Task 2.7

Data Analysis

Task-2.7 document (copy-paste into your “Task-2.7”)

Exercise 2.7: Data Analysis and Visualization in Django

Learning goals

- Work on two-way communication (forms and buttons)
- Implement search and visualization (reports/charts)
- Use QuerySet API, pandas DataFrames, and matplotlib charts

1) Search plan (what users can search + output format)

- Criteria:
  - Recipe name (supports partial/wildcard search; case-insensitive)
  - Difficulty filter (All, Easy, Medium, Intermediate, Hard)
  - Chart type (Bar, Pie, Line)

- Output:
  - Results table (columns: name, cooking_time, difficulty, ingredients_count, category)
  - Clickable recipe names leading to the detail page
  - Chart below the table matching selected chart type

2) Data analysis plan (charts shortlist)

For each, define axes and labels:

- Bar chart: Cooking Time by Recipe
  - X: Recipe Name
  - Y: Cooking Time (minutes)
  - Labels: Recipe names on x-axis; title “Cooking Time by Recipe”

- Pie chart: Recipe Distribution by Difficulty
  - Slices: Count per difficulty bucket

- Labels: Difficulty levels
- Title: “Recipe Distribution by Difficulty”
- Line chart: Cooking Time Trend
- X: Recipe Name (ordered by name or filtered order)
- Y: Cooking Time (minutes)
- Labels: “Cooking Time Trend”; grid enabled for readability

Whether charts are user-driven or fixed:

- The chart displayed is based on user input (chart type selector).

3) Execution flow (user journey)

ASCII flow chart

Home → Login → Recipes List → Search (optional filters) → View Results Table → View Chart → Click a Recipe → Detail Page → Logout

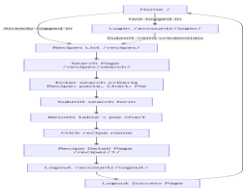
Detailed flow (ASCII):

- Start
- Open homepage (/)
- If not logged in:
  - Click “Log In” → /login/
  - Enter credentials → submit → redirect to /recipes/

- On /recipes/:
- Optionally enter Recipe Name (can be partial)
- Optionally choose Difficulty (or leave as All)
- Choose Chart Type (Bar/Pie/Line)
- Click Search → results reload on /recipes/
- See:
- Results table (clickable names)
- Chart for selected type
- Click a recipe → /recipes/<id> (detail)

- Click Logout → /logout/ → success page

- End



4) Run server and capture output (Windows PowerShell)

Note: Use your actual project paths and venv name. Based on your setup:

- Project: C:\Users\dasau\recipe-app\recipe_project
- Virtual environment: web-dev

Commands:
powershell

Navigate to the Django project

cd C:\Users\dasau\recipe-app\recipe_project

Activate your virtual environment (adjust the path if needed)

& "$Env:USERPROFILE\Envs\web-dev\Scripts\Activate.ps1"

Run server

py manage.py runserver

If activation path differs, you can locate it and run: .\Scripts\Activate.ps1 inside that venv directory.

5) User journey steps to screenshot (or screencast)

Take a screenshot at each step and paste below each bullet. Alternatively, record a 2–3 minute screencast (user-journey.mp4).

- Step 1: Homepage
  - URL: http://127.0.0.1:8000/
  - Expect: Welcome page with link to Log In

- Step 2: Login
  - URL: http://127.0.0.1:8000/login/
  - Action: Enter valid credentials → submit

- Step 3: Recipes list (protected)
  - URL: http://127.0.0.1:8000/recipes/
  - Expect: Search form (Recipe Name, Difficulty, Chart Type), All Recipes table

- Step 4: Search submission
  - URL: http://127.0.0.1:8000/recipes/ (POST submit, page reloads)
  - Expect: Results table showing filtered recipes, chart under the table
  - Try:
    - Partial search (e.g., “Spa” for “Spaghetti”)
    - Difficulty = Easy
    - Chart type = Bar, then Pie, then Line

- Step 5: Click a recipe
  - URL: http://127.0.0.1:8000/recipes/<id>/
  - Expect: Recipe detail with fields and computed difficulty

- Step 6: Logout
  - URL: http://127.0.0.1:8000/logout/
  - Expect: Success/confirmation page

Optional: Also capture an unauthenticated access to /recipes/ to show redirect to /login/.

