طراحی DSL تحلیل داده های آزمایشگاهی

مستندات جامع فنی و پیاده‌سازی و تست سناریو

پروژه پایانی درس کامپایلر

اعضای تیم پروژه:

محمدعلی احمدیان
علی جباری پور
سید احمد موسوی مالواجردی

تاریخ ارائه:

۱۰ بهمن ۱۴۰۴

فهرست مطالب

۱ مقدمه و نمای کلی سیستم

این پروژه یک `Compiler` برای یک زبان خاص‌منظوره[1] (DSL) فارسی در حوزه‌ی **تحلیل داده های آزمایشگاهی** است. هدف اصلی سیستم این است که:

- کاربر بتواند دستورات تحلیل داده را به زبان فارسی (نیمه طبیعی) و یا حتی انگلیسی، بنویسد.
- این دستورات به یک DSL میانی ساخت‌یافته و سپس به `Python + Pandas + Matplotlib + Seaborn` تبدیل شوند.
- در نهایت علاوه بر اجرای کد، موارد زیر تولید شود:
  - کد پایتون تولید شده که اگر همین کد را به تنهایی اجرا کنیم، کار هایی که از DSL خواستیم را برایمان انجام میدهد ولی نتایج cpmiler و ریزگزارشات و درخت را دیگر نمیسازد!
  - گزارش متنی آماری (`report.txt`)
  - نمودارها (`.png`) در پوشه‌ی `plots` و همچنین یک تصویر از AST ساخته شده توسط دستورات ما
  - فایل `ast.dot` برای مصورسازی درخت نحو (`AST`)

۱.۱ معماری کلان سیستم

معماری پروژه به طور خلاصه شامل این مراحل است:

۱. **نگاشت فارسی به DSL میانی:** تبدیل الگوهای فارسی به دستورات DSL انگلیسی‌مانند (مثلاً " بگیر از ... به نام ... " به " LOAD ... INTO ... ")

۲. **تحلیل نحوی (Parsing) :** استفاده از گرامر `EBNF` در `Lark` برای تولید `AST`.

۳. **تولید کد (Code Generation) :** پیمایش `AST` توسط `Transformer` و تولید کد پایتون، گزارش آماری و دستورات رسم نمودار.

۴. **اجرا و خروجی:** اجرای کد تولید شده در محیطی کنترل‌شده و ذخیره‌ی کد، گزارش، نمودارها و شِمای `AST`.

این ساختار عملاً یک **pipeline کامپایلر کامل** را نشان می‌دهد از زبان سطح بالا (DSL فارسی) تا کد اجرایی پایتون.

۲ کتابخانه‌های مورد استفاده و نقش هر کدام

در ابتدای فایل پایتون، کتابخانه‌های زیر فراخوانی شده‌اند:

```python
import re
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from lark import Lark, Transformer
from datetime import datetime
```

تحلیل نقش هر ماژول

- **`re`**: برای کار با عبارات باقاعده .(Regex) هسته‌ی نگاشت فارسی به DSL از این ماژول استفاده می‌کند.
- **`os`**: برای کار با فایل‌ها و پوشه‌ها:
  - ساخت پوشه‌ی `output` و `plots`
  - بررسی وجود فایل ورودی
  - ساخت مسیرهای نسبی برای نمودارها و کد تولیدی

[1] `Domain Specific Language`

- **pandas**: کتابخانه‌ی اصلی تحلیل داده. هر LOAD، FILTER، GROUPBY، MERGE، NORMALIZE و غیره در نهایت به توابع pandas نگاشت می‌شود.
- **matplotlib.pyplot (plt) & seaborn (sns)** : برای رسم نمودار های
  - نمودارهای هیستوگرام، خطی، پراکندگی، جعبه‌ای با matplotlib
  - نقشه‌ی حرارتی (Heatmap) با seaborn
- **lark.Lark, lark.Transformer** :
  - Lark : تحلیل نحوی بر اساس گرامر DSL
  - Transformer : تبدیل AST به کد پایتون (Code Generation)
- **datetime** : برای ثبت زمان عملیات در گزارش آماری (timestamp در لاگ‌ها).

۳ موتور ترجمه فارسی به DSL میانی: کلاس PersianToDSLMapper

۱.۳ انگیزه و ضرورت طراحی این لایه

پردازش مستقیم زبان طبیعی فارسی با استفاده از گرامرهای رسمی (نظیر Context-Free Grammar) به دلایل متعددی بسیار دشوار، شکننده و پرهزینه است. از جمله این چالش‌ها می‌توان به موارد زیر اشاره کرد:

- تنوع بسیار زیاد ساختارهای نحوی در زبان فارسی،
- آزادی بالای ترتیب کلمات در جملات،
- وجود مترادف‌ها و بیان‌های مختلف برای یک مفهوم واحد،
- دشواری نگهداری و توسعه گرامرهای بزرگ زبان طبیعی.

به همین دلیل، در این پروژه به‌جای تعریف مستقیم گرامر برای جملات فارسی، یک معماری دومرحله‌ای اتخاذ شده است. در این معماری:

۱. ابتدا جملات فارسی کاربر با استفاده از عبارات باقاعده (Regular Expressions) به یک **زبان میانی (Inter- DSL mediate)** ساده، محدود و ساخت‌یافته تبدیل می‌شوند؛

۲. سپس روی این زبان میانی، یک گرامر رسمی و دقیق با استفاده از ابزار Lark تعریف می‌شود.

کلاس PersianToDSLMapper مسئول پیاده‌سازی مرحله‌ی اول این فرآیند است و نقش یک «مترجم سطح بالا» از زبان طبیعی فارسی به DSL میانی را ایفا می‌کند.

۲.۳ معماری و ساختار کلاس `PersianToDSLMapper`

شکل ۱ نمای کلی ساختار این کلاس را نشان می‌دهد.

```python
class PersianToDSLMapper:
    def __init__(self):
        self.rules = [
            (r'از بگیر "([^"]+)" به نام (\w+)', r'LOAD "\1" INTO \2'),
            (r'خلاصه (\w+)', r'DESCRIBE \1'),
            (r'نمایش (\w+) : (\d+) اول سطر', r'HEAD \1 \2'),
            (r'پاکسازی (\w+) : حذف تکراری', r'CLEAN \1 DROP_DUPLICATES'),
            (r'پاکسازی (\w+) : حذف کلی (خالی|پرت)', r'CLEAN \1 DROP_ALL \2'),
            (r'پاکسازی (\w+) : حذف (\w+) (خالی|پرت)', r'CLEAN \1 DROP_SPECIFIC \2 \3'),
            ...
        ]

    def translate(self, text):
        for p, r in self.rules:
            text = re.sub(p, r, text)
        return text.strip()
```

شکل ۱: ساختار کلی کلاس `PersianToDSLMapper`

در این کلاس، مجموعه‌ای از قواعد نگاشت به صورت یک لیست از زوج‌های `(pattern, replacement)` نگهداری می‌شود. هر قاعده بیان می‌کند که:

- یک الگوی خاص در زبان فارسی چگونه شناسایی شود،
- و به چه دستور استانداردی در DSL میانی تبدیل گردد.

