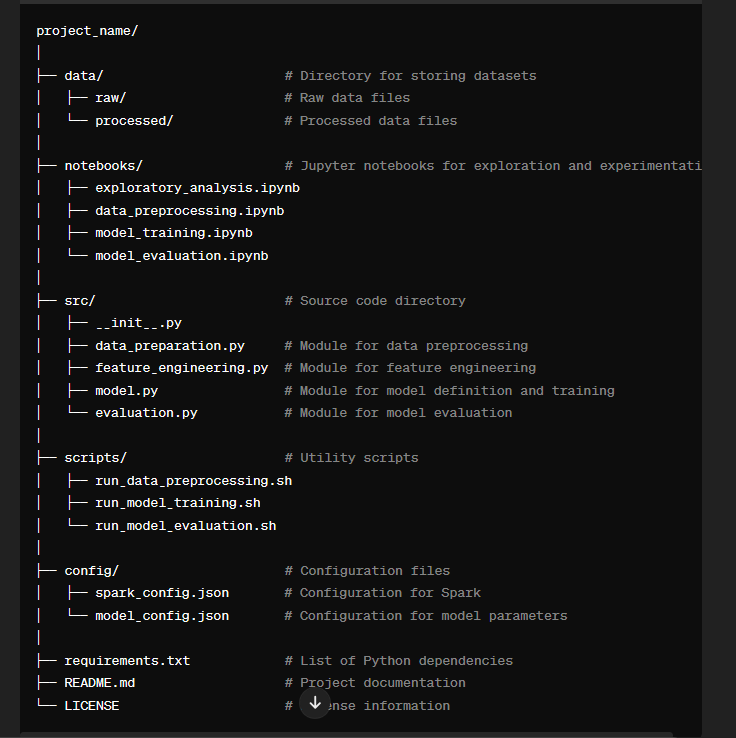
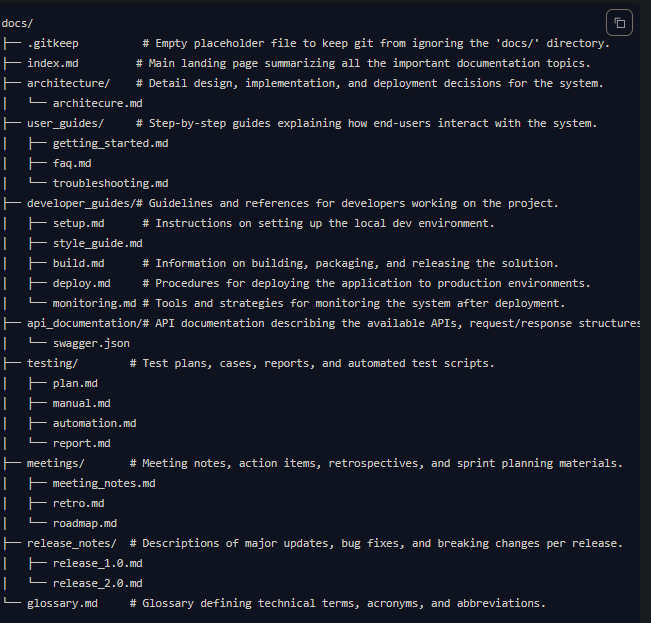
Project Layout



A screenshot of a computer

Description automatically generated



[Basic writing and formatting syntax - GitHub Docs](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax)

Sure, let's provide a simple example for each file and demonstrate how they might connect with each other in a PySpark machine learning project.

1. data\_preparation.py:

from pyspark.sql import SparkSession

def load\_data(spark, file\_path):

return spark.read.csv(file\_path, header=True, inferSchema=True)

def preprocess\_data(data):

# Perform data preprocessing steps

preprocessed\_data = data.fillna(0) # Filling missing values with zeros as an example

return preprocessed\_data

2. feature\_engineering.py:

from pyspark.ml.feature import VectorAssembler

def create\_feature\_vector(data):

feature\_cols = data.columns[:-1] # Assuming the last column is the target variable

assembler = VectorAssembler(inputCols=feature\_cols, outputCol='features')

return assembler.transform(data)

3. model.py;

from pyspark.ml.classification import LogisticRegression

def train\_model(data):

lr = LogisticRegression(featuresCol='features', labelCol='label', maxIter=10)

model = lr.fit(data)

return model

4. evaluation.py;

from pyspark.ml.evaluation import BinaryClassificationEvaluator

def evaluate\_model(model, test\_data):

predictions = model.transform(test\_data)

evaluator = BinaryClassificationEvaluator(rawPredictionCol='prediction', labelCol='label')

accuracy = evaluator.evaluate(predictions)

return accuracy

5. notebooks/data\_preprocessing.ipynb:

from src import data\_preparation

spark = SparkSession.builder.appName('DataPrep').getOrCreate()

data = data\_preparation.load\_data(spark, 'data/raw/data.csv')

preprocessed\_data = data\_preparation.preprocess\_data(data)

preprocessed\_data.write.csv('data/processed/preprocessed\_data.csv')

6. notebooks/model\_training.ipynb:

from src import data\_preparation, feature\_engineering, model

spark = SparkSession.builder.appName('ModelTraining').getOrCreate()

data = data\_preparation.load\_data(spark, 'data/processed/preprocessed\_data.csv')

feature\_vector = feature\_engineering.create\_feature\_vector(data)

trained\_model = model.train\_model(feature\_vector)

trained\_model.save('models/trained\_model')

7. notebooks/model\_evaluation.ipynb:

from src import model, evaluation

spark = SparkSession.builder.appName('ModelEvaluation').getOrCreate()

test\_data = data\_preparation.load\_data(spark, 'data/processed/test\_data.csv')

loaded\_model = model.load\_model('models/trained\_model')

accuracy = evaluation.evaluate\_model(loaded\_model, test\_data)

print("Model Accuracy:", accuracy)

This layout demonstrates how the modules interact with each other. The `data\_preparation.py` module loads and preprocesses data, the `feature\_engineering.py` module creates feature vectors, the `model.py` module defines and trains the model, and the `evaluation.py` module evaluates the trained model. The Jupyter notebooks orchestrate the flow by importing these modules and utilizing their functionalities.

1. scripts/run\_data\_preprocessing.sh:

#!/bin/bash

# Assuming Spark is installed and configured

spark-submit --master local[\*] src/data\_preparation.py

2. scripts/run\_model\_training.sh:

#!/bin/bash

# Assuming Spark is installed and configured

spark-submit --master local[\*] src/model.py

3. scripts/run\_model\_evaluation.sh:

#!/bin/bash

# Assuming Spark is installed and configured

spark-submit --master local[\*] src/evaluation.py

4. config/spark\_config.json:

{

"spark.executor.memory": "4g",

"spark.driver.memory": "4g",

"spark.executor.instances": "2"

}

5. config/model\_config.json:

{

"maxIter": 10,

"regParam": 0.01

}

6. requirements.txt;

pyspark==3.2.0

7. README.md:

# Project Name

## Description

Describe your project here.

## Installation

- Clone the repository

- Install dependencies using `pip install -r requirements.txt`

## Usage

- Follow instructions in scripts/ directory to run different parts of the project.

- Modify configurations in config/ directory as needed.`

## License

This project is licensed under the [License Name] - see the [LICENSE](LICENSE) file for details.

8. \*\*LICENSE\*\*: This can be any license file, such as MIT, Apache, etc., depending on your choice. For example, for MIT license:

MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

These files provide the necessary scripts for running different parts of the project, configurations for Spark and the model, Python dependencies, project documentation, and licensing information. Adjustments can be made based on specific project requirements and preferences.