archivo pickle: ",
e)

    else:

        print ("El archivo pickle no fue encontrado en el directorio
de artefactos")
```

---

---

```
else:

    print ("El directorio de artefactos no existe")

return loaded_model
```

```
# =====

def get_metadataOfBestModel (performanceRunningLst) :

# =====

    '''

    Encontrar la entrada de las corridas de los experimentos con el mejor
    rendimiento

    '''

    min_mse = 10000000                                # Establecer un
valor inicial muy grande

    min_entry = None                                    # Inicializar la
entrada mínima como None
```

---

---

```
for entry in performanceRunningLst :

    mse_actual = entry.get('mse_val', 10000000)    # Obtener el valor
de 'mse_val', si no está presente, usar un valor muy grande

    if mse_actual < min_mse:

        min_mse = mse_actual

        min_entry = entry

    print ("experiment_id          run_id
model          mse_val")

    print
    ("-----")

    print (min_entry['experiment_id'], ' ', min_entry['run_id'], ' ',
min_entry['model'], ' ', min_entry['mse_val'])

    print ("")

return min_entry
```

---

### Script code/models/experiments.py

```
from code.models.optimizeModel import get_optimizedValidationModel

from code.models.adminWithMLFlow import do_runningsModels


#
=====
=====
=====

def do_experiment (test_best_model, test_min_mse, stats_X_test,
param_grid, test_size, performanceRunningsLst, X_train, y_train, X_val,
y_val, data_file) :
```

---

---

```
#  
=====
```

'''  
Se realiza un experimento con el mejor modelo de la fase de pruebas,  
ahora con los datos de validación  
Se optimiza con una rejilla de hiper parametros  
'''

# V A L I D A C I O N

# El mejor modelo en las pruebas pero con diferentes hiper parametros

```
validation_best_model = get_optimizedValidationModel (param_grid,  
test_best_model, X_train, y_train)
```

# Corridas del experimento

```
y_pred_val, mse_val = do_runningsModels (validation_best_model,  
test_best_model, test_min_mse, stats_X_test, X_train, X_val, y_train,  
y_val, test_size, performanceRunningsLst, data_file)
```

return y\_pred\_val, mse\_val

---

### Script code/models/optimizeModel.py

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.linear_model import LinearRegression, ElasticNet
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.model_selection import GridSearchCV
```

---

---

```
#
=====

def gen_gridSearchHiperParam (best_model) :

#
=====

'''

    Generar las rejillas de búsqueda de los mejores hiper parametros  de
    cada algoritmo de aprendizaje

'''

if (best_model == 'LinearRegression') :

    param_grid = {

        'fit_intercept': [True, False],

        'n_jobs': [2, 4, 8],

        'positive': [True]

    }

elif (best_model == 'ElasticNet') :

    param_grid = [

        {

            'alpha': [0.1, 0.5, 0.9],
```

---



---

```
        'l1_ratio': [0.33, 0.66, 1],
        'fit_intercept': [True, False],
        'precompute': [False],
        'max_iter' : [500, 750, 1000],
        'tol' : [0.001],
        'random_state' : [42]
    },
    {
        'alpha': [0.00001, 0.0001, 0.001],
        'l1_ratio': [0.001, 0.01, 0.1],
        'fit_intercept': [True, False],
        'precompute': [False],
        'max_iter' : [1500, 2500, 5000],
        'tol' : [0.0001],
        'random_state' : [42]
    }
]

elif (best_model == 'RandomForestRegressor') :
    param_grid = [
        {
            'n_estimators': [100, 500, 1000],
            'max_depth': [5, 10, 15],
```

---

---

```
        'max_features' : [2],

# Máximo dos características

        'random_state' : [42]

    },

    {

        'n_estimators': [1500, 2500, 3500],

        'max_depth': [2, 3, 5],

        'max_features' : [2],

# Máximo dos características

        'random_state' : [42]

    }

]

return param_grid

#
=====
=====

def get_optimizedValidationModel (param_grid, test_best_model, X_train,
y_train) :

#
=====
=====
```

---

---

```
'''

    EL mejor modelo del testing, se optimizao utilizando rejillas de
    busqueda de mejores hiper parametros

'''

# Generar el grid search y usar 10 subconjuntos para realizar
validaciones cruzadas

if ('LinearRegression' in test_best_model) :

    grid_search = GridSearchCV(LinearRegression(), param_grid, cv=10)

elif ('ElasticNet' in test_best_model) :

    grid_search = GridSearchCV(ElasticNet(), param_grid, cv=10)

elif ('RandomForestRegressor' in test_best_model) :

    grid_search = GridSearchCV(RandomForestRegressor(), param_grid,
cv=10)

# Ajustar el mejor modelo obtenido con la rejilla de busqueda de hiper
parametros con los datos de entrenamiento

grid_search.fit(X_train, y_train)

# Obtener los modelos óptimos de acuerdo a los hiper parámetros de la
busqueda por rejilla

validation_best_model = grid_search.best_estimator_

'''
```

