LAMPIRAN

a. *Source Code* Halaman *Dashboard*

```
class DatasetController extends Controller
{
    // Display dataset
    public function index(Request $request)
    {
        // Get the unique months from the 'tanggal' column
        $months = Dataset::selectRaw('DISTINCT DATE_FORMAT(tanggal, "%Y-%m") as month')->get();
        // Get the datasets based on the filter (if any)
        $datasetsQuery = Dataset::query();
        // Filter by month if provided
        if ($request->has('month') && $request->input('month') != '') {
            $monthFilter = $request->input('month');
            $datasetsQuery->whereMonth('tanggal', '=', Carbon::parse($monthFilter)->month)
                ->whereYear('tanggal', '=', Carbon::parse($monthFilter)->year);
        }
        // Fetch the datasets
        $datasets = $datasetsQuery->orderBy('tanggal')->get();
        // Group datasets by month
        $groupedDatasets = $datasets->groupBy(function ($dataset) {
            return Carbon::parse($dataset->tanggal)->format('Y-m');
        });
        // Format the date and day in Indonesian
        foreach ($datasets as $dataset) {
            $carbonDate = Carbon::parse($dataset->tanggal)->locale('id');
            $dataset->tanggal = $carbonDate->isoFormat('D MMMM YYYY');
            $dataset->hari = $carbonDate->isoFormat('dddd');
        }
        return view('pages.dataset.index', compact('groupedDatasets', 'months'));
```

```
}


// Store new dataset
public function store(Request $request)

{

    $carbonDate = Carbon::parse($request->tanggal)->locale('id');

    $dataset = new Dataset();
    $dataset->tanggal = $carbonDate->format('Y-m-d'); // Store in 'YYYY-MM-DD'
    $dataset->hari = $carbonDate->isoFormat('dddd');  // Store the day in Indonesian
    $dataset->datang = $request->datang;
    $dataset->berangkat = $request->berangkat;
    $dataset->save();

    return redirect()->route('dataset.index')->with('success', 'Dataset created successfully!');
}
// Import Excel
public function import(Request $request)
{
    $request->validate([
        'file' => 'required|mimes:xlsx,csv',
    ]);
    Dataset::truncate();
    // Import the Excel file
    Excel::import(new DatasetImport, $request->file('file'));
    return redirect()->route('dataset.index')->with('success', 'Datasets imported successfully!');
}
// DatasetController
public function deleteAll()
{
    Dataset::truncate(); // This will delete all rows from the `datasets` table
    return redirect()->route('dataset.index')->with('success', 'All datasets deleted successfully!');
}
public function update(Request $request, $id)
{
    $dataset = Dataset::find($id);
    $carbonDate = Carbon::parse($request->tanggal)->locale('id');
    $dataset->tanggal = $carbonDate->format('Y-m-d'); // Store in 'YYYY-MM-DD'
    $dataset->hari = $carbonDate->isoFormat('dddd');  // Store the day in Indonesian
    $dataset->datang = $request->datang;
    $dataset->berangkat = $request->berangkat;
    $dataset->save();
    return redirect()->route('dataset.index')->with('success', 'Dataset updated successfully!');
}
public function destroy($id)
{
    $dataset = Dataset::find($id);
    $dataset->delete();
    return redirect()->route('dataset.index')->with('success', 'Dataset deleted successfully!');
}
                }
```

b. *Source Code* Halaman Dataset

```
class DatasetController extends Controller
```

```
{
// Display dataset
public function index(Request $request)
{
// Get the unique months from the 'tanggal' column
$months = Dataset::selectRaw('DISTINCT DATE_FORMAT(tanggal, "%Y-%m") as month')->get();
// Get the datasets based on the filter (if any)
$datasetsQuery = Dataset::query();
// Filter by month if provided
if ($request->has('month') && $request->input('month') != '') {
$monthFilter = $request->input('month');
$datasetsQuery->whereMonth('tanggal', '=', Carbon::parse($monthFilter)->month)
->whereYear('tanggal', '=', Carbon::parse($monthFilter)->year);
}
// Fetch the datasets
$datasets = $datasetsQuery->orderBy('tanggal')->get();
// Group datasets by month
$groupedDatasets = $datasets->groupBy(function ($dataset) {
return Carbon::parse($dataset->tanggal)->format('Y-m');
});
// Format the date and day in Indonesian
foreach ($datasets as $dataset) {
$carbonDate = Carbon::parse($dataset->tanggal)->locale('id');
$dataset->tanggal = $carbonDate->isoFormat('D MMMM YYYY');
$dataset->hari = $carbonDate->isoFormat('dddd');
}
return view('pages.dataset.index', compact('groupedDatasets', 'months'));
}
// Store new dataset
public function store(Request $request)
{
$carbonDate = Carbon::parse($request->tanggal)->locale('id');
$dataset = new Dataset();
$dataset->tanggal = $carbonDate->format('Y-m-d'); // Store in 'YYYY-MM-DD'
$dataset->hari = $carbonDate->isoFormat('dddd'); // Store the day in Indonesian
```

```
$dataset->datang = $request->datang;
$dataset->berangkat = $request->berangkat;
$dataset->save();
return redirect()->route('dataset.index')->with('success', 'Dataset created successfully!');
}
// Import Excel
public function import(Request $request)
{
$request->validate([
'file' => 'required|mimes:xlsx,csv',
]);
Dataset::truncate();
// Import the Excel file
Excel::import(new DatasetImport, $request->file('file'));
return redirect()->route('dataset.index')->with('success', 'Datasets imported successfully!');
}
// DatasetController
public function deleteAll()
{
Dataset::truncate(); // This will delete all rows from the `datasets` table
return redirect()->route('dataset.index')->with('success', 'All datasets deleted successfully!');
}
public function update(Request $request, $id)
{
$dataset = Dataset::find($id);
$carbonDate = Carbon::parse($request->tanggal)->locale('id');
$dataset->tanggal = $carbonDate->format('Y-m-d'); // Store in 'YYYY-MM-DD'
$dataset->hari = $carbonDate->isoFormat('dddd'); // Store the day in Indonesian
$dataset->datang = $request->datang;
$dataset->berangkat = $request->berangkat;
$dataset->save();
return redirect()->route('dataset.index')->with('success', 'Dataset updated successfully!');
