Code Implementations

Run Clustering PHP

Call Logger	`require_once 'logger.php';`
Convergence Log	`Logger::info("K-Means converged", [ 'iterations' => $iteration + 1, 'k' => $this->k ]);`
Refactored Cluster name generation	`function generateClusterName($avgAge, $avgIncome, $avgPurchase, $domGender, $domRegion) { // 1. Get Categories $ageCat = getAgeCategory($avgAge); $incCat = getIncomeCategory($avgIncome); $spendCat = getSpendingCategory($avgPurchase); // 2. Determine Financial Behavior // Logic: Compare Income vs Spending to determine intent $isHighSpend = ($spendCat == "Active" || $spendCat == "Premium"); $isHighInc = ($incCat == "Affluent" || $incCat == "High-Income"); $adjective = "Standard"; if ($isHighInc && $isHighSpend) { $adjective = "Elite"; // Wealthy & Spends } elseif ($isHighInc && !$isHighSpend) { $adjective = "Calculated"; // Wealthy & Saves } elseif (!$isHighInc && $isHighSpend) { $adjective = "Aspiring"; // Low Income & Spends } else { $adjective = "Thrifty"; // Low Income & Saves } // 3. Determine "Noun" // Handle specific phrasing for grammar $genderTerm = $domGender; if ($domGender === 'Mixed') { $genderTerm = 'Shoppers'; // Fallback for mixed groups`

	`} elseif ($domGender === 'Male') {` `$genderTerm = 'Men';` `} elseif ($domGender === 'Female') {` `$genderTerm = 'Women';` `}` `// 4. Assemble: [Adjective] [Region] [Age] [Gender/Noun]` `return "$adjective $domRegion $ageCat $genderTerm";` `}`
Log Update Database	`Logger::info("Database updated with new cluster results", ['count' => count($labels)]);`
Update database rollback log	`Logger::error("Critical: Database update failed. Transaction rolled back.", [` `'exception' => $e->getMessage()` `]);`

Index PHP

Call Logger	`require_once 'logger.php';`
Log unauthorized access attempts	`Logger::info("Unauthorized access attempt", ['ip' => $_SERVER['REMOTE_ADDR']]);`
Log fetching error	`Logger::error("Query execution failed", [` `'type' => $segmentationType,` `'error' => $e->getMessage(),` `'sql' => $sql // Use carefully in production if SQL contains user input` `]);`
Helper functions for division by 0	`// FIX: Helper function to safely calculate percentage and handle division by zero` `const getPercent = (value) => {` `if (!totalCustomers || totalCustomers === 0) return '0.0';` `return ((value / totalCustomers) * 100).toFixed(1);` `};`
Helper functions for empty arrays	`// FIX: Helper to safe-guard Math.max against empty arrays (which returns -Infinity)` `const maxVal = data.length > 0 ? Math.max(...data) : 0;`

	`// Helper to find the index of the max value safely` `const maxIndex = data.length > 0 ? data.indexOf(maxVal) : -1;`
Helper function for income statistics	`// Pre-calculate income stats safely to avoid repeating code in the HTML string` `let incomeStatsHTML = '';` `if (results.length > 0 && results[0].avg_income) {` `const incomes = results.map(r => parseFloat(r.avg_income || 0));` `const minInc = Math.min(...incomes);` `const maxInc = Math.max(...incomes);` `const gap = maxInc - minInc;` `incomeStatsHTML = ` ` ` - Average income across genders ranges from $${minInc.toLocaleString()} to $${maxInc.toLocaleString()} ` ` - Income gap between genders: $${gap.toLocaleString()} ` `` ` ``; `}`
Helper function for top 3 region	`// Calculate top 3 sum safely` `const top3Sum = (data[0] || 0) + (data[1] || 0) + (data[2] || 0);`
Logic for chart implementations	`// Logic to switch chart types based on segmentation` `if (segmentationType === 'region') {` `chartOptions.indexAxis = 'y'; // Switch to Horizontal` `} else if (segmentationType === 'purchase_tier') {` `// Switch to Polar Area Chart` `chartType = 'polarArea';` `// Distinct colors for tiers` `bgColors = [` `'rgba(255, 99, 132, 0.7)', // Red` `'rgba(255, 205, 86, 0.7)', // Yellow` `'rgba(75, 192, 192, 0.7)'  // Green` `];` `borderColors = '#ffffff'; // White borders look better on Polar` `// Polar specific options` `chartOptions = {`

