

بسم الله الرحمن الرحيم



دانشگاه بیرجند
University of Birjand

رشته کامپیوتر-گرایش نرم افزار

کارشناسی پیوسته

موضوع پروژه:

پیش بینی ارزان ترین قیمت بلیت هواپیما در بازه زمانی مشخص

استاد راهنما:

امیر کیوان شفیعی

دانشجو:

مصطفی وحدانی دهکلانی

سال تحصیلی:

۱۴۰۱-۱۴۰۲

۳.....	مقدمه
۴.....	فصل ۱
۴.....	معرفی پروژه و اهداف
۴.....	مروری بر دیتاست
۵.....	ساختار دایرکتوری ها
۸.....	کمی به ساختار عمیق تر بنگریم ■■■
۹.....	ماژول ها
۱۱.....	مابقی فایل ها ...
۱۲.....	چگونه از پروژه اجرا بگیریم؟
۱۴.....	فصل ۲
۱۴.....	تکنولوژی ها و ابزارهای به کار رفته
۱۵.....	کتابخانه های داخلی زبان پایتون
۱۶.....	کتابخانه و فریمورک های مشترک
۱۶.....	کتابخانه ها و فریمورک های زیر پروژه flight_tickets_scraper
۱۸.....	کتابخانه ها و فریمورک های زیر پروژه flight_tickets_preprocessing
۲۰.....	ابزارهای مرتبط استفاده شده در طول پروژه
۲۱.....	API های استفاده شده در پروژه
۲۲.....	فصل ۳
۲۲.....	عملکرد و توضیح کد ها
۲۲.....	فاز اول پروژه
۵۲.....	فاز دوم پروژه
۷۴.....	فاز سوم پروژه
۷۹.....	فاز چهارم پروژه
۸۵.....	فصل ۴
۸۵.....	خروجی های پروژه و اصلاحات مورد نیاز
۸۵.....	خروجی ها
۸۶.....	اصلاحات
۸۸.....	فصل ۵
۸۸.....	جمع بندی و نتیجه گیری
۸۹.....	منابع

مقدمه

اگر در تاریخ بنگریم از تمدن کهن گرفته که ساختار نوشتاری خاصی نداشتند و اطلاعات را با تعریف داستان در قالب افسانه‌ها به نسل های بعدی منتقل می کردند تا تمدن مدرن دیجیتال که اطلاعات را در بستر اینترنت و یا در قالب کتاب‌ها می‌نویسند و یا همین امروز که شما شروع به خواندن این داکيومنت کرده اید تا بوده، داشتن اطلاعات و طریقه استفاده کردن از آن‌ها همیشه مهم‌ترین بخش زندگی ما را تشکیل می‌دادند و نقش حیاتی را در اتخاذ تصمیم های شخصی روزانه و رویداد های تاریخی ایفا کرده‌اند و خواهند کرد ولی فقط و فقط در عصر امروزی است که اکثریت جامعه متوجه اهمیت به سزای آن در داشتنش شده‌اند. از نقش حیاتی استفاده از اطلاعات در تاریخ می‌توان به جنگ انقلابی آمریکا که نشان دهنده مبارزه سیزده مستعمره آمریکا علیه سلطه بریتانیا بود که در نهایت منجر به تولد ایالات متحده آمریکا شد اشاره کرد. جمع آوری، انتشار و مدیریت اطلاعات نقشی اساسی در شکل دادن به روند و نتیجه این جنگ داشت. هر دو طرف درگیری نیز از تبلیغات و گسترش شایعه برای شکل دادن به افکار عمومی استفاده می کردند. این تنها یکی از هزاران نمونه تاریخی است که در آن داشتن اطلاعات نقش مهمی را ایفا کرده است. اما شما نیز از داشتن اطلاعات در زندگی روزمره خودتان بهره برده‌اید به مثال ساده می‌تواند این باشد که فرض کنید می‌خواهید با خانواده خود به یک سفر تفریحی با هواپیما به شهری دیگر بروید برای شما باید مهم باشد که شرایط آب و هوایی در شهر مقصد چگونه است و یا چگونه می‌توانید یک بلیت ارزان قیمت را در یک بازه زمانی مشخص خریداری کنید این دقیقاً کاری است که ما می‌خواهیم توسط داشتن اطلاعات در این پروژه با کمک علم داده (Data Science) انجام دهیم. امروزه داده‌ها در جهان به شدت رو به افزایش هستند. باید بدانید که داده در عصر امروزی به معنای واقعی، یعنی همه چیز. همه چیز ما یک داده هست. پیشنهاد ما، آگاهی ما، داشته های ما، تاریخچه مرورگر ما، شماره های مخاطبین گوشی ما، پیام‌های درون پیام رسان های ما، دارایی ما، دانستنی های ما، علایقی که دنبال می‌کنیم و غیره همه و همه یک نوع داده محسوب می شوند. به طور کلی از طریق جمع آوری این داده‌ها می‌توان اطلاعات آماری خاصی را استخراج کرده و از آن‌ها در زمینه‌های متفاوت در جهت اهداف خیر و یا خصمانه بهره برد و یا اینکه با آن نیاز روزمره افراد را برطرف کرد. علم داده، ابزارها و روش‌هایی را به ما پیشنهاد می‌دهد که به ما در استخراج این اطلاعات آماری به بهترین شکل ممکن کمک خواهد. این ابزارها، روش‌ها، الگوریتم ها و مدل ها به پیدا کردن الگوهای پنهان درون دیتاست های خام جمع آوری شده می پردازند. باید اشاره کرد، به اشخاصی که در زمینه علم داده فعالیت می‌کنند دیتا ساینتیست (Data Scientist)، متخصص داده و یا دانشمند داده گفته می‌شوند و لازم است قبل از اینکه شما یک دیتا ساینتیست باشید حداقل در مباحثی مثل ریاضیات، آمار و احتمال، برنامه نویسی، پایگاه داده ها، یادگیری ماشین و مدل سازی تسلط کافی داشته باشید تا بتوانید قدم در دنیای بزرگ علم داده بگذارید.

فصل ۱

معرفی پروژه و اهداف

پروژه ایی که در پیش روی شما قرار دارد پروژه تحلیل قیمت بلیت هواپیما در یک بازه زمانی مشخص می باشد. تحلیل قیمت بلیت هواپیما برای پیدا کردن ارزان ترین قیمت در یک بازه زمانی معمولاً کار سخت و دشواری است زیرا که قیمت هواپیما در لحظه دچار تغییر و نوسانات می باشد. قیمت بلیت هواپیمایی که امروز می خرید می تواند با قیمت بلیت هواپیمایی مشابه که فردا می خرید بسیار متفاوت باشد و این نوسانات به سیاست های جداگانه هر شرکت هواپیمایی، آژانس های چارتر کننده بلیت، مسیرهایی متفاوت هواپیما برای مقاصد یکسان، فصل سال و غیره بستگی دارد. دیتاست خام جمع آوری شده دارای داده های سطری از تاریخ ۱۴۰۲/۰۵/۱۷ تا تاریخ ۱۴۰۲/۰۷/۱۵ در ۱۴۷۵۲ سطر و ۱۵ ستون و همچنین دیتاست پیش پردازش شده دارای داده های سطری از تاریخ ۱۴۰۲/۰۵/۱۷ تا تاریخ ۱۴۰۲/۰۷/۱۵ در ۱۳۹۴۷ سطر و ۱۶ ستون می باشد و نیز دیتاست نهایی ما دارای داده های سطری از تاریخ ۱۴۰۲/۰۵/۱۷ تا تاریخ ۱۴۰۲/۰۷/۱۵ در ۱۳۹۴۷ سطر و ۱۱ ستون می باشد. البته باید به این نکته نیز اشاره کرده که پروژه ما در مبحث یادگیری ماشین از نوع رگرسیون می باشد زیرا که ما می خواهیم یک مقدار پیوسته (هزینه بلیت) را پیش بینی کنیم پس باید توجه کنیم که معیارهای ارزیابی ما نیز باید بررسی میزان خطای موجود به ازای هر پیش بینی باشد که در این مورد می توان از روش های ارزیابی MSE و یا MAE استفاده کرد. گفتنی است که این پروژه در چهار فاز تهیه شده است و روند انجام آن به ترتیب زیر می باشد:

۱. کراول کردن سایت هدف در جهت جمع آوری یک دیتاست خام. در این پروژه از سایت tcharter.ir که یک موتور جستجو بلیت های هواپیما می باشد استفاده کرده ام.
۲. پیش پردازش داده های خام و آماده سازی یک دیتاست تمیز شده.
۳. آماده کردن داده های تمیز شده برای مدل یادگیر (اعمالی مثل نرمال سازی، تبدیل داده ها و غیره)
۴. آموزش دادن مدل و پیش بینی قیمت.

برای دریافت پروژه به صورت clone شده نیز می توانید با کلیک برروی [Mostafa-mvd/Flight-Tickets](https://github.com/Mostafa-mvd/Flight-Tickets) به لینک گیت هاب پروژه بنده مراجعه کنید. اگر بعد از مطالعه داکيومنت، این پروژه موجب خرسندی شما شده بسیار سپاس گزار خواهم بود که با یک ستاره پروژه را در گیت هاب حمایت کنید.

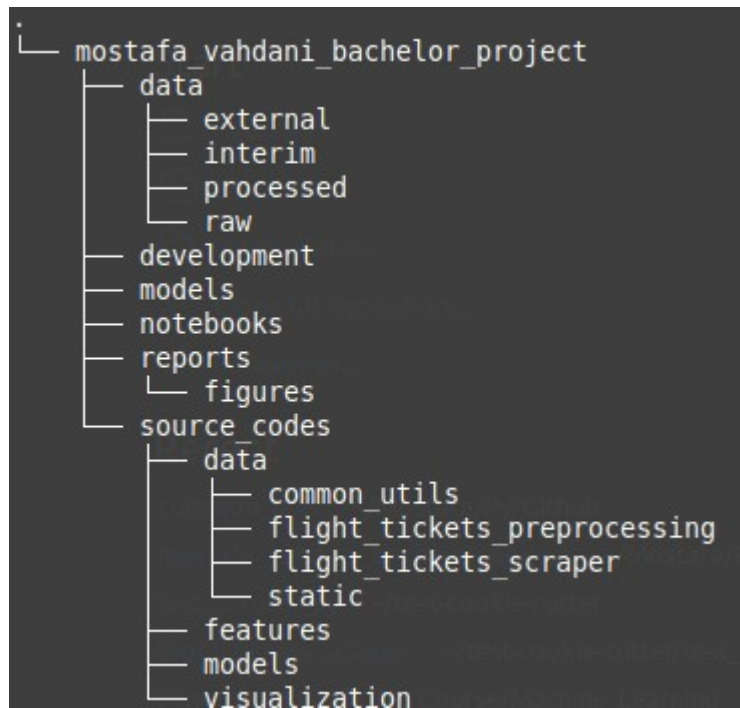
مروری بر دیتاست

۱. `national_departure_code` : کد استاندارد بین المللی برای شناسایی فرودگاه مقصد. (سه حرفی می باشد)
۲. `departure_city` : اسم شهر مقصد.
۳. `departure_airport` : اسم فرودگاه مقصد.
۴. `national_arrival_code` : کد استاندارد بین المللی برای شناسایی فرودگاه مبدا. (سه حرفی می باشد)

۵. **arrival_city** : اسم شهر مبدأ.
۶. **arrival_airport** : اسم فرودگاه مبدأ.
۷. **orthodromic_distance_KM** : فاصله دایره بزرگ (Great-circular distance) بین شهر مبدأ و مقصد. به عبارتی فاصله مستقیم بین شهر مبدأ و مقصد می باشد (با فاصله جاده ایی تفاوت دارد). این فاصله به کیلومتر می باشد که با KM در نام ستون مشخص شده است.
۸. **flight_length_min** : مقدار زمانی که طول می کشد تا پرواز تمام شود که واحد آن به دقیقه (min) - می باشد.
۹. **departure_date_YMD_format** : تاریخ حرکت هواپیما از مقصد که به فرمت Year-Month-Day می باشد.
۱۰. **local_departure_time** : ساعت محلی حرکت هواپیما از مقصد.
۱۱. **local_arrival_time** : ساعت محلی رسیدن هواپیما به مبدأ.
۱۲. **company_name** : اسم شرکت هواپیمایی یا همان ایرلاین.
۱۳. **flight_number** : شماره پرواز. (با این شماره می توان به اطلاعات پرواز از جمله ساعت حرکت، ساعت رسیدن، نوع بلیت و غیره دسترسی داشت)
۱۴. **flight_sale_type** : نوع فروش بلیت هواپیمایی که به صورت چارتری یا سیستمی می باشد.
۱۵. **fare_class_code** : کد کلاس نرخ بلیت هواپیمایی. (کد کلاس نرخ بعضی شرکت های هواپیمایی می تواند یکسان ولی مفهوم آن متفاوت از یکدیگر باشد). این کلاس نرخ می تواند با توجه به ایرلاین مربوطه مشخص کند که کلاس پروازی شما از چه نوعی است اکونومی، بیزنس، اکونومی پرمیوم و یا فرست.
۱۶. **ticket_price_T** : هزینه بلیت هواپیمایی که واحد آن به تومان (T) می باشد.
۱۷. **flight_capacity** : مقدار ظرفیت موجود برای هر بلیت هواپیمایی می باشد.
۱۸. **ticket_id** : مقدار id برای هر بلیت هواپیمایی یا همان داده سطری.

ساختار دایرکتوری ها

بعد از معرفی پروژه و اهداف آن حال به توضیح ساختار پروژه و دایرکتوری ها (فولدر یا همان پوشه) می پردازیم تا متوجه شویم هر قسمت از ساختار، مربوط به چه فعالیت و عملی می باشد. ساختار کلی پروژه ما بدون در نظر گرفتن مازول های پایتونی و دیگر فایل ها به شرح عکس زیر می باشد.



همانطور که مشاهده می‌کنید دایرکتوری روت و اصلی پروژه به نام `mostafa_vahdani_bachelor_project` می‌باشد که زیر دایرکتوری‌های آن عبارت‌اند از:

۱. **data**: داده‌هایی که مرتبط با بخش `Train` و `Predict` پروژه ما باشند در این دایرکتوری قرار خواهند گرفت مثل دیتاست‌های پردازش شده، تمیز شده و غیره.

- **external**: دیتاهایی که از طریق `Third Party` یا به فارسی همان شخص ثالث بدست آمده در این پوشه قرار خواهند گرفت. به عبارت ساده‌تر دیتاهایی خارجی‌ای که توسط خود شما تولید نشده‌اند را در این مسیر قرار دهید.

- **interim**: داده‌هایی که در اواسط کار بر روی ماشین تولید می‌شوند در این پوشه قرار خواهند گرفت.
- **processed**: مکان قرارگیری تمام داده‌های پیش پردازش شده و تمیز شده در این دایرکتوری می‌باشد.
- **raw**: داده‌های خامی را که از مرحله `Scraping` یا هر جای دیگر بدست آمده‌اند و یا به طور کلی داده‌هایی که هنوز آماده نیستند تا برای آموزش ماشین استفاده شوند را در این پوشه قرار می‌دهیم.

۲. **development**: در این دایرکتوری فایل‌های توسعه و یا فایل پیش نیازهای پروژه قرار خواهند گرفت مثل `requirements.txt` و یا در صورت لزوم فایل‌های `Docker`

۳. **models**: مدل `Train` شده یا `Serialized` شده ما و یا خلاصه‌ای از مدل نهایی در اینجا قرار می‌گیرند.

۴. **notebooks**: این دایرکتوری برای بخش یادگیری ماشین و اعمالی مثل پیش‌بینی، آموزش و پیش پردازش بسیار مهم می‌باشد زیرا که استفاده از ابزاری مثل `Jupyter` کار را برای ما آسان‌تر می‌کند (هرچند باید توجه شود که باز هم به دانش برنامه نویسی مثل `Python` همچنان نیاز است زیرا که ژوپیتر صرفاً یک محیط برای اجرا گرفتن از کد شما مثل `Google Colab` می‌باشد). در این پروژه فایل‌های مربوط به

ژوپیتر با فرمت اسمی مشخص و سراسری قرار خواهند گرفت. به طور مثال فرمت اسمی زیر را در نظر بگیرید:

1.1-arrival-time-data-preprocessing.ipynb

عدد ۱.۱ ترتیب استفاده از فایل‌های ژوپیتر را به شما می‌گوید یعنی اگر فایلی در این دایرکتوری بود که با عدد ۱.۳ شروع می‌شد یعنی اینکه باید بعد از ۱.۱ از آن استفاده کرد و اگر به فرض ۱.۰ بود باید قبل از ۱.۱ از آن استفاده کرد. در ادامه برای اینکه نشان دهیم چه کاری را در چه مرحله‌ای از یادگیری ماشین داریم انجام می‌دهیم توضیحات آنرا به صورت مختصر و مفید با “-” از یکدیگر جدا می‌کنیم.

مثال بالا را در نظر بگیرید ما از کلمه arrival-time استفاده کرده‌ایم یعنی اینکه می‌خواهیم در این فایل ژوپیتر روی خروجی arrival-time کار کنیم و این فعالیت را در مرحله data-preprocessing داریم انجام می‌دهیم.

۵. **report** : گزارش‌های تحلیلی نهایی خود را می‌توانید با تایپ‌های متفاوت مثل ، HTML ، PDF ، Figures ، Graphical و غیره را در این دایرکتوری تولید کنید و به افراد نشان دهید.

۶. **source_codes** : تمامی **کدهای اصلی** شما که می‌توانند شامل ، Pre-Processing ، Scraping ، Visualization و غیره می‌شود در این پوشه قرار می‌گیرند.

- **data** : در ابتدا باید توجه شود که این پوشه data با پوشه دیتایی که در بالا توضیح داده شد بسیار تفاوت دارد و یکسان نیستند پوشه دیتا بالایی برای دیتاست‌های تولید شده ما در هر مرحله از یادگیری ماشین بوده‌اند و این پوشه‌ی دیتایی که در مسیر source_codes قرار دارد پوشه‌ای است که تمامی کدهای ما برای جمع‌آوری و تولید دیتاست خام از طریق Scraping و انجام اعمال پیش‌پردازش بر روی دیتاست خام در آن قرار دارند. جلوتر به توضیح بیشتر در رابطه با دایرکتوری‌های موجود در این پوشه می‌پردازیم.
- **features** : بعد از عمل پیش‌پردازش، برای تولید مقادیر مناسب ویژگی‌ها در جهت استفاده از آن‌ها در مدل یادگیر مد نظر (به طور مثال تبدیل مقادیر Categorical به Numeric). از این دایرکتوری استفاده خواهیم کرد که به صورت پیش‌فرض یک فایل برای این امر به نام build_features.py در آن قرار دارد.
- **models** : کدهای مرتبط با شروع آموزش مدل، پیش‌بینی و ارزیابی مدل را می‌توانیم در این قسمت پیاده‌سازی کنیم (هرچند ما در پروژه خود از فایل‌های ژوپیتر برای آسانی کار استفاده کردیم). به صورت پیش‌فرض فایل‌هایی مثل predict_model.py و train_model.py در آن قرار دارند.
- **visualization** : خروجی کار بر روی دیتاست خود را می‌توانید در این پوشه نمایش دهید و دیتاست خود را به صورت بصری مشاهده کنید تا از این طریق ارزیابی‌هایی را روی دیتاست خود در جهت بهبود آن بدست آورید. به صورت پیش‌فرض یک فایل برای این امر به نام visualize.py در آن قرار دارد.

کمی به ساختار عمیق‌تر بنگریم...

حال که یک توضیح اجمالی در رابطه با دایرکتوری‌ها دادیم می‌خواهیم به ساختار دایرکتوری `source_codes/data` بیشتر نگاه بندازیم.

```
— common_utils
— flight_tickets_preprocessing
— flight_tickets_scraper
  └─ flight_tickets_scraper
    └─ spiders
— static
```

وقتی به ساختار پوشه دیتا درون مسیر `mostafa_vahdani_bachelor_project/source_codes/data` نگاه کنیم همچون ساختاری را مثل تصویر بالا مشاهده خواهیم کرد. که این ساختار عبارت‌اند از:

۱. **common_utils** : معمولاً از `utils.py` زمانی استفاده می‌شود که ما در پروژه خود ابزارهایی داریم که در آینده حتماً به آن‌ها نیاز پیدا خواهیم کرد پس این ابزارها که اعم از `function` و یا `class` های متعددی می‌باشند را در این فایل قرار خواهیم داد. اما ما در اینجا کار دیگری نیز انجام داده‌ایم ما در پروژه خود دو زیر پروژه غیر وابسته و مستقل به نام های `flight_ticket_preprocessing` و `flight_tickets_scraper` داشته‌ایم با اینکه این دو پروژه به هم وابسته نیستند ولی یکسری از اعمال در این زیر پروژه یکسان هستند و استفاده زیادی دارند پس با تعریف یک `package` سراسری به اسم `common_utils` این ابزارها را به شکل سراسری در هر دو زیر پروژه می‌توانیم استفاده کنیم. در مبحث توضیح پروژه نمی‌گنجد که راجع به چگونگی ایجاد پکیج و استفاده از آن توضیح دهیم پس از این مرحله گذر می‌کنیم و توضیح ساختار درون این دایرکتوری نمی‌پردازیم ولی در جلوتر حتماً راجه ابزارهایی که استفاده کردیم توضیحاتی را به شما ارائه خواهیم داد.

۲. **flight_tickets_preprocessing** : این دایرکتوری همانطور که از اسمش کاملاً مشخص است و جای فکر برای شما باقی نمی‌گذارد از مجموعه‌ای از ماژول‌های پایتونی تشکیل شده است که وظیفه اعمال پیش پردازش و تمیز کردن داده‌های خام جمع‌آوری شده را بر عهده دارند. بعداً راجه این ماژول‌ها توضیحاتی ارائه خواهیم داد.

۳. **flight_tickets_scraper** : از اسم این دایرکتوری هم کاملاً مشخص است که به عنوان یک کراولر/اسپایدر وظیفه جمع‌آوری داده خام را از سایت هدف بر عهده دارد. این دایرکتوری که دایرکتوری ریشه زیر پروژه `Flight Tickets Scraping` می‌باشد دارای یک نام به اسم `flight_tickets_scraper` و همچنین یک زیر دایرکتوری به همین اسم می‌باشد که تمامی کدهای **مربوط** به کراولر یا اسپایدر خود را در این بخش می‌نویسیم. بعداً در بخش ماژول‌ها بیشتر راجه عملکرد این کراولر توضیح خواهیم داد. همچنین

این دایرکتوری در دل خودش یک دایرکتوری به اسم spiders نیز دارد که در آن تمامی اسپایدر های مورد نیاز برای کراول کردن سایت هدف را تعریف خواهیم کرد.

۴. **static** : در این پوشه تمامی فایل های استاتیک یا همان ایستا که در طول پروژه از آن ها استفاده خواهیم کرد چه برای کراول و چه برای پیش پردازش را قرار می دهیم که شامل فایل هایی مثل:

- **airlines_code.json** : این فایل شامل کدهای استاندارد شناسایی IATA و یا ICAO مرتبط با هر ایرلاین یا همان شرکت هواپیمایی می باشد. (محتوای این فایل به صورت دستی فراهم شده است)

- **airport_city_codes.json** : این فایل شامل کدهای استاندارد شناسایی هر Airport یا همان فرودگاه می باشد. (محتوای این فایل توسط سایت هدف فراهم شده ولی به صورت دستی آن را ویرایش کرده ایم تا راحت تر قابل خواندن باشد).

- **airports_geometry.json** : این فایل شامل مختصات جغرافیایی هر فرودگاه به صورت lat یا همان latitude (عرض جغرافیایی - x) و lng یا همان longitude (طول جغرافیایی - y) در پارامتر geometry می باشد. (این فایل توسط یک اسکریپت پایتونی فراهم شده است)

- **airports_info.json** : این فایل شامل اطلاعات هر فرودگاه مثل نام رسمی و محلی فرودگاه به فارسی و انگلیسی، کد شناسایی فرودگاه، اسم شهر فرودگاه، ناحیه یا منطقه ایی که فرودگاه در آن قرار دارد و کد IATA دو حرفی کشور آن فرودگاه. (این فایل به صورت دستی فراهم شده است)

نکته: توجه شود که این فایل ها متناسب با فرودگاه های موجود در سایت هدف فراهم شده اند و ما اطلاعات کل فرودگاه های جهان یا ایران را فراهم نکرده ایم.

- **months.json** : دوازده ماه شمسی با شماره متناسب با آن ها در این فایل قرار دارد.

نکته: این فایل ها در طول نوشتن این پروژه و به مرور زمان ایجاد شده اند و از اولین زمان ایجاد پروژه وجود نداشتند.

ماژول ها

حال که ساختار دایرکتوری های پروژه را مورد بررسی قرار دادیم در ادامه به توضیح ماژول های موجود در هر دایرکتوری می پردازیم:

```
├── flight_tickets_scraper
│   ├── flight_tickets_scraper
│   │   ├── __init__.py
│   │   ├── items.py
│   │   ├── middlewares.py
│   │   ├── pipelines.py
│   │   ├── requests_method_enum.py
│   │   ├── settings.py
│   │   ├── spiders
│   │   │   ├── __init__.py
│   │   │   └── tcharter_airlines_tickets_spider.py
│   │   └── utils.py
│   ├── runner.py
│   └── scrapy.cfg
```

در ابتدا به توضیح ماژول های موجود در زیر پروژه **flight_tickets_scraper** می پردازیم. همانطور که قبلاً هم بیان کردیم این زیر پروژه وظیفه کروال کردن سایت هدف را برعهده دارد. همانطور که از عکس بالا مشخص است ماژول های این زیر پروژه عبارت اند از:

۱. **__init__**: این ماژول در هر جایی باشد موجب می شود که دایرکتوری مد نظر به صورت یک پکیج برای زبان برنامه نویسی پایتون دیده شود.

۲. **items.py**: قبل از اینکه داده های جمع آوری شده را ذخیره یا پردازش کنیم و مورد تغییر قرار دهیم مثل یک Container موقتی برای ذخیره داده های استخراج شده و تولید شده توسط اسپایدر عمل می کند.

۳. **middlewares.py**: این ماژول وظیفه تغییر در Request و Response ها را بر عهده دارد. درواقع قبل از اینکه Request شما به درون اینترنت و سایت هدف ارسال شود و یا قبل از اینکه Response به اسپایدر شما برسد توسط این ماژول می توانیم آن ها را دستکاری کنیم.

۴. **pipelines.py**: اگر می خواهیم تغییر اضافی بر روی داده استخراج شده که در قالب یک item می باشد انجام دهیم از این ماژول استفاده می کنیم و به طور کلی از این ماژول برای اعمالی مثل:

- Data Validation

- Cleaning

- Duplicate Checking

- Saving to database

استفاده می شود.

۵. **settings.py**: این ماژول همانطور که از اسمش هم پیداست کانفیگ ها و تمام تنظیمات کراولر ما در آن قرار دارد.

۶. **requests_method_enum.py**: از این ماژول برای تعریف انواع Request ها به صورت مقادیر شمارشی استفاده شده است.

۷. **tcharter_airlines_tickets_spider.py**: اسپایدر مربوطه ما برای استخراج داده ها از سایت هدف در این ماژول نوشته شده است

۸. **utils.py**: از این ماژول برای تعریف توابع کمکی و مشترک بین اسپایدرها در طول پروژه استفاده می کنیم.

۹. **runner.py**: این ماژول در زمان استفاده از قابلیت دیباگ کردن پروژه Scraping در IDE مربوطه کاربرد دارد.

۱۰. **scrapy.cfg**: تنظیمات فریمورک Scrapy برای اجرا کردن پروژه در این فایل قرار دارند.

حال به سراغ توضیح ماژول های موجود در زیر پروژه **flight_tickets_preprocessing** می رویم. این زیر پروژه وظیفه پیش پردازش کردن و تمیز کردن داده های خام ما را بعد از مرحله Scraping بر عهده دارد.

```

flight_tickets_preprocessing
├── __init__.py
├── airports_geographic_coordinate.py
├── make_processed_dataset.py.py
├── settings.py
├── tickets_preprocessing.py
└── utils.py

```

همانطور که از تصویر بالا مشخص است ماژول های زیر پروژه مربوطه عبارت اند از:

۱. **settings.py** : کانفینگ ها و تنظیمات سراسری مختص به این زیر پروژه در این فایل قرار دارند.
۲. **tickets_preprocessing.py** : ما در این ماژول توابع پیش پردازش خود را می نویسم تا از طریق آن ها داده های خام را تمیز کنیم. این فایل و محتوای آن فقط مختص به این پروژه می باشد زیرا محتوای درون آن متناسب با داده های جمع آوری شده برای تمیز و مرتب کردن آن نوشته است.
۳. **utils.py** : از این ماژول برای تعریف توابع کمکی، مشترک و پرکاربرد استفاده کرده ایم.
۴. **make_processed_dataset.py** : بعد از این توابع مورد نظر خود را در ماژول **ticket_preprocessing.py** نوشتیم در جهت اعمال بر روی دیتاست در اجرا آن ها را وارد این ماژول می کنیم. درواقع به مثل تابع **main** در **cpp** می ماند با این تفاوت که به جای تابع از ماژول استفاده کرده ایم. درواقع ما با اجرای این فایل تغییرات را روی دیتاست خود اعمال می کنیم.
۵. **airports_geographic_coordinate.py** : از این فایل برای نوشتن یک اسکریپت پایتونی در جهت پیدا کردن مختصات جغرافیایی هر فرودگاه به کمک یک **API** و ذخیره آن ها در فایل **airports_geometry.json** استفاده کرده ایم.

مابقی فایل ها ...

