Lampiran 1: data jumlah penduduk Dinas Kependudukan Dan Catatan Sipil Kota Palembang

DATA JUMLAH PENDUDUK KOTA PALEMBANG 2016 - 2020

berdasarkan catatan **DISDUKCAPIL Kota Palembang** tahun 2021:

NO	TAHUN	BULAN												JUMLAH
		JANUARI	FEBRUARI	MARET	APRIL	MEI	JUNI	JULI	AGUSTUS	SEPTEMBER	OKTOBER	NOVEMBER	DESEMBER	
1	**2016**	1.525.945	1.526.174	1.528.229	1.527.213	1.530.175	1.529.990	1531.731	1.532.102	1.532.000	1.534.673	1.535.980	1.555.980	1.555.980
2	**2017**	1.559.132	1.562.540	1.567.909	1.570.187	1.585.009	1.593.840	1.594.601	1.590.321	1.585.582	1.587.119	1.579.000	1.573.898	1.573.898
3	**2018**	1.578.689	1.575.221	1.579.991	1.574.185	1.577.540	1.580.312	1.583.221	1.588.139	1.589.003	1.587.841	1.589.187	1.592.248	1.592.248
4	**2019**	1.621.641	1.618.967	1.620.082	1.622.219	1.617.298	1.615.658	1.613.935	1.611.112	1.613.923	1.616.256	1.616.990	1.619.533	1.619.533
5	**2020**	1.623.146	1.628.435	1.626.213	1.632.645	1637.908	1.641.668	1.645.758	1.642.932	1.659.174	1.667.214	1.665.546	1.668.164	1.668.164
Jumlah keseluruhan														**6.320.290**

Kesimpulan:

1. Dalam kurun waktu 5 tahun banyak peningkatan dari jumlah penduduk kota palembang
2. Jumlah penduduk paling sedikit
3. Jumlah penduduk paling banyak pada bulan Desember 2020 (Semester II)

Lampiran 2: *source program*

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-
                fit=no">
  <title>
   FTS-PSO
  </title>
  <link rel="stylesheet"
                href="assets/bootstrap/css/bootstrap.min.css?h=c25e0cf726e61fe1945
                be655ed94b205">
  <link rel="stylesheet" href="assets/css/Navbar-Centered-Brand-
                Dark.css?h=befd8a398792e305b7ffd4a176b5b585">
  <link rel="stylesheet"
                href="assets/css/styles.css?h=b57243cc2e71ef3c66bc178bffbd4805">
</head>

<body>
  <nav class="navbar navbar-dark navbar-expand-md fixed-top bg-dark py-3">
    <div class="container">
      <a class="navbar-brand d-flex align-items-center" href="#">
        <span>
          FTS-PSO
        </span>
      </a>
      <button data-bs-toggle="collapse" class="navbar-toggler" data-bs-target="#navcol-
                5">
        <span class="visually-hidden">
          Toggle navigation
        </span>
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navcol-5">
        <ul class="navbar-nav ms-auto">
          <li class="nav-item">
            <button id="ctaRetry" class="btn btn-outline-light d-none" type="button"
                title="Retry">
              <svg
                xmlns="http://www.w3.org/2000/svg" width="1em" height="1em"
                fill="currentColor" viewBox="0 0 16 16"
                class="bi bi-arrow-counterclockwise">
                <path fill-rule="evenodd" d="M8 3a5 5 0 1 1-4.546 2.914.5.5 0 0 0-.908-
                .417A6 6 0 1 0 8 2v1z"></path>
                <path d="M8 4.466V.534a.25.25 0 0 0-.41-.192L5.23 2.308a.25.25 0 0 0 0
                .384l2.36 1.966A.25.25 0 0 0 8 4.466z"></path>
              </svg>
            </button>
          </li>
          <li class="nav-item">
            <div id="stageControl" class="btn-group d-none" role="group">
              <input type="radio" class="btn-check" name="stage" id="stageClient"
```

```
                autocomplete="off" value="client">
            <label class="btn btn-outline-primary" for="stageClient" data-bs-
                toggle="tooltip" data-bs-placement="bottom" title="Run in Client
                Side">
              Client
            </label>

            <input type="radio" class="btn-check" name="stage" id="stageServer"
                autocomplete="off" value="server">
            <label class="btn btn-outline-primary" for="stageServer" data-bs-
                toggle="tooltip" data-bs-placement="bottom" title="Run at Server
                Side">
              Server
            </label>
          </div>
        </li>
      </ul>
    </div>
  </div>
</nav>
<div class="container-fluid floor-down" id="mainContainer" style="height:
                calc(100vh - 72px);margin-top: 72px;">
  <form id="configForm" class="d-flex flex-column w-100 px-3 d-none">
    <fieldset class="mb-3">
      <legend>Dataset Source</legend>
      <input class="form-control" type="file" name="dataset" required=""
                accept="text/csv">
      <a href="assets/dataset.csv" target="_blank">Download example dataset</a>
    </fieldset>
    <div class="row flex-fill mb-3">
      <div class="col-md-12 col-lg-6 mb-md-3 mb-lg-0">
        <div class="card h-100">
          <div class="card-header">
            <h5 class="mb-0">
              Fuzzy Time Series
            </h5>
          </div>
          <div class="card-body floor-down">
            <div class="mb-3">
              <label class="form-label" for="ftsMinMargin">
                Lower Margin
              </label>
              <input class="form-control" type="number" id="ftsMinMargin"
                name="ftsMinMargin" placeholder="Minimum Margin" required=""
                step="0.0001">
            </div>
            <div class="mb-3">
              <label class="form-label" for="ftsMaxMargin">
                Upper Margin
              </label>
              <input class="form-control" type="number" id="ftsMaxMargin"
                name="ftsMaxMargin" placeholder="Maximum Margin" required=""
                step="0.0001">
            </div>
            <div class="mb-3">
              <label class="form-label" for="ftsInterval">
                Interval
```

