



# DE JUPYTER A LA TIERRA

Óscar Iglesias

Machine learning, análisis e  
ingeniería de datos, desarrollo...

Área SI

# Sobre esta presentación

Y por qué es interesante

## Ciencia de datos

- Datasets, corpus, transformaciones
- Modelos, entrenamiento, predicción
  - Parámetros, métricas
  - Visualizaciones
  - Libretas

## Desarrollo

- Infraestructura
- Servicio, despliegue
- Integración, mantenimiento
- Interfaz, orquestación
- Metodologías, patrones



# Un problema ML

## Destripando CatFoodDB

¿Qué tenemos?

Una página web increíblemente  
scrapeable con info sobre  
comida para gatos.

¿Qué queremos?

Un modelo ML entrenado,  
capaz de estimar la puntuación  
de una nueva comida.

The screenshot shows a web browser displaying the CatFoodDB website. The URL in the address bar is [catfooddb.com/product/friskies/Prime+Filets+With+Salmon+%26+Beef+In+Sauce](https://catfooddb.com/product/friskies/Prime+Filets+With+Salmon+%26+Beef+In+Sauce). The page title is "Friskies Prime Filets With Salmon & Beef In Sauce Review". On the left, there's a thumbnail image of a can of Friskies Prime Filets cat food. To the right of the image, under "Quick Analysis", are the following details: Ingredients: 5 paws (orange), Nutrition: 5 paws (orange), Potential Allergens: beef, seafood, corn, wheat gluten, soy, artificial colors and meat by-products, Est. Calories: 60 cal/100g, and Price: \$. Below these details are two buttons: "Check Price at Chewy.com" (blue) and "Check Price On Amazon" (orange). A note below states: "Overall, Friskies Prime Filets With Salmon & Beef In Sauce is a below average cat food, earning 5 out of a possible 10 paws based on its nutritional analysis and ingredient list. For more information on our cat food analysis techniques, please click here." Another note below says: "Please note that CatFoodDB may earn a small commission on purchases made via links on this site. Thanks for your support!" At the bottom, there's a section titled "Ingredient Analysis" with a note: "When evaluating a cat food, the first five ingredients can tell you a lot about the quality of a product as they make up the bulk of the product. Ideally, you're looking for quality protein". On the right side of the page, there are several sidebar boxes: "CatFoodDB Search" with a search input field and a magnifying glass icon; a promotional box for "chewy.com" offering "30% OFF YOUR FIRST AUTOSHIP"; a box for "CatFoodDB's Best Wet Cat Foods"; a box for "CatFoodDB November 2021 Update"; and a box asking if users want CatFoodDB in their inbox, with a "Sign up" button.

# El primer enfoque

Sponsored by Jupyter

1. **Scrapeamos** la página web, sacamos un gran CSV.
2. **Cargamos** los datos manualmente en la libreta.
3. Análisis **exploratorio**. Producimos tablas y gráficos según lo que vamos averiguando sobre el dataset.
4. **Entrenamos** un par de modelos, tuneamos sus parámetros a ojo, optimizamos alguna métrica.
5. Conseguimos unos primeros **resultados** decentes. Somos felices, hacemos `ctrl+S` y descansamos.



**La libreta en cuestión:**

# PERO ESO NO ES TODO

**Y no estamos preparad@s...**



**El CEO de los gatos solicita un sistema de estimación de puntuaciones de comida.**

- Un modelo robusto con resultados fiables.
- Flexibilidad para cambiar el modelo, la persistencia...
- Una API a la que enviar queries con info nutricional.
- Dataset frecuentemente actualizado.
- Del front ya se encargan ellos. ;)

# Una evaluación honesta

## Nuestra libreta se queda corta

- Proceso ETL totalmente manual.
- Carga de datos artesanal.
- Fuertemente acoplada al dataset.
- Configuración dentro de la propia libreta.

**Loading and cleaning the data**

I begin by loading the csv file created by the spider and taking a good look at it

```
In [3]: df = pd.read_csv('../scraping/catfood/spiders/food_2020-03-11T08-47-45.csv')
df.head()
```

	brand	dma_ash	dma_cals	dma_carbs	dma_fat	dma_fiber	dma_prot	ga_ash	ga_carbs	ga_fat	ga_fiber	ga_moist	ga_pr
0	1st choice	10%	313/100g	40%	11%	7%	32%	9.0%	36.0%	10.0%	6.0%	10.0%	29.0
1	bravo	0%	87/100g	11%	31%	6%	53%	0.0%	2.0%	5.5%	1.0%	82.0%	9.5

```
In [5]: # Convert all percent values to floats

percent_cols = ['dma_ash',
                 'dma_carbs',
                 'dma_fat',
                 'dma_fiber',
                 'dma_prot',
                 'ga_ash',
                 'ga_carbs',
                 'ga_fat',
                 'ga_fiber',
                 'ga_moist',
                 'ga_prot']

for col in percent_cols:
    df[col] = df[col].str.rstrip('%').astype('float')
```