}
public function destroy($id)
{
$dataset = Dataset::find($id);
```

```
$dataset->delete();
return redirect()->route('dataset.index')->with('success', 'Dataset deleted successfully!');
}
}
```

c. *Source Code* Halaman *Monte Carlo* Datang

```
class DatangController extends Controller
{
    public function index(Request $request)
    {
        $datangData = Dataset::select('datang', 'tanggal')->get();
        $datangData = $datangData->sortBy('datang');  // Sort by 'datang' in descending order
        $groupedDatasets = collect();
        $rangeMapping = [];
        $monthlyResults = [];

        // Buat dataset terpisah khusus Acuan Prediksi
        $datangDataForAcuan = $datangData->filter(function ($item) {
            $monthNum = intval(Carbon::parse($item->tanggal)->format('m'));
            return $monthNum >= 1 && $monthNum <= 11;
        });

        if (!$datangData->isEmpty()) {
            // Sort datangData to ensure datang is ordered from 0 upwards
            $datangData = $datangData->sortBy('datang');  // Sorting by 'datang' in ascending order

            $frequencies = $datangDataForAcuan->groupBy('datang')->map(fn($group) => $group->count());

            $total = $frequencies->sum();
            $cumulative = 0;
            $previousMax = 0;

            foreach ($frequencies as $value => $count) {
                $probability = $count / $total;
```

```
            $cumulative += $probability;


            // Calculate the minimum and maximum for the range
            $min = $previousMax;


            // Apply rundown: round down the cumulative probability multiplied by 100
            $max = (round($cumulative, 4) == 1.0000) ? 100 : floor($cumulative * 100);  // Rundown to the nearest integer


            // Ensure the range does not exceed 100
            if ($max > 100) {
                $max = 100;
            }


            // Update previousMax for the next iteration
            $previousMax = $max + 1;


            // Push the calculated values into the grouped dataset
            $groupedDatasets->push([
                'datang' => $value,
                'frekuensi' => $count,
                'probabilitas' => round($probability, 4),
                'komulatif' => round($cumulative, 4),
                'range' => $min . ' - ' . $max,
            ]);


            // Add the range data for future reference
            $rangeMapping[] = [
                'min' => $min,
                'max' => $max,
                'datang' => $value,
            ];
        }


        // Group data by month, sort the keys (months) in ascending order
            $groupedByMonth = $datangData->groupBy(fn($dataset) => Carbon::parse($dataset->tanggal)->format('Y-m'))->sortKeys();
```

```
foreach ($groupedByMonth as $month => $dailyData) {
    $simulasiPerMonth = [];
    $randomNumbersPerMonth = [];
    $apePerMonth = [];
    $comparisonPerMonth = [];

    foreach ($dailyData as $dayData) {
        $dailyRandomNumbers = [];
        $dailySimulation = [];
        $dailyAkurasi = [];
        $dailyAPE = [];

        for ($i = 0; $i < 5; $i++) {
            $randomValue = rand(0, 100);
            $dailyRandomNumbers[] = $randomValue;

            foreach ($rangeMapping as $range) {
                if ($randomValue >= $range['min'] && $randomValue <= $range['max']) {
                    $dailySimulation[] = $range['datang'];
                    break;
                }
            }
        }

        $actualValue = $dayData->datang;

        foreach ($dailySimulation as $sim) {
            // Calculate error using absolute difference between simulation and actual value
            $error = abs($sim - $actualValue);

            // Correct accuracy formula: MIN(predicted, actual) / MAX(predicted, actual) * 100
            $maxVal = max($sim, $actualValue);
            $accuracy = ($maxVal == 0) ? 0 : min($sim, $actualValue) / $maxVal;

            // Calculate absolute percentage error (APE)
            $ape = ($actualValue != 0)
                ? abs(($sim - $actualValue) / $actualValue)
```

```
                    : 0;

                // Add calculated accuracy and APE to respective arrays
                $dailyAkurasi[] = $accuracy;
                $dailyAPE[] = $ape;

                // Add APE to global array for monthly calculation
                $apePerMonth[] = $ape;
            }

            $comparisonPerMonth[] = [
                'random_numbers' => $dailyRandomNumbers,
                'simulations' => $dailySimulation,
                'accuracies' => $dailyAkurasi,
                'apes' => $dailyAPE,
                'actual' => $actualValue,
            ];

            $randomNumbersPerMonth[] = $dailyRandomNumbers;
            $simulasiPerMonth[] = $dailySimulation;
        }

        // Calculate average accuracy for each simulation column
        $numRows = count($comparisonPerMonth);
        $sumAccuracies = array_fill(0, 5, 0);
        $bestPredictions = [];

        foreach ($comparisonPerMonth as $row) {
            foreach ($row['accuracies'] as $i => $acc) {
                $sumAccuracies[$i] += $acc;
            }
        }

            $avgAccuracies = array_map(fn($sum) => $numRows > 0 ? $sum / $numRows : 0,
$sumAccuracies);
        $bestSimulationIndex = array_keys($avgAccuracies, max($avgAccuracies))[0];

        // Save the best prediction per row
```

```
            foreach ($comparisonPerMonth as $row) {
                $bestPredictions[] = $row['simulations'][$bestSimulationIndex];
            }

            $mapePerMonth = $this->calculateMape($apePerMonth);
            $accuracyPerMonth = 100 - $mapePerMonth;

            $monthlyResults[$month] = [
                'simulasi' => $simulasiPerMonth,
                'random_numbers' => $randomNumbersPerMonth,
                'comparison' => $comparisonPerMonth,
                'ape' => $apePerMonth,
                'mape' => $mapePerMonth,
                'accuracy' => $accuracyPerMonth,