```
        </label>
        <input class="form-control" type="number" id="ftsInterval"
                name="ftsInterval" placeholder="Interval" required=""
                step="0.0001">
      </div>
    </div>
  </div>
</div>
<div class="col-md-12 col-lg-6">
  <div class="card h-100">
    <div class="card-header">
      <h5 class="mb-0">
        Particle Swarm Optimization
      </h5>
    </div>
    <div class="card-body floor-down">
      <div class="mb-3">
        <label class="form-label">
          Spaces Configuration
        </label>
        <div class="row mx-1">
          <div class="col-sm-12 col-md-6">
            <div class="row">
              <div class="col-3">
                <label class="col-form-label" for="psoSpaceMin">
                  Min
                </label>
              </div>
              <div class="col">
                <input class="form-control" type="number" id="psoSpaceMin"
                    name="psoSpaceMin" placeholder="Minimum Value"
                    step="0.0001">
              </div>
            </div>
          </div>
          <div class="col-sm-12 col-md-6">
            <div class="row">
              <div class="col-3">
                <label class="col-form-label" for="psoSpaceMax">
                  Max
                </label>
              </div>
              <div class="col">
                <input class="form-control" type="number" id="psoSpaceMax"
                    name="psoSpaceMax" placeholder="Maximum Value"
                    step="0.0001">
              </div>
            </div>
          </div>
        </div>
      </div>
      <div class="mb-3">
        <label class="form-label" for="psoWeight">
          Weight
        </label>
        <input class="form-control" type="number" id="psoWeight"
                name="psoWeight" placeholder="Weight" required=""
```

```
            step="0.0001">
</div>
<div class="mb-3">
  <label class="form-label" for="psoMaxIteration">
    Maximum Iteration Count
  </label>
  <input class="form-control" type="number" id="psoMaxIteration"
            name="psoMaxIteration" placeholder="Maximum Iteration Count"
            min="1" step="1" required="">
</div>
<div class="mb-3">
  <label class="form-label" for="psoParticleCount">
    Spawned Particle Count
  </label>
  <input class="form-control" type="number" id="psoParticleCount"
            name="psoParticleCount" placeholder="Spawned Particle Count"
            min="1" step="1" required="">
</div>
<div class="mb-3">
  <label class="form-label" for="psoConfidence">
    Swarm Confidence
  </label>
  <input class="form-control" type="number" id="psoConfidence"
            name="psoConfidence" placeholder="Swarm Confidence"
            required="" step="0.0001">
</div>
<div class="mb-3">
  <label class="form-label" for="psoSelfConfidence">
    Self Confidence
  </label>
  <input class="form-control" type="number" id="psoSelfConfidence"
            name="psoSelfConfidence" placeholder="Self Confidence"
            required="" step="0.0001">
</div>
<div class="mb-3">
  <label class="form-label">
    Stopping Criteria
  </label>
  <div class="row mx-1">
    <div class="col-sm-12 col-md-6">
      <div class="row">
        <div class="col-3">
          <label class="col-form-label" for="psoStopMse">
            MSE
          </label>
        </div>
        <div class="col">
          <input class="form-control" type="number" id="psoStopMse"
              name="psoStopMse" placeholder="Minimum MSE" min="0"
              required="" step="0.0001">
        </div>
      </div>
    </div>
    <div class="col-sm-12 col-md-6">
      <div class="row">
        <div class="col-3">
          <label class="col-form-label" for="psoStopAfer">
```

```
                AFER
               </label>
              </div>
              <div class="col">
               <input class="form-control" type="number" id="psoStopAfer"
                   name="psoStopAfer" placeholder="Minimum AFER" min="0"
                   required="" step="0.0001">
              </div>
             </div>
            </div>
           </div>
          </div>
         </div>
        </div>
       </div>
       <button class="btn btn-primary mb-3" id="ctaExecute" type="button">
        Execute
       </button>
       <button class="btn btn-outline-primary mb-3 d-none" id="ctaFakeLoading"
                   type="button">
        <span class="spinner-border spinner-border-sm" role="status"></span>
       </button>
      </form>
      <div id="resultGraph" class="container d-none">
       <div class="row flex-column py-3" style="gap: 1rem;">
        <div class="col">
         <div class="position-relative">
          <canvas id="ftsChart" data-bss-
                   chart="{&quot;type&quot;:&quot;line&quot;,&quot;data&quot;:{&quot;labels&quot;:[&quot;January&quot;,&quot;February&quot;,&quot;March&quot;,&quot;April&quot;,&quot;May&quot;,&quot;June&quot;],&quot;datasets&quot;:[{&quot;label&quot;:&quot;Revenue&quot;,&quot;backgroundColor&quot;:&quot;#4e73df&quot;,&quot;borderColor&quot;:&quot;#4e73df&quot;,&quot;data&quot;:[&quot;4500&quot;,&quot;5300&quot;,&quot;6250&quot;,&quot;7800&quot;,&quot;9800&quot;,&quot;15000&quot;]}]},&quot;options&quot;:{&quot;maintainAspectRatio&quot;:true,&quot;legend&quot;:{&quot;display&quot;:true,&quot;labels&quot;:{&quot;fontStyle&quot;:&quot;normal&quot;},&quot;position&quot;:&quot;right&quot;},&quot;title&quot;:{&quot;fontStyle&quot;:&quot;bold&quot;,&quot;display&quot;:true,&quot;text&quot;:&quot;Fuzzy Time Series Result&quot;},&quot;scales&quot;:{&quot;xAxes&quot;:[{&quot;ticks&quot;:{&quot;fontStyle&quot;:&quot;normal&quot;}}],&quot;yAxes&quot;:[{&quot;ticks&quot;:{&quot;fontStyle&quot;:&quot;normal&quot;}}]}}}"></canvas>
         </div>
        </div>
        <div class="col">
         <div class="position-relative">
          <canvas id="ftsPsoChart" data-bss-
                   chart="{&quot;type&quot;:&quot;line&quot;,&quot;data&quot;:{&quot;labels&quot;:[&quot;January&quot;,&quot;February&quot;,&quot;March&quot;,&quot;April&quot;,&quot;May&quot;,&quot;June&quot;],&quot;datasets&quot;:[{&quot;label&quot;:&quot;Revenue&quot;,&quot;backgroundColor&quot;:&quot;#4e73df&quot;,&quot;borderColor&quot;:&quot;#4e73df&quot;,&quot;data&quot;:[&quot;4500
```