- Tests de datos a ojímetro.
- Tests unitarios inexistentes.
- No es obvio cómo ejecutar la libreta, poco robusta.
- Sistema de input de datos muy insuficiente. Validación nula.
- Las métricas hay que evaluarlas personalmente.
- Ya ni hablamos sobre APIs o contenedores.

```
In [6]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3165 entries, 0 to 3164
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   brand            3165 non-null    object  
 1   dma_ash          3165 non-null    float64 
 2   dma_cals         3165 non-null    float64 
 3   dma_carbs        3165 non-null    float64 
 4   dma_fat          3165 non-null    float64 
 5   dma_fiber         3165 non-null    float64 
 6   dma_prot         3165 non-null    float64 
 7   ga_ash           3165 non-null    float64 
 8   ga_carbs         3165 non-null    float64 
 9   ga_fat           3165 non-null    float64 
 10  ga_fiber          3165 non-null    float64 
 11  ga_moist          3165 non-null    float64
```

```
In [26]: n_paws_preds
Out[26]: ['dma_ash', 'dma_cals', 'dma_carbs', 'dma_prot', 'ga_moist']

In [29]: datos_whiskas = np.array([1.7, 76, 21, 50, 84]).reshape(1, -1)
lr.predict(datos_whiskas)

Out[29]: array([4])
```

```
In [24]: sns.heatmap(data=confusion_matrix(y_test, y_pred),
                     square=True,
                     annot=True,
                     fmt='d')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

precision      recall   f1-score  support
1             0.84      0.70      0.76       87
2             0.78      0.79      0.78      199
3             0.81      0.86      0.83      373
4             0.82      0.83      0.83      289
5             0.94      0.80      0.87       97

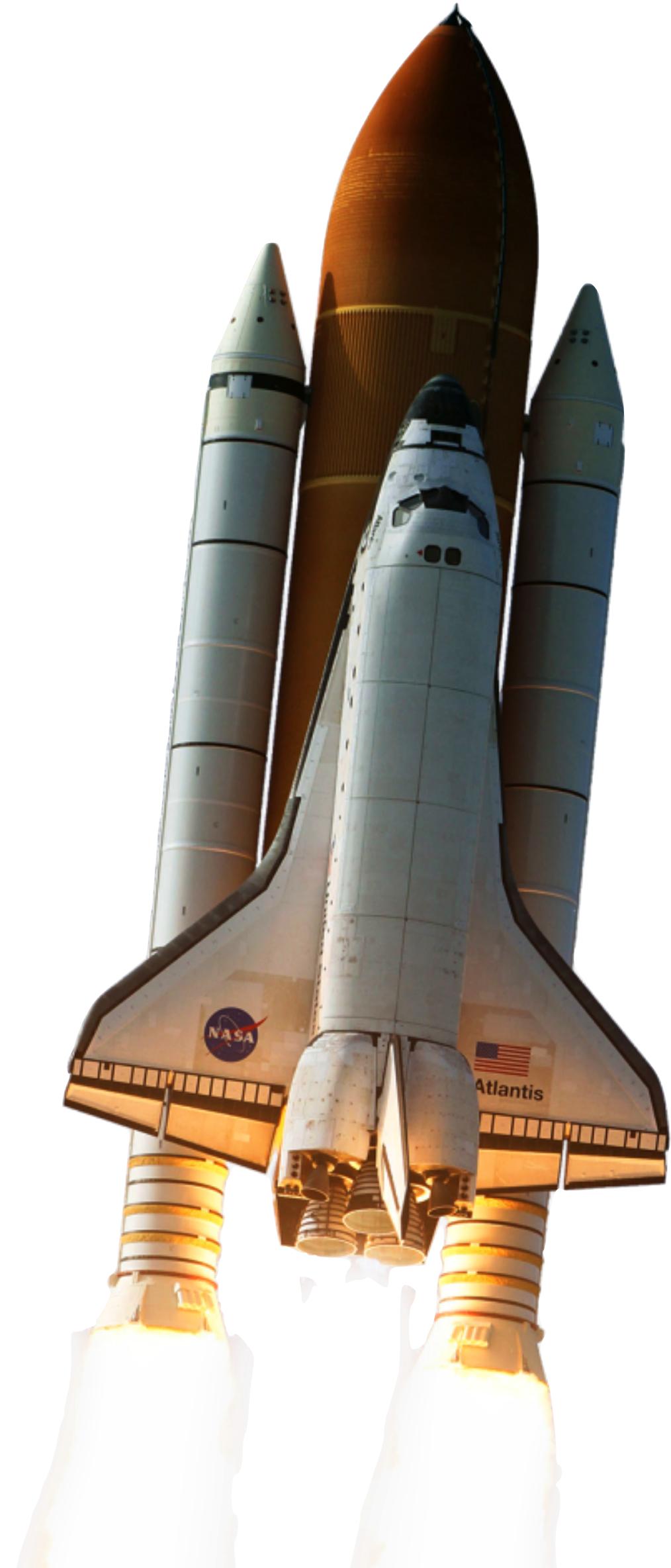
accuracy                  0.82      1045
macro avg                 0.84      0.80      0.81      1045
weighted avg               0.82      0.82      0.82      1045

[[ 61  26   0   0   0]
 [ 12 157  30   0   0]]
```

# De Jupyter a la Tierra

## Requerimientos para iniciar la canonización

- Tenemos una idea sobre cómo son las features del producto final.
- Tenemos claros cuáles son los defectos de nuestra libreta actual.
- Tenemos claro cuál es el proceso ML que queremos destilar.
- Tenemos una idea sobre qué es lo que queremos encapsular, monitorizar, configurar, modularizar...



# Presentando Kedro

## ¿Qué es?

- Un framework: un estándar robusto pero flexible con un vocabulario interno bien definido
- Una herramienta para construir pipelines de datos
- Una base sobre la que conectar otros plugins y funcionalidad
- Una buena solución para varios problemas

## ¿Y qué NO es?

- Una librería de ML
- Un orquestador
- Un servidor
- MLflow
- Un pokemon
- Una solución perfecta para todos los problemas

# Definiendo un lenguaje común

## Conceptos básicos de Kedro

**Catálogo:** Sistema de abstracción de carga y guardado de datos. Mapea datasets con nombres. Patrón repo.

catfood\_dataset,  
model\_input\_table, trained\_model,  
confusion\_matrix

**Nodo:** Bloque de construcción, trozo de tarea. Inputs y outputs bien definidas. Funciones puras.

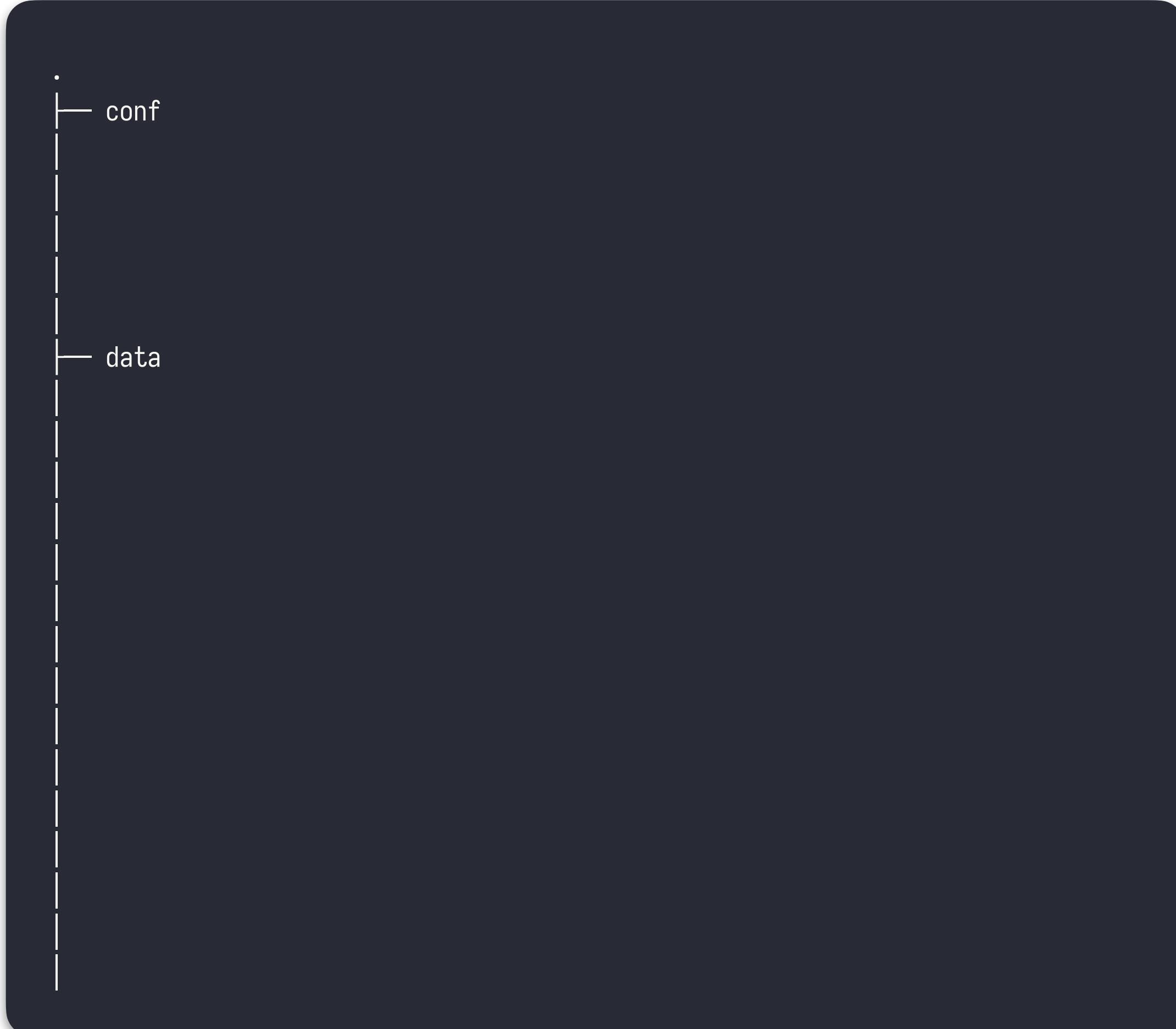
preprocess\_raw\_data, train\_model

**Pipeline:** Secuencias de nodos, inputs y outputs posiblemente encadenadas. Tareas completas, o simplemente *nodos complejos modulares*. DAGs.