فرآیند ترجمه به‌صورت ترتیبی انجام می‌شود؛ به این معنا که ورودی کاربر به ترتیب از روی تمام قواعد عبور داده شده و هر قاعده‌ای که منطبق باشد، جایگزینی متناظر خود را اعمال می‌کند.

۳.۳ تحلیل قواعد نگاشت (`Mapping Rules`)

هر عنصر از `self.rules` شامل دو بخش اصلی است:

- **Pattern (الگو):** یک عبارت باقاعده که ساختار یک دستور فارسی خاص را شناسایی می‌کند. این الگو می‌تواند شامل گروه‌های کَپچر باشد تا اجزای متغیر جمله (مانند نام دیتافریم یا ستون) استخراج شوند.
- **Replacement (جایگزین):** یک رشته‌ی متنی در قالب DSL میانی که با استفاده از گروه‌های استخراج‌شده، دستور معادل استاندارد را تولید می‌کند.

به عنوان مثال، جملات فارسی زیر:

- « پاکسازی df حذف همه خالی‌ها »
- « پاکسازی df حذف ستون age که خالی است »
- « پر کردن مقادیر خالی با میانگین در df »

می‌توانند به صورت دستورات DSL زیر نگاشت شوند:

- “ خالی `CLEAN df DROP_ALl` ”
- “ خالی `CLEAN df DROP_SPECIFIC age` ”
- “ میانگین خالی `CLEAN df FILL_ALL` ”

این استانداردسازی باعث می‌شود که تنوع بالای زبان طبیعی به مجموعه‌ای کوچک و کنترل‌شده از دستورات تقلیل یابد.

۴.۳ مزایای استفاده از زبان میانی

استفاده از لایه‌ی DSL میانی مزایای مهمی به همراه دارد:

- گرامر `Lark` تنها با یک زبان ساده، صریح و بدون ابهام سروکار دارد؛
- افزودن قابلیت‌های جدید صرفاً با اضافه‌کردن یک قاعده‌ی نگاشت انجام می‌شود، بدون نیاز به تغییر گرامر اصلی؛
- خطاهای نحوی فارسی پیش از ورود به مرحله‌ی پارس کاهش می‌یابند؛
- امکان ترکیب دستورات فارسی و انگلیسی در یک اسکریپت فراهم می‌شود.

در نتیجه، کلاس `PersianToDSLMapper` نقش یک لایه‌ی حیاتی در افزایش پایداری، توسعه‌پذیری و خوانایی کل کامپایلر را ایفا می‌کند.

۴ تعریف گرامر DSL در `Lark`

۱.۴ نمای کلی گرامر

گرامر DSL پروژه با استفاده از ابزار `Lark` و در قالب یک رشته‌ی چندخطی (`raw string`) تعریف شده است. این گرامر، زبان میانی استانداردی را توصیف می‌کند که پس از ترجمه‌ی جملات فارسی توسط لایه‌ی `PersianToDSLMapper`، به عنوان ورودی مرحله‌ی پارس و تولید AST استفاده می‌شود.

```
grammar = r"""
start: statement+

statement: load_stmt
         | duplicate_stmt
         | save_stmt
         | clean_stmt
         | calc_stmt
         | plot_stmt
         | filter_stmt
         | filter_range_stmt
         | level_stmt
         | sort_stmt
         | groupby_stmt
         | drop_col_stmt
         | describe_stmt
         | head_stmt
         | filter_complex_stmt
         | search_stmt
         | merge_stmt
         | create_col_stmt
         | rename_stmt
         | convert_time_stmt
         | normalize_stmt
         | corr_stmt

load_stmt: "LOAD" STRING "INTO" ID

...
"""
```

۲.۴ Non-Terminals (قواعد غیربنیادی)

گرامر ریشه‌ی

```
start: statement+
```

این قاعده بیان می‌کند که یک برنامه‌ی DSL شامل **یک یا چند دستور** است که به صورت ترتیبی اجرا می‌شوند.

دستورها (statement) هر دستور DSL می‌تواند یکی از حالت‌های زیر باشد:

- بارگذاری داده
- پاکسازی و پیش‌پردازش
- محاسبات آماری
- فیلتر و جستجو
- رسم نمودار
- تبدیل و مهندسی ویژگی
- تحلیل آماری و همبستگی

در ادامه، تمام های‌statement گرامر به تفصیل توضیح داده می‌شوند.

—

۳.۴ دستور بارگذاری داده (load_stmt)

```
load_stmt: "LOAD" STRING "INTO" ID
```

این دستور برای بارگذاری یک فایل داده (مانند CSV یا Excel) و ذخیره‌ی آن در یک متغیر (دیتافریم) استفاده می‌شود.
ساختار کلی:

```
LOAD ”file.csv” INTO df
```

مفهوم: فایل مشخص‌شده خوانده شده و به صورت یک دیتافریم با نام داده‌شده در حافظه قرار می‌گیرد.

—

۴.۴ دستور ذخیره‌سازی (save_stmt)

```
save_stmt: "SAVE" ID "TO" STRING
```

این دستور برای ذخیره‌ی یک دیتافریم در قالب فایل خروجی استفاده می‌شود.
مثال:

```
SAVE df TO ”output.csv”
```

—

۵.۴ دستور کپی داده (duplicate_stmt)

```
duplicate_stmt: "DUPLICATE" ID "TO" ID
```

یک کپی مستقل از یک دیتافریم ایجاد می‌کند.
مثال:

```
DUPLICATE df TO df_backup
```

—

۶.۴ دستورات پاکسازی (clean_stmt)

```
clean_stmt: "CLEAN" ID clean_op
```

این دستور مجموعه‌ای از عملیات پاکسازی داده را پوشش می‌دهد.

داده‌ها حذف

```
drop_all: "DROP_ALL" TARGET
drop_specific: "DROP_SPECIFIC" ID TARGET
```

مثال:

```
CLEAN df DROP_ALL خالی
```

داده‌ها کردن پر

```
fill_all: "FILL_ALL" TARGET METHOD
fill_specific: "FILL_SPECIFIC" ID TARGET METHOD
```

مثال:

```
CLEAN df FILL_ALL خالی نمیانگین
```

تکراری داده‌های حذف

```
drop_duplicates: "DROP_DUPLICATES"
```

—

۷.۴ دستورات محاسبات آماری (calc_stmt)

```
calc_stmt: "CALC" ID calc_op+
calc_op: MEAN "OF" ID
       | STD "OF" ID
```

این دستور برای محاسبه‌ی شاخص‌های آماری مانند میانگین و انحراف معیار استفاده می‌شود.
مثال:

```
CALC df MEAN OF age STD OF salary
```

—

۸.۴ دستور رسم نمودار (plot_stmt)

```
plot_stmt: "PLOT" ID PLOT_TYPE "OF" (ID|ALL) "IN" (ID|ALL)
```

این دستور انواع نمودارهای تحلیلی را پشتیبانی می‌کند.
مثال‌ها:

- `PLOT df هیستوگرام OF age IN ALL`
- `PLOT df پراکندگی OF salary IN age`
- `PLOT df HEATMAP OF ALL IN ALL`

—

۹.۴ دستورات فیلتر (filter_stmt و filter_range_stmt)

```
filter_stmt: "FILTER" ID "WHERE" ID COMP (STRING|NUMBER)
filter_range_stmt: "FILTER_RANGE" ID ID NUMBER NUMBER
```