                'best_simulation_index' => $bestSimulationIndex,
                'best_predictions' => $bestPredictions,
                'best_simulation_avg_accuracy' => $avgAccuracies[$bestSimulationIndex],
            ];
        }

        // Sort months in ascending order (from January to December)
        ksort($monthlyResults); // Ensure months are ordered from January to December
    }

    // Get selected month from request
    $selectedMonth = $request->input('month', null);
    $selectedMonthResults = [];

    if ($selectedMonth && isset($monthlyResults[$selectedMonth])) {
        $selectedMonthResults = $monthlyResults[$selectedMonth];
    }
    // In DatangController, store best prediction for December
    $desemberKeyDatang = collect($monthlyResults)->keys()->filter(function ($key) {
        return \Carbon\Carbon::parse($key)->month === 12;
    })->first();

    if ($desemberKeyDatang && isset($monthlyResults[$desemberKeyDatang]['best_predictions'])) {
```

```
            // Store the 'datang' forecast for December
                                                    session(['montecarlo_forecast_datang'          =>
$monthlyResults[$desemberKeyDatang]['best_predictions']]);
        }

        // Save or update the AkurasiMape record for Datang (ID = 1)
        $akurasiMape = AkurasiMape::find(1);

        if ($akurasiMape) {
            // If the row exists, update the values for monte_akurasi_datang and monte_mape_datang
            $akurasiMape->update([
                'monte_akurasi_datang' => $selectedMonthResults['accuracy'] ?? 0,
                'monte_mape_datang' => $selectedMonthResults['mape'] ?? 0,
            ]);
        } else {
            // If the row does not exist, create a new row
            AkurasiMape::create([
                'monte_akurasi_datang' => $selectedMonthResults['accuracy'] ?? 0,
                'monte_mape_datang' => $selectedMonthResults['mape'] ?? 0,
            ]);
        }

        // dd($selectedMonthResults); // This is where you can check the values

        return view('pages.monte-carlo.datang.index', compact(
            'groupedDatasets',
            'monthlyResults',
            'selectedMonthResults',
            'selectedMonth'
        ));
    }

    private function calculateMape($apeResults)
    {
        if (is_array($apeResults)) {
            $totalApe = array_sum($apeResults);
            $count = count($apeResults);
        } else {
```

```
            $totalApe = $apeResults;
            $count = 1;
        }

        return ($count > 0) ? $totalApe / $count : 0;
    }
}
```

d. *Source Code* Halaman *Monte Carlo* Berangkat

```
class BerangkatController extends Controller
{
    public function index(Request $request)
    {
        $berangkatData = Dataset::select('berangkat', 'tanggal')->get();
        $groupedDatasets = collect();
        $rangeMapping = [];
        $monthlyResults = [];

        // Buat dataset terpisah khusus Acuan Prediksi
        $berangkatDataForAcuan = $berangkatData->filter(function ($item) {
            $monthNum = intval(Carbon::parse($item->tanggal)->format('m'));
            return $monthNum >= 1 && $monthNum <= 11;
        });

        if (!$berangkatData->isEmpty()) {
            // Sort berangkatData to ensure berangkat is ordered from 0 upwards
            $berangkatData = $berangkatData->sortBy('berangkat'); // Sorting by 'berangkat' in ascending order
            $berangkatDataForAcuan = $berangkatData->filter(function ($item) {
                $monthNum = intval(Carbon::parse($item->tanggal)->format('m'));
                return $monthNum >= 1 && $monthNum <= 11;
            });

            $frequencies = $berangkatDataForAcuan->groupBy('berangkat')->map(fn($group) => $group->count());

            $total = $frequencies->sum();
```

```
        $cumulative = 0;
        $previousMax = 0;

        foreach ($frequencies as $value => $count) {
            $probability = $count / $total;
            $cumulative += $probability;

            // Calculate the minimum and maximum for the range
            $min = $previousMax;

            // Apply rounding down (rundown) to the cumulative probability multiplied by 100
            $max = (round($cumulative, 4) == 1.0000) ? 100 : floor(($cumulative * 100));  // Rundown to the nearest integer

            // Update previousMax for the next iteration
            $previousMax = $max + 1;

            // Push the calculated values into the grouped dataset
            $groupedDatasets->push([
                'berangkat' => $value,
                'frekuensi' => $count,
                'probabilitas' => round($probability, 4),
                'komulatif' => round($cumulative, 4),
                'range' => $min . ' - ' . $max,
            ]);

            // Add the range data for future reference
            $rangeMapping[] = [
                'min' => $min,
                'max' => $max,
                'berangkat' => $value,
            ];
        }

        // Group data by month, sort the keys (months) in ascending order
        $groupedByMonth = $berangkatData->groupBy(fn($dataset) => Carbon::parse($dataset->tanggal)->format('Y-m'))->sortKeys();
```

```
foreach ($groupedByMonth as $month => $dailyData) {
    $simulasiPerMonth = [];
    $randomNumbersPerMonth = [];
    $apePerMonth = [];
    $comparisonPerMonth = [];

    foreach ($dailyData as $dayData) {
        $dailyRandomNumbers = [];
        $dailySimulation = [];
        $dailyAkurasi = [];
        $dailyAPE = [];

        for ($i = 0; $i < 5; $i++) {
            $randomValue = rand(0, 100);
            $dailyRandomNumbers[] = $randomValue;

            foreach ($rangeMapping as $range) {
                if ($randomValue >= $range['min'] && $randomValue <= $range['max']) {
                    $dailySimulation[] = $range['berangkat'];
                    break;
                }
            }
        }

        $actualValue = $dayData->berangkat;

        foreach ($dailySimulation as $sim) {