```
&amp;quot;,&amp;quot;5300&amp;quot;,&amp;quot;6250&amp;quot;,&amp;quot;7800&amp;quot;,&amp;quot;9800&amp;quot;,&amp;quot;15000&amp;quot;]}]},&amp;quot;options&amp;quot;:{&amp;quot;maintainAspectRatio&amp;quot;:true,&amp;quot;legend&amp;quot;:{&amp;quot;display&amp;quot;:true,&amp;quot;labels&amp;quot;:{&amp;quot;fontStyle&amp;quot;:&amp;quot;normal&amp;quot;},&amp;quot;position&amp;quot;:&amp;quot;right&amp;quot;},&amp;quot;title&amp;quot;:{&amp;quot;fontStyle&amp;quot;:&amp;quot;bold&amp;quot;,&amp;quot;display&amp;quot;:true,&amp;quot;text&amp;quot;:&amp;quot;Fuzzy Time Series Result Optimized using Particle Swarm Optimization&amp;quot;},&amp;quot;scales&amp;quot;:{&amp;quot;xAxes&amp;quot;:[{&amp;quot;ticks&amp;quot;:{&amp;quot;fontStyle&amp;quot;:&amp;quot;normal&amp;quot;}}],&amp;quot;yAxes&amp;quot;:[{&amp;quot;ticks&amp;quot;:{&amp;quot;fontStyle&amp;quot;:&amp;quot;normal&amp;quot;}}]}}}"></canvas>
      </div>
     </div>
    </div>
   </div>
  </div>

  <script src="https://unpkg.com/systemjs@6.12.1/dist/system.min.js"></script>
  <script
          src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
  <script
          src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.bundle.min.js"></script>
  <script src="assets/js/bs-init.js?h=e2b0d57f2c4a9b0d13919304f87f79ae"></script>
  <script src="scripts/fts-pso/types.ts"></script>
  <script src="scripts/fts-pso/utils.ts"></script>
  <script src="scripts/fts-pso/fuzzy.ts"></script>
  <script src="scripts/fts-pso/FTS.ts"></script>
  <script src="scripts/fts-pso/particle.ts"></script>
  <script src="scripts/fts-pso/swarm.ts"></script>
  <script src="scripts/fts-pso/main.ts"></script>
</body>

</html>
```

FTS.ts

```
import { FuzzyTriangleGate } from './fuzzy';
import {
    FTSOptions,
    FTSTestOptions,
    KeyValuePair,
} from './types';

export class FTS<TKey, TValue extends number> {
    private _dataset: Array<KeyValuePair<TKey, TValue>>;
    private _minMargin: number;
    private _maxMargin: number;
    private _marginMultiplier: number;
    private _partitionInterval: number;
    private _partitionCount: number;
    private _partitionRef: Array<FuzzyTriangleGate>;
    private _ruleset: {
        [precedent: number]: Set<number>;
    };
```

```
public get maxValue() {
   return Math.max(...this._dataset.map(v => v.value));
}
public get minValue() {
   return Math.min(...this._dataset.map(v => v.value));
}
public get lowerBound() {
   return this.minValue - this._minMargin;
}
public get upperBound() {
   return this.maxValue + this._maxMargin;
}
public get partitionCount() {
   return this._partitionCount;
}
public get partitionLength() {
   return this._partitionInterval;
}

constructor(dataset: Array<KeyValuePair<TKey, TValue>>, options?: FTSOptions)
                  {
   this._dataset = dataset;
   this._marginMultiplier = options?.marginMultiplier || 0.1;
   this._minMargin = options?.minMargin || (this.minValue * this._marginMultiplier);
   this._maxMargin = options?.maxMargin || (this.maxValue *
                  this._marginMultiplier);
   if (options?.interval) {
      this._partitionInterval = options.interval;
      this._partitionCount = Math.ceil((this.upperBound - this.lowerBound) /
                  options.interval);
   } else {
      this._partitionCount = options?.partitionCount || 10;
      this._partitionInterval = (this.upperBound - this.lowerBound) /
                  this._partitionCount;
   }
   this._partitionRef = new Array<FuzzyTriangleGate>();
   this._ruleset = {};
   for (let i = 0; i < this._partitionCount; i++) {
      const prevPoint = i === 0 ? 0 : (this.lowerBound + (this._partitionInterval * (i -
                  1)));
      const maxPoint = this.lowerBound + (this._partitionInterval * i);
      const nextPoint = this.lowerBound + (this._partitionInterval * (i + 1));
      this._partitionRef.push(new FuzzyTriangleGate(prevPoint, maxPoint,
                  nextPoint));
      this._ruleset[i] = new Set<number>();
   }
   const _fuzzySet = new Array<number>();
   for (let i = 1; i < this.partitionCount; i++) {

   }
}

private nearestPartition(value: number) {
   const degrees = this._partitionRef.map(x => x.degree(value));
   const highestDegree = Math.max(...degrees);
   return degrees.findIndex(x => x === highestDegree);
}
```