6) Tests (screenshot for test-outcome.jpg)

Run the test suite with verbosity 2 and capture the terminal output:
powershell
cd C:\Users\dasau\recipe-app\recipe_project
& "$Env:USERPROFILE\Envs\web-dev\Scripts\Activate.ps1"
py manage.py test -v 2

Save the screenshot as test-outcome.jpg and add it to your Exercise 2.7 “screenshots” folder on GitHub.

7) Upload to GitHub (structure)

Achievement 2/

Exercise 2.7/

Task-2.7.md (this document)

- screenshots/
- test-outcome.jpg
  - step-1-home.jpg
  - step-2-login.jpg
  - step-3-recipes.jpg
  - step-4-results-bar.jpg
  - step-4-results-pie.jpg
  - step-4-results-line.jpg
  - step-5-detail.jpg
  - step-6-logout.jpg
  - user-journey.mp4 (optional)

Answers to Exercise 2.7 Reflection Questions

1.Favorite website/app data and how analysis helps

Example: CareerFoundry (or pick your favorite app with similar patterns)

- Collected data:
  - User sign-ups, course enrollments, lesson completion timestamps
  - Session duration, time-on-task, drop-off points
  - Quiz scores, project submissions, mentor feedback timestamps
  - Support ticket categories and resolution times

- Engagement events (clicks on resources, video play/pause, forum activity)

- How analysis helps:
- Learning effectiveness: Correlate time-on-task and quiz scores to identify which lessons need improvement.
- Personalization: Recommend content based on prior lessons where learners struggled or took longer.
- Retention: Detect at-risk learners (reduced engagement, missed deadlines) and intervene early.

- Operations: Optimize mentor staffing by forecasting peak hours based on historical session patterns.
- Product decisions: A/B test lesson formats (text vs video) and improve the ones with lower completion rates.
- Support: Identify recurring issues (e.g., environment setup) and create proactive guides or onboarding improvements.

2. Ways to evaluate a QuerySet (Django docs summary)

A QuerySet is lazy until evaluated. Common evaluation triggers:

- Iteration: for obj in qs (loads results)
- Slicing that isn't trivial: qs[0:10] (executes a limited query)
- Conversions:
- list(qs)
- bool(qs) or if qs: (executes SELECT EXISTS)
- len(qs) (executes SELECT COUNT or fetches and counts)
- Aggregations: qs.count(), qs.exists(), qs.aggregate(), qs.annotate()
- First/Last/Get:
- qs.first(), qs.last() (executes with ORDER BY/LIMIT)
- qs.get(...) (executes single-row SELECT, raises DoesNotExist/MultipleObjectsReturned)
- Values:
- qs.values(), qs.values_list() (executes immediately and returns dictionaries/tuples)
- Other:
- Serialization, caching into a variable in a context that forces data reading

- Printing the queryset in the shell often triggers evaluation (depending on repr behavior)

3. QuerySet vs DataFrame: pros, cons, and when DataFrame is better

- QuerySet (Django ORM)
  - Advantages:
    - Database-backed, lazy, optimized via SQL
    - Safe, composable filters; leverages indexes; avoids loading unnecessary rows
    - Transactions and integrity via models; relationships via foreign keys
    - Great for CRUD, filtering, pagination, and simple aggregations (annotate/aggregate)
  - Disadvantages:
    - Not designed for heavy in-memory analytics across large datasets
    - Limited vectorized operations and complex reshaping/pivoting compared to pandas
    - Cross-database analytics or multi-step transformations are awkward
- DataFrame (pandas)
  - Advantages:
    - Vectorized operations, fast columnar computations in memory
    - Rich analytics API: groupby, pivot, rolling windows, joins/merges, resampling
    - Easy export (CSV/Excel/HTML), rich plotting integration
  - Disadvantages:
    - Data must be loaded into memory; large datasets can be slow or memory-hungry
    - Loses ORM-level guarantees (validation, transactions)
    - Requires explicit conversion from QuerySet to DataFrame
- When DataFrame is better:
  - Complex analytics: groupby aggregations, pivot tables, time-series analysis, feature engineering
  - Rapid experimentation with transformations and visualizations
  - Preparing data for charts, exports, or reports where columnar operations dominate
- Practical workflow:
  - Use QuerySet to efficiently pull only needed rows/columns from the DB
  - Convert to DataFrame for heavy analytics and visualization steps
  - Keep expensive operations in DataFrame space; keep data volume manageable