Product Requirements Document

Cluster Visualization with Normal Distribution and Overlap

Author: Yair Levi
Date: October 15, 2025
Version: 1.0
Status: Final
Document Type: Technical Specification

---

1. Executive Summary

This document outlines the requirements for a Python-based cluster visualization system that generates synthetic data with normal distribution and visualizes overlapping clusters with clear boundary demonstrations. The system creates 3 distinct clusters with controlled overlap to demonstrate clustering concepts and K-Means algorithm behavior.

1.1 Purpose

Develop an educational and analytical tool for visualizing multi-cluster data distributions with normal distributions, featuring configurable overlap regions and multiple visualization approaches including statistical boundaries and machine learning-based clustering.

1.2 Business Goals

- Provide visual demonstration of clustering concepts for educational purposes
- Showcase the behavior of overlapping clusters in machine learning
- Enable researchers to understand cluster separation and K-Means performance
- Support data science education and training programs
- Create publication-quality visualizations for academic presentations

1.3 Success Criteria

- Generate exactly 6000 points with specified distribution
- Create 3 distinct clusters with different statistical parameters
- Implement 2000 overlapping points across all clusters
- Produce clear, professional visualizations with cluster boundaries
- Execute in under 10 seconds
- Generate publication-quality output (300 DPI)

---

2. Product Overview

2.1 Target Users

User Type	Description	Primary Use Case
Data Science Students	Learning clustering algorithms	Understanding K-Means and cluster separation
Educators	Teaching machine learning	Classroom demonstrations of clustering
Researchers	Analyzing cluster behavior	Algorithm performance evaluation
Data Scientists	Prototyping clustering solutions	Validation of clustering approaches
Academic Authors	Creating publications	Generating figures for papers

2.2 Key Features

Data Generation:

- 3 clusters with normal (Gaussian) distribution
- Configurable mean and standard deviation per cluster
- Exactly 2000 points per cluster (6000 total)
- 2000 overlapping points distributed across all clusters
- Reproducible results with random seed

Visualization:

- Dual-panel comparison (original vs K-Means)
- Cluster boundary visualization (2σ ellipses)
- Decision boundary demonstration
- Overlap region highlighting
- Color-coded clusters with transparency
- Professional formatting with legends and labels

Analysis:

- K-Means clustering (k=3)
- Cluster center identification
- Statistical boundary calculation
- Point classification (overlap vs unique)

2.3 System Architecture

Input Parameters
↓
[Data Generation Module]
• Generate overlap points (2000)
• Distribute to clusters (~667 each)
• Generate unique points per cluster (~1333 each)
↓
[Statistical Analysis]
• Calculate cluster parameters
• Compute boundaries (2σ)
↓
[K-Means Clustering]
• Fit K-Means (k=3)
• Predict labels
• Find cluster centers
↓
[Visualization Module]
• Plot 1: Original with ellipse boundaries
• Plot 2: K-Means with decision boundaries
↓
Output: Display + PNG file (300 DPI)

---

3. Functional Requirements

3.1 Data Generation (FR-1)

Priority: Critical
Complexity: Medium

Requirements

FR-1.1: Total Point Count

- Generate exactly **6000 points**
- Distributed as: 2000 per cluster × 3 clusters
- Verification: Assert total count equals 6000

FR-1.2: Cluster Distribution Each of 3 clusters must have:

- Exactly **2000 points**
- Unique mean (μx, μy)
- Unique standard deviation (σx, σy)
- Normal (Gaussian) distribution for both X and Y coordinates

FR-1.3: Cluster Parameters

Cluster	Mean (X, Y)	Std Dev (X, Y)	Color	Description
1	(2, 2)	(1.5, 1.0)	Red	Lower-left cluster, wider horizontally
2	(8, 3)	(1.2, 1.8)	Green	Right cluster, taller vertically
3	(5, 8)	(2.0, 1.3)	Blue	Top-center cluster, wider horizontally

FR-1.4: Overlap Points

- Total **2000 overlap points**
- Generated from central region: mean (5, 4.5), std (2.5, 2.0)
- Distributed across all 3 clusters (~667 per cluster)
- These points count toward each cluster's 2000-point total
- Overlap points must be distinguishable in visualization (yellow stars)

FR-1.5: Normal Distribution

python

```
X_cluster = np.random.normal(mean_x, std_x, n_points)
Y_cluster = np.random.normal(mean_y, std_y, n_points)
```

FR-1.6: Reproducibility

- Use fixed random seed (42) for consistent results
- Same output every execution
- Enable comparison and debugging

Acceptance Criteria:

- [x] Total points = 6000 (verified with assertion)
- [x] Each cluster has exactly 2000 points
- [x] Overlap points = 2000 (verified with assertion)
- [x] All points follow normal distribution
- [x] Each cluster has unique mean and std dev
- [x] Results reproducible across runs

3.2 Statistical Analysis (FR-2)

Priority: High
Complexity: Low

Requirements

FR-2.1: Cluster Boundaries

- Calculate 2-sigma (2σ) ellipse for each cluster
- Ellipse dimensions: 4×std_x by 4×std_y
- Covers ~95% of normal distribution
- Display as dashed lines in cluster colors

FR-2.2: Point Classification

- Track each point type: "overlap" or "unique"

- Store as metadata array parallel to points
- Enable differential visualization

FR-2.3: Statistics Reporting Display:

- Total point count
- Points per cluster
- Overlap point count and percentage
- Unique point count

Acceptance Criteria:

- [x] Ellipses correctly sized (4σ width/height)
- [x] Point types correctly classified
- [x] Statistics accurate and displayed

3.3 K-Means Clustering (FR-3)

Priority: High
Complexity: Low

Requirements

FR-3.1: Algorithm Configuration

- Use scikit-learn KMeans
- Number of clusters: k=3
- Random state: 42 (reproducible)
- n_init: 10 (multiple initializations)

FR-3.2: Clustering Execution

python

```
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
predicted_labels = kmeans.fit_predict(all_points)
centers = kmeans.cluster_centers_
```

FR-3.3: Output Requirements

- Predicted cluster labels for all 6000 points
- 3 cluster centers (x, y coordinates)
- Cluster assignments for visualization

FR-3.4: Performance Analysis

- Compare K-Means results with original labels
- Demonstrate how overlap affects clustering
- Show cluster center locations

Acceptance Criteria:

- [x] K-Means converges successfully
- [x] 3 cluster centers identified
- [x] All points assigned to clusters
- [x] Results reproducible

3.4 Visualization (FR-4)

Priority: Critical
Complexity: High

Requirements

FR-4.1: Figure Layout

- Two side-by-side plots (1×2 subplot grid)
- Figure size: 16" × 7"
- Main title: "Cluster Visualization with Normal Distribution"
- Subtitle: "Author: Yair Levi"

FR-4.2: Plot 1 - Original Clusters with Boundaries

Content:

- Unique points: Colored by cluster (red/green/blue), small circles
- Overlap points: Yellow stars, larger size
- Cluster boundaries: Dashed ellipses (2σ)
- Legend with all elements
- Grid with transparency

Styling:

- Point size: 30 (unique), 40 (overlap)
- Alpha: 0.6 (unique), 0.7 (overlap)
- Edge colors: Black, linewidth 0.3-0.5
- Ellipse: Dashed line, width 2, no fill

Labels:

- Title: "Original Clusters with Overlap Points Highlighted"
- X-axis: "X Coordinate"
- Y-axis: "Y Coordinate"
- Aspect ratio: Equal

FR-4.3: Plot 2 - K-Means Results with Decision Boundaries

Content:

- Points colored by K-Means prediction
- Overlap points highlighted with yellow stars
- Cluster centers: Black X markers, large size
- Decision boundaries: Dashed contour lines
- Legend with clusters and centers

Decision Boundaries:

- Create 500×500 mesh grid
- Predict cluster for each mesh point

- Draw contour at decision boundaries
- Color: Black, dashed, width 2

Cluster Centers:

- Marker: 'X'
- Size: 300
- Color: Black
- Edge: White, width 2
- Z-order: 5 (on top)

Labels:

- Title: "K-Means Clustering Result (k=3)"
- X-axis: "X Coordinate"
- Y-axis: "Y Coordinate"
- Aspect ratio: Equal

FR-4.4: Statistics Box Display in figure:

Statistics:
Total Points: 6000
Points per Cluster: 2000
Overlap Points: 2000 (33.3%)
Unique Points: 4000

Position: Lower-left corner Style: Rounded box, wheat color, alpha 0.5

FR-4.5: Output Files

- Display: Interactive matplotlib window
- File: cluster_visualization.png
- Resolution: 300 DPI
- Format: PNG with tight bounding box

Acceptance Criteria:

- [x] Both plots display correctly
- [x] All elements visible and properly styled
- [x] Legends accurate and complete
- [x] Statistics box shows correct values
- [x] Output file saved at 300 DPI
- [x] Professional publication quality

3.5 Console Output (FR-5)

Priority: Medium
Complexity: Low

Requirements

FR-5.1: Progress Reporting Display:

1. Program header with author
2. Configuration summary
3. Data generation progress
4. Cluster parameter details
5. Visualization creation status
6. K-Means execution confirmation
7. File save confirmation
8. Final summary

FR-5.2: Output Format

```
================================================================================
CLUSTER VISUALIZATION WITH NORMAL DISTRIBUTION
Author: Yair Levi
================================================================================

Generating 6000 points total:
  - 2000 points per cluster (3 clusters)
  - 2000 points are overlap (shared across all clusters)
  - Distribution: Each cluster gets ~667 overlap + ~1333 unique

Cluster Parameters:

Cluster 1 (Red):
  Mean: (2, 2)
  Std Dev: (1.5, 1.0)
  Points: 667 overlap + 1333 unique = 2000 total

[...]

✓ Total points generated: 6000

[...]

✓ Program execution completed successfully

Author: Yair Levi
```

Acceptance Criteria:

- [x] Clear progress indicators
- [x] All statistics displayed
- [x] Professional formatting

- [x] Success confirmations visible

---

4. Non-Functional Requirements

4.1 Performance (NFR-1)

Priority: High

Metric	Requirement	Target	Measurement
Data Generation	< 5 seconds	2 seconds	Time module
K-Means Clustering	< 5 seconds	1 second	Time module
Visualization	< 10 seconds	5 seconds	Time module
Total Execution	< 15 seconds	8 seconds	End-to-end
Memory Usage	< 500 MB	200 MB	memory_profiler
File Size (PNG)	< 5 MB	2 MB	File size

4.2 Quality (NFR-2)

Priority: High

Visual Quality:

- Resolution: 300 DPI minimum
- Anti-aliasing: Enabled
- Color contrast: Sufficient for colorblind accessibility
- Text: Readable at print size
- Lines: Smooth, no pixelation

Code Quality:

- PEP 8 compliant
- Docstrings for main functions
- Comments explaining complex logic
- No magic numbers (use constants)
- Type hints where appropriate