data\_processing, data\_science,  
prediction

# El árbol de directorios

```
$ tree .
```



# El árbol de directorios

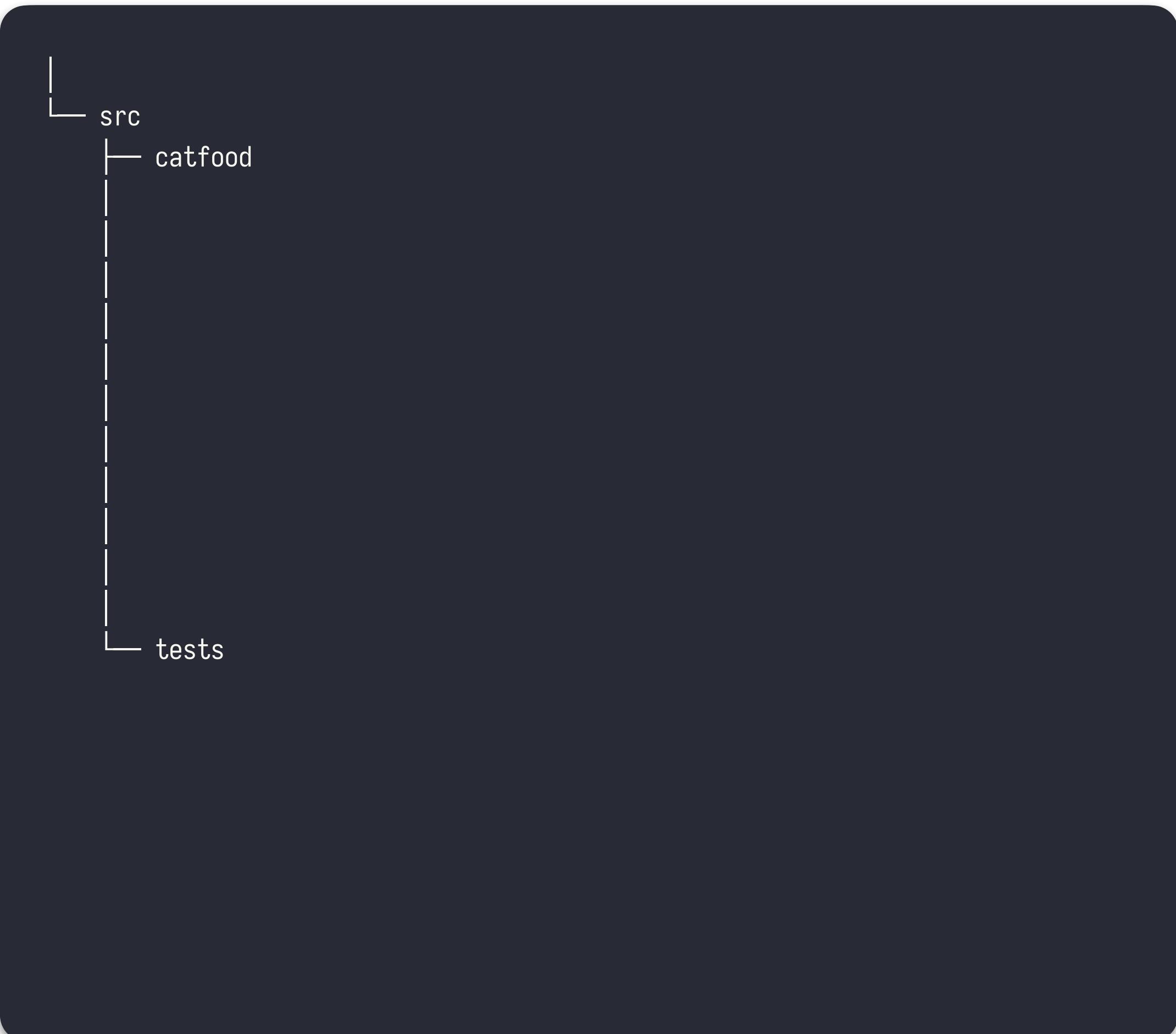
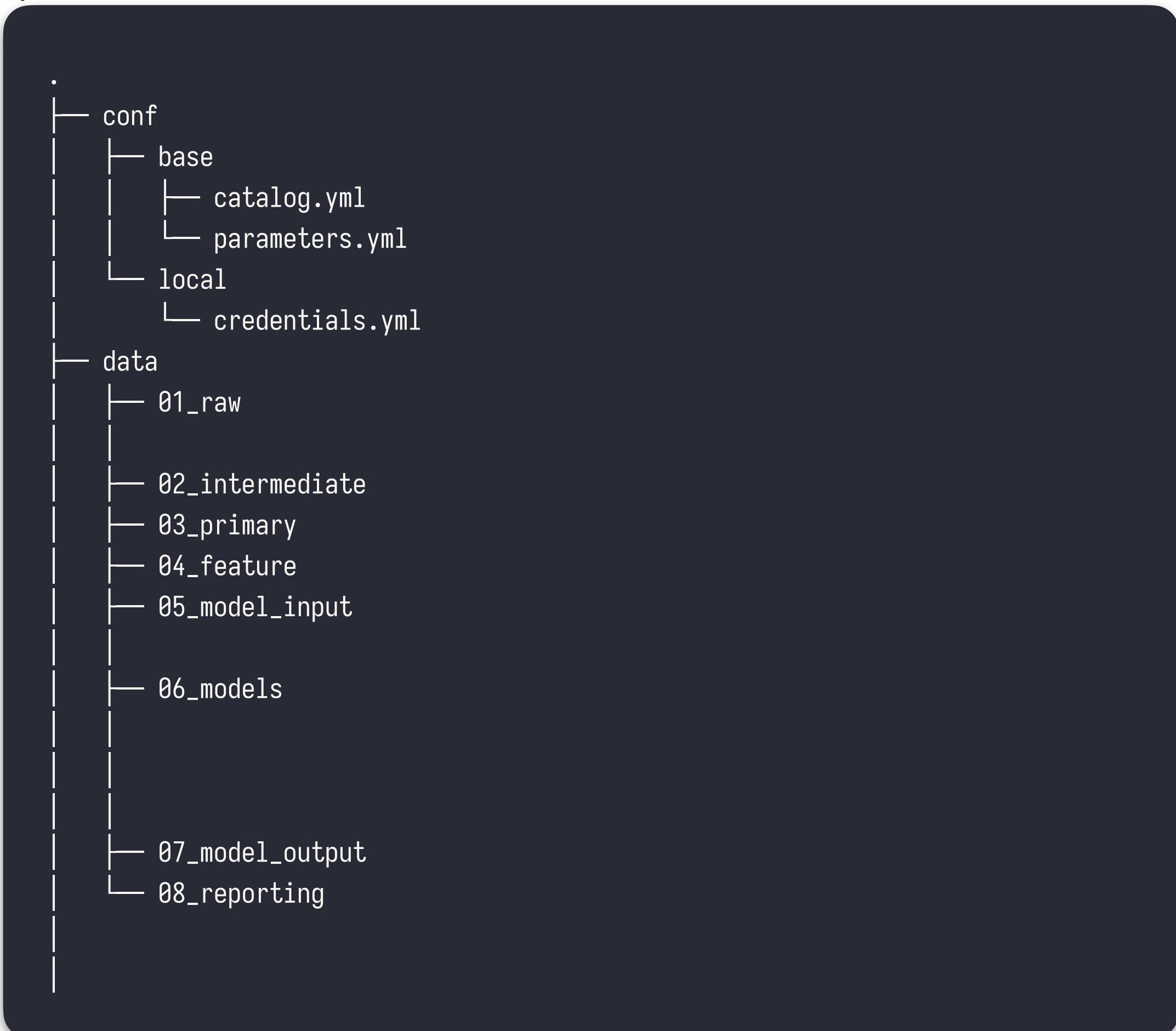
```
$ tree .
```

```
.
├── conf
│   ├── base
│   │   ├── catalog.yml
│   │   └── parameters.yml
│   └── local
│       └── credentials.yml
└── data
```

```
└── src
    ├── catfood
    └── tests
```