---

```
# Imprimir los hiper parámetros de cada modelo

print ('')

print ('Mejores hiper parámetros por modelo')

print ('-----')

print ('validation_best_model: ', validation_best_model)

print ("\n")

return validation_best_model
```

### Script code/models/optimizeModel.py

```
import pandas as pd

import numpy as np

from sklearn.linear_model import LinearRegression, ElasticNet

from sklearn.ensemble import RandomForestRegressor
```

---

```
from sklearn.model_selection import GridSearchCV

#
=====

def gen_gridSearchHiperParam (best_model) :

#
=====

'''
    Generar las rejillas de búsqueda de los mejores hiper parametros  de
    cada algoritmo de aprendizaje
'''

if (best_model == 'LinearRegression') :

    param_grid = {

        'fit_intercept': [True, False],

        'n_jobs': [2, 4, 8],

        'positive': [True]

    }

elif (best_model == 'ElasticNet') :
```

---

---

```
param_grid = [  
    {  
        'alpha': [0.1, 0.5, 0.9],  
        'l1_ratio': [0.33, 0.66, 1],  
        'fit_intercept': [True, False],  
        'precompute': [False],  
        'max_iter' : [500, 750, 1000],  
        'tol' : [0.001],  
        'random_state' : [42]  
    },  
    {  
        'alpha': [0.00001, 0.0001, 0.001],  
        'l1_ratio': [0.001, 0.01, 0.1],  
        'fit_intercept': [True, False],  
        'precompute': [False],  
        'max_iter' : [1500, 2500, 5000],  
        'tol' : [0.0001],  
        'random_state' : [42]  
    }  
]  
  
elif (best_model == 'RandomForestRegressor') :  
    param_grid =
```

---

---

```
{  
    'n_estimators': [100, 500, 1000],  
    'max_depth': [5, 10, 15],  
    'max_features' : [2],  
# Máximo dos características  
    'random_state' : [42]  
},  
{  
    'n_estimators': [1500, 2500, 3500],  
    'max_depth': [2, 3, 5],  
    'max_features' : [2],  
# Máximo dos características  
    'random_state' : [42]  
}  
]  
  
return param_grid  
  
#  
=====
```

---

```
def get_optimizedValidationModel (param_grid, test_best_model, X_train,  
y_train) :
```

---

---

```
#  
=====
```

'''  
EL mejor modelo del testing, se optimizao utilizando rejillas de  
busqueda de mejores hiper parametros  
'''

# Generar el grid search y usar 10 subconjuntos para realizar  
validaciones cruzadas

```
if ('LinearRegression' in test_best_model) :  
    grid_search = GridSearchCV(LinearRegression(), param_grid, cv=10)  
  
elif ('ElasticNet' in test_best_model) :  
    grid_search = GridSearchCV(ElasticNet(), param_grid, cv=10)  
  
elif ('RandomForestRegressor' in test_best_model) :  
    grid_search = GridSearchCV(RandomForestRegressor(), param_grid,  
cv=10)  
  
# Ajustar el mejor modelo obtenido con la rejilla de busqueda de hiper  
parametros con los datos de entrenamiento  
grid_search.fit(X_train, y_train)
```

---



---

```
# Obtener los modelos óptimos de acuerdo a los hiper parámetros de la
busqueda por rejilla
```

```
validation_best_model = grid_search.best_estimator_
```

```
# Imprimir los hiper parámetros de cada modelo
```

```
print ('')
```

```
print ('Mejores hiper parámetros por modelo')
```

```
print ('-----')
```

```
print ('validation_best_model: ', validation_best_model)
```

```
print ("\n")
```

```
return validation_best_model
```

## Script code/models/predictAbalonRings.py

```
import pandas as pd
```

---

---

```
import numpy as np

from code.models.adminWithMLFlow import load_bestModelFromMLFlow


#
=====

def predict_abalonRings_withUserData (X_new, info_entry, flg) :

#
=====

    '''

    Se realizará la predicción de la edad de un abulón dados el peso de su
    caparazón y su diámetro

    '''

    # Cargar en memoria el mejor modelo guardado en MLFlow

    loaded_model = load_bestModelFromMLFlow (info_entry, flg)

    # Realizar predicción final

    y_pred_new = loaded_model.predict(X_new)                # Se obtiene el
número de anillos

    y_pred_new_rounded = round (y_pred_new [0])            # Se redondea el
número de anillos

    return y_pred_new_rounded
```

---

## Script code/models/trainModels.py

```
# Importar librerías
```

---

---

```
from sklearn.linear_model import LinearRegression, ElasticNet

from sklearn.ensemble import RandomForestRegressor


# -----

def initialTrainModels (X_train, y_train) :

# -----

    '''

    Se entrenan los 3 modelos de regresión con los hiper parametros mas
    default posible

    '''

    print ('Inicia entrenamiento ...')

    # Se entrenan 3 modelos con algoritmos diferentes para ajustarlos a
    los datos de entrenamiento

    linear_model = LinearRegression()

    linear_model.fit(X_train, y_train)

    rf_model = RandomForestRegressor(n_estimators=100, max_features=2,
max_depth=5, random_state=42)

    rf_model.fit(X_train, y_train)
```