            // Calculate error using absolute difference between simulation and actual value
            $error = abs($sim - $actualValue);

            // Correct accuracy formula: MIN(predicted, actual) / MAX(predicted, actual) * 100
            $maxVal = max($sim, $actualValue);
            $accuracy = ($maxVal == 0) ? 0 : min($sim, $actualValue) / $maxVal;
            // Calculate absolute percentage error (APE)
            $ape = ($actualValue != 0)
                ? abs(($sim - $actualValue) / $actualValue)
                : 0;
```

```
                // Add calculated accuracy and APE to respective arrays
                $dailyAkurasi[] = $accuracy;
                $dailyAPE[] = $ape;

                // Add APE to global array for monthly calculation
                $apePerMonth[] = $ape;
            }

            $comparisonPerMonth[] = [
                'random_numbers' => $dailyRandomNumbers,
                'simulations' => $dailySimulation,
                'accuracies' => $dailyAkurasi,
                'apes' => $dailyAPE,
                'actual' => $actualValue,
            ];

            $randomNumbersPerMonth[] = $dailyRandomNumbers;
            $simulasiPerMonth[] = $dailySimulation;
        }

        // Calculate average accuracy for each simulation column
        $numRows = count($comparisonPerMonth);
        $sumAccuracies = array_fill(0, 5, 0);
        $bestPredictions = [];

        foreach ($comparisonPerMonth as $row) {
            foreach ($row['accuracies'] as $i => $acc) {
                $sumAccuracies[$i] += $acc;
            }
        }

        $avgAccuracies = array_map(fn($sum) => $numRows > 0 ? $sum / $numRows : 0, $sumAccuracies);
        $bestSimulationIndex = array_keys($avgAccuracies, max($avgAccuracies))[0];

        // Save the best prediction per row
        foreach ($comparisonPerMonth as $row) {
```

```
                $bestPredictions[] = $row['simulations'][$bestSimulationIndex];
            }

            $mapePerMonth = $this->calculateMape($apePerMonth);
            $accuracyPerMonth = 100 - $mapePerMonth;

            $monthlyResults[$month] = [
                'simulasi' => $simulasiPerMonth,
                'random_numbers' => $randomNumbersPerMonth,
                'comparison' => $comparisonPerMonth,
                'ape' => $apePerMonth,
                'mape' => $mapePerMonth,
                'accuracy' => $accuracyPerMonth,
                'best_simulation_index' => $bestSimulationIndex,
                'best_predictions' => $bestPredictions,
                'best_simulation_avg_accuracy' => $avgAccuracies[$bestSimulationIndex],
            ];
        }

        // Sort months in ascending order (from January to December)
        ksort($monthlyResults); // Ensure months are ordered from January to December
    }

    // Get selected month from request
    $selectedMonth = $request->input('month', null);
    $selectedMonthResults = [];

    if ($selectedMonth && isset($monthlyResults[$selectedMonth])) {
        $selectedMonthResults = $monthlyResults[$selectedMonth];
    }

    // Simpan prediksi terbaik Desember untuk 'berangkat' ke session
    $desemberKeyBerangkat = collect($monthlyResults)->keys()->filter(function ($key) {
        return \Carbon\Carbon::parse($key)->month === 12;
    })->first();

    if ($desemberKeyBerangkat && isset($monthlyResults[$desemberKeyBerangkat]['best_predictions'])) {
```

```
        // Store the 'berangkat' forecast for December
                                                session(['montecarlo_forecast_berangkat' =>
$monthlyResults[$desemberKeyBerangkat]['best_predictions']]);
    }


    // Save or update the AkurasiMape record for Berangkat (ID = 1)
    $akurasiMape = AkurasiMape::find(1);


    if ($akurasiMape) {
            // If the row exists, update the values for monte_akurasi_berangkat and
monte_mape_berangkat
        $akurasiMape->update([
          'monte_akurasi_berangkat' => $selectedMonthResults['accuracy'] ?? 0,
          'monte_mape_berangkat' => $selectedMonthResults['mape'] ?? 0,
        ]);
    } else {
        // If the row does not exist, create a new row
        AkurasiMape::create([
          'monte_akurasi_berangkat' => $selectedMonthResults['accuracy'] ?? 0,
          'monte_mape_berangkat' => $selectedMonthResults['mape'] ?? 0,
        ]);
    }
    // dd($selectedMonthResults);


    return view('pages.monte-carlo.berangkat.index', compact(
        'groupedDatasets',
        'monthlyResults',
        'selectedMonthResults',
        'selectedMonth'
    ));
  }


  private function calculateMape($apeResults)
  {
    if (is_array($apeResults)) {
        $totalApe = array_sum($apeResults);
        $count = count($apeResults);
    } else {
```

```
            $totalApe = $apeResults;
            $count = 1;
        }

        return ($count > 0) ? $totalApe / $count : 0;
    }
}
```

e. *Source Code* Halaman *Triple Exponential Smoothing* Datang

```
class DatangControllers extends Controller
{
    public function index()
    {
        $datasets = Dataset::orderBy('tanggal')->get();
        $maxDatangValue = $datasets->max('datang');

        $datasets_filtered = $datasets
            ->groupBy(function ($data) {
                return Carbon::parse($data->tanggal)->format('Y-m');
            })
            ->flatMap(function ($group) {
                return $group->sortBy('tanggal');  // Remove 20 data limit
            })
            ->sortBy('tanggal')
            ->values();

        $first_20_data = $datasets_filtered->take(20);
        $average = $first_20_data->avg('datang');

        $firstMonth = Carbon::parse($datasets->first()->tanggal)->month;
        $secondMonth = $firstMonth + 1;

        $datasets_initial_trend = $datasets->filter(function ($data) use ($firstMonth, $secondMonth) {
            $month = Carbon::parse($data->tanggal)->month;
```

```
            return $month == $firstMonth || $month == $secondMonth;
        });

