Summer Internship Report

Name: Jishnu Teja Dandamudi

Internship Duration: 45 Days

Organization/Institute: NetElixir

Project Title: SEM Campaign QA Checklists 2023 - Internal API Analysis

Mentor/Supervisor: Harshitha Yeddala

Agenda: SEM Campaign QA via Internal API Analysis

1. Understand Campaign Goals and Conversion Objectives
    - Identify and validate the different types of campaign objectives.
    - Ensure proper tracking and uniqueness of primary conversion actions.
2. Evaluate Campaign Structure and Naming Conventions
    - Check naming consistency across campaigns.
    - Verify structural alignment with naming standards (e.g., prefix NX_).
3. Assess Campaign and Adgroup Performance
    - Review keyword distribution across ad groups.
    - Identify campaigns/ad groups exceeding optimal thresholds (e.g., >20 keywords).
4. API-Based Validation of Metrics and Dimensions
    - Use Google Ads APIs (e.g., Campaign, Conversion, Adgroup APIs) to extract real-time campaign data.
    - Match retrieved metrics/dimensions against checklist expectations.
5. Automate QA Checklist Execution
    - Link internal data points with pass/fail criteria.

- Document and automate checks for conversion presence, naming, budget caps, etc.

6. Generate Insights for Optimization
    - Highlight non-compliant campaigns/ad groups.
    - Suggest actionable changes to align with best practices.
7. Document Findings and Report Compliance
    - Maintain a record of campaigns reviewed, API capabilities used, and QA completion status.
    - Prepare a consolidated report for stakeholders.

What I Had Learnt

This internship presented an intensive, hands-on opportunity to independently build a fully functional web-based QA automation platform using Django, despite having no prior experience with the framework at the outset. Over 45 days, I not only learned Django from scratch but also designed, implemented, and deployed a robust backend system that automates SEM campaign quality analysis using real-time and structured data inputs. Key learnings and technical proficiencies gained are as follows:

1. Full-Stack Web Application Development using Django (From Scratch)

- Gained a comprehensive understanding of the Model-View-Template (MVT) architecture and Django's core components including URL routing, views, templates, and middleware.
- Implemented file upload handling with FileSystemStorage, routing POST requests securely, and managing user inputs via HTML forms rendered through home.html.
- Developed custom backend logic in views.py that dynamically responds to user-uploaded Excel files, processes them in-memory, and returns intelligent insights.

2. Data Engineering with Pandas for Multi-Sheet Excel Analysis

- Used pandas to read and parse Excel files containing multiple worksheets, enabling structured campaign data ingestion.
- Applied complex dataframe transformations, filtering, grouping, and aggregation logic for rule-based analysis across ad group, campaign, and keyword dimensions.

3. Intelligent Data Validation and Business Logic Implementation

- Designed and coded over 20 automated QA checks, including:
    - Validation of primary conversion setup for “Purchase” objectives.
    - Analysis of ad group keyword volume thresholds.
    - Consistency enforcement of naming conventions (e.g., "NX_").
    - Audits for impression share loss due to budget, RSA ad strength, broken landing pages, and legacy keyword usage.
- Built logic to dynamically match data context to the correct validation rule using semantic matching and keyword detection.

4. API and Web Resource Integration

- Connected to external APIs (e.g., Google Ads API links and documentation references) for verifying campaign and conversion setup.
- Integrated real-time link testing using the requests module to detect broken landing pages and HTTP errors such as 404 and 405, with intelligent fallbacks to GET when HEAD is restricted.

5. Fuzzy Logic and Adaptive Matching

- Implemented fuzzy string matching using the fuzzywuzzy library to robustly associate user-supplied or predefined checklist questions with corresponding backend logic.
- Ensured graceful handling of input variations, making the system adaptive and resilient to natural language ambiguity.

6. Data Visualization and Result Reporting

- Used matplotlib to dynamically generate visual summaries (e.g., pie charts for keyword distribution) embedded directly in the result set.
- Converted HTML-based tabular summaries to clean Excel sheets using pandas.read_html() and openpyxl, allowing structured downloads with automated naming.

7. HTML & Data Sanitization

- Applied BeautifulSoup for HTML-to-text conversion and report cleaning, enabling readable and clean summaries in the Excel output and in-browser result display.

8. Robust Error Handling and Scalability Practices

- Developed extensive exception handling layers across the pipeline to manage malformed input, missing data, and API timeouts without disrupting user experience.
- Structured the codebase for modular reusability, ensuring each validation module could be expanded or edited independently.