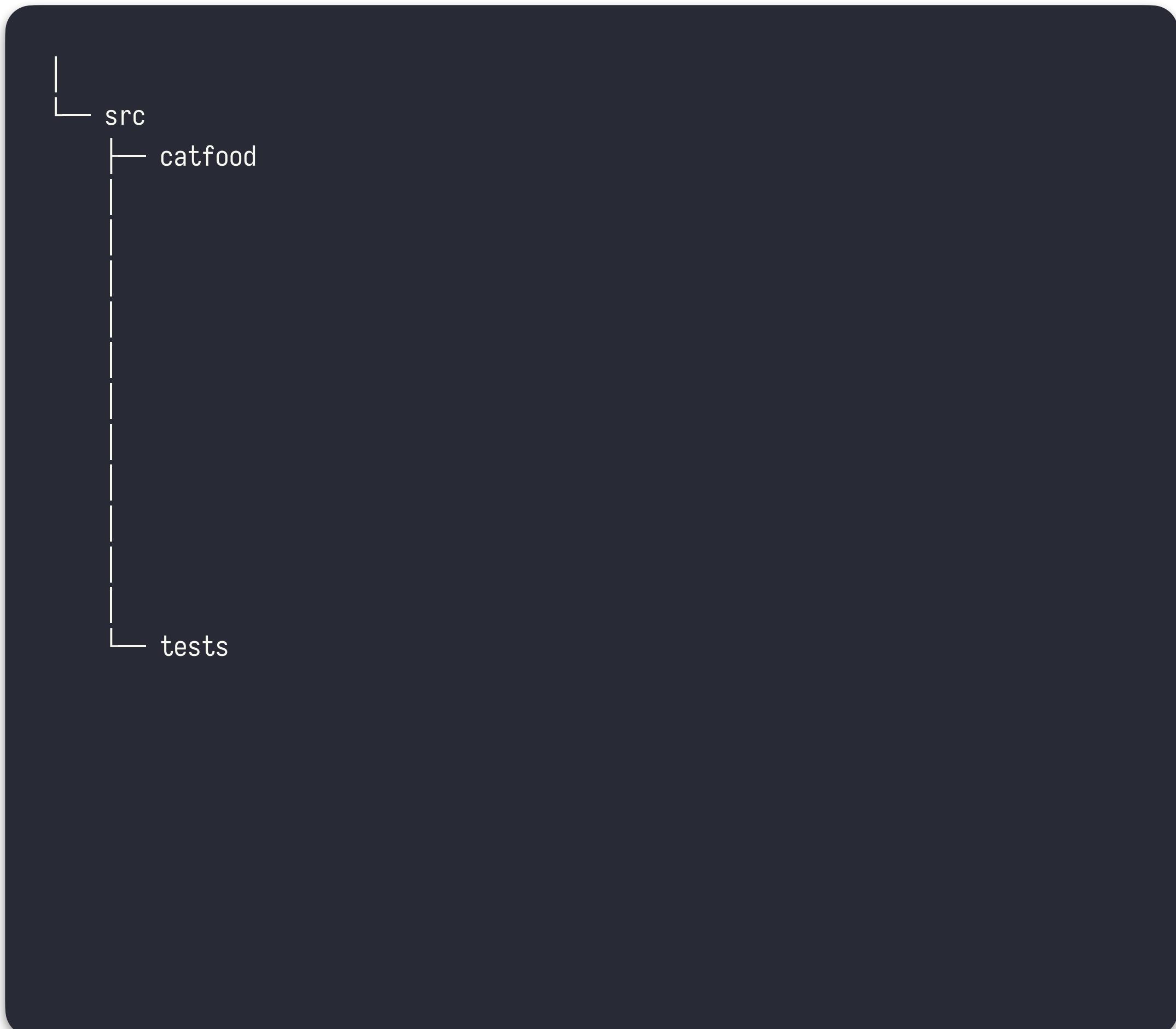
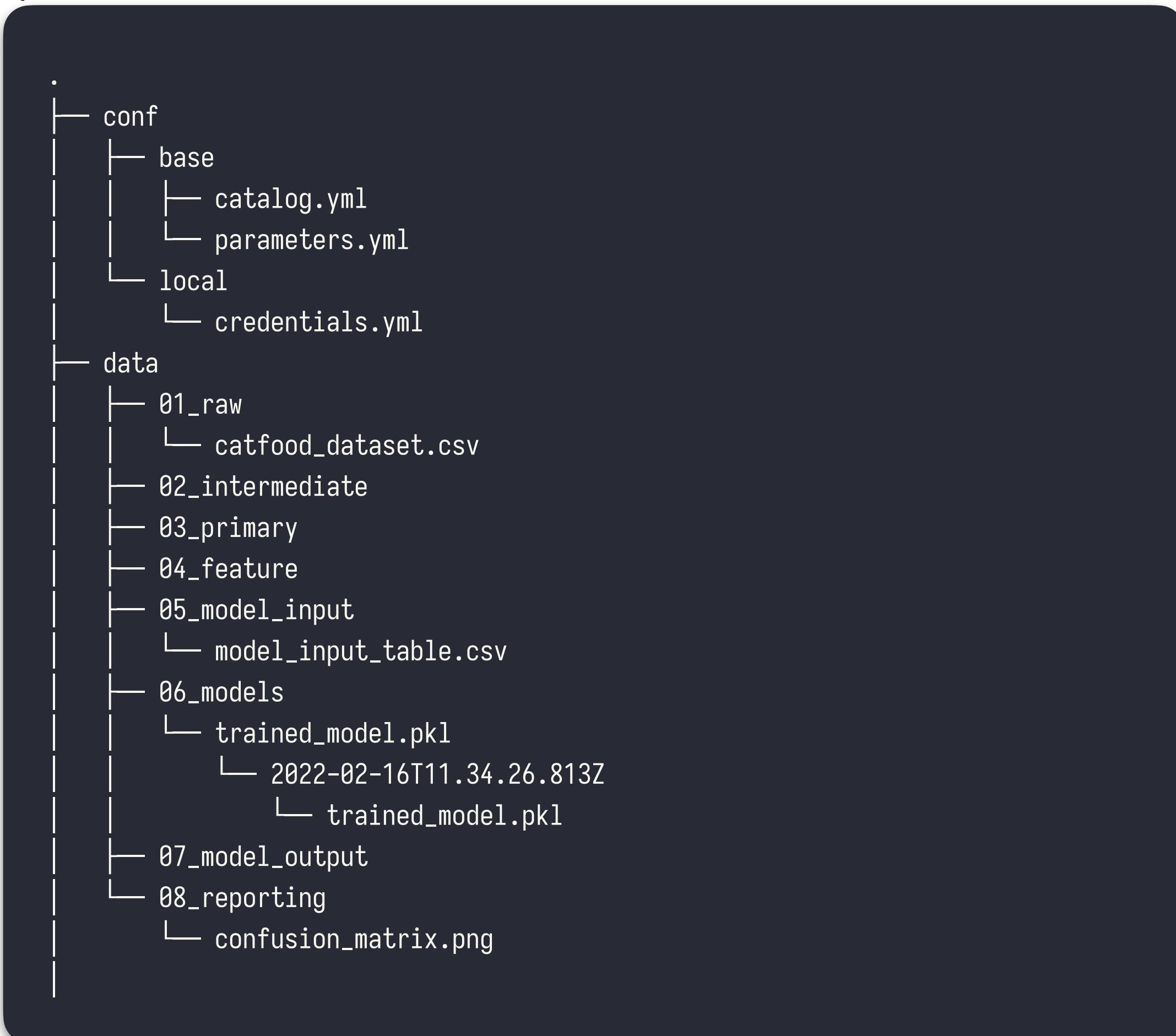
# El árbol de directorios

```
$ tree .
```