4.3 Usability (NFR-3)

Priority: High

Ease of Use:

- Single command execution
- No user input required
- Automatic file saving
- Clear console output
- No configuration files needed

Configuration:

- Constants at top of file
- Easy to modify parameters
- Clear variable names
- Commented configuration section

4.4 Reliability (NFR-4)

Priority: High

Robustness:

- No crashes under normal conditions
- Assertions verify data integrity
- Graceful handling of edge cases
- Reproducible results (fixed seed)

Error Prevention:

- Input validation (not applicable - no user input)
- Array dimension checks
- Cluster count verification

4.5 Portability (NFR-5)

Priority: Medium

Cross-Platform:

- Windows 10/11: Full support
- macOS (Intel/Apple Silicon): Full support
- Linux (Ubuntu 20.04+): Full support

Dependencies:

- Python 3.7+
- NumPy ≥1.19.0
- Matplotlib ≥3.3.0
- scikit-learn ≥0.24.0
- All available via pip

Virtual Environment:

- requirements.txt provided
- Compatible with venv, conda, virtualenv
- No system-level dependencies

---

5. Technical Specifications

5.1 Technology Stack

Component	Technology	Version	Purpose
Language	Python	3.7+	Implementation
Numerical Computing	NumPy	≥1.19.0	Array operations, random generation
Visualization	Matplotlib	≥3.3.0	Plotting and figure generation
Machine Learning	scikit-learn	≥0.24.0	K-Means clustering

5.2 Data Structures

Points Array:

python

```
all_points: np.ndarray
Shape: (6000, 2)
dtype: float64
# Column 0: X coordinates
# Column 1: Y coordinates
```

Labels Array:

python

```
all_labels: np.ndarray
Shape: (6000,)
dtype: int
# Values: 0, 1, 2 (cluster assignments)
```

Point Types Array:

python

```
point_types: np.ndarray
Shape: (6000,)
dtype: str
# Values: 'overlap' or 'unique'
```

Cluster Parameters:

python

```
cluster_params = [
    {
        'name': str,
        'mean': [float, float],
        'std': [float, float],
        'color': str,
        'alpha': float
    },
    ...
]
```

5.3 Algorithm Complexity

Data Generation:

- Time: O(n) where n = 6000
- Space: O(n)

K-Means:

- Time: O(k × n × i × d) where:
  - k = 3 (clusters)
  - n = 6000 (points)
  - i = iterations to converge
  - d = 2 (dimensions)
- Typical: O(18000) per iteration

Visualization:

- Time: O(n + m²) where:
  - n = 6000 (points to plot)
  - m = 500 (mesh grid resolution)
- Total: O(256000)

Overall:

- Time Complexity: O(n + k×n×i + m²)
- Space Complexity: O(n + m²)

5.4 Configuration Parameters

python

```
# Point Distribution
TOTAL_POINTS = 6000
POINTS_PER_CLUSTER = 2000
OVERLAP_POINTS = 2000

# Cluster 1 Parameters
CLUSTER1_MEAN = [2, 2]
CLUSTER1_STD = [1.5, 1.0]

# Cluster 2 Parameters
CLUSTER2_MEAN = [8, 3]
CLUSTER2_STD = [1.2, 1.8]

# Cluster 3 Parameters
CLUSTER3_MEAN = [5, 8]
CLUSTER3_STD = [2.0, 1.3]

# Overlap Parameters
OVERLAP_CENTER = [5, 4.5]
OVERLAP_STD = [2.5, 2.0]

# Visualization
FIGURE_SIZE = (16, 7)
DPI = 300
OUTPUT_FILE = 'cluster_visualization.png'

# K-Means
N_CLUSTERS = 3
RANDOM_STATE = 42
N_INIT = 10

# Colors
COLORS = ['red', 'green', 'blue']
OVERLAP_COLOR = 'yellow'
```

---

6. Use Cases

6.1 Educational Demonstration (UC-1)

Actor: Data Science Instructor
Goal: Teach clustering concepts in machine learning class
Preconditions: Python environment with dependencies installed

Main Flow:

1. Instructor runs the program in class
2. Shows data generation process
3. Explains normal distribution parameters
4. Discusses concept of overlapping clusters
5. Demonstrates K-Means algorithm behavior
6. Shows how overlap affects clustering accuracy
7. Explains decision boundaries
8. Students see visual comparison of true vs predicted clusters

Postconditions: Students understand clustering challenges with overlap

Success Metrics:

- Student comprehension: > 85%
- Engagement: High (visual aids effective)
- Question quality: Improved understanding

6.2 Research Analysis (UC-2)

Actor: Machine Learning Researcher
Goal: Analyze K-Means performance with overlapping clusters
Preconditions: Understanding of clustering algorithms

Main Flow:

1. Researcher modifies overlap percentage
2. Runs multiple experiments with different parameters
3. Analyzes cluster center positioning
4. Measures classification accuracy in overlap region
5. Documents K-Means behavior with varying overlap
6. Compares with other clustering algorithms
7. Publishes findings with generated visualizations

Postconditions: Research insights on clustering with overlap

Success Metrics:

- Clear visualization of phenomenon
- Reproducible results
- Publication-quality figures

6.3 Algorithm Comparison (UC-3)

Actor: Data Scientist
Goal: Compare clustering algorithms on overlapping data
Preconditions: Familiarity with multiple clustering methods

Main Flow:

1. Uses program to generate standard test dataset
2. Saves point coordinates for reuse
3. Applies K-Means (current implementation)
4. Applies DBSCAN on same data
5. Applies Gaussian Mixture Models
6. Compares results visually and quantitatively
7. Selects best algorithm for specific use case

Postconditions: Algorithm selection decision made

Success Metrics:

- Clear performance differences visible
- Objective comparison possible
- Decision confidence: High

6.4 Publication Figure Generation (UC-4)

Actor: Academic Author
Goal: Create figure for research paper
Preconditions: Paper describing clustering research

Main Flow:

1. Author runs program with specific parameters
2. Adjusts colors and styling if needed
3. Generates 300 DPI PNG output
4. Incorporates into paper manuscript
5. Passes journal quality requirements
6. Publishes paper with figure

Postconditions: High-quality figure in published paper

Success Metrics:

- Image quality: Passes journal standards
- Visual clarity: Reviewers satisfied
- File size: Acceptable for submission

---

7. Constraints and Assumptions

7.1 Technical Constraints

Hard Constraints:

- Python 3.7+ required
- NumPy, Matplotlib, scikit-learn must be installed
- Graphical environment required (for display)
- Minimum 2GB RAM
- Single-threaded execution

Soft Constraints:

- Recommended: 8GB+ RAM for smooth execution
- Recommended: Modern CPU (2015+)
- Display resolution: ≥1920×1080 for best viewing

7.2 Data Constraints

Point Distribution:

- Total must equal 6000
- Each cluster must equal 2000
- Overlap must equal 2000
- Overlap distributed equally: ~667 per cluster

Normal Distribution:

- Assumes NumPy random normal is sufficient
- No validation of actual distribution shape
- Accepts NumPy default precision

7.3 Visualization Constraints

Display:

- Requires graphical display capability
- No headless server support without modification
- Interactive backend needed for plt.show()

File Output:

- PNG format only
- Fixed filename (overwrite if exists)
- No dynamic path selection

7.4 Assumptions

User Assumptions:

- Basic command-line proficiency
- Python installation capability
- Ability to install packages with pip
- Understanding of clustering concepts (optional but helpful)

Environment Assumptions:

- Stable Python environment
- Internet access for package installation
- Sufficient disk space (~50MB for packages + output)
- Write permissions in current directory

Algorithm Assumptions:

- K-Means appropriate for this data
- k=3 is correct cluster count
- Euclidean distance is appropriate metric
- 10 initializations sufficient for convergence

---

8. Dependencies and Integrations

8.1 External Dependencies

Dependency	Version	Purpose	Critical	License
NumPy	≥1.19.0	Numerical computing, random generation	Yes	BSD
Matplotlib	≥3.3.0	Visualization and plotting	Yes	PSF
scikit-learn	≥0.24.0	K-Means clustering	Yes	BSD

8.2 Standard Library Dependencies

- `random` (via NumPy): Random number generation
- `sys`: System operations

- Built-in math functions

8.3 Future Integrations

Planned (Version 2.0):

- Export to CSV for external analysis
- Integration with pandas DataFrames
- Support for other clustering algorithms (DBSCAN, GMM)
- Interactive parameter adjustment GUI
- Jupyter notebook version

Potential (Version 3.0):

- Web-based visualization
- Real-time parameter sliders
- 3D clustering support
- Animation of clustering process
- Batch processing multiple configurations

---

9. Success Metrics and KPIs

9.1 Technical Performance Metrics

Metric	Target	Measurement	Frequency
Execution Time	< 10s	Time module	Per run
Memory Usage	< 200 MB	memory_profiler	Weekly
Output Quality	300 DPI	File properties	Per run
Code Correctness	100%	Assertions	Per run
Reproducibility	100%	Manual verification	Daily

9.2 Quality Metrics

Metric	Target	Measurement
Visual Clarity	Excellent	User survey
Code Readability	> 4.0/5	Peer review
Documentation	Complete	Manual review
Error Rate	0	Error logs

9.3 Educational Impact Metrics

Metric	Target	Measurement
Student Comprehension	> 85%	Quiz scores
Instructor Satisfaction	> 4.5/5	Survey
Usage in Courses	10+ courses	Adoption tracking
Student Engagement	High	Observation

9.4 Research Impact Metrics

Metric	Target	Measurement
Papers Using Tool	5+	Citation tracking
Downloads	100+	GitHub stats
Stars/Forks	50+	GitHub metrics
Community Feedback	Positive	Issues/comments

10. Testing Requirements

10.1 Unit Tests

Required Test Cases:

Data Generation Tests:


python

```
test_total_point_count()  # Verify 6000 points
test_cluster_sizes()  # Each cluster has 2000
test_overlap_count()  # Verify 2000 overlap points
test_normal_distribution()  # Statistical validation
test_reproducibility()  # Same seed = same results
test_point_classification()  # Overlap vs unique correct
```

Statistical Tests:


python

```
test_cluster_means()  # Means close to specified
test_cluster_stds()  # Std devs within tolerance
test_ellipse_calculations()  # 2σ ellipses correct
```

K-Means Tests:


python

```
test_kmeans_convergence()  # Algorithm converges
test_center_count()  # 3 centers found
test_label_assignment()  # All points labeled
```

Visualization Tests:


python

```
test_figure_creation()  # Figure created successfully
test_file_output()  # PNG file saved
test_file_size()  # File not empty
test_dpi()  # Correct resolution
```

10.2 Integration Tests

End-to-End Workflow:

python

```
test_full_execution()  # Complete run succeeds
test_output_validity()  # All outputs present
test_no_errors()  # No exceptions raised
```