	responsive: true, plugins: { title: { display: true, text: 'Spending Power Distribution' }, legend: { position: 'right', display: true } }, scales: { r: { ticks: { backdropColor: 'transparent', z: 1 } } } }; } else if (segmentationType === 'age_group' || segmentationType === 'income_bracket') { chartType = 'line'; bgColors = 'rgba(54, 162, 235, 0.2)'; }
Donut chart implementation	// --- 3. Initialize Doughnut Chart (Side Chart) --- const ctx2 = document.getElementById('doughnutChart').getContext('2d'); const doughnutColors = ['rgba(255, 99, 132, 0.8)', 'rgba(54, 162, 235, 0.8)', 'rgba(255, 206, 86, 0.8)', 'rgba(75, 192, 192, 0.8)', 'rgba(153, 102, 255, 0.8)', 'rgba(255, 159, 64, 0.8)']; new Chart(ctx2, { type: 'doughnut', //Code here

Login PHP

Call Logger	require_once 'logger.php';
Log Successful attempt	// Log successful login Logger::info("User logged in successfully", ['username' => $username, 'ip' => $_SERVER['REMOTE_ADDR']]);
Log Failed Attempt	// Log failed attempt Logger::error("Failed login attempt", ['attempted_username' => $username, 'ip' => $_SERVER['REMOTE_ADDR'],

	`'user_agent' => $_SERVER['HTTP_USER_AGENT']` `]);`

Logout PHP

Log User Logout	`// Log the logout before destroying the session if you want to know who left` `if (isset($_SESSION['logged_in'])) {` `Logger::info("User logged out", [` `'ip' => $_SERVER['REMOTE_ADDR']` `]);` `}`

CLV Tier Segmentation PHP

Handle CLV Tier Segmentation	`<?php` `/**` ` * Handle CLV Tier Segmentation` ` */` `case 'clv_tier':` `    // Get tier distribution summary` `    $tierSummaryQuery = "` `        SELECT` `            clv_tier,` `            COUNT(*) as customer_count,` `                ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM customers WHERE clv_tier IS NOT NULL), 2) as percentage,` `            MIN(calculated_clv) as min_clv,` `            MAX(calculated_clv) as max_clv,` `                ROUND(AVG(calculated_clv), 2) as avg_clv,` `            ROUND(AVG(income), 2) as avg_income,` `            ROUND(AVG(age), 2) as avg_age,` `                ROUND(AVG(purchase_amount), 2) as avg_purchase` `        FROM customers` `        WHERE clv_tier IS NOT NULL` `        GROUP BY clv_tier` `            ORDER BY FIELD(clv_tier, 'Platinum', 'Gold', 'Silver', 'Bronze')` `    ";`

```
        $tierSummaryResult = mysqli_query($conn, $tierSummaryQuery);

    if (!$tierSummaryResult) {
        die("Error fetching CLV tier summary: " . mysqli_error($conn));
    }

    // Store tier summary data
    $tierSummary = [];
    while ($row = mysqli_fetch_assoc($tierSummaryResult)) {
        $tierSummary[] = $row;
    }

    // Get detailed customer data by tier
    $customerDataQuery = "
        SELECT
            customer_id,
            name,
            age,
            gender,
            income,
            region,
            purchase_amount,
            avg_purchase_amount,
            purchase_frequency,
            customer_lifespan_months,
            calculated_clv,
            clv_tier
        FROM customers
        WHERE clv_tier IS NOT NULL
        ORDER BY
            FIELD(clv_tier, 'Platinum', 'Gold', 'Silver', 'Bronze'),
            calculated_clv DESC
    ";

        $customerDataResult = mysqli_query($conn, $customerDataQuery);

    if (!$customerDataResult) {
```

	`` die("Error fetching customer data: " . mysqli_error($conn)); } // Store customer data grouped by tier $customersByTier = ['Platinum' => [], 'Gold' => [], 'Silver' => [], 'Bronze' => []]; while ($row = mysqli_fetch_assoc($customerDataResult)) { $customersByTier[$row['clv_tier']][] = $row; } // Prepare data for export $segmentationData = ['type' => 'clv_tier', 'summary' => $tierSummary, 'customers_by_tier' => $customersByTier, 'total_customers' => array_sum(array_column($tierSummary, 'customer_count'))]; // Display results or prepare for view include 'views/clv_tier_results.php'; break; ?> ```