```
    public train() {
        const generatedPattern = this._dataset.map(v => this.nearestPartition(v.value));
        for (let i = 1; i < generatedPattern.length; i++) {
            const precedent = generatedPattern[i - 1];
            const consequent = generatedPattern[i];
            this._ruleset[precedent].add(consequent);
        }
    }

    public test(options?: FTSTestOptions<TKey, TValue>) {
        const baseData = options?.dataset || this._dataset;
        const predicted = baseData.map(({ key, value }) => {
            const partitionIndex = this.nearestPartition(value);
            const partitionConsequent = [...(this._ruleset[partitionIndex] || new
                        Set<number>()).values()];
            const predictedValue = partitionConsequent.length === 0 ?
                (this._partitionRef[partitionIndex].median) :
                ([...partitionConsequent].map(x => this._partitionRef[x].median).reduce((p, c)
                        => p + c, 0) / partitionConsequent.length);
            return {
                key,
                value: value,
                predicted: predictedValue,
            };
        });
        return predicted;
    }
}
```

App.ts

```
import { FTS } from './FTS';
import { Swarm } from './swarm';
import { KeyValuePair } from './types';
import { averageForecastingErrorRate, meanSquaredError } from './utils';

async function main() {
    const dataset: Array<KeyValuePair<string, number>> = [
        { key: '2016-Januari', value: 1525945 },
        { key: '2016-Februari', value: 1526174 },
        { key: '2016-Maret', value: 1528229 },
        { key: '2016-April', value: 1527213 },
        { key: '2016-Mei', value: 1530175 },
        { key: '2016-Juni', value: 1529990 },
        { key: '2016-Juli', value: 1531731 },
        { key: '2016-Agustus', value: 1532102 },
        { key: '2016-September', value: 1532000 },
        { key: '2016-Oktober', value: 1534673 },
        { key: '2016-November', value: 1535980 },
        { key: '2016-Desember', value: 1555980 },
        { key: '2017-Januari', value: 1559132 },
        { key: '2017-Februari', value: 1562540 },
        { key: '2017-Maret', value: 1567909 },
        { key: '2017-April', value: 1570187 },
        { key: '2017-Mei', value: 1585009 },
        { key: '2017-Juni', value: 1593840 },
        { key: '2017-Juli', value: 1594601 },
```

```
    { key: '2017-Agustus', value: 1590321 },
    { key: '2017-September', value: 1585582 },
    { key: '2017-Oktober', value: 1587119 },
    { key: '2017-November', value: 1579000 },
    { key: '2017-Desember', value: 1573898 },
    { key: '2018-Januari', value: 1578689 },
    { key: '2018-Februari', value: 1575221 },
    { key: '2018-Maret', value: 1579991 },
    { key: '2018-April', value: 1574185 },
    { key: '2018-Mei', value: 1577540 },
    { key: '2018-Juni', value: 1580312 },
    { key: '2018-Juli', value: 1583221 },
    { key: '2018-Agustus', value: 1588139 },
    { key: '2018-September', value: 1589003 },
    { key: '2018-Oktober', value: 1587841 },
    { key: '2018-November', value: 1589187 },
    { key: '2018-Desember', value: 1592248 },
    { key: '2019-Januari', value: 1621641 },
    { key: '2019-Februari', value: 1618967 },
    { key: '2019-Maret', value: 1620082 },
    { key: '2019-April', value: 1622219 },
    { key: '2019-Mei', value: 1617298 },
    { key: '2019-Juni', value: 1615658 },
    { key: '2019-Juli', value: 1613935 },
    { key: '2019-Agustus', value: 1611112 },
    { key: '2019-September', value: 1613923 },
    { key: '2019-Oktober', value: 1616256 },
    { key: '2019-November', value: 1616990 },
    { key: '2019-Desember', value: 1619533 },
    { key: '2020-Januari', value: 1623146 },
    { key: '2020-Februari', value: 1628435 },
    { key: '2020-Maret', value: 1626213 },
    { key: '2020-April', value: 1632645 },
    { key: '2020-Mei', value: 1637908 },
    { key: '2020-Juni', value: 1641668 },
    { key: '2020-Juli', value: 1645758 },
    { key: '2020-Agustus', value: 1642932 },
    { key: '2020-September', value: 1659174 },
    { key: '2020-Oktober', value: 1667214 },
    { key: '2020-November', value: 1665546 },
    { key: '2020-Desember', value: 1668164 },
];
const min = Math.min(...dataset.map(v => v.value));
const max = Math.max(...dataset.map(v => v.value));
const minBorder = min * 0.1;
const maxBorder = max * 0.1;
const engine = new FTS(dataset, {
    minMargin: minBorder,
    maxMargin: maxBorder,
    interval: ((max * 1.1) - (min * 0.9)) / 10,
});
engine.train();
const singleResult = engine.test();
const forecasted = singleResult.slice(0, singleResult.length - 1).map(v =>
                    v.predicted);
const actual = singleResult.slice(1, singleResult.length).map(v => v.value);
const mse = meanSquaredError(actual, forecasted);
```

```
const afer = averageForecastingErrorRate(actual, forecasted);
console.table(singleResult);
console.log({ mse, afer });

const swarm = new Swarm<{ mse: number; afer: number; }>({
   spaces: { min: 10, max: 1000 },
   weight: 0.1,
   maxIteration: 10000,
   particleCount: 100,
   swarmConfidence: 2,
   selfConfidence: 2,
   fitnessFunction: (n) => {
      const subFts = new FTS(dataset, {
         minMargin: minBorder,
         maxMargin: maxBorder,
         interval: ((max * 1.1) - (min * 0.9)) / Math.abs(n),
      });
      subFts.train();
      const subResult = subFts.test();
      const subForecasted = subResult.slice(0, subResult.length - 1).map(v =>
               v.predicted);
      const subActual = subResult.slice(1, singleResult.length).map(v => v.value);
      return {
         mse: meanSquaredError(subActual, subForecasted),
         afer: averageForecastingErrorRate(subActual, subForecasted),
      };
   },
   selectorFunction: (results) => {
      let bestIndex = 0;
      for (let i = 1; i < results.length; i++) {
         if (results[i].mse < results[bestIndex].mse || results[i].afer <
               results[bestIndex].afer) {
            bestIndex = i;
         }
      }
      return bestIndex;
   },
   stopCriteria: (n) => n.mse <= 10 || Math.abs(n.afer) <= 0.000001,
});
const lastIteration = swarm.optimize();
console.log('Best interval count:', swarm.bestPosition);
console.log('Iteration passed:', lastIteration);
const optimizedEngine = new FTS(dataset, {
   minMargin: minBorder,
   maxMargin: maxBorder,
   interval: ((max * 1.1) - (min * 0.9)) / swarm.bestPosition,
});
optimizedEngine.train();
const optimizedResult = optimizedEngine.test();
const optimizedForecast = optimizedResult.slice(0, optimizedResult.length -
               1).map(v => v.predicted);
const optimizedActual = optimizedResult.slice(1, optimizedResult.length).map(v =>
               v.value);
console.table(optimizedResult);
console.log({
   mse: meanSquaredError(optimizedActual, optimizedForecast),
   afer: averageForecastingErrorRate(optimizedActual, optimizedForecast),
```