فایل هایی مخفی ای مثل **gitkeep** ، **gitignore** و غیره در پروژه ما وجود دارند و اما این فایل ها چه نقشی در پروژه ما دارند؟

۱. **gitignore** : در زمانی که نمیخواهیم یک سری فایل ها و دایرکتوری های ما وارد ریپوزیتوری گیت هابمان شوند از این فایل استفاده می کنیم و با مراجعه به محتوای این فایل به راحتی می توانید متوجه شوید که چه فایل ها و دایرکتوری نیاز نیست که وارد مخزن یا ریپو ما شوند.
۲. **gitkeep** : زمانی که در پروژه ما یک دایرکتوری وجود دارد ولی این دایرکتوری در حال حاضر خالی است اما بعداً از فایل و ماژول هایی پر می شود از این فایل استفاده می کنیم چرا که ابزار گیت به صورت پیش فرض نمی تواند دایرکتوری های خالی را وارد گیت هاب ما کند یک روش این می باشد که ما میتوانیم هر نوع فایلی حتی فایل **gitignore** را درون آن دایرکتوری خالی قرار دهیم تا بتوانیم آن دایرکتوری را به

داخل گیت هاب خود Push کنیم اما روش بهتری که همه از آن استفاده می کنند این است که به درون آن دایرکتوری فایل gitkeep را اضافه می کنند.

۳. **env** : این فایل مخفی به صورت پیش فرض در پروژه ایی که دست شماست وجود ندارد و شما باید خودتان آنرا در کنار دایرکتوری ریشه `mostafa_vahdani_bachelor_project` ایجاد کنید چرا که اطلاعات حیاتی مثل پسورد و لاغیر در آن ذخیره می شود که نباید در محیط پابلیک مثل گیت هاب قرار گیرند.

چگونه از پروژه اجرا بگیریم؟

نکته: در اینجا مراحل اجرا گرفتن را با جزئیات توضیح نمی دهیم چرا که اگر به سطحی رسیدید که دارید این داکيومنت را مشاهده و مطالعه می کنید پس جزئیات و چرایی مراحل را خودتان می دانید.

۱. یک محیط مجازی برای نصب کتابخانه ها در جهت اجرا گرفتن کد های پروژه با هر ابزاری که می شناسید ایجاد کنید می توانید از `virtualenv` و یا `virtualenvwrapper` استفاده کنید.

۲. از طریق Command Line خود به درون دایرکتوری `development` پروژه بروید و در آنجا قطعه کد زیر را اجرا کنید.

```
pip install -r requirements.txt
```

از قبل بررسی کنید اینترنت و همچنین VPN شما وصل باشد.

۳. به درون دایرکتوری `common_utils` در مسیر `source_codes/data` پروژه بروید و در آنجا قطعه کد زیر را اجرا کنید.

```
pip install .
```

اون نقطه رو اشتباهی نداشتم اونجا اون نقطه هم باید باشه وقتی کد رو کپی می کنی (:

۴. سرویس Tor را روی کامپیوتر خود بروی پورت ۹۰۵۰ و TorControl را بروی ۹۰۵۱ تنظیم کنید.

۵. سرویس Privoxy را بروی سیستم خود فعال کنید تا Scrapy از طریق آن بتواند به سرویس Tor متصل شود.

۶. مرحله آخر هم این هست که باید در کنار دایرکتوری ریشه پروژه فایل مخفی `env` رو ایجاد کنید و پارامترهای زیر رو طبق عکس بهش بدید.

```
OPENCAGEDATA_API_KEY=Your-API-Key  
SCRAPEOPS_API_KEY=Your-API-Key  
TOR_PASSWORD=Your-Tor-Password
```

نکته : توجه شود که برای گرفتن کلید API باید در سایت‌های OpenCageData و ScrapeOps ثبت نام کنید و کلید خودتون رو در پارامترهای بالا قرار دهید.

نکته : پارامترها دقیقاً باید به همان اسمی باشند که در تصویر می‌بیند پس پارامترها را به همین شکل نامگذاری کنید و به هیچ وجه آن‌ها تغییر ندهید.

فصل ۲

تکنولوژی ها و ابزارهای به کار رفته

تنها زبان برنامه نویسی و تکنولوژی به کار رفته در این پروژه زبان برنامه نویسی قدرتمند و بسیار ساده پایتون می باشد از این زبان برای تمامی فازهای پروژه استفاده شده است. زبان برنامه نویسی پایتون از ماژول ها (modules) و بسته ها (packages) استفاده می کند، بدین معنا که برنامه های این زبان قابل طراحی به سبک ماژولار (modular) هستند و کدهای نوشته شده در یک پروژه در پروژه های گوناگون دیگر نیز قابل استفاده مجدد محسوب می شوند. هنگامی که کاربری ماژول یا بسته مورد نیاز خود را توسعه داد، خودش یا دیگر علاقمندان (در صورتی که کد در اختیار عموم قرار بگیرد) می توانند آن را برای استفاده در دیگر پروژه ها گسترش دهند. در این پروژه نیز از کتابخانه، فریمورک و ابزارهای Built-in و External متفاوتی به صورت مشترک و یا مستقل استفاده شده است اگر به دایرکتوری development مراجعه کنید فایل به نام requirements.txt را مشاهده خواهید کرد که محتوای آن که در عکس زیر موجود است و نشان دهنده موارد External می باشد که در پروژه وجود دارند.

```
1 pip==23.2.1
2 pipdeptree==2.12.0
3 autopep8==2.0.2
4 chardet==3.0.4
5 hazm==0.9.3
6 numpy==1.25.2
7 matplotlib==3.7.2
8 PyQt5==5.15.9
9 pandas==2.0.3
10 scikit-learn==1.3.0
11 mlxtend==0.22.0
12 seaborn==0.12.2
13 Scrapy==2.9.0
14 stem==1.8.2
15 cookiecutter==2.3.0
16 python-dotenv==1.0.0
17 opencage==2.3.0
18 requests==2.31.0
19 pytz==2023.3
20 jdatetime==4.1.1
21 haversine==2.8.0
22
```

همانطور که در عکس بالا مشاهده می کنید از موارد بالا در پروژه استفاده کرده ایم در ادامه نیز راجبه استفاده هرکدام توضیحاتی را ارائه می دهیم و حتماً به این نکته نیز توجه کنید برای اجرا گرفتن پروژه حتماً باید موارد بالا بر روی سیستم شما همانطور که در بخش قبلی گفته شد نصب شده باشند.

کتابخانه‌های داخلی زبان پایتون

۱. **کتابخانه os** : ماژول os در پایتون راهی برای تعامل با سیستم عامل ارائه می دهد که به شما امکان می دهد تسک های مختلف مربوط به سیستم را انجام دهید.
۲. **کتابخانه sys** : ماژول sys در پایتون دسترسی به پارامترها و توابع مختلف سیستم را فراهم می کند. معمولاً برای تعامل با Python runtime environment و سیستم عامل استفاده می شود.
۳. **کتابخانه itertools** : ماژول itertools در پایتون مجموعه ای قدرتمند از توابع است که ابزارهای مختلف سریع و کارآمدی را برای کار با Iterator ها ارائه می دهد. بخشی از کتابخانه استاندارد پایتون است و معمولاً برای ایجاد با Iterator برای لوپ زدن کارآمد و دستکاری داده ها استفاده می شود.
۴. **کتابخانه pathlib** : ماژول pathlib در پایتون یک رویکرد شی گرا برای کار با مسیرها و دایرکتوری های سیستم فایل ارائه می دهد.
۵. **کتابخانه enum** : شمارش (enum) مجموعه ای از نام های نمادین (اعضا) است که به مقادیر ثابت و منحصر به فرد محدود می شوند. Enum ها با دادن نام های معنی دار به ثابت ها، راهی برای ایجاد کد قابل خواندن و نگهداری بیشتر ارائه می دهند.
۶. **کتابخانه random** : به شما امکان می دهد با اعداد تصادفی کار کنید.
۷. **کتابخانه time** : این ماژول توابع مختلف مرتبط با زمان را ارائه می دهد.
۸. **کتابخانه urllib** : کتابخانه urllib در پایتون ماژولی برای کار با URL ها است. و چندین ماژول برای کار با URL ها و ایجاد درخواست های HTTP فراهم می کند.
۹. **کتابخانه json** : پایتون کتابخانه داخلی را برای کار با فایل هایی با فرمت JSON از طریق ماژول json فراهم می کند.

کتابخانه و فریمورک های مشترک

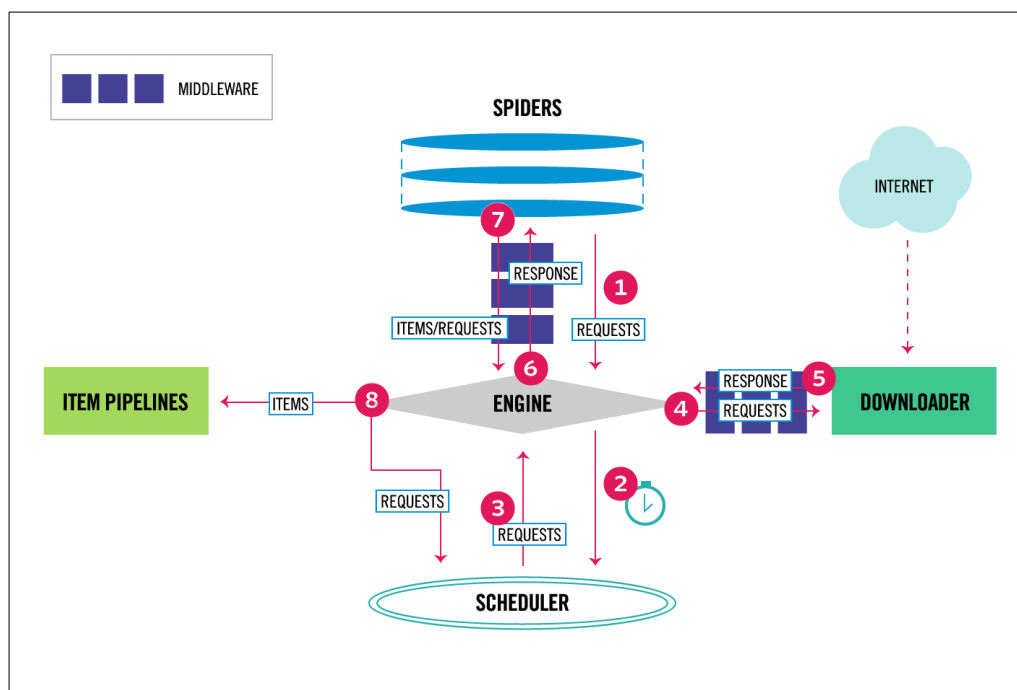
۱. **پکیج autopep8** : یک ابزار کامند لاینی و کتابخانه پایتونی است که به طور خودکار کد پایتون را برای مطابقت با راهنمای سبک PEP-8 پایتون (راهنمای طریقه نوشتن کد تمیز پایتون) قالب بندی می کند. این به حفظ استایل کد یکنواخت کمک می کند و کد شما را خواناتر و قابل نگهداری تر می کند.
۲. **پکیج یا کتابخانه pipdeptree** : این ابزار کامند لاینی به نمایش پکیج های پایتونی نصب شده در سیستم شما کمک می کند. هرچند که این ابزار در روند اصلی پروژه و ایجاد خروجی تأثیری ندارد ولی صرفاً به دلیل استفاده از آن تصمیم گرفتیم آنرا در لیست قرار دهیم.
۳. **پکیج pip** : این پکیج برای نصب پکیج های نرم افزار مثل کتابخانه ها و فریمورک های زبان برنامه نویسی پایتون می باشد.

۴. کتابخانه **python-dotenv** : این کتابخانه از یک فایل مخفی به نام env که توسط شما هم باید ایجاد شود جفت‌های کلید-مقدار را می‌خواند و آن‌ها را در Environment Variables سیستم شما قرار می‌دهد تا بتوانید از این عمل برای افزایش امنیت و تمیزی کدهای کد استفاده کنید.

۵. کتابخانه **common_utils** : این یک پکیج خیلی ساده است که ابزارهای درون آن توسط بنده بر روی پکیج json جمع آوری شده است و در حال حاضر همانطور که از اسم پکیج مشخص است صرفاً در آن یکسری تابع پرکاربرد در انواع پروژه‌ها برای کار با فایل و آبجکت‌های JSON قرار دارد.

کتابخانه‌ها و فریمورک‌های زیر پروژه **flight_tickets_scraper**

- فریمورک **scrapy** : مواردی مثل Web Scraping و Web Crawling دو مورد از مهم‌ترین موارد بررسی و تحلیل صفحات وب هستند. در سال ۲۰۰۸ فریمورکی تحت عنوان Scrapy انتشار یافت که برای تازه کاران ساده و برای حرفه‌ای‌ها به اندازه کافی دارای ویژگی‌های مثبت است. وبسایت‌ها حاوی اطلاعات معنی‌دار هستند که می‌توانیم آن‌ها را استخراج کنیم و در کاربرد مورد نظرمان (طراحی موتور جستجو یا ایجاد یک دیتاست برای کار تحلیل داده یا ...) از این داده‌ها بهره ببریم. از Scrape کردن می‌توان برای جمع‌آوری محتویات خام از صفحات وب و ذخیره کردن آنها برای استفاده بعدی در یک قالب ساختارمند و معنادار استفاده کرد. بهتر از یک نگاه به معماری که این فریمورک از الان پیروی می‌کند هم نگاهی بی‌اندازیم که چرا که در بخش توضیح کدها بسیار با این فریمورک درگیر خواهیم شد. معماری این فریمورک قدرتمند به شرح عکس زیر می‌باشد:



- عکسی که مشاهده می‌کنید نشان می‌دهد که Engine مهم‌ترین بخش این معماری می‌باشد چرا که به گفته داکيومنت وظیفه کنترل جریان دیتاهای عبوری کل سیستم (جا به جا کردن Request و

Response بین تمام کامپوننت ها و تحویل دادن Item ها) و همچنین تحریک کردن Event ها را بر عهده دارد. رکیوئست شما در ماژول اسپایدر مد نظرتان تولید می شود و سپس این رکیوئست ها توسط Engine در Scheduler زمانبندی شده تا Engine بتواند دوباره از اسپایدر Request بعدی را تحویل بگیرد در ادامه Request زمان بندی شده دوباره به Engine داده می شود تا آن را به Downloader منتقلش کند. وظیفه Downloader این می باشد که Request شما را به درون اینترنت ارسال کند، صفحات وب دریافتی را fetch کند و در نهایت Response را در اختیار Engine قرار دهد. Engine نیز Response دریافتی را توسط Middleware ها جهت parse شدن به اسپایدر بر می گرداند و در پایان اسپایدر نیز داده های استخراج شده را با ساختن یک Item و yield کردن، توسط Engine به Pipeline ها منتقل می کند به طور مختصر Pipeline ها وظیفه پردازش Item ها را بر عهده دارند.

- **کتابخانه stem** : این کتابخانه دارای ابزاری برای تعامل با شبکه تور می باشد. با استفاده از این کتابخانه و ابزارهایی که برای شما فراهم آورده است می توانید اسکریپت هایی را بنویسید که instance های شبکه تور را کنترل و اطلاعات لازمه را از relay و exit node ها و مدارهای ایجاد شده دریافت کند.

- **پکیج chardet** : کتابخانه chardet یک کتابخانه پایتونی است که برای تشخیص Encoding کاراکتر استفاده می شود. با تجزیه و تحلیل بایت ها و الگوهای آماری موجود، به تعیین Encoding کاراکتر یک متن یا فایل معین کمک می کند. این امر به ویژه هنگام کار با داده های متنی که ممکن است از منابع مختلف آمده و دارای رمزگذاری های متفاوت باشند، مفید است.

- **کتابخانه requests** : یک پکیج محبوب برای ایجاد درخواست های HTTP می باشد که فرآیند ارسال درخواست های HTTP و رسیدگی به پاسخ ها را ساده تر می کند. به طور گسترده برای کارهایی مانند واکنشی داده ها از API ها، دانلود صفحات وب و تعامل با سرویس های وب استفاده می شود.

کتابخانه ها و فریمورک های زیر پروژه flight_tickets_preprocessing

- **کتابخانه hazm** : کتابخانه hazm یک پکیج پایتون است که برای پردازش متن فارسی طراحی شده است. عملکردهای مختلفی را برای نرمال سازی متن، ریشه یابی، واژه سازی، نشانه گذاری، برچسب گذاری بخشی از گفتار و موارد دیگر فراهم می کند. این یک ابزار ارزشمند برای پردازش زبان طبیعی وظایف مربوط به زبان فارسی است.

- **کتابخانه matplotlib** : یک کتابخانه پایتونی است که به طور گسترده برای ایجاد شکل و نمودار های بصری ایستا، تعاملی و متحرک در پایتون استفاده می شود. این کتابخانه برای مصورسازی و تحلیل داده ها بسیار کارآمد می باشد.

- **کتابخانه pandas** : یک کتابخانه OpenSource پایتونی است که برای دستکاری و تجزیه و تحلیل داده ها استفاده می شود. ساختارهای داده ایی و توابع مورد نیاز برای دستکاری کارآمد داده های ساخت یافته را فراهم می کند. pandas به طور گسترده در علم داده، یادگیری ماشین و سایر زمینه های مرتبط با داده استفاده می شود.
 - **کتابخانه PyQt5** : یک انقیاد پایتونی برای فریمورک Qt است که به توسعه دهندگان اجازه می دهد تا برنامه های کراس پلتفرمی را با رابط کاربری گرافیکی ایجاد کنند. به طور گسترده ای برای ساخت برنامه های دسکتاپ که بر روی سیستم عامل های مختلف از جمله ویندوز، مک و لینوکس اجرا می شوند، استفاده می شود. در این پروژه صرفاً ما این پکیج را نصب می کنیم تا بتوانیم نمودارهای خود را به نمایش در آوریم.
 - **کتابخانه scikit-learn** : یک کتابخانه یادگیری ماشین OpenSource بسیار پرکاربرد و قدرتمند پایتونی است. ابزارهای ساده و کارآمدی برای داده کاوی و تجزیه و تحلیل داده ها فراهم می کند. در یک ساختار انتزاعی با تکیه بر دیگر ابزارها بر روی کتابخانه های محبوب دیگر مانند NumPy، SciPy و Matplotlib ساخته شده است. Scikit-Learn برای کار با وظایف مختلف یادگیری ماشین، از جمله طبقه بندی، رگرسیون، خوشه بندی، کاهش ابعاد و غیره طراحی شده است و در این پروژه نیز بسیار به ما کمک خواهد کرد.
 - **ابزار cookiecutter** : اگر مثل بنده همیشه به دنبال این باشید که پروژه شما باید یک ساختار منظم و مرتب داشته باشد تا هم خودتان هم بقیه که بعدها کد شما را می خوانند بدانند دقیقاً کجا دنبال چه چیزی بگردند متوجه می شوید در این موضوع پروژه های دیتاساینس یک ضعف بزرگ دارند و همچنین یک سردرد بزرگ برای شما هستند چرا که اصلاً و به هیچ وجه یک ساختار منظم و واحد بر خلاف پروژه های دیگر مثل پروژه های طراحی بک اند با django یا طراحی یک کراولر با scrapy ندارند و با شروع هر پروژه دیتاساینس جدید شما ساختار قبلی را فراموش می کنید و یک ساختار جدید را برای کد های پروژه خود ایجاد می کنید. اما با مقداری سرچ متوجه خواهید شد که پروژه هایی مثل cookiecutter وجود دارند که به راحتی به شما امکان ایجاد یک Template برای پروژه دیتاساینس یا هر پروژه دیگری را به شما خواهند داد و این مورد برای ما فوق العاده خواهد بود.
- Cookiecutter یک ابزار کامند لاینی است که به شما امکان می دهد پروژه ها را در Template های متفاوت ایجاد کنید که در آن شما با یک الگو شروع می کنید، و این ابزار به شما کمک می کند یک ساختار پروژه را بر اساس آن الگو ایجاد کنید. این به ویژه زمانی مفید است که می خواهید یک پروژه جدید راه اندازی کنید که از یک ساختار دایرکتوری خاص پیروی می کند، شامل فایل های خاصی است و دارای تنظیمات از پیش تعریف شده می باشد.
- گفتنی است که ما دقیقاً نمیخواهیم دوباره شروع کنیم چرخ را اختراع کرده و ساختار پروژه های دیتاساینس را تعریف کنیم چرا که قبلاً یکی اینکار رو برای ما انجام داده (این است زیبایی دنیای نرم

افزار) اگر مقدار بیشتر سرچ کنیم متوجه می‌شویم پروژه‌ای به نام CookieCutter Data Science وجود دارد که می‌توانیم از آن بهره ببریم.

- **ابزار Cookie Cutter Data Science** : یک الگوی پروژه است که به شما کمک می‌کند پروژه‌های علم داده خود را به شیوه‌ای سازگار و سازمان‌یافته ساختار دهید. این یک ابزار کامند لایینی است که به شما امکان می‌دهد با پیروی از یک ساختار دایرکتوری از پیش تعریف شده و پر کردن آن با فایل‌های ضروری، الگوهای پروژه خود را ایجاد کنید. این می‌تواند در زمان شروع پروژه‌های جدید علم داده در زمان و تلاش شما صرفه جویی کند، زیرا مجبور نخواهید بود که ساختار و فایل‌های مشابه را به طور مکرر تنظیم کنید.
- **کتابخانه opencage** : ما در ادامه درون پروژه خود از API سایت OpenCage برای جمع‌آوری مختصات جغرافیایی هر فرودگاه استفاده خواهیم کرد این سایت برای راحتی استفاده برنامه نویسی‌ها از API خودش یک کتابخانه ارائه کرده که می‌توانیم از آن بهره ببریم.
- **کتابخانه mlxtend** : یک کتابخانه پایتونی است که توسط سباستین راشکا توسعه یافته است که بر ارائه مجموعه‌ای از ابزارها و برنامه‌های افزودنی برای ارتقای قابلیت‌های Sikit-Learn و دیگر کتابخانه‌های یادگیری ماشین تمرکز دارد. این شامل پیاده‌سازی الگوریتم‌های مختلف یادگیری ماشین، معیارهای ارزیابی، و توابع کاربردی است که می‌تواند هم برای مبتدیان و هم برای متخصصان با تجربه مفید باشد.
- **کتابخانه seaborn** : یک کتابخانه است که بر روی matplotlib ساخته شده است. برای مصورسازی داده‌ها و تجزیه و تحلیل داده‌های اکتشافی استفاده می‌شود. Seaborn به راحتی با دیتافریم‌ها و کتابخانه pandas کار می‌کند. نمودارهای ایجاد شده را نیز می‌توان به راحتی شخصی‌سازی کرد.
- **کتابخانه pytz** : قابلیت کار با مناطق زمانی مختلف را فراهم می‌کند. به شما امکان می‌دهد تاریخ را محلی‌سازی کنید، مناطق زمانی مختلف را به هم تبدیل کنید. این کتابخانه به ویژه در هنگام برخورد با محاسبات مربوط به نمایش تاریخ و زمان در مناطق زمانی مختلف مفید است.
- **کتابخانه haversine** : فرمول haversine برای محاسبه فاصله بین دو نقطه روی سطح یک کره مانند زمین استفاده می‌شود. این فرمول اغلب در جاهای استفاده می‌شود که شامل موقعیت جغرافیایی و محاسبه فاصله بین مختصات می‌شود. پکیج haversine روشی ساده برای محاسبه فاصله بین دو مجموعه مختصات با استفاده از فرمول haversine ارائه می‌دهد. این می‌تواند برای برنامه‌هایی که شامل محاسبه فاصله بین نقاط جغرافیایی هستند مفید باشد.
- **کتابخانه numpy** : یک کتابخانه قدرتمند در پایتون برای عملیات عددی و ریاضی است. پشتیبانی از آرایه‌ها و ماتریس‌ها، همراه با طیف وسیعی از توابع ریاضی برای کار بر روی این آرایه‌ها را فراهم می‌کند. Numpy یک پکیج اساسی برای محاسبات علمی با پایتون است.

- **کتابخانه jdatetime** : کتابخانه jdatetime به شما امکان می دهد با تاریخ و زمان جلالی (فارسی) در پایتون کار کنید. قابلیت هایی برای تبدیل، فرمت کردن و دستکاری تاریخ های جلالی فراهم می کند.

علاوه بر موارد گفته شده ابزارهای دیگری نیز به کمک ما برای توسعه کدهای پروژه آمده اند. البته باید گفت که این ابزار در کد نویسی پروژه تأثیر غیر مستقیم داشته اند.

ابزارهای مرتبط استفاده شده در طول پروژه

۱. **Git** : یک سیستم کنترل نسخه توزیع شده است که به توسعه دهندگان کمک می کند تغییرات را در کد های خود پیگیری و دنبال کرده و با دیگران همکاری کنند تا تاریخچه پروژه را به طور موثر مدیریت کنند. این نرم افزار توسط لینوس توروالدز در سال ۲۰۰۵ ایجاد شد و از آن زمان به یک ابزار اساسی در توسعه نرم افزار تبدیل شده است.

۲. **Tor Service** : با فعال کردن این سرویس در سیستم خود شما به یک شبکه ایی از اتصال نظیر به نظیر گره های متصل بهم دسترسی دارید. مسیریابی ترافیک اینترنتی شما از طریق یک شبکه ای از سرورهای داوطلب به نام گره ها انجام می شود. هر گره در شبکه Tor لایه ای از رمزگذاری را به داده های شما اضافه می کند و ردیابی به مکان یا هویت اصلی شما را دشوار می کند. استفاده ما از تور در پروژه به عنوان یک پروکسی رایگانی است که در هر چند ثانیه IP ما را در اینترنت عوض می کند. هر چند اگر می خواهید از تور در پروژه های Scraping دیگر خود استفاده کنید توجه کنید که باید سرور مقصد یا همان سایت هدف باید از exit node های شبکه تور پشتیبانی کند.

۳. **Privoxy** : یک Web Proxy با قابلیت های فیلترینگ پیشرفته برای افزایش حریم خصوصی، دستکاری کوکی ها، کنترل دسترسی و موارد دیگر است. اغلب در ترکیب با ابزارهای دیگر مانند Tor برای ارائه یک لایه اضافی از ناشناس بودن و امنیت استفاده می شود. Privoxy به عنوان یک واسطه بین کامپیوتر شما، نرم افزار مربوطه با شبکه تور و اینترنت عمل می کند و محتوا و درخواست ها را با توجه به کانفیگ های آن فیلتر می کند. ما در پروژه خود برای اتصال Scrapy به سرویس Tor و سپس انتساب IP که Relay های Tor در اختیارمان قرار می دهند به Request ها به سایت هدف بهره برده ایم. به همین دلیل به Privoxy نام Intermediate Proxy نیز داده اند.

۴. **Postman** : نرم افزاری با رابط گرافیکی است که به ما امکان می دهد تست API را انجام دهیم. مانند مرورگری است که html را رندر نمی کند. در مرورگر می توانیم فقط GET HTTP Request ارسال کنیم، اما در اینجا می توانیم GET، POST، PUT، PATCH، DELETE و بسیاری دیگر از درخواست های HTTP را ارسال کنیم. در postman ما می توانیم درخواست را ذخیره کرده و مجموعه ای از پوشه ها را برای سازماندهی Request ها ایجاد کنیم. کاربرد ما از این نرم افزار در این پروژه در بخش هایی مثل بررسی

Response های درخواست های Ajax سایت هدف، استفاده از API ها و همچنین کمک به نوشتن اسکریپت پایتونی برای API ها و یک کراولر چند سطحی بوده است.

۵. **Jupyter** : نوت بوک های Jupyter به طور گسترده برای تجزیه و تحلیل داده ها، یادگیری ماشینی، تحقیقات علمی و آموزش استفاده می شود. این نوت بوک از زبان برنامه نویسی Python و R پشتیبانی می کند و یکی از قابلیت هایی این نوت بوک که از آن در پروژه استفاده کردیم قابلیت Interactive Code Execution می باشد که به ما این اجازه را می دهد هر خط کد را به صورت جداگانه اجرا کنیم و نتیجه آن را به صورت inline مشاهده کنیم.

API های استفاده شده در پروژه

۱. **OpenCageAPI** : یک سرویس geocoding است که روشی آسان برای تبدیل مختصات جغرافیایی (طول و عرض جغرافیایی) به اطلاعات مکان قابل خواندن توسط انسان و بالعکس ارائه می دهد. معمولاً در برنامه هایی که به خدمات موقعیت جغرافیایی نیاز دارند استفاده می شود.

۲. **ScrapOpsAPI** : سرویسی است که به شما این امکان را می دهد تا پارامترهای موجود در Web Scraping یک سایت را کنترل کنید. این سرویس برای ساده سازی و خودکارسازی اجزای عملیات Web Scraping طراحی شده است و جمع آوری داده ها از وب سایت ها را برای توسعه دهندگان برای اهداف مختلف آسان تر می کند. یکی از امکاناتی که این API در اختیارمان قرار می دهد این می باشد که برای ما FakeUserAgent تولید می کند تا سایت هدف Request ما را مثل رکیوئستی در نظر بگیرد که از یک مرورگر واقعی در حال ارسال شدن است به عبارتی به کمک این API می توانیم خودمان را جای یک کاربر واقعی جا بزنیم تا سایت هدف به ربات بودن ما شک نکند.

