Institute for Advanced Studies
in Basic Sciences
GavaZang, Zanjan, Iran

Air Quality Prediction Pipeline

Prof. Mahmood Shirazi

Report for Final Project

Reyhane Mohammadi (14044112)

Faeze Ahmadi (14044141)

December 2025

پایپلاین پیش‌بینی کیفیت آب و هوا

کد کامل این پروژه و فایل‌های مربوطه در ریپازیتوری گیت‌هاب قابل دسترسی هست:

https://github.com/Faeze-Ahmadi/air-quality-prediction-pipeline

این پروژه به بررسی و پیش‌بینی کیفیت هوا با استفاده از داده‌های مختلف و مدل‌های یادگیری ماشین پرداخته است. هدف اصلی آن پیش‌بینی شاخص کیفیت هوا (AQI) برای مناطق مختلف است، که یکی از مهم‌ترین شاخص‌ها برای ارزیابی آلودگی هوا و تاثیرات آن بر سلامت انسان‌ها محسوب می‌شود. این پروژه در دو بخش اصلی طراحی شده است: یکی برای استفاده از داده‌های UCI Air Quality Dataset و دیگری برای دریافت داده‌های ریل‌تایم از سیستم AQICN (پلتفرم جهانی کیفیت هوا).

در قسمت اول پروژه، ما از داده‌های UCI Air Quality استفاده کردیم که شامل اندازه‌گیری‌های مختلفی از آلاینده‌ها و گازهای مضر در هوای محیط است. با استفاده از این داده‌ها، مدلی برای پیش‌بینی کیفیت هوا آموزش داده شده که نتایج قابل قبولی از خود ارائه می‌دهد. این مدل به طور خاص برای پیش‌بینی مقدار مونوکسید کربن (CO) در مناطق مختلف طراحی شده است. با استفاده از رگرسیون خطی، مدلی ساختیم که توانایی پیش‌بینی غلظت CO را دارد و به این ترتیب قادر است به عنوان یک ابزار مفید برای پیش‌بینی وضعیت کیفیت هوا در آینده مورد استفاده قرار گیرد.

اما پروژه فقط به داده‌های گذشته محدود نشده و از آن مهم‌تر، ما قابلیت استفاده از داده‌های آلودگی هوا در زمان واقعی را هم به آن اضافه کردیم. از طریق AQICN API، پروژه به طور مستقیم به داده‌های هوای واقعی و لحظه‌ای از شهرهای مختلف ایران متصل می‌شود. این بخش از پروژه برای استفاده از داده‌های زنده در نظر گرفته شده تا بتوان به طور دقیق‌تری وضعیت کیفیت هوا را پیش‌بینی و بررسی کرد. هدف این قسمت از پروژه، دریافت اطلاعات از شهرهای مختلف ایران و پردازش آن‌ها برای به‌دست آوردن پیش‌بینی‌های لحظه‌ای و دقیق از وضعیت آلودگی هوا بود. در حال حاضر، مسئله‌ای که باقی مانده، دریافت داده‌ها از API AQICN است که به دلیل مشکلات مربوط به توکن API، هنوز گاهی با خطا مواجه می‌شود. البته همین موضوع برای ما فرصت خوبی بود که نشان دهیم از دانسته‌هایمان چطور استفاده کردیم.

یکی از موضوعات مهم در کلاس دکتر شیرازی بحث خطاگیری بود، اینکه اگر متغیر درست وارد نشد، دیتا وجود نداشت، تابع پیدا نشد و از این دست موارد، خروجی چه می‌شود؟ برنامه کِرَش می‌کند یا یک پیام خطا به ما می‌دهد و در مورد اینکه مشکل از کجاست، اطلاع می‌دهد؟

پروژه ما در مسیر دیتاست AQICN کامل و درست است، تمام فرآیندهای لود دیتا، Preprocessing، آموزش مدل و .. به خوبی انجام شده؛ اما اینکه API گاهی درست نیست، مشکلی است که می‌توان درآینده هم حل نمود. برای همین ما تصمیم گرفتیم، این مشکل را در قالب یک فرصت ببینیم و به استاد نشان دهیم که درس برنامه نویسی پیشرفته را آن‌طور که ایشان از ما انتظار داشتند، یاد گرفتیم.

به همین ترتیب، این پروژه با دو مود اجرایی طراحی شده که هرکدام مختص یک دیتاست خاص است. در مود اول، با استفاده از دیتاست UCI، مدل رگرسیون خطی آموزش داده شده و پیش‌بینی‌هایی از میزان مونوکسید کربن (CO) در هوای محیط انجام می‌شود. در مود دوم، برای دریافت داده‌های AQICN و پردازش داده‌های زنده از شهرهای ایران، پروژه آماده است و در آینده نزدیک با اصلاحات لازم به طور کامل فعال خواهد شد.

در نهایت، این پروژه به عنوان یک ابزار پیش‌بینی وضعیت کیفیت هوا و تحلیل داده‌های محیطی طراحی شده است که در بهبود کیفیت زندگی و کاهش تاثیرات منفی آلودگی هوا بر سلامت انسان‌ها موثر خواهد بود. این پروژه نه تنها یک گام مهم در جهت به‌کارگیری یادگیری ماشین برای مسائل محیطی است، بلکه نقطه شروعی برای تحقیقات بیشتر در زمینه داده‌های زنده و پیش‌بینی وضعیت هوا در سطح جهانی و محلی خواهد بود.

ساختار پروژه:

src/

```
├── ml/
|  ├── train_aqicn_model.py
|  ├── train_uci_model.py
|  ├── pipeline/
|  |  ├── __init__.py
|  |  ├── aqicn_runner.py
|  |  ├── collector.py
|  |  └── uci_runner.py
|  ├── storage/
|  |  ├── __init__.py
|  |  └── sqlite_storage.py
|  ├── tests/
|  |  └── test_aqicn_token.py
|  ├── visualization/
|  |  ├── __init__.py
|  |  ├── plots.py
|  |  └── uci_plots.py
|  ├── config/
|  |  └── settings.py
|  ├── data_loader/
|  |  ├── __init__.py
|  |  ├── aqi_api_client.py
|  |  └── uci_loader.py
|  └── __init__.py
├── data/
|  ├── models/
|  |  └── uci_co_model.onnx
|  ├── plots/
|  |  ├── latest_aqi.png
|  |  ├── uci_actual_vs_pred.png
|  |  └── uci_prediction_error_hist.png
|  └── uci/
|     └── aqi_history.sqlite
├── .env
├── .gitignore
├── LICENSE
├── README.md
└── requirements.txt
```

توضیحات مربوط به فولدرها و هر فایل:

برای آشنایی بیشتر با ساختار پروژه، خوب است نگاهی به مسئولیتی که هر فولدر و هر فایل بر عهده دارد هم بی‌اندازیم:

/src

این فولدر شامل تمام کدهای مربوط به پروژه است که مسئولیت‌های مختلف مانند آموزش مدل‌ها، جمع‌آوری داده‌ها، ذخیره‌سازی اطلاعات، و ویژوالیزیشن را بر عهده دارند.

/ml

این پوشه اصلی‌ترین کدها و منطق مدل‌های یادگیری ماشین را شامل می‌شود.

train_aqicn_model.py: این فایل مسئول آموزش مدل برای دیتاست AQICN است. در این فایل، داده‌های مربوط به کیفیت هوا از API دریافت شده و مدل آموزش داده می‌شود.

train_uci_model.py: مشابه فایل قبلی، این فایل برای آموزش مدل روی دیتاست UCI (Air Quality) طراحی شده است. از این مدل برای پیش‌بینی CO استفاده می‌شود.

/Pipeline

فولدر pipeline/ شامل فایل‌های مربوط به جریان کاری (pipeline) است که مسئولیت اجرای کلی پروژه را بر عهده دارند.

aqicn_runner.py: این فایل مسئول اجرای پایپلاین AQICN است. داده‌ها از API گرفته می‌شوند، در دیتابیس ذخیره می‌شوند و سپس گراف‌های مربوط به پیش‌بینی‌ها تولید می‌شوند.

collector.py: این فایل مسئول جمع‌آوری داده‌ها از API AQICN و ذخیره‌سازی آن‌ها در SQLite است. در این فایل از توکن API برای ارتباط با پلتفرم AQICN استفاده می‌شود.

uci_runner.py: مشابه فایل aqicn_runner.py، این فایل مربوط به اجرای پایپلاین UCI است. در این فایل مدل روی دیتاست UCI آموزش داده می‌شود و پیش‌بینی‌ها ذخیره می‌شوند.

/storage

این فولدر مربوط به ذخیره‌سازی داده‌ها است.

sqlite_storage.py: این فایل مدیریت پایگاه داده SQLite را انجام می‌دهد. داده‌های AQICN و UCI در این پایگاه ذخیره می‌شوند.

/tests

فولدر تست‌ها که برای اطمینان از صحت عملکرد بخش‌های مختلف پروژه استفاده می‌شود. البته فقط برای دیتاست aqicn کد تست نوشتیم، چون از دریافت api مطمئن نبودیم و فکر می‌کردیم شاید کدها مشکلی دارند.

test_aqicn_token.py: این فایل برای تست صحت توکن API AQICN طراحی شده است. به طور خاص، این فایل بررسی می‌کند که آیا توکن به درستی تنظیم شده است یا خیر.

/visualization

فولدر ویژوالیزیشن برای تولید گراف‌ها و نمودارها است.

plots.py: این فایل برای تولید نمودارهای مختلف از داده‌ها استفاده می‌شود، مانند نمودار بار برای AQI و هیستوگرام خطای پیش‌بینی.

uci_plots.py: این فایل مختص به تولید نمودارهایی برای دیتاست UCI است.

/config

این فولدر شامل فایل‌های پیکربندی است.

settings.py: در این فایل تنظیمات پروژه مانند مسیر فایل‌ها و توکن‌های API ذخیره می‌شود. این فایل به‌طور خودکار از فایل .env پیکربندی‌ها را بارگذاری می‌کند.

/data_loader

فولدری برای بارگذاری داده‌ها و پردازش اولیه آن‌ها.

aqi_api_client.py: این فایل مسئول ارتباط با API AQICN است و داده‌ها را از پلتفرم AQICN برای هر شهر دریافت می‌کند.

uci_loader.py: این فایل وظیفه بارگذاری داده‌های دیتاست UCI را بر عهده دارد.

__init__.py: در حالت کلی، این فایل‌ها برای تبدیل فولدرها به ماژول‌های پایتون استفاده می‌شوند. این فایل‌ها به پایتون اعلام می‌کنند که این فولدرها باید به عنوان ماژول‌های پایتون شناسایی شوند.

src/main.py: این فایل برای مشخص کردن مود پروژه است، اینکه پروژه از کدام دیتاست اجرا شود. در حالت دیفالت اگر دستور python -m src.main را در ترمینال بزنیم، مسیر دیتاست uci اجرا خواهد شد.

/data

فولدری که تمام داده‌های مورد استفاده در پروژه در آن قرار دارد.

/models

این فولدر برای ذخیره فایل‌های onnx است.

uci_co_model.onnx: مدل آموزش داده‌شده با استفاده از داده‌های UCI که به فرمت ONNX صادر شده است. فایل ONNX یک فرمت استاندارد برای ذخیره و جابه‌جایی مدل‌های یادگیری ماشین بین فریم ورک‌های مختلف مثل PyTorch و TensorFlow است.

/plots

این فولدر برای ذخیره تصاویر و خروجی‌های حاصل از ران کردن است.

latest_aqi.png: نمودار آخرین پیش‌بینی‌های AQI.

uci_actual_vs_pred.png: گراف مربوط به پیش‌بینی‌های مدل و مقایسه با داده‌های واقعی.

uci_prediction_error_hist.png: هیستوگرام خطای پیش‌بینی مدل UCI.

/uci

این فولدر برای ذخیره پایگاه داده مربوط به کیفیت هوا است.

aqi_history.sqlite: پایگاه داده SQLite که داده‌های مربوط به کیفیت هوا ذخیره می‌شود.

فایل‌های دیگر:

.env: این فایل برای ذخیره توکن‌های API و سایر تنظیمات محیطی استفاده می‌شود.

.gitignore: برای نادیده گرفتن فایل‌های غیر ضروری در گیت (مانند فایل‌های متنی یا داده‌هایی که نباید در کنترل نسخه قرار گیرند).

LICENSE: شرایط استفاده از پروژه و حقوق قانونی آن.