9. Real-World Exposure to Project Lifecycle and Team Practices

- Followed a professional approach to iterative development, versioning, testing, and deployment.
- Documented key functions, maintained logical consistency, and ensured production-level readability and extensibility.

Project Objectives

1. To design and develop a full-stack Django-based QA automation system for SEM campaign audits

The project aimed to build a robust and user-friendly web application from the ground up using the Django framework. This system allows users to upload structured Excel files containing Google Ads campaign data and automatically evaluates them against a predefined checklist of 21 critical QA questions (as provided in questions.txt). The application dynamically processes multi-sheet inputs, executes validation logic in the backend, and displays clear results in a formatted HTML view with an option to download all findings in a consolidated Excel report.

2. To apply advanced rule-based logic, dataframe analytics, and fuzzy matching techniques for campaign verification

At the heart of the application lies a rule engine that maps each of the 21 business QA questions to custom logic implementations. These include checks for:

- Conversion action tracking integrity.
- Campaign naming conventions.
- Keyword volume thresholds.
- Landing page health and Final URL relevance.
- Ad format compliance (e.g., RSA utilization, legacy ETA detection).
- Budget-based impression share loss.

- Audience signals in Performance Max and Observation settings.

Additionally, the system incorporates fuzzy string matching (using fuzzywuzzy**)** to ensure that variations or ambiguities in checklist phrasing do not prevent logic from executing — making it resilient and adaptable to changes in wording or input formats.

3. To evaluate campaign structure and performance issues using programmatic validation and visual feedback mechanisms

The application analyses and validates large volumes of Excel data using Pandas, performs aggregations and filtering based on campaign/adgroup/keyword metadata, and computes KPIs to identify problematic trends. For select checks (e.g., keyword distribution), visualizations like pie charts are automatically generated using matplotlib and embedded in the HTML output to enhance interpretability. Tabular summaries of findings (e.g., broken URLs, underperforming adgroups, missing audience settings) are also rendered for clarity and exported into a structured Excel workbook using openpyxl.

4. To ensure scalability, reusability, and maintainability of the QA framework for long-term use

From the beginning, the application was designed with modularity and future extensibility in mind. Each validation logic function is encapsulated, making it easy to update or add new QA rules without altering the core system. Error handling was extensively implemented to ensure the system remains stable across a wide range of Excel file variations and schema inconsistencies. The solution supports multi-sheet parsing, enabling a single uploaded file to be comprehensively audited without manual segmentation.

Furthermore, the reporting output is both user-facing (via HTML views) and archival (via downloadable Excel files), ensuring usability for real-time review as well as offline documentation.

Implementation Steps

The implementation of this project followed a deeply iterative and self-driven approach. As a newcomer to the Django framework and enterprise-scale SEM data analysis, each step involved a combination of learning, experimentation, debugging, and architectural decision-making. Below is an expanded breakdown of the full development cycle:

1. Learning Django Framework from the Ground Up

- With zero prior exposure to Django, the first phase involved:

    - Setting up a Python virtual environment and installing Django.
    - Studying the Model-View-Template (MVT) architecture and Django's design philosophy through official documentation and tutorials.
    - Creating a new Django project with an app named main to encapsulate logic.
- Explored and practiced:
    - URL routing via urls.py.
    - View logic via views.py.
    - Static file management and template rendering via the templates/ and static/ directories.
    - Form handling using both GET and POST requests to render HTML responses dynamically.

2. Initial Experiment: Using MySQL for Question Storage (Abandoned Design)

- The original idea was to store all 21 checklist questions in a MySQL database and access them using Django ORM models.
    - Created a Django model representing a Question with fields like text, category, and weightage.
    - Migrated schema to a local MySQL database and attempted to populate it with values from questions.txt.
- However, during backend integration:
    - Realized that the questions were static and didn't require dynamic CRUD operations.
    - Database overhead introduced unnecessary complexity and potential data sync issues.
- After evaluating pros and cons, decided to discard the database-driven design in favour of a lighter, file-based workflow.

3. Switching to File-Based Question Loading via Pandas

- Created a utility function (load_predefined_questions()) to read questions from a simple .txt file.
    - Used native Python file I/O and list comprehension to parse the contents of questions.txt.
    - Cleaned the questions and stored them in a list, which was looped over for validation purposes.
- This method proved to be far more efficient, flexible, and version-controlled, since the question list was unlikely to change often.