فصل ۳

عملکرد و توضیح کد ها

تا به اینجای کار به معرفی پروژه و اهداف پرداختیم و در رابطه با ساختار دایرکتوری ها و ماژول های پروژه، ساختار دیتاست، تکنولوژی و ابزارهای به کار رفته در پروژه و همچنین طریقه اجرا کردن کدها توضیحاتی را به شما ارائه دادیم. حال میخواهیم کدهای نوشته شده را به صورت فنی و تکنیکی مورد بررسی قرار دهیم تا متوجه شویم پروژه نوشته شده در جزئیات به چه شکل کار می کند بدین طریق شما خواننده گرامی بتوانید تغییرات دلخواه را در جهت بهبود پروژه به آن اضافه کنید. ترتیب توضیح دادن کدها به ترتیب فازهای پروژه است که در ابتدای بخش معرفی پروژه به شما توضیح دادیم.

نکته : ممکن است در عکس های تهیه شده کدهایی را مشاهده کنید کامنت شده اند، بنده در مورد آن کدها توضیحاتی ارائه نمی دهم زیرا که واضح است آن کدهای کامنت شده تأثیری در روند برنامه ندارند. و ممکن است کدهای تولید شده توسط خود فریمورک باشند که ما در حال حاضر از آن ها در پروژه استفاده نکرده ایم.

فاز اول پروژه

کراول کردن سایت هدف در جهت جمع آوری یک دیتاست خام که در این پروژه از سایت `tcharter.ir` که یک موتور جستجو بلیت های هواپیما می باشد استفاده کرده ام.

ماژول `settings.py` :

- `load_dotenv()`

این خط کد تمام کانفیگ های موجود در فایل `env` را در `Python Environments Variable` برای ما لود می کند و اجازه دسترسی به متغیرهای محیطی تعریف شده را به ما می دهد.

- `BASE_PATH = pathlib.Path(file).parent.parent.parent`

ذخیره کردن آدرس دایرکتوری `data` درون دایرکتوری `source_codes` از این آدرس به عنوان آدرس پایه برای دسترسی به دایرکتوری های زیر مجموعه دایرکتوری `data` مثل دایرکتوری `static` استفاده می شود. درواقع با قطعه کد `pathlib.Path(file)` به آدرس مسیر فعلی فایل `setting.py` دسترسی پیدا می کنیم و با هر `parent` یک دایرکتوری به عقب برمی گردیم تا به دایرکتوری `data` برسیم.

- `BOT_NAME = "flight_tickets_scraper"`

فریمورک اسکریپت موقع اجرا ربات کراولر ما را با ذخیره شده در متغیر `BOT_NAME` می شناسد که با اسم زیر پروژه بخش `Scraping` یکی می باشد.

- `SPIDER_MODULES = ["flight_tickets_scraper.spiders"]`

اسکریپت برای پیدا کردن اسپایدر های نوشته شده توسط شما در این لیست به دنبالش می گردد.

- `NEWSPIDER_MODULE = "flight_tickets_scraper.spiders"`

ماژول پیش فرضی که اسپایدرهای جدید شما در آن تعریف می شود در این متغیر قرار می گیرند.

- **ROBOTSTXT_OBEY = False**

اگر می‌خواهید از robot.txt سایت هدف پیروی کنید مقدار این متغیر باید به True تغییر پیدا کند (مقدار پیش‌فرض این متغیر True می‌باشد) این فایل مشخص می‌کند ربات‌های خزنده به چه مسیرهایی در سایت می‌توانند با چه محدودیتی دسترسی داشته باشند. در این پروژه بنده مقدار آنرا برابر با False قرار دادم زیرا که در حین درخواست به سایت هدف، سایت ما را با کد ۳۰۲ به URL اصلی tcharter.ir هدایت می‌کرد با تغییر دادن این متغیر به False دیگر این اتفاق نمی‌افتد.

- **DOWNLOAD_DELAY = 3.75**

مقدار تأخیر در ارسال هر Request را با این متغیر مشخص می‌کنیم که ما مقدار آنرا با آزمون خطا‌های متعدد ترجیه دادیم روی ۳.۷۵ ثانیه تنظیم کنیم.

- **RANDOMIZE_DOWNLOAD_DELAY = True**

MinSec = 0.5 * DOWNLOAD_DELAY

MaxSec = 1.5 * DOWNLOAD_DELAY

[MinSec, MaxSec]

با قرار دادن True در این متغیر میزان تأخیر در ارسال درخواست به سایت هدف را تصادفی می‌کنیم که مقدار آن در بازه [MinSec, MaxSec] تصادفی تغییر می‌کند.

- **DEFAULT_REQUEST_HEADERS**

س در بعضی مواقع هدرهایی که خود اسکریپت به صورت پیش‌فرض برای ما می‌سازد تا پارامترهای مورد نیاز برای ارسال رکویست را پر کند توسط سایت هدف پذیرفته نمی‌شود و ممکن است که وقتی ما را به عنوان یک ربات شناسایی می‌کند Reponse مورد انتظار ما را به ما برنگرداند در نتیجه می‌توانیم از این پارامتر برای ارسال تمام رکویست‌های خود استفاده کنیم.

```
# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
    "Accept": "*/*",
    "Accept-Language": "en-US,en;q=0.9",
    "Connection": "keep-alive",
    "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
    "Cookie": "TCH=8ktqu1bpqr7u74p2tb5kchrnl; _gcl_au=1.1.849212697.1690208445; _ga=GA1.2.1776701195.1690208446; _gid=GA1.2.2026748026.1690208446; GoReferer=Q2FrZQ%3D%3D.ZmZiOTljNGYzNzQ3NjI1MGRmYwVkyZA5ZTk4ZmExYWUzMzBmMzI0MzNmYzExNwNmYzQ1NTgxMzIwNmI1Y2RiZm%2FM%2FF0kD4jQwXnN8awOIamkxmDFlfNLQsSJITEMiAr5rrwF10W6JUmjVjLZVW4RBLdf0T1a5iDpK5IFUKOV5puSILH36ld4ZUhr8KP7csFHVrSa%2FCeSqAPqe7MU9pYxK0vjMGJcq0sbVioImH%2FeHYSJJqoDse6ZJhuFUMXqb9VPgPufuY%2BN4UD%2BK9yx%2FJ0i0CiAIungjXbNkomZY2cdLtt0QV7cDsUTmJVdFW4ornQK005zt%2F42TrQMjD5s6H0pWaMeifvuFITE9vXcQJz0NuYPegFGweJnLjuukpM3wyy28XU69o0qrw5cLnC1pDCt%2BAzM0%2F%2BIeolk%2BIaIdeL0tiwJICR658eCz%2BHXCjb13jx37tC1qLdspCFcdag%2BlfwKc6w%3D%3D; _ga_R30088QYSJ=GS1.2.1690484040.21.0.1690484040.60.0.0",
    "DNT": "1",
    "Origin": "https://www.tcharter.ir",
    "Referer": "https://www.tcharter.ir/",
    "Sec-Fetch-Dest": "empty",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Site": "same-origin",
    "X-Requested-With": "XMLHttpRequest",
    "sec-ch-ua": "'Not/A)Brand';v='99', 'Google Chrome';v='115', 'Chromium';v='115'",
    "sec-ch-ua-mobile": "?0",
    "sec-ch-ua-platform": "'Linux'",
}
```

با توجه به عکس بالا می بینید که ما برای پارامتر گفته شده مقادیر بالا را در نظر گرفته ایم باید توجه کرد که این مقادیر، مقادیر من درآوردی نیست و توسط Developer Tools مرورگر بخش Network با توجه به رکویست مد نظر پیدا شده و صرفاً مقدار پیدا شده را کپی کردیم و در این متغیر قرار دادیم. تا سایت ما از طریق هدر به عنوان ربات شناسایی نکند.

- **DOWNLOADER MIDDLEWARES = {**

'flight_tickets_scraper.middlewares.ScrapeOpsFakeUserAgentMiddleware': 400,

'flight_tickets_scraper.middlewares.TorMiddleware': 610,

}

در این پارامتر Middleware هایی که برای دستکاری یا ریسپانس می نویسیم را قرار می دهیم توجه شود عدد هایی که به هر Middleware ها انتساب می دهیم درواقع ترتیب اجرا شده آن ها توسط اسکریپ می باشد.

- **ITEM PIPELINES = {**

"flight_tickets_scraper.pipelines.DuplicatesItemsPipeline": 300,

"flight_tickets_scraper.pipelines.TicketArrivalTimePipeline": 800,

}

در این پارامتر هم pipline هایی که نوشتیم را با انتساب عددهایشان برای اجرا قرار می دهیم.

- **SOURCES = {**

"airport_city_codes_path": BASE_PATH.joinpath("static", "airport_city_codes.json"),

}

توسط این متغیر و همچنین به کمک متغیر `BASE_PATH` می‌توانیم آدرس سورس‌هایی که در طول پروژه احتمال استفاده از آن‌ها وجود دارد را مشخص می‌کنیم و با فرمت کلید-مقدار به آن‌ها دسترسی داشته باشیم. این قطعه کد به خوانا تر و منعطف تر شدن کد ما کمک شایانی خواهد کرد زیرا که هم کد ما را داینامیک تر می‌کند و اگر بعدها آدرس سورس تغییر کرد فقط از اینجا آنرا تغییر می‌دهیم و لازم نیست چندین جای کد را دست‌بزنیم و هم اینکه نیاز نیست هر جایی از پروژه از آدرس مطلق فایل سورس دیتا استفاده کنیم.

- **`SCRAPEOPS_API_KEY = os.getenv("SCRAPEOPS_API_KEY")`**

تابع `getenv` برای دسترسی به متغیر محیطی سیستم استفاده می‌شود که در مورد قطعه کد بالا مشخص است که ما می‌خواهیم به مقدار کلید `API` ای که قبلاً راجبه آن توضیح دادیم دسترسی داشته باشیم و آنرا در متغیر ذکر شده ذخیره کنیم. (بعداً از این متغیر در `FakeUserAgent Middleware` استفاده می‌کنیم).

- **`SCRAPEOPS_FAKE_USER_AGENT_ENABLED = True`**

برای اینکه بتوانیم از `FakeUserAgent Middleware` استفاده کنیم علاوه بر اینکه باید این `Middleware` را درون `Download_Middleware` قرار دهیم باید مقدار این متغیر را طبق داکيومنت سایت `ScrapeOps` به `True` تغییر دهیم.

- **`INTERMEDIATE_PROXY = 'http://127.0.0.1:8118'`**

در متغیر آدرس دسترسی به سرویس `Privoxy` که بر روی سیستم خود فعال کردیم را قرار می‌دهیم تا `scrapy` بتواند به سرویس `Tor` دسترسی داشته باشد. (بعداً از این متغیر در `TorMiddleware` استفاده می‌کنیم).

- **`IP_CHECKER_SITE = "http://icanhazip.com"`**

از این متغیر استفاده می‌کنیم تا آدرس سایتی را در آن قرار دهیم که خروجی سایت نمایش آدرس `IP` فعلی سیستم ما می‌باشد. (بعداً از این متغیر در `TorMiddleware` استفاده می‌کنیم).

- **`TOR_PASSWORD = os.getenv("TOR_PASSWORD")`**

پسورد سرویس تور را می‌توانیم از طریق این کد بخوانیم و در متغیر ذکر شده قرار دهیم. (بعداً از این متغیر در `TorMiddleware` استفاده می‌کنیم).

- **`TOR_RANDOMIZE_CHANGING_IP_DELAY = True`**

`MinSec = 0.5 * TOR_CHANGING_IP_DELAY_SEC`

`MaxSec = 1.5 * TOR_CHANGING_IP_DELAY_SEC`

`[MinSec, MaxSec]`

با `True` قرار دادن مقدار این متغیر می‌توانیم موجب تغییر `IP` تور در بازه‌های زمانی (ثانیه) متفاوت به صورت رندوم انجام شویم. (بعداً از این متغیر در `TorMiddleware` استفاده می‌کنیم).

- **`TOR_CHANGING_IP_DELAY_SEC = 120`**

مقدار تأخیر در ارسال هر `Request` برای تغییر `IP` را با این متغیر مشخص می‌کنیم. (بعداً از این متغیر در `TorMiddleware` استفاده می‌کنیم).

- **TOR_CONTROL_PORT = 9051**

تنظیم پورت سرویس کنترل تور از طریق این متغیر انجام می شود. (بعداً از این متغیر در TorMiddleware استفاده می کنیم.)

- **"REQUEST_FINGERPRINTER_IMPLEMENTATION = "2.7"**

مشخص می کند که باید از کدام اسکریپت باید از کدام الگوریتم Request Fingerprinting استفاده کند.

- **TWISTED_REACTOR = "twisted.internet.asyncioreactor.AsyncioSelectorReactor"**

اسکریپت به صورت Async کار می کند و برای این امر از ری اکتور Twisted استفاده می کند.

- **FEED_EXPORT_ENCODING = "utf-8"**

با تنظیم این پارامتر اسکریپت از UTF-8 Encoding استفاده می کند.

ماژول items.py :

• کلاس FlightTicketsScrapperItem

```
class FlightTicketsScrapperItem(Item):
    ticket_id = Field()

    # going out of source airport
    national_departure_code = Field()

    # coming into the destination airport
    national_arrival_code = Field()

    arrival_city_name_persian = Field()
    departure_city_name_persian = Field()

    departure_time = Field()
    arrival_time = Field()
    departure_date = Field()
    departure_date_YMD_format = Field()

    company_name = Field()
    capacity = Field()
    flying_number = Field()
    flying_class = Field()
    ticket_price_T = Field()
    flying_type = Field()

    def set_all_default_value(self, value=None):
        for keys, _ in self.fields.items():
            self[keys] = value
```

همانطور که در تصویر بالا مشاهده می کنید یک کلاس به اسم FlightTicketsScraperItem تعریف کرده ایم که از کلاس والدی به نام Item که توسط فریمورک Scrapy نوشته شده ارث بری کرده است. Class Attribute های این کلاس از کلاس دیگری به اسم Field ایجاد می شوند که در ورودی خود مقداری نمی گیرد. زمانی که یک Element را از سایت هدف استخراج کرده ایم برای اینکه مقدار آن را به صورت موقتی در زمان اجرای اسپایدر در دسترس داشته باشیم میتوانیم یک کلاس Item به شکلی که دیدید با فیلد های مد نظر ایجاد کنیم.

در بدنه کلاس از تابعی به نام `set_all_default_value` استفاده کردیم زمانی که این تابع صدا زده شود به ازای تمام فیلد هایی که در آیتم شما موجود می باشد مقدار پیش فرض None را قرار می دهد که البته مشخص است می توان مقدار پیش فرض None را توسط پارامتر ورودی value می توان به صورت دلخواه تغییر داد.

ماژول pipelines.py :

• کلاس TicketArrivalTmePipeline

```
class TicketArrivalTimePipeline:
    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        arrival_time = adapter.get("arrival_time")

        if arrival_time:
            splitted_arrival_time = arrival_time.split(":")

            if splitted_arrival_time[0] == "ورود":
                item["arrival_time"] = None

        return item
```

وظیفه این کلاس ایجاد تغییر بر روی فیلد arrival_time آن آیتم مربوطه می باشد. اسکریپی به صورت خودکار از این کلاس استفاده می کند و تابع process_item را صدا می زند تا تغییر لازمه روی فیلد انجام شود. اتفاقی در این تابع می افتد این می باشد که در زمان کراول کردن سایت و استخراج فیلد arrival_time زمانی که مقدار این فیلد وجود داشت و مقدار این فیلد برابر با "ورود" بود مقدار آن را به None تغییر می دهیم به عبارتی ساده تر اگر مقدار فیلد arrival_time برابر با فرمت «hour : minute» نبود مقدار آنرا برابر با None قرار می دهیم و سپس آبجکت item را بر می گردانیم و ادامه پروسه را به فریمورک اسکریپی می سپاریم.

- کلاس DuplicatesItemsPipeline

```
class DuplicatesItemsPipeline:
    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        if adapter["ticket_id"] in self.ids_seen:
            raise DropItem(f"Duplicate item found:\n")
        else:
            self.ids_seen.add(adapter["ticket_id"])
            return item
```

همانطور که از اسم کلاس پیداست این کلاس توسط تابع `__init__` یک Instance Variable به نام `ids_seen` ایجاد می‌کند در این متغیر مجموعه `id` های هر بلیت قرار خواهد گرفت تا از ذخیره کردن یک آیتم تکراری جلوگیری شود. در ادامه نیز در تابع `process_item` مقدار `id` هر بلیت را بررسی می‌کنیم اگر مقدار آن در مجموعه `ids_seen` وجود نداشت یعنی این آیتم از قبل وجود ندارد و می‌تواند با `return` به ادامه روند اسکرپینگ بازگشت داده شود در مقابل اگر مقدار `id` در مجموعه وجود داشت با `raise` کردن یک ارور به کاربر اطلاع می‌دهیم که یک آیتم تکراری پیدا شده است.

ماژول `requests_method_enum.py`:

- کلاس RequestMethods

```
class RequestMethods(Enum):
    POST = "POST"
    GET = "GET"
    PUT = "PUT"
    PATCH = "PATCH"
```

این کلاس از کلاس دیگر به `Enum` ارث بری کرده است که موجب می‌شود قابلیت یک مقدار شمارشی به این کلاس اضافه شود مقدار های شمارشی ای که ما از آن‌ها استفاده کردیم انواع اکشن رکیوئست HTTP مربوطه می‌باشد مثل `POST` ، `GET` ، `PUT` ، `PATCH`

استفاده از این ساختار کد به جای استفاده مستقیم مقادیر String موجب می‌شود کد ما خواناتر و تمیز تر باشد و علاوه بر آن مشخص است که این مقادیر رشته ایی برای چه چیزی می‌باشند که در مورد ما برای نمایش انواع روش‌های ارسال درخواست HTTP می‌باشند.

ماژول `utils.py` :

• تابع `two_permutation_airports_codes`

```
def two_permutation_airports_codes():
    settings = get_project_settings()
    json_file_path = settings.get("SOURCES")["airport_city_codes_path"]

    jsn_obj = get_json_obj(json_file_path)
    airports_codes_lst = extract_values_from_json_obj(jsn_obj, "id")
    two_permutation_codes = permutations(airports_codes_lst, 2)

    return two_permutation_codes
```

با خواندن `settings.py` توسط تابع `get_project_settings` و دسترسی به مسیر فایل `airport_city_codes.json` توسط کلید `airport_city_codes_path` در متغیر دیکشنری `SOURCE` میتوانیم این فایل `json` را توسط تابعی که در پکیج `common_utils` به نام `get_json_obj` قرار دارد بخوانیم و در متغیر `jsn_obj` قرار دهیم سپس توسط تابع `extract_values_from_json_obj` می‌توانیم به دنبال تمام کلیدهای `id` درون `jsn_obj` بگردیم و آن‌ها را درون متغیر لیست `airports_codes_lst` نگه داریم و در نهایت توسط تابع `permutations` که در پکیج `itertools` قرار دارد می‌توانیم عمل جایگشت دوتایی مجموعه (نمونه برداری ترتیبی بدون جایگزینی دوباره) را انجام دهیم و خروجی این جایگشت را توسط کلمه رزرو شده `return` برگردانیم.

به طور مثال داریم:

`['THR', 'MHD', 'ISF'] → Permutations 2 → [('THR', 'MHD'), ('THR', 'ISF'), ('MHD', 'THR'), ('MHD', 'ISF'), ('ISF', 'THR'), ('ISF', 'MHD')]`

بدین ترتیب می‌توانیم در سایت هدف بررسی کنیم که بین چه مبدأ و مقصد هایی هواپیما وجود دارد.

• تابع `filter_selectors(selectors, css_selector)` :

```
def filter_selectors(selectors, css_selector):
    for selector in selectors:
        if selector.css(css_selector):
            yield selector
```

در ابتدا باید توضیح دهیم زمانی که درخواستی را به سایت هدف ارسال می‌کنیم سایت هدف برای ما یک response حاوی کدهای HTML بر می‌گرداند برای جستجو در این کدها و پیدا کردن Element مربوطه فریمورک اسکریپی دو روش به نام های Select via css و Select via xpath را اختیار ما می‌گذرد و اما کاربردی که تابع filter_selectors برای ما دارد همانطور که از اسمش پیداست وظیفه فیلتر کردن تمامی selector ها را به ازای css_selector که به آن می‌دهیم بر عهده دارد و در نهایت آن selector پیدا شده را با کلمه رزرو شده yield نگه می‌داریم تا بعداً در صورت نیاز از آن استفاده کنیم (کاربرد return با yield متفاوت می‌باشد وقتی از return استفاده می‌کنیم کار تابع در همان لحظه تمام می‌شود اما وقتی از yield استفاده می‌کنیم وضعیت فعلی تابع در حین اجرا نگه داشته خواهد شد تا زمانی که تابع دوباره فراخوانی شود و ما به وضعیت بعدی تابع برویم در واقع ما یک lazy function توسط کلمه کلیدی yield ایجاد کرده‌ایم که هر موقع به آن نیاز داشتیم صدایش می‌زدیم تا از خروجی این generator استفاده کنیم).

نکته : توابع دیگری نیز در ماژول utils.py وجود دارند ولی چون از آن‌ها استفاده ایی نشده از توضیح آن‌ها امتناع خواهیم کرد.

ماژول middlewares.py :

```
> class ScrapeOpsFakeUserAgentMiddleware: ...  
  
> class TorMiddleware: ...
```

در این ماژول دو کلاس ScrapeOpsFakeUserAgentMiddleware و TorMiddleware را نوشته‌ایم این دو کلاس درخواست های HTTP را قبل از اینکه به درون اینترنت بروند به دلخواه ما دستکاری می‌کنند. وظیفه کلاس ScrapeOpsFakeUserAgentMiddleware این می‌باشد که به API سایت ScrapeOps یک درخواست جهت دریافت FakeUserAgent تصادفی دورانی ارسال می‌کند (منظور از تصادفی دورانی این می‌باشد که تعداد این user-agent های فیک از طرف API محدود بوده و این سایت حول یک صف به صورت دورانی این user-agent ها را در اختیار ما قرار می‌دهد) تا در هدر user-agent خود به جای user-agent اسکریپی جایگزین کنیم. همانطور که مشخص است یکی از راه‌های جلوگیری از تشخیص ربات بودن کراولر توسط سایت هدف استفاده از این user-agent ها می‌باشد. از راه‌های دیگری که ما در settings.py به آن‌ها اشاره کردیم استفاده از Download_Delay و Randomize_Download_Delay می‌باشند. در ادامه راجه مابقی راه‌های موجود توضیحاتی را ارائه خواهیم داد. البته ناگفته نماند که این روش‌های گفته شده با استفاده از متغیر ROBOTTXT_OBEY متفاوت هستند چرا که با استفاده از این متغیر همچنان سایت متوجه می‌شود که ما ربات هستیم ولی استفاده از این متغیر موجب می‌شود

که ربات ما از محدودیت موجود که توسط سایت هدف برای کراولر شدن توسط اسپایدرها تعیین شده فراتر نرود تا IP ربات توسط سایت بلاک نشود.

در ادامه می‌توان بیان کرد که وظیفه کلاس TorMiddleware تغییر IP کراولر به صورت تصادفی دروانی می‌باشد شبکه تور لیستی از گره‌های موجود با وضعیت آماده باش را که هرروز آپدیت می‌شوند در اختیار کاربران خودش قرار می‌دهد با وجود این لیست می‌توانیم درخواست هایی را به شبکه تور ارسال کنیم تا IP در مدت زمان های متفاوتی تغییر دهد. این عمل ما را از کمین سایت هدف برای بلاک کردن اسپایدرها نجات خواهد داد هرچند که باید در نظر داشت برای انجام این عمل سایت هدف باید و حتماً از exit node های شبکه تور پشتیبانی لازمه را داشته باشد.

• کلاس ScrapeOpsFakeUserAgentMiddleware

○ تابع from_crawler

```
@classmethod
def from_crawler(cls, crawler):
    return cls(crawler.settings)
```

زمانی که وارد کلاس ذکر شده فوق شویم اولین تابعی که با آن برخورد خواهیم کرد تابع from_crawler می‌باشد که با دکوریتور classmethod ترکیب شده است این دکوریتور به ما این اجازه را خواهد داد که بدون نیاز به ساختن یک obj و فقط با صدا زدن نام کلاس به این تابع دسترسی داشته باشیم خروجی تابع فوق کانفیگ هایی می‌باشد که در settings.py قرار دارند و قرار است در کلاس ذکر شده استفاده شوند.

○ تابع __init__

```
def __init__(self, settings):
    self.scrapeops_api_key = settings.get('SCRAPEOPS_API_KEY')

    self.scrapeops_endpoint = settings.get(
        'SCRAPEOPS_FAKE_USER_AGENT_ENDPOINT',
        'http://headers.scrapeops.io/v1/user-agents?')

    self.scrapeops_fake_user_agents_active = settings.get(
        'SCRAPEOPS_FAKE_USER_AGENT_ENABLED',
        False)

    self.scrapeops_num_results = settings.get('SCRAPEOPS_NUM_RESULTS')

    self.headers_list = []

    self._get_user_agents_list()
    self._scrapeops_fake_user_agents_enabled()
```

با استفاده از تابع `__init__` در زمان ساخته شدن یک Instance از یک کلاس می‌توانیم پارامترهای اولیه لازمه را در آن مقدار دهی کنیم اگر دقت کنید می‌بیند که در ورودی تابع از آرگومان `settings` استفاده شده است که این آرگومان توسط تابع `from_crawler` و `crawler.settings` مقدار دهی می‌شود تا `setting` های ما در دسترس `middleware` ها قرار گیرند. همانطور که تا الان متوجه شده‌اید کانفیگ‌هایی مثل `SCRAPEOPS_API_KEY` و `SCRAPEOPS_FAKE_USER_AGENT_ENABLED` را قبلاً دیده ایم و راجه آن‌ها صحبت کردیم فقط اتفاقی که در تابع `__init__` می‌افتد این است که با استفاده از تابع `get` (دیگه حتماً متوجه شدین که `settings` یه دیکشنریه چون تابع `get` برای دسترسی به مقدار یه کلید استفاده میشه) به مقدار تنظیم شده دسترسی خواهیم داشت حال اگر این کانفیگ در `settings.py` وجود داشت که مقدارش در `Instance Variable` مد نظر قرار خواهد گرفت اگر وجود نداشت می‌توانی توسط تابع `get` مشخص کنیم چه مقدار پیش‌فرض در صورت عدم وجود کلید در آن قرار بگیرد (این کار موجب می‌شود تا در حین اجرا برنامه به ارور `KeyError` مواجه نشود) هر چند که اگر این مقدار پیش‌فرض را مشخص نکنیم تابع `get` مقدار `None` را در صورت عدم وجود برای عمل انتساب مد نظر قرار خواهد داد. با این توضیحات می‌توان به راحتی حدس زد که در هنگام خواندن مقدار کلیدهای `SCRAPEOPS_NUM_RESULTS` و `SCRAPEOPS_FAKE_USER_AGENT_ENDPOINT` چه اتفاقی رخ خواهد داد.

در `self.headers_list` مقادیر `header` ها قرار خواهند گرفت (البته ما از این متغیر استفاده نمی‌کنیم ولی به پیشنهاد داکيومنت `ScrapeOps` تو کد قرارش دادیم)

در رابطه‌های تابع‌های صدا زده شده در تابع `__init__` مثل `get_user_agents_list` و `scrapeops_fake_user_agents_enabled` در ادامه توضیح خواهیم داد هرچند که از روی اسم آن‌ها واضح است چه عملی را برای ما انجام می‌دهند.

○ تابع `get_user_agents_list`

```
def _get_user_agents_list(self):
    payload = {'api_key': self.scrapeops_api_key}
    if self.scrapeops_num_results is not None:
        payload['num_results'] = self.scrapeops_num_results
    response = requests.get(self.scrapeops_endpoint,
                           params=urlencode(payload))
    json_response = response.json()
    self.user_agents_list = json_response.get('result', [])
```

طبق داکيومنت سایت `ScrapeOps` در هنگام استفاده از API باید یک `form` ، `payload` یا همان `body` را به سمت API ارسال کرد که این `payload` می‌تواند شامل `api_key` و `num_results` باشد. همانطور که از کد مشخص

است استفاده از `api_key` اجباری ولی استفاده از `num_results` اختیاری است اگر آنرا تنظیم کنید می‌توانید مشخص کنید که چه تعداد `user-agent` فیک را به شما برگرداند. باید توجه کرد که فرمت ارسال `payload` توسط کتابخانه `requests` حتماً باید به صورت دیکشنری و همچنین به گفته داکيومنت `ScrapeOps` حتماً باید به صورت `Encode` شده درون `URL` یا همان `Endpoint` مد نظر ارسال شوند. تابع `get` نیز در کتابخانه `requests` وظیفه ارسال درخواست `HTTP` با نوع `GET` را برعهده دارد در نهایت نیز بعد از آماده سازی `payload` و ارسال آن توسط پارامتر `params` به تابع `get` نوبت خواندن `response` دریافتی توسط تابع `json` و ذخیره `json` در متغیر `json_response` می‌باشد در مرحله آخر با کمک تابع `get` دیکشنری (دیگه حتماً می‌دونید که فرمت دیکشنری پایتون مثل `json` می‌ماند و درواقع پایتون با فرمت `json` مثل یک دیکشنری رفتار می‌کند) می‌توانیم مقدار کلید `result` موجود در `json` را بخوانیم و خروجی آنرا در متغیر `Instance` به نام `user_agents_list` ذخیره کنیم.