---

---

```
    en_model = ElasticNet(alpha=0.1, l1_ratio=0.1, max_iter=1000,  
tol=0.001, random_state=42)  
  
    en_model.fit(X_train, y_train)  
  
    return linear_model, rf_model, en_model
```

### Script code/models/trialModels.py

```
# Importar librerías
```

---

---

```
import numpy as np

from sklearn.metrics import mean_squared_error


#
-----
-----

def testModels_gettingStatistics (X_test) :

#
-----
-----

    """

    Calcula la media y la varianza de los atributos en X_test.

    Parámetros:

    - X_test: array-like, shape (n_samples, 2). Conjunto de datos de
prueba con las variables atributos

    Retorna:

    - stats: dict

        Diccionario con las estadísticas de media y varianza de X_test

    """

    # Calcula la media y la varianza de X_test

    mean_X_test = np.mean (X_test, axis=0)
```

---

---

```
var_X_test = np.var (X_test, axis=0)

# Almacena las estadísticas en un diccionario
stats_X_test = {'mean_X_test': mean_X_test, 'var_X_test': var_X_test}

return stats_X_test

#
-----
-----

def testModels_gettingMSE (linear_model, en_model, rf_model, X_test,
y_test) :

#
-----
-----

'''
Predicciones con 3 modelos diferentes y el mismo dataset de pruebas
Regresa la métrica MSE obtenida con cada modelo
'''

print ('Inician pruebas ...')

y_pred_linear_test = np.round(linear_model.predict(X_test)) #
Predicciones con Regresión Lineal en conjunto de datos de prueba
```

---

---

```
mse_linear_test = mean_squared_error(y_test, y_pred_linear_test)      #  
Métrica error cuadrático medio para regresión lineal  
  
y_pred_en_test = np.round(en_model.predict(X_test))                    #  
Predicciones con Elastic net en conjunto de datos de prueba  
  
mse_en_test = mean_squared_error(y_test, y_pred_en_test)              #  
Métrica error cuadrático medio para regresion elastic net  
  
y_pred_rf_test = np.round(rf_model.predict(X_test))                    #  
Predicciones con Random Forest en conjunto de datos de prueba  
  
mse_rf_test = mean_squared_error(y_test, y_pred_rf_test)              #  
Métrica error cuadrático medio para Random forest  
  
return mse_linear_test, mse_en_test, mse_rf_test
```



---

```
import pandas as pd

import numpy as np

#
=====

def validateBestModel_withValuationDataset(X_val, y_pred_evaluation,
y_val, info_entry_model) :

#
=====

'''

Se predice la edad del abulon para todo el dataset de valuacion

'''

if info_entry_model == 'LinearRegression':

    flg = "1"

elif info_entry_model == 'ElasticNet':

    flg = "2"

elif info_entry_model == 'RandomForestRegressor':

    flg = "3"

# Obtener los valores reales de la variable objetivo "rings" para los
datos de validación
```

---

---

```
y_val_real = y_val.reset_index(drop=True) # Reiniciar los índices
para que coincidan con las predicciones

# Convertir y_pred_evaluation a un DataFrame de Pandas

predictions_df = pd.DataFrame(y_pred_evaluation,
columns=['Rings_Predicted'])

# Concatenar las predicciones con X_val y y_val_real

predictions_df = pd.concat([X_val.reset_index(drop=True),
predictions_df, y_val_real], axis=1)

return flg, predictions_df
```

---



**Script code/utils/mseMetric.py**

```
# =====

def find_minMseMetric (mse_0, mse_1, mse_2) :

# =====

    """

    Función para determinar el menor de tres errores cuadráticos medios
    (MSE)

    """

    mse_list = [mse_0, mse_1, mse_2]

    min_val = min(mse_list)

    min_idx = mse_list.index(min_val)

    return min_val, min_idx


# =====

def print_mse_tests (mse_linear_test, mse_en_test, mse_rf_test) :

# =====

    """
```

---

```
Se imprime la métrica mse para cada uno de los 3 algoritmos de
regresion

'''

print ("\n")

print ('                Test mse')

print
('-----')

print('Linear Regression                ',
'{:.6f}'.format(mse_linear_test))

print('Elastic Net                        ', '{:.6f}'.format(mse_en_test))

print('Random Forest                      ', '{:.6f}'.format(mse_rf_test))

print ("\n")

#
=====
=====

def print_mseMetrics (test_best_model, test_min_mse, mse_val) :

#
=====
=====

'''

# Imprimir resultados de los errores cuadráticos medios (mse) de cada
modelo en cada conjunto de datos
```

---

```
'''

print ("\n")

print ('                                Test mse          Val mse          ')

print
('-----')

print(test_best_model + "                ", "{:.6f}".format(test_min_mse), "
", "{:.6f}".format(mse_val))

print ("\n")

#
=====

def metrics_report (mse_linear_test, mse_en_test, mse_rf_test, mse_val1,
mse_val2, test_best_model) :

#
=====

'''
```