```
    });
}

main().catch(e => {
    console.trace(e);
});
```

Main.ts

```
import { FuzzyTriangleGate } from './fuzzy';
import {
    FTSOptions,
    FTSTestOptions,
    KeyValuePair,
} from './types';

export class FTS<TKey, TValue extends number> {
    private _dataset: Array<KeyValuePair<TKey, TValue>>;
    private _minMargin: number;
    private _maxMargin: number;
    private _marginMultiplier: number;
    private _partitionInterval: number;
    private _partitionCount: number;
    private _partitionRef: Array<FuzzyTriangleGate>;
    private _ruleset: {
        [precedent: number]: Set<number>;
    };

    public get maxValue() {
        return Math.max(...this._dataset.map(v => v.value));
    }
    public get minValue() {
        return Math.min(...this._dataset.map(v => v.value));
    }
    public get lowerBound() {
        return this.minValue - this._minMargin;
    }
    public get upperBound() {
        return this.maxValue + this._maxMargin;
    }
    public get partitionCount() {
        return this._partitionCount;
    }
    public get partitionLength() {
        return this._partitionInterval;
    }

    constructor(dataset: Array<KeyValuePair<TKey, TValue>>, options?: FTSOptions)
                    {
        this._dataset = dataset;
        this._marginMultiplier = options?.marginMultiplier || 0.1;
        this._minMargin = options?.minMargin || (this.minValue * this._marginMultiplier);
        this._maxMargin = options?.maxMargin || (this.maxValue *
                        this._marginMultiplier);
        if (options?.interval) {
            this._partitionInterval = options.interval;
            this._partitionCount = Math.ceil((this.upperBound - this.lowerBound) /
```

```
                options.interval);
        } else {
            this._partitionCount = options?.partitionCount || 10;
            this._partitionInterval = (this.upperBound - this.lowerBound) /
                this._partitionCount;
        }
        this._partitionRef = new Array<FuzzyTriangleGate>();
        this._ruleset = {};
        for (let i = 0; i < this._partitionCount; i++) {
            const prevPoint = i === 0 ? 0 : (this.lowerBound + (this._partitionInterval * (i -
                1)));
            const maxPoint = this.lowerBound + (this._partitionInterval * i);
            const nextPoint = this.lowerBound + (this._partitionInterval * (i + 1));
            this._partitionRef.push(new FuzzyTriangleGate(prevPoint, maxPoint,
                nextPoint));
            this._ruleset[i] = new Set<number>();
        }
        const _fuzzySet = new Array<number>();
        for (let i = 1; i < this.partitionCount; i++) {

        }
    }

    private nearestPartition(value: number) {
        const degrees = this._partitionRef.map(x => x.degree(value));
        const highestDegree = Math.max(...degrees);
        return degrees.findIndex(x => x === highestDegree);
    }

    public train() {
        const generatedPattern = this._dataset.map(v => this.nearestPartition(v.value));
        for (let i = 1; i < generatedPattern.length; i++) {
            const precedent = generatedPattern[i - 1];
            const consequent = generatedPattern[i];
            this._ruleset[precedent].add(consequent);
        }
    }

    public test(options?: FTSTestOptions<TKey, TValue>) {
        const baseData = options?.dataset || this._dataset;
        const predicted = baseData.map(({ key, value }) => {
            const partitionIndex = this.nearestPartition(value);
            const partitionConsequent = [...(this._ruleset[partitionIndex] || new
                Set<number>()).values()];
            const predictedValue = partitionConsequent.length === 0 ?
                (this._partitionRef[partitionIndex].median) :
                ([...partitionConsequent].map(x => this._partitionRef[x].median).reduce((p, c)
                    => p + c, 0) / partitionConsequent.length);
            return {
                key,
                value: value,
                predicted: predictedValue,
            };
        });
        return predicted;
    }
}
```

```
import { FTS } from "./FTS";
import { Swarm } from "./swarm";
import { averageForecastingErrorRate, meanSquaredError } from "./utils";

let strategy: 'client' | 'server' = 'client';
const parsedDataset = new Array<{ key: string; value: number; }>();

const stageControlRef = document.getElementById('stageControl') as
                    HTMLDivElement;
const stageClientRef = document.getElementById('stageClient') as
                    HTMLInputElement;
const stageServerRef = document.getElementById('stageServer') as
                    HTMLInputElement;
const retryRef = document.getElementById('ctaRetry') as HTMLButtonElement;
const configFormRef = document.getElementById('configForm') as
                    HTMLFormElement;
const resultRef = document.getElementById('resultGraph') as HTMLDivElement;

const dataSourceInputRef = document.querySelector('input[name=dataset]') as
                    HTMLInputElement;
const ftsMinMarginInputRef = document.querySelector('input[name=ftsMinMargin]')
                    as HTMLInputElement;
const ftsMaxMarginInputRef = document.querySelector('input[name=ftsMaxMargin]')
                    as HTMLInputElement;
const ftsIntervalInputRef = document.querySelector('input[name=ftsInterval]') as
                    HTMLInputElement;