○ تابع `scrape_fake_user_agents_enabled`

```
def _scrapeops_fake_user_agents_enabled(self):
    if (self.scrapeops_api_key is None or
        self.scrapeops_api_key == '' or
        self.scrapeops_fake_user_agents_active == False):

        self.scrapeops_fake_user_agents_active = False
        self.scrapeops_fake_user_agents_active = True
```

راستش خود بنده هم نفهمیدم هدف از ایجاد این تابع توسط داکيومنت `ScrapeOps` چی می‌باشد زیرا که در نهایت خروجی `self.scrapeops_fake_user_agents_active` چه شرط اجرا بشود و چه نشود برابر با مقدار `True` خواهد بود اما در هر حال به پیشنهاد داکيومنت ما از آن در کد خود استفاده کرده ایم. شرط فوق به ما می‌گوید اگر `self.scrapeops_api_key` هویتی برابر با `None` داشته باید یا مقدارش برابر با `""` باشد یا مقدار `self.scrapeops_fake_user_agents_active` برابر با مقدار لیترال `False` باشد. در نهایت مقدار `self.scrapeops_fake_user_agents_active` برابر با `False` شود.

○ تابع `get_random_user_agent`

```
def _get_random_user_agent(self):
    random_index = randint(0, len(self.user_agents_list) - 1)
    return self.user_agents_list[random_index]
```

اگر یادتان باشد خروجی تابع `get_user_agents_list` مقدار `self.user_agents_list` می‌باشد که همان `response` سایت `ScrapeOps` به ما بوده است. فقط ما در تابع فوق صرفاً عمل انتخاب تصادفی بین از اعضای لیست موجود را با تابع `randint` کتابخانه `random` در بازه ۰ تا تعداد اعضای لیست مربوطه انجام داده ایم.

○ تابع `process_request`

```
def process_request(self, request, spider):
    random_user_agent = self._get_random_user_agent()
    request.headers['User-Agent'] = random_user_agent
```

همانطور که اشاره شد خروجی `ScrapeOpsFakeUserAgentMiddleware` باید یک `user-agent` فیک باشد که توسط سایت `ScrapeOps` به ما ارائه می‌شوند. در نهایت خروجی مد نظر را توسط تابع `process_request` در بخش `headers` درخواست خود قرار خواهیم داد تا تغییر مربوطه بر روی درخواست `HTTP` اعمال شود.

• کلاس `TorMiddleware`

○ تابع `from_crawler`

```
@classmethod
def from_crawler(cls, crawler):
    settings = crawler.settings

    intermediate_proxy_url = settings.get('INTERMEDIATE_PROXY')
    ip_checker_site = settings.get("IP_CHECKER_SITE")
    tor_changing_ip_delay_sec = settings.get('TOR_CHANGING_IP_DELAY_SEC')
    tor_control_port = settings.get('TOR_CONTROL_PORT')
    tor_randomize_changing_ip_delay = settings.get('TOR_RANDOMIZE_CHANGING_IP_DELAY')
    tor_password = settings.get('TOR_PASSWORD')

    return cls(
        intermediate_proxy_url,
        ip_checker_site,
        tor_changing_ip_delay_sec,
        tor_control_port,
        tor_randomize_changing_ip_delay,
        tor_password)
```

با تابع `from_crawler` در `ScrapeOpsFakeUserAgentMiddleware` آشنا شده‌ایم همان عمل کردی که در کلاس گفته شده داشتیم همان عمل کرد را در کلاس `TorMiddleware` خواهیم داشت با این تفاوت که فقط در اینجا کانفیگ های بیشتری را از `settings.py` می‌خوانیم و به تابع `__init__` می‌فرستیم باید توجه شود که حتماً ترتیب برگرداندن متغیرها در تابع `from_crawler` و دریافت آن‌ها در تابع `__init__` باید یکسان باشد وگرنه مقادیر متغیرها جا به جا خواهند شد و این اتفاق مساعد نیست و کد شما را با مشکلاتی همراه خواهد کرد.

```
def __init__(self,
              intermediate_proxy_url: str,
              ip_checker_site: str,
              tor_changing_ip_delay_sec: int = 10,
              tor_control_port: int = 9051,
              tor_randomize_changing_ip_delay: bool = False,
              tor_password: str = None) -> None:

    self.intermediate_proxy_url = intermediate_proxy_url
    self.ip_checker_site = ip_checker_site
    self.tor_changing_ip_delay_sec = tor_changing_ip_delay_sec
    self.tor_control_port = tor_control_port
    self.tor_randomize_changing_ip_delay = tor_randomize_changing_ip_delay
    self.tor_password = tor_password

    self.last_time_ip_changed = 0
    self.tmp_tor_changing_ip_delay_sec = self.tor_changing_ip_delay_sec
```

کلاس TorMiddleware برای اینکه بتواند درست کار کند به مقادیر دهی اولیه ایی که توسط تابع __init__ انجام می‌شود نیاز دارد همانطور که مشاهده می‌کنید یکسری از آرگومان های تابع فوق دارای مقدار پیش فرض هستند این یعنی اینکه اگر شما این متغیرها را از طریق settings.py مقدار دهی نکرده باشید تا به واسطه تابع from_crawler در اختیار تابع __init__ قرار گیرند در نتیجه این آرگومان توسط مقدارهای پیش فرض مشخص شده درون عکس مقدار دهی خواهند شد (به عبارتی این متغیرها اختیاری هستند و میتوانید در صورت لزوم settings.py آنها مقدار دهی نکنید.) اما حتماً باید مقدار آرگومان های intermediate_proxy_url و ip_checker_size از قبل اجرا در settings.py مشخص کنید(اگر نمیدانید این متغیرها چیکاری انجام می‌دهند به بخش مازول settings.py در زیر پروژه flight_tickets_scraper مراجعه کنید).

علاوه بر موارد گفته شده متغیرهای Instance به نام self.last_time_ip_changed و self.tmp_tor_changing_ip_delay_sec وجود دارند به ترتیب این متغیرها برای آخرین زمانی که Tor IP تغییر کرده و مقدار تأخیر موقتی مورد نیاز (اگر مقدار TOR_RANDOMIZE_CHANGING_IP_DELAY برابر با True باشد آنگاه مقدار self.tmp_tor_changing_ip_delay_sec تغییر خواهد کرد) برای تغییر IP فعلی می باشد.

○ تابع connect_to_tor

```
def _connect_to_tor(self, spider):
    controller = Controller.from_port(port=self.tor_control_port)

    if self.tor_password:
        controller.authenticate(password=self.tor_password)
        spider.logger.debug('Authentication has done correctly.')

    spider.logger.debug('Connection to tor through control port is established.')

    return controller
```

از این تابع برای وصل شدن به سرویس تور استفاده کرده‌ایم این تابع به این طریق کار می‌کند که در ابتدا با استفاده از کلاس Controller و classmethod به نام from_port به TorController که پورت آن را از قبل روی سیستم باز کرده‌ایم متصل می‌شویم و مطابق تصویر یک آبجکت controller اینجا خواهیم کرد حال اگر پسوردی بر روی سرویس تور تنظیم کرده باشیم عمل احراز هویت را توسط متد authenticate انجام می‌دهیم و در یک لاگ به گزارش می‌کنیم که عمل احراز هویت با موفقیت انجام شد و موفقیت آمیز بود احراز هویت به معنی ایجاد اتصال با سرویس تور می‌باشد که آنرا هم در لاگ سیستم گزارش می‌کنیم. در نهایت آبجکت controller را بر می‌گردانیم.

○ تابع change_ip_address

```
def _change_ip_address(self, spider):
    with self._connect_to_tor(spider) as controller:
        controller.signal(Signal.NEWNYM)

        spider.logger.debug('Your tor ip changed.')

    spider.logger.debug('Tor connection is closed now.')
```

این تابع توسط تابع connect_to_tor و کلمه رزرو شده with یک اتصال باز با سرویس تور روی سیستم شما ایجاد می‌کند و با ارسال سیگنال Signal.NEWNYM توسط متد signal به سرویس تور بازگو می‌کند که IP را برای کراولر ما تغییر دهد. در نهایت بعد از انجام این مراحل لاگ تغییر IP را در سیستم ثبت می‌کنیم و همچنین گزارش می‌کنیم که اتصال با سرویس تور توسط مدیریت کننده ریسورس with بسته شد.

○ تابع `exit_tor_node_ip_address`

```
def _exit_tor_node_ip_address(self):
    proxy = dict()
    protocol, domain_port = self.intermediate_proxy_url.split("://")
    proxy[protocol] = domain_port

    response = requests.get(
        url=self.ip_checker_site,
        proxies=proxy)

    return response.text.strip()
```

شبکه تور به طور پیش فرض از سه لایه relay (همان گره) استفاده می کند تا ترافیک مد نظر شما را منتقل کند در لایه آخر گره ایی وجود دارد به نام exit node یا گره خروجی که وظیفه ارسال درخواست شما به سایت هدف را برعهده دارد (در واقع این گره با سایت هدف ارتباط می گیرد و نتیجه را به شما می دهد بدون اینکه سایت هدف بداند آدرس IP واقعی شما چیست این همان چیزی است که ما به آن معجزه Tor Proxy می گوییم) اما کتابخانه stem این قابلیت را ندارد که آدرس IP گره خروجی را برای ما گزارش کند (در اصل این کار خیلی سخت و دشواره) و تنها راه فهمیدن آدرس گره خروجی استفاده از سایت ها و یا API هایی هست که به ما آدرس IP فعلی ما را گزارش می کنند به همین منظور است که ما متغیر IP_CHECKET_SITE را در setting.py مقدار دهی می کنیم چرا که می خواهیم با استفاده از سایت مد نظر به IP گره خروجی خود پی ببریم و این دقیقاً کاری است که تابع `exit_tor_node_ip_address` برای ما انجام خواهد داد از تابع `split` استفاده می کنیم تا مقدار `protocol` و `domain` `port` موجود در متغیر مقدار دهی شده `INTERMEDIATE_PROXY` را از هم جدا کنیم از این امر استفاده می کنیم تا به `requests` بگوییم برای ارسال درخواست به سایت مد نظر از چه پروکسی باید گذر کنی (کتابخانه `requests` به صورت پیش فرض نمیدونه که سرویس تور روی سیستم ما فعاله و باید از اون استفاده کنه تا IP گره خروجی رو بتوانیم مشاهده کنیم به همین دلیل است که از `Privoxy` کمک گرفته ایم اگر از `Privoxy` استفاده نکنیم اونوقت سایت مد نظر آدرس IP واقعی ما را بر می گرداند نه آدرس IP گره خروجی شبکه تور)

○ تابع `choose_random_number_between`

```
def _choose_random_number_between(self, min_number, max_number):
    return round(random.uniform(min_number, max_number), 1)
```

وظیفه تولید یک عدد تصادفی اعشاری با تخمین یک رقم اعشار در بازه عددی مشخص را بر عهده دارد. به دلیل اینکه از تابع `uniform` در کتابخانه `random` استفاده کرده‌ایم این قابلیت را نیز به ما می‌دهد تا انتخاب تصادفی هر کدام از اعداد بین بازه مشخص شده باهم برابر باشد.

○ تابع `set_tor_randomize_changing_ip_delay_sec_or_fixed`

```
def _set_tor_randomize_changing_ip_delay_sec_or_fixed(self):
    if self.tor_randomize_changing_ip_delay:
        self.tmp_tor_changing_ip_delay_sec = self._choose_random_number_between(
            0.5 * self.tor_changing_ip_delay_sec,
            1.5 * self.tor_changing_ip_delay_sec)
```

در صورتی که مقدار `self.tor_randomize_changing_ip_delay` برابر با `True` باشد، مقدار متغیر `Instance` به `self.tmp_tor_changing_ip_delay_sec` در بازه عددی مشخص تغییر خواهد کرد. این امر موجب خواهد شد تا IP ما در بازه های زمانی تصادفی تغییر کند.
بازه ها به صورت زیر می باشد:

[MinDelay, MaxDelay]

$\text{MinDelay} = 0.5 * \text{self.tor_changing_ip_delay_sec}$

$\text{MaxDelay} = 1.5 * \text{self.tor_changing_ip_delay_sec}$

○ تابع `set_new_ip`

```
def _set_new_ip(self, spider) -> None:
    self._change_ip_address(spider)

    current_tor_ip = self._exit_tor_node_ip_address()

    spider.logger.info(f'Tor exit node ip: {current_tor_ip}\n')

    self._set_tor_randomize_changing_ip_delay_sec_or_fixed()

    self.last_time_ip_changed = time.time()
```

حال که توابع بالا را توضیح دادیم کاملاً مشخص است که تابع `set_new_ip` به چه بخش‌هایی تقسیم شده و با صدا زدن تابع `set_new_ip` هر بخش چه کاری را برعهده دارد فقط لازم به ذکر است که هر بار یک IP جدید تنظیم کردیم باید مقدار متغیر `self.last_time_ip_changed` را با تابع `time` کتابخانه `time` مقدار دهی کنیم تا بعداً بتوانیم بررسی کنیم که در چه زمانی باید دوباره تابع `set_new_ip` را صدا بزنیم تا IP جدیدی را به کراولر ما انتساب دهد.

○ تابع process_request

```
def process_request(self, request, spider) -> None:
    now = time.time()

    if now - self.last_time_ip_changed > self.tmp_tor_changing_ip_delay_sec:
        self._set_new_ip(spider)

    request.meta['proxy'] = self.intermediate_proxy_url
```

هر بار که اسکریپ درخواستی ارسال می‌کند و درخواست به TorMiddleware می‌رسد این تابع به صورت خودکار صدا زده خواهد شد زمانی که این تابع صدا زده شد مقدار متغیر now برابر زمان فعلی می‌باشد که این تابع صدا زده شده از این متغیر استفاده می‌کنیم تا اختلافش را با آخرین زمانی که IP تغییر کرده بسنجیم و اگر این اختلاف بیشتر از مقدار تأخیر مورد نیاز برای تغییر IP بوده است باید تابع set_new_ip صدا زده شود تا IP جدیدی به کراولر انتساب داده شود. در نهایت هم برای اتصال اسکریپ با سرویس تور از آدرس Privoxy استفاده می‌کنیم و آن را در کلید proxy دیکشنری meta آبجکت request قرار می‌دهیم.

نکته : کلاس‌های دیگری نیز در ماژول middlewares.py وجود دارند ولی این کلاس‌ها توسط خود فریم‌ورک اسکریپ ایجاد شده‌اند و ما از آن‌ها در طول پروژه استفاده نکرده‌ایم به همین دلیل از توضیح‌ها امتناع خواهیم کرد.

ماژول tcharter_airlines_tickets_spider.py :

در این ماژول اسپایدرهای خود را کد نویسی می‌کنیم ما در این ماژول صرفاً یک اسپایدر به نام AirlinesTickets نوشته‌ایم و این اسپایدر در **چهار سطح** عمل Parse کردن سایت را برای ما انجام خواهد داد. به طور کلی parse کردن به معنی تجزیه و تحلیل کردن یک ساختار می‌باشد و به طور دقیق‌تر در مبحث Web Scraping عمل parse به معنی استخراج کردن اطلاعات در جهت یافتن Element های مد نظر از سورس های HTML و همچنین فرمت کردن آن‌ها برای استفاده کردن از آن‌ها در آینده می‌باشد.

همانطور که در متن اشاره کردیم parse کردن ما در چهار سطح انجام خواهد شد این چهار سطح به این معنی می‌باشد که همه اطلاعات مد نظر ما فقط با فرستادن درخواست به یک URL بدست نمی‌آیند و ما باید مرحله به مرحله به اندازه چهار سطح جلو برویم تا بتوانیم تمام فیلدهای مد نظر را برای آیتم های خود جمع‌آوری کنیم. قبل از اینکه به سراغ توضیح کدهای اسپایدر برویم این چهار سطح را به صورت گرافیکی به شما نشان می‌دهیم تا متوجه شوید که یک کاربر واقعی چگونه با یک مرورگر به این اطلاعات دسترسی خواهد داشت و بعد به همان شکل باید کدهای خود را بنویسیم.

سطح اول:

در سطح اول با ارسال درخواست به آدرس زیر:

www.tcharter.ir/tickets/search/0/{source}-{destination}

تصویر زیر را در نهایت مشاهده خواهیم کرد:

تهران به				
THR				
مشهد				
۱,۳۱۴,۵۰۰ تومان				
یکشنبه ۵ شهریور ۱,۳۱۴,۸۰۰ تومان	دوشنبه ۶ شهریور ۱,۶۹۱,۶۰۰ تومان	سه شنبه ۷ شهریور ۱,۶۹۱,۶۰۰ تومان	چهارشنبه ۸ شهریور ۱,۶۹۱,۶۰۰ تومان	پنج شنبه ۹ شهریور ۱,۶۹۱,۶۰۰ تومان
جمعه ۱۰ شهریور ۱,۶۲۰,۰۰۰ تومان	شنبه ۱۱ شهریور ۱,۶۹۱,۶۰۰ تومان	یکشنبه ۱۲ شهریور ۱,۶۹۱,۶۰۰ تومان	دوشنبه ۱۳ شهریور ۱,۶۹۱,۶۰۰ تومان	سه شنبه ۱۴ شهریور کلیک کنید
چهارشنبه ۱۵ شهریور ۱,۶۲۰,۰۰۰ تومان	پنج شنبه ۱۶ شهریور ۱,۶۲۰,۰۰۰ تومان	جمعه ۱۷ شهریور ۱,۶۲۰,۰۰۰ تومان	شنبه ۱۸ شهریور کلیک کنید	یکشنبه ۱۹ شهریور کلیک کنید
۱۵ روز قبل		گوش به زنگ برای استفاده از این سرویس باید ابتدا عضو...		۱۵ روز بعد

مقدار source و destination درواقع شناسه سه حرفی بین‌المللی فرودگاه مبدأ و مقصد می باشند. هر بار که با آدرس ذکر شده درخواستی را ارسال می‌کنیم نتیجه ای را مشابه تصویر بالا مشاهده خواهیم کرد نتیجه ایی که سایت هدف به ما نمایش می‌دهد از چهار section تشکیل شده که در هر section تعداد روزهایی به اندازه عدد ۱۵ وجود خواهد داشت. درواقع کلیک برروی دکمه "۱۵ روز بعد" یا "۱۵ روز قبل" section بعدی و قبلی را به شما نشان می دهد.

سطح دوم:

در سطح دوم درخواستی به آدرس زیر ارسال می کنیم:

www.tcharter.ir/tickets/dates/{source}-{destination}-airplane?section={section_counter}

مقدار source و destination همان مقدار هایی هستند که در سطح قبلی توضیح دادیم و اما مقدار section_counter این مقدار به هر section که در سایت مشاهده می‌کنید انتساب داده می‌شود که مقدار آن بین بازه [۴, ۱] می باشد.

در واقع با آدرسی که در بالا دارید می‌توانید بین هر section جا به جا شوید دقیقاً همان عملی که با کلیک بر روی دکمه های "۱۵ روز بعد" و "۱۵ روز قبل" اتفاق می افتد.

سطح سوم:


در این سطح با ارسال درخواست به آدرس زیر:

www.tcharter.ir/tickets/tickets/{city_airline_date_code}/

تصویری مشابه تصویر زیر را مشاهده خواهیم کرد:

From THR To MHD
Date ۲۸ August

از تهران به مشهد
دوشنبه ۰۶ شهریور ۱۴۰۲

شرکت	ساعت	ظرفیت	پرواز	کلاس	بهای نهایی	نوع/توضیح	فروشنده/رزرو
	۰۶:۲۵	۱	۰۲۴	DEFTOR	۱,۶۲۰,۰۰۰ تومان	با ضمانت-خدمات گردشگری	رزرو
	۰۵:۵۰	۴	۵۹۰۲	MA۰TOR	۱,۶۹۱,۶۰۰ تومان	با ضمانت-خدمات گردشگری	رزرو
	۲۱:۰۰	۷	۶۷۰۴	MA۰TOR	۱,۶۹۱,۶۰۰ تومان	با ضمانت-خدمات گردشگری	رزرو

مقدار city_airline_data_code متفاوت با هر پرواز بین مبدأ و مقصد و همچنین متناسب با تاریخ و روز حرکت متغیر است. درواقع مقدار این پارامتر ترکیبی به طول ۲۴ از کاراکترها و اعداد است که توسط خود سایت با توجه به مواردی که به آن‌ها اشاره کردیم تولید می‌شود. فرض می‌شود که سایت از این کد تولید شده برای بازیابی پروازهای مد نظر خویشتن در تاریخ انتخابی استفاده می‌کند.

توجه شود ممکن است در این تصویر که مشاهده می‌کنید یک جدول در چند صفحه با ایجاد حالت Pagination داشته باشیم. برای اینکه بتوانیم به هر صفحه این جدول دسترسی داشته باشیم از آدرس زیر استفاده می‌کنیم:

www.tcharter.ir/tickets/tickets/{city_airline_date_code}?page={next_page_number}


آدرس بالا مشابه آدرس سطح سومی است که مشاهده کردید منتهی یک پارامتر page به آن اضافه شده که با next_page_number مقدار دهی خواهد شد. با استفاده از آدرس بالا می‌توانید بین هر صفحه جدول منتقل شوید.

سطح چهارم:

در این سطح به ازای هر سطر جدول بلیت‌ها با ارسال درخواست به آدرس زیر:

www.tcharter.ir/reservations/step1/airplane/{reservation_number}

تصویر مشابه زیر را مشاهده خواهید کرد:

شرکت	ساعت	ظرفیت	پرواز	کلاس	بهای نهایی	نوع/توضیح	فروشنده/رزرو
	۱۱:۴۰	۱	۶۰۲	Y۳YD	۳,۱۳۵,۳۷۰ تومان	با ضمانت-بیزنس	رزرو

اطلاعات پرواز

از تهران به مشهد در تاریخ ۱۴۰۲/۰۶/۲۷ با هواپیمایی آسمان به شماره پرواز ۶۰۲ ساعت خروج ۱۱:۴۰ و ساعت ورود ۱۳:۱۰ می‌باشد.

۰۶ : ۵۶
ثانیه : دقیقه

اطلاعات مسافر

کد ملی: Last Name First Name بزرگسال آقا Iranian

+ افزودن

مشاهده جرایم کنسلی بلیت

مبلغ قابل پرداخت شما ۳,۱۳۵,۳۷۰ تومان می‌باشد.

ایمیل: ۰۹۱۲ ۳۴۵ ۶۷۸۹

کد تصویر: 1 9 5 6 2

ثبت اطلاعات

درواقع وقتی که تا سطح سوم پیش برویم اطلاعات لازمه را از هر سطر جدول می‌توانیم استخراج کنیم اما به نظر بنده در مرحله اسکرپینگ هر آنچه که سایت ارائه می‌دهد را استخراج کنیم بهتر می‌باشد (بعدا اگر نیازشان نداشتیم خیلی راحت می‌توانیم آن‌ها حذف کنیم ولی به فرض اگر نیاز داشتیم باید برگردیم و با بررسی لاجیک کار خود دوباره کراولر خود را کامل‌تر کنیم حال فرض کنید که مجبور باشید داده‌های زیادی را از سایت استخراج کنید آن وقت باید مدت زمان بیشتری را منتظر باشید تا کراول شما تمام شود). پس ترجیه دادم یک سطح دیگر هم به کراولر خود اضافه کنم تا اطلاعات اضافی مثل ساعت ورود، ساعت خروج و تاریخ حرکت را نیز استخراج کرده باشم.

• کلاس AirlinesTickets

خب بعد از تمامی توضیحاتی که تا به الان راجه ساختار فریمورک اسکرپی و ماژول‌هایی که در زیر پروژه flight_tickets_scraper وجود دارند ارائه دادیم حال نوبت به بخش اصلی پروژه یعنی استفاده از تمام قابلیت‌هایی کد نویسی کردیم و یا در فایل settings.py قرار دادیم می‌باشد. قبل از ورود به متد‌های موجود در کلاس AirlinesTickets ابتدا باید نگاهی به Class Variable این کلاس بی‌اندازیم:

```
class AirlinesTickets(scrapy.Spider):
    name = "airlines_tickets"
    allowed_domains = ["www.tcharter.ir"]

    base_url = "https://www.tcharter.ir"
    curl_request_raw_payload = r'types=%5B%22all%22%2C%22system%22%2C%22provider%22%2C%22bclass%22%2C%22economy%22%5D&tab=airplane'
    item_id = 1
```

متغیرهای کلاس را در عکس مشاهده می‌کنیم حال به ترتیب راجه هرکدام توضیحاتی می‌دهم:

۱. **name**: این متغیر اسم اسپایدر ما می‌باشد و اسکرپی اسپایدر ما را به این نامی که مشخص کرده‌ایم خواهد شناخت و هر موقع نیاز بود که اسپایدر ما اجرا شود و عملیات خود را آغاز کند از این نام استفاده خواهیم کرد.

۲. **allowed_domain**: این متغیر مشخص می‌کند به چه دامنه‌هایی مجاز به دسترسی هستیم.

۳. **base_url**: این متغیر URL اصلی سایت هدف می‌باشد از طریق این URL به Endpoint های سایت هدف می‌توانیم دسترسی داشته باشیم.

۴. **curl_request_raw_payload**: مقدار خام payload را مشخص می‌کند که در درخواست های cURL استفاده می‌شوند. این مقدار با آزمون و خطاهای متعدد بدست آمده درواقع سایت هدف با فرستادن payload به روش عادی به ما خروجی مناسب را نمی‌دهد در نتیجه از این روش برای فرستادن payload مورد نیاز استفاده کرده ایم.

۵. **item_id**: از این متغیر برای انتساب id به هر بلیت استفاده خواهیم کرد.

```
def start_requests(self):
    """send request for every possible permutation of two city airports."""

    two_permutation_result = two_permutation_airports_codes()

    for source, destination in two_permutation_result:
        cb_kwargs = {
            "source_city": source,
            "destination_city": destination
        }

        url = f"{self.base_url}//tickets/search/0/{source}-{destination}"

        yield scrapy.Request(
            url=url,
            callback=self.parse,
            method=RequestMethods.POST.value,
            body=self.curl_request_raw_payload,
            cb_kwargs=cb_kwargs)
```

تابع start_request همانطور که از اسمش پیداست اولین تابعی است که شما از آن برای شروع عملیات درخواست HTTP و کراول استفاده خواهید کرد و همچنین گفتنی است که این تابع متعلق به والدی است که ما در کلاس AirlinesTickets از آن ارث بری کرده ایم (به عبارتی از Polymorphism در اینجا بهره برده ایم). خوب با توضیحاتی که از ابتدای داکيومنت تا به الان ارائه کردیم توضیح دادن این قطعه کد و فهمیدنش نباید کار سختی باشد در خط اول تابع فوق شاهد استفاده از تابع two_permutation_airports_codes هستید که از ماژول utils می باشد و همانطور که قبلاً گفتیم این تابع جایگشت های دوتایی مجموعه ایی از شناسه های فرودگاه ها را به ما می دهد و در ادامه در حلقه ایی که نوشتیم به ازای جایگشت های دوتایی بدست آمده ما یک شناسه برای فرودگاه مبدأ و یک شناسه برای فرودگاه مقصد خواهیم داشت این مقادیر را با کلید هایی درون دیکشنری cd_kwargs قرار خواهیم داد و همچنین از شناسه ها در url استفاده می کنیم و در نهایت یک درخواست HTTP با کلاس Request ایجاد می کنیم. طبق عکس فوق کلاس Request برای ورودی های خود به مقادیر زیر احتیاج دارد:

۱. url : آدرسی که است که می خواهیم آنرا کراول کنیم.

۲. callback : ورودی این پارامتر یک تابع parser می باشد به عبارت دیگر با این پارامتر مشخص می کنیم بعد اینکه response از سایت هدف بازگشت برای parse کردن آن به کدام تابع آنرا ارجاع دهیم.

۳. method : روشی که با آن درخواست HTTP باید ارسال شود. در اینجا ما از روش POST استفاده کردیم که به صورت Enum می باشد.

۴. **body** : بدنه یا payload که نیاز آدرس URL مد نظر به آن نیاز دارد تا خروجی مطلوب را به تحویل دهد.

۵. **cb_kwargs** : از این پارامتر استفاده کرده‌ایم تا مقداری که در parser های بعدی به آن‌ها نیاز داریم را منتقل کنیم

تا الان باید متوجه شده باشید که چگونه ما یک درخواست HTTP که روزانه در مرورگر شما و میلیون‌ها کاربر عادی اتفاق می‌افتد را در کدنویسی با فریمورک اسکریپی شبیه سازی کرده ایم. این نوع شبیه سازی به کرات در دنیای برنامه استفاده می‌شود و در ادامه کدهای همین پروژه نیز به شکل مشابه چندین بار استفاده شده است.

• تابع parse

```
def parse(self, response, **kwargs):
    """first level of parsing for gathering airplanes dates sections urls."""

    airplane_dates_table = response.css(".ftmini")

    if airplane_dates_table:
        request_payload = {
            "types": ["all", "system", "provider", "bclass", "economy"],
            "tab": "airplane",
            "selected_date": "",
        }

        source = response.cb_kwargs["source_city"]
        destination = response.cb_kwargs["destination_city"]

        # this section counter goes to number 4 because in tcharter js goes to number 4 for ticket calendar table
        # (show_calendar_page js function) in otherwise It will be out next page in calendar page.
        for section_counter in range(1, 5):
            url = f"{self.base_url}/tickets/dates/{source}-{destination}-airplane?section={section_counter}"

            yield scrapy.Request(
                url=url,
                callback=self.parse_airplane_dates_table,
                method=RequestMethods.POST.value,
                body=json.dumps(request_payload),
                cb_kwargs=response.cb_kwargs
            )
```

تابع parse نیز تابعی می‌باشد که توسط والد کلاس اسپایدر AirlinesTickets پیاده‌سازی شده است و ما آن را در اینجا Override کرده‌ایم. این تابع وظیفه parse کردن در اولین سطح را کراول را برعهده دارد.