---

Escribe en un archivo la evaluación de los modelos con las métricas obtenidas

```
'''

ruta_archivo =
"C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\evaluation_report
.txt"

with open(ruta_archivo, 'w') as f:

    f.write("Reporte de Validación de Modelos\n")

    f.write("                                Test mse\n")

f.write("-----
\n")

    f.write("Linear Regression                " + str(mse_linear_test) +
"\n")

    f.write("Elastic Net                        " + str(mse_en_test) + "\n")

    f.write("Random Forest                          " + str(mse_rf_test) + "\n")

f.write("-----
\n\n")

    if (mse_val1 <= mse_val2) :

        f.write("El modelo optimizado para generalizar es: " +
test_best_model + " con mse: " + str(mse_val1))

    else :
```

---

---

```
f.write("El modelo optimizado para generalizar es: " +  
test_best_model + " con mse: " + str(mse_val2))
```

### Script code/utils/visualization.py

```
# Importar librerías  
  
import seaborn as sns  
  
import matplotlib.pyplot as plt  
  
import numpy as np  
  
import pandas as pd
```

```
#  
=====
```

```
def histplots_figure (data_Fr, cols, nrows, ncols):
```

```
#  
=====
```

```
'''
```

---

---

Esta función dibuja varios histogramas en una sola gráfica utilizando 'subplots'

Parámetros:

```
data_Fr : dataframe con los datos a graficar

cols : nombre de las columnas del dataframe

nrows : número de renglones

ncols : número de columnas

'''

# Crear un dataframe temporal para graficar los histogramas de las
características mas importantes

data_temp_Fr = pd.DataFrame(data_Fr, columns=cols)

# Crear una figura con varios 'subplots' divididos en "nrows" renglones
y "ncols" columnas

fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(14, 7))

# Iterar sobre las columnas y renglones dibujando un histograma en
cada 'subplot'

for i in range(nrows):

    for j in range(ncols):

        column_name = cols[i * ncols + j]

        axes[i, j].hist(data_temp_Fr[column_name])

        axes[i, j].set_title(f'Histogram {column_name}')
```

---



---

```
plt.tight_layout()                # Ajustar automaticamente el espacio
entre los 'subplots'

plt.gcf().canvas.manager.set_window_title('Menu Opciones')

plt.show()                        # Desplegar

plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\histplots.png")

#
=====

def matrixCorrCoef_figure (data_Fr, cols_names_Arr) :

#
=====

'''

    Esta función gráfica un mapa de calor de los coeficientes de la matriz
    de correlaciones

    Parámetros:

        data_Fr : dataframe con los datos a graficar

        cols_names_Arr : nombre de las columnas del dataframe

'''

# Visualizar la matriz de correlación como grafico
```

---

---

```
#

# Criterios:

# Buscar variables de entrada con una correlación más fuerte con la
variable de salida

# Evitar variables de entrada con una correlación muy fuerte con otra
variables de entrada para evitar posibles redundancias


matrixCorrCoef = np.corrcoef(data_Fr.T)

plt.figure(figsize=(14, 7))

sns.heatmap(matrixCorrCoef, cbar=True, annot=True, annot_kws={"size":
11}, yticklabels=cols_names_Arr, xticklabels=cols_names_Arr)

plt.title('Mapa Calor Matriz Factores Correlacion', fontsize=14)

sns.set(font_scale=1.5)


# Dar nombre al icono de la parte superior izquierda

plt.gcf().canvas.manager.set_window_title('Menu Opciones')


#plt.show()


plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\matrizCorrCoef.png")
```

---

```
#
=====

def boxplot_figure (data_Fr, names) :

#
=====

'''
    Diagramas de caja de las variables numericas continuas de tipo
    atributo

    Parámetros:

        data_Fr : dataframe con los datos a graficar
        names : nombre de las columnas del dataframe
'''

plt.figure(figsize=(14, 7))
sns.boxplot(data=data_Fr[names])

plt.title('Diagrama Caja Variables Atributos', fontsize=14)
plt.xlabel('Variables de entrada', fontsize=12)
plt.ylabel('Valores', fontsize=12)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
```

---

```
plt.gcf().canvas.manager.set_window_title('Menu Opciones')

# plt.show()

plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\boxplot.png")


#
=====

def boxplot_figure2 (sets, labels) :

#
=====

'''

    Diagramas de caja de los conjuntos de datos reales, de entrenamiento,
    prueba y validacion

    Parámetros:

        sets : conjuntos de datos a graficar

        labels : etiquetas de los conjuntos de datos

    '''
```

---



```
#
=====

'''

Gráficas de densidad a dos subplots

Parámetros:

    y                : datos reales de la variable objetivo

    y_pred_linear_test : datos predecidos de prueba algoritmo
regresión lineal

    y_pred_rf_test    : datos predecidos de prueba algoritmo random
forest

    y_pred_poly_test   : datos predecidos de prueba algoritmo
regresión polinomial

    y_pred_linear_val  : datos predecidos de validación algoritmo
regresión lineal

    y_pred_rf_val      : datos predecidos de validación algoritmo
random forest

    y_pred_poly_val    : datos predecidos de validación algoritmo
regresión polinomial

'''

# Crear gráfica de densidad para comparar las predicciones de las
pruebas y las validaciones contra los valores reales
```