        $datasets_first_month = $datasets_initial_trend->filter(function ($data) use ($firstMonth) {
            return Carbon::parse($data->tanggal)->month == $firstMonth;
        })->values();

        $datasets_second_month = $datasets_initial_trend->filter(function ($data) use ($secondMonth)
{
            return Carbon::parse($data->tanggal)->month == $secondMonth;
        })->values();

        $initialTrendData = [];
        $pairCount = min($datasets_first_month->count(), $datasets_second_month->count());

        for ($i = 0; $i < $pairCount; $i++) {
            $m1 = $datasets_first_month[$i]->datang;
            $m2 = $datasets_second_month[$i]->datang;

            $initialTrendData[] = [
                'Month1' => $m1,
                'Month2' => $m2,
                'M2-M1' => $m2 - $m1,
                '(M2-M1)/20' => ($m2 - $m1) / 20,
            ];
        }

        $averageInitialTrend = collect($initialTrendData)->avg(function ($item) {
            return $item['(M2-M1)/20'];
        });

        // === LOOP 1: JANUARI–NOVEMBER
        foreach ($datasets_filtered as $index => $data) {
            $carbonDate = Carbon::parse($data->tanggal);
            $month = $carbonDate->month;
            $dateString = $carbonDate->format('Y-m-d');
            $previousData = $datasets_filtered[$index - 1] ?? null;
```

```
        if ($month == 12)
            continue;

        if ($dateString === '2023-01-28') {
            $data->trend_t = $averageInitialTrend;
        }

        if ($index < 20) {
            // For the first 20 records, use the average for level_at
            $data->level_at = $average;
        } else {
            $alpha = 0.14527154135641;  // Smoothing factor for level calculation
            $levelPrev = $previousData->level_at ?? 0;  // Previous record's level_at
            $trendPrev = $previousData->trend_t ?? 0;  // Previous record's trend_t

            // Cari seasonal dari index ke-(t-p)
            $correspondingIndex = $index - 20;
            $seasonalFromFixed = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;  // Get seasonal from previous record

            if ($seasonalFromFixed != 0) {
                $data->level_at = $alpha * ($data->datang / $seasonalFromFixed) + (1 - $alpha) * ($levelPrev + $trendPrev);
            } else {
                $data->level_at = 0;
            }
        }

        if ($index > 19 && $month >= 2 && $month <= 11) {
            $beta = 0.05;
            $levelNow = $data->level_at ?? 0;
            $levelPrev = $previousData->level_at ?? 0;
            $trendPrev = $previousData->trend_t ?? 0;

            $data->trend_t = $beta * ($levelNow - $levelPrev) + (1 - $beta) * $trendPrev;
        }

        if ($index < 20) {
```

```
            $data->seasonal_st = ($data->datang > 0 && $data->level_at != 0)
                ? $data->datang / $data->level_at
                : 0;
        } else {
            $gamma = 0.1;
            $correspondingIndex = $index - 20;
            $correspondingSeasonal = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;

            $levelAtNow = $data->level_at ?? 0;
            $datangNow = $data->datang ?? 0;

            $data->seasonal_st = ($levelAtNow != 0)
                ? $gamma * ($datangNow / $levelAtNow) + (1 - $gamma) * $correspondingSeasonal
                : 0;
        }

        if ($index >= 20) {
            $seasonal_base = $datasets_filtered->values();
            $previousData = $datasets_filtered[$index - 1] ?? null;

            $levelPrev = $previousData->level_at ?? 0;
            $trendPrev = $previousData->trend_t ?? 0;
            $correspondingIndex = $index - 20;
            $seasonalFromFixed = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;

            if ($levelPrev == 0 || $seasonalFromFixed == 0) {
                $data->forecast = 0;
            } else {
                $data->forecast = ($levelPrev + $trendPrev) * $seasonalFromFixed;
            }
        }

        $actual = $data->datang ?? 0;
        $forecast = $data->forecast ?? null;

        if (!is_null($forecast)) {
            $data->error = $actual - $forecast;
```

```
            $data->absolute_error = abs($data->error);
            $data->squared_error = pow($data->error, 2);

            if ($actual != 0 && ($data->level_at ?? 0) != 0 && ($data->seasonal_st ?? 0) != 0) {
                $data->absolute_percentage_error = abs($data->error - $actual) / $actual;
            } else {
                $data->absolute_percentage_error = 0;
            }
        }

        $data->tanggal_iso = $carbonDate->format('Y-m-d');
    }

    // === Ambil nilai level dan trend terakhir November
    $novemberLastData = $datasets_filtered->filter(fn($d) => Carbon::parse($d->tanggal)->month
=== 11)->last();
    $levelNovemberLast = $novemberLastData->level_at ?? 0;
    $trendNovemberLast = (float) $novemberLastData->trend_t ?? 0;
    $seasonal_november = $datasets_filtered
        ->filter(fn($d) => Carbon::parse($d->tanggal)->month === 11)
        ->values()
        ->take(20);

    // === LOOP 2: DESEMBER (forecast + error, hide columns)
    $desemberUrutan = 0;
    foreach ($datasets_filtered as $index => $data) {
        $carbonDate = Carbon::parse($data->tanggal);
        if ($carbonDate->month !== 12)
            continue;

        $desemberUrutan++;
        $seasonalIndex = ($desemberUrutan - 1) % $seasonal_november->count();
        $seasonal = $seasonal_november[$seasonalIndex]->seasonal_st ?? 1;

        $data->trend_t = $trendNovemberLast * $desemberUrutan;
        $forecast = ($levelNovemberLast + $desemberUrutan * $trendNovemberLast) * $seasonal;

        // Ensure forecast is within bounds (between 0 and 13)
```

```
        if ($forecast < 0) {
            $forecast = 0; // Set forecast to 0 if it's less than 0
        } elseif ($forecast > 13) {
            $forecast = 13; // Cap forecast at 13 if it's greater than 13
        }

        $data->forecast = $forecast;
        $data->level_at = null; // Tidak ditampilkan di view