const psoSpaceMinInputRef = document.querySelector('input[name=psoSpaceMin]') as
                    HTMLInputElement;
const psoSpaceMaxInputRef = document.querySelector('input[name=psoSpaceMax]')
                    as HTMLInputElement;
const psoWeightInputRef = document.querySelector('input[name=psoWeight]') as
                    HTMLInputElement;
const psoMaxIterationInputRef =
                    document.querySelector('input[name=psoMaxIteration]') as
                    HTMLInputElement;
const psoParticleCountInputRef =
                    document.querySelector('input[name=psoParticleCount]') as
                    HTMLInputElement;
const psoConfidenceInputRef = document.querySelector('input[name=psoConfidence]')
                    as HTMLInputElement;
const psoSelfConfidenceInputRef =
                    document.querySelector('input[name=psoSelfConfidence]') as
                    HTMLInputElement;
const psoStopMseInputRef = document.querySelector('input[name=psoStopMse]') as
                    HTMLInputElement;
const psoStopAferInputRef = document.querySelector('input[name=psoStopAfer]') as
                    HTMLInputElement;

const ctaExecuteButtonRef = document.getElementById('ctaExecute') as
                    HTMLButtonElement;
const ctaFakeLoadingButtonRef = document.getElementById('ctaFakeLoading') as
                    HTMLButtonElement;

const ftsChartRef = document.getElementById('ftsChart') as HTMLCanvasElement;
const ftsPsoChartRef = document.getElementById('ftsPsoChart') as
                    HTMLCanvasElement;
```

```
function formRequirementCheck() {
    if (!configFormRef.checkValidity()) {
        ctaExecuteButtonRef.disabled = true;
        ctaExecuteButtonRef.classList.add('disabled');
    } else {
        ctaExecuteButtonRef.disabled = false;
        ctaExecuteButtonRef.classList.remove('disabled');
    }
}

stageClientRef.checked = true;
stageClientRef.addEventListener('click', () => {
    strategy = 'client';
    stageClientRef.checked = true;
});
stageServerRef.addEventListener('click', () => {
    strategy = 'server';
    stageServerRef.checked = true;
});
ftsMinMarginInputRef.addEventListener('input', () => {
    ftsMaxMarginInputRef.min = ftsMinMarginInputRef.value;
});
ftsMaxMarginInputRef.addEventListener('input', () => {
    ftsMinMarginInputRef.max = ftsMaxMarginInputRef.value;
});
psoSpaceMinInputRef.addEventListener('input', () => {
    psoSpaceMaxInputRef.min = psoSpaceMinInputRef.value;
});
psoSpaceMaxInputRef.addEventListener('input', () => {
    psoSpaceMinInputRef.max = psoSpaceMaxInputRef.value;
});
dataSourceInputRef.addEventListener('input', () => {
    try {
        const files = dataSourceInputRef.files;
        if (!files) {
            return;
        }
        let target = [...files].find(current => current.type === 'text/csv');
        if (!target) {
            throw new TypeError('Only accepting csv file as dataset source.');
        }
        const reader = new FileReader();
        reader.addEventListener('load', () => {
            try {
                const data = reader.result as string;
                if (!data) {
                    throw new EvalError('Dataset source is empty.');
                }
                const pairs = data.replace(/(\r\n)/g, '\n').replace(/\r/g, '\n').split('\n');
                const header = pairs.shift();
                if (!header) {
                    throw new EvalError('Unrecognized dataset, header is empty.');
                }
                const splitHeader = header.split(',');
                const keyIndex = splitHeader.findIndex(head => head.toLowerCase() ===
                        'key');
```

```
            if (keyIndex < 0) {
                throw new EvalError('"Key" header is not exists.');
            }
            const valueIndex = splitHeader.findIndex(head => head.toLowerCase() ===
                    'value');
            if (valueIndex < 0) {
                throw new EvalError('"Value" header is not exists.');
            }
            const rawDataset = new Array<{ key: string; value: string; }>();
            for (const line of pairs) {
                const splitted = line.split(',');
                rawDataset.push({
                    key: splitted[keyIndex],
                    value: splitted[valueIndex],
                });
            }
            parsedDataset.length = 0;
            rawDataset.forEach((rawPair, index) => {
                if (!rawPair.key) {
                    throw new EvalError(`Key at line ${index + 1} were empty.`);
                }
                const tryCast = Number(rawPair.value);
                if (isNaN(tryCast)) {
                    throw new EvalError(`Value for key "${rawPair.key}" at line ${index +
                        1} was not a number.`);
                }
                parsedDataset.push({
                    key: rawPair.key,
                    value: tryCast,
                });
            });
            const min = Math.min(...parsedDataset.map(v => v.value));
            const max = Math.max(...parsedDataset.map(v => v.value));
            ftsMinMarginInputRef.valueAsNumber = Number((min * 0.1).toFixed(4));
            ftsMaxMarginInputRef.valueAsNumber = Number((max * 0.1).toFixed(4));
            ftsIntervalInputRef.valueAsNumber = Number((((max * 1.1) - (min * 0.9)) /
                    10).toFixed(4));
        } catch (e) {
            alert(e.toString());
            dataSourceInputRef.value = '';
        }
    });
    reader.readAsText(target);
} catch (e) {
    alert(e.toString());
    dataSourceInputRef.value = '';
}
});
ctaExecuteButtonRef.addEventListener('click', async () => {
    const valid = configFormRef.reportValidity();
    if (valid) {
        ctaExecuteButtonRef.classList.add('d-none');
        ctaFakeLoadingButtonRef.classList.remove('d-none');
        const config = {
            fts: {
                minMargin: ftsMinMarginInputRef.valueAsNumber,
                maxMargin: ftsMaxMarginInputRef.valueAsNumber,
```