README.md: توضیحات و مستندات پروژه در گیت هاب.

requirements.txt: لیست وابستگی‌های پروژه که با استفاده از آن می‌توان تمام پکیج‌های مورد نیاز پروژه را نصب کرد.

فایل main.py

فایل main.py به عنوان بخش اصلی و کنترل‌کننده اجرای پروژه عمل می‌کند و این امکان را می‌دهد که بین دو حالت اجرایی مختلف (مدل UCI یا مدل AQICN) انتخاب کرده و آن‌ها را به طور خودکار اجرا کنیم.

1. بارگذاری تنظیمات: در ابتدا، تنظیمات پروژه با استفاده از تابع load_settings از فایل settings.py بارگذاری می‌شود. این تنظیمات شامل اطلاعات مختلفی مانند مسیر فایل‌های داده‌ها، مدل‌ها و گراف‌ها است که در ادامه در مراحل مختلف استفاده خواهند شد.
2. ساختار ورودی پارامترها (Argument Parsing): برای مدیریت ورودی‌های خط فرمان، از argparse استفاده می‌شود. این ابزار به ما اجازه می‌دهد تا پارامترهایی را برای انتخاب حالت اجرایی وارد کنیم. در این کد، از پارامتر --mode استفاده شده که به ما این امکان را می‌دهد که انتخاب کنیم مدل uci یا مدل aqicn اجرا شود. به طور پیش‌فرض، این پارامتر بر روی حالت uci تنظیم شده است، که به این معناست که اگر هیچ پارامتری وارد نشود، حالت uci اجرا می‌شود.
3. انتخاب حالت اجرایی و اجرای پایپلاین: پس از اینکه ورودی‌های لازم از خط فرمان دریافت شد، اگر حالت uci انتخاب شود، تابع run_uci_pipeline فراخوانی می‌شود که مربوط به اجرای مدل با دیتاست UCI است. در این حالت، مسیر فایل داده‌ها (uci_csv_path)، مدل ذخیره‌شده به فرمت ONNX و مسیر برای ذخیره نمودارها به این تابع داده می‌شود. اگر به جای آن حالت aqicn انتخاب شود، تابع run_aqicn_pipeline فراخوانی می‌شود که برای دریافت و پردازش داده‌های کیفیت هوای زمان واقعی از سیستم AQICN طراحی شده است. این تابع با استفاده از توکن API و اطلاعات مربوط به شهرها و پایگاه داده اطلاعات مربوطه به پردازش داده‌ها می‌پردازد.
4. مدیریت خطاها: برای اطمینان از اجرای صحیح برنامه و گزارش خطاهای احتمالی، یک بلوک try-except در نظر گرفته شده است. اگر در حین اجرای هرکدام از پایپلاین‌ها خطایی پیش آید، این خطا در لاگ ثبت شده و به کاربر نمایش داده می‌شود.

این کد به طور کلی دو حالت اجرایی برای مدل‌های مختلف فراهم می‌کند و با استفاده از ورودی‌های پارامتر خط فرمان، می‌توان به صورت انعطاف‌پذیر مدل‌های مختلف را اجرا کرد. به طور خاص، این فایل به عنوان کنترل‌کننده اصلی پروژه عمل می‌کند و به شما این امکان را می‌دهد که بسته به نیاز خود از داده‌های UCI یا داده‌های AQICN برای پیش‌بینی کیفیت هوا استفاده کنید.

UCI Air Quality Dataset

دیتاست UCI Air Quality شامل داده‌های مربوط به کیفیت هوای شهرهای مختلف است که برای پیش‌بینی و تحلیل آلودگی هوا استفاده می‌شود. این دیتاست از ایستگاه‌های سنجش کیفیت هوا در شهر پادوا در ایتالیا جمع‌آوری شده است و شامل اطلاعاتی از قبیل غلظت گازها (CO، NO2، O3 و SO2) و سایر آلاینده‌ها است. هدف اصلی استفاده از این دیتاست در پروژه، پیش‌بینی غلظت CO (گاز مونوکسید کربن) بر اساس ویژگی‌های مختلف است. این داده‌ها برای تحلیل کیفیت هوا و ارائه راهکارهایی جهت بهبود وضعیت محیط زیست مورد استفاده قرار میگیرند.

برای آموزش مدل روی این دیتاست از روش رگرسیون خطی استفاده شده است. رگرسیون خطی یک مدل ساده و کارآمد برای پیش‌بینی مقدار یک ویژگی (در اینجا، غلظت CO) بر اساس ویژگی‌های دیگر است. خروجی مدل، پیش‌بینی‌هایی خواهد بود که نشان می‌دهد غلظت CO در آینده یا در شرایط مختلف محیطی چقدر خواهد بود. این پیش‌بینی‌ها به متخصصین محیط زیست کمک می‌کنند تا وضعیت کیفیت هوا را بهتر درک کرده و تصمیمات بهتری برای کاهش آلودگی هوا اتخاذ کنند.

فایل AirQualityUCI.csv (داده‌ها)

فایل AirQualityUCI.csv شامل داده‌های مربوط به کیفیت هوا در شهر پادوا، ایتالیا است. این داده‌ها از ایستگاه‌های سنجش کیفیت هوا جمع‌آوری شده‌اند و شامل ویژگی‌هایی مانند غلظت گازهای آلاینده (CO، NO2، O3، SO2) و آلودگی‌ها است. علاوه بر این، داده‌ها شامل تاریخ و زمان هر اندازه‌گیری نیز هستند. هدف از این داده‌ها پیش‌بینی غلظت گاز CO بر اساس ویژگی‌های دیگر است که برای آموزش مدل‌ها در پروژه استفاده می‌شوند.

فایل settings.py (تنظیمات پروژه)

در کدی که نوشته شده است، ما به طراحی و پیاده‌سازی تنظیمات پروژه پرداخته‌ایم که به طور کلی مسئول تعیین مسیرهای مربوط به داده‌ها، مدل‌ها و گراف‌ها است و همچنین از محیط‌های مختلف برای بارگذاری توکن‌های API و دیگر تنظیمات پروژه استفاده می‌کند.

اولین بخش کد، شامل یک کلاس به نام Settings است که به طور خاص برای ذخیره و مدیریت تنظیمات مختلف پروژه طراحی شده است. این کلاس از dataclass استفاده می‌کند که به صورت خودکار قابلیت‌های ذخیره‌سازی داده‌ها را فراهم می‌آورد. در این کلاس، چهار مسیر اصلی پروژه (project_root، data_dir، plots_dir، models_dir) تعریف شده است که نشان‌دهنده مسیرهای اصلی فایل‌ها و مدل‌ها در پروژه هستند. همچنین، یک متغیر uci_csv_path برای مسیر فایل CSV داده‌های UCI به‌عنوان منبع داده پیش‌بینی کیفیت هوا و یک متغیر aqicn_api_token برای توکن ارتباط با API AQICN نیز در

نظر گرفته شده است. این توکن به طور اختیاری مقداردهی میشود، البته در صورتی که توکن از محیط بارگذاری شده باشد.

در قسمت دوم کد، تابع load_settings پیاده‌سازی شده است. این تابع به طور خودکار از فایل .env برای بارگذاری متغیرهای محیطی استفاده می‌کند. سپس مسیرهای اصلی پروژه (مانند مسیرهای داده‌ها، مدل‌ها و گراف‌ها) را تنظیم کرده و پوشه‌های مربوطه را در صورتی که وجود نداشته باشند، ایجاد می‌کند. این عملیات به طور خودکار انجام می‌شود تا از ایجاد اشتباهات و نیاز به تنظیمات دستی جلوگیری شود. پس از آن، مسیر فایل داده‌های UCI (در اینجا AirQualityUCI.csv) به صورت دقیق تنظیم می‌شود. در این مرحله همچنین توکن API از محیط بارگذاری می‌شود. اگر توکن وجود داشته باشد، از آن برای ارتباط با AQICN استفاده خواهد شد. در نهایت، یک شیء از کلاس Settings ایجاد می‌شود که تمام این اطلاعات را در خود نگه می‌دارد و به عنوان تنظیمات نهایی پروژه در دسترس قرار می‌دهد.

این پیاده‌سازی، ساختاری را فراهم می‌کند که پروژه به راحتی قابلیت تغییر مسیرها و تنظیمات مربوط به داده‌ها و مدل‌ها را داشته باشد و به طور خودکار از محیط‌های متغیر برای بارگذاری تنظیمات استفاده کند. همچنین، مدیریت توکن‌ها و دسترسی به APIهای خارجی به صورت ساده و مقیاس‌پذیر انجام می‌شود.

فایل train_uci_model.py (آموزش مدل)

در فایل train_uci_model.py، ما یک مدل رگرسیون خطی برای پیش بینی غلظت مونوکسید کربن (CO) از داده‌های دیتاست UCI آموزش می‌دهیم و آن را به فرمت ONNX صادر می‌کنیم تا بتوانیم از مدل در دیگر فریم ورک‌ها یا پلتفرم‌ها استفاده کنیم. این کد شامل چندین مرحله است که در ادامه به طور کامل توضیح داده می‌شود.

در ابتدا، داده‌ها از فایل CSV که مسیر آن به عنوان ورودی به تابع train_and_export_uci_model داده شده است، با استفاده از pandas بارگذاری می‌شوند. داده‌ها با استفاده از جداکننده (sep=";") و نقطه به عنوان اعشار (decimal=",") خوانده می‌شوند. سپس دو ستون از داده‌ها که شامل غلظت CO هستند ("CO(GT)" و "PT08.S1(CO)") به نوع عددی تبدیل می‌شوند. در اینجا از متد to_numeric استفاده شده است که هرگونه مقدار غیرمجاز را با مقدار NaN جایگزین می‌کند و سپس از dropna برای حذف ردیف‌هایی که دارای مقدار NaN در این دو ستون هستند، استفاده می‌شود.

پس از پردازش داده‌ها، ویژگی (X) که در اینجا تنها ستون "PT08.S1(CO)" است، به عنوان ورودی مدل و هدف (y) که مقدار غلظت CO است، مشخص می‌شود. در این مرحله، داده‌ها به دو بخش آموزشی و تست تقسیم می‌شوند. این کار با استفاده از تابع train_test_split از کتابخانه sklearn.model_selection انجام می‌شود. 80 درصد داده‌ها برای آموزش مدل و 20 درصد باقیمانده برای ارزیابی مدل انتخاب می‌شود.

سپس مدل رگرسیون خطی از کتابخانه sklearn.linear_model ساخته می‌شود و با داده‌های آموزشی (X_train, y_train) آموزش داده می‌شود. بعد از آموزش، مدل برای پیش بینی بر اساس داده‌های تست (X_test) استفاده می‌شود و پیش‌بینی‌ها در متغیر preds ذخیره می شوند.

برای ارزیابی دقت مدل، از معیار mean_absolute_error استفاده می‌شود که اختلاف میان پیش‌بینی‌ها و مقادیر واقعی را محاسبه می‌کند. مقدار MAE که نشان‌دهنده خطای میانگین مطلق است، چاپ می‌شود تا مشخص شود مدل چقدر دقیق عمل کرده است.

در نهایت، مدل رگرسیون خطی به فرمت ONNX تبدیل می‌شود تا قابل استفاده در دیگر فریم ورک‌ها و ابزارهای یادگیری ماشین باشد. این تبدیل با استفاده از کتابخانه skl2onnx و تابع convert_sklearn انجام می‌شود. در اینجا، نوع ورودی مدل (initial_type) به طور دستی تعریف شده است که نشان می‌دهد ورودی مدل یک عدد اعشاری (float) است. سپس مدل به صورت باینری صادر شده و در مسیر مشخص‌شده توسط out_path ذخیره می‌شود. قبل از ذخیره مدل، پوشه‌ای که مدل در آن قرار می‌گیرد، در صورتی که وجود نداشته باشد، ایجاد می‌شود. با این اوصاف، این کد مدلی آموزش‌دادشده است که برای پیش‌بینی غلظت CO در آینده بر اساس ویژگی‌های داده‌های موجود استفاده خواهد شد و به فرمت ONNX صادر می‌شود تا در دیگر پلتفرم‌ها و محیطهای مختلف قابل استفاده باشد.