---

# Trazar la estimación de la KDE (Kernel Density Estimation) de una variable unidimensional

# Es una forma de estimar la distribución de probabilidad de una variable continua basada en una muestra de datos

```
plt.figure(figsize=(20, 6)) # Definir el tamaño de la figura
```

# Primer subgráfico: comparación de predicciones en las pruebas vs valores reales

# Se colorean las áreas bajo las curvas

```
plt.subplot(1, 2, 1) # 1 fila, 2 columnas, primer subgráfico
```

```
sns.kdeplot(y, color='blue', label='Valores Reales', fill=True)
```

```
sns.kdeplot(y_pred_linear_test, color='red', label='Predicciones Lineales', fill=True)
```

```
sns.kdeplot(y_pred_rf_test, color='green', label='Predicciones Random Forest', fill=True)
```

```
sns.kdeplot(y_pred_poly_test, color='orange', label='Predicciones Polinómicas', fill=True)
```

```
plt.title('Densidad Predicciones Prueba vs Reales')
```

```
plt.xlabel('Valor')
```

```
plt.ylabel('Densidad')
```

```
plt.legend()
```

```
plt.xlim(5, 15)
```

---

---

```
# Segundo subgráfico: comparación de predicciones en las validaciones
vs valores reales

# Se colorean las áreas bajo las curvas

plt.subplot(1, 2, 2) # 1 fila, 2 columnas, segundo subgráfico

sns.kdeplot(y, color='blue', label='Valores Reales', fill=True)

sns.kdeplot(y_pred_linear_val, color='red', label='Predicciones
Lineales', fill=True)

sns.kdeplot(y_pred_rf_val, color='green', label='Predicciones Random
Forest', fill=True)

sns.kdeplot(y_pred_poly_val, color='orange', label='Predicciones
Polinómicas', fill=True)

plt.title('Densidad Predicciones Validación vs Reales')

plt.xlabel('Valor')

plt.ylabel('Densidad')

plt.legend()

plt.xlim(5, 15)


plt.tight_layout() # Ajustar automáticamente la disposición de
los subgráficos para evitar superposiciones

plt.gcf().canvas.manager.set_window_title('Menu Opciones')

#plt.show() # Mostrar la figura con los dos subgráficos


plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\twoSubplotsDensity.png")
```

---



---

```
#
=====

def dispersión_figure (data_Fr, cols_names_Arr) :

#
=====

'''

    Diagramas de dispersión por pares de variables tanto de atributos como
    objetivo

    Parámetros:

        data_Fr : dataframe con los datos a graficar

        cols_names_Arr : nombre de las columnas del dataframe

    '''

# Criterios :

# Comparar la variable de salida vs las de entrada para tomar las de
la mas clara relacion

# Comparar entre sí variables de entrada para descartar las mas
correlacionadas pues pueden ser redundantes
```

---

---

```
sns.set(style='whitegrid')

sns.pairplot(data_Fr[cols_names_Arr], height=1)

plt.gcf().canvas.manager.set_window_title('Menu Opciones')

#plt.show()

plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\dispersión.png")

#
=====

def twoSubplotsScatters_figure (predictions_df, Shell_weight, Rings_Real,
Rings_Predicted, Rings, Diameter) :

#
=====

'''

Gráficas de scatter a dos subplots

Parámetros:

    predictions_df :

    Shell_weight    :
```

---

---

```
Rings_Real      :

Rings_Predicted :

Rings           :

Diameter        :

'''

# Crear la figura y los ejes

fig, axs = plt.subplots(1, 2, figsize=(15, 6))

# Graficar la comparación de "Rings_Real" y "Rings_Predicted" en
función de Shell_weight

axs[0].scatter(predictions_df[Shell_weight],
predictions_df[Rings_Real], color='blue', label=Rings_Real)

axs[0].scatter(predictions_df[Shell_weight],
predictions_df[Rings_Predicted], color='orange', label=Rings_Predicted)

axs[0].set_title('Comparación Variable Objetivo Real vs Predicción')

axs[0].set_xlabel(Shell_weight)

axs[0].set_ylabel(Rings)

axs[0].legend()

# Graficar la comparación de "Rings_Real" y "Rings_Predicted" en
función de Diameter

axs[1].scatter(predictions_df[Diameter], predictions_df[Rings_Real],
color='blue', label=Rings_Real)
```

---

---

```
    axs[1].scatter(predictions_df[Diameter],
predictions_df[Rings_Predicted], color='orange', label=Rings_Predicted)

    axs[1].set_title('Comparación Variable Objetivo Real vs Predicción')

    axs[1].set_xlabel(Diameter)

    axs[1].set_ylabel(Rings)

    axs[1].legend()


    plt.tight_layout()                                # Ajustar automáticamente
la disposición de los subgráficos para evitar superposiciones

    plt.gcf().canvas.manager.set_window_title('Menu Opciones')

    # plt.show()                                    # Mostrar la gráfica


plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\twoSubplotsScatters.png")
```

```
#
=====

def validation_figures (y, y_pred_linear_test, y_pred_poly_test,
y_pred_rf_test, y_pred_linear_val, y_pred_poly_val, y_pred_rf_val) :

#
=====
```