CLV Insights JS

Handles CLV Segmentation Insights	``` <script> const segmentationType = '<?= $segmentationType ?>'; const labels = <?= json_encode(array_column($results, array_keys($results[0])[0])) ?>; ```

	const data = <?= json_encode(array_column($results, array_keys($results[0])[1])) ?>; const results = <?= json_encode($results) ?>; // Generate insights based on segmentation type let insights = ''; const totalCustomers = data.reduce((a, b) => a + b, 0); switch (segmentationType) { case 'gender': insights = ` - Total customers analyzed: ${totalCustomers.toLocaleString()} - Gender distribution shows ${labels.length} categories ……

Unit Test Cases PHP

Unit Test Cases for normalizeData()	<?php class KMeansClusteringTest extends PHPUnit\Framework\TestCase { public function testNormalizeDataNormalDataset() { $kmeans = new KMeansClustering(); $data = [['age' => 25, 'income' => 50000, 'purchase_amount' => 1000], ['age' => 35, 'income' => 60000, 'purchase_amount' => 2000], ['age' => 45, 'income' => 70000, 'purchase_amount' => 3000]]; $normalized = $kmeans->normalizeData($data); // Verify output structure

```
        $this->assertCount(3, $normalized);
                    $this->assertArrayHasKey('age',
$normalized[0]);

          // Verify z-score properties (mean ≈ 0,
std ≈ 1 for each feature)
        $ages = array_column($normalized, 'age');
                    $this->assertEqualsWithDelta(0,
array_sum($ages)/count($ages), 0.1);
    }

                                public       function
testNormalizeDataZeroStandardDeviation() {
        $kmeans = new KMeansClustering();
        $data = [
                  ['age' => 30, 'income' => 50000,
'purchase_amount' => 1000],
                  ['age' => 30, 'income' => 50000,
'purchase_amount' => 1000],
                  ['age' => 30, 'income' => 50000,
'purchase_amount' => 1000]
        ];

                                      $normalized    =
$kmeans->normalizeData($data);

          // Should handle zero variance by using
divisor of 1
                                $this->assertEquals(0,
$normalized[0]['age']); // (30-30)/1 = 0
                                $this->assertEquals(0,
$normalized[0]['income']);
                                $this->assertEquals(0,
$normalized[0]['purchase_amount']);
    }

                                public       function
testNormalizeDataNegativeValues() {
        $kmeans = new KMeansClustering();
        $data = [
                  ['age' => -5, 'income' => -1000,
'purchase_amount' => -100],
                   ['age' => 5, 'income' => 1000,
'purchase_amount' => 100]
```

	``]; $normalized = $kmeans->normalizeData($data); // Verify negative values are handled correctly $this->assertIsFloat($normalized[0]['age']); $this->assertIsFloat($normalized[1]['age']); // Check that mean is approximately 0 $ages = array_column($normalized, 'age'); $this->assertEqualsWithDelta(0, array_sum($ages)/count($ages), 0.01); } public function testNormalizeDataEmptyDataset() { $kmeans = new KMeansClustering(); $data = []; $this->expectException(\Exception::class); $normalized = $kmeans->normalizeData($data); } } ```
Unit Test Cases for euclideanDistance()	``` <?php class KMeansClusteringTest extends PHPUnit\Framework\TestCase { public function testEuclideanDistanceSamePoint() { $kmeans = new KMeansClustering(); $point1 = ['age' => 30, 'income' => 50000, 'purchase_amount' => 1000]; $point2 = ['age' => 30, 'income' => 50000, 'purchase_amount' => 1000]; // Use reflection to access private method $reflection = new \ReflectionClass($kmeans); ```

```
                                        $method     =
$reflection->getMethod('euclideanDistance');
        $method->setAccessible(true);

             $distance  =  $method->invoke($kmeans,
$point1, $point2);
        $this->assertEquals(0.0, $distance);
    }

                                public       function
testEuclideanDistanceDifferentPoints() {
        $kmeans = new KMeansClustering();
         $point1 = ['age' => 20, 'income' => 30000,
'purchase_amount' => 500];
         $point2 = ['age' => 40, 'income' => 70000,
'purchase_amount' => 2500];

                               $reflection    =    new
\ReflectionClass($kmeans);
                                        $method     =
$reflection->getMethod('euclideanDistance');
        $method->setAccessible(true);

             $distance  =  $method->invoke($kmeans,
$point1, $point2);

                  //  Expected:  sqrt((20-40)^2  +
(30000-70000)^2 + (500-2500)^2)
          // = sqrt(400 + 1600000000 + 4000000) =
sqrt(1600400400)
          $expected = sqrt(pow(20, 2) + pow(40000,
2) + pow(2000, 2));
           $this->assertEqualsWithDelta($expected,
$distance, 0.001);
    }

                                public       function
testEuclideanDistanceMissingFeatures() {
        $kmeans = new KMeansClustering();
             $point1  =  ['age'  =>  30,  'income'  =>
50000]; // Missing purchase_amount
         $point2 = ['age' => 35, 'income' => 55000,
'purchase_amount' => 1500];
```