فایل uci_runner.py (اجرای پایپلاین)

در فایل uci_runner.py، هدف اجرای کامل پایپلاین پیش‌بینی کیفیت هوای UCI است. این پایپلاین شامل مراحل مختلف از جمله بارگذاری داده‌ها، پیش‌پردازش، آموزش مدل، ارزیابی مدل، صادرات مدل به فرمت ONNX، پیش‌بینی با مدل ONNX، و تولید نمودارهای گرافیکی برای تحلیل نتایج است. در ادامه، این کد به طور کامل توضیح داده شده است.

اولین بخش کد، وارد کردن کتابخانه‌ها و تنظیمات اولیه است. اینجا از logging برای ثبت لاگ‌ها استفاده می‌شود تا بتوان روند اجرای برنامه را پیگیری کرد. همچنین، از کتابخانه onnxruntime برای بارگذاری و استفاده از مدل ONNX و از sklearn برای آموزش مدل رگرسیون خطی و ارزیابی آن استفاده می‌شود. توابع مربوط به بارگذاری داده‌ها و تولید نمودارها نیز از فایل‌های دیگر پروژه وارد شده‌اند. در ابتدای تابع run_uci_pipeline، ابتدا چک می‌شود که فایل داده‌های UCI در مسیر مشخص‌شده وجود دارد یا خیر. اگر فایل وجود نداشته باشد، یک خطای FileNotFoundError ایجاد می‌شود. سپس در مرحله بعدی، داده‌ها از فایل CSV بارگذاری شده و برای پیش‌بینی غلظت CO آماده می‌شوند. برای این کار از دو تابع load_uci_air_quality و preprocess_uci_for_co_regression استفاده می‌شود که به ترتیب داده‌ها را بارگذاری و برای پیش‌بینی CO آماده می‌کنند.

پس از پیش‌پردازش داده‌ها، ویژگی‌ها (X) و هدف (y) برای آموزش مدل انتخاب می‌شوند. در اینجا، ویژگی مدل تنها شامل ستون "PT08.S1(CO)" است و هدف، ستون "CO(GT)" می‌باشد. سپس داده‌ها به دو بخش آموزشی و تست تقسیم می‌شوند. این تقسیم‌بندی با استفاده از تابع train_test_split انجام می‌شود که 80 درصد داده‌ها برای آموزش مدل و 20 درصد برای ارزیابی مدل اختصاص می‌یابد.

در مرحله بعد، مدل رگرسیون خطی از کتابخانه sklearn.linear_model ایجاد می‌شود و بر روی داده‌های آموزشی آموزش داده می‌شود. پس از آموزش، مدل برای پیش‌بینی بر روی داده‌های تست استفاده می‌شود و نتایج پیش‌بینی در متغیر sk_preds ذخیره می‌شود. سپس خطای مدل با استفاده از معیار mean_absolute_error محاسبه می‌شود و در لاگ ثبت می‌شود. پس از ارزیابی مدل، مرحله بعدی صادرات مدل به فرمت ONNX است. برای این کار از کتابخانه skl2onnx استفاده می‌شود. مدل رگرسیون خطی به فرمت ONNX تبدیل می‌شود و در مسیر مشخص‌شده ذخیره می‌شود. این مرحله برای استفاده از مدل در دیگر فریمورک‌ها و پلتفرم‌ها مفید است.

در ادامه، مدل ONNX با استفاده از onnxruntime بارگذاری می‌شود و برای پیش‌بینی بر روی داده‌های تست استفاده می‌شود. پیش‌بینی‌ها در متغیر onnx_preds ذخیره می‌شوند و خطای مدل ONNX نیز با استفاده از همان معیار mean_absolute_error محاسبه می‌شود و در لاگ ثبت می‌شود.

در نهایت، دو نمودار برای تحلیل نتایج تولید می‌شود. اولین نمودار مقایسه‌ای بین مقادیر واقعی و پیش‌بینی‌شده است که خطای مدل و خط y=x را نیز نمایش می‌دهد. دومین نمودار، هیستوگرام خطای پیش‌بینی مدل است که توزیع خطاهای مدل را نشان می‌دهد. هر دو نمودار در مسیر مشخص‌شده ذخیره می‌شوند و در لاگ ثبت می‌شود.

این فایل به طور کامل یک پایپلاین پیش‌بینی را از ابتدا تا انتها اجرا می‌کند، مدل را آموزش می‌دهد، آن را به فرمت ONNX صادر می‌کند، و نتایج را به صورت گرافیکی نمایش می‌دهد تا بتوان تحلیل دقیقی از عملکرد مدل داشت.

فایل uci_plots.py (ویژوالیزیشن نتایج)

در فایل uci_plots.py، هدف تولید نمودارهای ویژوالیزیشن برای نتایج پیش‌بینی مدل است. این فایل شامل دو تابع اصلی است که برای تحلیل و نمایش نتایج پیش‌بینی به صورت تصویری طراحی شده‌اند: یک تابع برای نمایش مقایسه واقعی و پیش‌بینی‌شده و دیگری برای نمایش توزیع خطاهای پیش‌بینی.

اولین تابع، plot_actual_vs_predicted است که نمودار پخش (scatter plot) مقادیر واقعی و پیش‌بینی شده را به همراه خط مرجع y=x رسم می کند. این نمودار به ما کمک می‌کند تا تفاوت بین مقادیر واقعی و پیش‌بینی‌شده مدل را به صورت بصری مشاهده کنیم. ابتدا، داده‌های ورودی (y_true و y_pred) به صورت آرایه‌های یک بعدی در می‌آیند تا به‌طور ساده‌تری پردازش شوند. سپس، کمینه و بیشینه مقادیر

واقعی و پیش‌بینی‌شده محاسبه می‌شود تا محدوده مقادیر روی محورهای X و Y مشخص شود. در ادامه، نمودار پخش رسم می‌شود که در آن مقادیر واقعی در محور X و مقادیر پیش‌بینی‌شده در محور Y قرار می‌گیرند. همچنین، یک خط مرجع با معادله y = x (که نشان‌دهنده پیش‌بینی کاملا درست است) رسم می‌شود. اگر مقدار mae (خطای میانگین مطلق) وارد شود، این مقدار نیز در عنوان نمودار نمایش داده می‌شود. در نهایت، نمودار ذخیره می‌شود و پنجره گرافیکی بسته می‌شود.

دومین تابع، plot_error_histogram است که هیستوگرام خطاهای پیش‌بینی را رسم می‌کند. در اینجا، خطاهای پیش‌بینی با تفاضل بین مقادیر پیش‌بینی‌شده و واقعی محاسبه می‌شوند. سپس این خطاها در یک هیستوگرام نمایش داده می‌شوند تا توزیع خطاهای مدل به‌صورت گرافیکی قابل مشاهده باشد. در این تابع، تعداد bins (بازه‌های هیستوگرام) به صورت پیش‌فرض 40 است که می‌تواند برای دقت بیشتر یا کمتر تغییر کند. مشابه تابع اول، نمودار در مسیر مشخص‌شده ذخیره می‌شود و پنجره گرافیکی بسته می‌شود.

در هر دو تابع، از کتابخانه matplotlib.pyplot برای ایجاد و ذخیره نمودارها استفاده می‌شود. همچنین، برای هر نمودار قبل از ذخیره‌سازی، پوشه مقصد در صورتی که وجود نداشته باشد، ایجاد می‌شود. این فایل برای تحلیل و نمایش نتایج پیش‌بینی به‌طور بصری طراحی شده است و به محققین و تحلیلگران کمک می‌کند تا عملکرد مدل را از طریق نمودارهای مختلف ارزیابی کنند.

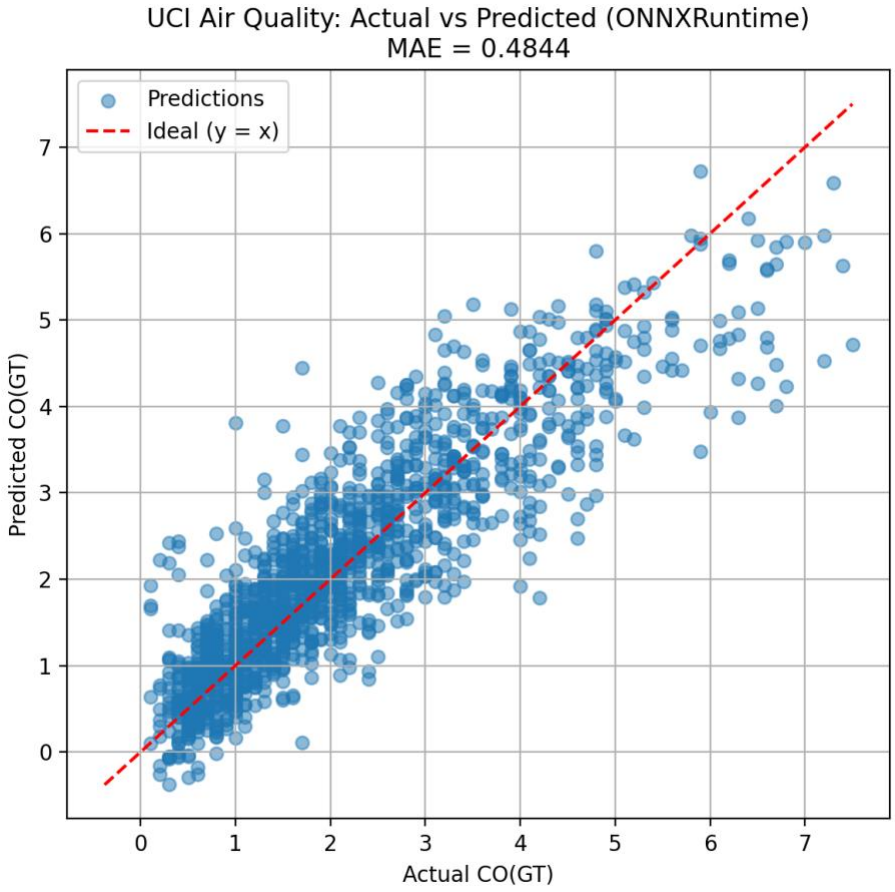
فایل uci_co_model.onnx (مدل ذخیره‌شده)

فایل uci_co_model.onnx در واقع یک مدل ذخیره‌شده به فرمت ONNX است که برای استفاده در دیگر پلتفرم‌ها و فریم ورک‌های یادگیری ماشین طراحی شده است. این فرمت به طور خاص برای انتقال مدل‌ها بین فریم‌ورک‌های مختلف مانند PyTorch و TensorFlow استفاده می‌شود. به این ترتیب، این فایل باید حاوی اطلاعات مربوط به معماری مدل، وزن‌ها و سایر پارامترهای آموزش‌داده‌شده باشد.

به احتمال زیاد شما هم مثل ما نمی‌توانید محتوای آن را باز کنید، اما چرا؟ چون اولا، فایل به صورت باینری ذخیره شده است و به طور مستقیم نمی‌توان آن را باز کرد مانند فایل‌های متنی. برای مشاهده مدل، باید از ابزارهایی مثل onnxruntime یا Netron استفاده کنید که قادرند مدل‌های ONNX را بارگذاری و نمایش دهند. دلیل دیگر این است که فرایند ذخیره‌سازی مدل با مشکل مواجه شده است یا در زمان ذخیره‌سازی به درستی عمل نکرده است. در چنین شرایطی، ممکن است فایل به طور کامل یا به درستی تولید نشده باشد و این باعث شود که نتوانید آن را به طور عادی مشاهده کنید. برای حل این مشکل، می‌توانید فایل را مجددا تولید کنید یا بررسی کنید که در فرآیند تبدیل مدل به ONNX مشکلی وجود نداشته باشد.