```
      interval: ftsIntervalInputRef.valueAsNumber,
    },
    pso: {
      spaces: {
        min: psoSpaceMinInputRef.valueAsNumber,
        max: psoSpaceMaxInputRef.valueAsNumber,
      },
      weight: psoWeightInputRef.valueAsNumber,
      maxIteration: psoMaxIterationInputRef.valueAsNumber,
      particleCount: psoParticleCountInputRef.valueAsNumber,
      swarmConfidence: psoConfidenceInputRef.valueAsNumber,
      selfConfidence: psoSelfConfidenceInputRef.valueAsNumber,
      stopCriteria: {
        mse: psoStopMseInputRef.valueAsNumber,
        afer: psoStopAferInputRef.valueAsNumber,
      },
    },
};

try {
    const fts = {
      result: new Array<{ key: string; value: number; predicted: number; }>(),
      mse: 0,
      afer: 0,
    };
    const ftsPso = {
      result: new Array<{ key: string; value: number; predicted: number; }>(),
      mse: 0,
      afer: 0,
      lastIteration: 0,
      best: 0,
    };
    switch (strategy) {
      case 'client': {
        const min = Math.min(...parsedDataset.map(v => v.value));
        const max = Math.max(...parsedDataset.map(v => v.value));
        const engine = new FTS(parsedDataset, {
          minMargin: config.fts.minMargin,
          maxMargin: config.fts.maxMargin,
          interval: config.fts.interval,
        });
        engine.train();
        fts.result = engine.test();
        const forecasted = fts.result.slice(0, fts.result.length - 1).map(v =>
            v.predicted);
        const actual = fts.result.slice(1, fts.result.length).map(v => v.value);
        fts.mse = meanSquaredError(actual, forecasted);
        fts.afer = averageForecastingErrorRate(actual, forecasted);

        const swarm = new Swarm<{ mse: number; afer: number }>({
          spaces: config.pso.spaces,
          weight: config.pso.weight,
          maxIteration: config.pso.maxIteration,
          particleCount: config.pso.particleCount,
          swarmConfidence: config.pso.swarmConfidence,
          selfConfidence:config.pso.selfConfidence,
          fitnessFunction: (n) => {
```

```
                const subFts = new FTS(parsedDataset, {
                    minMargin: config.fts.minMargin,
                    maxMargin: config.fts.maxMargin,
                    interval: ((max * 1.1) - (min * 0.9)) / Math.abs(n),
                });
                subFts.train();
                const subResult = subFts.test();
                const subForecasted = subResult.slice(0, subResult.length - 1).map(v
                 => v.predicted);
                const subActual = subResult.slice(1, subResult.length).map(v =>
                 v.value);
                return {
                    mse: meanSquaredError(subActual, subForecasted),
                    afer: averageForecastingErrorRate(subActual, subForecasted),
                };
            },
            selectorFunction: (results) => {
                let bestIndex = 0;
                for (let i = 1; i < results.length; i++) {
                    if (results[i].mse < results[bestIndex].mse || results[i].afer <
                     results[bestIndex].afer) {
                        bestIndex = i;
                    }
                }
                return bestIndex;
            },
            stopCriteria: (n) => n.mse <= config.pso.stopCriteria.mse ||
                 Math.abs(n.afer) <= config.pso.stopCriteria.afer,
        });
        ftsPso.lastIteration = await new Promise(resolve => {
            setTimeout(() => resolve(swarm.optimize()), 1);
        });
        const optimizedEngine = new FTS(parsedDataset, {
            minMargin: config.fts.minMargin,
            maxMargin: config.fts.maxMargin,
            interval: ((max * 1.1) - (min * 0.9)) / swarm.bestPosition,
        });
        optimizedEngine.train();
        const optimizedResult = optimizedEngine.test();
        const optimizedForecast = optimizedResult.slice(0, optimizedResult.length
                - 1).map(v => v.predicted);
        const optimizedActual = optimizedResult.slice(1,
                optimizedResult.length).map(v => v.value);
        ftsPso.result = optimizedResult;
        ftsPso.mse = meanSquaredError(optimizedActual, optimizedForecast);
        ftsPso.afer = averageForecastingErrorRate(optimizedActual,
                optimizedForecast);
        ftsPso.best = swarm.bestPosition;
        break;
    }
    case 'server': {
        break;
    }
}

const ftsChart = ftsChartRef['chart'];
ftsChart.options.title.text = `Fuzzy Time Series Result: (interval = 10)`;
```

```
            ftsChart.data.labels = fts.result.map(v => v.key);
            ftsChart.data.datasets = [
               { label: 'Data', borderColor: '#4e73df', data: fts.result.map(v => v.value) },
               { label: 'Prediction', borderColor: '#5bbf21', data: fts.result.map(v =>
                        v.predicted) },
            ];
            ftsChart.update();
            const ftsPsoChart = ftsPsoChartRef['chart'];
            ftsPsoChart.options.title.text = `Fuzzy Time Series Result Optimized using
                        Particle Swarm Optimization: (interval = ${ftsPso.best})`;
            ftsPsoChart.data.labels = ftsPso.result.map(v => v.key);
            ftsPsoChart.data.datasets = [
               { label: 'Data', borderColor: '#4e73df', data: ftsPso.result.map(v => v.value) },
               { label: 'Prediction', borderColor: '#5bbf21', data: ftsPso.result.map(v =>
                        v.predicted) },
            ];
            ftsPsoChart.update();

            configFormRef.classList.add('d-none');
            resultRef.classList.remove('d-none');
            retryRef.classList.remove('d-none');
            ctaFakeLoadingButtonRef.classList.add('d-none');
            ctaExecuteButtonRef.classList.remove('d-none');
         } catch (e) {
            ctaFakeLoadingButtonRef.classList.add('d-none');
            ctaExecuteButtonRef.classList.remove('d-none');
            alert(e.toString());
         }
      }
   });
   retryRef.addEventListener('click', () => {
      configFormRef.classList.remove('d-none');
      resultRef.classList.add('d-none');
      retryRef.classList.add('d-none');
      ctaFakeLoadingButtonRef.classList.add('d-none');
      ctaExecuteButtonRef.classList.remove('d-none');
   });

   configFormRef.classList.remove('d-none');
```

swarm.ts

```
import { Particle } from "./particle";
import { FitFuncDelegate, SelectorFuncDelegate, StopCriteria } from "./utils";

interface SwarmOption<TFit = any> {
   spaces: {
      min: number;
      max: number;
   };
   weight?: number;
   maxIteration: number;
   particleCount: number;
   swarmConfidence?: number;
   selfConfidence?: number;
   fitnessFunction: FitFuncDelegate<TFit>;
   selectorFunction: SelectorFuncDelegate<TFit>;
```