مثال:

```
FILTER df WHERE age >= 18
```

—

۱۰.۴ فیلتر شرطی پیچیده (filter_complex_stmt)

```
filter_complex_stmt: "FILTER_COMPLEX" ID CONDITION_EXPR
```

این دستور امکان استفاده از شروط ترکیبی و پیچیده را فراهم می‌کند.
—

۱۱.۴ دستور جستجو (search_stmt)

```
search_stmt: "SEARCH" ID "IN" ID "CONTAINS" STRING
```

مثال:

```
SEARCH df IN name CONTAINS "Ali"
```

—

۱۲.۴ سطح‌بندی داده (level_stmt)

```
level_stmt: "LEVELING" ID ID level_op+
level_op: NUMBER ":" ID
```

برای تبدیل مقادیر عددی به دسته‌های معنایی استفاده می‌شود.
مثال:

```
LEVELING df score 0:low 0.5:medium 0.8:high
```

—

۱۳.۴ مرتب‌سازی (sort_stmt)

```
sort_stmt: "SORT" ID "BY" ID ORDER
```

—

۱۴.۴ گروه‌بندی و تجمیع (groupby_stmt)

```
groupby_stmt: "GROUPBY" ID "BY" ID "OP" AGG_FUNC "OF" ID
```

مثال:

```
GROUPBY df BY gender OP mean OF salary
```

—

۱۵.۴ سایر دستورات

- DROP_COL: حذف ستون
- RENAME: تغییر نام ستون
- CREATE_COL: ایجاد ستون جدید
- CONVERT_TIME: تبدیل ستون زمانی
- NORMALIZE: نرمال‌سازی داده
- DESCRIBE: خلاصه آماری
- HEAD: نمایش سطرهای ابتدایی
- CORRELATE: محاسبه‌ی همبستگی

این مجموعه دستورات، یک زبان DSL قدرتمند، منعطف و قابل توسعه برای تحلیل داده فراهم می‌کند.

۵ مصورسازی AST : تابع ast_to_dot

برای مشاهده‌ی ساختار AST و استفاده در مستندات/دیباگ، تابع ast_to_dot در نظر گرفته شده است:

```python
def ast_to_dot(tree, output_name="ast", output_dir="outputs"):
    """
    Generates AST image (PNG) using Graphviz and saves it in outputs directory.
    """

    os.makedirs(output_dir, exist_ok=True)
    dot_path = os.path.join(output_dir, f"{output_name}.dot")
    img_path = os.path.join(output_dir, f"{output_name}.png")
    node_id = 0
    lines = [
        "digraph AST {",
        "node [shape=box, fontname=\"Helvetica\"];"
    ]

    def visit(node, parent=None):
        nonlocal node_id
        my_id = node_id
        label = node.data if hasattr(node, "data") else str(node)
        # Escape quotes for Graphviz
        label = label.replace('"', '\\"')
        lines.append(f'node{my_id} [label="{label}"];')
        if parent is not None:
            lines.append(f'node{parent} -> node{my_id};')
        node_id += 1
        if hasattr(node, "children"):
            for child in node.children:
                visit(child, my_id)
    visit(tree)
    lines.append("}")

    # Write DOT file
    with open(dot_path, "w", encoding="utf-8") as f:
        f.write("\n".join(lines))

    # Convert DOT to PNG
    subprocess.run(
        ["dot", "-Tpng", dot_path, "-o", img_path],
        check=True
    )
```

خلاصه عملکرد

- این تابع یک درخت AST تولیدشده توسط Lark را دریافت می‌کند.
- در ابتدای اجرا، پوشه‌ی خروجی (به‌طور پیش‌فرض outputs/) در صورت عدم وجود ساخته می‌شود.
- متغیر node_id به‌عنوان یک شمارنده‌ی سراسری برای اختصاص شناسه‌ی یکتا به هر گره استفاده می‌شود.
- تابع بازگشتی visit کل درخت را پیمایش کرده و:
  - برای هر گره یک نود Graphviz با قالب nodeX [label="..."] تولید می‌کند.
  - در صورت وجود گره‌ی والد، یال nodeParent -> nodeChild ایجاد می‌نماید.
  - تمامی فرزندان گره (از طریق node.children) را به‌صورت بازگشتی پردازش می‌کند.
- برای جلوگیری از خطا در Graphviz، کاراکترهای نقل‌قول (") در برچسب گره‌ها Escape می‌شوند.
- پس از تکمیل پیمایش، فایل DOT در مسیر مشخص‌شده ذخیره می‌شود.
- در مرحله‌ی نهایی، با استفاده از دستور dot از ابزار Graphviz، فایل DOT به تصویر PNG تبدیل شده و در همان پوشه ذخیره می‌گردد.

۶ کلاس CodeGenerator: تبدیل AST به کد پایتون

کلاس CodeGenerator وظیفه‌ی تبدیل درخت نحوی (AST) حاصل از پارس گرامر DSL به کد اجرایی پایتون را بر عهده دارد. این کلاس از Transformer در کتابخانه‌ی Lark ارث‌بری می‌کند و برای هر statement در گرامر، یک متد متناظر دارد.

۱.۶ پیکربندی اولیه و سیستم گزارش‌دهی

```
class CodeGenerator(Transformer):
    def __init__(self, output_dir):
        self.output_dir = output_dir
        self.plots_dir = os.path.join(output_dir, "plots")
        os.makedirs(self.plots_dir, exist_ok=True)

        self.current_var = None
        self.report_lines = []
        self.log_counter = 1
```

- output_dir: پوشه‌ی اصلی خروجی که شامل گزارش‌ها و فایل‌های تولیدی است.
- plots_dir: زیرپوشه‌ای برای ذخیره‌ی نمودارهای تولیدشده.
- current_var: آخرین دیتافریم فعال در فرآیند کامپایل.
- report_lines: لیستی از متن‌های گزارش که در نهایت در فایل report.txt ذخیره می‌شوند.
- log_counter: شماره‌گذاری ترتیبی عملیات‌ها.

۱.۱.۶ سیستم لاگ‌گیری

```
def add_log(self, title, body=""):
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    block = f"[{self.log_counter}] {title}\n{body}\n\nTimestamp: {ts}\n"
    self.report_lines.append(block)
    self.log_counter += 1
```

این متد بعد از هر عملیات:

- عنوان عملیات (مثلاً LOAD یا CLEAN)
- جزئیات آن عملیات
- زمان اجرا

را در گزارش نهایی ذخیره می‌کند.

۲.۶ مدیریت ساختار کلی AST

```
def statement(self, items):
    return items[0]

def start(self, items):
    return "\n".join(items)
```

- هر statement به یک قطعه کد پایتون تبدیل می‌شود.
- در متد start تمام دستورات به صورت متوالی کنار هم قرار می‌گیرند.