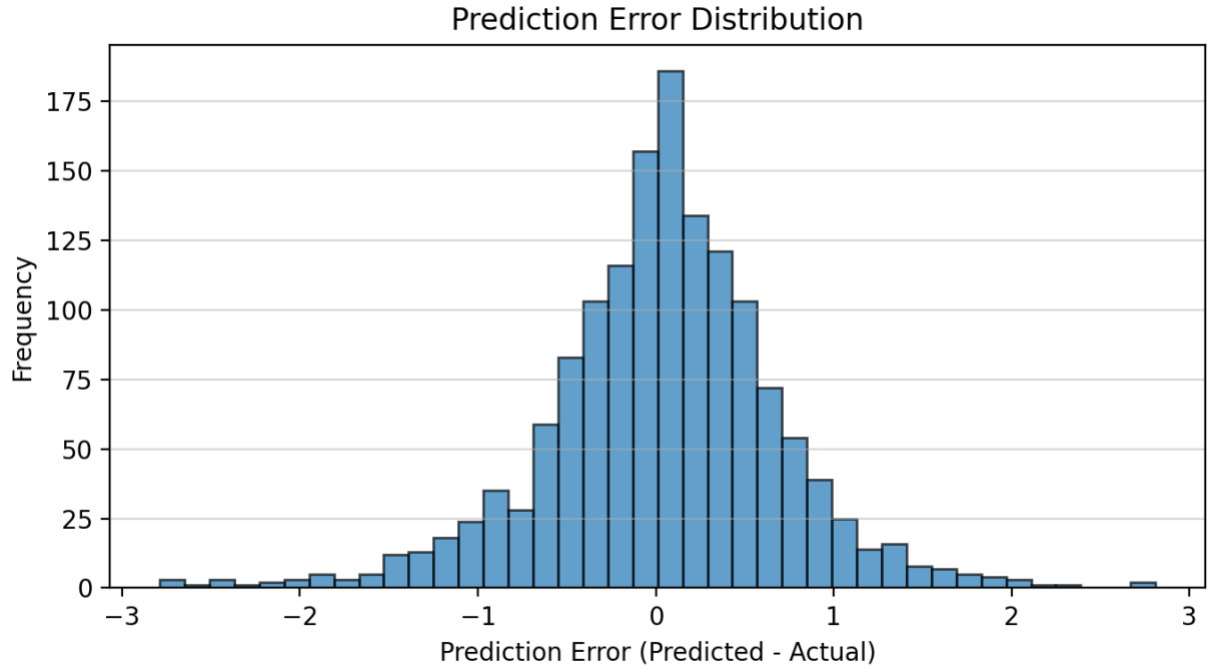
نمودار uci_actual_vs_pred.png (نمودار Actual vs Predicted)

نمودار اول که uci_actual_vs_pred.png نام دارد، مقایسه‌ای بین مقادیر واقعی و پیش‌بینی‌شده CO (مونوکسید کربن) را نشان می‌دهد. در این نمودار، محور افقی (X) مقادیر واقعی غلظت CO (از داده‌های تست) را نشان می دهد و محور عمودی (Y) مقادیر پیش‌بینی‌شده از مدل را نشان می‌دهد. هر نقطه در این نمودار نشان‌دهنده یک پیش‌بینی است. خط قرمز نقطه‌چین که با نام "Ideal (y = x)" مشخص شده، خط مرجع است که نشان می‌دهد اگر پیش‌بینی‌ها دقیقا با مقادیر واقعی برابر باشند، نقاط باید روی این خط قرار گیرند. از این نمودار می‌توان دید که به طور کلی مدل پیش‌بینی‌ها را به خوبی انجام داده است، چرا که بیشتر نقاط نزدیک به خط مرجع قرار دارند. همچنین، مقدار MAE = 0.4844 که در عنوان نمودار ذکر شده، خطای میانگین مطلق مدل را نشان می‌دهد و نشان دهنده این است که مدل به طور متوسط چقدر از مقادیر واقعی فاصله دارد.



نمودار uci_prediction_error_hist.png (هیستوگرام خطای پیش‌بینی)

نمودار دوم که uci_prediction_error_hist.png نام دارد، هیستوگرام توزیع خطاهای پیش‌بینی مدل را نمایش می‌دهد. در این نمودار، محور افقی (X) نشان‌دهنده خطاهای پیش‌بینی (تفاضل بین مقادیر پیش‌بینی‌شده و واقعی) است و محور عمودی (Y) نشان‌دهنده تعداد دفعاتی است که هر خطا تکرار شده است. این هیستوگرام به وضوح نشان می‌دهد که بیشتر خطاها در نزدیکی صفر قرار دارند، یعنی مدل بیشتر پیش‌بینی‌ها را با دقت بالا انجام داده است. به طور خاص، شکل توزیع نشان‌دهنده یک توزیع نرمال است، که نشان می‌دهد مدل به طور کلی عملکرد خوبی داشته و خطاهای پیش‌بینی در بیشتر موارد کم بوده‌اند.



به صورت کلی این دو نمودار وضعیت پیش‌بینی مدل را از جنبه‌های مختلف بررسی کرده و به ما کمک می‌کنند تا کیفیت مدل را به صورت بصری ارزیابی کنیم.

خروجی نهایی حاصل از دیتاست UCI

این خروجی که شما مشاهده می‌کنید، از اجرای دستور python -m src.main --mode uci به دست آمده و شامل مراحل مختلف اجرای پایپلاین پیش‌بینی کیفیت هوای مدل UCI است. در اینجا، هر بخش از خروجی به‌طور خلاصه و شفاف توضیح داده می‌شود:

1. UCI MAE (sklearn): این بخش نشان‌دهنده میزان خطای مدل رگرسیون خطی است که با استفاده از داده‌های UCI آموزش دیده است. مقدار MAE (Mean Absolute Error) برابر با 0.4844 است. این یعنی مدل به طور متوسط 0.4844 واحد از مقادیر واقعی CO (مونوکسید کربن) خطا دارد. این عدد هرچه کوچکتر باشد، مدل دقیق‌تر است.

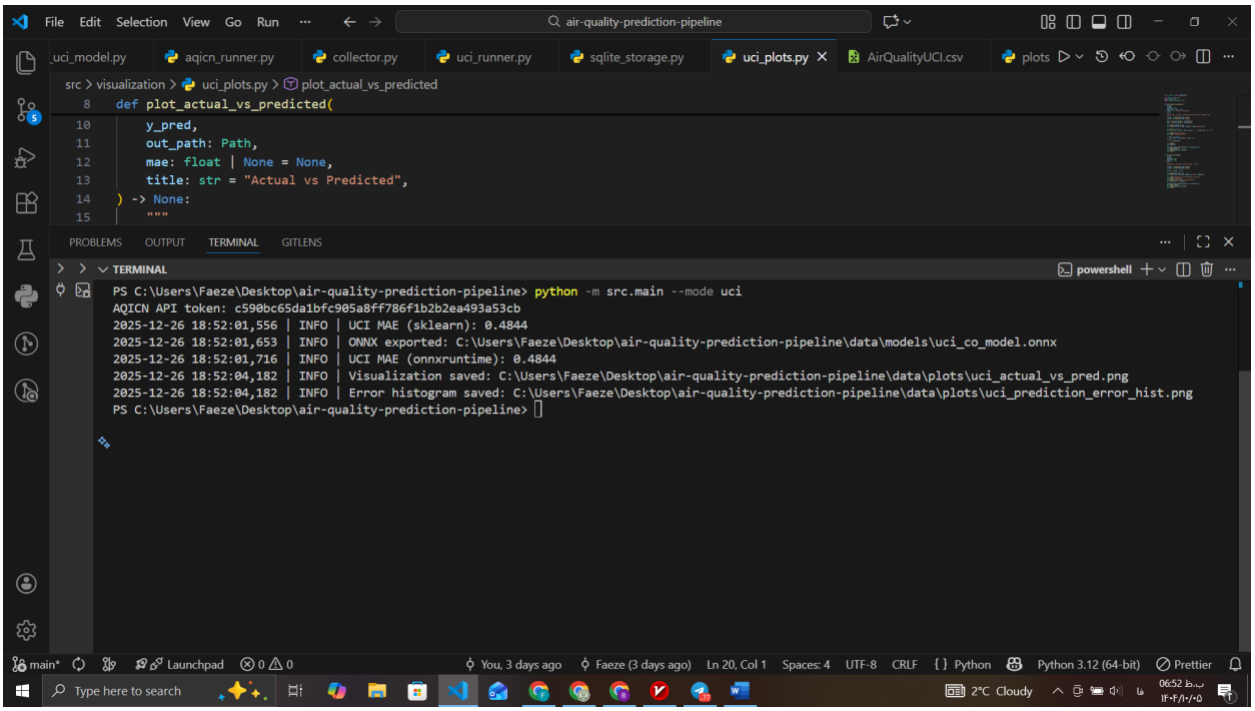
2. ONNX Exported: در این مرحله، مدل آموزش‌داده‌شده به فرمت ONNX صادر می‌شود. این مدل به طور باینری ذخیره می‌شود تا در پلتفرم‌های مختلف و برای پیش‌بینی‌های آینده استفاده شود. مسیر ذخیره مدل در این قسمت نمایش داده می‌شود.

3. UCI MAE (onnxruntime): این بخش به ارزیابی مدل ONNX پس از تبدیل به این فرمت می‌پردازد. مدل ONNX با استفاده از کتابخانه onnxruntime بارگذاری و ارزیابی شده و خطای MAE آن برابر با 0.4844 است که مشابه مدل اولیه است. این نشان‌دهنده عملکرد خوب مدل بعد از تبدیل به فرمت ONNX است.

4. Visualization saved: این بخش مسیر ذخیره‌سازی نمودار "Actual vs Predicted" را نشان می‌دهد که در آن مقادیر واقعی و پیش‌بینی‌شده CO برای داده‌های تست به صورت تصویری مقایسه شده‌اند. این نمودار به‌طور کلی نشان‌دهنده میزان تطابق پیش‌بینی‌ها با مقادیر واقعی است.

5. Error histogram saved: در نهایت، هیستوگرام خطاهای پیش‌بینی ذخیره می‌شود. این هیستوگرام توزیع خطاهای پیش‌بینی را نشان می‌دهد که می‌تواند به‌طور واضح عملکرد مدل را در پیش‌بینی مقادیر مختلف نشان دهد.

در مجموع، این خروجی نشان می‌دهد که مدل آموزش‌داده‌شده با داده‌های UCI، پس از آموزش، تبدیل به فرمت ONNX شده و از آن برای پیش‌بینی استفاده شده است. همچنین، خطای مدل (MAE) در هر دو مرحله (استفاده از مدل sklearn و مدل ONNX) مشابه است، که نشان‌دهنده عملکرد خوب مدل است. در نهایت، نمودارهای مربوط به مقایسه واقعی و پیش‌بینی‌شده و توزیع خطاهای پیش‌بینی تولید شده‌اند و ذخیره شده‌اند.


File Edit Selection View Go Run
air-quality-prediction-pipeline
uci_model.py
aqicn_runner.py
collector.py
uci_runner.py
sqlite_storage.py
uci_plots.py
AirQualityUCI.csv
plots
src > visualization > uci_plots.py > plot_actual_vs_predicted
def plot_actual_vs_predicted(
y_pred,
out_path: Path,
mae: float | None = None,
title: str = "Actual vs Predicted",
) -> None:
"""
PROBLEMS OUTPUT TERMINAL GITLENS
TERMINAL
powershell
PS C:\Users\Faeze\Desktop\air-quality-prediction-pipeline> python -m src.main --mode uci
AQICN API token: c590bc65da1bfc905a8ff786f1b2b2ea493a53cb
2025-12-26 18:52:01,556 | INFO | UCI MAE (sklearn): 0.4844
2025-12-26 18:52:01,653 | INFO | ONNX exported: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\models\uci_co_model.onnx
2025-12-26 18:52:01,716 | INFO | UCI MAE (onnxruntime): 0.4844
2025-12-26 18:52:04,182 | INFO | Visualization saved: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\plots\uci_actual_vs_pred.png
2025-12-26 18:52:04,182 | INFO | Error histogram saved: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\plots\uci_prediction_error_hist.png
PS C:\Users\Faeze\Desktop\air-quality-prediction-pipeline>
main* Launchpad 0 0 You, 3 days ago Faeze (3 days ago) Ln 20, Col 1 Spaces: 4 UTF-8 CRLF { } Python Python 3.12 (64-bit) Prettier
Type here to search
2°C Cloudy

توضیح فایل‌های دیگر:

.gitignore: این فایل برای مشخص کردن فایل‌ها و دایرکتوری‌هایی است که نباید در سیستم کنترل نسخه گیت (Git) قرار بگیرند. به عبارت ساده‌تر، این فایل به گیت می‌گوید که کدام فایل‌ها و پوشه‌ها نباید تحت پیگیری نسخه قرار بگیرند، مانند فایل‌های موقتی، تنظیمات محیطی یا داده‌های شخصی. در پروژه ما فایل env. که به api دیتاست aqicn مرتبط است، در gitignore. قرار گرفته تا توکن عمومی نباشد و هر شخصی که نیاز به اجرای پروژه داشت، توکن مخصوص خود را از سایت AQICN دریافت کند.

