Transition Banking Analytics

Undertaken at: Nucleus Software

Mentor: Abhishek Gupta

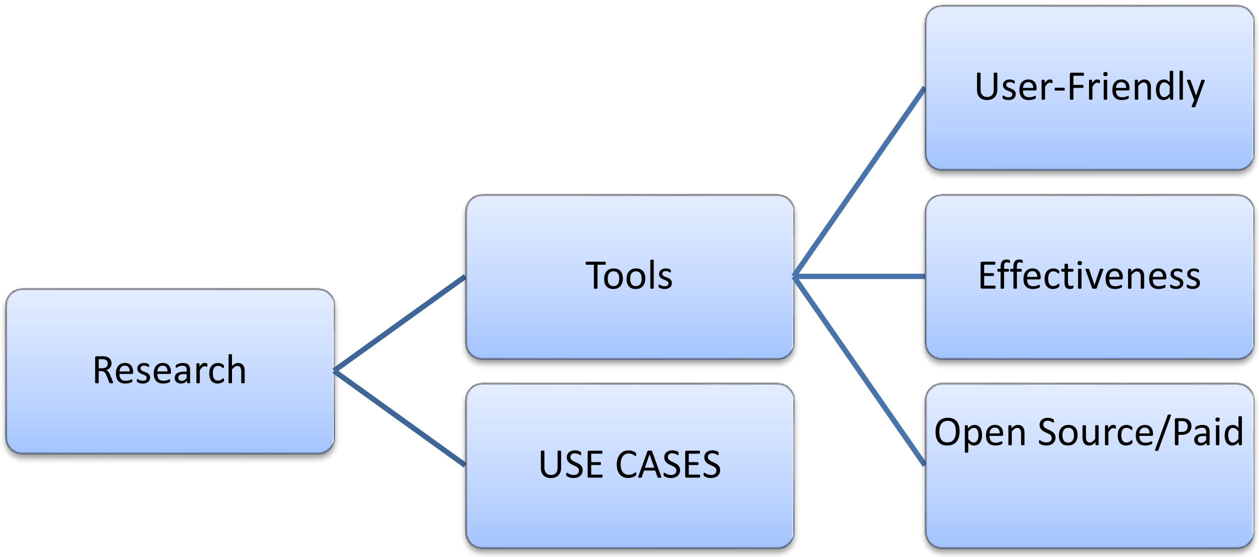
Rishikesh

Interns: Prerna Kakkar

Aayush Mittal

IDEA BEHIND THE PROJECT

The idea behind this project is that user/bank will provide the app with data. After preprocessing the data the result will be forecasted for some specified time period. For instance, the data provided by the user is from 1-Dec-2016 to 15-May-2017 and he wish to know his transaction amount on 18-April-2018; so this library will forecast the result for it. Other part we are doing in this project is to preprocess the data and let the bank know what is the share of each customer in transaction using pie charts and frequency of various modes in a day using bar graphs.



Project Overview

AIM OF THE PROJECT

- To create a forecasting library
- Creating an example web app to illustrate the use of the library
- Follow up: Integrating the library with products for discussed use cases

USE CASES

- Effective capital management by identifying working capital cycles
- Enhancing capital protection
- Identifying bottlenecks and provide statistics for efficient capital allocation
- Identifying periods of time with high-traffic for better load balancing
- Fraud detection by identifying abnormally out of trend data

