140509_23.md

README

23. Autonomous Data Analysis Agent

Summary: Develop an AI agent that automates exploratory data analysis (EDA), pattern detection, statistical testing, and visualization for diverse datasets.

Problem Statement: Exploratory data analysis is time-consuming and requires statistical expertise, limiting accessibility for non-experts. Your task is to create an autonomous AI agent that automates EDA by profiling datasets, detecting patterns, performing statistical tests, generating visualizations, and providing natural language insights. The system should adapt to various data types (numerical, categorical, time-series) and deliver actionable findings for data-driven decision-making.

Steps: - Design automated data profiling to summarize dataset characteristics. - Implement adaptive analysis strategies for pattern detection. - Create statistical testing and hypothesis generation mechanisms. - Build dynamic visualization tools with interactive charts. - Develop natural language explanations for insights. - Include ranking and prioritization of findings based on significance.

Suggested Data Requirements: - Diverse datasets (numerical, categorical, time-series) for testing. - Statistical test catalogs and benchmarks (e.g., NIST datasets). - Visualization templates and user feedback logs. - Domain-specific ontologies for contextual insights.

Themes: Agentic AI, Classical AI/ML

The steps and data requirements outlined above are intended solely as reference points to assist you in conceptualizing your solution.

PRD (Product Requirements Document)

Product Vision and Goals

The Autonomous Data Analysis Agent aims to democratize data analysis by automating EDA, reducing analysis time by 70% while delivering statistically significant insights accessible to non-experts. Goals include supporting diverse datasets (e.g., financial, healthcare, retail), generating interactive visualizations, and providing clear, narrative-driven insights, enabling faster decision-making for analysts, business users, and researchers.

Target Audience and Stakeholders

- **Primary Users**: Data analysts, business intelligence professionals, researchers, non-expert data enthusiasts.
- **Stakeholders**: Data scientists (for validation), business managers (for decision-making), educators (for teaching tools).
- **Personas**:
  - A business analyst exploring sales trends to optimize pricing.
  - A researcher analyzing patient data for clinical patterns.

Key Features and Functionality

- **Data Profiling**: Automatically summarize dataset (e.g., mean, variance, missing values).
- **Pattern Detection**: Identify trends, outliers, and correlations adaptively.
- **Statistical Testing**: Perform hypothesis tests (e.g., t-tests, ANOVA) based on data type.
- **Visualization**: Generate interactive charts (e.g., histograms, scatter plots) with Plotly.
- **Narrative Generation**: Produce natural language summaries of findings using LLMs.
- **Insight Ranking**: Prioritize findings by statistical significance (e.g., p-value).
- **Export**: Save insights and visuals as Markdown/PDF/HTML.

Business Requirements

- Integration with data sources (CSV, SQL, APIs like Kaggle).
- Freemium model: Basic analysis free, premium for advanced stats and large datasets.
- Export compatibility with BI tools (e.g., Tableau, Power BI).

Success Metrics

- **Efficiency**: <5min analysis for 100k-row datasets.
- **Accuracy**: >90% insight relevance (user-rated).
- **Engagement**: >10 insights generated per session.
- **User Satisfaction**: NPS >75.

Assumptions, Risks, and Dependencies

- **Assumptions**: Access to open datasets (e.g., UCI, Kaggle) and ML libraries (Pandas, SciPy).
- **Risks**: Overfitting to noisy data; mitigate with robust preprocessing.
- **Dependencies**: Datasets like Titanic, Iris; libraries like Pandas, Plotly, Hugging Face.

Out of Scope

- Real-time streaming data analysis.
- Custom model training for predictions.

FRD (Functional Requirements Document)

System Modules and Requirements

1. **Data Profiling Module (FR-001)**:
    - **Input**: Dataset (CSV, SQL, JSON).
    - **Functionality**: Compute summary statistics (mean, std, quartiles, missing values) using Pandas; detect data types (numerical, categorical).
    - **Output**: Profile report (e.g., JSON with stats, distributions).
    - **Validation**: Cross-check stats with manual calculations.
2. **Pattern Detection Module (FR-002)**:
    - **Input**: Profile report.
    - **Functionality**: Identify trends (e.g., linear regression slopes), outliers (z-scores >3), correlations (Pearson/Spearman).
    - **Output**: List of patterns (e.g., {â€œtypeâ€: â€œcorrelationâ€, â€œvarsâ€: [â€œsalesâ€, â€œpriceâ€], â€œvalueâ€: 0.85}).
    - **Validation**: Filter low-confidence patterns (e.g., correlation <0.3).
3. **Statistical Testing Module (FR-003)**:
    - **Input**: Patterns, dataset.
    - **Functionality**: Select tests (e.g., t-test for numerical, chi-square for categorical) based on data type; generate hypotheses (e.g., â€œmeans differâ€).
    - **Output**: Test results with p-values, effect sizes.
    - **Validation**: Ensure p-values <0.05 for significant findings.
4. **Visualization Module (FR-004)**:
    - **Input**: Patterns, test results.
    - **Functionality**: Generate Plotly charts (e.g., histograms, heatmaps); support interactivity (zoom, hover).
    - **Output**: HTML/JSON visualizations.
    - **Validation**: Verify chart data matches raw stats.
5. **Narrative Generation Module (FR-005)**:
    - **Input**: Patterns, test results.
    - **Functionality**: Use LLM to generate explanations (e.g., â€œStrong correlation between sales and price suggestsâ€¦â€).
    - **Output**: Natural language summaries in Markdown.
    - **Validation**: Check readability (Flesch score >60).
6. **Insight Ranking Module (FR-006)**:
    - **Input**: Patterns, test results.
    - **Functionality**: Rank by significance (p-value, effect size) or user-defined criteria (e.g., business impact).
    - **Output**: Prioritized list of insights.
    - **Validation**: Ensure top insights align with statistical significance.