در اولین خط تابع توسط css_selector در response دریافتی به دنبال جدول تاریخی که قبلاً آن را به شما نشان دادیم می‌گردیم و خروجی را درون متغیر airplane_dates_table قرار می‌دهیم و با یک شرط وجود یا عدم وجود جدول را بررسی می‌کنیم زیرا که منطقی است که ما از همه فرودگاه‌ها به همه فرودگاه بلیت برای فروش نخواهیم داشت این شرط موجب خواهد شد که در ادامه رکویست‌های اضافی را به سمت سایت هدف نفرستیم. در ادامه مقادیر source و destination را از طریق متغیر cb_kwargs که در مرحله قبل به تابع parse انتقال دادیم مقدار دهی کردیم و همچنین payload مورد نیاز را برای رکویست مد نظر را فراهم کردیم.

یک حلقه ایجاد کردیم که به ازای محدوده ۱ تا ۵ آدرس URL هر section را برای میسازد و در نهایت با کلاس Request باز هم درخواست HTTP را برای مرحله بعدی parse شبیه سازی می‌کنیم. اگر به پارامتر body کلاس Request دقت کنید متوجه خواهید شد که مقداری متفاوت از body است که در مرحله قبل استفاده کردیم در

اینجا از تابع dumps کتابخانه json بهره برده ایم که تابع به ما کمک می‌کند تا آبجکت دیکشنری ساخته شده را به رشته تبدیل کنیم (ساختار دیکشنری دست نخواهد خورد فقط کل دیکشنری ما با رشته تبدیل خواهد شد).

• تابع parse_airplane_dates_table

```
def parse_airplane_dates_table(self, response, **kwargs):
    """second level of parsing for gathering tickets selling dates."""

    city_airline_date_rows = response.css(".daterow")
    filtered_city_airline_rows = filter_selectors(city_airline_date_rows, ".currency_symbol")

    for filtered_city_airline_row in filtered_city_airline_rows:
        city_airline_date_code = filtered_city_airline_row.css(".daterow::attr(data)").get()
        date_values = filtered_city_airline_row.css("#dateItem > span:nth-child(1)::text").getall()

        url = f"{self.base_url}/tickets/tickets/{city_airline_date_code}/"

        response.cb_kwargs["city_airline_date_code"] = city_airline_date_code
        response.cb_kwargs["date"] = f"{date_values[0].strip()} {date_values[1]}"
        response.cb_kwargs["page_number"] = 1

    yield scrapy.Request(
        url=url,
        callback=self.parse_tickets_table,
        method=RequestMethods.POST.value,
        body=self.curl_request_raw_payload,
        cb_kwargs=response.cb_kwargs
    )
```

این تابع وظیفه استخراج داده در سطح دوم عملیات کراول را بر عهده دارد. در خط اول این تابع می‌بینید که از response دریافت شده توسط css_selector که استفاده کردیم (.daterow) تمام grid ها را از جدول تاریخ ها در اولین section بیرون کشیدیم و در متغیر city_airline_date_rows قرار دادیم و سپس توسط تابع filter_selector آن grid هایی که دارای سی اس اس سلکتور currency_symbol هستند را استخراج کردیم. به تصویر زیر قبلاً به شما نشان دادیم دوباره دقت کنید:

تهران به				
مشهد				
یکشنبه ۰۵ شهریور ۱,۳۱۴,۸۰۰ تومان	دوشنبه ۰۶ شهریور ۱,۶۹۱,۶۰۰ تومان	سه شنبه ۰۷ شهریور ۱,۶۹۱,۶۰۰ تومان	چهارشنبه ۰۸ شهریور ۱,۶۹۱,۶۰۰ تومان	پنج شنبه ۰۹ شهریور ۱,۶۹۱,۶۰۰ تومان
جمعه ۱۰ شهریور ۱,۶۲۰,۰۰۰ تومان	شنبه ۱۱ شهریور ۱,۶۹۱,۶۰۰ تومان	یکشنبه ۱۲ شهریور ۱,۶۹۱,۶۰۰ تومان	دوشنبه ۱۳ شهریور ۱,۶۹۱,۶۰۰ تومان	سه شنبه ۱۴ شهریور کلیک کنید
چهارشنبه ۱۵ شهریور ۱,۶۲۰,۰۰۰ تومان	پنج شنبه ۱۶ شهریور ۱,۶۲۰,۰۰۰ تومان	جمعه ۱۷ شهریور ۱,۶۲۰,۰۰۰ تومان	شنبه ۱۸ شهریور کلیک کنید	یکشنبه ۱۹ شهریور کلیک کنید
<div> <div>۱۵ روز قبل</div> <div>گوش به زنگ برای استفاده از این سرویس باید ابتدا عضو شو...</div> <div>۱۵ روز بعد</div> </div>				

این تصویری بود که در سطح اول به شما نشان دادیم در سطح دوم می‌خواهیم آن grid هایی که دارای قیمت بلیت هواپیما هستند را استخراج کنیم از دلایلی که چرا همه grid ها را در هر section با هم استخراج نمی‌کنیم در زیر آمده است:

۱. کاهش فزاینده تعداد رکویست هایی که به سایت هدف ارسال می‌کنیم چرا که نیاز نیست وقتی در آن grid سایت هدف جدولی به ما ارائه نمی‌دهد ما یک رکویست اضافی به سایت ارسال و برای سرور هایش سربار ایجاد کنیم. (چرا؟ چون فقط می‌خواهیم مجانی و با زور از سایتش داده جمع آوری کنیم؟)
 ۲. سایت هدف در grid هایی که قیمت در آن‌ها وجود ندارد جدول بلیت ها را ارائه نمی‌دهد.
- در ادامه کدها می‌بینید که به ازای grid های فیلتر شده متغیرهای city_airline_date_code و date_values را از طریق css_selector هایی که از سایت هدف پیدا کرده‌ایم مقدار دهی می‌کنیم. دیکشنری cb_kwargs را طبق عکس با کلیدهایی مقدار دهی کرده تا در تابع parse بعدی از آن‌ها استفاده کنیم و در نهایت درخواست HTTP را شبیه سازی و ارسال می‌کنیم.

• تابع parse_tickets_table

کدهای مربوط به این تابع را در چند بخش توضیح خواهیم داد:
بخش اول:

```
def parse_tickets_table(self, response, **kwargs):
    """third level of parsing for gathering tickets information."""

    # if there is no next pages then response is "error", or
    # if status code response of server is 500, or
    # table existence then there is no tickets for the day we want.
    if (response.body == b"error" or
        response.status == 500 or
        not response.css("table").get()):
        return None

    tickets_table = response.css('.table.main-ticket-list tbody')
    ticket_detail_trs = tickets_table.css(".airplane-row")
    ticket_detail_blue_tr = tickets_table.css(".blue-light")

    if ticket_detail_blue_tr:
        ticket_detail_trs.extend(ticket_detail_blue_tr)
```

این تابع وظیفه کراول در سطح سوم را برعهده دارد درواقع درخواست از سطح دوم ارسال می‌شود و نتیجه آن در تابع فوق کراول خواهد شد. در بدو شروع بدنه تابع مشاهده می‌کنید که ما از یک شرط استفاده کرده‌ایم این شرط

درواقع شرط تابع بازگشتی ما می‌باشد دروابع تابع فوق یک Parser بازگشتی است به این دلیل که وقتی شما به سطح سوم کراول برسید نیاز دارید همه اطلاعات جدول بلیت ها را که در قالب pagination به صورت صفحه به صفحه ارائه شده‌اند را به شکلی **یکنواخت** parse کنید. خیلی واضح این شرط به ما می‌گوید اگر body ریسپانس برابر با رشته "error" بود یا status برابر با مقدار ۵۰۰ بود و یا جدول بلیت ها پیدا نشد مقدار None را برگردان و دیگر به کراول کردن ادامه نده این بدان معنی است که یا صفحات pagination تمام شده و صفحه بعدی وجود ندارد یا سایت هدف در آن لحظه جدولی نمی‌تواند ارائه دهد با اینکه ما از grid هایی استفاده کردیم که هزینه بلیت در آن‌ها نوشته شده بود و یا اینکه به دلایلی سایت هدف به اررور ۵۰۰ خورده است. توسط css_selector هایی که در تصویر می‌بینید response را parse کرده و متغیر های tickets_table و ticket_detail_trs و ticket_detail_blue_tr را مقدار دهی می‌کنیم. گفتنی است که ticket_table دروابع کدهای HTML جدول بلیت ها، ticket_detail_trs تمام سطرهایی با رنگ سبز و سفید و ticket_detail_blue_tr تمام سطرهایی با رنگ آبی می‌باشند. شرط بعدی هم مشخص می‌کند که اگر مقدار ticket_detail_blue_tr پر بود اعضای مجموعه این متغیر به متغیر ticket_detail_trs اضافه شوند.

بخش دوم:

```
for ticket_detail_tr in ticket_detail_trs:
    flight_ticket_item = FlightTicketsScraperItem()
    flight_ticket_item.set_all_default_value()

    ticket_detail_tds = ticket_detail_tr.css("td")

    flight_ticket_item["company_name"] = ticket_detail_tds[0].css("::attr(data-hint)").get()
    flight_ticket_item["departure_time"] = ticket_detail_tds[1].css("::text").get()
    flight_ticket_item["capacity"] = ticket_detail_tds[2].css("::text").get()
    flight_ticket_item["flying_number"] = ticket_detail_tds[3].css("::text").get()
    flight_ticket_item["flying_class"] = ticket_detail_tds[4].css("::text").get()
    flight_ticket_item["ticket_price_T"] = ticket_detail_tds[6].css("::text").getall()[1].strip()
    flight_ticket_item["flying_type"] = ticket_detail_tds[7].css("::text").getall()[1].strip()

    flight_ticket_item["national_departure_code"] = response.cb_kwargs["source_city"]
    flight_ticket_item["national_arrival_code"] = response.cb_kwargs["destination_city"]
    flight_ticket_item["departure_date"] = response.cb_kwargs["date"]

    flight_ticket_item["ticket_id"] = self.item_id
    self.item_id += 1

    ticket_extra_detail_url = ticket_detail_tds[7].css(".finishDescription a::attr(href)").get()

    if (flight_ticket_item["flying_type"] and
        'go' not in ticket_extra_detail_url.split("/")):

        flight_ticket_item_cb_kwargs = {
            "flight_ticket_item": flight_ticket_item
        }

        yield scrapy.Request(
            url=ticket_extra_detail_url,
            callback=self.parse_extra_detail,
            method=RequestMethods.GET.value,
            cb_kwargs=flight_ticket_item_cb_kwargs)
    else:
        yield flight_ticket_item
```


حال به ازای تمامی سطرهایی که از جدول بلیت ها در آن تاریخ مد نظر پیدا کرده ایم وقت این است که یک آبجکت از کلاس FlightTicketsScraperItem که نوشته ایم ایجاد کنیم و در ابتدا توسط متد `set_all_default_value` آیتم خود را مقدار دهی اولیه کرده و سپس بعد از استخراج داده ها توسط `css_selector` هایی که در تصویر می بینید مقدار کراول شده را به آیتم انتساب می دهیم. همانطور که مشاهده می کنید مقداردهی به آبجکت آیتم دقیقاً به مانند مقدار دهی به صورت کلید-مقدار در دیکشنری می باشد. در رابطه با متدهای دیگر که در تابع فوق استفاده شده می توان به متد `getall` ، `get` و `strip` اشاره کرد به ترتیب متد `getall` تمام المنت هایی که `selector` پیدا کرده را برمی گرداند، متد `get` صرفاً اولین المنت پیدا شده توسط `selector` را برمی گرداند و متد `strip` نیز متد آبجکت `String` می باشد که وظیفه آن پاک کردن تمامی `white space` های درون رشته مد نظر می باشد. حال بعد از مقدار دهی به آیتم خود باید آدرس `URL` سطح بعدی یعنی سطح چهارم را توسط `attribute` با نام `href` از طریق سلکتور `"finishDescription a"` استخراج کنیم ولی باید توجه کرد که طبق سایت هدف این آدرس همیشه آن آدرسی که ما برای سطح چهارم می خواهیم نیست از شرایطی که آدرس استخراج شده مناسب نمی باشد در زیر بیان شده اند:

۱. کلید `flying_number` در آیتم مربوطه باید دارای مقداری غیر از `None` باشد.

۲. آدرس `URL` استخراج شده در متغیر `ticket_extra_detail_url` نباید حاوی کلمه `"go"` باشد.

اگر شرایط فوق درست بود در نتیجه می توانیم درخواست `HTTP` را توسط کلاس `Request` ایجاد کرده و توسط پارامتر `cb_kwargs` آیتم فعلی را برای کامل تر شدن آن به پارسر سطح چهارم انتقال دهیم. در غیر این صورت اگر شرایط فوق معتبر نبود آیتم فعلی را فقط توسط کلمه رزرو شده `yield` برای اعمالی مثل `Saving to` ، `Validation` ، `Manipulation` ، `database` و غیره به `pipeline` ها انتقال می دهیم یا اینکه خیلی ساده آن را درون فایل `csv` ذخیره می کنیم.

بخش سوم:

```
#When parsed the all items in each page's tables comes here.
response.cb_kwargs["page_number"] += 1
next_page_number = response.cb_kwargs["page_number"]

city_airline_date_code = response.cb_kwargs["city_airline_date_code"]

next_page_url = f"{self.base_url}/tickets/tickets/{city_airline_date_code}?page={next_page_number}"

yield scrapy.Request(
    url=next_page_url,
    callback=self.parse_tickets_table,
    method=RequestMethods.POST.value,
    body=self.curl_request_raw_payload,
    cb_kwargs=response.cb_kwargs)
```

نکته ایی که باید آن اشاره کنم این می باشد که فریمورک اسکریپی به صورت `Asynchronous` کار می کند این بدان معنی است که وقتی اسکریپی یک درخواست `HTTP` را ارسال می کند منتظر نمی ماند که `Response` آن به سیستم

شما برگردد تا رکویوست بعدی را ارسال کند، بلکه تمام رکویوست های شما را پشت سر هم ارسال، و با وجود یک Scheduler تمام Response ها را بعداً درون callback function که شما موقع ارسال رکویوست مشخص کردید بررسی می کند. طبق کامنتی که درون عکس نوشتیم حتماً متوجه شدید بعد از اینکه **هر بار** بخش دوم تابع parse_tickets_table تمام شد (یعنی زمانی که تمام سطر های درون جدول **فعلی** استخراج شد) ما به بخش سوم تابع مورد بحث خود می رسیم باید توجه کنید که این بدان معنا نیست که کار بخش دوم کاملاً تمام شده بلکه رکویوست های باقی مانده باز هم پشت سر هم ارسال خواهند شد. (اینجاست که مفهوم Asynchronous نمود پیدا خواهد کرد). به طور ساده تر شما در نظر بگیرید که در **سطح سوم کراول** خود هستید و یک جدول چند صفحه ایی دارید زمانی که تمامی داده های سطر های جدول صفحه اول را استخراج کردید تازه به **بخش سوم تابع مورد بحث** خواهید رسید. از آنجایی که با توجه به ساختار سایت هدف به راحتی نمی توان تشخیص داد که جدول ما چند صفحه ایی می باشد تا بدانیم که تا کی باید parse کردن جدول را ادامه دهیم باز هم به ابزار Network مرورگر خود در بخش Developer مراجعه کرده و درخواستی که برای هر صفحه ارسال می شود را پیدا می کنیم، آن آدرس دقیقاً مشابه آدرسی است که در **سطح سوم** به شما توضیح دادیم و در **بخش سوم تابع** نیز آن را مشاهده می کنید. از آنجایی که هر درخواست اسکریپت از هم مستقل بوده و جدا از هم parse می شوند ما یک کلید به نام page_number در دیکشنری cb_kwargs دقیقاً بعد از parse شدن **تمامی سطر های هر جدول** (یعنی هر بار تمام شدن بخش دوم تابع مورد بحث) قرار می دهیم تا بتوانیم با آن آدرس URL **صفحه بعدی هر جدول** را مشخص کند. دلیل اینکه از cb_kwargs چرا استفاده کردیم هم باید برایتان واضح باشد برای اینکه بتوانیم هر بار به صفحه بعدی هر جدول برویم باید مقدار صفحه فعلی هر جدول را داشته باشیم اینگونه مطمئن خواهیم شد که تمامی سطر های تمام صفحات هر جدول را parse خواهیم کرد. دقت شود برای اینکه آدرس URL صفحه بعدی ما کامل شود باید از مقدار city_airline_date_code که از سطح دوم با دیکشنری cb_kwargs به سطح سوم انتقال داده ایم هم استفاده کنیم و آن را در URL خود قرار دهیم بعد از طی کردن این مراحل حال نوبت به ایجاد درخواست با کلاس Request می باشد فقط حتماً توجه کنید که مقدار callback این کلاس باید با نام تابع سطح سوم یعنی parse_tickets_table دوباره پر شود چرا که در متن هم به آن اشاره کردیم که پارسر سطح سوم یک پارسر بازگشتی می باشد و در رابطه با شروط پایان این پارسر بازگشتی هم توضیحات کاملی را ارائه کردیم.

• تابع parse_extra_detail

```
def parse_extra_detail(self, response, **kwargs):
    """fourth level of parsing for gathering tickets detail."""

    flight_ticket_item = response.cb_kwargs["flight_ticket_item"]
    ticket_extra_detail = response.css(".ps-2 div::text").get()

    if ticket_extra_detail:
        splitted_ticket_extra_detail = ticket_extra_detail.strip().split()

        flight_ticket_item["arrival_time"] = splitted_ticket_extra_detail[-3]
        flight_ticket_item["arrival_city_name_persian"] = splitted_ticket_extra_detail[3]
        flight_ticket_item["departure_city_name_persian"] = splitted_ticket_extra_detail[1]
        flight_ticket_item["departure_date_YMD_format"] = splitted_ticket_extra_detail[6]

    yield flight_ticket_item
```

این تابع به عنوان یک پارسر سطح چهارم اطلاعات اضافی مثل ساعت ورود هواپیما به شهر مقصد، ساعت خروج هواپیما از شهر مبدأ، تاریخ حرکت هواپیما به فرمت YMD و اسم شهر مبدأ و مقصد به فارسی را در صورت وجود المنت ticket_extra_detail در آیتم مربوطه مقدار دهی کرده و در نهایت آن را yield می کند.

ماژول runner.py :

```
5 os.chdir(os.path.dirname(os.path.realpath(__file__)))
6
7 scrapy_file_name = sys.argv[1]
8
9 try:
10     execute(
11         [
12             'scrapy',
13             'runspider',
14             scrapy_file_name,
15         ]
16     )
17 except SystemExit:
18     pass
```

دیباگر Visual Studio Code به طور پیش فرض عمل دیباگ یک کراولر را پشتیبانی نمی کند و ما باید به طریقی بتوانیم عمل دیباگ را بر روی کراولر خود انجام دهیم این قابلیت توسط ماژول runner به کدهای ما اضافه شده و در صورت نیاز می توانیم به راحتی کد خود را دیباگ کنیم. زمانی که روی دکمه دیباگ vscode کلیک کنید، ماژول

runner از طریق فایل مخفی launch.json اجرا خواهد شد (بعداً در رابطه با چگونگی تنظیم ماژول runner در این فایل توضیحاتی خواهم داد) خط اول این ماژول از چند بخش تشکیل شده است:

۱. `os.path.realpath(__file__)` : آدرس مطلق ماژول runner را برمی گرداند.

۲. `os.path.dirname` : آدرس مطلق دایرکتوری runner را تشخیص خواهد کرد.

۳. `os.chdir` : مسیر فعلی را به مسیر دایرکتوری runner تغییر خواهد داد.

هدف از قطعه کد بالا این می باشد که بتوانیم از قطعه کد `"scrapy runspider {spider_file_name}"` در پروژه اسکریپی که ایجاد کردیم استفاده کنیم چرا که این قطعه کد در جایی اجرا خواهد شد که اولاً اسکریپی در آن نصب باشد دوماً پروژه مد نظر ایجاد شده باشد.

در خط بعدی کد بالا از `sys.argv[1]` استفاده شده زمانی که شما فایل runner را اجرا کنید به همراه آن باید یک آرگومان هم به این فایل ارسال کنید (دقیقاً مثل ارسال یک مقدار به آرگومان تابع فقط اینجا ما به ماژول داریم مقداری را ارسال می کنیم) که مقدار این آرگومان باید فایل اسپایدر مد نظر شما باشد.

در نهایت با قرار دادن یک Exception برای مواقعی که دیباگ ما تمام شد یا به هر دلیلی دیباگ ما متوقف شد و همچنین استفاده از تابع `execute` از فریمورک اسکریپی برای اجرای قطعه کد `"scrapy runspider {spider_file_name}"` کار ما در نوشتن و توضیح ماژول runner تمام می شود.

Vscode برای اینکه بتواند عمل دیباگ را انجام دهد به فایل کانفیگ launch.json نیاز خواهد داشت درواقع اگر می خواهید هر نوع دیباگی انجام دهید در ابتدا باید این فایل را کانفیگ کنید. اگر می خواهیم یک اسپایدر را دیباگ کنیم می توانیم از کانفیگ زیر استفاده کنیم:

```
{
  "name": "Python: Launch Scrapy Spider",
  "type": "python",
  "request": "launch",
  "program": "/home/magnus9102/Mostafa/Py/Github/data-science/mostafa_vahdani_bachelor_project/source_codes/data/flight_tickets_scraper/runner.py",
  "args": [
    "${file}"
  ],
  "console": "integratedTerminal"
}
```

توضیح هرکدام از کلید ها به شرح زیر خواهد بود:

۱. **name** : نرم افزار vscode کانفیگ شما را به اسمی که در این کلید مشخص خواهید کرد خواهد شناخت.

۲. **type** : در کلید باید مشخص کنید که کد شما بر اساس چه زبان برنامه نویسی نوشته شده است.

۳. **request** : خواسته شما از این کانفیگ چیست، در مورد دیباگ کردن ما می خواهیم که برنامه یا ماژول مربوطه لانچ یا همان اجرا شود. پس از مقدار launch استفاده خواهیم کرد.

۴. **program** : نام برنامه یا ماژولی که می خواهید برای انجام عمل دیباگ اجرا شود (بنده از آدرس دهی مطلق برای ماژول runner استفاده کردم اگر شما می توانید از آدرس دهی نسبی استفاده کنید بهتر است)

۵. **args** : آرگومان هایی که ماژول runner انتظار دارد به آن بدیم، این آرگومان باید در یک لیست داده شوند و ترتیب آن ها دارای اهمیت می باشد. `"${file}"` درواقع یک متغیر محیطی است که vscode از آن استفاده می کند تا آدرس فایل را که می خواهیم دیباگ شود را مشخص کند.

۶. **console** : محیطی که ما می خواهیم دیباگر ما در آن کد ها را اجرا کند و به آن مقدار **intergratedTerminal** را انتساب داده ایم. در صورت استفاده از این مقدار vscode از ترمینال خودش برای اجرای کد های استفاده می کند.

فاز دوم پروژه

در این فاز به پیش پردازش داده های خام و آماده سازی یک دیتاست تمیز شده خواهیم پرداخت. تا به اینجای کار در رابطه با تمام قسمت های زیر پروژه `flight_tickets_scraper` توضیحات کاملی را ارائه کردیم از اینجا به بعد در رابطه با زیر پروژه `flight_tickets_preporcessing` توضیح خواهیم داد.

ماژول `settings.py` :

```
load_dotenv()
|
DATASET_BASE_PATH = pathlib.Path(__file__).parent.parent.parent.parent
PROJECT_BASE_PATH = pathlib.Path(__file__).parent.parent

SOURCES = {
    # datasets
    "dataset_file_path_from": DATASET_BASE_PATH.joinpath("data", "raw", "flight_tickets_dataset.csv"),
    "dataset_file_path_to": DATASET_BASE_PATH.joinpath("data", "processed", "flight_tickets_dataset.csv"),
    # statics
    "static_dir_path": PROJECT_BASE_PATH.joinpath("static"),
    "airport_codes_json_file_path": PROJECT_BASE_PATH.joinpath("static", "airport_city_codes.json"),
    "airports_info_json_file_path": PROJECT_BASE_PATH.joinpath("static", "airports_info.json"),
    "airports_geometry_json_file_path": PROJECT_BASE_PATH.joinpath("static", "airports_geometry.json"),
    "months_json_file_path": PROJECT_BASE_PATH.joinpath("static", "months.json"),
    "airlines_code_json_path": PROJECT_BASE_PATH.joinpath("static", "airlines_code.json"),
}
```

```
COLUMNS_NEED_TO_MOVE = {
    "arrival_city": 4,
    "national_arrival_code": 3,
    "departure_city": 1,
    "national_departure_code": 0,
    "departure_airport": 2,
    "arrival_airport": 5,
    "departure_date_YMD_format": 8,
    "local_departure_time": 9,
    "local_arrival_time": 10,
    "flight_number": 12,
    "ticket_price_T": 15,
    "company_name": 11,
    "orthodromic_distance_KM": 6,
    "flight_length_min": 7,
    "flight_sale_type": 13,
    "fare_class_code": 14
}

OPENCAGEDATA_API_KEY = os.getenv("OPENCAGEDATA_API_KEY")
```

۱. تابع **load_dotenv** : تابعی برای لود کردن پارامترهای مشخص شده در فایل مخفی env درون متغیرهای محیطی.
۲. **DATASET_BASE_PATH** : آدرس مطلق دایرکتوری data که دیتاست های ما درون آن قرار دارند.
۳. **PROJECT_BASE_PATH** : آدرس مطلق دایرکتوری data که درون دایرکتوری source_codes وجود دارد
۴. **SOURCES** : دیکشنری ای می باشد که مقدار آن آدرس متصل شده DATASET_BASE_PATH و یا PROJECT_BASE_PATH به دایرکتوری و نام فایل های مد نظر برای ایجاد کردن مسیر مطلق آنها می باشد. و کلید هم رشته ایی برای دسترسی آسان و راحت به مسیر مطلق فایل ها می باشد. برای متصل کردن آدرس ها نیز می توانیم از تابع joinpath استفاده کنیم.
۵. **COLUMNS_NEED_TO_MOVE** : بعد از ایجاد تغییرات، ستون های بالا ها را خواهیم داشت و می توانیم ترتیب قرار گرفتن این ستون ها را در دیتاست خود با یک عدد مشخص کنیم.
۶. **OPENCAGEDATA_API_KEY** : مقدار کلید API که از متغیرهای محیطی توسط تابع getenv خوانده شده است.

ماژول **utils.py** :

• تابع **difference_drop**

```
def difference_drop(df, *args):
    """difference drop with column names that you will give on *args"""
    return df.drop(columns=df.columns.difference([*args]), axis=1)
```

این تابع به ما کمک می کند تا هر ستونی به غیر از ستون هایی که در ورودی این تابع به عنوان args می دهیم از دیتاست مربوطه پاک کنیم. تابع difference لیستی از نام ستون ها را از شما خواهد گرفت و هر ستونی غیر از آن ستون هایی بهش دادین را به شما برمیگرداند در نتیجه با برگرداندن خروجی تابع difference و دادن این خروجی به تابع drop هر ستونی به غیر ستون هایی که به تابع difference_drop داده ایم را می توانیم از دیتاست حذف کنیم.

• تابع **check_existence_of_null_values**

```
def check_existence_of_null_values(df, cols_name):
    for col_name in cols_name:
        # if printed true then there is null value in your column
        print(f"{col_name}: {df[col_name].isnull().any()}")
```

در این تابع به ازای لیستی از نام ستون‌ها می‌توانیم بررسی کنیم که آیا مقدار null در آن ستون وجود دارد یا خیر. تابع isnull به ازای هر سطر مقدار True برای اعلام وجود null و مقدار False را برای اعلام عدم وجود null بر می‌گرداند و تابع any در صورتی که حتی یک سطر را به عنوان True شناسایی کند مقدار True را برای ما چاپ خواهد کرد که در نتیجه این به معنی وجود مقدار null در ستون مد نظر می‌باشد.

- تابع values_count

```
def values_count(df, col_name):  
    return df[col_name].value_counts()
```

وظیفه شمارش تعداد هر کدام از مقادیر Categorical در ستون مد نظر را بر عهده دارد.

- تابع count_unique_values

```
def count_unique_values(df, col_name):  
    return f"Length: {df[col_name].nunique()}"
```

وظیفه شمارش تعداد سطرهای یکتا موجود در ستون مد نظر را بر عهده دارد.

- تابع count_null_values

```
def count_null_values(df, col_name):  
    return df[col_name].isnull().sum()
```

تعداد مقادیر null موجود در ستون مد نظر را شمارش می‌کند.

- تابع check_space_existence

```
def check_space_existence(df, cols_name):  
    for col_name in cols_name:  
        # if result is true then there is field that has space value.  
        result = any(list(map(lambda x: True if str(x).find(' ') != -1 else False, df[col_name])))  
        print(f"{col_name}: {result}")
```

در این تابع به ازای لیستی از نام ستون‌ها می‌توانیم بررسی کنیم که آیا space در آن ستون وجود دارد یا خیر. تابع find به ازای هر سطر مقدار True برای اعلام وجود space و مقدار False را برای اعلام عدم وجود space برمی‌گرداند و تابع any در صورتی که حتی یک سطر را به عنوان True شناسایی کند مقدار True را برای ما چاپ خواهد کرد که در نتیجه این به معنی وجود space در ستون مدنظر می‌باشد. تابع map درواقع عمل نگاشت تابع خطی lambda را بروی Iterable مدنظر (df[col_name]) انجام می‌دهد و به ما یک generator برمی‌گرداند که با تابع list خروجی map را به تابع any می‌دهیم تا بررسی حضور و عدم space را انجام دهد.