---

```
'''
```

Gráficas de validación

```
'''
```

```
# Crear gráfica de diagrama de caja
```

```
# Para comparar datos reales con las predicciones prueba y validación  
de los modelos
```

```
# Se notará que todas las medias de las predicciones de las pruebas y  
validaciones de los 3 modelos modelos
```

```
# están un poco más altas que la de los datos reales
```

```
boxplot_figure2 ( [y, y_pred_linear_test, y_pred_poly_test,  
y_pred_rf_test, y_pred_linear_val, y_pred_poly_val, y_pred_rf_val],  
                  ['Reales', 'Pruebas Lineal', 'Pruebas Polinomial',  
'Pruebas Random Forest', 'Validación Lineal', 'Validación Polinomial',  
'Validación Random Forest'] )
```

```
# Se notará que la densidad de las predicciones polinomiales son las  
mas parecidas a los valores reales
```

```
# pues las predicciones lineal y "random forest" tienen picos mas  
altos o mas de 1 pico
```

```
twoSubplotsDensity_figure ( y, y_pred_linear_test, y_pred_rf_test,  
y_pred_poly_test,
```

---

```
y_pred_linear_val, y_pred_rf_val,  
y_pred_poly_val    )  
  
plt.savefig("C:\\Users\\52477\\Documents\\ProyectoWizeLine\\reports\\figures\\validation.png")
```

### Script tests/test\_loadData.py

```
import unittest  
  
import pandas as pd  
  
import sys  
  
import os  
  
from code.data.loadData import load_data  
  
class TestDataLoading(unittest.TestCase):
```

---

---

```
# Prueba para verificar si se cargan los datos correctamente

# -----

def test_data_loading(self):

# -----

    '''

    La prueba asegura de que la función load_data() cargue
    correctamente los datos y que devuelva un DataFrame de Pandas con las
    columnas correctamente etiquetadas

    '''

    # Obtener la ruta relativa al archivo 'abalone.data'

    file_name = os.path.join("data", "raw", "abalone.data")

    print ('file_name', file_name)

    data, column_names = load_data(file_name)

    self.assertIsInstance(data, pd.DataFrame)

    self.assertEqual(len(data), len(pd.read_csv(file_name))) #
    Asumiendo que los datos de prueba tienen la misma longitud


    # Prueba para verificar si los nombres de las columnas se asignan
    correctamente
```

---

---

```
# -----  
  
def test_column_names(self):  
  
# -----  
  
'''  
  
    La prueba verifica si los nombres de las columnas obtenidos al  
    cargar los datos coinciden exactamente con los nombres de columnas  
    esperados  
  
    '''  
  
    if len(sys.argv) > 1 and not sys.argv[1].startswith("-"):  
        file_name = sys.argv[1]  
    else:  
        file_name = ".\\data\\raw\\abalone.data"  
  
    print ('file_name', file_name)  
  
    data, column_names = load_data(file_name)  
  
    self.assertEqual(list(data.columns), column_names)  
  
# Prueba para verificar si la columna 'Rings' se convierte a tipo int  
correctamente
```

---



---

```
# -----  
  
def test_column_conversion(self):  
  
# -----  
  
    '''  
  
    Prueba verifica que la columna variable de salida (objetivo) tiene  
el tipo correcto  
  
    '''  
  
    if len(sys.argv) > 1 and not sys.argv[1].startswith("-"):  
        file_name = sys.argv[1]  
    else:  
        file_name = ".\\data\\raw\\abalone.data"  
  
    print ('file_name', file_name)  
    data, _ = load_data(file_name)  
    self.assertEqual(data['Rings'].dtype, int)  
  
# -----  
  
def test_data_types(self):  
  
# -----  
  
    '''
```

---

---

La prueba asegura que las columnas en el DataFrame tengan los tipos de datos esperados según lo definido en el diccionario 'expected\_types'

Es decir se verifica si los datos se cargaron correctamente y si tienen los tipos de datos correctos

```
'''

if len(sys.argv) > 1 and not sys.argv[1].startswith("-"):

    file_name = sys.argv[1]

else:

    file_name = ".\\data\\raw\\abalone.data"


print ('file_name', file_name)

data, _ = load_data(file_name)

expected_types = {'Sex': object, 'Length': float, 'Diameter':
float, 'Height': float,

                  'Whole_weight': float, 'Shucked_weight': float,
'Viscera_weight': float,

                  'Shell_weight': float, 'Rings': int}


for column, expected_type in expected_types.items():

    self.assertEqual(data[column].dtype, expected_type)
```

---

```
if __name__ == '__main__':  
    unittest.main()
```

### Script tests/test\_optimizeModel.py

```
import unittest  
  
from code.models.optimizeModel import gen_gridSearchHiperParam  
  
# -----  
  
class TestYourFunctions (unittest.TestCase) :  
  
# -----
```