Interfaces and Integrations

- **UI**: Web-based dashboard (Streamlit) for data upload, insight review, and visualization.

- **API**: RESTful endpoints (e.g., POST /analyze, GET /insights) with JSON payloads.
- **Data Flow**: Upload data -> Profile -> Detect patterns -> Test -> Visualize -> Narrate -> Rank.
- **Integrations**: Pandas for profiling, SciPy for stats, Plotly for visuals, Hugging Face for LLM.

Error Handling and Validation

- **Invalid Data**: Handle missing values with imputation (mean/median) or flag errors.
- **Statistical Errors**: Skip tests with insufficient data (e.g., <30 samples for t-test).
- **Tests**: Unit tests for profiling (90% coverage), E2E for full pipeline.

NFRD (Non-Functional Requirements Document)

Performance Requirements

- **Latency**: <5min for 100k-row dataset analysis on standard hardware (16GB RAM, 4 vCPUs).
- **Throughput**: 20 concurrent analyses.

Scalability and Availability

- **Scalability**: Dockerized services with horizontal scaling for large datasets.
- **Availability**: 99% uptime; cache common datasets in Redis.

Security and Privacy

- **Data Privacy**: Process data locally or encrypt uploads; delete after session.
- **Authentication**: Optional OAuth for premium features.
- **Compliance**: GDPR for user-uploaded data, audit logs for analysis.

Reliability and Maintainability

- **Error Rate**: <1% analysis failures.
- **Code Quality**: Modular design, 85% test coverage, CI/CD with GitHub Actions.
- **Monitoring**: Prometheus for latency, error tracking; Grafana for dashboards.

Usability and Accessibility

- **UI/UX**: Responsive Streamlit app, WCAG 2.1 AA compliance (e.g., high-contrast mode).
- **Documentation**: API docs via Swagger, user guides with examples.

Environmental Constraints

- **Deployment**: Cloud (AWS/GCP) or on-prem with Docker.
- **Cost**: Optimize for <0.01 USD per analysis.

AD (Architecture Diagram)

+â€”â€”â€”â€”â€”â€”â€”+ | User Interface | (Streamlit: Data Upload, Insight Dashboard, Interactive Charts) +â€”â€”â€”â€”â€”â€”â€”+ | v +â€”â€”â€”â€”â€”â€”â€”+ | API Gateway | (FastAPI: Endpoints for Analysis, Visualizations) +â€”â€”â€”â€”â€”â€”â€”+ / | v v v +â€”â€”â€”â€”â€”â€”â€”+ +â€”â€”â€”â€”â€”â€”â€”+ +â€”â€”â€”â€”â€”â€”â€”+ | Profiler | | Pattern/Stats | | Visualizer/Narrator | | (Pandas) | | (SciPy, sklearn) | | (Plotly, LLM) | +â€”â€”â€”â€”â€”â€”â€”+ +â€”â€”â€”â€”â€”â€”â€”+ +â€”â€”â€”â€”â€”â€”â€”+ | ^ v | +â€”â€”â€”â€”â€”â€”â€”+ | | Data Store | <â€”â€”-+ | (Redis, Local Cache)| +â€”â€”â€”â€”â€”â€”â€”+ text## HLD (High Level Design)

- **Components**:
  - **Frontend**: Streamlit for dashboard, Plotly for interactive charts.
  - **Backend**: FastAPI for APIs, Celery for async analysis tasks.
  - **AI/ML**: Pandas for profiling, SciPy for stats, Hugging Face Transformers for narratives (e.g., Llama-3-8B).
  - **Storage**: Redis for caching dataset profiles, local storage for temporary uploads.
- **Design Patterns**:
  - **Pipeline**: Sequential flow (profile -> detect -> test -> visualize -> narrate).