۳.۶ دستور :LOAD بارگذاری داده

```
def load_stmt(self, items):
    file_path, var = items
    self.current_var = var
```

در مرحله‌ی **compile-time** اگر فایل وجود داشته باشد، اطلاعات آماری آن استخراج می‌شود:

```
if os.path.exists(file_name):
    df_tmp = pd.read_csv(file_name)
    body += f"Rows: {len(df_tmp)}\nCols: {df_tmp.shape[1]}"
```

کد تولیدشده برای زمان اجرا:

```
if not os.path.exists(file_path):
    raise FileNotFoundError("File not found")

if file_path.endswith(".csv"):
    df = pd.read_csv(file_path)
elif file_path.endswith(".xlsx"):
    df = pd.read_excel(file_path)
```

۴.۶ دستورات اطلاعاتی: DESCRIBE و HEAD

```
print(df.describe())
print(df.head(n))
```

این دستورات فقط خروجی تحلیلی تولید می‌کنند و دیتافریم را تغییر نمی‌دهند.

۵.۶ دستور DUPLICATE و SAVE

```
dest = source.copy()
```

برای ذخیره:

```
df.to_csv(path, index=False)
df.to_excel(path, index=False)
df.to_json(path, orient="records")
```

۶.۶ دستور :CLEAN پاکسازی داده

۱.۶.۶ تشخیص نوع عملیات

```
op_type = actual_op.data.upper()
params = [str(c) for c in actual_op.children]
```

۲.۶.۶ حذف داده‌های پرت (IQR)

$$IQR = Q_3 - Q_1$$

$$x < Q_1 - 1.5 \times IQR \quad \text{یا} \quad x > Q_3 + 1.5 \times IQR$$

پیاده‌سازی:

```
q1, q3 = df[col].quantile([0.25, 0.75])
iqr = q3 - q1
df = df[(df[col] >= q1-1.5*iqr) & (df[col] <= q3+1.5*iqr)]
```

۳.۶.۶ جایگزینی مقادیر خالی

```
df[col] = df[col].fillna(df[col].mean())
df[col] = df[col].fillna(df[col].mode()[0])
```

۷.۶ دستورات :CALC محاسبات آماری

```
mean_col = df["col"].mean()
std_col = df["col"].std()
```

مقادیر محاسبه‌شده در محیط اجرا ذخیره شده و بعداً در گزارش استفاده می‌شوند.

۸.۶ دستورات :PLOT تولید نمودار

```
plt.figure(figsize=(10,6))
df["col"].hist(bins=20)
plt.savefig("plots/hist_col.png")
plt.close()
```

ویژگی‌ها:

- پشتیبانی از نمودارهای Histogram، Line، Box Scatter و Heatmap
- ذخیره خودکار در پوشه `plots`
- عدم استفاده از `plt.show()` برای جلوگیری از بلاک شدن برنامه

۹.۶ دستورات FILTER

```
df = df[df["age"] >= 18]
df = df.query("age > 18 & salary < 5000")
```

۱۰.۶ :LEVELING دسته‌بندی بازه‌ای

```
df["age_level"] = pd.cut(
    df["age"],
    bins=[18,30,60,inf],
    labels=["Teen","Adult","Senior"],
    right=False
)
```

۱۱.۶ دستورات CRUD روی ستون‌ها

- `pd.merge(df1, df2, on=key)` :MERGE
- `df.eval(expr)` :CREATE_COL
- DROP_COL: حذف ستون
- RENAME: تغییر نام ستون
- CONVERT_TIME: تبدیل به `datetime`

۱۲.۶ :NORMALIZE نرمال‌سازی Min-Max

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
df[col] = (df[col] - df[col].min()) / (df[col].max() - df[col].min())
```

۱۳.۶ CORRELATE: محاسبه هم‌بستگی

```
corr_val = df["col1"].corr(df["col2"])
```

این مقدار به صورت عددی در خروجی چاپ و در گزارش ثبت می‌شود.

۷ نحوه استفاده از CodeGenerator و تولید کد نهایی

پس از تعریف گرامر ،DSL نگاشت دستورات و ساخت ،AST کلاس `CodeGenerator` در مرحله‌ی نهایی کامپایل استفاده می‌شود تا درخت نحوی به کد اجرایی پایتون تبدیل گردد.

۱.۷ جریان کامل کامپایل

فرآیند کلی کامپایل به صورت زیر است:

۱. دریافت کد DSL (فارسی یا انگلیسی)
۲. تبدیل به DSL میانی (در صورت وجود Mapper)
۳. پارس کد با `Lark` و تولید AST
۴. عبور AST از `CodeGenerator`
۵. تولید کد پایتون قابل اجرا
۶. اجرای کد تولیدشده و تولید خروجی‌ها

۲.۷ نحوه استفاده از CodeGenerator

در تابع اصلی کامپایل (مثلاً `run_compiler`):

```
parser = Lark(grammar, parser="lalr")
ast_tree = parser.parse(dsl_code)

generator = CodeGenerator(OUTPUT_DIR)
python_code = generator.transform(ast_tree)
```

در این مرحله:

- `ast_tree`: درخت نحوی برنامه DSL
- `generator`: نمونه‌ای از کلاس `CodeGenerator`
- `transform`: متد اصلی `Transformer` که:
  - روی تمام نودهای AST پیمایش می‌کند
  - برای هر دستور DSL متد متناظر را صدا می‌زند
  - خروجی هر دستور را به صورت رشته کد پایتون جمع‌آوری می‌کند

۳.۷ خروجی متد transform

```
python_code = generator.transform(ast_tree)
```

نوع خروجی:

- یک `string` شامل کد کامل پایتون
- این کد شامل:
  - دستورات بارگذاری داده
  - پاکسازی
  - محاسبات آماری
  - فیلترها
  - تولید نمودار
  - عملیات CRUD

نکته مهم: کلاس `CodeGenerator` فایل تولید نمی‌کند، بلکه فقط کد را به صورت رشته بازمی‌گرداند.

۴.۷ تولید فایل generated_code.py

کد بازگشتی در فایل ذخیره می‌شود:

```
with open("generated_code.py", "w", encoding="utf-8") as f:
    f.write("import pandas as pd\n")
    f.write("import matplotlib.pyplot as plt\n")
    f.write("import seaborn as sns\n")
    f.write("import os\n\n")

    f.write(f'OUTPUT_DIR = r"{OUTPUT_DIR}"\n')
    f.write(f'PLOTS_DIR = "plots"\n')
    f.write('PLOT_PATH = os.path.join(OUTPUT_DIR, PLOTS_DIR)\n\n')

    f.write('os.makedirs(PLOT_PATH, exist_ok=True)\n\n')
    f.write(python_code)
```

در نتیجه فایل `generated_code.py`:

- کاملاً مستقل است
- بدون نیاز به کامپایلر DSL قابل اجراست
- تمام خروجی‌ها را در پوشه‌ی `output/` تولید می‌کند

۵.۷ اجرای کد تولیدشده

کد تولیدشده بلافاصله اجرا می‌شود:

```
env = {"pd": pd, "plt": plt, "os": os, "sns": sns}
exec(python_code, env)
```

- `env`: محیط اجرای کنترل‌شده
- تمام متغیرهای محاسبه‌شده (میانگین‌ها، انحراف معیارها و ...) در این دیکشنری ذخیره می‌شوند

۶.۷ تولید گزارش نهایی

پس از اجرای کد:

```
for line in generator.report_lines:
    f.write(line.format(**env))
```

- `report_lines`: خروجی **side-effect** کلاس `CodeGenerator`
- شامل:
  - لاگ تمام عملیات‌ها
  - نتایج محاسبات آماری
  - زمان اجرای هر دستور

۷.۷ خلاصه خروجی‌های CodeGenerator

مورد	نوع	توضیح
`transform()`	`string`	کد پایتون تولیدشده
`report_lines`	`list[string]`	گزارش عملیات‌ها
نمودارها	فایل	`.png` در پوشه plots
گزارش نهایی	فایل	`report.txt`
`generated_code.py`	فایل	اسکریپت مستقل پایتون

۸.۷ جمع‌بندی

کلاس `CodeGenerator` قلب کامپایلر DSL است که:

- AST را به کد اجرایی تبدیل می‌کند
- گزارش تحلیلی تولید می‌کند
- نمودارها و فایل‌های خروجی را مدیریت می‌کند
- خروجی نهایی قابل اجرا و مستقل تحویل می‌دهد

۸ پایپ‌لاین `run_compiler` و ارتباط با کد تولیدی

تابع `run_compiler` قلب اصلی کامپایلر DSL فارسی–انگلیسی است و کل فرآیند `Translation → Parsing → Code Generation → Execution → Reporting` را به‌صورت یکپارچه مدیریت می‌کند.

```
def run_compiler(persian_code, capture_output=True):
    import sys
    import io

    OUTPUT_DIR = "output"
    PLOTS_DIR = "plots"
    os.makedirs(OUTPUT_DIR, exist_ok=True)

    if capture_output:
        old_stdout = sys.stdout
        old_stderr = sys.stderr
        captured = io.StringIO()
        sys.stdout = sys.stderr = captured

    try:
        mapper = PersianToDSLMapper()
        dsl_code = mapper.translate(persian_code)

        parser = Lark(grammar, parser="lalr")
        ast_tree = parser.parse(dsl_code)

        ast_to_dot(ast_tree, output_name="ast", output_dir=OUTPUT_DIR)

        generator = CodeGenerator(OUTPUT_DIR)
        python_code = generator.transform(ast_tree)

        gen_path = os.path.join("./", "generated_code.py")
        with open(gen_path, "w", encoding="utf-8") as f:
            f.write("import pandas as pd\n"
                    "import matplotlib.pyplot as plt\n"
                    "import os\n"
                    "import seaborn as sns\n"
                    "import warnings\n"
                    "warnings.filterwarnings('ignore')\n\n")

            f.write(f'OUTPUT_DIR = r"{OUTPUT_DIR}"\n')
            f.write(f'PLOTS_DIR = r"{PLOTS_DIR}"\n')
            f.write(f'PLOT_PATH = os.path.join(OUTPUT_DIR, PLOTS_DIR)\n\n')

            f.write('os.makedirs(OUTPUT_DIR, exist_ok=True)\n')
            f.write('os.makedirs(PLOT_PATH, exist_ok=True)\n\n')

            f.write(python_code)

        env = {
            "pd": pd,
            "plt": plt,
            "os": os,
            "sns": sns,
```

```
            "OUTPUT_DIR": OUTPUT_DIR,
            "PLOTS_DIR": PLOTS_DIR,
            "PLOT_PATH": os.path.join(OUTPUT_DIR, PLOTS_DIR)
        }

        exec(python_code, env)

        report_path = os.path.join(OUTPUT_DIR, "report.txt")
        with open(report_path, "w", encoding="utf-8") as f:
            for line in generator.report_lines:
                try:
                    f.write(line.format(**env) + "\n\n")
                except Exception:
                    f.write(line + "\n\n")

        return (True, captured.getvalue() if capture_output else "", "")

    except Exception as e:
        import traceback
        return (False, captured.getvalue(), str(e))

    finally:
        if capture_output:
            sys.stdout = old_stdout
            sys.stderr = old_stderr
```

تحلیل گام‌به‌گام پایپ‌لاین

۱. **ایجاد پوشه‌های خروجی**
پوشه‌ی `output` و زیرپوشه‌ی `plots` ساخته می‌شوند تا تمام خروجی‌ها (کد، نمودار، گزارش و AST) در یک ساختار منظم ذخیره شوند.

۲. **ترجمه DSL فارسی به DSL میانی**
`PersianToDSLMapper.translate` با استفاده از `re.sub`:

- کلمات فارسی را به دستورات DSL انگلیسی تبدیل می‌کند
- خروجی آن یک رشته‌ی DSL استاندارد است

۳. **Parsing و تولید AST**
کد DSL توسط `Lark` تجزیه شده و یک درخت نحوی انتزاعی (`AST`) ساخته می‌شود که نمایش ساختاری برنامه است.

۴. **تولید فایل AST**
درخت AST به‌صورت گراف (dot/png) ذخیره می‌شود تا تحلیل ساختار برنامه ممکن باشد.

۵. **تولید کد پایتون**
کلاس `CodeGenerator`:

- AST را پیمایش می‌کند
- برای هر دستور DSL، کد پایتون معادل می‌سازد
- خروجی نهایی، رشته‌ی `python_code` است

۶. **نوشتن فایل `generated_code.py`**
این فایل شامل:

- import کتابخانه‌ها
- تعریف مسیرهای خروجی
- کد پایتون تولیدشده از AST

۷. **اجرای کد تولیدی**
کد با دستور `exec(python_code, env)` اجرا می‌شود. دیکشنری `env` نقش محیط اجرای ایمن را دارد و شامل:

- DataFrameها

- متغیرهای محاسبه‌شده
- مسیر نمودارها

۸. **تولید گزارش نهایی**
لیست `generator.report_lines` شامل قالب‌های متنی گزارش است. این خطوط با استفاده از:

```
line.format(**env)
```

به مقادیر واقعی محاسبه‌شده در زمان اجرا متصل می‌شوند.

۹ رابط کاربر (get_user_input) و تابع __main__

تابع `get_user_input` مسئول دریافت کد DSL از کاربر است و دو حالت اصلی برای ورودی ارائه می‌دهد:

- **ورودی دستی:** کاربر می‌تواند کد DSL را خط به خط در کنسول وارد کند. برای این منظور، تابع یک حلقه `while` باز می‌کند و تا زمانی که کاربر خط خالی وارد نکند، هر خط را به یک لیست اضافه می‌کند.
- **بارگذاری از فایل متنی:** کاربر می‌تواند مسیر یک فایل `.txt` را وارد کند. تابع بررسی می‌کند که فایل وجود داشته باشد و سپس محتوای آن را با `encoding="utf-8"` می‌خواند.

پس از دریافت کد، تابع آن را به صورت یک رشته واحد (`string`) بازمی‌گرداند.

در فایل اصلی، بخش زیر وظیفهٔ اجرای برنامه را بر عهده دارد:

```
if __name__ == "__main__":
    try:
        user_code = get_user_input()
        run_compiler(user_code)
    except Exception as e:
        print("Error:", e)
```

توضیح گام به گام:

۱. شرط `if __name__ == "__main__":` تضمین می‌کند که این بخش تنها زمانی اجرا شود که اسکریپت به صورت مستقیم اجرا شده باشد و نه وقتی که به عنوان ماژول وارد (`import`) شود.

۲. `get_user_input()` فراخوانی می‌شود تا کد DSL از کاربر دریافت گردد.

۳. سپس `run_compiler(user_code)` برای پردازش و اجرای کد وارد شده استفاده می‌شود.

۴. در صورت بروز هرگونه خطا، با بلوک `try-except` آن خطا گرفته شده و پیغام مناسب به کاربر نمایش داده می‌شود.

این ساختار ساده، کامپایلر را به یک ابزار `CLI` قدرتمند تبدیل می‌کند که هم انعطاف‌پذیری دریافت ورودی از کاربر را دارد و هم مدیریت خطاها را به شکل مناسبی انجام می‌دهد.

۱۰ توضیح سناریوی نمونه و نحوه اجرای آن

در این بخش، یک سناریوی کامل نوشته‌شده با زبان خاص‌منظوره فارسی (Persian DSL) معرفی و تحلیل می‌شود. این سناریو با هدف پردازش، تحلیل آماری و مصورسازی داده‌های آزمایشگاهی طراحی شده و توسط کامپایلر DSL فارسی پیاده‌سازی‌شده در این پروژه اجرا می‌گردد.

۱.۱۰ معرفی سناریو

کد زیر یک اسکریپت DSL فارسی است که عملیات مختلفی از جمله بارگذاری داده، پاکسازی، فیلتر، ایجاد ستون‌های محاسباتی، نرمال‌سازی، سطح‌بندی، رسم نمودار و ذخیره خروجی را انجام می‌دهد:

۲.۱۰ جریان اجرای سناریو در کامپایلر

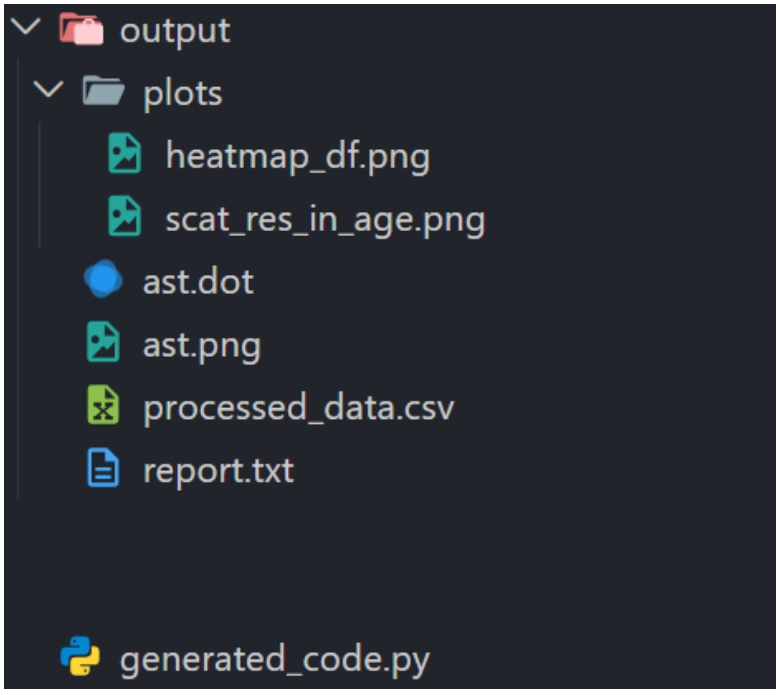
در ابتدا برنامه را اجرا میکنیم :

```
PS S:\FivethSemester\Compiler\lastProject> python LabDsl.py
=======================================
 Persian / English DSL Compiler
=======================================
HINT: You may mix Persian and English DSL commands in the same script.
---------------------------------------

Select input method:
 1) Enter code manually
 2) Load code from text file
Your choice (1/2): 2

```

سپس اجرا به شکل خواندن دستورات از روی فایل رو اجرا میکنیم :