        $data->seasonal_st = null; // Tidak ditampilkan di view

        $actual = $data->datang ?? 0;
        if (!is_null($data->forecast) && $actual != 0) {
            $data->error = $data->forecast - $actual;
        }

        $data->absolute_error = null;
        $data->squared_error = null;
        $data->absolute_percentage_error = null;

        $data->tanggal_iso = $carbonDate->format('Y-m-d');
    }

    // === Format output for Blade
    foreach ($datasets_filtered as $data) {
        $carbon = Carbon::parse($data->tanggal)->locale('id');
        $data->tanggal = $carbon->isoFormat('D MMMM YYYY');
        $data->hari = $carbon->isoFormat('dddd');
    }

    $desemberDataForLog = collect($datasets_filtered)
        ->filter(fn($d) => Carbon::parse($d->tanggal_iso)->month === 12)
        ->values()
        ->map(function ($d, $i) {
            return [
                'urutan' => $i + 1,
                'tanggal' => $d->tanggal_iso,
                'datang' => $d->datang,
```

```
            'trend_t' => $d->trend_t,
            'forecast' => $d->forecast,
            'error' => $d->error,
        ];
    });

$averageLevelAt = $datasets_filtered->take(20)->avg('level_at');
$onlyForecastDesember = collect($datasets_filtered)
    ->filter(fn($d) => Carbon::parse($d->tanggal_iso)->month === 12)
    ->values()
    ->pluck('forecast')
    ->map(fn($v) => round($v)) // optional rounding
    ->toArray();

session(['forecast_desember_only' => $onlyForecastDesember]);

// Calculate Akurasi and MAPE for December
list($sumAkurasi, $sumAPE, $total) = $this->calculateAkurasiAndAPE($datasets_filtered);

$tesAkurasiDatang = $sumAkurasi / $total * 100;
$tesMapeDatang = $sumAPE / $total * 100;

// Save or update the AkurasiMape record for Datang
$akurasiMape = AkurasiMape::find(1);

if ($akurasiMape) {
    $akurasiMape->update([
        'tes_akurasi_datang' => $tesAkurasiDatang,
        'tes_mape_datang' => $tesMapeDatang,
    ]);
} else {
    AkurasiMape::create([
        'tes_akurasi_datang' => $tesAkurasiDatang,
        'tes_mape_datang' => $tesMapeDatang,
    ]);
}
```

```
        // Filter data bukan Desember dan hanya yang ada APE-nya
        $filteredApe = $datasets_filtered->filter(function ($d) {
            $month = Carbon::parse($d->tanggal_iso)->month ?? null;
                    return $month !== 12 && isset($d->absolute_percentage_error) && $d->absolute_percentage_error !== null;
        });
        // // Ambil semua nilai APE yang digunakan
        // $apeValues = $filteredApe->pluck('absolute_percentage_error');


        // // Hitung jumlah data yang dihitung APE-nya
        // $totalData = $filteredApe->count();
        // Hitung rata-rata APE
        $averageApe = $filteredApe->avg('absolute_percentage_error');
        // dd([
        //     'APE Values' => $apeValues,
        //     'Total Data' => $totalData,
        //     'Average APE' => $averageApe
        // ]);
        return view('pages.smothing.datang.index', compact(
            'datasets_filtered',
            'initialTrendData',
            'averageInitialTrend',
            'averageLevelAt',
            'desemberDataForLog',
            'tesAkurasiDatang',
            'tesMapeDatang',
            'averageApe' // tambahkan ini

        ));
    }

    private function calculateAkurasiAndAPE($datasets_filtered)
    {
        $sumAkurasi = 0;
        $sumAPE = 0;
        $total = 0;

        foreach ($datasets_filtered as $row) {
```

```
            $actual = $row->datang;
            $forecast = $row->forecast;

            $akurasi = $forecast && $actual ? min($forecast, $actual) / max($forecast, $actual) : 0;
            $ape = 1 - $akurasi;

            $sumAkurasi += $akurasi;
            $sumAPE += $ape;
            $total++;
        }

        return [$sumAkurasi, $sumAPE, $total];
    }
}
```

f. *Source Code* Halaman *Triple Exponential Smoothing* Berangkat

```
class BerangkatControllers extends Controller
{
    public function index()
    {
        $datasets = Dataset::orderBy('tanggal')->get();
        $maxberangkatValue = $datasets->max('berangkat');

        $datasets_filtered = $datasets
            ->groupBy(function ($data) {
                return Carbon::parse($data->tanggal)->format('Y-m');
            })
            ->flatMap(function ($group) {
                return $group->sortBy('tanggal');  // Remove 20 data limit
            })
            ->sortBy('tanggal')
            ->values();

        $first_20_data = $datasets_filtered->take(20);
        $average = $first_20_data->avg('berangkat');
```

```
        $firstMonth = Carbon::parse($datasets->first()->tanggal)->month;
        $secondMonth = $firstMonth + 1;

        $datasets_initial_trend = $datasets->filter(function ($data) use ($firstMonth, $secondMonth)
{
            $month = Carbon::parse($data->tanggal)->month;
            return $month == $firstMonth || $month == $secondMonth;
        });

        $datasets_first_month = $datasets_initial_trend->filter(function ($data) use ($firstMonth) {
            return Carbon::parse($data->tanggal)->month == $firstMonth;
        })->values();

        $datasets_second_month = $datasets_initial_trend->filter(function ($data) use
($secondMonth) {
            return Carbon::parse($data->tanggal)->month == $secondMonth;
        })->values();

        $initialTrendData = [];
        $pairCount = min($datasets_first_month->count(), $datasets_second_month->count());

        for ($i = 0; $i < $pairCount; $i++) {
            $m1 = $datasets_first_month[$i]->berangkat;
            $m2 = $datasets_second_month[$i]->berangkat;

            $initialTrendData[] = [
                'Month1' => $m1,
                'Month2' => $m2,
                'M2-M1' => $m2 - $m1,
                '(M2-M1)/20' => ($m2 - $m1) / 20,
            ];
        }

        $averageInitialTrend = collect($initialTrendData)->avg(function ($item) {
            return $item['(M2-M1)/20'];
        });