• تابع get_masked_df

```
def get_masked_df(df, col_name, specific_value):  
    mask = (df[col_name] == specific_value)  
    return mask, df[mask]
```

ماسک کردن، فرآیند پنهان کردن مقادیر یا ردیف‌های خاص از یک چارچوب داده بر اساس معیارهای خاص است. خروجی این تابع یک دیتافریم با شرایط ماسک کردن فوق (بررسی برابری با مقداری به خصوص) و خود ماسک مدنظر می‌باشد.

• تابع apply_to_df_by_mask

```
def apply_to_df_by_mask(main_df, mask, masked_df, col_name, your_func):  
    main_df.loc[mask, col_name] = masked_df[col_name].apply(func=your_func)
```

با این تابع می‌توانیم توسط دیتاست ماسک شده، دیتاست اصلی و با استفاده از یک تابع خارجی (your_func) روی ستون مدنظر تغییرات دلخواه‌مان را اعمال کنیم. به عبارت ساده‌تر اگر بخواهیم روی مقادیر سطرهای خاصی از یک ستون دیتافریم تغییری به وجود آوریم از این تابع استفاده می‌کنیم. تابع loc برای انتخاب داده‌ها جهت انتساب مقدار جدید از یک DataFrame بر اساس labelها یا شرایط Boolean استفاده می‌کند. تابع apply نیز برای اعمال کردن یک تابع خارجی روی محور ستونی یا سطری (axis) استفاده می‌شود. تا به الان باید متوجه شده باشید که از تابع فوق باید در کنار تابع get_masked_df استفاده کنیم.

• تابع group_by_count

```
def group_by_count(df, *args):  
    """last column name on the args is the column you want to apply count() on it"""  
    return df.groupby([*args])[args[-1]].count()
```

این تابع لیستی از نام ستون‌ها را از شما خواهد گرفت و در خروجی بر اساس ترتیبی که در لیست ستون‌ها مشخص کردید بعد از اعمال گروه بندی مد نظرتان، برای شما عمل شمارش را روی آخرین عضو لیست ستون‌ها انجام خواهد داد.

- تابع `count_duplicated`

```
def count_duplicated(df):  
    return df.duplicated().sum()
```

این تابع برای شما سطرهای تکراری را در دیتافریم شمارش خواهد کرد. درواقع تابع `duplicated` هر سطر که مقدار تکراری آن را پیدا کند یا برابر با `True` قرار خواهد داد و تابع `sum` تعداد `True` ها را برای شما شمارش خواهد کرد.

- تابع `check_col_distribution`

```
def check_col_distribution(col_df):  
    col_df.hist()  
    plt.show()
```

نوع توزیع داده‌های ستون مد نظر را با نمودار هیستوگرام به شما نمایش خواهد داد. توزیعی که در خروجی مشاهده می‌کنید می‌تواند `normal` ، `right skewed` ، `left skewed` ، `symmetric` و غیره باشد.

- تابع `filter_rows_by_values`

```
def filter_rows_by_values(df, col, values):  
    df.drop(df[df[col].isin(values)].index, inplace=True)
```

این تابع وظیفه فیلتر کردن سطرهایی با مقادیر خاص روی ستون مد نظر را برعهده دارد. اجزای این تابع در زیر به ترتیب شرح داده شده اند:

۱. `df[col].isin(values)` : توسط تابع `isin` بررسی می‌شود که به ازای سطرهایی که یکی از مقادیر خاص ما (`values`) در آن وجود داشته باشد مقدار `True` برای آن سطر در نظر گرفته می‌شود.
۲. `df[df[col].isin(values)].index` : بعد از پیدا کردن سطرهای مد نظر، اندیس آن‌ها توسط `index` برگشت داده خواهد شد.

۳. تابع **drop** : ایندکس های سطر های مد نظر بعد از قرار گرفتن در این تابع حذف خواهند شد.

۴. آرگومان **inplace** : این آرگومان اگر True باشد به ما این اجازه را خواهد داد تا بدون نیاز به عمل انتساب دیتافریم خود را آپدیت کنیم.

• تابع **count_specific_value_in_col**

```
def count_specific_value_in_col(df, col_name, sp_value):  
    return (df[col_name] == sp_value).sum()
```

با استفاده از این تابع می توانیم یک مقدار خاص را درون ستون مد نظر شمارش کنیم. قطعه کد زیر را در نظر بگیرید:

• **(df[col_name] == sp_value)**

با این قطعه کد هر جایی در ستون مد نظر مقداری برابر با مقدار مشخص شده (sp_value) بشود مقدار True برایش در نظر گرفته شده و سپس توسط تابع sum این مقادیر True شمارش خواهند شد.

• تابع **advance_mode**

```
def advance_mode(group):  
    mode = group.mode()  
    if not mode.empty:  
        return group.fillna(group.mode().iloc[0])  
    return group
```

این تابع برای پر کردن مقادیر گم شده (missing values) Categorical یک ستون گروه بندی شده توسط تابع mode استفاده می شود حال به چه علت از تابع mode به تنهایی استفاده نکردیم، همانطور که از تصویر فوق مشخص است تابع mode در جایی که تعداد فراوانی های یک مقدار Categorical برابر با None باشد مقدار None را به ما بر خواهد گرداند در نتیجه با ایجاد یک شرط None بودن مقدار mode را روی گروه مد نظر مشخص می کنیم اگر None بود که همان گروه را برمی گردانیم ولی اگر مقدار mode برایمان None نبود تمامی سطرهای None آن گروه را برابر با مقدار mode قرار می دهیم. گفتنی که خروجی تابع mode یک دیتافریم می باشد پس برای دسترسی به مقدار mode می توانیم از تابع iloc با ایندس صفر استفاده کنیم.

```
def fill_with_random(df, column):
    df2 = df.copy()
    df2[column] = df2[column].apply(lambda x: np.random.choice(
        df2[column].dropna().values) if pd.isnull(x) else x)
    return df2
```

این تابع برای پر کردن مقادیر گم شده Categorical استفاده می‌شود و از انتخاب تصادفی روی خود مقادیر موجود درون ستون مد نظر عمل می‌کند. کد بالا را به ترتیب زیر شرح می‌دهیم:

۱. **تابع copy** : یک کپی از دیتافریم به ما می‌دهد و آنرا درون متغیر df2 قرار می‌دهیم این کار انجام دادیم چون جلوتر نمی‌خواستیم تغییری روی دیتافریم اصلی ما اتفاق بی‌افتد.

۲. **تابع dropna** : این تابع هر جا که در ستون مد نظر مقدار null داشته باشد برای ما حذف می‌کند.

۳. **values** : این attribute تمام مقادیر ستون مد نظر را در قالب یک آرایه به بر خواهد گرداند. توجه شود که استفاده از این attribute بعد از حذف کردن مقادیر null درون ستون اتفاق افتاده.

۴. **تابع isnull** : بررسی می‌کند که آیا مقدار x یک مقدار null می‌باشد یا خیر.

۵. **تابع apply** : اعمال کردن یک تابع برای ایجاد تغییر بر روی مقادیر ستون مد نظر.

۶. **تابع choice** : وظیفه انتخاب تصادفی از اعضای یک آرایه را بر عهده دارد.

طبق توضیحات بالا باید متوجه شده باشید که روی ستون مدنظرتان تابع apply اعمال می‌شود و تابع apply نیز درون خودش دارای یک تابع خطی برای اعمال تغییرات می‌باشد. این تابع خطی هر بار مقدار هر فیلد درون ستون را برمی‌دارد و در متغیر x قرار می‌دهد حال توسط تابع isnull مقدار x بررسی می‌شود اگر این مقدار برابر با مقدار null نبود خود x برای ما برگردانده می‌شود چرا که نمی‌خواهیم جایی از ستون که مقدار دارد را تغییر دهیم ما صرفاً می‌خواهیم مقادیر گم شده را پر کنیم در ادامه اگر مقدار x برابر با null بود از دیتافریم df2 ستون مد نظر را انتخاب کرده با تابع dropna مقادیر null حذف می‌کنیم و با attribute به نام values یک آرایه از تمامی مقدار باقی‌مانده در ستون ایجاد می‌کنیم و سپس توسط تابع choice یک انتخاب تصادفی انجام می‌دهیم و مقدار بدست آمده را برمی‌گردانیم تا در مکان null مد نظر در ستون قرار گیرد.

ماژول : airports_geographic_coordinate.py

• تابع get_airport_geographic_coordinate

کدهای مربوط به این تابع را در چند بخش توضیح خواهیم داد:

بخش اول:

```
def get_airport_geographic_coordinate(geo_coder, **kwargs):
    """This function is just working on 'OpenCageGeocode web api' results for my specific task."""
    global max_recursive_depth

    if max_recursive_depth == 2:
        return None

    q = kwargs["query"]
    country_code = kwargs["country_code"]
    airport_name = kwargs["airport_name"]

    airport_code = kwargs.get("airport_code", q)
    no_dedupe = kwargs.get("no_dedupe", 1)
    no_annotations = kwargs.get("no_annotations", 1)
    pretty_output = kwargs.get("pretty", 1)
    language = "en"

    result_lst = geo_coder.geocode(query=q, countrycode=country_code, no_dedupe=no_dedupe,
                                   no_annotations=no_annotations, pretty=pretty_output,
                                   language=language)
    sleep(1)
```

این تابع مختص به API سایت OpenCageGeocode نوشته شده است و وظیفه آن بدست آوردن مختصات جغرافیایی فرودگاه ها می باشد. این تابع به صورت بازگشتی نوشته شده است و شرط پایان آن توسط یک متغیر سراسری به نام max_recursive_depth مشخص می شود، مقدار اولیه این متغیر صفر می باشد و زمانی که مقدار این متغیر برابر با مقدار لیترال ۲ بشود مقدار None برگشت داده شده و کار تابع بازگشتی پایان می یابد. دلیل اینکه خواستیم عمق بازگشتی تابع فوق برابر با ۲ باشد این می باشد که ما یکبار می خواهیم با کد فرودگاه به دنبال مختصات جغرافیایی آن بگردیم و اگر با کد فرودگاه نتوانستیم مختصات را پیدا کنیم می خواهیم که با اسم فرودگاه دوباره درخواست به API بدهیم تا مختصات جغرافیایی را به ما باز گرداند. از کد بالا مشخص است که متغیرهایی مثل country_code ، q و airport_name حتماً باید از طریق kwargs مقدار دهی شود اگر این کلید ها در دیکشنری kwargs وجود نداشته باشد آن وقت برنامه با ارور KeyError مواجه خواهد شد مابقی متغیرهای محلی همانطور که در کد مشاهده می کنید توسط تابع get مقدار دهی پیش فرض خواهند شد. در ادامه نیز از آبجکت geo_coder متد geocode که ایجاد و ارسال درخواست به سمت API فرا می خوانیم و پارامترهای لازمه را توسط متغیرهای محلی مقدار دهی می کنیم. البته طبق داکيومنت باید گفت که سایت OpenCage پارامترهای بیشتری را برای ارسال رکيوئست قبول می کند ولی ما برای دستیابی به مختصات جغرافیای فرودگاه مد نظر صرفاً به پارامترهای تهیه شده نیاز خواهیم داشت پارامترها را به ترتیب زیر شرح می دهیم:

۱. query : کوئری که نیاز داریم باید توسط این پارامتر ارسال کنیم در مورد مسأله ایی که ما باهاش درگیر بوده ایم کوئری ما می تواند کد فرودگاه یا اسم فرودگاه مد نظر باشد.

۲. countrycode : کدی ۲ حرفی IATA که یک کشور را با آن شناسایی خواهند کرد.

۳. language : با این پارامتر می توان مشخص کرد که response به چه زبانی به شما ارسال شود.

۴. pretty : اگر مقدار آن برابر با ۱ باشد خروجی را به شکلی نمایش خواهد داد که خواندن و دیباگ کردن راحت تر باشد.

۵. no_dedupe : اگر مقدار این پارامتر برابر با ۱ باشد از ارسال مقادیر تکراری در response جلوگیری خواهد شد.

۶. no_annotation : اگر مقدار این پارامتر برابر با ۱ باشد از ارسال اطلاعات اضافی جلوگیری خواهد شد.

شاید متوجه شده باشید ما نیازی به مورد ۵ و ۶ نخواهیم داشت دلیل استفاده کردن از این موارد صرفاً به خاطر کمتر از سر بار به روی سرور می باشد(هرچند که ما درخواست زیادی به سمت سرور مربوطه ارسال نخواهیم کرد ولی استفاده از این پارامتر ها خالی از لطف نیست)

در ادامه از تابع sleep در کتابخانه time استفاده کرده ایم و مقدار آن را برابر با ۱ ثانیه قرار داده ایم. دلیل اینکه از ۱ ثانیه تأخیر استفاده کرده ایم به گفته داکيومنت می باشد چرا که در داکيومنت بیان شده برای API هایی که از اکانت رایگان استفاده می کنند درخواست های خود را باید با تأخیر ۱ ثانیه ایی ارسال کنند. در نهایت نیز response خود را درم متغیر result_lst برای ادامه کار ذخیره می کنیم.

بخش دوم:

```
for result_dict in result_lst:
    components_dict = result_dict["components"]
    _category, _type = components_dict["_category"], components_dict["_type"]

    if (_category == "transportation" and
        _type == "aeroway"):

        aeroway_name = components_dict.get("aeroway", airport_name)
        geometry_tpl = (result_dict["geometry"]["lat"],
                        result_dict["geometry"]["lng"])

        result_tpl = (
            airport_code, {
                "aeroway_name": aeroway_name,
                "geometry": geometry_tpl
            }
        )

        return result_tpl

max_recursive_depth += 1

params = {"query": f"{airport_name} Airport",
          "country_code": country_code,
          "airport_code": airport_code,
          "airport_name": airport_name}

return get_airport_geographic_coordinate(geo_coder, **params)
```

گفتنی که result_lst همانطور که از اسمش مشخص است ساختمان داده ایی از نوع لیست پایتونی می باشد که اعضای آن دیکشنری می باشد.

حال به ازای دیکشنری هایی که درون این لیست قرار دارند که حلقه با for ایجاد می کنیم و متغیرهای _type ، component_dict و category_ را توسط مقادیر کلیدهایی که مشاهده می کنید مقدار دهی خواهیم کرد. طبق گفته داکيومنت مقدار category_ می تواند مقادیری مثل ، social ، commerce ، road ، transportation ، place و غیره باشد. در این API فرودگاه در دسته transportation قرار دارند پس منطقی است که در چندین مکانی که API بابت کوئری که زدیم به ما پیشنهاد داده به دنبال این مقدار در category_ باشیم علاوه بر اینکه مقدار _type دقیقاً مشخص می کند در دسته transportation ما به دنبال حمل و نقل هوایی هستیم که مقدار آن با aeroway مشخص می شود. در نتیجه فقط با این دو مقدار می توانیم محدوده بررسی را کوچک کرده و فرودگاه مد نظر را از بین چند مکانی که API پیشنهاد می دهد پیدا کنیم در نهایت هم با معتبر بودن شرط طبق مطالبی که توضیح دادیم از کلیدهای مرتبطة دیکشنری مقدار طول و عرض جغرافیایی را پیدا می کنیم و با توجه به ساختار فوق درون تاپل result_tpl قرار می دهیم و مقدار آنرا به خارج از تابع باز می گردانیم. اگر بعد از تمام شدن حلقه for همچنان مقداری که به دنبالشان بودیم را پیدا نکردیم به مقدار max_recursive_depth یکی اضافه کرده و تابع فوق را با مقادیری که مشاهده می کنید دوباره فرا می خوانیم و همین فرایند را دوباره از سر می گیریم فقط با این تفاوت که query ما این بار نام فرودگاه می باشد.

طریقه استفاده از تابع get_airport_geographic_coordinate :

```
if __name__ == "__main__":
    airports_geometry_dict = dict()
    airports_info_dict = get_json_obj(SOURCES["airports_info_json_file_path"])
    geo_coder = OpenCageGeocode(OPENCAGEDATA_API_KEY)

    for airport_val in airports_info_dict.values():

        params = {"query": airport_val["airport_code"],
                  "country_code": airport_val["country_code"],
                  "airport_name": airport_val["airport_names"][-1]}

        airport_geometry_tpl = get_airport_geographic_coordinate(geo_coder, **params)

        if airport_geometry_tpl:
            key, value = airport_geometry_tpl
            airports_geometry_dict[key] = value

    airports_geometry_dict["Total Length"] = len(airports_geometry_dict)

    with open(SOURCES["airports_geometry_json_file_path"], 'w') as fp:
        json.dump(airports_geometry_dict, fp, indent=4)
```

در اولین خط یک متغیر خاص پایتونی به نام `__name__` را مشاهده می‌کنید این متغیر زمانی که مستقیم ماژول پایتونی را اجرا کنید مقدار آن با `__main__` برابر خواهد بود ولی زمانی که ماژول پایتون را در ماژول دیگر import کنید مقدارش برابر نام آن ماژول خواهد بود. پس طبق توضیحات فقط در صورتی که ماژول `airports_geographic_coordinate.py` را مستقیماً اجرا کنیم شرطی که در خط اول عکس قرار دارد معتبر خواهد بود. یک متغیر به نام `airports_geometry_dict` ایجاد کردیم که این متغیر مقادیر مرتبط با مختصات جغرافیایی هر فرودگاه را درون خودش جای خواهد داد. مسیر فایل `airports_info.json` را از طریق کلید خودش در دیکشنری `SOURCES` بدست می‌آوریم و توسط تابع `get_json_obj` محتوای آنرا می‌خوانیم و درون متغیر `airports_info_dict` قرار می‌دهیم. در ادامه یک آبجکت با کلاس `OpenCageGeoCode` با کلید API ایی که در `setting.py` قرارداد ایجاد می‌کنیم تا بتوانیم با اکانت خود از API مربوطه استفاده کنیم. سپس توسط متد `values` به ازای تمام مقادیر درون کلیدهای دیکشنری `airports_info_dict` یک حلقه ایجاد می‌کنیم و تابع `get_airport_geographic_coordinate` را با مقداردهی به پارامترهای مورد نیازش صدا می‌زنیم و خروجی اش را در متغیر `airport_geometry_tpl` نگهداری می‌کنیم. اگر مقدار این متغیر برابر با `None` باشد یعنی ما نتوانسته‌ایم نه با کد فرودگاه و نه با اسم فرودگاه مختصات جغرافیایی آنرا پیدا کنیم پس شرط مربوطه معتبر نخواهد اما در غیر این صورت اگر مقدار این متغیر برابر `None` نبود مقدار `key` و `value` را جدا می‌کنیم و درون دیکشنری `airports_geometry_dict` قرار می‌دهیم باید دقت شود که مقدار `key` طبق متغیر `airport_geometry_tpl` برابر با کد فرودگاه و مقدار `value` برابر با یک دیکشنری می‌باشد که خود این دیکشنری شامل نام فرودگاه و مختصات جغرافیایی آن می‌باشد. بعد از اینکه کار حلقه ما تمام شد تعداد مقادیر پیدا شده درون متغیر `airports_geometry_dict` را با تابع `len` بدست می‌آوریم و با کلید `"Total Length"` دوباره درون دیکشنری `airports_geometry_dict` قرار می‌دهیم. در نهایت هم محتوای این متغیر را در فایل `json` مد نظر ذخیره سازی می‌کنیم تا بعد از آن استفاده لازمه را برای تمیز کردن دیتاست خام خود ببریم.

ماژول `tickets_preprocessing.py` :

• تابع `correct_field`

```
def correct_field(main_col_field, secondary_col_field):
    if main_col_field != secondary_col_field:
        return secondary_col_field
    return main_col_field
```

در زمانی که می‌خواهیم برابری دو فیلد متفاوت از هم را بسنجیم و در صورت برابر نبود آن‌ها مقدار فیلد دوم و در صورت برابر بودن آن‌ها فیلد اول (همان فیلدی است که در حال حاضر در ستون مد نظر مد ما وجود دارد) برگشت داده شود و در دیتاست بنشیند.

• تابع semi_space_correction

```
def semi_space_correction(x, normalizer):  
    return normalizer.normalize(x)
```

بر خلاف زبان انگلیسی در زبان فارسی کلمات مرکب و افعالی وجود دارند که برای خوانتر و زیباتر دیده شدن آن‌ها در متن باید از یک نیم فاصله (semi-space یا half-space) به جای فاصله کامل (space) برای آن‌ها استفاده کرد چرا که باید این نوع کلمات در واحد یک کلمه توسط مخاطب دیده شوند تا متن مورد نظر برای آن‌ها خوانتر باشد. به طور مثال در متون فارسی "آن‌ها" با "آن‌ها" و یا «سه شنبه» با "سه‌شنبه" متفاوت است چرا که در اولی از فاصله کامل و در دومی از نیم فاصله استفاده شده است. به طور خلاصه و کلی نیم فاصله در جایی از متن به کار می‌رود که می‌خواهیم دو کلمه از هم جدا باشند ولی چسبیده دیده شوند و بین آن‌ها فاصله نباشد. این امر موجب خواهد شد که بعداً در کد نویسی آن ترکیب کلمه ایی مد نظر در واحد یک کلمه در دسترس قرار گیرد. در ساختار Unicode Encoding مقدار نیم فاصله با \u200c دیده خواهد شد. درواقع به دید شما در متن چنین چیزی بین ترکیب کلمه ایی مد نظر وجود ندارد ولی حتماً در ساختار Unicode آنرا مشاهده خواهید کرد. تابع فوق به کمک آبجکت normalizer از کلاس Normalizer در کتابخانه hazm می‌توانید در مقدار x، فاصله را به نیم فاصله تبدیل کند.

• تابع update_city_persian_name_fields

```
def update_city_persian_name_fields(data, x_col_name, y_col_name, city_codes_dict):  
    national_code = data[x_col_name]  
    city_name_persian_col_field = data[y_col_name]  
    city_name_persian_dict_value = city_codes_dict[national_code]  
  
    return correct_field(city_name_persian_col_field, city_name_persian_dict_value)
```

هدف ما از نوشتن این تابع این می‌باشد که به ما کمک کند تا فیلدهای درون ستون city_persian_name دیتاست خام خود را در صورت null بودن فیلد مد نظر پر و آپدیت کنیم. توجه شود که آرگومان data یک دیتافریم تهیه شده از روی دیتاست خام می‌باشد و مقدار x_col_name و y_col_name نام ستون‌هایی می‌باشند که علاوه بر اینکه حتماً باید در دیتافریم وجود داشته باشند همچنین باید مقدار y_col_name بر اساس x_col_name تعریف شده باشد چرا که هدف این تابع این می‌باشد که از مقدار x_col_name و y_col_name به کمک دیکشنری city_codes_dict به نام فارسی فرودگاه مد نظر دست یابد و همانگونه که در تصویر مشاهده می‌کنید data[x_col_name] مقدار national_code را به ما می‌دهد (این متغیر می‌تواند برحسب مکان استفاده از این تابع

شناسه فرودگاه مقصد یا مبدأ باشد) و همچنین `data[y_col_name]` مقدار فارسی نام فرودگاه می باشد(این متغیر نیز بر حسب مکان استفاده از این تابع می تواند نام فارسی فرودگاه مقصد یا مبدأ باشد). و در ادامه نام فارسی فرودگاه مد نظر در دیکشنری `city_codes_dict` را توسط کلید `national_code` استخراج کرده و در نهایت نیز توسط تابع `correct_field` که قبلاً آنرا تعریف کردیم می توانیم در مکان هایی که مقدار `null` وجود دارد تغییر مربوطه را ایجاد کنیم.

• تابع `update_departure_date_YMD_format_fields`

```
def update_departure_date_YMD_format_fields(data, x_col_name, y_col_name, months_dict):
    _, day_of_month, month = data[x_col_name].split()
    month_number_str = months_dict[month]
    created_departure_date_YMD_formats = f"1402-{month_number_str}-{day_of_month}"
    departure_date_YMD_formats_col_field = data[y_col_name]

    return correct_field(departure_date_YMD_formats_col_field, created_departure_date_YMD_formats)
```

هدف ما از نوشتن این تابع این می باشد که به ما کمک کند تا فیلدهای درون ستون `departure_date_YMD_formats` دیتاست خام خود را در صورت `null` بودن فیلد مد نظر پر و آپدیت کنیم. توجه شود که آرگومان `data` یک دیتافریم تهیه شده از روی دیتاست خام می باشد و مقدار `x_col_name` و `y_col_name` نام ستون هایی می باشند که علاوه بر اینکه حتماً باید در دیتافریم وجود داشته باشند همچنین باید مقدار `y_col_name` بر اساس `x_col_name` تعریف شده باشد چرا که هدف این تابع این می باشد که از مقدار `x_col_name` و `y_col_name` به کمک دیکشنری `months_dict` به تاریخ حرکت هواپیما دست یابد و همانگونه که در تصویر مشاهده می کنید `data[x_col_name]` مقدار ستون `departure_date` موجود در دیتاست خام را در سه متغیر که جدا از هم می باشند به ما می دهد. و همچنین `data[y_col_name]` مقدار فعلی فیلد `departure_date_YMD_formats` را به ما خواهد داد. و در ادامه شماره ماه مربوطه را از دیکشنری `months_dict` توسط کلید `month` استخراج کرده و در نهایت نیز توسط تابع `correct_field` که قبلاً آنرا تعریف کردیم می توانیم در مکان هایی که مقدار `null` وجود دارد تغییر مربوطه را ایجاد کنیم.

• تابع `update_dependent_col`

```
def update_dependent_col(main_df, func, x_col_name, y_col_name, your_dict):
    secondary_df = difference_drop(main_df, x_col_name, y_col_name)
    # secondary_df = main_df[main_df.columns.intersection([x_col_name, y_col_name])]

    main_df[y_col_name] = secondary_df.apply(
        func=func,
        args=(x_col_name, y_col_name, your_dict),
        axis=1)

    return main_df
```


از طریق این تابع می‌توانیم فیلدهای ستون *y* را به کمک مقادیرهای ستون *x* و دیکشنری مد نظر شما آپدیت کنیم. توجه شود که ستون *y* باید و حتماً به ستون *x* وابستگی داشته باشد. در کد خود از ترکیب این تابع با توابعی مثل `update_city_persian_name_fields` و `update_departure_date_YMD_format_fields` می‌توانیم بهره ببریم. در رابطه با بدنه تابع فوق می‌توان اضا‌حار کرد که از توابعی مثل `difference_drop` و `apply` استفاده شده که در رابطه با این توابع قبلاً توضیحاتی را خدمتتان ارائه کردیم تنها مورد جدیدی که در این قسمت وجود دارد این است که در تابع `apply` علاوه بر آرگومان `func` از آرگومان‌های `axis` و `args` هم استفاده شده که به ترتیب آرگومان `args` برای انتقال پارامترهای مورد نیاز تابع `func` و آرگومان `axis=1` برای ارسال مقادیر ستونی روی هر سطر (ارسال هر سطر دیتافریم) به تابع `func` استفاده شده است.

• تابع `update_flight_number_col`

```
def update_flight_number_col(x, airline_codes_dict):
    company_name = x["company_name"]
    flight_number = x["flight_number"]
    airline_code = airline_codes_dict[company_name]
    return f"{airline_code}-{flight_number}"
```

کد پرواز ترکیبی از حروف و اعداد می‌باشد که با در دست داشتن آن می‌توانیم به اطلاعاتی دیگر مثل ساعت حرکت از مبدا، ساعت ورود به مقصد، کلاس پروازی، فرودگاه مبدأ و مقصد، تعداد توقف‌های هواپیما در بین مسیر و غیره و غیره به راحتی دسترسی داشته باشیم. ما در دیتاست خام خود به صورت پیش‌فرض چنین فقط بخش شماره ایی کد پرواز را در دست داریم اما با استفاده از فایل `airlines_codes.json` که با توجه به شرکت‌های هواپیمایی موجود در دیتاست خود فراهم کرده‌ایم می‌توانیم فرمت این کد پروازی را ایجاد کرده به صورتی که در عکس فوق مشاهده می‌کنید می‌باشد. درواقع به طور کلی هر شرکت هواپیمایی می‌تواند ساختار کد پروازی خودش را داشته باشد اما با مقداری سرچ متوجه خواهیم شد که اکثریت شرکت‌ها از استاندارد زیر استفاده می‌کنند:

`"{airline_code}-{flight_number}"`

• تابع `change_city_names_to_en`

```
def change_city_names_to_en(x_field, airports_info_dict):
    return extract_value_from_json_obj(airports_info_dict, [x_field, "city_name"])
```

این تابع به کمک دیکشنری `airports_info_dict` و از طریق تابع `extract_value_from_json_obj` معادل انگلیسی اسم فارسی فرودگاه بدست می‌آورد و آن را برمی‌گرداند. هر چند باید خاطر نشان کرد که در دیکشنری

airports_info_dict در ابتدا به دنبال کلیدی با مقدار x_field خواهیم گشت و پس آن در دیکشنری بدست آمده مقدار کلید city_name را بر می گردانیم. پس توجه شود ترتیبی که در لیست کلیدها ایجاد می کنید دارای اهمیت در پیدا کردن مقدار نهایی می باشد.

• تابع get_city_airport_names

```
def get_city_airport_names(x_field, airports_info_dict):  
    return extract_value_from_json_obj(airports_info_dict, [x_field, "airport_names"])[-1]
```

دقیقاً به همانند تابع changes_city_name_to_en عمل می کند عمل می کند با این تفاوت که خروجی تابع extract_value_from_json_obj با توجه به ترتیب کلیدهای استفاده شده براساس ساختار فایل airports_info.json یک لیست از نام فرودگاه ها می باشد که در هر لیست فقط آخرین عضو این لیست را بر می گردانیم.