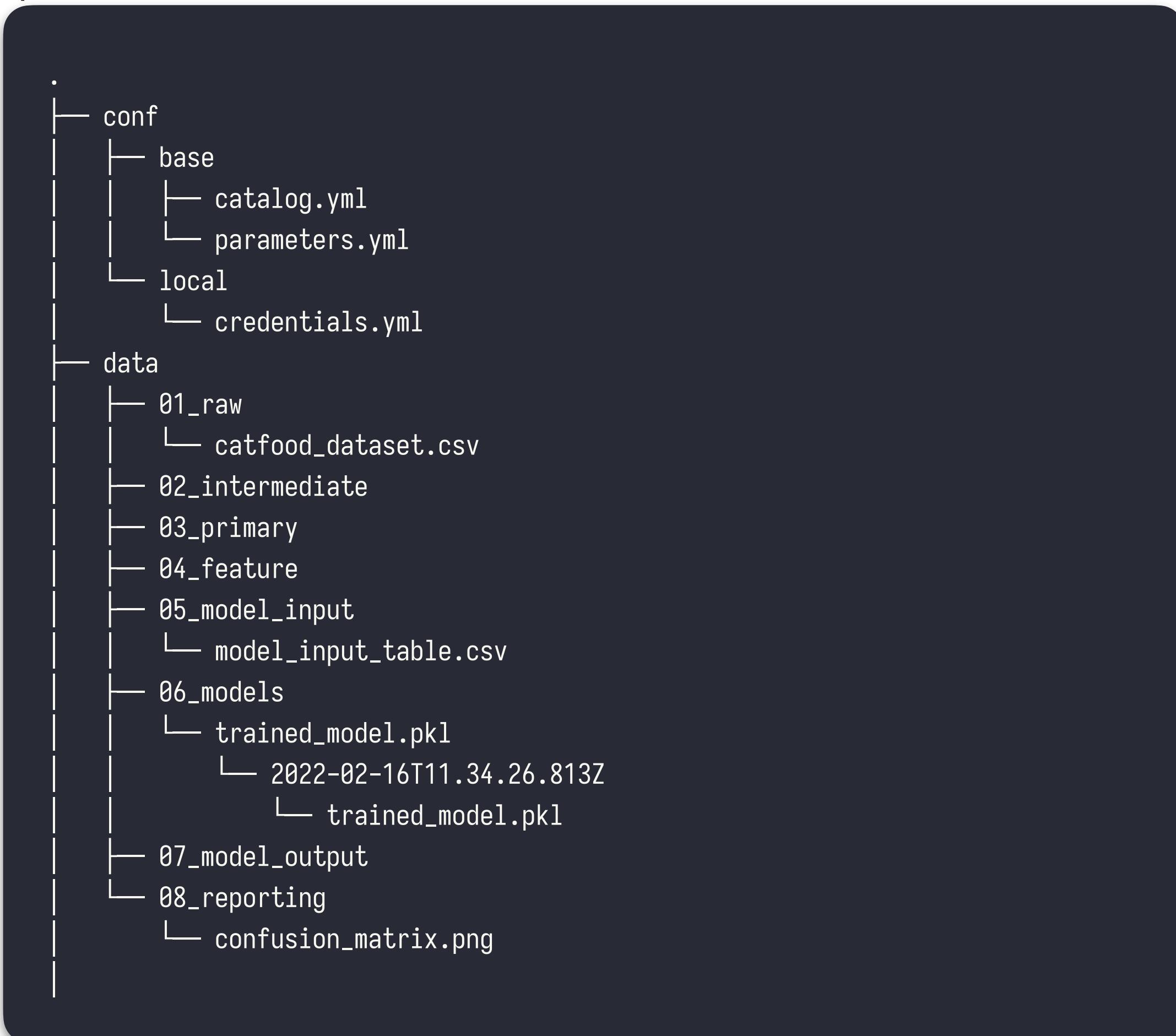
# El árbol de directorios

```
$ tree .
```