        // === LOOP 1: JANUARI–NOVEMBER
```

```
        foreach ($datasets_filtered as $index => $data) {
            $carbonDate = Carbon::parse($data->tanggal);
            $month = $carbonDate->month;
            $dateString = $carbonDate->format('Y-m-d');
            $previousData = $datasets_filtered[$index - 1] ?? null;

            if ($month == 12)
                continue;

            if ($dateString === '2023-01-28') {
                $data->trend_t = $averageInitialTrend;
            }

            if ($index < 20) {
                // For the first 20 records, use the average for level_at
                $data->level_at = $average;
            } else {
                $alpha = 0.14527154135641;  // Smoothing factor for level calculation
                $levelPrev = $previousData->level_at ?? 0;  // Previous record's level_at
                $trendPrev = $previousData->trend_t ?? 0;  // Previous record's trend_t

                // Cari seasonal dari index ke-(t-p)
                $correspondingIndex = $index - 20;
                // Use the seasonal index from the previous record (index-1) for the calculation
                $seasonalFromFixed = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;  // Get seasonal from previous record

                // Ensure the seasonal index is valid (not 0) before performing the calculation
                if ($seasonalFromFixed != 0) {
                    // Calculate level_at using the seasonal index from the previous record
                    $data->level_at = $alpha * ($data->berangkat / $seasonalFromFixed) + (1 - $alpha) * ($levelPrev + $trendPrev);
                } else {
                    // If seasonal index is 0, set level_at to 0
                    $data->level_at = 0;
                }
            }
```

```
if ($index > 19 && $month >= 2 && $month <= 11) {
    $beta = 0.05;
    $levelNow = $data->level_at ?? 0;
    $levelPrev = $previousData->level_at ?? 0;
    $trendPrev = $previousData->trend_t ?? 0;

    $data->trend_t = $beta * ($levelNow - $levelPrev) + (1 - $beta) * $trendPrev;
}

if ($index < 20) {
    $data->seasonal_st = ($data->berangkat > 0 && $data->level_at != 0)
        ? $data->berangkat / $data->level_at
        : 0;
} else {
    $gamma = 0.1;

    // Cari seasonal dari index ke-(t-p)
    $correspondingIndex = $index - 20;
    $correspondingSeasonal = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;

    $levelAtNow = $data->level_at ?? 0;
    $berangkatNow = $data->berangkat ?? 0;

    $data->seasonal_st = ($levelAtNow != 0)
     ? $gamma * ($berangkatNow / $levelAtNow) + (1 - $gamma) * $correspondingSeasonal
      : 0;
}

if ($index >= 20) {
    $seasonal_base = $datasets_filtered->values();  // Take all data, not just the first 20
    $previousData = $datasets_filtered[$index - 1] ?? null;

    $levelPrev = $previousData->level_at ?? 0;
    $trendPrev = $previousData->trend_t ?? 0;
    // Cari seasonal dari index ke-(t-p)
    $correspondingIndex = $index - 20;
    // Use the seasonal index from the previous record (index-1) for the calculation
```

```
            $seasonalFromFixed = $datasets_filtered[$correspondingIndex]->seasonal_st ?? 0;  // Get seasonal from previous record


            if ($levelPrev == 0 || $seasonalFromFixed == 0) {
                $data->forecast = 0;
            } else {
                $data->forecast = ($levelPrev + $trendPrev) * $seasonalFromFixed;

            }
        }

        $actual = $data->berangkat ?? 0;
        $forecast = $data->forecast ?? null;

        if (!is_null($forecast)) {
            $data->error = $actual - $forecast;
            $data->absolute_error = abs($data->error);
            $data->squared_error = pow($data->error, 2);

            if ($actual != 0 && ($data->level_at ?? 0) != 0 && ($data->seasonal_st ?? 0) != 0) {
                $data->absolute_percentage_error = abs($data->error - $actual) / $actual;
            } else {
                $data->absolute_percentage_error = 0;
            }
        }

        $data->tanggal_iso = $carbonDate->format('Y-m-d');
    }

    // === Ambil nilai level dan trend terakhir November
        $novemberLastData = $datasets_filtered->filter(fn($d) => Carbon::parse($d->tanggal)->month === 11)->last();
    $levelNovemberLast = $novemberLastData->level_at ?? 0;
    $trendNovemberLast = (float) $novemberLastData->trend_t ?? 0;
    $seasonal_november = $datasets_filtered
        ->filter(fn($d) => Carbon::parse($d->tanggal)->month === 11)
        ->values()
        ->take(20);
```

```
// === LOOP 2: DESEMBER (forecast + error, hide columns)
$desemberUrutan = 0;
foreach ($datasets_filtered as $index => $data) {
    $carbonDate = Carbon::parse($data->tanggal);
    if ($carbonDate->month !== 12)
        continue;

    // Kalkulasi forecast untuk Desember
    $desemberUrutan++;
    $seasonalIndex = ($desemberUrutan - 1) % $seasonal_november->count();
    $seasonal = $seasonal_november[$seasonalIndex]->seasonal_st ?? 1;

    $data->trend_t = $trendNovemberLast * $desemberUrutan;
    $forecast = ($levelNovemberLast + $desemberUrutan * $trendNovemberLast) * $seasonal;

    // Ensure forecast is within bounds (between 0 and 13)
    if ($forecast < 0) {
        $forecast = 0; // Set forecast to 0 if it's less than 0
    } elseif ($forecast > 13) {
        $forecast = 13; // Cap forecast at 13 if it's greater than 13
    }
    $data->forecast = $forecast;
    $data->level_at = null; // Hidden in view
    $data->seasonal_st = null; // Hidden in view

    // Calculate error only 'error'
    $actual = $data->berangkat ?? 0;
    if (!is_null($data->forecast) && $actual != 0) {
        $data->error = $data->forecast - $actual;
    }

    // Remove other columns in view
    $data->absolute_error = null;
    $data->squared_error = null;
    $data->absolute_percentage_error = null;
```

```
        $data->tanggal_iso = $carbonDate->format('Y-m-d');
    }

    // === Format output for Blade
    foreach ($datasets_filtered as $data) {