README.md: این فایل برای ارائه توضیحات و مستندات پروژه استفاده می‌شود. در این فایل توضیحاتی درباره هدف پروژه، نحوه استفاده، نصب پیش‌نیازها، روش اجرای برنامه و هر اطلاعات مفید دیگر را قرار داده‌ایم که به کاربران و توسعه‌دهنده‌های به صورت جهانی کمک می‌کند تا با پروژه آشنا شوند.

requirements.txt: این فایل لیستی از کتابخانه‌ها و نسخه‌های آن‌ها را شامل می‌شود که برای اجرای پروژه لازم هستند. زمانی که دیگران می‌خواهند پروژه را اجرا کنند، می‌توانند از این فایل برای نصب تمامی کتابخانه‌های مورد نیاز استفاده کنند. در ادامه، توضیحاتی در مورد هر کتابخانه آورده شده است:

کتابخانه‌ها در requirements.txt:

pandas: یک کتابخانه قدرتمند برای کار با داده‌ها است. این کتابخانه برای پردازش و تحلیل داده‌های جدولی مانند داده‌های CSV و Excel کاربرد دارد. از طرفی ابزارهایی برای عملیات‌هایی مثل مرتب‌سازی، گروه‌بندی و فیلتر کردن داده‌ها فراهم می‌کند.

numpy: این کتابخانه برای کار با آرایه‌ها و عملیات‌های ریاضی در پایتون استفاده می‌شود. در پروژه‌های علمی و مهندسی، numpy مخصوصا برای انجام محاسبات ماتریسی و ریاضیاتی سنگین بسیار مهم است.

matplotlib: یک کتابخانه برای ترسیم نمودارها و گراف‌ها در پایتون است. matplotlib به شما این امکان را می‌دهد که به راحتی داده‌ها را به صورت گرافیکی و تصویری (مانند نمودارهای خطی، میله‌ای و پراکندگی) نمایش دهید.

scikit-learn: این کتابخانه یکی از معروف‌ترین ابزارها برای یادگیری ماشین در پایتون است. scikit-learn شامل مجموعه‌ای از الگوریتم‌ها و ابزارها برای پیش‌پردازش داده‌ها، آموزش مدل‌های یادگیری ماشین و ارزیابی آن‌هاست. این کتابخانه شامل مدل‌هایی مانند رگرسیون، دسته‌بندی، خوشه‌بندی و کاهش ابعاد است.

onnxruntime: این کتابخانه برای اجرای مدل‌های یادگیری ماشین که به فرمت ONNX تبدیل شده‌اند استفاده می‌شود. onnxruntime امکان استفاده از مدل‌های ONNX را در پایتون فراهم می‌کند، مخصوصا برای انجام پیش‌بینی‌ها به‌صورت سریع و بهینه.

skl2onnx: این کتابخانه برای تبدیل مدل‌های آموزش‌داده‌شده با استفاده از scikit-learn به فرمت ONNX استفاده می‌شود. به طور ساده‌تر، این کتابخانه به شما اجازه می‌دهد تا مدل‌های ساخته‌شده با scikit-learn را برای استفاده در پلتفرم‌ها و فریم ورک‌های دیگر مانند onnxruntime صادر کنید.

requests: این کتابخانه برای ارسال درخواست‌های HTTP در پایتون استفاده می‌شود. به کمک requests می‌توان به راحتی داده‌ها را از APIها دریافت کرد یا به سرورها ارسال کرد. این کتابخانه برای کار با داده‌های ریل‌تایم از منابع اینترنتی (مثل دریافت داده‌های AQICN) بسیار مفید است.

python-dotenv: این کتابخانه برای بارگذاری متغیرهای محیطی از فایل‌های env. استفاده می‌شود. این مورد مخصوصا برای مدیریت توکن‌ها و اطلاعات حساس (مانند API keys) در پروژه‌ها مفید است. به کمک python-dotenv می‌توانید اطلاعات حساس را در فایل‌هایی جداگانه ذخیره کنید و به راحتی آن‌ها را در محیط برنامه‌نویسی بارگذاری کنید.

در حالت کلی این کتابخانه‌ها برای کار با داده‌ها، یادگیری ماشین، مدل‌های ONNX و درخواست‌های API در پروژه‌های پایتون استفاده می‌شوند.

AQICN Real-Time Air Quality Dataset

در پروژه‌ای که طراحی کرده‌ایم، هدف اصلی پیش‌بینی کیفیت هوا با استفاده از دیتاست UCI بود که به طور سنتی داده‌های تاریخی کیفیت هوا را در اختیار می‌گذارد. اما ما تصمیم گرفتیم که این پروژه را گسترش دهیم و به سمت استفاده از داده‌های لحظه‌ای حرکت کنیم. ایده این است که علاوه بر دیتاست تاریخی UCI، می‌توانیم داده‌های واقعی و لحظه‌ای از کیفیت هوای شهرهای مختلف ایران به طور مستقیم از پلتفرم AQICN دریافت کنیم. این موضوع به ما این امکان را می‌دهد که پروژه‌ای پویا و بروز داشته باشیم که نه تنها از داده‌های گذشته استفاده کند، بلکه قادر به پیش‌بینی و تحلیل وضعیت فعلی آلودگی هوا در زمان واقعی باشد.

بنابراین، ما یک پایپلاین اضافی برای این دیتاست طراحی کردیم که شامل اتصال به API AQICN است و به طور منظم داده‌ها را از این سرویس برای شهرهای ایران مانند تهران، اصفهان، مشهد و اهواز جمع‌آوری می‌کند. این داده‌ها به طور خودکار ذخیره شده و سپس برای تحلیل و پیش‌بینی‌ها به کار گرفته می‌شوند. این کار نه تنها قابلیت‌های پروژه را به سطح جدیدی ارتقا داد، بلکه به آن کمک کرد تا کاملا کاربردی و قابل استفاده در شرایط واقعی باشد، در حالی که امکان مشاهده وضعیت کیفیت هوای لحظه‌ای شهرهای مختلف را نیز فراهم کرد.

فایل aqi_api_client.py (دریافت داده از API)

در فایل aqi_api_client.py، ما یک کلاس به نام AQIAPIClient داریم که وظیفه دریافت داده‌ها از API AQICN (Air Quality Index China) را بر عهده دارد. این API اطلاعات مربوط به کیفیت هوا از شهرهای مختلف جهان را ارائه می‌دهد و این کلاس به طور خاص برای دریافت و پردازش این داده‌ها طراحی شده است.

کلاس AQIAPIClient شامل متدهای مختلفی است:

متغیر BASE_URL: در ابتدا، یک متغیر ثابت به نام BASE_URL تعریف می‌شود که آدرس اصلی API برای دریافت داده‌ها است. این URL برای اتصال به سرور و دریافت اطلاعات از API استفاده می‌شود.

متد __init__: این متد سازنده کلاس است که در ابتدا توکن API را تنظیم می‌کند. اگر توکن به صورت ورودی به متد داده شود، از آن استفاده می‌کند. در غیر این صورت، از فایل .env برای بارگذاری توکن از متغیر محیطی AQICN_API_TOKEN استفاده می‌کند. اگر هیچ توکنی یافت نشود، یک خطای ValueError صادر می‌شود تا به کاربر اطلاع دهد که باید توکن را تنظیم کند.

متد fetch_city_aqi: این متد مسئول دریافت داده‌های کیفیت هوا برای یک شهر خاص است. ابتدا، URL مورد نظر برای درخواست به API ساخته می‌شود و سپس از کتابخانه requests برای ارسال درخواست

به API استفاده می‌شود. اگر درخواست موفقیت‌آمیز باشد، داده‌های بازگشتی از API به فرمت JSON پردازش می‌شود. در صورتی که مشکلی در ارسال درخواست وجود داشته باشد (مثلا مشکل شبکه یا خطای API)، یک استثنا (RuntimeError) ایجاد می‌شود.

متد _parse_response: پس از دریافت پاسخ از API، این متد مسئول پردازش داده‌های خام (raw data) است. داده‌های مربوط به کیفیت هوا مانند مقادیر AQI (شاخص کیفیت هوا)، غلظت‌های مختلف آلاینده‌ها مانند pm25, pm10, co, no2, so2, o3 استخراج شده و در قالب یک دیکشنری ساختار یافته بازگشت داده می‌شود. همچنین، زمان دریافت داده‌ها نیز به‌صورت یک تاریخ و زمان UTC به فرمت ISO ذخیره می‌شود.

در کل، این کلاس به شما اجازه می‌دهد که با وارد کردن نام یک شهر، داده‌های مربوط به کیفیت هوا را از API AQICN دریافت کرده و اطلاعات مورد نظر مانند AQI و غلظت آلاینده‌ها را استخراج کنید. این داده‌ها برای تحلیل کیفیت هوا در هر شهر و انجام پیش‌بینی‌ها یا تحلیل‌های بیشتر بسیار مفید هستند.

فایل collector.py (جمع‌آوری داده‌ها)

فایل collector.py مسئول جمع‌آوری داده‌ها از API AQICN برای شهرهای مختلف ایران است. این فایل به طور کلی داده‌های کیفیت هوا را از API دریافت کرده، آن‌ها را پردازش کرده و سپس در قالب یک لیست از AQIRecord ذخیره می‌کند. در ادامه، به بررسی دقیق‌تر این فایل و عملکرد آن می‌پردازیم.

تعریف CollectorResult: این کلاس داده‌ها را در قالب دو لیست می‌گیرد؛ یکی برای ذخیره‌سازی رکوردهای AQI که شامل اطلاعات مربوط به کیفیت هوا در هر شهر است و دیگری برای ذخیره‌سازی خطاها که ممکن است در هنگام جمع‌آوری داده‌ها رخ دهد.

تابع _to_float: این تابع برای تبدیل داده‌های ورودی به نوع عدد اعشاری (float) طراحی شده است. برخی از داده‌ها ممکن است به‌طور نادرست به عنوان رشته‌ها یا مقادیر نامعتبر (مانند "-") ارسال شوند که این تابع وظیفه تبدیل آن‌ها به عدد اعشاری یا تهیه مقدار None در صورت خطا را دارد.

تابع collect_records: این تابع اصلی است که از آن برای جمع‌آوری داده‌ها استفاده می‌شود. در این تابع، داده‌های کیفیت هوا برای هر شهر از API گرفته شده و به شیوه‌ای مرتب و ساختار یافته ذخیره می‌شود. هر رکورد شامل اطلاعاتی مانند AQI (Air Quality Index)، PM2.5، PM10، CO، NO2، SO2، O3 و timestamp است که در AQIRecord ذخیره می‌شود. اگر خطایی در فرآیند جمع‌آوری داده‌ها به وجود آید، آن خطا در لیست errors ذخیره می‌شود.

این فایل با استفاده از API AQICN داده‌های لحظه‌ای کیفیت هوا را از شهرهای مختلف دریافت کرده و آن‌ها را در قالب یک ساختار داده‌ای ذخیره می‌کند که برای پردازش‌های بعدی، مانند ذخیره‌سازی در پایگاه داده یا تحلیل‌های مدل، قابل استفاده است.

فایل sqlite_storage.py (ذخیره داده‌ها در SQLite)

در فایل sqlite_storage.py، ما یک لایه ذخیره‌سازی داده‌ها با استفاده از پایگاه داده SQLite برای ذخیره اطلاعات مربوط به کیفیت هوا (AQI) طراحی کرده‌ایم. این فایل شامل کلاس‌هایی است که به صورت خاص برای ذخیره‌سازی و بازیابی داده‌ها از پایگاه داده SQLite ایجاد شده‌اند.

کلاس AQIRecord: این کلاس از نوع dataclass است و وظیفه ذخیره‌سازی داده‌های مربوط به کیفیت هوا برای یک شهر را بر عهده دارد. در این کلاس، اطلاعاتی مانند:

- شهر (city)
- AQI (شاخص کیفیت هوا)
- غلظت آلاینده‌ها (مانند pm25, pm10, co, no2, so2, o3)

زمان ثبت داده‌ها (timestamp): این داده‌ها به‌صورت یک شیء از نوع AQIRecord ذخیره می‌شوند و سپس در پایگاه داده ذخیره میشوند.

کلاس SQLiteStorage: این کلاس مسئول ارتباط با پایگاه داده SQLite و انجام عملیات ذخیره‌سازی و بازیابی داده‌ها است. در ادامه توضیح هر بخش از این کلاس آورده شده است:

سازنده (__init__):

- سازنده کلاس مسیر پایگاه داده را به‌عنوان ورودی دریافت می‌کند و متغیر db_path را تنظیم می‌کند.
- سپس، متد _init_db فراخوانی می‌شود تا پایگاه داده و جداول لازم را در صورت عدم وجود ایجاد کند.