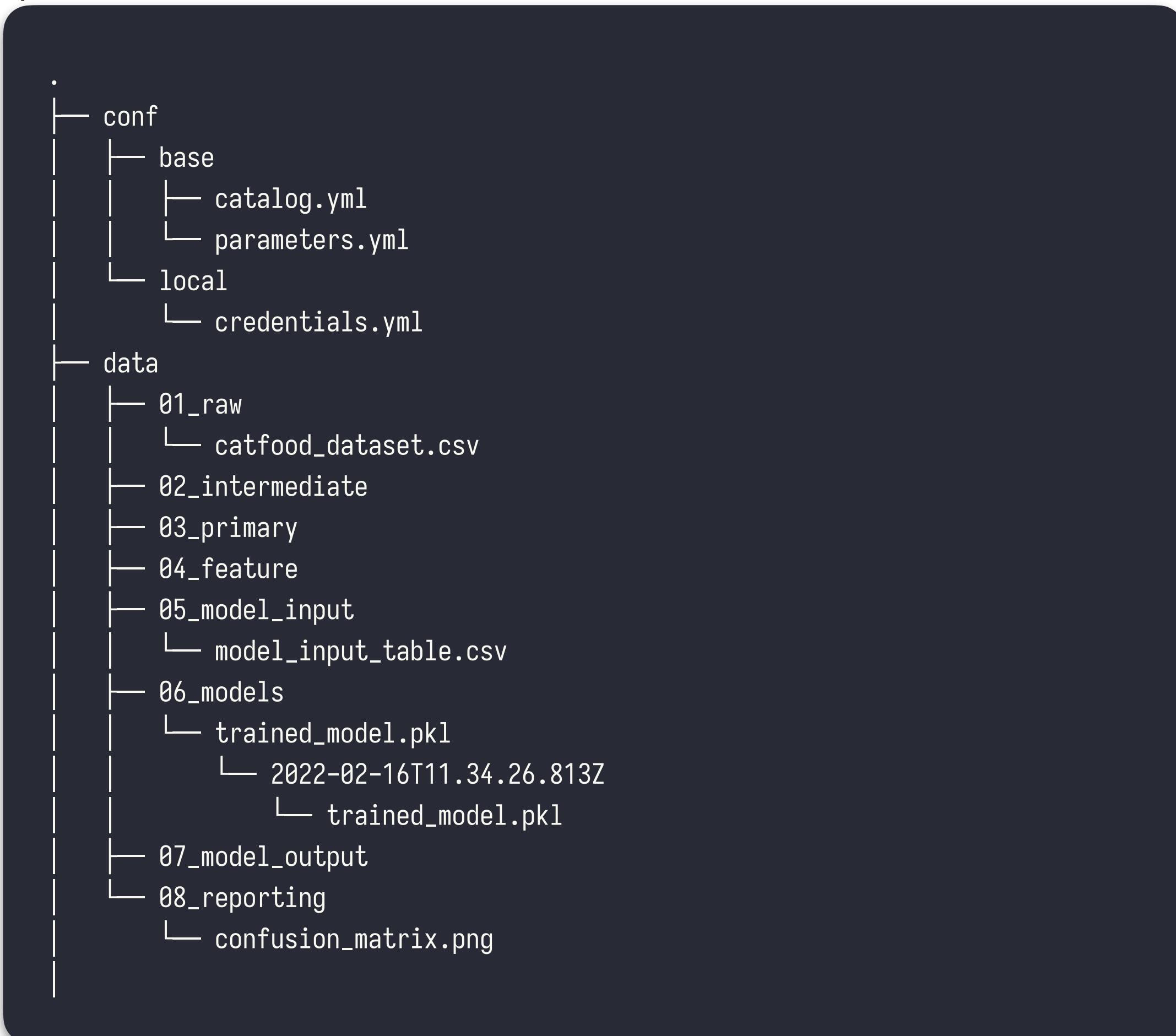
# El árbol de directorios

```
$ tree .
```



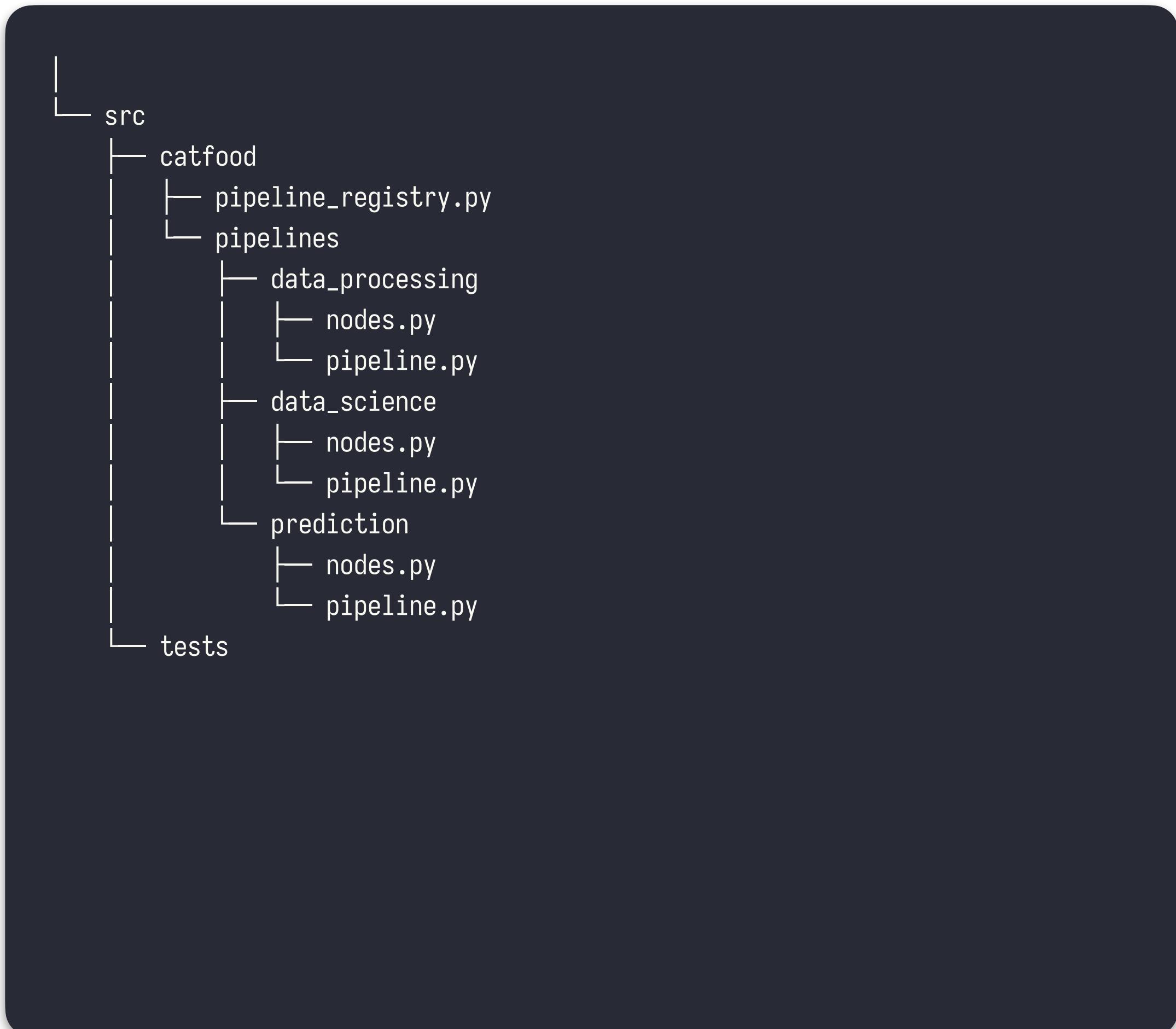
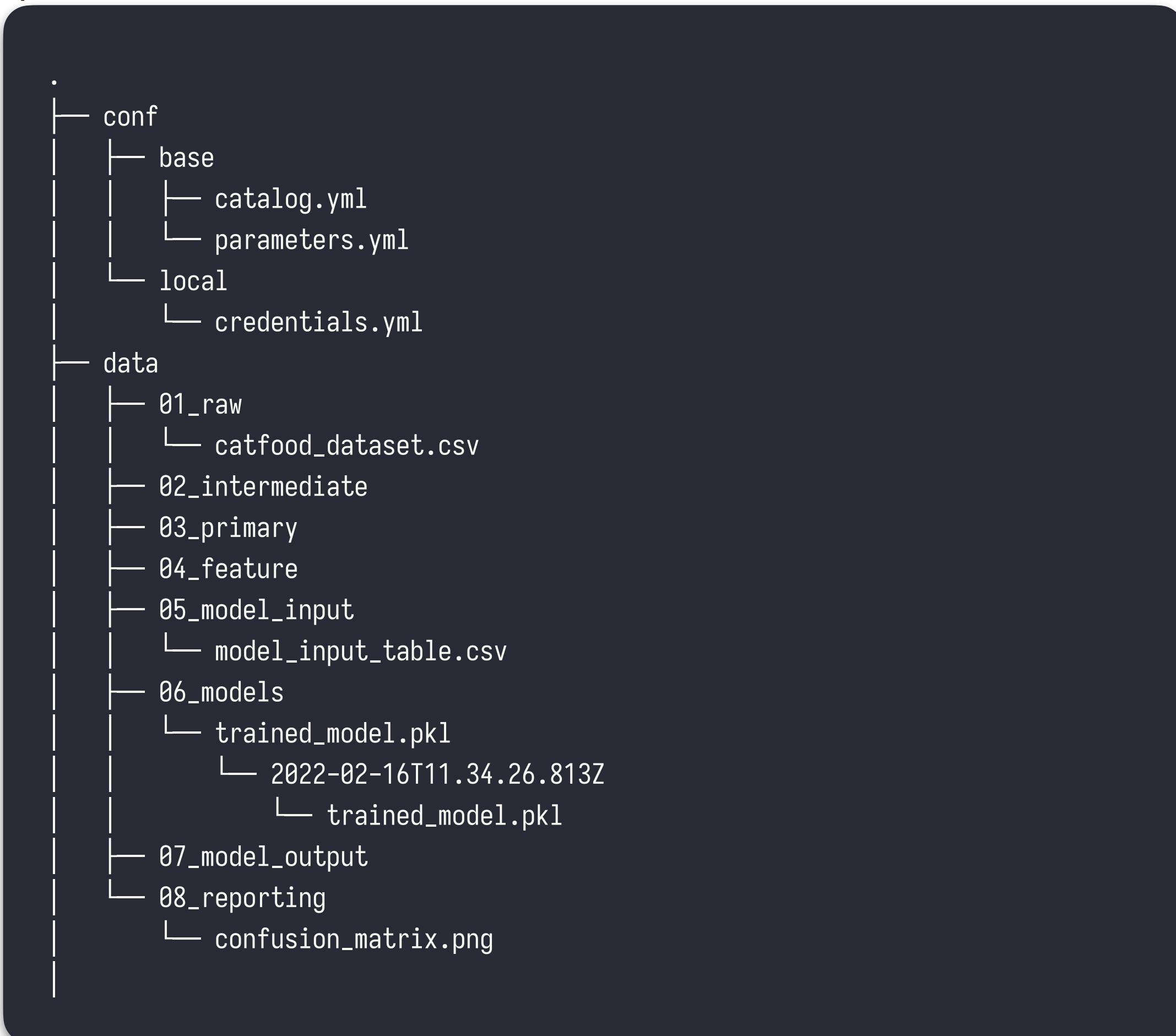
# El árbol de directorios

```
$ tree .
```