4. Designing the Rule Engine & Validation Dispatcher

- Built a custom function run_all_checks(sheet_dict) to:
    - Iterate over each question.
    - Loop through all sheets in the uploaded Excel file.
    - Call run_analysis(matched_question, df) to determine if a match exists for each rule.
- Implemented fuzzy logic using fuzzywuzzy to allow imperfect matches between question text and logic identifiers.
    - Example: "Are there legacy BMM keywords?" would still map to the corresponding code block even if phrased differently.
    - This gave the system flexibility and robustness against wording variations.

5. Developing 20+ QA Validation Rules

Each rule is written as an if-elif block within run_analysis(). Key highlights include:

- **Conversion Checks:** Verified that only one primary conversion is set and that “Purchase” conversions are actively capturing data.
- **Naming Convention Analysis:** Checked if campaign names begin with standard prefixes like "NX_".
- **Volume & Budget Evaluations:** Identified ad groups exceeding 20 keywords, or campaigns losing impression share due to budget limitations.

- **Ad Format Validations:** Detected legacy ad formats (ETAs, BMM), verified RSA coverage, and ensured optimal use of headlines and descriptions.
- **URL Checks:** Implemented HTTP header and fallback GET requests to detect 404 errors and validate final URLs.
- **Audience & Targeting Checks:** Checked Performance Max campaign configurations for audience signals and ensured observation mode was applied at the campaign level.

6. File Upload Handling and Sheet Parsing

- Used Django's FileSystemStorage to temporarily save the uploaded Excel file.
- Loaded the file using pandas.ExcelFile and parsed all sheets into a dictionary ({sheet_name: dataframe}).
    - Standardized column headers with .str.strip() to remove whitespace inconsistencies.

7. Frontend Integration via Django Templating

- Created home.html using Django template syntax ({% for %}) to:
    - Allow users to upload files via an HTML form.
    - Display success/failure messages and render result tables dynamically.
    - Embed charts and messages returned from run_analysis().
- Ensured that each question's result was styled with success/failure icons, summaries, and expandable HTML tables if needed.

8. Visual Reporting with Matplotlib

- For certain rules (e.g., keyword volume thresholds), used matplotlib to:
    - Generate pie charts representing compliant vs. non-compliant ad groups.
    - Save each chart as a uniquely named PNG in the /static/ folder using uuid to prevent conflicts.
    - Display the image inline in the HTML using a relative <img> path.

9. HTML Sanitization and Summary Extraction

- For output that involved embedded tables, used BeautifulSoup to:
    - Strip HTML and render plain-text summaries in the Excel export.
    - Maintain formatting consistency across both browser and downloadable outputs.

10. Excel Report Generation with OpenPyXL

- Implemented save_results_to_excel() to:
    - Loop over all question results.
    - For results containing HTML tables, extracted data via pandas.read_html() and exported each to a named Excel sheet (Q01, Q02, etc.).
    - Constructed a Summary sheet with clean text summaries of each question's outcome.
    - Ensured formatting, indexing, and error-resilient I/O via exception handling.

11. Robust Error Handling and Fallback Mechanisms

- Introduced validation to handle:
    - Missing expected columns in user files.
    - Non-responsive or malformed URLs.
    - Invalid file formats or unsupported Excel versions.
- Provided user-friendly fallback messages, such as:
    - "Required columns missing: 'Conversions', 'Campaign Type'."
    - "All URLs are working (no 404 errors detected)."

12. Testing with Real-World Campaign Files

- Collected sample Excel data mimicking real Google Ads campaign structures.

- Simulated edge cases like:
    - Sheets with missing or misnamed columns.
    - Campaigns with no active ad groups.
    - Keyword rows with null or corrupted Final URLs.
- Refined each logic block to withstand diverse data layouts and schemas.

13. Final Output and Usability Enhancements

- Ensured that every upload:
    - Produces a real-time web preview of results.
    - Returns a clean downloadable Excel report with a UUID filename.
- Structured output to allow stakeholders to:
    - View summary insights in-browser.
    - Retain a permanent record of compliance via the Excel report.

14. Documentation and Reporting

- Documented the workflow and results.
- Prepared final report and presentation slides for review.