• تابع move_columns

```
def move_columns(main_df, cols_dict):  
    sorted_cols_dict = dict(sorted(cols_dict.items(), key=lambda item: item[1]))  
  
    for col_name, next_position in sorted_cols_dict.items():  
        column_to_move = main_df.pop(col_name)  
        main_df.insert(next_position, col_name, column_to_move)  
    return main_df
```

همانطور که از اسم این تابع پیداست وظیفه مرتب سازی ستون های دیتاست را بر اساس ترتیبی که در متغیر COLUMNS_NEED_TO_MOVE فراهم شده بر عهده دارد. برای انجام این عمل در ابتدا باید تمامی مقادیر موجود در دیکشنری COLUMNS_NEED_TO_MOVE را بر اساس value آنها مرتب سازی کنیم و سپس به ازای جفت Item های key و value موجود در این دیکشنری یک حلقه ایجاد کرده و در ابتدا ستون مربوطه را از دیتافریم مد نظر را توسط تابع pop حذف کرده و درون متغیر column_to_move قرار می دهیم و سپس با استفاده از تابع insert در موقعیت مد نظر درون دیتافریم جای خواهیم داد و بعد از اتمام حلقه دیتافریم آپدیت شده را به عنوان باز می گردانیم.

• تابع replace_with

```
def replace_with(col_df, origin_value, replacement_value, type_value=None):  
    col_df = col_df.replace(origin_value, replacement_value)  
  
    if type_value:  
        col_df = col_df.astype([type_value])  
  
    return col_df
```

این تابع وظیفه جا به جایی مقدار اصلی که فیلد با مقدار جایگزین آن را بر عهده دارد و همچنین با پارامتر type_value می‌تواند نوع یک ستون را توسط تابع astype عوض کند به طور مثال اگر یک ستون از نوع int64 بود می‌توان آن را با این پارامتر به float تبدیل کرد.

• تابع orthodromic_distance

```
def orthodromic_distance(row, airports_geometry_dict):
    departure_airport_code = row["national_departure_code"]
    arrival_airport_code = row["national_arrival_code"]

    departure_airport_coordinate = extract_value_from_json_obj(airports_geometry_dict,
                                                                [departure_airport_code,
                                                                 "geometry"])

    arrival_airport_coordinate = extract_value_from_json_obj(airports_geometry_dict,
                                                            [arrival_airport_code,
                                                             "geometry"])

    return round(haversine(departure_airport_coordinate,
                           arrival_airport_coordinate,
                           Unit.KILOMETERS,
                           normalize=True))
```

این تابع وظیفه تخمین فاصله دایره-بزرگ (فاصله مستقیم یا فاصله هوایی) بین فرودگاه مبدأ و مقصد را بر عهده دارد. آرگومان row یک سطر به صورت دیتافریم می‌باشد که توسط آن می‌توان مقدار ستون national_departure_code و national_arrival_code سطر مد نظر را بدست آورد. توسط این شناسه ها هم می‌توان از دیکشنری airports_geometry_dict مختصات جغرافیای فرودگاه مبدأ و مقصد را با تابع extract_value_from_json_obj و کلیدهای ترتیبی airport_code و geometry بدست آورد. در نهایت نیز با فرمول haversine زیر فاصله دایره-بزرگ بین مبدأ و مقصد محاسبه کرد:

$$\text{hav}(\theta) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)$$

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

where

- φ_1, φ_2 are the latitude of point 1 and latitude of point 2,
- λ_1, λ_2 are the longitude of point 1 and longitude of point 2.

To solve for the distance d , apply the archaversine (inverse haversine) to $h = \text{hav}(\theta)$ or use the arcsine (inverse sine) function:

$$d = r \text{ archav}(h) = 2r \arcsin(\sqrt{h})$$

می‌توانیم خودمان این فرمول را با استفاده از توابع موجود در کتابخانه math پیاده‌سازی کرده و فاصله را محاسبه کنیم اما به جای اختراع کردن دوباره چرخ پکیج haversine را در محیط مجازی ای که ایجاد کردیم نصب می‌کنیم و از تابع haversine که در این کتابخانه موجود است برای محاسبه فاصله دایره-بزرگ استفاده می‌کنیم. این تابع همانطور که می‌بینید فقط از ما طول و عرض جغرافیایی فرودگاه و همچنین یکا محاسباتی (کیلومتر) را از ما می‌گیرد و همچنین اگر مقدار پارامتر normalize برابر با True باشد در صورتی که طول و عرض جغرافیایی شما خارج از محدوده بود آن را در محدوده مجاز نرمال می‌کند. و سپس به محاسبه فاصله haversine می‌پردازد. در نهایت زیر با استفاده از تابع round خروجی بدست آمده را به یک عدد صحیح رند می‌کنیم.

• تابع estimate_flight_length

```
def estimate_flight_length(distance):
    v_km_per_min = randint(885, 965) / 60

    # per minutes
    taxi_time = randint(5, 10)
    attach_stairs_time = randint(5, 10)
    takeoff_clearance = randint(5, 10)
    takeoff_time = randint(10, 20)
    cruise_time = distance / v_km_per_min
    vectoring_time = 10
    busy_airplane_time = 10
    descent_time = randint(10, 15)
    landing_time = randint(10, 20)

    flight_length = descent_time + landing_time + takeoff_time + \
                    taxi_time + attach_stairs_time + takeoff_clearance + \
                    busy_airplane_time + vectoring_time + cruise_time

    return round(flight_length)
```

از این تابع برای محاسبه تخمینی مدت زمان پرواز از مبدأ به مقصد می‌توان استفاده کرد همانطور که مشاهده می‌کنیم در ورودی این تابع از آرگومان distance استفاده کرده‌ایم این آرگومان مقدار فاصله هوایی یا همان فاصله دایره-بزرگ بین مبدأ و مقصد می‌باشد احتمالاً متوجه شده‌اید که بعد از اینکه از تابع orthodromic_distance استفاده کردیم می‌توانیم از تابع فوق برای محاسبه مدت زمان پرواز بهره ببریم. شاید به این فکر بی‌افتید که خب صرفاً تقسیم کردن مقدار فاصله به روی میانگین سرعت هواپیماهای مسافربری می‌تواند خروجی مد نظر ما را تولید

کند ولی باید بگوییم که سخت در اشتباهید چرا که مدت زمان پرواز صرفاً برابر با مقدار تقسیم فاصله به میانگین سرعت نیست و پارامترهای دیگری هم در مدت زمان پرواز دخیل می‌باشند که هرکدام از این عوامل به شرح زیر هستند:

۱. **v_km_per_min** : میزان میانگین سرعت هواپیما مسافربری (کیلومتر بر دقیقه)
 ۲. **taxi_time** : میزان زمان ترک کردن سوله / ترمینال تا بلند شدن هواپیما از زمین
 ۳. **attach_stairs_time** : میزان زمان وصل کردن پله ها به هواپیما برای سوار شدن و پیاده شدن مسافران
 ۴. **takeoff_clearance** : میزان زمانی که طول می‌کشد تا اجازه پرواز از برج مراقبت به خلبان هواپیما داده شود.
 ۵. **takeoff_time** : میزان زمان بلند شدن هواپیما از زمین تا رسیدن به ارتفاع کروز.
 ۶. **cruise_time** : از زمانی که در مبدأ هواپیما به ارتفاع کروز خودش رسید تا زمانی که برای فرود به هر دلیلی (رسیدن به مقصد، توقف اجباری، سقوط و غیره) ارتفاع خودش را از ارتفاع کروز کاهش دهد می‌گویند. در بعضی جاها به این زمان، **flight duration** یا **flight length** نیز گفته می‌شود. درواقع وقتی شما فاصله دایره-بزرگ را تقسیم بر میانگین سرعت هواپیما مسافربری کنید مثل این می‌ماند که دارید **cruise time** را محاسبه می‌کنید.
 ۷. **vectoring_time** : در نظر بگیرید که فاصله ایی که شما بدست آوردید فاصله مستقیم بین مبدأ و مقصد می‌باشد اما مسیر هواپیما هیچ وقت به صورت مستقیم نیست. بلکه می‌توان گفت مسیر هواپیما با تغییر مسیرهایی جزئی در امتداد خط مستقیم بین مبدأ و مقصد می‌باشد و خلبان هواپیما همیشه در بین مسیر به دلایل متفاوت در حین پرواز، تغییر زاویه ایی را در مسیر خود ایجاد خواهد کرد.
 ۸. **busy_airplane_time** : میزان زمانی که هواپیما باید در آسمان معطل بماند تا خطوط هوایی برای فرود خلوت شود. در نظر بگیرید که هواپیما به محض رسیدن به مقصد عملیات کاهش ارتفاع خود آغاز نمی‌کند بلکه در صورت لزوم باید صبر کند تا شرایط برای فرود سالم مهیا شود.
 ۹. **descent_time** : مقدار زمانی که طول می‌کشد تا هواپیما از ارتفاع کروز به ارتفاع فرود برسد.
 ۱۰. **landing_time** : میزان زمانی که طول می‌کشد تا هواپیما از ارتفاع فرود به سطح زمین برسد و توقف کامل در فرودگاه مقصد داشته باشد.
- تمام پارامترهایی که در بالا ذکر شده‌اند بر حسب دقیقه هستند و مجموع تمام پارامترهای بالا نیز مسلماً به دقیقه می‌باشند. تمام بازه های زمانی ای که در عکس فوق در تابع **randint** استفاده شده میانگین مقادیری می‌باشند که در سطح اینترنت گزارش شده‌اند. برای اینکه تخمین ما در خروجی به واقعیت نزدیک‌تر باشد از تابع **randint** برای انتخاب تصادفی در بازه های استفاده شده بهره برده ایم.

```
def estimate_arrival_time(row):
    arrival_time = row["local_arrival_time"]
    if pd.isnull(arrival_time):
        local_departure_timezone = pytz.timezone('Asia/Tehran')
        departure_time_hour, departure_time_minute = row["local_departure_time"].split(":")
        year, month, day = row["departure_date_YMD_format"].split("-")
        flight_length_hour = row["flight_length_min"] / 60
        hour_part = int(flight_length_hour)
        min_part = (flight_length_hour - hour_part) * 60

        departure_time_obj = datetime(
            year=int(year), month=int(month), day=int(day), hour=int(departure_time_hour),
            minute=int(departure_time_minute), tzinfo=local_departure_timezone)

        estimated_arrival_time = departure_time_obj + timedelta(hours=hour_part, minutes=min_part)
        return datetime.strftime(estimated_arrival_time, '%H:%M')
    return arrival_time
```

از این تابع میتوان برای تخمین زمان رسیدن هواپیما به فرودگاه مقصد استفاده کرد اما همانطور که مستحضر هستید تنها زمانی می‌توانید زمان رسیدن به مقصد را تخمین بزنید که پارامترهایی مثل ساعت حرکت از مبدا، فاصله بین مبدا و مقصد، سرعت هواپیما و غیره را داشته باشید پس کاملاً مشخص است که در ابتدا باید از تابع orthodromic_distance و بعد از آن از تابع estimate_flight_length و سپس از تابع فوق استفاده کنید. از آرگومان row ستون local_arrival_time را می‌خوانیم و درون متغیر arrival_time قرار می‌دهیم سپس با تابع isnull به بررسی null بودن این متغیر می‌پردازیم اگر این متغیر مقدارش برابر با null نبود یعنی اینکه سایت هدف زمان رسیدن را به ما ارائه کرده و نیاز نیست که آن را تخمین بزنیم پس مقدار خودش را برمی‌گردانیم اما اگر این مقدار برابر با null بود باید مقدار تخمینی خود را با مقدار null عوض کنیم برای اینکار ابتدا باید توسط تابع timezone کتابخانه pytz یک آبجکت از روی کلاس BaseTzInfo با منطقه زمانی Asia / Tehran ایجاد می‌کنیم چرا که می‌خواهیم زمان ورود هواپیما به مبدا را به منطقه زمانی کشور خودمان ذخیره کنیم. از row ستون local_departure_time را خوانده و توسط تابع split مقادیر درون آن را با ":" از هم جدا کرده و درون متغیرهای مد نظر قرار می‌دهیم. همین عمل را برای ستون departure_date_YMD_format درون row دوباره تکرار می‌کنیم. اینبار از row مقدار flight_length_min که قبلاً آنرا محاسبه کردیم بدست آورده و بر ۶۰ تقسیم می‌کنیم تا میزان ساعت پرواز بر ساعت به دست آید با استفاده از این مقدار قسمت‌های ساعت و دقیقه را محاسبه می‌کنیم حال می‌توانیم با کتابخانه datetime یک آبجکت از نوع کلاس datetime ایجاد کنیم. و همچنین توسط کلاس timedelta می‌توانیم اختلاف زمانی بدست آمده را در قالب یک آبجکت داشته باشیم تا بتوانیم با مقدار departure_time_obj آن را جمع کرده تا زمان تخمینی رسیدن به مقصد را بدست آوریم در نهایت هم با متد strftime می‌توانی خروجی را فرمت "Hour : Minute" به خروجی تحویل دهیم.

• تابع `change_company_name_specific_value`

```
def change_company_name_specific_value(series_value):
    if 1 <= len(series_value) <= 3:
        series_value = series_value.upper()
    elif len(series_value) > 3:
        series_value = series_value.capitalize()

    if series_value == "فری برد":
        return "Freebird"
    elif series_value == "ماوی گوی":
        return "MaviGok"
    elif series_value == "G6":
        return "GlobalX"
    elif series_value == "CPN":
        return "Caspian"
    elif series_value == "IZG":
        return "Zagros"
    elif series_value == "3F":
        return "FlyOne"
    elif series_value == "VRH":
        return "Varesh"

    return series_value
```

برای یک‌دست کردن مقادیر درون فیلد `company_name` از این تابع استفاده می‌کنیم تا یکسری مقادیر خاص را که که دارای شرایط زیر هستند تغییر دهیم:

۱. اگر طول مقدار فیلد مد نظر بین [۱, ۳] بود کل مقدار به حروف بزرگ تبدیل شود.
۲. در غیر این صورت اگر بزرگ‌تر از ۳ بود فقط کلمه اول آن با تابع `Capitalize` بزرگ شود
۳. اگر مقادیر فیلد فارسی باشند با معادل انگلیسی آن‌ها جایگزین شوند.
۴. در غیر این صورت اگر بعضی فیلد‌ها از کدهای `IATA` دو حرفی یا `ICAO` سه حرفی به جای نام شرکت‌ها استفاده شده این مقادیر با نام انگلیسی شرکت جایگزین شوند.
۵. در این صورت اگر هیچ کدام از شرایط بالا ملاقات نشد مقدار فعلی فیلد بدون تغییر برگشت داده خواهد شد.

• تابع `change_flight_class_type_specific_value`

```
def change_flight_class_type_specific_value(x):
    if "ایرباس" in x:
        return x.replace("ایرباس", "Airbus")
    elif "بوئینگ" in x:
        return x.replace("بوئینگ", "Boeing")
    elif "یوئینگ" in x:
        return x.replace("یوئینگ", "Boeing")
    return x
```

این تابع نیز برای یک‌دست کردن مقادیر فیلدهای درون ستون flight_class_type کاربرد دارد در شرایط زیر مقدار فیلد تغییر خواهد کرد:

۱. اگر کلمه "ایرباس" در x وجود داشت باید توسط متد آبجکت String به نام replace مقدار "ایرباس" با Airbus جا به جا شود.

۲. در غیر این صورت اگر کلمه "بوئینگ" یا "بوئینگ" در x وجود داشته باشد باید با Boeing جا به جا شود.

۳. در غیر این صورت مقدار x بدون تغییر برگشت داده شود.

• تابع extract_fare_class_code

```
def extract_fare_class_code(x):
    """Fare classes are typically 'alphanumeric codes' that airlines use to categorize and
    differentiate the various types of tickets they offer, otherwise
    fare classes are codes used by airlines to categorize and manage the pricing and conditions of tickets,
    they provide valuable information about the ticket's flexibility, conditions, and benefits"""

    # Note -> The site we scrapped did not have a specific format to easily extract the fare class.
    # Format -> [AircraftModel][-][RegistrationCode]AirlineClassCode[_D]

    if len(x) in (1, 2):
        # Y, B, L, K, C, J, F, P, RY, RP, RB, etc
        return x
    elif x == "A17":
        return x
    elif x == "MDY" or x == "BoeingMDY":
        # MDY or BoeingMDY
        return "Y"
    elif 'TOR' in x:
        # 'M83TOR', 'MD8TOR', etc
        return None
    elif x.find("_") != -1:
        # 'BAE-87Y_D', 'AB346-MMRC_D', etc
        return x.split("_")[0][1:]
    elif x.find("DEF") == -1 and x[-2].isalpha() and x[-2].isupper():
        # 73HFS, 'MD8RP', 'MD8RY', 'MD8RB', 'Boeing737QQ', Boeing737QQ, etc
        return x[-2:]
    elif x.count("Y") == 1:
        # A310Y33, BoeingMD-83Y, BoeingMD-Y, 'BoeingY', Boeing737Y, '320Y', 'AirbusY', 'Airbus320Y', 'AB6Y',
        # '737Y', 'M80Y', 'DEFY', '100Y', 'B737-800Y', etc
        return x[x.index("Y")+1:]
    # 'AirbusA310W', 'MD8L', '320C', 'DEFJ', 'DEFB', 'AB6W', 'A300B4-203V', 'Boeing737X', 'Boeing737Q', etc
    return x[-1]
```

سایت هدف مقادیر یک دست و تمیزی برای ستون **کلاس نرخ** به ما ارائه نمی‌دهد ولی با بررسی کلاس نرخ شرکت‌ها از طریق سایت رسمی آن‌ها و سرچ در اینترنت و همچنین بررسی مقادیر کلاس نرخ ای که سایت هدف در اختیار ما قرار می‌دهد می‌توان متوجه شد که اکثریت شرکت‌ها از کلاس‌های نرخ تک حرفی یا دو حرفی استفاده می‌کنند تا اطلاعاتی مثل کلاس پروازی، میزان تخفیف بلیت، جریمه کنسلی و غیره را برای بلیت خریداری شده توسط مسافر مشخص کنند. ولی تعداد کمی از شرکت‌ها هم هستند که از کلاس نرخ ۳ حرفی هم استفاده می‌کنند. مثل شرکت هواپیمایی Emirates که از کلاس نرخ A17 برای نمایش کلاس پروازی اکونومی خود استفاده می‌کند. همچنین اگر به بررسی عمیق تر مقادیر کلاس نرخ سایت هدف بپردازیم متوجه می‌شویم که تقریباً از ساختار زیر برای نمایش کلاس نرخ هواپیماها استفاده می‌کند که برای ما مطلوب نیست و باید آن را در دیتاست خود اصلاح کنیم:

[AircraftModel][-][RegistrationCode]AirlineClassCode[_D]

این ساختاری است که بنده با بررسی تعداد زیادی کلاس نرخ متوجه آن شده‌ام و اکثریت کلاس‌های نرخ فراهم شده توسط سایت هدف این ساختار را دنبال می‌کنند. احتمالاً متوجه شده‌اید تنها مقداری که همیشه وجود دارد مقدار AirlineClassCode است و مابقی موارد حالت اختیاری دارند می‌توانند وجود نداشته باشند و ما در تابع فوق به دنبال استخراج بخش AirlineClassCode می‌باشیم به همین دلیل در پیرو این هدف باید شروط زیر را طبق عکس فوق پیاده‌سازی کنیم:

۱. اگر طول x برابر با ۱ و یا ۲ بود آنگاه خود x را برگردان.
 ۲. در غیر این صورت اگر x برابر با MDY یا BoeingMDY بود آنگاه مقدار Y برگردان.
 ۳. در غیر این صورت اگر مقدار TOR درون x وجود داشت آنگاه مقدار None را برگردان (مقدار TOR هر جا بوده سایت هدف مشخص نکرده این بلیت برای چه کلاس نرخ است)
 ۴. در غیر این صورت اگر مقدار " _ " درون x وجود داشت آنگاه یک کاراکتر قبل از " _ " را برگردان.
 ۵. در غیر این صورت اگر تعداد Y درون x برابر با ۱ بود آنگاه از ایندکس Y تا آخر رشته x را برگردان.
 ۶. در غیر این صورت آخرین کاراکتر رشته x را برگردان.
- نکته :** توجه شود که ترتیب این شروط بسیار دارای اهمیت است و باید حتماً رعایت شوند.
- طبق عکس می‌توانید به راحتی متوجه شوید که هر شرط با توجه به کدام مقدار رشته x اجرا خواهد شد نمونه‌هایی را فراهم کرده‌ایم تا فهم این تابع برای شما ساده‌تر شود.

• تابع update_flight_sale_type

```
def update_flight_sale_type(x):  
    if pd.notnull(x):  
        if "سیستمی" in x:  
            return "Scheduled"  
        elif "چارتری" in x:  
            return "Charter"
```

از این تابع نیز برای تبدیل مقادیر درون ستون flight_sale_type به معادل انگلیسی آن‌ها استفاده خواهد شد. همانطور که از تابع بالا مشخص است مقدار x با تابع notnull بررسی خواهد شد اگر این مقدار برابر با null بود تغییری رخ نخواهد داد ولی اگر این مقدار برابر با null نبود، اگر کلمه "سیستمی" در x وجود داشت مقدار Scheduled برگشت داده شود و در غیر این صورت اگر کلمه "چارتری" در x وجود داشت آنگاه رشته Charter برگشت داده خواهد شد.

ماژول `make_processed_dataset.py` :

این ماژول صرفاً یک فایل اجرایی است از تمام توابعی که در فاز دوم به شما توضیح دادیم به عبارتی دیگر علاوه بر تمام توابعی که در ماژول هایی مثل `tickets_preprocessing.py` و `utils.py` وجود دارند از کانفیگ های ماژول `settings.py` و توابع پر کاربرد پکیج `common_utils` و کتابخانه هایی مثل `pandas` و `hazm` بهره برده ایم. درواقع با به اجرا در آوردن این ماژول تمامی موارد گفته شده به اجرا خواهند درآمد و تغییرات لازمه را روی دیتاست خام ما انجام خواهند داد تا بتوانیم از خروجی این فاز یک یک دیتاست تمیز شده می باشد در فازهای بعدی پروژه استفاده کنیم.

فاز سوم پروژه

آماده کردن داده های تمیز شده برای مدل یادگیر (اعمالی مثل نرمال سازی، تبدیل داده ها و غیره)

ماژول `build_features.py`

• تابع `flight_dep_time` :

```
def flight_dep_time(X):
    if int(X[:2]) >= 0 and int(X[:2]) < 6:
        return 'mid_night'
    elif int(X[:2]) >= 6 and int(X[:2]) < 12:
        return 'morning'
    elif int(X[:2]) >= 12 and int(X[:2]) < 18:
        return 'afternoon'
    elif int(X[:2]) >= 18 and int(X[:2]) < 20:
        return 'evening'
    elif int(X[:2]) >= 20 and int(X[:2]) < 24:
        return 'night'
```

از این تابع برای تبدیل ساعت خروج از مبدأ به ۴ دسته داده Categorical استفاده می شود. مقدار `X[:2]` برابر با قسمت ساعت می باشد به طور مثال اگر `X` برابر با "۱۲:۳۳" باشد مقدار `X[:2]` برابر با ۱۲ خواهد بود که برای انجام عمل مقایسه ریاضیاتی، آن را توسط تابع `int` به عدد صحیح تبدیل می کنیم.

• تابع `flight_duration_sec` :

```
def flight_duration_sec(X):
    return int(X) * 60
```

کاملاً واضح است که این تابع با دریافت مدت زمان طول پرواز در متغیر `X` آن ها از دقیقه به ثانیه تبدیل می کند.

• مابقی کدهای درون ماژول `build_features.py` :

- `df['departure date_YMD format']=df['departure date_YMD format'].apply(lambda d: datetime.strptime(d, '%Y-%m-%d'))`

تبدیل نوع مقادیر درون ستون `departure date_YMD format` از آبجکت `String` به آبجکت `datetime` توسط اعمال با تابع `apply` و تابع خطی `lambda` با فرمت تاریخ "Y-m-d"

- `df["year"] = df['departure date_YMD format'].map(lambda x: x.year)`
- `df["month"] = df['departure date_YMD format'].map(lambda x: x.month)`
- `df["day"] = df['departure date_YMD format'].map(lambda x: x.day)`

قطعه کدهای بالا برای تجزیه کردن ستون `departure date_YMD format` به سه ستون سال، ماه و روز می باشد. البته باید توجه شود که مقدار `x` یک آبجکت از نوع `datetime` می باشد به همین دلیل خیلی راحت می توانیم مقادیر سال، روز و ماه را جدا کنیم.

- `# Converting the flight local departure time into proper time`
`df['dep flight time'] = df['local departure time'].apply(flight dep time)`
- `# Converting the flight duration to seconds`
`df['duration sec'] = df['flight length min'].apply(flight duration sec)`

از قطعه کدهای بالا به ترتیب برای تبدیل مقادیر ستون `local departure time` به مقادیر `Categorical` و قرار دادنشان در ستون جدیدی به نام `dep_flight_time` و تبدیل مقادیر `flight_length_min` از دقیقه به ثانیه قرار دادن آنها در ستون جدیدی به نام `duration_sec` استفاده خواهد شد.

- `df.drop(["national departure code",
"departure airport",
"national arrival code",
"arrival airport",
"orthodromic distance KM",
"flight_length_min",
"departure date_YMD format",
"local departure time",
"local arrival time",
"flight_number"],
inplace=True,
axis=1)`

از تابع `drop` برای حذف ستون های فوق استفاده شده است. آرگومان `inplace` به ما این اجازه را خواهد داد که بدون انجام عمل انتساب تغییرات را بروی دیتافریم خود اعمال کنیم. اگر هم مقدار `axis` برابر با ۱ باشد آنگاه ستون های مدنظر در تمام سطرها پاک می شوند.

- `df.to_csv("path of your csv file", index=False)`

بعد انجام تغییرات برای دیتافریم خود توسط متد to_csv می‌توانیم تغییرات اعمال شده را برروی یک فایل csv ذخیره کنیم.

- `# Converting categorical column`

```
df = pd.get_dummies(df, columns=['departure_city', 'arrival_city',  
                                'company name', 'flight sale type',  
                                'fare class code', 'dep flight time'])
```

تابع `get_dummies` در کتابخانه `pandas` برای تبدیل متغیرهای طبقه‌بندی (Categorical) (همچنین به عنوان متغیرهای اسمی شناخته می‌شوند) به مجموعه‌ای از ستون‌هایی با مقدار باینری استفاده می‌شود که اغلب به آنها متغیرهای ساختگی یا متغیرهای نشانگر می‌گویند. این فرآیند هنگام کار با الگوریتم‌های یادگیری ماشینی که نیاز به ورودی عددی دارند ضروری است، زیرا بسیاری از الگوریتم‌ها نمی‌توانند مستقیماً داده‌های طبقه‌بندی را مدیریت کنند.

فایل ژوپیر 1.4: `check-correlation-build-features.ipynb`

- `# Correlation Matrix`

```
df.corr()
```

تابع `corr` ماتریس همبستگی بین ویژگی‌ها را در خروجی اش به ما نمایش خواهد داد از این تابع زمانی استفاده می‌کنیم که تا متوجه شویم کدام ویژگی با ویژگی هدف ارتباط نزدیک تر دارد و یا اینکه کدام ویژگی‌ها به هم مشابه هستند. در خصوص پروژه ایی که ما با آن درگیر هستیم خروجی زیر را مشاهده خواهیم کرد.

	ticket_price_T	year	month	day	duration_sec
ticket_price_T	1.000000	NaN	0.208362	-0.110827	0.405100
year	NaN	NaN	NaN	NaN	NaN
month	0.208362	NaN	1.000000	-0.640668	0.129460
day	-0.110827	NaN	-0.640668	1.000000	-0.073003
duration_sec	0.405100	NaN	0.129460	-0.073003	1.000000

ضریب همبستگی برای سنجیدن میزان همبستگی بین داده‌ها استفاده می‌شود حال اگر:

۱. ضریب همبستگی نزدیک به مقدار ۱ باشد بدین معنی است که ضریب همبستگی مثبت قوی است.
۲. ضریب همبستگی نزدیک به مقدار -۱ باشد بدین معنی است که ضریب همبستگی منفی قوی است.
۳. ضریب همبستگی نزدیک به مقدار ۰.۵ باشد بدین معنی است که ضریب همبستگی مثبت ضعیف است.
۴. ضریب همبستگی نزدیک به مقدار -۰.۵ باشد بدین معنی است که ضریب همبستگی منفی ضعیف است.
۵. ضریب همبستگی نزدیک به مقدار صفر باشد بدین معنی است که همبستگی بین داده‌ها وجود نداشته و داده‌ها با هم رابطه‌ای ندارند.

حال به طبق تصویر بالا داریم:

۱. بین `ticket_price_T` و `ticket_price_T` مقدار ۱ مشاهده شده این بدین معنی است که این دو ویژگی با هم کاملاً یکسان هستند و این منطقی می‌باشد چرا ویژگی `ticket_price_T` با خودش مقایسه شده است. همین مورد برای `year`، `month` و دیگر ویژگی‌ها هم وجود دارد.

۲. بین `ticket_price_T` و `duration_sec` مقدار نزدیک به ۰.۵ مشاهده شده این بدین معنی است این دو ویژگی باهم ارتباط دارند ولی این ارتباط از نوع مثبت ضعیف می‌باشد. ساده‌تر توضیح دهیم یعنی اینکه هرچی مقدار `duration_sec` بیشتر شود مقدار `ticket_price_T` هم بیشتر خواهد شد هرچند که باید توجه کرد این ارتباط همچنان ضعیف است یعنی اینکه `ticket_price_T` صرفاً به `duration_sec` وابستگی ندارد و باید ویژگی‌های دیگر را هم بررسی کنیم.

۳. بین `day` و `month` مقداری نزدیک به ۰.۵- مشاهده شده این بدان معنی است که این دو ویژگی باهم ارتباط داشته و این ارتباط از نوع منفی ضعیف می‌باشد.