اتصال به پایگاه داده (_connect): این متد یک اتصال به پایگاه داده SQLite برقرار می‌کند و همچنین برخی از تنظیمات مانند فعال کردن حالت WAL (Write-Ahead Logging) برای بهبود عملکرد و فعال کردن کلیدهای خارجی برای تضمین یکپارچگی داده‌ها را اعمال می‌کند.

ایجاد جداول پایگاه داده (_init_db): این متد جدول aqi_readings را در پایگاه داده ایجاد می‌کند که شامل ستون‌های مختلفی برای ذخیره‌سازی داده‌های AQI، غلظت آلاینده‌ها، شهر و زمان است. اگر این جدول قبلاً وجود داشته باشد، هیچ تغییری اعمال نمی‌شود.

درج داده‌ها (insert_many): این متد برای درج چندین رکورد AQI به‌صورت همزمان به پایگاه داده استفاده می‌شود. ورودی این متد یک لیست از اشیاء AQIRecord است که سپس به فرمت مناسب برای درج در پایگاه داده تبدیل می‌شوند. داده‌ها با استفاده از دستور executemany به پایگاه داده اضافه می‌شوند.

بازیابی آخرین داده‌ها برای هر شهر (fetch_latest_per_city): این متد برای بازیابی آخرین مقادیر AQI از هر شهر از پایگاه داده طراحی شده است. این کار با استفاده از یک کوئری SQL انجام می‌شود که برای هر شهر جدیدترین رکورد بر اساس تاریخ و زمان (timestamp) را انتخاب می‌کند. داده‌های بازیابی‌شده به‌صورت یک لیست از دیکشنری‌ها بازگشت داده می‌شود که هر دیکشنری شامل اطلاعات مربوط به یک رکورد AQI است.

در کل، این کلاس به شما اجازه می‌دهد تا داده‌های کیفیت هوای AQI را به‌راحتی در یک پایگاه داده SQLite ذخیره کرده و در آینده آن‌ها را بازیابی کنید. این کار با استفاده از روش‌های مختلف ذخیره‌سازی و بازیابی داده‌ها انجام می‌شود و برای پروژه‌هایی که نیاز به نگهداری داده‌های طولانی‌مدت دارند، بسیار مفید است.

فایل train_aqicn_model.py (آموزش مدل)

در فایل train_aqicn_model.py، ما یک مدل رگرسیون خطی برای پیش‌بینی شاخص کیفیت هوای AQI (Air Quality Index) با استفاده از داده‌های کیفیت هوای دریافت‌شده از سیستم AQICN می‌سازیم و مدل را به فرمت ONNX صادر می‌کنیم تا بتوانیم آن را در دیگر پلتفرم‌ها استفاده کنیم.

اولین بخش کد مربوط به بارگذاری داده‌ها است. تابع load_aqicn_dataframe مسئول بارگذاری داده‌ها از پایگاه داده SQLite است. این تابع یک اتصال به پایگاه داده برقرار می‌کند، داده‌های مربوط به خوانش‌های AQI را از جدول aqi_readings به‌صورت یک DataFrame از کتابخانه pandas می‌خواند و سپس اتصال به پایگاه داده را می‌بندد. این داده‌ها ابتدا برای پردازش آماده می‌شوند.

پس از بارگذاری داده‌ها، داده‌ها باید پیش‌پردازش شوند تا آماده آموزش مدل شوند. در اینجا، تابع preprocess_aqicn داده‌ها را پردازش می‌کند. ابتدا یک کپی از DataFrame اصلی ایجاد می‌شود تا داده‌ها به‌صورت مستقل تغییر کنند. سپس تمامی ستون‌های مربوط به ویژگی‌ها (pm25, pm10, co, no2, so2, o3) و هدف (aqi) به نوع عددی تبدیل می‌شوند. اگر در تبدیل داده‌ها مشکلی وجود داشته باشد (مانند داده‌های متنی یا خالی)، این مقادیر به NaN تبدیل می‌شوند. سپس ردیف‌هایی که شامل مقادیر NaN در هرکدام از ویژگی‌ها یا هدف هستند، حذف می‌شوند.

در مرحله بعد، داده‌های پیش‌پردازش‌شده به‌طور جداگانه برای ویژگی‌ها و هدف تفکیک می‌شوند. ویژگی‌ها در متغیر X قرار می‌گیرند و هدف که همان شاخص AQI است در متغیر y قرار می‌گیرد. سپس داده‌ها به

دو بخش آموزش و تست تقسیم می‌شوند. این تقسیم‌بندی با استفاده از تابع train_test_split انجام می‌شود، به طوری که 80 درصد داده‌ها برای آموزش مدل و 20 درصد برای ارزیابی مدل اختصاص می‌یابد.

مدل رگرسیون خطی با استفاده از کتابخانه scikit-learn ایجاد می‌شود و آموزش می‌بیند. پس از آموزش مدل، پیش‌بینی‌هایی برای داده‌های تست انجام می‌شود. برای ارزیابی دقت مدل، از معیار خطای میانگین مطلق (mean_absolute_error) استفاده می‌شود که نشان می‌دهد مدل چقدر در پیش‌بینی مقدار AQI دقیق عمل کرده است.

پس از ارزیابی مدل با استفاده از داده‌های تست، مدل به فرمت ONNX صادر می‌شود. این کار با استفاده از کتابخانه skl2onnx انجام می‌شود. مدل رگرسیون خطی به‌صورت باینری صادر شده و در مسیر مشخص‌شده ذخیره می‌شود. این مرحله به‌ویژه برای استفاده از مدل در پلتفرم‌های مختلف و اجرای سریع‌تر مدل‌ها در محیط‌هایی که از ONNX پشتیبانی می‌کنند مفید است.

در نهایت، مدل ONNX که به فرمت استاندارد ONNX تبدیل شده است، با استفاده از کتابخانه onnxruntime برای پیش‌بینی بر روی داده‌های تست استفاده می‌شود. پیش‌بینی‌ها انجام شده و دوباره خطای میانگین مطلق مدل ONNX محاسبه می‌شود. این مقدار خطا در خروجی نمایش داده می‌شود تا عملکرد مدل ONNX بررسی شود. در مجموع، این فایل برای آموزش مدل پیش‌بینی AQI با استفاده از داده‌های AQICN طراحی شده است و پس از آموزش مدل، آن را به فرمت ONNX صادر می‌کند تا بتوان از آن در دیگر پلتفرم‌ها و ابزارهای یادگیری ماشین استفاده کرد.

فایل aqicn_runner.py (اجرای پایپلاین)

در فایل aqicn_runner.py، یک پایپلاین کامل برای دریافت داده‌های کیفیت هوای AQICN، ذخیره‌سازی آن‌ها در پایگاه داده SQLite، و تولید نمودارهای مرتبط طراحی شده است.

بررسی وجود توکن API: در ابتدای کد، از ورودی api_token اطمینان حاصل می‌شود که یک توکن معتبر برای ارتباط با API سیستم AQICN وجود دارد. اگر این توکن وجود نداشته باشد، یک خطای RuntimeError ایجاد می‌شود که نشان‌دهنده لزوم وجود یک توکن معتبر برای دسترسی به داده‌ها است.

ایجاد مشتری API و ذخیره‌سازی SQLite: در این مرحله، یک شیء از کلاس AQIAPIClient برای تعامل با API AQICN ساخته می‌شود. این شیء به طور خودکار از توکن API برای دریافت داده‌ها استفاده خواهد کرد. همچنین یک شیء از کلاس SQLiteStorage برای ذخیره‌سازی داده‌ها در پایگاه داده SQLite ایجاد می‌شود. این شیء مسئول ذخیره‌سازی داده‌های AQI در پایگاه داده است.

دریافت داده‌ها از API: سپس با استفاده از تابع collect_records داده‌ها برای شهرهای مختلف دریافت می‌شود. این داده‌ها شامل اطلاعاتی مانند AQI و غلظت آلاینده‌ها هستند. در این مرحله، اگر خطاهایی در حین جمع‌آوری داده‌ها وجود داشته باشد، این خطاها در لاگ ثبت می‌شوند.

ذخیره‌سازی داده‌ها در پایگاه داده SQLite: داده‌های دریافتی با استفاده از متد insert_many در پایگاه داده SQLite ذخیره می‌شوند. پس از انجام این عملیات، تعداد ردیف‌های واردشده به پایگاه داده در لاگ ثبت می‌شود تا مشخص شود که چه تعداد رکورد به صورت موفقیت‌آمیز وارد شده است.

دریافت آخرین داده‌ها از پایگاه داده: پس از ذخیره‌سازی داده‌ها، از پایگاه داده آخرین مقادیر AQI برای هر شهر بازیابی می‌شود. این کار با استفاده از متد fetch_latest_per_city انجام می‌شود. اگر داده‌های AQI برای هر شهری موجود نباشد، این مسئله در لاگ ثبت شده و از ادامه پردازش آن رکورد جلوگیری می‌شود.

جمع‌آوری دوباره داده‌ها و بررسی وجود رکورد: پس از بازیابی آخرین داده‌ها از پایگاه داده، دوباره داده‌های جدید از API جمع‌آوری می‌شود. در صورتی که هیچ رکوردی برای ذخیره‌سازی یافت نشود، پیام خطا در لاگ ثبت می‌شود و ادامه پردازش متوقف می‌شود.

ایجاد نمودارها: در صورتی که داده‌های کافی جمع‌آوری شده باشد، یک شیء از کلاس PlotService برای تولید نمودارهای مختلف ایجاد می‌شود. در ابتدا، یک نمودار میله‌ای از داده‌های AQI تولید می‌شود که نشان‌دهنده میزان کیفیت هوا در هر شهر است. سپس، یک هیستوگرام خطا (که اختلاف بین مقادیر پیش‌بینی‌شده و واقعی را نمایش می‌دهد) تولید می‌شود. این نمودارها در مسیر مشخص‌شده ذخیره می‌شوند و پیام‌های موفقیت در لاگ ثبت می‌شود.

در این فایل، یک پایپلاین کامل برای جمع‌آوری داده‌ها از API AQICN، ذخیره‌سازی داده‌ها در پایگاه داده SQLite و تولید نمودارهای مربوط به تحلیل داده‌ها طراحی شده است. این فرآیند به‌طور خودکار داده‌ها را دریافت کرده، در پایگاه داده ذخیره می‌کند و سپس با استفاده از نمودارها به تحلیل وضعیت کیفیت هوا پرداخته و نتایج را به صورت تصویری ذخیره می‌کند.

فایل plots.py (ویژوالیزیشن نتایج)

در فایل plots.py، یک کلاس به نام PlotService طراحی شده است که وظیفه ایجاد نمودارهای مختلف از داده‌های کیفیت هوای AQI را بر عهده دارد. این کلاس به طور خاص برای ایجاد نمودارهایی آماده شده که می‌توانند برای ارائه نتایج تحلیل داده‌ها در پروژه‌های مختلف استفاده شوند. در ادامه، به شرح دقیق نحوه عملکرد این کلاس و توابع آن پرداخته می‌شود.

سازنده کلاس (__init__): سازنده کلاس PlotService یک مسیر (out_dir) برای ذخیره‌سازی فایل‌های خروجی دریافت می‌کند و اطمینان حاصل می‌کند که این مسیر وجود دارد. اگر مسیر مشخص‌شده

قبلا وجود نداشته باشد، با استفاده از دستور mkdir(exist_ok=True) آن را ایجاد می‌کند. این مسیر برای ذخیره‌سازی نمودارهایی که در ادامه تولید می‌شوند استفاده می‌شود.

نمودار میله‌ای AQI (plot_latest_aqi_bar): این تابع یک نمودار میله‌ای از داده‌های AQI برای هر شهر تولید می‌کند. ورودی این تابع یک لیست از دیکشنری‌ها است که شامل داده‌های AQI برای هر شهر است. این داده‌ها به یک DataFrame از کتابخانه pandas تبدیل می‌شوند تا بتوان آن‌ها را به راحتی پردازش و تحلیل کرد.