10.3 Visual Validation Tests

Manual Inspection:

- ☐ Clusters visually distinct
- ☐ Overlap region visible
- ☐ Boundaries correctly positioned
- ☐ Colors distinguishable
- ☐ Labels readable
- ☐ Legend complete
- ☐ Statistics accurate

10.4 Performance Tests

Benchmarks:

python

```
test_execution_speed()  # < 10 seconds
test_memory_usage()  # < 200 MB
test_scalability()  # Test with 60K points
```

---

11. Risks and Mitigation

Risk	Impact	Probability	Mitigation	Owner
Memory exhaustion with large datasets	High	Low	Add memory checks, limit max points	Dev Team
Matplotlib display issues	Medium	Medium	Add error handling, test multiple backends	Dev Team
K-Means convergence failure	Medium	Low	Increase n_init, add convergence checking	Dev Team
File permission errors	Low	Medium	Add try-catch, check write permissions	Dev Team
Dependency version conflicts	Medium	Medium	Document exact versions, test on clean environment	Dev Team
Random seed not working	High	Very Low	Validate reproducibility in tests	QA Team
Visual quality insufficient	Medium	Low	User testing, adjust DPI and styling	Design Team
Cross-platform display issues	Medium	Medium	Test on all major platforms	QA Team

---

12. Future Enhancements

12.1 Phase 2 Features (Q1 2026)

Priority: High

- [ ] Command-line arguments for parameters
- [ ] CSV export of generated points
- [ ] Configurable number of clusters (k)
- [ ] Additional clustering algorithms (DBSCAN, GMM)
- [ ] PDF output format option
- [ ] Batch processing multiple configurations

Priority: Medium

- [ ] Interactive parameter tuning (GUI)
- [ ] Animation of K-Means iterations
- [ ] Cluster quality metrics (silhouette score)
- [ ] 3D visualization option
- [ ] Custom color schemes
- [ ] Jupyter notebook integration

Priority: Low

- [ ] Web-based interface
- [ ] Real-time parameter sliders
- [ ] Export to various formats (SVG, EPS)
- [ ] Comparison with ground truth
- [ ] Statistical significance tests

12.2 Advanced Features (Version 2.0 - Q3 2026)

- Different distribution types (uniform, exponential)
- Non-spherical cluster shapes
- Density-based clustering visualization
- Hierarchical clustering dendrograms
- Dimensionality reduction (PCA, t-SNE)
- Cluster evolution over time
- Multi-page PDF reports
- Integration with ML frameworks (TensorFlow, PyTorch)

---

13. Documentation Requirements

13.1 User Documentation

Required Documents:

1. **README.md** (Complete)
   - Installation instructions
   - Quick start guide
   - Configuration options
   - Troubleshooting section
   - Output explanation
2. **User Guide** (5 pages)
   - Detailed usage instructions
   - Parameter tuning guide
   - Interpretation of results
   - Best practices
   - FAQ
3. **Examples** (Optional)
   - Different parameter configurations
   - Use case demonstrations
   - Expected outputs

13.2 Technical Documentation

Required Documents:

1. **Technical Specification** (This PRD)
   - Complete requirements
   - Algorithm details
   - Implementation notes
2. **Code Documentation**
   - Inline comments
   - Docstrings for main sections
   - Algorithm explanations
   - Mathematical formulas
3. **API Documentation** (If extracted as library)
   - Function signatures
   - Parameter descriptions
   - Return value specifications
   - Usage examples

13.3 Requirements File

requirements.txt (Provided)

- All dependencies listed
- Version constraints specified
- Comments explaining each package

---

14. Acceptance Criteria

14.1 Functional Acceptance

The system is accepted if:

✅ **Data Generation:**

- ☑ Generates exactly 6000 points
- ☑ Creates 3 clusters with 2000 points each
- ☑ Includes 2000 overlap points
- ☑ Uses normal distribution with correct parameters
- ☑ Results are reproducible

✅ **Visualization:**

- ☑ Creates two side-by-side plots
- ☑ Shows cluster boundaries (ellipses)
- ☑ Shows decision boundaries (K-Means)
- ☑ Highlights overlap points distinctly
- ☑ Includes complete legends
- ☑ Displays statistics box

✅ **K-Means:**

- ☑ Executes successfully
- ☑ Finds 3 cluster centers
- ☑ Assigns all points to clusters
- ☑ Shows reasonable clustering

✅ **Output:**

- ☑ Displays interactive plot
- ☑ Saves PNG file at 300 DPI
- ☑ Console output is clear and informative

14.2 Performance Acceptance

The system is accepted if:

✅ **Speed:**

- ☑ Executes in < 10 seconds
- ☑ Data generation < 5 seconds
- ☑ Visualization < 5 seconds

✅ **Resources:**

- ☑ Memory usage < 500 MB
- ☑ File size < 5 MB
- ☑ CPU usage reasonable

14.3 Quality Acceptance

The system is accepted if:

✅ **Code Quality:**

- ☑ Follows PEP 8 style guide
- ☑ Includes meaningful comments
- ☑ No critical bugs

- ☑ Passes all assertions

✅ **Visual Quality:**

- ☑ 300 DPI output
- ☑ Colors distinguishable
- ☑ Labels readable
- ☑ Professional appearance
- ☑ Suitable for publication

✅ **Usability:**

- ☑ Single command execution
- ☑ No user input required
- ☑ Clear console feedback
- ☑ Easy to modify parameters

---

15. Deployment Plan

15.1 Deployment Method

Distribution:

- GitHub repository (public)
- Two files: Python script + requirements.txt
- README with instructions
- License file (MIT)

15.2 Installation Steps

For End Users:


bash

```
# Step 1: Clone repository
git clone https://github.com/yairlevi/cluster-visualization.git
cd cluster-visualization

# Step 2: Create virtual environment
python -m venv venv

# Step 3: Activate virtual environment
# Windows:
venv\Scripts\activate
# Linux/Mac:
source venv/bin/activate

# Step 4: Install dependencies
pip install -r requirements.txt

# Step 5: Run program
python cluster_visualization.py
```

15.3 Verification Steps

Post-Installation Verification:

1. Check Python version: `python --version` (should be 3.7+)
2. Verify packages: `pip list | grep -E "numpy|matplotlib|scikit-learn"`
3. Test run: `python cluster_visualization.py`
4. Check output: Verify PNG file created
5. Verify display: Confirm plots appear

Expected Behavior:

- Program runs without errors
- Console shows progress messages
- Two plots display in window
- File `cluster_visualization.png` created
- Execution time < 10 seconds

15.4 Deployment Checklist

Pre-Release:

- ☐ Code reviewed and approved
- ☐ All assertions pass
- ☐ Tested on Windows, Mac, Linux
- ☐ Documentation complete
- ☐ requirements.txt verified
- ☐ README.md finalized
- ☐ License file added
- ☐ Version number updated

Release:

- [ ] Git tag created (v1.0.0)
- [ ] GitHub release published
- [ ] Release notes written
- [ ] Sample output image uploaded
- [ ] Repository description updated

Post-Release:

- [ ] Monitor for issues
- [ ] Respond to user questions
- [ ] Collect feedback
- [ ] Plan next version

---

16. Maintenance Plan

16.1 Regular Maintenance

Monthly:

- Check for dependency updates
- Review and respond to issues
- Update documentation if needed
- Test with latest Python version

Quarterly:

- Major dependency updates
- Feature enhancements
- Performance optimization
- Security audit

Annually:

- Major version release
- Architecture review
- User survey
- Roadmap update

16.2 Support Plan

Support Channels:

- GitHub Issues: Bug reports and feature requests
- GitHub Discussions: Questions and community support
- Email: Direct support for critical issues
- Documentation: Self-service help

Response SLAs:

- Critical bugs: 24-hour response
- Feature requests: 7-day response
- Questions: 3-day response
- Documentation updates: 14-day turnaround

16.3 Known Limitations

Current Limitations:

1. **Fixed cluster count:** Always generates 3 clusters (k=3)
2. **No user input:** All parameters hardcoded
3. **Single output format:** PNG only, no PDF/SVG
4. **Display required:** Cannot run headless without modification
5. **Fixed overlap:** 2000 points overlap, not configurable
6. **2D only:** No 3D visualization support
7. **Single algorithm:** Only K-Means, no alternatives
8. **No data import:** Generates synthetic data only

Workarounds:

- Modify source code for different parameters
- Use virtual display for headless (Xvfb)
- Convert PNG to other formats externally
- Manually save points to CSV for other uses

---

17. Compliance and Standards

17.1 Code Standards

PEP 8 Compliance:

- Line length: ≤79 characters (relaxed to 100 for readability)
- Indentation: 4 spaces
- Naming: snake_case for variables, UPPER_CASE for constants
- Docstrings: At module and function level
- Comments: Clear, concise, explaining "why" not "what"

Documentation Standards:

- Markdown for all documentation
- Code examples in fenced blocks
- Clear section headers
- Table of contents for long docs

17.2 Licensing

MIT License:

Rationale:

- Permissive license for maximum adoption
- Compatible with academic and commercial use
- Widely recognized and understood
- No copyleft requirements

17.3 Data Privacy

No Personal Data:

- Program generates synthetic data only
- No user input collected
- No telemetry or analytics
- No network communication
- No data stored externally

GDPR Compliance:

- Not applicable (no personal data)
- No cookies or tracking
- No user identification

---

18. Glossary

18.1 Technical Terms

Term	Definition
Cluster	A group of data points with similar characteristics
K-Means	Unsupervised learning algorithm that partitions data into k clusters
Normal Distribution	Gaussian probability distribution characterized by mean and standard deviation
Standard Deviation (σ)	Measure of spread in data; square root of variance
Overlap Points	Points that could belong to multiple clusters based on proximity
Decision Boundary	Line/surface separating different cluster regions
Centroid	Center point of a cluster (mean of all points)
2σ Ellipse	Ellipse covering ~95% of normally distributed data
Contour	Line connecting points of equal value

18.2 Algorithm Terms

Term	Definition
`n_clusters`	Number of clusters (k) to form
`n_init`	Number of times K-Means runs with different centroid seeds
`random_state`	Seed for random number generator (for reproducibility)
`fit_predict`	Train K-Means model and predict cluster labels
`cluster_centers_`	Coordinates of cluster centroids after fitting

18.3 Visualization Terms

Term	Definition
`DPI`	Dots per inch; measure of image resolution
`Alpha`	Transparency level (0=transparent, 1=opaque)
`Z-order`	Layer order in plot (higher = on top)
`Aspect Ratio`	Ratio of width to height; "equal" means 1:1
`Mesh Grid`	Regular grid of points for continuous visualization
`Bounding Box`	Rectangle enclosing all visual elements

---

19. References

19.1 Academic References

1. **K-Means Algorithm:**
   - Lloyd, S. P. (1982). "Least squares quantization in PCM". IEEE Transactions on Information Theory.
   - MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations".
2. **Clustering Evaluation:**
   - Rousseeuw, P. J. (1987). "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis".
   - Dunn, J. C. (1974). "Well-separated clusters and optimal fuzzy partitions".
3. **Normal Distribution:**
   - Gauss, C. F. (1809). "Theoria motus corporum coelestium".
   - Box, G. E. P., & Muller, M. E. (1958). "A Note on the Generation of Random Normal Deviates".