۴. همانطور که متوجه شده‌اید در رابطه با ویژگی `year` با دیگر ویژگی‌ها و بالعکس و همچنین ارتباط ویژگی `year` با خودش، مقدار `NaN` گزارش شده است این امر به چند دلیل امکان‌پذیر است اتفاق بی‌افتد:

- **وجود Missing Values** : اگر در ویژگی مد نظر مقدار `null` وجود داشته باشد ضریب همبستگی این ویژگی برابر با مقدار `NaN` خواهد شد.

- **وجود Constant Values** : اگر در ویژگی مد نظر مقادیر یکسانی وجود داشته باشند به طوری که موجب شوند واریانس مجموعه پایین بیاید (هر چه واریانس پایین‌تر باشد مقادیر به خط میانگین نزدیک‌تر خواهند شد این بدان معنی است که مقادیر مجموعه شما خیلی به هم نزدیک هستند) حال اگر همه مقادیر ویژگی مد نظر یکسان باشند در نتیجه تابع `corr` مقدار `NaN` را برای نمایش خواهد داد.

- **Division By Zero** : ممکن است حین محاسبه همبستگی تقسیم بر روی عدد صفر یا یک عدد کوچک خیلی نزدیک به صفر اتفاق می‌افتد که این موجب نمایش `NaN` خواهد شد.

در مورد پروژه ما اگر به دیتاست آماده سازی شده دقت کنید متوجه خواهید شد که در همه سطرهای ستون `year` مقدار ۱۴۰۲ وجود دارد که این همان دلیل نمایش `NaN` در ضریب همبستگی این ویژگی با خودش و بقیه ویژگی‌ها می‌باشد چرا که `Constant Values` وجود دارد.

- `df.describe()`

این متد خلاصه‌ای از اطلاعات آماری هر یک از ستون‌های یک دیتافریم را به شما نمایش خواهد داد. در خصوص پروژه ما خروجی زیر را مشاهده خواهید کرد:

	ticket_price_T	year	month	day	duration_sec
count	1.394700e+04	13947.0	13947.000000	13947.000000	13947.000000
mean	7.323801e+06	1402.0	5.694271	18.196100	9770.793719
std	4.524460e+06	0.0	0.632341	9.195327	2327.331682
min	7.609000e+05	1402.0	5.000000	1.000000	4680.000000
25%	4.778324e+06	1402.0	5.000000	10.000000	8040.000000
50%	7.000000e+06	1402.0	6.000000	20.000000	9540.000000
75%	9.121082e+06	1402.0	6.000000	26.000000	11340.000000
max	5.516013e+07	1402.0	7.000000	31.000000	17760.000000

طبق تصویر بالا داریم:

۱. مقدار انحراف معیار (std) ویژگی year برابر با صفر است این یعنی اینکه تمام مقادیر موجود در ویژگی year مقدار یکسانی دارند.
۲. تعداد مقادیر موجود در ویژگی ticket_price_T برابر ۱۳۹۴۷ می باشد.
۳. مقدار ماکسیمم ویژگی day برابر با ۳۱ می باشد.
۴. مقدار انحراف معیار ویژگی month عدد کوچکی می باشد این بدان معنی است که تنوع مقداری زیادی در تعداد ماه ها نداریم که منطقی است ما در پروژه فقط بلیت های موجود در ماه مرداد، شهریور و مهر را کراول کردیم یعنی مقادیر ما ۵، ۶ و ۷ می باشد.
۵. مقدار انحراف معیار ویژگی ticket_price_T بسیار زیاد می باشد که این نشان از پراکندگی فراوان مقادیر موجود در این ویژگی است.
۶. مقادیری که در ۲۵٪، ۵۰٪ و ۷۵٪ قرار دارد به ترتیب مقادیری هستند که نشانه گر چارک اول، چارک دوم (میانه) و چارک سوم می باشد.

فایل ژوپیتتر 1.5train-and-predict-model.ipynb :

```
scaler = StandardScaler()

X_t = scaler.fit_transform(X)
```

StandardScaler کلاسی برای انجام یک نوع تکنیک پیش پردازش در فریمورک (sklearn) scikit-learn می باشد که برای استانداردسازی ویژگی ها با حذف میانگین و مقیاس بندی به واریانس استفاده می شود. این فرآیند می تواند برای بسیاری از الگوریتم های یادگیری ماشین مفید باشد زیرا به بهبود همگرایی و عملکرد کمک می کند. به عبارتی ساده تر در نظر بگیرید که ویژگی هایی به نام حقوق و سن وجود داشته، این دو ویژگی با هم فاصله مقداری زیادی دارند و این امر تأثیر منفی را در پیش بینی مدل ما خواهند گذاشت برای جلوگیری از این تأثیر می توانیم با یک تبدیل مقیاس این دو ویژگی را در یک بازه عددی مشخص نرمال سازی کنیم. در ابتدا یک آبجکت از کلاس StandardScaler ایجاد می کنیم و سپس توسط متد fit_transform تبدیل مربوطه را روی ویژگی های مد نظر اعمال خواهیم کرد.

- `x_train, x_test, y_train, y_test = train_test_split(X_t, y, test_size=0.2)`

از تابع train_test_split برای جدا کردن داده های تست و آموزش برای تعلیم مدل یادگیر و سنجش پیش بینی آن از طریق روش **نمونه برداری تصادفی بدون جایگزینی** به صورت ۲۰-۸۰ یا ۳۰-۷۰ می توان استفاده کرد. مقدار X_t درواقع ویژگی های مقیاس بندی شده و مقدار y ویژگی هدف ما می باشد. از آرگومان test_size هم برای اندازه تقسیم داده استفاده کرده ایم به این صورت که اگر اگر میزان ۲۰-۸۰ استفاده کنیم یعنی اینکه ۸۰ درصد داده ها برای آموزش مدل و ۲۰ درصد آن ها برای تست استفاده خواهند شد. خروجی این تابع چهار مقداری می باشد که در قطعه کد بالا مشاهده می کنید. x_train و y_train مقادیری می باشند که برای آموزش مدل و y_test و x_test مقادیری هستند که برای تست پیش بینی مدل استفاده خواهند شد.

فاز چهارم پروژه

این فاز مربوط به آموزش دادن مدل و پیش بینی قیمت می باشد. در بخش معرفی پروژه اشاره کردیم، مسأله ایی که ما با آن رو به رو هستیم یک مسأله از نوع **رگرسیون** می باشد چرا که در نهایت می خواهیم هزینه یک بلیت هواپیمایی را پیش بینی کنیم و این مقدار، مقداری از نوع عددی و پیوسته می باشد و مسائلی که در آن ها به پیش بینی یک عدد پیوسته می پردازیم مسائلی هستند که باید به روش رگرسیون حل شوند. پس مدل هایی که در این فاز استفاده می کنیم مدل هایی هستند که به عنوان یک **Regressor** عمل خواهند کرد.

فایل ژوپیتر 1.5train-and-predict-model.ipynb :

```
linear_model = LinearRegression()
bayesian_ridge = BayesianRidge(max_iter=250, alpha_1=0.01)
elastic_net = ElasticNet(alpha=0.8, l1_ratio=0.7, max_iter=300)

model = StackingRegressor(regressors=[linear_model, bayesian_ridge, elastic_net], meta_regressor=elastic_net)
model.fit(x_train, y_train)
```

در قطعه کد بالا از سه کلاس BayesianRidge ، LinearRegression و ElasticNet استفاده شده است این سه کلاس به عنوان یک Regressor روی داده های ما عمل خواهند کرد. کلاس StackingRegressor یک نوع مدل

یادگیر به روش Ensemble می‌باشد که با انباشته کردن (Stacking) اجازه می‌دهد تا با استفاده از خروجی هر تخمینگر (مدل یادگیر) به عنوان ورودی تخمینگر دیگر، از قدرت هر تخمینگر استفاده شود. در بعضی مسائل صرفاً استفاده از یک مدل یادگیر پیش‌بینی خوبی را به عمل نخواهد آورد در نتیجه با ترکیب چند مدل با هم دیگر می‌توان پیش‌بینی بهتری را انجام داد. در نهایت نیز با استفاده از متد fit میتوان مدل یادگیر را روی داده‌های خود با استفاده از x_{train} و y_{train} فیت کرد. سینتکس استفاده از این متد برای تمامی مدل‌های یادگیر یکسان است نمونه‌ها را همراه با مقدار هدف به متد فیت می‌دهیم تا اعمال لازمه را برای آموزش مدل مدنظر به کار ببرد. کلمه فیت اصلاح عامیانه ایی است که برای هر مدل یادگیر چه مدل‌های رگرسیون و چه مدل‌های دسته بندی می‌توانیم به کار ببریم اما اینکه تابع fit دقیقاً برای هر مدل به چه شکل کار می‌کند نیازمند بررسی و مطالعه عمیق‌تر مدل مدنظر می‌باشد اما به طور کلی می‌توان گفت که در مدل‌های رگرسیون زمانی که آن‌ها را فیت می‌کنیم به دنبال یک θ می‌گردیم که این θ ها درواقع ضریب ویژگی‌های مدل مدنظر ما می‌باشند به عبارتی باید گفت که کل عملیات fitting طریقه رسیدن به مقدار مناسب و بهینه ای برای θ های مدل مدنظر خواهد بود. در ادامه در رابطه با آرگومان‌های استفاده شده توضیحاتی را خدمت شما ارائه خواهم داد:

۱. **max_iter** : حداکثر پیمایشی که نیاز است تا به همگرایی برسیم. باید گفته شود که در الگوریتم مدل

یادگیر رگرسیونی مدنظر برای اینکه بهترین مقدار را برای θ ها پیدا کنیم از الگوریتم گرادیان کاهشی بهره خواهیم برد. گرادیان کاهشی نیز همیشه به دنبال یک کمینه محلی برای تابع هزینه مدنظر خودش است. تابع هزینه نیز برای الگوریتم گرادیان کاهشی باید یک تابع محدب باشد که ضرایب θ را به گونه ایی پیدا خواهد کرد که مقدار هزینه یا همان خطای تابع کم باشد.

۲. **alpha_1** : مقدار پارامتر α معروف به پارامتر میزان نرخ یادگیر مدل می‌باشد این پارامتر وظیفه دارد

که از overfit شدن مدل جلوگیری کند به عبارتی وظیفه کنترل ضرایب ویژگی‌ها یا همان θ ها برعهده دارد و همچنین به سرعت همگرا شدن در الگوریتم گرادیان کاهشی کمک خواهد کرد.

۳. **l1_ratio** : وظیفه ایجاد توازن بین تنظیم کننده های (Regularizations) $L1$ و $L2$ را در تابع هزینه

برعهده دارد. وقتی مقدار این آرگومان برابر با ۱ باشد یعنی داریم از Lasso Regression و وقتی برابر با مقدار صفر باشد یعنی داریم از Ridge Regression و همچنین وقتی مقداری بین ۱ و صفر داشته باشد یعنی داریم از ترکیبی از هر دوی این‌ها استفاده می‌کنیم. گفتمی است در هر دوی این‌ها از عبارتی به نام Term استفاده شده که اگرچه اسمشان یکسان است ولی کاربردی متفاوت از هم دارند. عبارت Term در Lasso ضریب ویژگی‌هایی که اثر کمی در پیش‌بینی دارند را صفر می‌کند (به نوعی ویژگی را حذف می‌کند) اما عبارت Term در Ridge تأثیر ویژگی‌های کم اهمیت را کاهش خواهد داد و به نوعی جلوی بزرگ‌تر شدن ضرایب را خواهد گرفت.

۴. **regressors** : به کمک این آرگومان تمامی Base Regressor های ساخته شده را برای انجام عمل

یادگیری به روش Ensemble به مدل StackingRegressor خواهیم داد.

۵. **meta_regressor** : مقدار این آرگومان یک مدل یادگیر می‌باشد که پیش‌بینی‌هایی را بر اساس خروجی‌های ترکیبی آرگومان regressors انجام خواهد داد.

```
y_train_pred = model.predict(x_train)
y_train_pred
```

```
y_test_pred = model.predict(x_test)
y_test_pred
```

در تصویر بالا مشخص است که عمل پیش‌بینی را بر روی داده‌های Train و Test توسط متد predict انجام داده‌ایم و خروجی را در متغیرهای md نظر ذخیره کردیم. حال چرا بر روی نمونه‌های آموزشی خود عمل پیش‌بینی را انجام داده‌ایم؟ این عمل به ما کمک می‌کند که بعداً توسط روش‌های ارزیابی مدل متوجه شویم که مدل ما بر روی خود داده‌های آموزشی که قبلاً آن‌ها را در حین آموزش خود دیده بود چگونه عمل کرده است بدین ترتیب می‌توانیم متوجه overfit یا underfit شدن مدل خود شویم.

```
# The best possible score is 1 which is obtained when the predicted values are the same as the actual values.
"R-squared: ", round(r2_score(y_test, y_test_pred), 3)
```

Python

```
# A perfect "mean squared error" value is 0.0, which means that all predictions matched the expected values exactly.
"Mean squared error: ", round(mean_squared_error(y_test, y_test_pred), 3)
```

Python

```
# A perfect "root mean squared error" value is 0.0, which means that all predictions matched the expected values exactly.
"Root mean squared error: ", round(sqrt(mean_squared_error(y_test, y_test_pred)), 3)
```

Python

```
# A perfect "mean absolute error" value is 0.0, which means that all predictions matched the expected values exactly.
"Mean absolute error: ", round(mean_absolute_error(y_test, y_test_pred), 3)
```

Python

همانطور که در بخش تعریف پروژه اشاره کردم روش‌های ارزیابی یک مدل یادگیر رگرسیونی با استفاده از معیارهای میزان سنجش خطا انجام می‌شود. این معیارها مشخص خواهند کرد پیش‌بینی شما چقدر دچار خطا هستند مطلوب ترین مقدار برای وجود عدم خطا مقدار صفر می‌باشد. خطای صفر نشان می‌دهد که مدل شما تمام پیش‌بینی‌ها را درست انجام داده است که در دنیای واقعی محال ممکن است که پیش‌بینی شما بدون خطا باشد چرا که در دنیای واقعی دیتاهایی که شما برای مدل خود فراهم می‌کنید دارای نویز خواهند بود و این نویزها پیش‌بینی شما را دچار خطا خواهد کرد هرچند که روش‌هایی برای تشخیص نویز و کاهش آن‌ها وجود دارد اما کافی نخواهند بود و باز هم مواردی وجود خواهند داشت که موجب خواهند شد پیش‌بینی شما دارای خطا باشد درواقع تلاش ما در این است که بتوانیم مدل خود را در سمتی بهبود بخشیم که به عدد صفر فقط و فقط نزدیک تر شود. از معیارهای خطایی که ما در این پروژه استفاده کردیم می‌توان موارد زیر را نام برد:

۱. **mean_squared_error** : میانگین مربعات خطاها را اندازه‌گیری می‌کند که در واقع همان میانگین مجذور (مربع کامل - عدد به توان دو) اختلاف بین مقادیر پیش‌بینی شده و مقادیر واقعی آن‌ها می‌باشد. MSE معیاری برای سنجش کیفیت یک مدل است که از مجذور فاصله اقلیدسی به دست می‌آید و همیشه یک مقدار مثبت می‌باشد که با نزدیک شدن به مقدار صفر یعنی خطا کاهش یافته است. این معیار از فرمول زیر پیروی خواهد کرد:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error
n = number of data points
 Y_i = observed values
 \hat{Y}_i = predicted values

۲. **Root of mean_squared_error** : ریشه ی میانگین مربعات خطا می‌باشد. به عبارتی جذر از میانگین مربعات خطا می‌باشد. در برخی مواقع RMSE ترجیح داده می‌شود چرا که به خطاهای بزرگتر وزن بیشتری می‌دهد و آن را به موارد پرت حساس می‌کند. این بدان معنی است که خطاهای بزرگتر تأثیر نسبتاً بیشتری بر مقدار RMSE دارند. معیار خوبی برای زمانی است که می‌خواهید انحرافات قابل توجه بین مقادیر واقعی و پیش‌بینی شده را بدست آورید.

۳. **mean_absolute_error** : میانگین خطای مطلق، میانگین تفاضل بین مقادیر واقعی و خروجی مدل است. علامت خروجی نادیده گرفته می‌شود چرا که اگر علامت را نادیده نگیریم، MAE محاسبه شده

احتمالاً بسیار کمتر از تفاوت واقعی بین مدل و داده خواهد بود. میانگین خطای مطلق از فرمول زیر پیروی خواهد کرد:

$$MAE = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

MAE = mean absolute error
 y_i = prediction
 x_i = true value
 n = total number of data points

۴. **r2_score** : در مورد اینکه یک مدل رگرسیون چقدر خوب با داده‌های واقعی فیت شده است استفاده می‌شود. به عنوان مثال، مدلی با مجذور r حدود ۸۰ درصد به این معنی است که ۸۰ درصد داده‌ها با مدل رگرسیونی فیت شده‌اند. از حالتی که بیان شد، می‌توان فهمید که مقادیر بالاتر مربع r نشان دهنده مدل بهتری است. اگرچه، این همیشه درست نیست. در برخی موارد بسیار نادر، ممکن است مقدار مربع r پایین را نیز بخواهیم. مقدار مربع r از صفر تا ۱ است که صفر نشان دهنده یک مدل ضعیف و ۱ نشان دهنده یک مدل مناسب است. چیزی که در اینجا باید به آن توجه کرد این است که **r2_score** در **sklearn** می‌تواند یک مقدار منفی نیز باشد این یعنی اینکه مدل شما حتی از حالت صفر هم بدتر کار می‌کند. این زمانی اتفاق می‌افتد که خط پیش‌بینی رگرسیون شما بدتر از حالت مقدار میانگین عمل می‌کند. همچنین، نکته‌ای که باید به خاطر داشته باشید این است که مقدار **r2_score** تحت تأثیر مقادیر پرت نیز قرار می‌گیرد. در نهایت این معیار از فرمول زیر پیروی خواهد کرد:

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Here,

SSres = Residual sum of squared errors

SStot = total sum of squared errors

از فرمول بالا می‌توان متوجه شد که هرگاه دقیقاً پیش‌بینی‌های مدل مشابه با مقادیر واقعی باشد صورت کسر همواره صفر خواهد شد و در نتیجه تفاضل ۰ - ۱ برابر با ۱ خواهد بود که همانطور که گفته شد وقتی مقدار

r^2_score برابر با عدد یک باشد یعنی ما یک مدل مناسب را دارا هستیم اما در مقابل اگر صورت کسر برابر با مخرج کسر باشد طبیعتاً تفاضل ۱ با ۱ را خواهیم داشت که برابر با صفر می‌باشد و این یعنی اینکه مدل ما ضعیف عمل کرده است.

فصل ۴

خروجی های پروژه و اصلاحات مورد نیاز

در این بخش از داکيومنت به ارائه خروجی های پروژه می پردازیم و به قسمت هایی از پروژه اشاره خواهیم کرد که می توان در جهت گرفتن خروجی مناسب تر و بهتر، بیشتر روی آن ها کار کرد.

خروجی ها

خروجی ها در چهار فاز آماده شده و خدمت شما در زیر ارائه خواهند شد:

۱. با مراجعه به دایرکتوری `.../mostafa_vahdani_bachelor_project/data/raw/` می توانید خروجی فاز

اول پروژه را در فایل `CSV` به نام `flight_tickets_dataset` مشاهده کنید. این فایل شامل تمام داده های خام ممکن است که سایت هدف از تاریخ `۱۴۰۲/۰۵/۱۷` تا تاریخ `۱۴۰۲/۰۷/۱۵` در اختیار ما قرار داده و مورد کراول قرار گرفته اند.

۲. خروجی فاز دوم پروژه در دو مرحله تهیه شده است در مرحله اول شما با مراجعه به دایرکتوری /

`.../mostafa_vahdani_bachelor_project/data/processed` فایل `CSV` به نام `flight_tickets_dataset` را مشاهده کنید. این فایل شامل تمام داده های تمیز شده ایی است که از خروجی فاز اول پروژه بدست آورده و از مرحله پیش پردازش گذر داده ایم. و در مرحله دوم نیز شما با مراجعه به دایرکتوری / `.../mostafa_vahdani_bachelor_project/data/interim` می توانید فایلی با فرمت `CSV` به نام `build_features_flight_tickets_dataset` مشاهده کنید. این فایل شامل تمام داده های است که برای آموزش دادن مدل یا مدل های مد نظر به آن ها نیاز داریم.

۳. و در نهایت نیز با مراجعه به دایرکتوری `.../mostafa_vahdani_bachelor_project/data/interim/`

می توانید خروجی فاز سوم پروژه را در فایل `CSV` به نام `final_flight_tickets_dataset` مشاهده کنید. این فایل شامل تمام داده های آماده سازی شده برای مدل های یادگیر مد نظر می باشد.

۴. در فاز چهارم پروژه همانطور که می دانید و در بخش توضیح کدها مشاهده کردید به آموزش، پیش بینی و ارزیابی مدل خود پرداخته ایم. خروجی زیر نشان دهنده ارزیابی ما از پیش بینی مدل یادگیر بر روی داده های آموزش دیده می باشد:

نکته : توجه کنید که ارزیابی را روی داده پیش بینی شده توسط داده های آموزش دیده بر روی خود مدل انجام داده ایم نه ارزیابی داده های پیش بینی شده توسط داده های تست.

- ('R-squared: ', 0.734)
- ('Mean squared error: ', 5386668392338.481)
- ('Root mean squared error: ', 2320919.73)
- ('Mean absolute error: ', 1425455.501)

از میزان خطای بدست آمده باید متوجه شده باشید که اولاً خروجی تابع `r2_score` به تنهایی برای ارزیابی مدل های ما کافی نمی باشد چرا که می بینید از نظر معیار `r2_score` مدل ما خوب روی داده های آموزشی فیت شده است و دوماً از نظر معیارهای دیگر حتی مدل روی انجام عمل پیش بینی بر روی داده های آموزش دیده خودش هم اصلاً خوب عمل نکرده و میزان خطای بدست آمده با عدد صفر فاصله بسیار زیادی دارد. برای بهبود هر چه بهتر این میزان و نزدیک تر شدن آن به عدد صفر در بخش اصلاحات چند راهکار را ارائه خواهیم داد.

اصلاحات

۱. با توجه به بالا بودن معیارهایی مثل MSE و MAE میتوان نتیجه گرفت که یا از مدل مناسبی استفاده نمی کنیم و یا دیتاست ما به خوبی پیش پردازش نشده است. با استفاده کردن از مدل های رگرسیونی مختلف با پارامترهای متفاوت به طور قطع می توان گفت که نقص کار از انتخاب مدل یادگیر نمی باشد و ما باید بر روی آماده سازی داده ها بیشتر کار کنیم. چرا که آماده سازی هرچه بهتر داده ها از انتخاب مدل یادگیر اولویت بسیار بالا و حیاتی تری دارد.

۲. می توانید ویژگی های مرتبط بیشتری را اضافه و یا ویژگی های نامرتبط را حذف کنید و یا به دنبال این باشید که مقادیر بهتر و دقیقتری را برای ستون مد نظر پیدا کنید به طور مثال هنگامی که ما در حال محاسبه فاصله دایره-بزرگ (فاصله هوایی) در تابع `orthodromic_distance` بودیم تعداد توقف هایی که بین مسیر وجود دارند و همچنین موقعیت جغرافیایی ایستگاه های توقف را در نظر نگرفتیم. ممکن است در یک پرواز، مسیر مستقیم (بدون توقف) بین مبدأ و مقصد وجود نداشته باشد و هواپیما مربوطه مجبور باشد بین مسیر توقف هایی را انجام دهد. گفتنی است به پروازهایی که دارای توقف هستند `Connecting Flight` و یا `Direct Flight` و به پروازهای بدون توقف `Non-Stop Flight` گفته می شود. اطلاعات مربوط به توقف یا عدم توقف هواپیما را می توان توسط **کد پرواز** تهیه شده در دیتاست در ستون `flight_number` پیدا کرد.

۳. با در نظر گرفتن مورد ۲ که اشاره کردیم باید حواستان باشد که اگر تعداد توقف های شما بیشتر شود پس مدت زمان پرواز شما نیز بیشتر خواهد شد اینجاست که باید `Layover Time` را با بدست آوردن تعداد توقف ها بدست آورید و در تابع `estimate_flight_length` آن را اضافه کنید تا تخمین بهتری را از مدت زمان پرواز بدست آورید و نتیجه را در ستون های `flight_length_min` و `local_arrival_time` جایگذاری کنید.

۴. با توجه به اینکه سایت هدف به درستی کلاس پروازی (اکونومی، بیزنس، فرست و غیره) را برای بلیت های خود مشخص نمی کرد، شما می توانید با استفاده از کلاس نرخیه تهیه شده و نام شرکت هواپیمایی به دنبال کلاس پروازی شرکت بگردید و آن ها را در دیتاست خود جایگذاری کنید چرا که کلاس پروازی نقش موثری در تعیین قیمت بلیت های هواپیمایی خواهد داشت.

۵. به جای استفاده از روش `get_dummies` می‌توانید از روش‌های دیگر برای تبدیل داده Categorical به داده عددی (ایندکس گذاری شده) استفاده کنید چرا که اگر کاردینالیتی داده‌های Categorical ها دیتاست شما بالا باشد (تعداد داده‌های Categorical یکتا درون ستون‌های دیتاست زیاد باشد)، تبدیل آن‌ها به ستون‌هایی با مقادیر باینری صرفاً تعداد ستون‌های شما را زیاد می‌کند و ستون بیشتر به معنی ویژگی بیشتر می‌باشد و استفاده از تعداد ویژگی‌هایی زیاد از حد می‌تواند روی پیش‌بینی مدل شما تأثیر منفی بگذارد چرا که ابعاد ویژگی‌های شما بیشتر شده اند.

۶. قیمت بلیت هواپیما در روزهای تعطیل، مناسبت‌های مختلف و بعضی فصل‌های سال به دلیل تقاضای بیشتر مسافران افزایش می‌یابد مخصوصاً تعطیلاتی که براساس ماه‌های قمری می‌باشند برای نمونه روز میلاد امام رضا به طور قطع بر روی قیمت بلیت مشهد تأثیرگذار خواهد بود. می‌توانید با دسته بندی کردن انواع تعطیلات و اختصاص دادن آن‌ها به تاریخ‌های مربوطه ورودی‌ای با جزئیات مرتبط بیشتر برای مدل خود فراهم کنید تا پیش‌بینی دقیق‌تری داشته باشید.

هدف ما در نهایت این بوده که بتوانیم ارزان‌ترین قیمت بلیت را در یک بازه زمانی مشخص پیش‌بینی کنیم برای این امر می‌توان بعد از اینکه آموزش، پیش‌بینی و ارزیابی‌های مد نظر را انجام دادیم با ذخیره پیش‌بینی‌ها و نمونه‌های مد نظر و سپس مینیمم‌گیری روی قیمت بلیت در بازه زمانی دلخواه، ارزان‌ترین قیمت بلیت را بازه زمانی مد نظر قبل از سفر خود پیدا کنیم.

فصل ۵

جمع‌بندی و نتیجه‌گیری

در این داکيومنت از ابتدا سعی بر این بوده که شما خواننده گرامی را از اهمیت و طریقه استفاده از اطلاعات آگاه کرده، به تعریف پروژه و اهداف مد نظر پرداخته، ساختار دایرکتوری ها و ماژول ها را به شما توضیح داده، به توضیح بخش بخش کدها بپردازیم، خروجی های پروژه را در فازهای متفاوت به شما توضیح دهیم و در نهایت به توضیح نکات تکمیلی پروژه در جهت بهبود آن بپردازیم و برای اصلاح آن‌ها راه‌حلهایی را پیشنهاد دهیم. در نهایت باید گفت که دنیای علم داده بسیار دنیای عظیمی می‌باشد و اگر شما علاقه به ادامه در این زمینه به عنوان یک متخصص علم داده را دارید و می‌خواهید در این زمینه پروژه‌ای به عمل می‌آورید همانطور که در مقدمه نیز به آن اشاره کردم نه تنها باید برنامه نویس خوبی باشید بلکه باید از علوم‌های دیگر به‌خصوص ریاضیات سر در بیاورید، مطالعه عمیق و متوالی و همچنین تمرین دائمی در این زمینه داشته باشید تا بتوانید با علم به اینکه می‌دانید دقیقاً دارید چه کاری انجام می‌دهید پروژه‌های خود را جلو برده و به اتمام برسانید.

در پایان نیز از شما خواننده گرامی تشکر می‌کنم که به این داکيومنت توجه لازمه را داشته و آن را مطالعه کرده اید، امیدوارم که این پروژه و داکيومنت برای شما مفید و ارزشمند واقع شود و نکات خوبی را از آن یاد گرفته باشید تا بتوانید بعدها از آن نکات در پروژه‌های خویش بهره ببرید.

منابع

- www.geeksforgeeks.org
- medium.com
- en.wikipedia.org
- scikit-learn.org
- stackoverflow.com
- stackexchange.com
- www.datacamp.com
- www.kaggle.com
- www.linkedin.com
- github.com
- www.gps-coordinates.net
- sites.utoronto.ca/webdocs/HTMLdocs/Book/Book-3ed/appe/iso3166.html
- <https://www.iata.org/en/publications/directories/code-search/>
- <https://metar-taf.com/airport/>
- <https://www.flightradar24.com/data/airlines/#t>
- https://en.wikipedia.org/wiki/List_of_airline_codes
- https://en.wikipedia.org/wiki/List_of_aircraft_registration_prefixes#Post-1928_table_notes
- https://en.wikipedia.org/wiki/List_of_aircraft_type_designators