REQUIREMENTS

1) jdk 1.7 or above

2) d3.js ,c3.js and papaparse.js

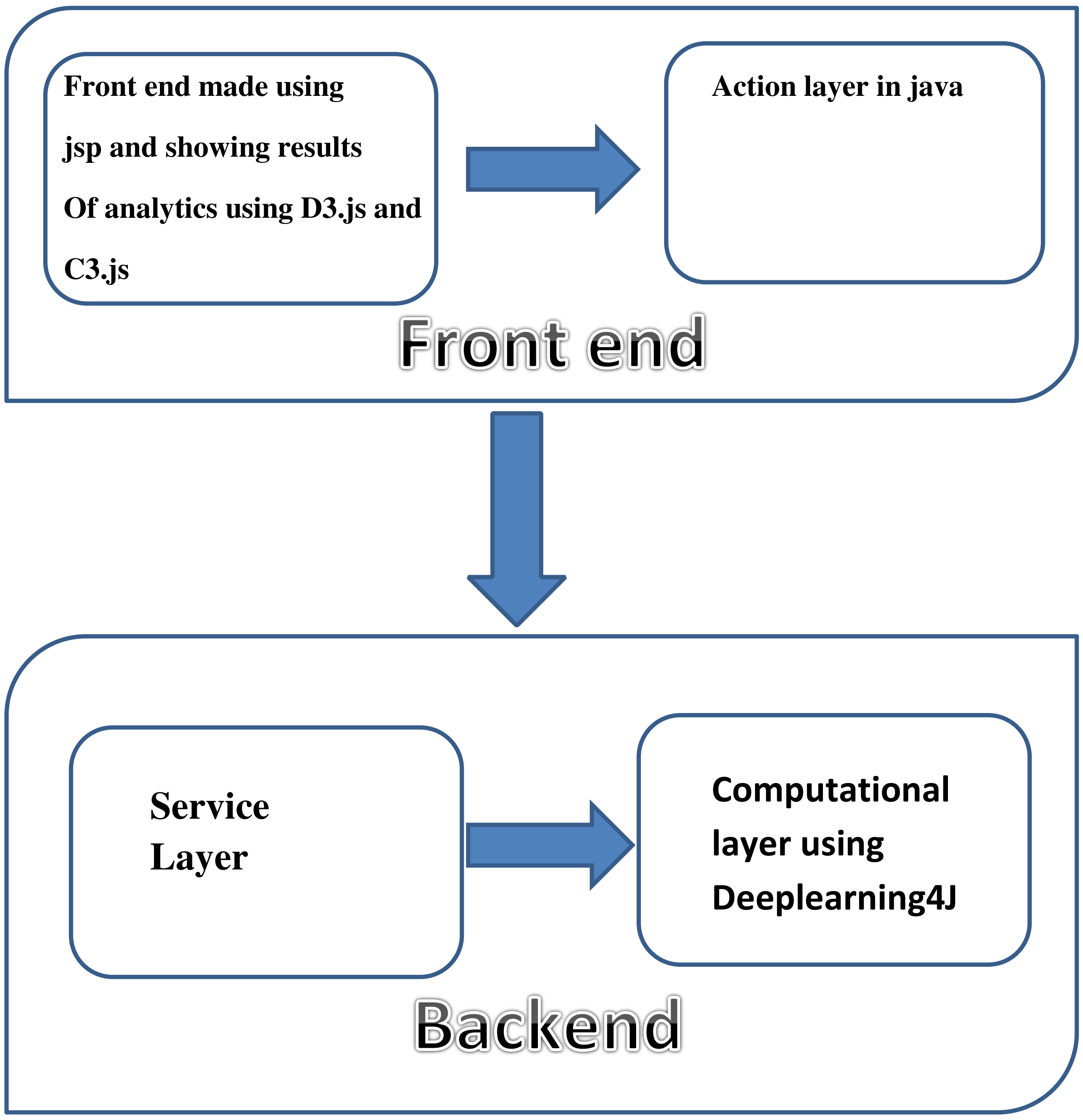
3) Deeplearning4j jar files if maven is not working

4) Apache Maven (automated build and dependency manager)

5) Eclipse

6) Apache Tomcat-7.0

Layers in Project


Front end made using jsp and showing results Of analytics using D3.js and C3.js
Action layer in java
Front end
Service Layer
Computational layer using Deeplearning4J
Backend

Frontend

D3.js is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3’s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

To link directly to the latest release, copy this snippet:

```
<script src="https://d3js.org/d3.v4.min.js"></script>
```

C3.js is a D3.js based library. C3 makes it easy to generate D3-based charts by wrapping the code required to construct the entire chart. We don't need to write D3 code any more. C3 gives some classes to each element when generating, so you can define a custom style by the class and it's possible to extend the structure directly by D3. C3 provides a variety of APIs and callbacks to access the state of the chart. By using them, you can update the chart even after it's rendered.