Key Insights and Learnings

The internship not only expanded my technical capabilities but also deeply shaped the way I approach real-world problem-solving, system design, and execution. The following are the most significant insights and takeaways from the 45-day project:

1. The Importance of Structured Problem-Solving and Time Management

- At the outset, the scope of the project seemed overwhelming — learning Django from scratch, understanding SEM campaign mechanics, and delivering a production-ready tool.
- By breaking down the entire system into modular, manageable components (e.g., file upload → parsing → validation → report generation), I learned how to deconstruct complex problems into solvable units.
- Setting micro-goals each week helped ensure continuous progress — from backend logic implementation to frontend rendering and Excel export functionality.
- This taught me that consistency, prioritization, and clarity of goals are far more valuable than just raw coding hours.

2. Gained Deep Understanding of Django and Web Application Development

- Coming into the internship with no background in Django, I now have a working proficiency in:
    - Django's MVT architecture.
    - Routing mechanisms (urls.py), view handling (views.py), and form submissions.
    - Static and dynamic file serving, and error-tolerant server-side data processing.
- Beyond syntax, I now understand how backend logic interacts with user input, how views connect with templates, and how security and performance are managed in real applications.
- This foundation is crucial for any scalable software engineering or data pipeline development in the future.

3. Learned How to Handle Real-World, Unstructured, and Messy Data

- Unlike textbook datasets, the SEM campaign Excel files I worked with had:
    - Inconsistent column names (extra whitespace, casing issues).
    - Missing or null fields (especially in critical columns like conversions or URLs).
    - Sheets with varied structures — some wide, some deeply nested.
- I learned to:

    - Use .str.strip() and column validation before processing.
    - Build defensive code with conditionals and fallbacks (e.g., checking for required columns).
    - Communicate errors clearly to the user to enable fast debugging.
- This helped me develop resilience and adaptability when working with real-world data pipelines.

4. Strengthened My Coding, Debugging, and Logic Design Skills

- Over 20 custom rules were written, each with unique logic, conditions, and validation mechanisms — covering campaign settings, keyword volume, ad formats, URL validation, and more.
- This pushed me to:
    - Write clean, modular functions for reusability.
    - Use detailed logging and try/except blocks to trap and fix edge cases.
    - Make my logic both explainable and efficient — not just “working,” but also interpretable and testable.
- I also learned how to balance readability and functionality, especially under tight development timelines.

5. Understood the Significance of Documentation, UI Feedback, and Testing

- I learned that even the most powerful backend logic is ineffective unless:
    - Users receive clear, actionable feedback (e.g., "Campaign Name column is missing").
    - The interface reflects results in a readable, non-overwhelming manner.
    - There is documentation for each function, so future developers or users can maintain or enhance the codebase.
- Whether through in-line HTML tables, summarized Excel sheets, or console-level debugging — I saw how communication through code is as important as logic itself.

6. Exposure to Iterative Development and Research-Oriented Thinking

- The project involved repeated cycles of:
    - Researching how a campaign validation works (e.g., What makes an RSA excellent?).
    - Designing logic based on documentation (e.g., Google Ads API standards).
    - Implementing a first version, testing it on real data, and refining it for edge cases.
- This iterative loop taught me to:
    - Embrace failure early and improve incrementally.
    - Base code design on evidence, patterns, and reusability — not assumptions.
    - Build systems that are extensible (e.g., adding new rules or inputs with minimal code changes).

7. Learned How to Translate Business Requirements into Code

- The checklist questions provided were written in business terms: “Are sitelinks missing descriptions?” or “Are audience signals present in Performance Max?”
- I had to bridge the gap between business logic and technical logic:
    - Identify which columns in the Excel file relate to the question.
    - Translate that into code that validates, quantifies, and explains the result.
- This helped me appreciate the skill of abstracting domain knowledge into engineering logic, which is a critical competency in data roles and full-stack engineering.

8. Developed Confidence to Deliver End-to-End Functional Software

- The final product is a full-stack, real-world application that:
- Accepts user input.
- Processes it securely and accurately.
- Displays results in-browser and generates downloadable reports.

- This internship gave me the confidence to start from zero and ship something complete, independently — an experience far more powerful than academic exercises or classroom projects.

Conclusion

The 45-day summer internship was a valuable learning experience that helped bridge the gap between academic knowledge and practical application. It not only improved my technical skills but also taught me how to work collaboratively and independently in a professional setting. I am confident that the experience gained during this internship will significantly contribute to my academic and career growth.