```
Your choice (1/2): 2
Enter .txt file path: path.txt

```

نگاهی به فایل های ایجاد شده پس از اجرای دستورات :



فایل دستورات با نام path.txt حاوی دستور های زیر است به ترتیب :

- بگیر از "lab_data.csv" به نام df
- پاکسازی df : حذف تکراری
- پاکسازی df : حذف کلی خالی
- فیلتر df : وقتی age < ۱۵
- ایجاد_ستون df : result = ((glucose * ۱.۰) + (cholesterol * ۹.۰))
- تغییر_نام df : result به sick
- نرمال_سازی df : ستون sick
- نمودار df : نقشه_حرارتی
- مرتب_سازی df : sick صعودی

- ایجاد_ستون df : sick = res
- سطح_بندی df : در res به low:۰ normal:۳.۰ heigh:۶.۰ danger:۸.۰
- نمودار df : پراکندگی res در age
- ذخیره df : در "processed_data.csv"

اجرای این سناریو در کامپایلر DSL فارسی شامل مراحل زیر است:

۱. **نگاشت دستورات فارسی به DSL میانی** در این مرحله، کلاس PersianToDSLMapper با استفاده از عبارات باقاعده (Regex)، دستورات فارسی را به معادل انگلیسی DSL تبدیل می‌کند (مانند LOAD, CLEAN, FILTER, PLOT).

۲. **تحلیل نحوی (Parsing)** کد DSL میانی با استفاده از گرامر تعریف‌شده در Lark تحلیل شده و یک درخت نحو انتزاعی (AST) تولید می‌شود.

۳. **تولید AST و مصورسازی آن** AST تولیدشده به کمک تابع ast_to_dot به فایل ast.dot تبدیل می‌شود که امکان مشاهده ساختار برنامه را فراهم می‌کند.

۴. **تولید کد پایتون** کلاس CodeGenerator که از Transformer ارث‌بری می‌کند، AST را پیمایش کرده و کد معادل پایتون را تولید می‌نماید.

۵. **اجرای کد و تولید خروجی‌ها** کد پایتون تولیدشده به‌صورت پویا اجرا شده و فایل‌ها، نمودارها و گزارش آماری ایجاد می‌شوند.

—

۳.۱۰ توضیح گام‌به‌گام عملیات سناریو

- ابتدا فایل lab_data.csv بارگذاری شده و در DataFrame با نام df ذخیره می‌شود.
- سطرهای تکراری حذف می‌شوند تا داده‌ها یکتا گردند.
- تمام سطرهایی که شامل مقادیر خالی هستند حذف می‌شوند.
- فقط رکوردهایی که مقدار ستون age آن‌ها بزرگ‌تر از ۱۵ است نگه داشته می‌شوند.
- یک ستون محاسباتی جدید با نام result بر اساس ترکیب وزنی glucose و cholesterol ایجاد می‌شود.
- نام ستون result به sick تغییر می‌یابد.
- ستون sick با استفاده از روش Min–Max نرمال‌سازی می‌شود.
- یک نقشه حرارتی (Heatmap) از همبستگی ویژگی‌های عددی رسم می‌گردد.
- داده‌ها بر اساس ستون sick به‌صورت صعودی مرتب می‌شوند.
- ستون جدیدی به نام res به‌عنوان کپی از sick ایجاد می‌شود.
- ستون res به سطوح معنایی low, normal, heigh, danger تقسیم‌بندی می‌گردد.
- نمودار پراکندگی بین res و age رسم می‌شود.
- در نهایت، داده‌های پردازش‌شده در فایل processed_data.csv ذخیره می‌شوند.

۴.۱۰ تحلیل نتایج اجرا بر اساس داده‌ی ورودی نمونه

۱.۴.۱۰ بررسی داده‌ی ورودی

داده‌ی اولیه از فایل lab_data.csv بارگذاری شده است. این فایل شامل ۱۰ رکورد و ۵ ستون به شرح زیر است:

- id : شناسه بیمار
- gender : جنسیت
- age : سن
- glucose : میزان قند خون
- cholesterol : میزان کلسترول

بررسی آماری اولیه نشان می‌دهد که:

- ستون‌های `glucose` و `cholesterol` شامل مقادیر خالی هستند.
- حداقل سن برابر ۱۰ بوده که در مراحل بعدی فیلتر می‌شود.
- همبستگی اولیه بین قند خون و کلسترول نسبتاً مثبت است.

۲.۴.۱۰ تأثیر پاکسازی و فیلتر داده‌ها

در مراحل اولیه پردازش، عملیات زیر انجام شده است:

- حذف سطرهای تکراری
- حذف کامل سطرهای دارای مقدار خالی
- فیلتر داده‌ها بر اساس شرط `age > 15`

پس از اعمال این مراحل:

- تعداد رکوردها از ۱۰ به ۶ کاهش یافته است.
- تمامی ستون‌های عددی بدون مقدار خالی هستند.
- داده‌ها برای انجام تحلیل‌های آماری آماده شده‌اند.

۳.۴.۱۰ محاسبه شاخص سلامت و نرمال‌سازی

در این سناریو، یک ستون محاسباتی جدید با رابطه زیر ایجاد شده است:

$$\text{sick} = 0.1 \times \text{glucose} + 0.9 \times \text{cholesterol}$$

این شاخص، وزن بیشتری به کلسترول داده و نمایانگر یک معیار ترکیبی از وضعیت سلامت بیمار است.
سپس ستون `sick` با استفاده از روش Min–Max نرمال‌سازی شده است، به‌طوری که مقادیر آن در بازه $[0, 1]$ قرار می‌گیرند.
نتیجه نهایی نشان می‌دهد:

- کمترین مقدار `sick` برابر ۰
- بیشترین مقدار برابر ۱
- میانگین شاخص سلامت برابر ۳۹۲.۰

۴.۴.۱۰ سطح‌بندی معنایی بیماران

برای تفسیر ساده‌تر شاخص سلامت، ستون `res` به سطوح معنایی زیر تقسیم شده است:

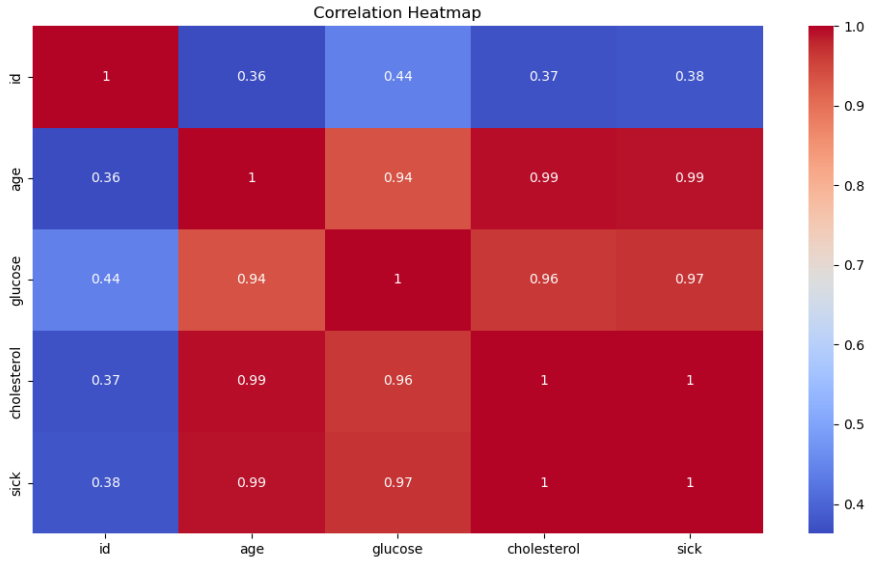
سطح سلامت	بازه مقدار
low	$[0, 0.3)$
normal	$[0.3, 0.6)$
heigh	$[0.6, 0.8)$
danger	$[0.8, 1.0]$

بر اساس این تقسیم‌بندی:

- بیماران جوان‌تر عمدتاً در سطح `low` قرار دارند.
- بیماران با سن بالاتر و کلسترول بیشتر در سطح `danger` مشاهده می‌شوند.

۵.۴.۱۰ تحلیل نقشه حرارتی همبستگی

شکل ۲ نقشه حرارتی همبستگی بین ویژگی‌های عددی را نمایش می‌دهد.

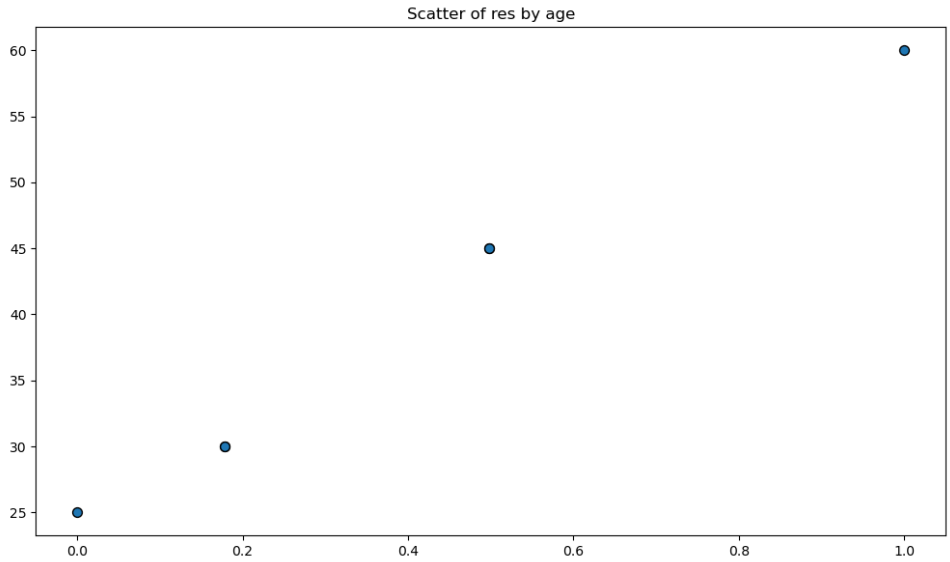


شکل ۲: نقشه حرارتی همبستگی ویژگی‌های عددی

این نمودار نشان می‌دهد که بین `glucose` و `cholesterol` همبستگی مثبتی وجود دارد که توجیه‌کننده استفاده از آن‌ها در شاخص سلامت است.

۶.۴.۱۰ نمودار پراکندگی سن و شاخص سلامت

نمودار پراکندگی شکل ۳ رابطه بین سن بیماران و شاخص سلامت نهایی را نمایش می‌دهد.



شکل ۳: نمودار پراکندگی سن و شاخص سلامت

بر اساس این نمودار، با افزایش سن، تمایل به افزایش شاخص بیماری مشاهده می‌شود، هرچند رابطه کاملاً خطی نیست.

۵.۱۰ فایل ریپورت

یکی از خروجی های ما فایل ریپورت است که در آن تمام کارهایی که انجام شده به ترتیب به سکل لاگ موجود است :