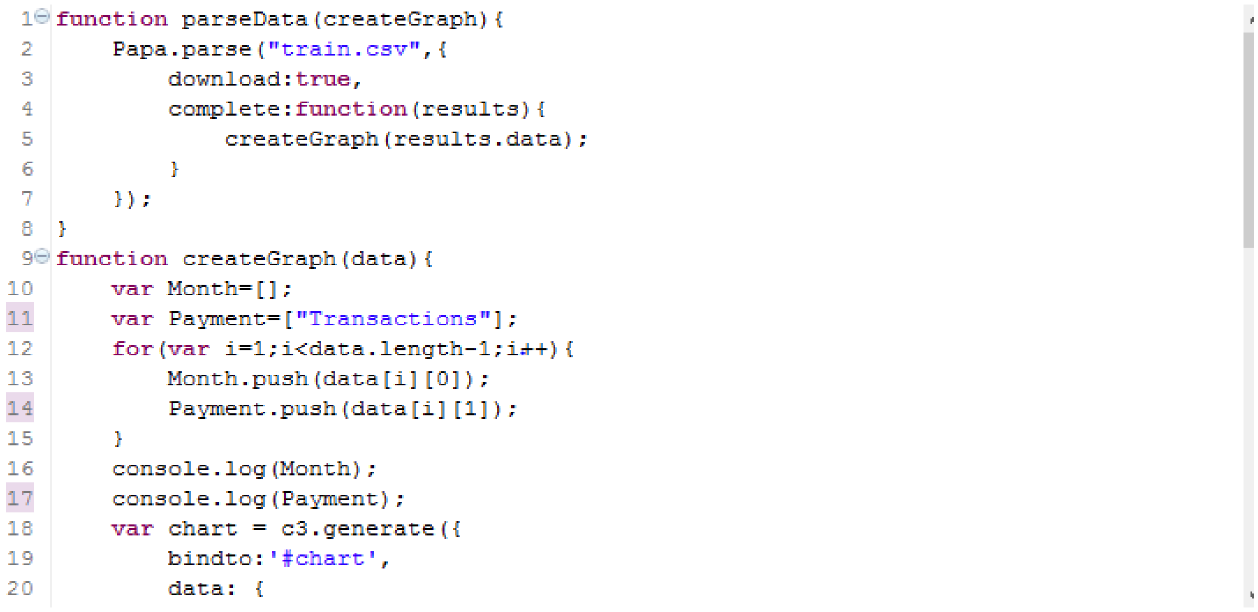
Papa Parse is a powerful parser used for parsing csv in javascript.