- ابتدا، داده‌های AQI به نوع عددی تبدیل می‌شوند و مقادیر نادرست یا ناقص (مثل NaN) حذف می‌شوند.
- سپس داده‌ها بر اساس مقدار AQI مرتب می‌شوند تا شهرهایی که AQI کمتری دارند در ابتدا قرار گیرند.
- یک نمودار میله‌ای رسم می‌شود که در آن محور X شهرها و محور Y مقدار AQI است.
- در بالای هر میله، مقدار AQI به‌طور مستقیم نمایش داده می‌شود تا تحلیل‌گر بتواند به راحتی میزان AQI هر شهر را مشاهده کند.
- در نهایت، نمودار در مسیر مشخص‌شده ذخیره می‌شود و فایل تصویری با فرمت PNG تولید می‌شود.

هیستوگرام خطاهای پیش‌بینی (plot_error_histogram): این تابع یک هیستوگرام از خطاهای پیش‌بینی تولید می‌کند. در اینجا، فرض شده است که داده‌های واقعی برای مقایسه با پیش‌بینی‌ها موجود هستند.

- ابتدا داده‌ها به یک DataFrame تبدیل میشوند و سپس خطاهای پیش‌بینی (تفاضل بین مقادیر واقعی و پیش‌بینی‌شده) محاسبه می‌شود. در این مثال، خطا به‌صورت تفاضل بین AQI و pm25 محاسبه شده است که یک نمونه از مقایسه‌های ممکن است.
- سپس یک هیستوگرام از این خطاها رسم می‌شود که توزیع خطاهای پیش‌بینی را نشان می‌دهد.
- این هیستوگرام می‌تواند به تحلیلگر کمک کند تا بفهمد مدل در کدام بخش‌ها عملکرد بهتری داشته و در کجا دچار اشتباهات بیشتری بوده است.
- مانند تابع قبلی، نمودار در مسیر مشخص‌شده ذخیره می‌شود.

کلاس PlotService برای تولید نمودارهای مختلف طراحی شده است که می‌توانند برای نمایش تحلیل داده‌ها و نتایج پیش‌بینی‌های مدل AQI استفاده شوند. این کلاس شامل دو تابع اصلی است: یکی برای رسم نمودار میله‌ای AQI برای هر شهر و دیگری برای تولید هیستوگرام خطاهای پیش‌بینی. در هر دو حالت، نمودارها به‌طور خودکار در مسیر تعیین‌شده ذخیره می‌شوند و آماده استفاده در گزارش‌ها و تحلیل‌ها هستند.

فایل aqi_history.sqlite (پایگاه داده)

فایل aqi_history.sqlite یک پایگاه داده SQLite است که داده‌های مربوط به کیفیت هوا (AQI) را ذخیره می‌کند. این پایگاه داده شامل جدول‌هایی است که داده‌های مربوط به شاخص کیفیت هوا (AQI) و

غلظت آلاینده‌ها برای شهرهای مختلف را نگهداری می‌کنند. به‌طور خاص، این فایل شامل اطلاعاتی مانند مقادیر AQI، غلظت آلاینده‌ها (مانند pm25, pm10, co, no2, so2, o3) و زمان ثبت داده‌ها است. این داده‌ها از سیستم AQICN جمع‌آوری می‌شوند و در پایگاه داده ذخیره می‌شوند تا در مراحل بعدی پردازش و تحلیل شوند. چون SQLite یک پایگاه داده سبک و فایل‌محور است، شما نمی‌توانید به‌طور مستقیم محتوای آن را مشاهده کنید، اما می‌توانید با استفاده از ابزارهایی مانند DB Browser for SQLite یا دستورات SQL در پایتون به داده‌ها دسترسی پیدا کرده و آن‌ها را بررسی کنید.

فایل test_aqicn_token.py (تست صحت توکن API)

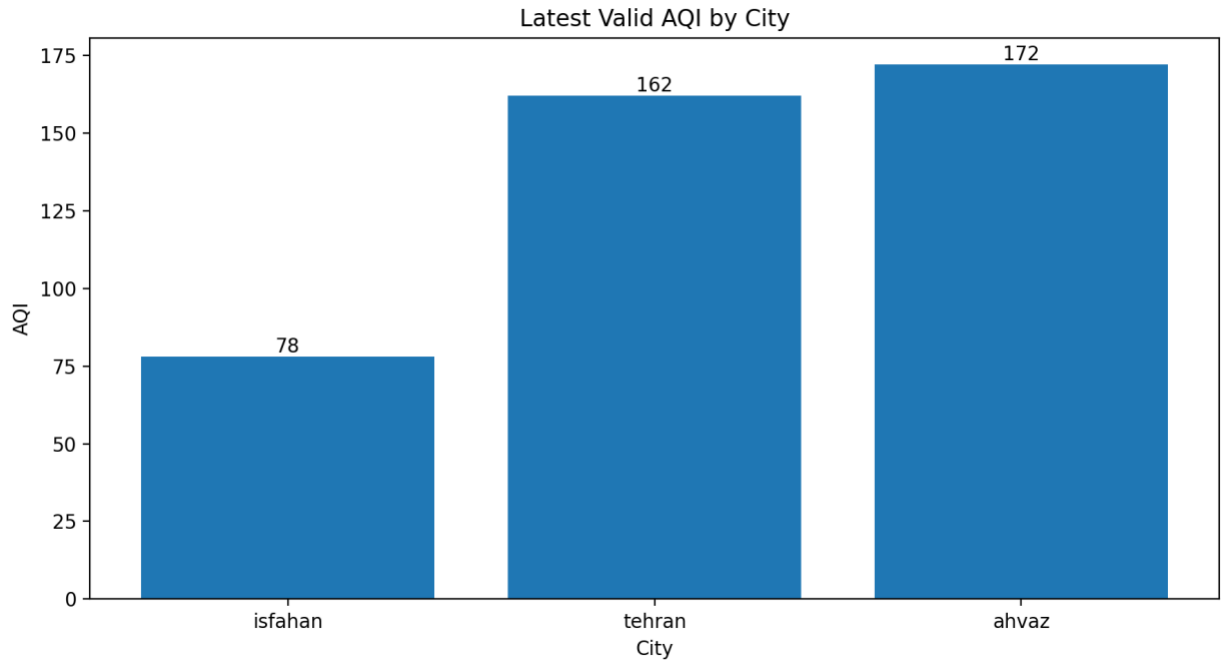
فایل test_aqicn_token.py برای تست صحت توکن API مربوط به سیستم AQICN طراحی شده است. این کد بررسی می‌کند که آیا توکن API به‌درستی تنظیم شده است و آیا می‌توان از آن برای دریافت داده‌های کیفیت هوای یک شهر استفاده کرد یا خیر. ابتدا، در ابتدای فایل، کتابخانه‌های مورد نیاز مانند os، requests و dotenv وارد می‌شوند. سپس، با استفاده از load_dotenv()، متغیرهای محیطی از فایل .env بارگذاری می‌شوند. این فایل معمولاً شامل اطلاعات حساس مانند توکن‌های API است که باید از آن برای تنظیم متغیرهای محیطی استفاده کرد.

بعد از بارگذاری متغیرهای محیطی، با استفاده از os.getenv('AQICN_API_TOKEN')، توکن API که در فایل .env ذخیره شده، بارگذاری می‌شود. این توکن باید برای دسترسی به داده‌های AQICN استفاده شود. اگر توکن به‌درستی تنظیم شده باشد، مقدار آن در متغیر token ذخیره می‌شود و در کنسول چاپ می‌شود.

سپس، برای تست صحت توکن، یک درخواست GET به API AQICN ارسال می‌شود. این درخواست شامل آدرس API برای دریافت داده‌های کیفیت هوای شهر تهران است که به‌صورت دینامیک با استفاده از نام شهر (city = "tehran") و توکن API ساخته می‌شود. درخواست به آدرس https://api.waqi.info/feed/tehran/?token={token} ارسال می‌شود که برای دسترسی به داده‌های AQI شهر تهران است. در نهایت، پاسخ دریافتی از API که به‌صورت JSON است، با استفاده از response.json() چاپ می‌شود. این قسمت از کد به شما این امکان را می‌دهد که داده‌های دریافتی را بررسی کنید و مطمئن شوید که توکن به درستی کار می‌کند و اطلاعات کیفیت هوا برای شهر تهران به‌درستی بارگذاری می‌شود.در مجموع، این فایل به عنوان یک تست برای بررسی صحت توکن API و صحت دریافت داده‌های AQICN از یک شهر خاص طراحی شده است.

نمودار latest_aqi.png (نمودار آخرین پیش‌بینی‌ها)

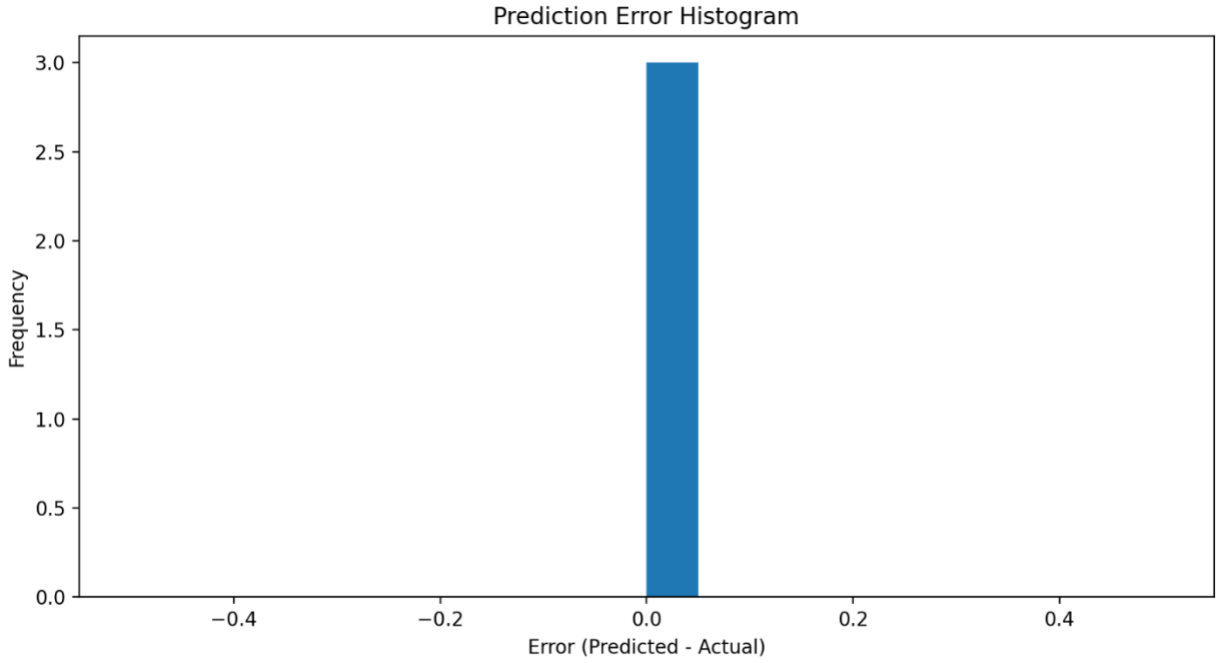
این نمودار میله‌ای نشان‌دهنده مقادیر آخرین پیش‌بینی‌های شاخص کیفیت هوا (AQI) برای چند شهر مختلف است. در این نمودار، هر میله نشان‌دهنده AQI یک شهر خاص است و ارتفاع میله‌ها نمایانگر مقدار AQI آن شهر می‌باشد. برای هر شهر، مقدار AQI در بالای میله نوشته شده است تا به راحتی قابل مشاهده باشد. این نمودار به طور خاص برای نمایش مقادیر AQI آخرین پیش‌بینی‌ها بر اساس داده‌های واقعی و پیش‌بینی‌شده ایجاد شده است. از این نمودار می‌توان برای مقایسه وضعیت کیفیت هوای مختلف شهرها استفاده کرد و خیلی سریع دید که کدام شهرها در معرض آلودگی بیشتر قرار دارند. در این مثال، تهران، اهواز و اصفهان به‌عنوان شهرهای نمونه انتخاب شده‌اند و مقدار AQI برای هرکدام از آن‌ها به نمایش درآمده است.