	`$reflection = new \ReflectionClass($kmeans);` `$method = $reflection->getMethod('euclideanDistance');` `$method->setAccessible(true);` `$this->expectException(\ErrorException::class);` `$distance = $method->invoke($kmeans, $point1, $point2);` `}` `}`
Deterministic Testing Approach	`<?php` `class KMeansClusteringTest extends PHPUnit\Framework\TestCase {` `public function testKMeansPlusPlusInitializationDeterministic() {` `// Set a fixed seed to make randomness deterministic` `srand(42); // Same seed as production code` `$kmeans = new KMeansClustering(3);` `$data = [` `['age' => 25, 'income' => 40000, 'purchase_amount' => 800],` `['age' => 35, 'income' => 60000, 'purchase_amount' => 1500],` `['age' => 45, 'income' => 80000, 'purchase_amount' => 2200],` `['age' => 55, 'income' => 100000, 'purchase_amount' => 3000],` `['age' => 65, 'income' => 120000, 'purchase_amount' => 3500]` `];` `// Use reflection to access private method` `$reflection = new \ReflectionClass($kmeans);` `$method = $reflection->getMethod('initializeCentroids');` `$method->setAccessible(true);`

	`$centroids = $method->invoke($kmeans, $data);` `// Verify correct number of centroids` `$this->assertCount(3, $centroids);` `// Verify centroids are from original data points` `foreach ($centroids as $centroid) {` `$found = false;` `foreach ($data as $point) {` `if ($point['age'] == $centroid['age'] &&` `$point['income'] == $centroid['income'] &&` `$point['purchase_amount'] == $centroid['purchase_amount']) {` `$found = true;` `break;` `}` `}` `$this->assertTrue($found, "Centroid not found in original data");` `}` `// Test reproducibility with same seed` `srand(42); // Reset seed` `$centroids2 = $method->invoke($kmeans, $data);` `$this->assertEquals($centroids, $centroids2);` `}` `}`
Cluster Segmentation Data Flow	`<?php` `class SegmentationIntegrationTest extends PHPUnit\Framework\TestCase {` `public function testClusterSegmentationDataFlow() {` `// Setup test database with known data` `$this->setupTestDatabase();` `// Execute clustering`

```
        $kmeans = new KMeansClustering(3);
                                    $customerData    =
$this->getTestCustomerData();
        $labels = $kmeans->fit($customerData);

        // Verify clustering results
          $this->assertCount(100, $labels); // All
customers labeled
          $this->assertContains(0, $labels); // At
least one cluster 0
          $this->assertContains(1, $labels); // At
least one cluster 1
          $this->assertContains(2, $labels); // At
least one cluster 2

        // Verify cluster metadata generation
                                          $metadata    =
$this->generateClusterMetadata($labels,
$customerData);
        $this->assertCount(3, $metadata);

        foreach ($metadata as $cluster) {

$this->assertArrayHasKey('cluster_name',
$cluster);
                $this->assertArrayHasKey('avg_age',
$cluster);
                          $this->assertGreaterThan(0,
$cluster['customer_count']);
        }
    }

                                public       function
testMetadataVisualizationIntegration() {
             // Test that metadata is correctly
formatted for Chart.js
                                          $metadata    =
$this->getSampleClusterMetadata();

                                         $chartData    =
$this->formatChartData($metadata);

                 $this->assertArrayHasKey('labels',
$chartData);
```

```
                $this->assertArrayHasKey('datasets',
$chartData);
                                $this->assertCount(3,
$chartData['datasets'][0]['data']); // 3 clusters

        // Verify data integrity
                                 $totalCustomers    =
array_sum($chartData['datasets'][0]['data']);
        $this->assertEquals(100, $totalCustomers);
    }
}
```