List of files

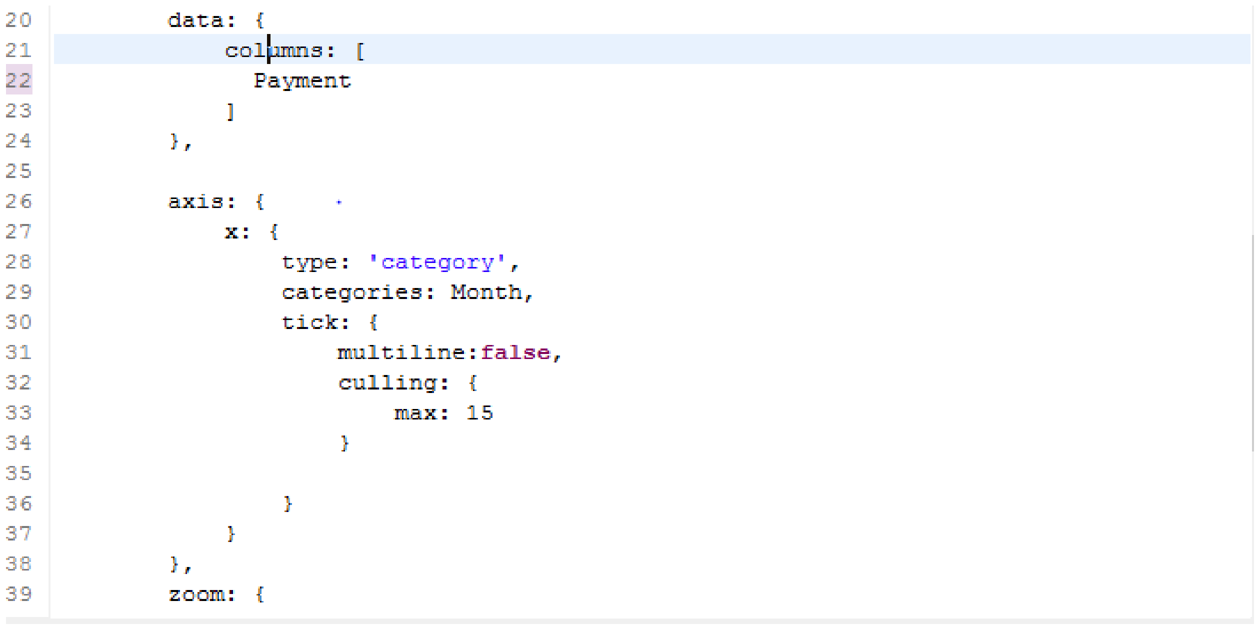
1. v1.jsp (jsp file for creating line charts based on prediction)
2. NewFile.jsp (jsp file for creating pie charts showing transaction share of each customer)
3. Bar.jsp (jsp file for creating bar graph between various modes of payment)
4. loader.js (script for creating loader with random tags)
5. creategraph.js(script for generating line chart)
6. createpie.js(script for creating pie chart)
7. createbar.js(script for creating bar charts)
8. filter1.ipynb (python script used for showing various data analysis)
9. filter2.ipynb(python script used for preprocessing the data for creating pie charts)

Code

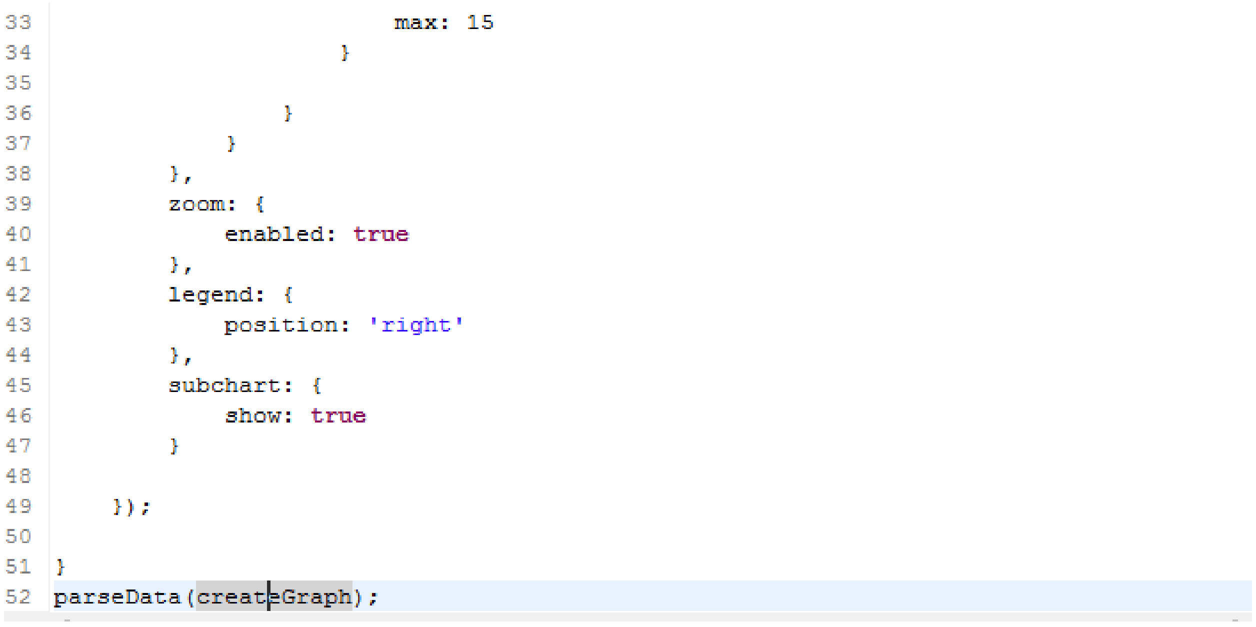
1) creategraph.js



```
function parseData(createGraph){
    Papa.parse("train.csv",{
        download:true,
        complete:function(results){
            createGraph(results.data);
        }
    });
}
function createGraph(data){
    var Month=[];
    var Payment=["Transactions"];
    for(var i=1;i<data.length-1;i++){
        Month.push(data[i][0]);
        Payment.push(data[i][1]);
    }
    console.log(Month);
    console.log(Payment);
    var chart = c3.generate({
        bindto:'#chart',
        data: {
```