نمودار aqi_error_hist.png (هیستوگرام خطای پیش بینی)

این هیستوگرام نمایش‌دهنده توزیع خطاهای پیش‌بینی مدل است. در این نمودار، محور افقی (X) نشان‌دهنده تفاوت یا خطا بین پیش‌بینی مدل و مقدار واقعی AQI است. در این حالت، خطا به صورت تفاضل بین مقادیر پیش‌بینی‌شده و واقعی محاسبه می‌شود. محور عمودی (Y) نیز نشان‌دهنده تعداد دفعاتی است که هر میزان خطا در داده‌ها مشاهده شده است. به عبارت دیگر، این نمودار توزیع خطاهای پیش‌بینی مدل را به صورت گرافیکی نمایش می‌دهد. در این هیستوگرام، بیشترین تعداد خطاها در بازه‌های نزدیک به صفر

قرار دارند، که نشان دهنده این است که بیشتر پیش‌بینی‌ها با خطاهای کم انجام شده‌اند و مدل عملکرد مناسبی در پیش‌بینی مقادیر AQI داشته است. از این نمودار می‌توان برای ارزیابی صحت مدل و شناسایی بخش‌هایی که مدل در آن‌ها دقت کمتری دارد، استفاده کرد.



خروجی نهایی حاصل از دیتاست aqicn

در این خروجی، اجرای دستور python -m src.main --mode aqicn را مشاهده می‌کنیم که مربوط به اجرای پایپلاین داده‌های AQICN است. این پایپلاین مسئول جمع‌آوری داده‌ها از سیستم AQICN، ذخیره‌سازی آن‌ها در پایگاه داده SQLite و تولید نمودارها است.

ورود داده‌ها به پایگاه داده SQLite:

- خط اول نشان می‌دهد که تعداد 4 رکورد به پایگاه داده SQLite وارد شده است. این رکوردها شامل اطلاعات مربوط به AQI (شاخص کیفیت هوا) برای شهرهای مختلف هستند.
- بعد از وارد کردن داده‌ها، مسیر پایگاه داده که داده‌ها در آن ذخیره شده، نشان داده می‌شود:

(C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\aqi_history.sqlite).

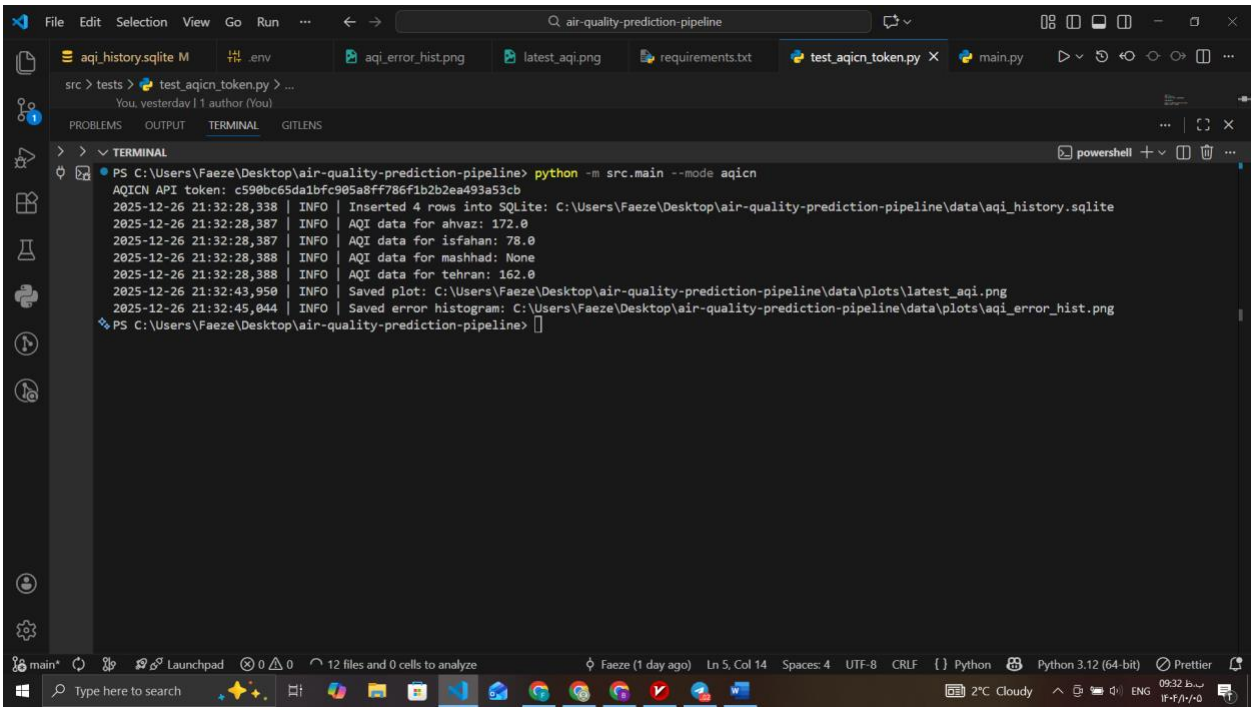
داده‌های AQI برای شهرهای مختلف: در ادامه، داده‌های AQI برای چند شهر مختلف به‌طور جداگانه گزارش می‌شود.

- برای شهر اهواز (Ahvaz) مقدار AQI برابر با 172.0 است.
- برای شهر اصفهان (Isfahan) مقدار AQI برابر با 78.0 است.
- برای شهر مشهد (Mashhad) داده AQI موجود نیست (مقدار None).
- برای شهر تهران (Tehran) مقدار AQI برابر با 162.0 است.

ذخیره‌سازی نمودارها: پس از پردازش داده‌ها، نمودارها ایجاد و ذخیره می‌شوند:

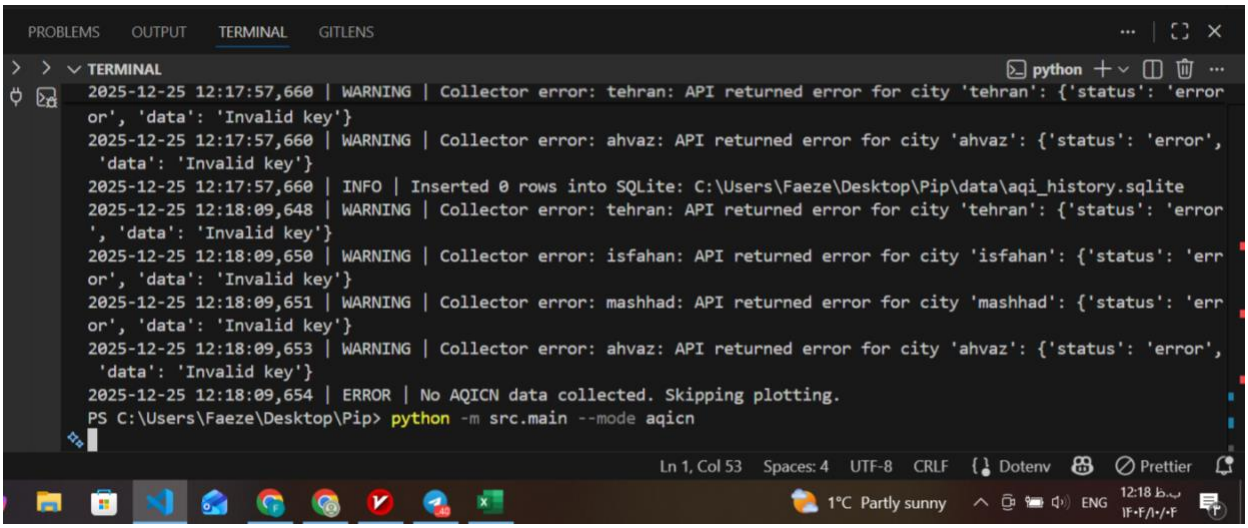
- اولین نمودار که به‌عنوان latest_aqi.png ذخیره شده است، نشان‌دهنده آخرین مقادیر AQI برای هر شهر است. این نمودار به‌صورت میله‌ای نمایش داده می‌شود.
- دومین نمودار که به‌عنوان aqi_error_hist.png ذخیره شده، هیستوگرام خطاهای پیش‌بینی مدل را نمایش می‌دهد.

در این خروجی، به‌طور کلی، داده‌ها برای چندین شهر از سیستم AQICN دریافت شده، وارد پایگاه داده شده و نمودارهای مختلفی برای تحلیل و ارزیابی عملکرد مدل تولید شده‌اند. همچنین، خطاهای پیش‌بینی و مقادیر AQI برای هر شهر به‌طور دقیق گزارش شده‌اند.


File Edit Selection View Go Run
air-quality-prediction-pipeline
aqi_history.sqlite M
.env
aqi_error_hist.png
latest_aqi.png
requirements.txt
test_aqicn_token.py
main.py
src > tests > test_aqicn_token.py > ...
You, yesterday | 1 author (You)
PROBLEMS OUTPUT TERMINAL GITLENS
TERMINAL
powershell
PS C:\Users\Faeze\Desktop\air-quality-prediction-pipeline> python -m src.main --mode aqicn
AQICN API token: c590bc65da1bfc905a8ff786f1b2b2ea493a53cb
2025-12-26 21:32:28,338 | INFO | Inserted 4 rows into SQLite: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\aqi_history.sqlite
2025-12-26 21:32:28,387 | INFO | AQI data for ahvaz: 172.0
2025-12-26 21:32:28,387 | INFO | AQI data for isfahan: 78.0
2025-12-26 21:32:28,388 | INFO | AQI data for mashhad: None
2025-12-26 21:32:28,388 | INFO | AQI data for tehran: 162.0
2025-12-26 21:32:43,950 | INFO | Saved plot: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\plots\latest_aqi.png
2025-12-26 21:32:45,044 | INFO | Saved error histogram: C:\Users\Faeze\Desktop\air-quality-prediction-pipeline\data\plots\aqi_error_hist.png
PS C:\Users\Faeze\Desktop\air-quality-prediction-pipeline>
main* Launchpad 0 0 12 files and 0 cells to analyze
Faeze (1 day ago) Ln 5, Col 14 Spaces: 4 UTF-8 CRLF { } Python Python 3.12 (64-bit) Prettier
Type here to search
2°C Cloudy
ENG

اگر api کار نکند چه می‌شود؟ مثلا منقضی یا باطل شود؟

اگر توکن API وجود نداشته باشد یا به هر دلیلی معتبر نباشد (مانند توکن اشتباه یا منقضی شدن)، در خروجی خطاهایی مانند "API returned error for city" مشاهده می‌شود، همانطور که در این اسکرین‌شات هم مشخص است. در این حالت، پایپلاین به درستی نمی‌تواند داده‌ها را از API دریافت کند و تمام داده‌های مربوط به شهرها به‌صورت `Invalid key` یا خطای مشابه برمی‌گردند. در نتیجه، هیچ داده‌ای در پایگاه داده SQLite وارد نمی‌شود و پیام "No AQICN data collected. Skipping plotting." در نهایت چاپ می‌شود که به این معنی است که هیچ داده‌ای جمع‌آوری نشده است و بخش مربوط به ترسیم نمودارها نیز انجام نمی‌شود. در این شرایط، باید توکن API را بررسی و مطمئن شوید که معتبر و به‌درستی در فایل `.env` تنظیم شده است.



هدف این پروژه چه بود؟

هدف این پروژه پیش‌بینی و تحلیل کیفیت هوا (AQI) با استفاده از داده‌های API AQICN و داده‌های محلی (UCI) بود. در این پروژه، داده‌ها از دو منبع مختلف جمع‌آوری، پردازش، ذخیره و تحلیل شدند. سپس مدل‌های یادگیری ماشین برای پیش‌بینی AQI آموزش داده شدند و در نهایت نتایج به صورت نمودارهای تصویری برای تحلیل و ارزیابی بهتر کیفیت هوا نمایش داده شدند.

پایان پروژه این است که با استفاده از داده‌های کیفیت هوا و مدل‌های پیش‌بینی، توانستیم وضعیت کیفیت هوای شهرهای مختلف را تجزیه و تحلیل کنیم و ابزارهایی برای نمایش نتایج به صورت تصویری فراهم کنیم. این ابزار در تصمیم‌گیری‌های زیست محیطی و برنامه‌ریزی شهری مفید خواهد بود.

مسیر پروژه و منابع:

ما یک هفته مشغول یادگیری و فهم مسیر پروژه و دو هفته دیگر را مشغول اجرای ایده‌ها و خواسته‌های دکتر شیرازی بودیم. در این راه از سایت‌های مختلفی چون w3school، stack overflow و GeeksforGeeks و Raddith استفاده کردیم. ناگفته نماند که ویدئوهای موجود در یوتیوب و گوگل هم در آموختن ماشین لرنینگ، پیش پردازش داده‌ها و پیاده‌سازی رگرسیون خطی، به ما کمک زیادی کردند.

در نهایت، امیدواریم بتوانیم با یاد گرفتن بیشتر، در راستای ارتقا و توسعه این پروژه نیز کوشا باشیم.

باتشکر از همراهی شما