19.2 Technical Documentation

- **NumPy Documentation:** https://numpy.org/doc/
- **Matplotlib Documentation:** https://matplotlib.org/stable/contents.html
- **scikit-learn Documentation:** https://scikit-learn.org/stable/
- **Python Documentation:** https://docs.python.org/3/

19.3 Related Work

- **Scikit-learn Examples:** Clustering demonstrations
- **Matplotlib Gallery:** Visualization examples
- **Seaborn:** Alternative visualization library
- **Plotly:** Interactive visualization alternative

---

20. Approval and Sign-off

20.1 Stakeholder Approval

Role	Name	Signature	Date	Status
Author	Yair Levi	_____	Oct 15, 2025	Approved
Technical Reviewer	_____	_____	_____	Pending
Educational Reviewer	_____	_____	_____	Pending
Quality Assurance	_____	_____	_____	Pending
Project Manager	_____	_____	_____	Pending

20.2 Review History

Version	Date	Reviewer	Changes	Status
0.1	Oct 10, 2025	Yair Levi	Initial draft	Draft
0.5	Oct 12, 2025	Yair Levi	Added technical specs	Review
0.8	Oct 14, 2025	Yair Levi	Added use cases	Review
1.0	Oct 15, 2025	Yair Levi	Final version	Approved

---

21. Revision History

Version	Date	Author	Changes Summary
0.1	Oct 10, 2025	Yair Levi	Initial requirements gathering
0.2	Oct 11, 2025	Yair Levi	Added functional requirements
0.3	Oct 11, 2025	Yair Levi	Added visualization specifications
0.4	Oct 12, 2025	Yair Levi	Added technical specifications
0.5	Oct 12, 2025	Yair Levi	Added algorithm details
0.6	Oct 13, 2025	Yair Levi	Added use cases and constraints
0.7	Oct 14, 2025	Yair Levi	Added testing requirements
0.8	Oct 14, 2025	Yair Levi	Added deployment plan
0.9	Oct 15, 2025	Yair Levi	Added maintenance and support
1.0	Oct 15, 2025	Yair Levi	Final review and approval

Appendix A: Mathematical Formulas

A.1 Normal Distribution

Probability Density Function:

$$f(x \mid \mu, \sigma^2) = (1 / \sqrt{(2\pi\sigma^2)}) \times e^{(-(x-\mu)^2/(2\sigma^2))}$$

Where:
- μ = mean
- σ^2 = variance
- σ = standard deviation
- x = random variable

2-Sigma Rule:

- $\mu \pm 1\sigma$ covers ~68% of data
- $\mu \pm 2\sigma$ covers ~95% of data
- $\mu \pm 3\sigma$ covers ~99.7% of data

A.2 K-Means Algorithm

Objective Function:

$$J = \Sigma(i=1 \text{ to } k) \Sigma(x \in Ci) ||x - \mu i||^2$$

Minimize: Sum of squared distances from points to cluster centers

Where:
- k = number of clusters
- Ci = cluster i
- μi = centroid of cluster i
- $||\cdot||$ = Euclidean distance

Algorithm Steps:

1. Initialize k centroids randomly
2. Repeat until convergence:
   a. Assign each point to nearest centroid
   b. Recalculate centroids as mean of assigned points
3. Return final centroids and assignments

A.3 Euclidean Distance

2D Distance Formula:

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

Where:
- (x_1, y_1) = point 1
- (x_2, y_2) = point 2

A.4 Ellipse Parameters

Standard Form:

$$((x - h)^2 / a^2) + ((y - k)^2 / b^2) = 1$$

Where:
- (h, k) = center coordinates
- a = semi-major axis (width/2)
- b = semi-minor axis (height/2)

For 2σ Ellipse:

$$a = 2 \times \sigma x$$
$$b = 2 \times \sigma y$$

Appendix B: Sample Output

B.1 Console Output Example

```
================================================================================
CLUSTER VISUALIZATION WITH NORMAL DISTRIBUTION
Author: Yair Levi
================================================================================

Generating 6000 points total:
  - 2000 points per cluster (3 clusters)
  - 2000 points are overlap (shared across all clusters)
  - Distribution: Each cluster gets 667 overlap + 1333 unique

Cluster Parameters:

Cluster 1 (Red):
  Mean: (2, 2)
  Std Dev: (1.5, 1.0)
  Points: 667 overlap + 1333 unique = 2000 total

Cluster 2 (Green):
  Mean: (8, 3)
  Std Dev: (1.2, 1.8)
  Points: 667 overlap + 1333 unique = 2000 total

Cluster 3 (Blue):
  Mean: (5, 8)
  Std Dev: (2.0, 1.3)
  Points: 666 overlap + 1334 unique = 2000 total

✓ Total points generated: 6000
  - Cluster 1: 2000 points
  - Cluster 2: 2000 points
  - Cluster 3: 2000 points
  - Overlap points distributed across clusters: 2000

================================================================================
CREATING VISUALIZATION
================================================================================

Plot 1: Original clusters with overlap region highlighted...
Drawing cluster boundaries (2σ ellipses)...

Plot 2: K-Means clustering visualization...
Running K-Means algorithm (k=3)...
✓ K-Means clustering complete
```

```
  Cluster centers found at:
    Cluster 1: (2.05, 2.03)
    Cluster 2: (7.98, 3.15)
    Cluster 3: (5.12, 7.89)
  Drawing decision boundaries...

  ✓ Visualization complete

  ================================================================================
  SAVING FIGURE
  ================================================================================

  ✓ Figure saved as: cluster_visualization.png

  ================================================================================
  ANALYSIS COMPLETE
  ================================================================================

  Summary:
    Total points: 6000
    Cluster 1: 2000 points
    Cluster 2: 2000 points
    Cluster 3: 2000 points
    Overlapping: 2000 points (shared across clusters)
    Unique: 4000 points

  Displaying plot...

  ✓ Program execution completed successfully

  Author: Yair Levi
```