```
    stopCriteria: StopCriteria<TFit>;
}

export class Swarm<TFit = any> {
    private _maxIteration: number;
    private _particles: Particle<TFit>[];
    private _fitFn: FitFuncDelegate<TFit>;
    private _selectFn: SelectorFuncDelegate<TFit>;
    private _stopCriteria: StopCriteria<TFit>;
    private _bestFit: TFit;
    private _bestPosition: number;
    private _swarmConfidence: number;
    private _w: number;

    public get bestPosition() {
        return this._bestPosition;
    }

    public get bestFit() {
        return this._bestFit;
    }

    constructor(options: SwarmOption<TFit>) {
        if (options.weight && (options.weight >= 1 || options.weight <= 0)) {
            console.warn('Weight out of boundaries, generating random weight...');
        }
        this._w = options.weight && options.weight > 0 && options.weight < 1 ?
                    options.weight : Math.random();
        this._maxIteration = options.maxIteration;
        this._fitFn = options.fitnessFunction;
        this._selectFn = options.selectorFunction;
        this._stopCriteria = options.stopCriteria;
        this._swarmConfidence = options.swarmConfidence || Math.random();
        this._particles = new Array<null>(options.particleCount).fill(null).map(() => new
                    Particle<TFit>({
            min: options.spaces.min,
            max: options.spaces.max,
            fitFn: this._fitFn,
            selectorFn: this._selectFn,
            selfConfidence: options.selfConfidence,
            weight: this._w,
        }));
        const bestParticle = this._selectFn.call(this, this._particles.map(v =>
                    v.bestFitResult));
        this._bestFit = this._particles[bestParticle].bestFitResult;
        this._bestPosition = this._particles[bestParticle].bestPosition;
    }

    public optimize() {
        let stop = this._stopCriteria.call(this, this._bestFit);
        let currentIteration = 0;
        while (!stop && currentIteration < this._maxIteration) {
            for (const particle of this._particles) {
                particle.update(this._swarmConfidence, this._bestPosition);
            }
            const bestParticle = this._selectFn.call(this, this._particles.map(v =>
                    v.bestFitResult));
```

```
            this._bestFit = this._particles[bestParticle].bestFitResult;
            this._bestPosition = this._particles[bestParticle].bestPosition;
            stop = this._stopCriteria.call(this, this._bestFit);
            currentIteration += 1;
        }
        return currentIteration;
    }
}
```

Utils.ts

```
function baseLookup(interval: number) {
    return interval <= 0.1 ? 0.1 : (10 ** Math.ceil(Math.log10(interval) - 1));
}

export function averageInterval(dataset: number[]) {
    const range = dataset.map((v, i, a) => i === 0 ? null : Math.abs(v - a[i - 1])).filter(v
                => v !== null) as number[];
    const rangeAverage = range.reduce((p, c) => p + c, 0) / range.length;
    const halfAverage = rangeAverage / 2;
    return baseLookup(halfAverage);
}

export function meanSquaredError(actual: number[], forecast: number[]) {
    if (actual.length !== forecast.length) {
        throw new RangeError('Actual data and forecast data has different length');
    }
    return actual.map((v, i) => (v - forecast[i]) ** 2).reduce((p, c) => p + c, 0) /
                actual.length;
}

export function averageForecastingErrorRate(actual: number[], forecast: number[]) {
    if (actual.length !== forecast.length) {
        throw new RangeError('Actual data and forecast data has different length');
    }
    return actual.map((v, i) => Math.abs(v - forecast[i]) / v).reduce((p, c) => p + c, 0) /
                actual.length;
}

export type FitFuncDelegate<TResult = any> = (position: number) => TResult;
export type SelectorFuncDelegate<TResult = any> = (fitResult: TResult[]) => number;
export type StopCriteria<TResult = any> = (bestResult: TResult) => boolean;
```

css

```
bs-icon {
  --bs-icon-size: .75rem;
  display: flex;
  flex-shrink: 0;
  justify-content: center;
  align-items: center;
  font-size: var(--bs-icon-size);
  width: calc(var(--bs-icon-size) * 2);
  height: calc(var(--bs-icon-size) * 2);
  color: var(--bs-primary);
}

.bs-icon-xs {
```

```
  --bs-icon-size: 1rem;
  width: calc(var(--bs-icon-size) * 1.5);
  height: calc(var(--bs-icon-size) * 1.5);
}

.bs-icon-sm {
  --bs-icon-size: 1rem;
}

.bs-icon-md {
  --bs-icon-size: 1.5rem;
}

.bs-icon-lg {
  --bs-icon-size: 2rem;
}

.bs-icon-xl {
  --bs-icon-size: 2.5rem;
}

.bs-icon.bs-icon-primary  {
  color: var(--bs-white);
  background: var(--bs-primary);
}

.bs-icon.bs-icon-primary-light {
  color: var(--bs-primary);
  background: rgba(var(--bs-primary-rgb), .2);
}

.bs-icon.bs-icon-semi-white {
  color: var(--bs-primary);
  background: rgba(255, 255, 255, .5);
}

.bs-icon.bs-icon-rounded {
  border-radius: .5rem;
}

.bs-icon.bs-icon-circle {
  border-radius: 50%;
}
```