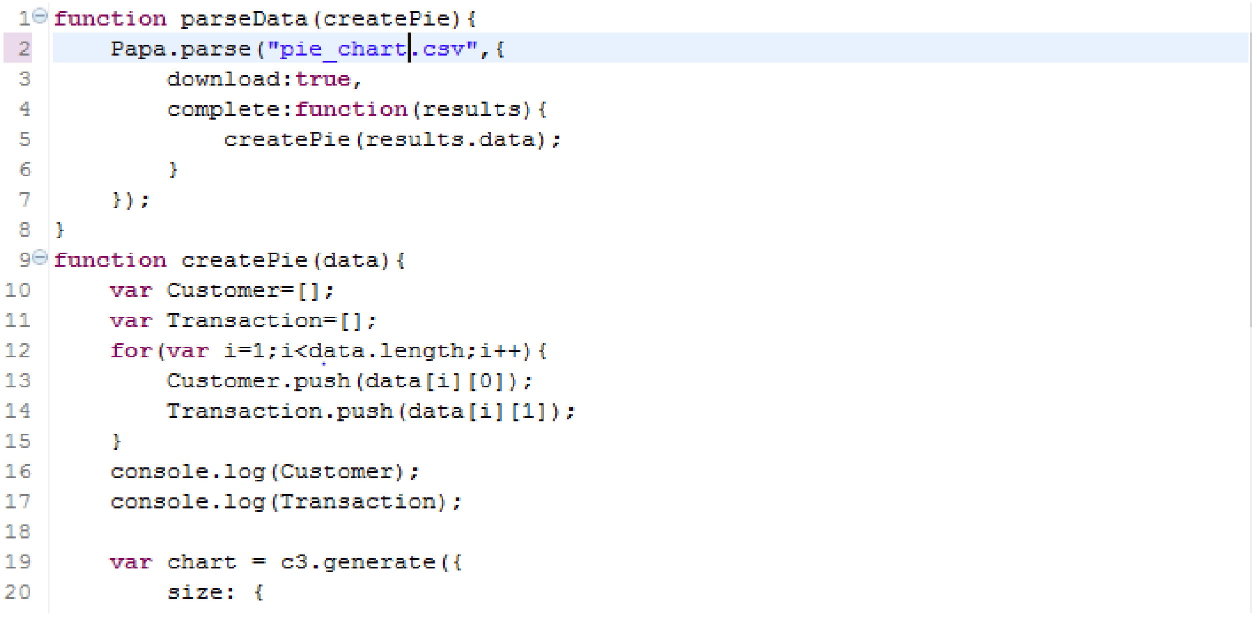
```
        data: {
            columns: [
              Payment
            ]
        },

        axis: {
            x: {
                type: 'category',
                categories: Month,
                tick: {
                    multiline:false,
                    culling: {
                        max: 15
                    }

                }
            }
        },
        zoom: {
```