B.2 Expected Visualization Description

Plot 1 (Left):

- Red dots: Cluster 1 unique points (lower-left)
- Green dots: Cluster 2 unique points (right)
- Blue dots: Cluster 3 unique points (top-center)
- Yellow stars: Overlap points (distributed across all regions)
- Dashed ellipses: 2σ boundaries for each cluster
- Legend: All elements labeled
- Grid: Light gray background grid

Plot 2 (Right):

- Points colored by K-Means prediction

- Yellow stars: Original overlap points highlighted
- Black X markers: Cluster centers
- Dashed black lines: Decision boundaries
- Shows how K-Means groups the overlapping data

Statistics Box (Lower-left):

- Total Points: 6000
- Points per Cluster: 2000
- Overlap Points: 2000 (33.3%)
- Unique Points: 4000

Appendix C: Configuration Examples

C.1 Increased Overlap

python

```
# Generate more overlap (3000 points)
OVERLAP_POINTS = 3000
OVERLAP_STD = [3.0, 2.5]  # Wider spread
```

Effect:

- More yellow stars in visualization
- Harder clustering problem
- K-Means may struggle more

C.2 Tighter Clusters

python

```
# Reduce standard deviations
cluster_params = [
    {'mean': [2, 2], 'std': [0.8, 0.5], ...},  # Tighter
    {'mean': [8, 3], 'std': [0.6, 0.9], ...},  # Tighter
    {'mean': [5, 8], 'std': [1.0, 0.7], ...},  # Tighter
]
```

Effect:

- More separated clusters
- Easier clustering problem
- K-Means performs better

C.3 Different Cluster Locations

python

```
# Spread clusters further apart
cluster_params = [
    {'mean': [0, 0], 'std': [1.5, 1.0], ...},
    {'mean': [10, 0], 'std': [1.2, 1.8], ...},
    {'mean': [5, 10], 'std': [2.0, 1.3], ...},
]
```

Effect:

- Greater separation
- Less natural overlap
- Clearer cluster boundaries

Appendix D: Troubleshooting Guide

D.1 Common Issues

Issue: "ModuleNotFoundError: No module named 'numpy'"

Cause: Dependencies not installed

Solution:

bash

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

Issue: "AttributeError: 'Axes' object has no attribute 'add_patch'"

Cause: Matplotlib version too old

Solution:

bash

```
pip install --upgrade matplotlib
```

Issue: Plot window doesn't appear

Cause: No display backend configured

Solution (Windows):

python

```
import matplotlib
matplotlib.use('TkAgg')  # Add before other imports
```

Solution (Linux - headless):

bash

```
# Install virtual display
sudo apt-get install xvfb
# Run with virtual display
xvfb-run python cluster_visualization.py
```

Issue: "MemoryError"

Cause: Insufficient RAM

Solution:

- Close other applications
- Reduce point count temporarily
- Use 64-bit Python

Issue: Output image is blurry

Cause: DPI too low

Solution:

python

```
DPI = 300  # Increase if needed
plt.savefig(output_filename, dpi=DPI)
```

Issue: Colors look wrong on display

Cause: Monitor color calibration

Solution:

- Check monitor settings
- Colors should be correct in saved PNG
- Verify with other image viewer

Appendix E: Extension Ideas

E.1 Add More Clusters

python

```python
# Modify to 4 clusters
N_CLUSTERS = 4
cluster_params = [
    # Original 3 clusters
    {...},
    {...},
    {...},
    # New 4th cluster
    {
        'name': 'Cluster 4 (Magenta)',
        'mean': [8, 8],
        'std': [1.5, 1.5],
        'color': 'magenta',
        'alpha': 0.6
    }
]
```

E.2 Add Command-Line Arguments

python

```python
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--clusters', type=int, default=3)
parser.add_argument('--points', type=int, default=6000)
parser.add_argument('--overlap', type=int, default=2000)
parser.add_argument('--output', type=str, default='cluster_visualization.png')
args = parser.parse_args()
```

E.3 Add Interactive Mode

python

```python
from matplotlib.widgets import Slider

# Add sliders for real-time parameter adjustment
# Update plot dynamically
```

E.4 Export Data to CSV

python

```python
import pandas as pd

df = pd.DataFrame(all_points, columns=['X', 'Y'])
df['Original_Cluster'] = all_labels
df['KMeans_Cluster'] = predicted_labels
df['Point_Type'] = point_types
df.to_csv('cluster_data.csv', index=False)
```

Document Control:

- Classification: Public
- Distribution: Open Source Community
- Review Cycle: Quarterly
- Next Review: January 15, 2026
- Document Owner: Yair Levi
- Maintained By: Development Team

End of Document