```
Laboratory Data Statistical Report
========================================

[1] LOAD
Loaded file: lab_data.csv
Rows: 10
Cols: 5

Timestamp: 2026-01-31 23:22:39


[2] CLEAN - DROP_DUPLICATES
Params:

Timestamp: 2026-01-31 23:22:39


[3] CLEAN - DROP_ALL
Params:

Timestamp: 2026-01-31 23:22:39


[4] FILTER
Condition: age > 15

Timestamp: 2026-01-31 23:22:39


[5] CREATE_COL
Created column: result in df

Timestamp: 2026-01-31 23:22:39


[6] RENAME
Renamed column result to sick in df

Timestamp: 2026-01-31 23:22:39


[7] NORMALIZE
Normalized column: sick in df

Timestamp: 2026-01-31 23:22:39


[8] PLOT
Type: HEATMAP
Target: ALL
Group: ALL

Timestamp: 2026-01-31 23:22:39


[9] SORT
Sort by sick   ))

Timestamp: 2026-01-31 23:22:39


[10] CREATE_COL
Created column: res in df

Timestamp: 2026-01-31 23:22:39
```

```
[11] LEVELING
Column: res
Levels: ['low', 'normal', 'heigh', 'danger']

Timestamp: 2026-01-31 23:22:39


[12] PLOT
Type: SCAT
Target: res
Group: age

Timestamp: 2026-01-31 23:22:39


[13] SAVE
Saved df to "processed_data.csv"

Timestamp: 2026-01-31 23:22:39

```

۶.۱۰ کد تولید شده

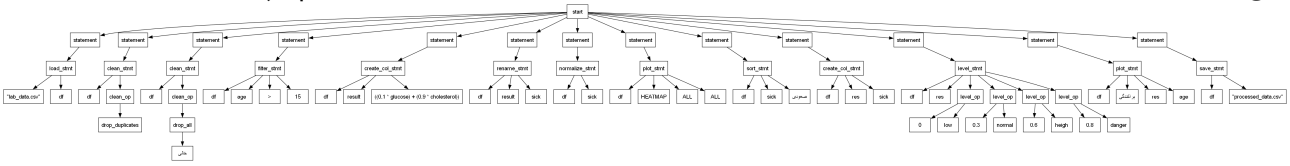
در این قسمت هم کد تولید شده که مد به زبان سطح پایین است آورده شده و این فایل با هنوان generated_code در فایل output موجود است :

```python
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns


# --- Load Data ---
if not os.path.exists("lab_data.csv"):
    raise FileNotFoundError("File not found")

if "lab_data.csv".endswith(".csv"):
    df = pd.read_csv("lab_data.csv")
elif "lab_data.csv".endswith(".xlsx"):
    df = pd.read_excel("lab_data.csv")
else:
    raise ValueError("Only CSV or Excel files are supported")


# --- Cleaning: DROP_DUPLICATES ---
df = df.drop_duplicates()

# --- Cleaning: DROP_ALL ---
df = df.dropna()
df = df[df["age"] > 15]
df['result'] = df.eval('''((0.1 * glucose) + (0.9 * cholesterol))''')
df = df.rename(columns={"result": "sick"})

# --- Min-Max Normalization ---
col_min = df["sick"].min()
col_max = df["sick"].max()
df["sick"] = (df["sick"] - col_min) / (col_max - col_min)


# --- Plotting HEATMAP ---
plt.figure(figsize=(10, 6))
corr = df.select_dtypes(include=["number"]).corr()
sns.heatmap(corr, annot=True, cmap="coolwarm")
```

```python
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.savefig(r"output\plots\heatmap_df.png")
plt.show()
plt.close()

df = df.sort_values(by="sick", ascending=True)
df['res'] = df.eval('''sick''')

# --- Leveling Of res From df ---
bins = [0.0, 0.3, 0.6, 0.8]
bins.append(float('inf'))
labels = ['low', 'normal', 'heigh', 'danger']
df["res_level"] = pd.cut(df["res"], bins=bins, labels=labels, right=False)


# --- Plotting SCAT ---
plt.figure(figsize=(10, 6))
plt.scatter(df["res"], df["age"], edgecolors="black", s=50)
plt.title("Scatter of res by age")
plt.tight_layout()
plt.savefig(r"output\plots\scat_res_in_age.png")
plt.show()
plt.close()


    # --- Save DataFrame ---
if "processed_data.csv".endswith(".csv"):
    df.to_csv(r"processed_data.csv", index=False)
elif "processed_data.csv".endswith(".xlsx"):
    df.to_excel(r"processed_data.csv", index=False)
elif "processed_data.csv".endswith(".json"):
    df.to_json(r"processed_data.csv", orient="records")
else:
    raise ValueError("Supported formats: .csv, .xlsx, .json")


```

۷.۱۰ تصویر ast نهایی از دستورات

تابع ast_to_dot یک نمونه تصویر از این ast خواهد ساخت که در کنار فایل ast.dot با همین نام و پسوند png. موجود است :



۸.۱۰ دیتاست نهایی ذخیره شده

دیتاست پروسس شده جدید به اضافه مقادیر محاسبه شده در فایل output به نام process_data.csv ذخیره خواهد شد.
قبل پروسس در فایل lab_data :

lab_data.csv

	id	gender	age	glucose	cholesterol
1	1	Male	25	92	180
2	2	Female	30	105	190
3	3	Male	45	110	210
4	4	Female	50	98	
5	5	Male	10	115	220
6	6	Female	40		195
7	7	Male	60	130	240
8	8	Female	55	120	
9	9	Male	45	110	210
10	10	Female	30	105	190
11					

بعد از پروسس در فایل processed_data :

output > processed_data.csv

	id	gender	age	glucose	cholesterol	sick	res	res_level
1	1	Male	25	92.0	180.0	0.0	0.0	low
2	2	Female	30	105.0	190.0	0.17820069204152264	0.17820069204152264	low
3	10	Female	30	105.0	190.0	0.17820069204152264	0.17820069204152264	low
4	3	Male	45	110.0	210.0	0.4982698961937717	0.4982698961937717	normal
5	9	Male	45	110.0	210.0	0.4982698961937717	0.4982698961937717	normal
6	7	Male	60	130.0	240.0	1.0	1.0	danger
7								

۹.۱۰ جمع‌بندی نهایی سناریو

نتایج این سناریو نشان می‌دهد که DSL فارسی طراحی‌شده قادر است:

- داده‌های واقعی دارای نقص را پاکسازی کند
- شاخص‌های سلامت قابل تفسیر ایجاد نماید
- تحلیل‌های آماری و تصویری معنادار تولید کند
- خروجی‌های استاندارد قابل استفاده در تحلیل‌های بعدی ارائه دهد

این سطح از خودکارسازی، DSL را به ابزاری مناسب برای کاربران غیرمتخصص حوزه داده و علوم آزمایشگاهی تبدیل می‌کند.