# El árbol de directorios

```
$ tree .
```



# El árbol de directorios

```
$ tree .
```



# El catálogo y los parámetros

conf/base/catalog.yml

```
catfood_dataset:  
  type: pandas.CSVDataSet  
  filepath: data/01_raw/catfood_dataset.csv  
  layer: raw  
  
model_input_table:  
  type: pandas.CSVDataSet  
  filepath: data/05_model_input/model_input_table.csv  
  layer: model_input  
  
trained_model:  
  type: pickle.PickleDataSet  
  filepath: data/06_models/trained_model.pkl  
  layer: models  
  versioned: true  
  
confusion_matrix:  
  type: matplotlib.MatplotlibWriter  
  filepath: data/08_reporting/confusion_matrix.png  
  layer: reporting
```

conf/base/parameters.yml

```
# Model params  
multi_class: multinomial  
solver: newton-cg  
target: nutrition_paws  
features:  
  - dma_ash  
  - dma_cals  
  - dma_carbs  
  - dma_prot  
  - ga_moist  
  
# Model testing params  
test_size: 0.2  
random_state: 3  
stratify: true
```

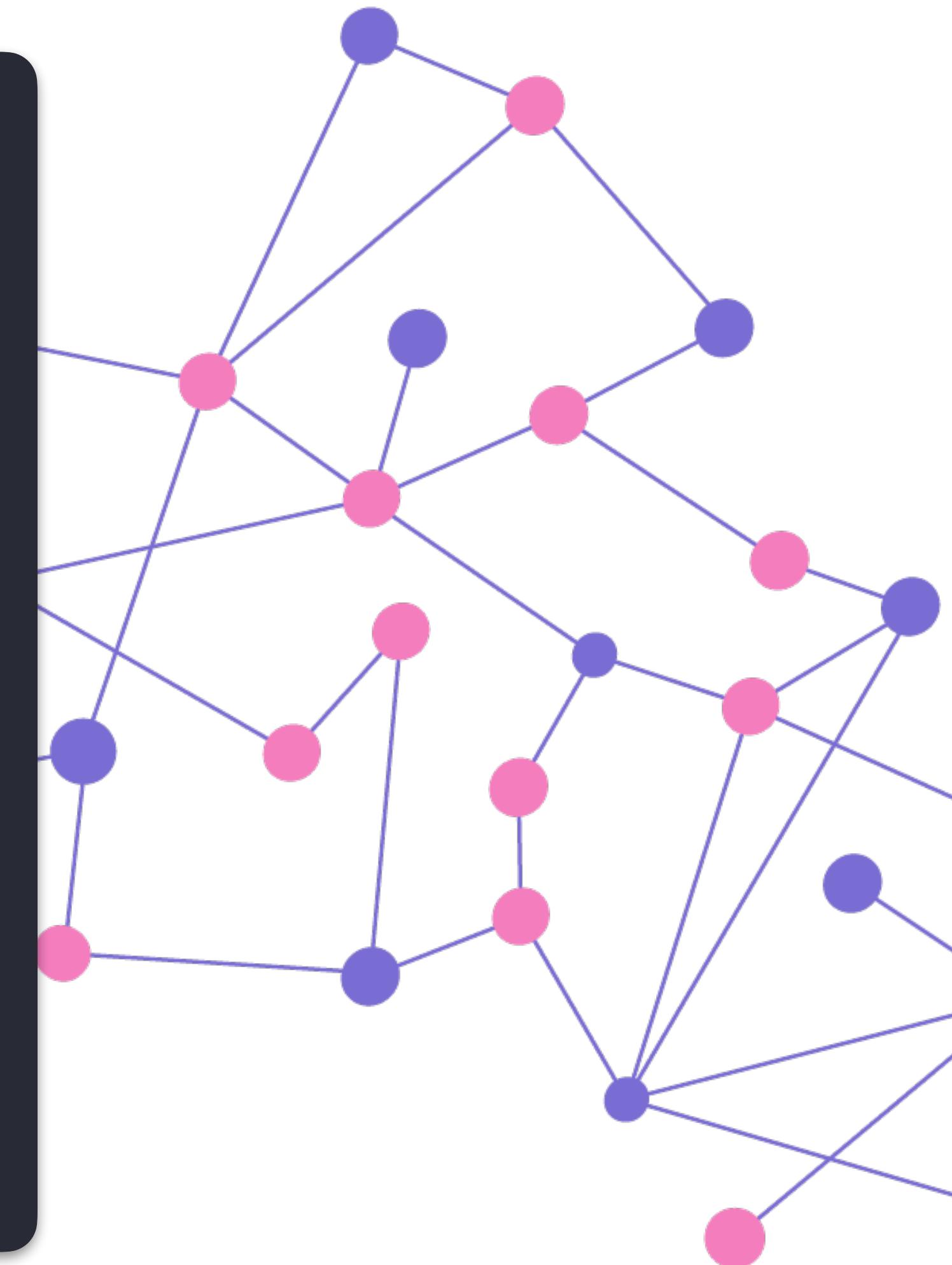
# Los nodos...