---

```
'''

Prueba unitaria de la función que genera las rejillas de búsqueda de
hiper parámetros

'''

# -----

def test_gen_gridSearchHiperParam (self) :

# -----

    # Test for LinearRegression. Verifica si se genera correctamente
    el parámetro de la cuadrícula para LinearRegression

    param_grid_lr = gen_gridSearchHiperParam('LinearRegression')

    self.assertIsNotNone(param_grid_lr)                # Asegura que
    el parámetro de la cuadrícula no sea None

    self.assertIsInstance(param_grid_lr, dict)          # Asegura que
    el parámetro de la cuadrícula sea un diccionario

    # Test for ElasticNet. Verifica si se genera correctamente el
    parámetro de la cuadrícula para ElasticNet

    param_grid_en = gen_gridSearchHiperParam('ElasticNet')

    self.assertIsNotNone(param_grid_en)                # Asegura que
    el parámetro de la cuadrícula no sea None

    self.assertIsInstance(param_grid_en, list)         # Asegura que
    el parámetro de la cuadrícula sea una lista
```

---

```
# Test for RandomForestRegressor. Verifica si se genera
correctamente el parámetro de la cuadrícula para RandomForestRegressor

param_grid_rf = gen_gridSearchHiperParam('RandomForestRegressor')

self.assertIsNotNone(param_grid_rf)                # Asegura que
el parámetro de la cuadrícula no sea None

self.assertIsInstance(param_grid_rf, list)         # Asegura que
el parámetro de la cuadrícula sea una lista


if __name__ == '__main__':

    unittest.main()
```

## Script tests/test\_preprocessData.py

```
import unittest

import pandas as pd

from ProyectoWizeLine.code.data.preprocessData import preprocess_data

# -----

class TestPreprocessData (unittest.TestCase) :
```

---

---

```
# -----

# -----

def test_preprocess_data_output (self) :

# -----

'''

    La prueba unitaria verifica si la función preprocess_data produce
    un DataFrame correctamente formateado

    con las columnas adecuadas después de realizar las conversiones e
    imputaciones de datos

'''

# Crear un DataFrame de ejemplo

data_Mtrx = pd.DataFrame({

    'Sex': ['F', 'M', 'I'],

    'Length': [1.0, 2.0, 3.0],

    'Diameter' : [0.20, 0.40, 0.60],

    'Height' : [0.010, 0.020, 0.030],

    'Whole_weight' : [0.10, 0.20, 0.30],

    'Shucked_weight' : [0.01, 0.02, 0.03],

    'Viscera_weight' : [0.001, 0.002, 0.003],

    'Shell_weight' : [0.099, 0.188, 0.277],

    'Rings' : [7, 8, 9]
```

---

---

```
    })

    # Llamar a la función preprocess_data

    processed_data, _, _, _ = preprocess_data(data_Mtrx)

    # Verificar si el DataFrame devuelto tiene las columnas esperadas

    expected_columns = ['Length', 'Diameter', 'Height',
                        'Whole_weight', 'Shucked_weight', 'Viscera_weight', 'Shell_weight',
                        'Rings', 'Sex_F', 'Sex_I', 'Sex_M']

    self.assertEqual(processed_data.columns.tolist(),
                     expected_columns)

# -----

def test_preprocess_data_imputation (self) :

# -----

    '''

    La prueba unitaria es para verificar si no hay valores faltantes
    en el DataFrame data_Fr

    '''
```

---

---

```
# Crear un DataFrame de ejemplo con valores faltantes

data_Mtrx = pd.DataFrame({

    'Sex': ['F', 'M', 'I'],

    'Length': [1.0, None, 3.0],

    'Diameter' : [None, 0.40, 0.60],

    'Height' : [0.010, 0.020, None],

    'Whole_weight' : [0.10, None, 0.30],

    'Shucked_weight' : [0.01, None, 0.03],

    'Viscera_weight' : [0.001, None, 0.003],

    'Shell_weight' : [None, 0.188, 0.277],

    'Rings' : [7, 8, None]

})


# Llamar a la función preprocess_data

_, _, _, processed_data = preprocess_data(data_Mtrx)


# Extraer el DataFrame procesado (data_Fr)

data_Fr = processed_data


# Verificar si no hay valores faltantes en el DataFrame procesado
(data_Fr)

self.assertFalse(data_Fr.isnull().any().any())
```

---



```
if __name__ == '__main__':  
    unittest.main()
```

### Script tests/test\_splitData.py

```
import unittest  
  
import numpy as np  
  
from ProyectoWizeLine.code.data.splitData import split_dataset
```

---

```
# -----  
  
class TestSplitDataset (unittest.TestCase) :  
  
# -----  
  
# -----  
  
def setUp (self) :  
  
# -----  
  
    '''  
  
        Prepara el entorno de prueba antes de ejecutar cada prueba  
individual  
  
        En este caso específico, el método setUp se utiliza para  
configurar los datos de ejemplo  
  
        que se utilizarán en múltiples pruebas dentro de la clase de  
prueba  
  
    '''  
  
    # Configurar datos de ejemplo  
  
    self.X = np.random.rand(100, 2)          # 100 muestras, 2  
características  
  
    self.y = np.random.rand(100)            # 100 muestras de variable  
objetivo  
  
    self.test_size = 0.2
```