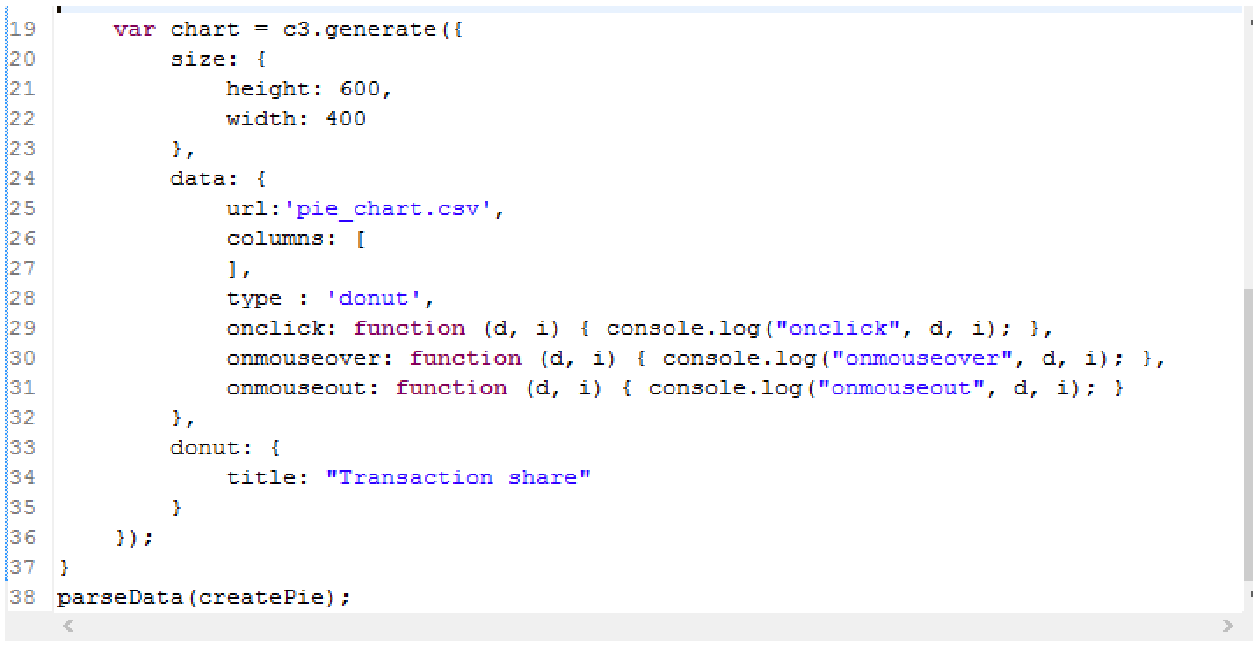
```
33                         max: 15
34                     }
35 
36                 }
37             }
38         },
39         zoom: {
40             enabled: true
41         },
42         legend: {
43             position: 'right'
44         },
45         subchart: {
46             show: true
47         }
48 
49     });
50 
51 }
52 parseData(createGraph);
```

In this file just change the location of the input csv file in line 2 and see the results reflected in v1.jsp.