    - **Strategy**: Adaptive test selection based on data type.
    - **Observer**: Real-time chart updates on user interaction.
- **Data Management**:
    - **Sources**: UCI datasets (e.g., Titanic, Iris), Kaggle (e.g., Sales Forecasting).
    - **Storage**: Cache processed datasets in Redis for repeat analyses.
- **Security Design**:
    - Encrypt uploads with AES-256.
    - Optional JWT for premium API access.
- **High-Level Flow**:
    1. Upload dataset (CSV/SQL).
    2. Profile data (stats, types).
    3. Detect patterns and run statistical tests.
    4. Generate visualizations and narratives.
    5. Rank insights and display/export results.

LLD (Low Level Design)

- **Data Profiling**:
    - Load: `df = pd.read_csv(file)`
    - Stats: `profile = df.describe(include='all')`
    - Types: `types = df.dtypes.to_dict()`
- **Pattern Detection**:
    - Trends: `slope, _ = scipy.stats.linregress(df['time'], df['value'])`
    - Outliers: `outliers = df[abs(stats.zscore(df['col'])) > 3]`
    - Correlations: `corr = df.corr(method='pearson')`
- **Statistical Testing**:
    - Select: `test = 'ttest' if is_numerical(df['col']) else 'chi2'`
    - Run: `p_value = scipy.stats.ttest_ind(df['col1'], df['col2'])[1]`
    - Hypothesize: `hypo = f"Means differ (p={p_value})" if p_value < 0.05 else "No difference"`
- **Visualization**:
    - Plot: `fig = go.Figure(go.Histogram(x=df['col']))`
    - Export: `fig.write_html('chart.html')`
- **Narrative Generation**:
    - Prompt: `"Summarize: {patterns}, {test_results}"`
    - Generate: `narrative = llm.generate(prompt, max_length=200)`
- **Insight Ranking**:
    - Rank: `insights.sort(key=lambda x: x['p_value'] if x['p_value'] else 1)`

Pseudocode

```python class DataAnalysisAgent: def **init**(self): self.profiler = pandas self.stats = scipy.stats self.llm = HuggingFaceModel(â€œmeta-llama/Llama-3-8bâ€) self.viz = plotly.graph_objects self.cache = RedisClient()

```
def profile(self, dataset):
    df = self.profiler.read_csv(dataset) if dataset.endswith('.csv') else self.profiler.read_sql(dataset)
    profile = df.describe(include='all').to_dict()
    types = df.dtypes.to_dict()
    self.cache.save(profile, dataset_id)
    return profile, types

def detect_patterns(self, df, profile):
    patterns = []
    for col in df.columns:
        if profile['types'][col] == 'numerical':
            slope, _ = self.stats.linregress(df.index, df[col])
            outliers = df[abs(self.stats.zscore(df[col])) > 3]
            patterns.append({'type': 'trend', 'col': col, 'slope': slope})
            patterns.append({'type': 'outlier', 'col': col, 'values': outliers.tolist()})
    corr = df.corr(method='pearson')
    for c1, c2 in combinations(df.columns, 2):
        if abs(corr.loc[c1, c2]) > 0.3:
            patterns.append({'type': 'correlation', 'vars': [c1, c2], 'value': corr.loc[c1, c2]})
    return patterns

def test_stats(self, df, patterns):
    results = []
    for p in patterns:
```
```

```
        if p['type'] == 'correlation':
            test = self.stats.pearsonr(df[p['vars'][0]], df[p['vars'][1]])
            results.append({'hypothesis': f"{p['vars']} correlated", 'p_value': test[1], 'effect': test[0]})
        elif p['type'] == 'outlier':
            results.append({'hypothesis': f"Outliers in {p['col']}", 'p_value': None})
    return results

def visualize(self, df, patterns, results):
    figs = []
    for p in patterns:
        if p['type'] == 'trend':
            figs.append(self.viz.Figure(self.viz.Scatter(x=df.index, y=df[p['col']])))
        elif p['type'] == 'correlation':
            figs.append(self.viz.Figure(self.viz.Heatmap(z=df.corr())))
    return [fig.to_json() for fig in figs]

def narrate(self, patterns, results):
    prompt = f"Summarize patterns: {patterns}\nTest results: {results}"
    narrative = self.llm.generate(prompt, max_length=200)
    return narrative

def rank_insights(self, patterns, results):
    insights = patterns + [{'type': 'test', 'data': r} for r in results]
    return sorted(insights, key=lambda x: x.get('p_value', 1) or x.get('effect', 0), reverse=True)

def analyze(self, dataset):
    profile, types = self.profile(dataset)
    patterns = self.detect_patterns(dataset, profile)
    results = self.test_stats(dataset, patterns)
    visuals = self.visualize(dataset, patterns, results)
    narrative = self.narrate(patterns, results)
    ranked = self.rank_insights(patterns, results)
    return {"profile": profile, "insights": ranked, "visuals": visuals, "narrative": narrative}
```