        $carbon = Carbon::parse($data->tanggal)->locale('id');
        $data->tanggal = $carbon->isoFormat('D MMMM YYYY');
        $data->hari = $carbon->isoFormat('dddd');
    }

    $desemberDataForLog = collect($datasets_filtered)
        ->filter(fn($d) => Carbon::parse($d->tanggal_iso)->month === 12)
        ->values()
        ->map(function ($d, $i) {
            return [
                'urutan' => $i + 1,
                'tanggal' => $d->tanggal_iso,
                'berangkat' => $d->berangkat,
                'trend_t' => $d->trend_t,
                'forecast' => $d->forecast,
                'error' => $d->error,
            ];
        });

    $averageLevelAt = $datasets_filtered->take(20)->avg('level_at');
    // Create pure forecast array for December (only numbers)
    $onlyForecastBerangkat = collect($datasets_filtered)
        ->filter(fn($d) => Carbon::parse($d->tanggal_iso)->month === 12)
        ->values()
        ->pluck('forecast')
        ->map(fn($v) => round($v)) // optional rounding
        ->toArray();

    // Save to session
    session(['forecast_berangkat_only' => $onlyForecastBerangkat]); // Forecast untuk berangkat
    // Calculate Akurasi and MAPE for December
                list($sumAkurasiBerangkat, $sumAPEBerangkat, $total) = $this-
>calculateAkurasiAndAPEForBerangkat($datasets_filtered);
```

```
    // Calculate the final values
    $tesAkurasiBerangkat = $sumAkurasiBerangkat / $total * 100;
    $tesMapeBerangkat = $sumAPEBerangkat / $total * 100;

    // Check if there's already a record with id = 1, and update or create accordingly
    $akurasiMape = AkurasiMape::find(1);

    if ($akurasiMape) {
        // If it exists, update it
        $akurasiMape->update([
            'tes_akurasi_berangkat' => $tesAkurasiBerangkat,
            'tes_mape_berangkat' => $tesMapeBerangkat,
        ]);
    } else {
        // Otherwise, create a new one
        AkurasiMape::create([
            'tes_akurasi_berangkat' => $tesAkurasiBerangkat,
            'tes_mape_berangkat' => $tesMapeBerangkat,
        ]);
    }
    $filteredApe = $datasets_filtered->filter(function ($d) {
        $month = Carbon::parse($d->tanggal_iso)->month ?? null;
                return $month !== 12 && isset($d->absolute_percentage_error) && $d->absolute_percentage_error !== null;
    });
    // Ambil semua nilai APE yang digunakan
    // $apeValues = $filteredApe->pluck('absolute_percentage_error');

    // Hitung jumlah data yang dihitung APE-nya
    // $totalData = $filteredApe->count();
    $averageApe = $filteredApe->avg('absolute_percentage_error');
    // dd([
    //     'APE Values' => $apeValues,
    //     'Total Data' => $totalData,
    //     'Average APE' => $averageApe
    // ]);
    // Return the view with the necessary data
```

```
        return view('pages.smothing.berangkat.index', compact(
            'datasets_filtered',
            'initialTrendData',
            'averageInitialTrend',
            'averageLevelAt',
            'desemberDataForLog',
            'tesAkurasiBerangkat',
            'tesMapeBerangkat',
            'averageApe' // tambahkan ini

        ));
    }

    private function calculateAkurasiAndAPEForBerangkat($datasets_filtered)
    {
        $sumAkurasi = 0;
        $sumAPE = 0;
        $total = 0;

        foreach ($datasets_filtered as $row) {
            $actual = $row->berangkat;  // Use berangkat
            $forecast = $row->forecast;

            $akurasi = $forecast && $actual ? min($forecast, $actual) / max($forecast, $actual) : 0;
            $ape = 1 - $akurasi;

            $sumAkurasi += $akurasi;
            $sumAPE += $ape;
            $total++;
        }

        return [$sumAkurasi, $sumAPE, $total];
    }
}
```

g. *Source Code* Halaman Akhir

```
class HasilAkhirController extends Controller
{
    public function index(Request $request)
    {
        // Fetch datasets ordered by 'tanggal'
        $datasets = Dataset::orderBy('tanggal')->get();

        // Filter only December data
        $desemberData = $datasets->filter(function ($data) {
            return Carbon::parse($data->tanggal)->month === 12;
        })->values(); // Make sure the indexing starts from 0

        // Fetch AkurasiMape data from the database
        $akurasiMape = AkurasiMape::find(1); // Assuming the relevant data is in row with ID = 1

        // Get forecasts from the session
        $tesForecastsDatang = session('forecast_desember_only', []);
        $tesForecastsBerangkat = session('forecast_berangkat_only', []);
        // Fetch forecasts from the session
        $monteCarloForecastDatang = session('montecarlo_forecast_datang', []); // Correct session key for 'datang'
        $monteCarloForecastBerangkat = session('montecarlo_forecast_berangkat', []);

        // Debugging: Log session values for monitoring
        Log::info('Forecast for Datang: ', session('montecarlo_forecast_desember', []));
        Log::info('Forecast for Berangkat: ', session('montecarlo_forecast_berangkat', []));

        // Combine all data into a final array for the view
        $finalData = $desemberData->map(function ($data, $index) use ($tesForecastsDatang, $tesForecastsBerangkat, $monteCarloForecastDatang, $monteCarloForecastBerangkat) {
            return [
                'id' => $index + 1,
                'tanggal' => $data->tanggal,
                'datang' => $data->datang,
                'berangkat' => $data->berangkat,
                'prediksi_montecarlo_datang' => $monteCarloForecastDatang[$index] ?? 'No Data',
```

```
            'prediksi_tes_datang' => $tesForecastsDatang[$index] ?? 'No Data',
            'prediksi_montecarlo_berangkat' => $monteCarloForecastBerangkat[$index] ?? 'No Data',
            'prediksi_tes_berangkat' => $tesForecastsBerangkat[$index] ?? 'No Data'
        ];
    });

    // Pass both final data and AkurasiMape to the view
    return view('pages.hasil-akhir.index', [
        'finalData' => $finalData,
        'akurasiMape' => $akurasiMape,
    ]);
  }
}
```