2) createpie.js

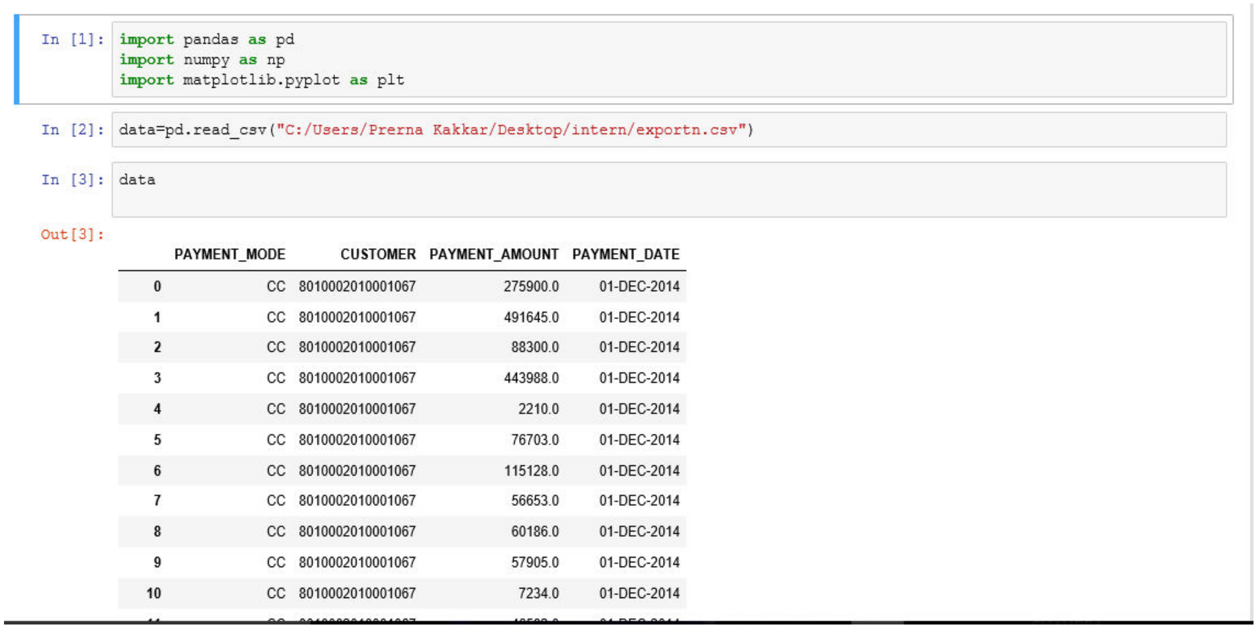


```
1  function parseData(createPie){
2      Papa.parse("pie_chart.csv",{
3          download:true,
4          complete:function(results){
5              createPie(results.data);
6          }
7      });
8  }
9  function createPie(data){
10     var Customer=[];
11     var Transaction=[];
12     for(var i=1;i<data.length;i++){
13         Customer.push(data[i][0]);
14         Transaction.push(data[i][1]);
15     }
16     console.log(Customer);
17     console.log(Transaction);
18 
19     var chart = c3.generate({
20         size: {
```



```
    var chart = c3.generate({
        size: {
            height: 600,
            width: 400
        },
        data: {
            url:'pie_chart.csv',
            columns: [
            ],
            type : 'donut',
            onclick: function (d, i) { console.log("onclick", d, i); },
            onmouseover: function (d, i) { console.log("onmouseover", d, i); },
            onmouseout: function (d, i) { console.log("onmouseout", d, i); }
        },
        donut: {
            title: "Transaction share"
        }
    });
}
parseData(createPie);
```

Replace the file pie_chart.csv with the file generated from pre processing using python script as shown below:



```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: data=pd.read_csv("C:/Users/Prerna Kakkar/Desktop/intern/exportn.csv")

In [3]: data

Out[3]:
```

	PAYMENT_MODE	CUSTOMER	PAYMENT_AMOUNT	PAYMENT_DATE
0	CC	8010002010001067	275900.0	01-DEC-2014
1	CC	8010002010001067	491645.0	01-DEC-2014
2	CC	8010002010001067	88300.0	01-DEC-2014
3	CC	8010002010001067	443988.0	01-DEC-2014
4	CC	8010002010001067	2210.0	01-DEC-2014
5	CC	8010002010001067	76703.0	01-DEC-2014
6	CC	8010002010001067	115128.0	01-DEC-2014
7	CC	8010002010001067	56653.0	01-DEC-2014
8	CC	8010002010001067	60186.0	01-DEC-2014
9	CC	8010002010001067	57905.0	01-DEC-2014
10	CC	8010002010001067	7234.0	01-DEC-2014