---

```
# -----  
  
def test_split_dataset_sizes (self) :  
  
# -----  
  
    '''  
  
        Prueba unitaria que garantiza que la función split_dataset divida  
correctamente los datos en conjuntos de  
  
        entrenamiento, prueba y validación, y que los tamaños de los  
conjuntos divididos sean coherentes  
  
        con el tamaño total de los datos de entrada  
  
    '''  
  
    # Llamar a la función split_dataset  
  
    _, _, X_train, y_train, X_test, y_test, X_val, y_val =  
split_dataset(self.test_size, self.X, self.y)  
  
    # Verificar los tamaños de los conjuntos  
  
    total_samples = len(self.X)  
  
    # Verificar que la suma del tamaño de los 3 subconjuntos de datos  
es igual al total de datos  
  
    self.assertEqual(len(X_train) + len(X_test) + len(X_val),  
total_samples)
```

---

---

```
self.assertEqual(len(y_train) + len(y_test) + len(y_val),  
total_samples)
```

```
# -----  
  
def test_split_dataset_proportions (self) :  
  
# -----  
  
'''  
  
    Esta prueba garantiza que la función split_dataset divida  
    correctamente los datos en conjuntos de  
  
    entrenamiento, prueba y validación, y que los tamaños de los  
    conjuntos de prueba y entrenamiento  
  
    estén en proporción con el tamaño total de los datos de entrada y  
    el tamaño de prueba especificado  
  
    '''  
  
    # Llamar a la función split_dataset  
  
    _, _, X_train, y_train, X_test, y_test, X_val, y_val =  
    split_dataset(self.test_size, self.X, self.y)  
  
    # Calcular tamaños esperados  
  
    total_samples = len(self.X)
```

---

---

```
        expected_test_size = int(total_samples * self.test_size)

        expected_train_size = int(total_samples * (1 - self.test_size) *
(1 - self.test_size))

        # Verificar los tamaños de los conjuntos que coincidan con los
tamaños esperados

        self.assertEqual(len(X_test), expected_test_size)

        self.assertEqual(len(X_train), expected_train_size)


if __name__ == '__main__':

    unittest.main()
```

### Script tests/test\_trainModel.py

```
# Importar librerías
```

---

---

```
import unittest

from sklearn.linear_model import LinearRegression, ElasticNet

from sklearn.ensemble import RandomForestRegressor

from code.models.trainModels import initialTrainModels


# -----

class TestInitialTrainModels (unittest.TestCase) :

# -----

    # -----

    def test_initialTrainModels (self) :

# -----

        '''

        Prueba si la función initialTrainModels es capaz de entrenar
modelos de regresión lineal, bosque aleatorio y ElasticNet correctamente

        '''

        # Configurar los datos de las características de entrada y la
variable objetivo

        X_train = [[0.330, 0.10]]          # 'Shell_weight' y
'Diameter'

        y_train = [8]                      # 'Rings'
```

---

---

```
linear_model, rf_model, en_model = initialTrainModels(X_train,
y_train)

# Probar si los modelos están entrenados

self.assertIsNotNone(linear_model)

self.assertIsNotNone(rf_model)

self.assertIsNotNone(en_model)

# Probar si los modelos son del tipo esperado

self.assertIsInstance(linear_model, LinearRegression)

self.assertIsInstance(rf_model, RandomForestRegressor)

self.assertIsInstance(en_model, ElasticNet)

if __name__ == '__main__':

    unittest.main()
```

**Script tests/test\_validateBestModel.py**

---

---

```
# Importar librerías

import unittest

import pandas as pd

from code.models.validateBestModel import
validateBestModel_withValuationDataset


# -----

class TestValidation(unittest.TestCase):

# -----

# -----

def setUp(self):

# -----

    '''

    Se preparan los datos para las pruebas

    '''

# Configura datos de prueba

    self.X_val = pd.DataFrame({'Shell_weight': [0.01, 0.02, 0.03],
'Diameter': [0.1, 0.2, 0.3]})

    self.y_pred_evaluation = [5, 10, 15]

    self.y_val = pd.Series([6, 9, 14])

    self.info_entry_model = 'LinearRegression'
```

---



```
# -----  
  
def test_output_type(self):  
  
# -----  
  
    '''  
  
    Prueba que el tipo de dato del 1er dato que regresa la función  
'validateBestModel_withValuationDataset' sea una instancia de un modelo  
  
    '''  
  
    # Verifica el tipo de dato de salida  
  
    result, _ = validateBestModel_withValuationDataset(self.X_val,  
self.y_pred_evaluation, self.y_val, self.info_entry_model)  
  
    self.assertIsInstance(result, str)  
  
  
  
# -----  
  
def test_output_format(self):  
  
# -----  
  
  
  
    '''
```



▼ ▼ ▼

```
self.assertTrue(all(col in result.columns for col in
['Shell_weight', 'Diameter', 'Rings Predicted']))
```

```
unittest.main()
```