src/catfood/pipelines/data\_science/nodes.py

```
def split_data(data: pd.DataFrame, parameters: Dict) -> Tuple:
    X, y = data[parameters["features"]], data[parameters["target"]]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=parameters["test_size"], random_state=parameters["random_state"]
    )
    return X_train, X_test, y_train, y_test

def train_model(
    X_train: pd.DataFrame, y_train: pd.Series, parameters: Dict
) -> LogisticRegression:
    regressor = LogisticRegression(multi_class=parameters["multi_class"], solver=parameters["solver"])
    regressor.fit(X_train, y_train)
    return regressor

def evaluate_model(
    trained_model: LogisticRegression, X_test: pd.DataFrame, y_test: pd.Series
):
    y_pred = trained_model.predict(X_test)
    f = sns.heatmap(data=confusion_matrix(y_test, y_pred), square=True, annot=True, fmt="d")
    return f
```



# ...y las pipelines

src/catfood/pipelines/data\_science/pipeline.py

```
def create_pipeline(**kwargs):
    return Pipeline(
        [
            node(
                func=split_data,
                inputs=["model_input_table", "parameters"],
                outputs=["X_train", "X_test", "y_train", "y_test"],
                name="split_data_node",
            ),
            node(
                func=train_model,
                inputs=["X_train", "y_train", "parameters"],
                outputs="trained_model",
                name="train_model_node",
            ),
            node(
                func=evaluate_model,
                inputs=["trained_model", "X_test", "y_test"],
                outputs="confusion_matrix",
                name="evaluate_model_node",
            ),
        ],
    )
```



# El registro de pipelines

src/catfood/pipeline\_registry.py

```
from catfood.pipelines import data_processing as dp
from catfood.pipelines import data_science as ds
from catfood.pipelines import prediction as pred

def register_pipelines() -> Dict[str, Pipeline]:
    data_processing_pipeline = dp.create_pipeline()
    data_science_pipeline = ds.create_pipeline()
    prediction_pipeline = pred.create_pipeline()

    return {
        "__default__": data_processing_pipeline + data_science_pipeline,
        "dp": data_processing_pipeline,
        "ds": data_science_pipeline,
        "pred": prediction_pipeline,
    }
```

**kedro-viz:**

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo
- Añadir tests de datos
- Añadir tests de integración
- Monitorizar parámetros y resultados
- Orquestar
- Contenedorizar y distribuir
- Diseñar una pipeline de ETL con nuestro scraper

# Ejecutando pipelines desde el código

src/catfood/mi\_aplicacion\_ejecutable.py

```
from pathlib import Path

from kedro.framework.startup import _get_project_metadata, _add_src_to_path
from kedro.framework.session import KedroSession

project_path = Path.cwd()
metadata = _get_project_metadata(project_path)
_add_src_to_path(metadata.source_dir, project_path)

with KedroSession.create(
    package_name=metadata.package_name,
    project_path=project_path
) as session:
    output = session.run(pipeline_name="pred")
```

# Una API sencilla

src/catfood/api/app.py

```
app = FastAPI(
    title="CatfoodAPI",
    version="0.1",
)

@app.post("/predict_score", response_model=NutritionalScore)
def predict_score(nutritional_info: NutritionalInfo):
    with KedroSession.create(
        package_name=metadata.package_name,
        project_path=project_path,
        extra_params={"prediction_features": dict(nutritional_info)},
    ) as session:
        output = session.run(pipeline_name="pred")
    return NutritionalScore(score=output["score"])

def run():
    uvicorn.run(
        "app:app",
        host="0.0.0.0",
        port=1234,
    )
```

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo
- Añadir tests de datos
- Añadir tests de integración
- Monitorizar parámetros y resultados
- Orquestar
- Contenedorizar y distribuir
- Diseñar una pipeline de ETL

`fastapi`, `uvicorn`, `mlflow`, `bentoml`

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo fastapi, uvicorn, mlflow, bentoml
- Añadir tests de datos pandera, great-expectations
- Añadir tests de integración
- Monitorizar parámetros y resultados
- Orquestar
- Contenedorizar y distribuir
- Diseñar una pipeline de ETL

# Una buena base

## Para construir una app real

Diseñar una API y servir el modelo

fastapi, uvicorn, mlflow, bentoml

Añadir tests de datos

pandera, great-expectations

Añadir tests de integración

kedro test

Monitorizar parámetros y resultados

Orquestar

Contenedorizar y distribuir

Diseñar una pipeline de ETL

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo fastapi, uvicorn, mlflow, bentoml
- Añadir tests de datos pandera, great-expectations
- Añadir tests de integración kedro test
- Monitorizar parámetros y resultados kedro-mlflow
- Orquestar
- Contenedorizar y distribuir
- Diseñar una pipeline de ETL

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo **fastapi, uvicorn, mlflow, bentoml**
- Añadir tests de datos **pandera, great-expectations**
- Añadir tests de integración **kedro test**
- Monitorizar parámetros y resultados **kedro-mlflow**
- Orquestar **crontab, prefect, airflow**
- Contenedorizar y distribuir
- Diseñar una pipeline de ETL

# Una buena base

## Para construir una app real

- Diseñar una API y servir el modelo fastapi, uvicorn, mlflow, bentoml
- Añadir tests de datos pandera, great-expectations
- Añadir tests de integración kedro test
- Monitorizar parámetros y resultados kedro-mlflow
- Orquestar crontab, prefect, airflow
- Contenedorizar y distribuir kedro-docker, kedro package
- Diseñar una pipeline de ETL



Todo el código disponible en  
[github.com/oscarigrejas/de-jupyter-a-la-tierra](https://github.com/oscarigrejas/de-jupyter-a-la-tierra)

**¡Muchas gracias!**