Replace the path in line 2 with the original data

65001 rows × 4 columns

```
In [5]:  df = pd.DataFrame(data, columns = ['PAYMENT_MODE', 'CUSTOMER', 'PAYMENT_AMOUNT', 'PAYMENT_DATE'])
```

```
In [6]:  df.head()
```

Out[6]:

	PAYMENT_MODE	CUSTOMER	PAYMENT_AMOUNT	PAYMENT_DATE
0	CC	8010002010001067	275900.0	01-DEC-2014
1	CC	8010002010001067	491645.0	01-DEC-2014
2	CC	8010002010001067	88300.0	01-DEC-2014
3	CC	8010002010001067	443988.0	01-DEC-2014
4	CC	8010002010001067	2210.0	01-DEC-2014

```
In [7]:  list(set(df.CUSTOMER))
```

```
Out[7]:  [437768,
          505355,
          2077714,
          2081811,
          2079764,
          8007002010000406,
          2081820,
          2076708,
          2078257,
```

```
In [66]:  lis=[df]
          for i in a:
                df1=pd.DataFrame(df[df.CUSTOMER==i])
                lis.append(df1)
```

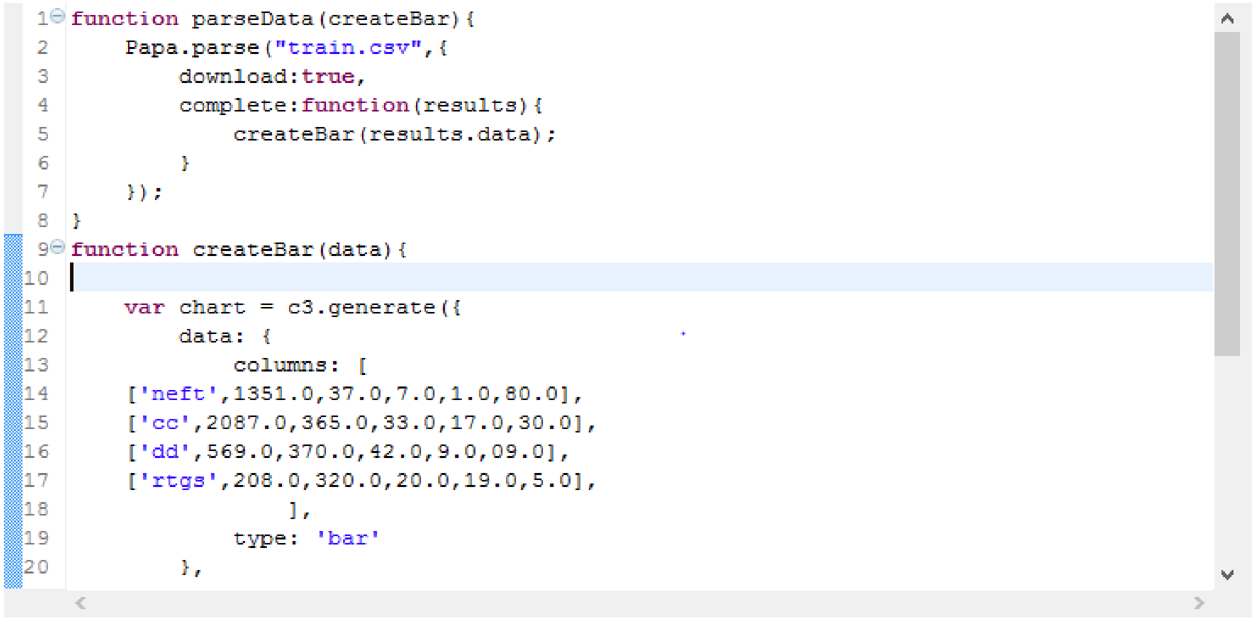
```
In [75]:  lis[0][lis[0].PAYMENT_DATE=='01-DEC-2014'].PAYMENT_AMOUNT.sum()
```

```
Out[75]:  195991768.0
```

```
In [134]:  d1=pd.DataFrame()
```

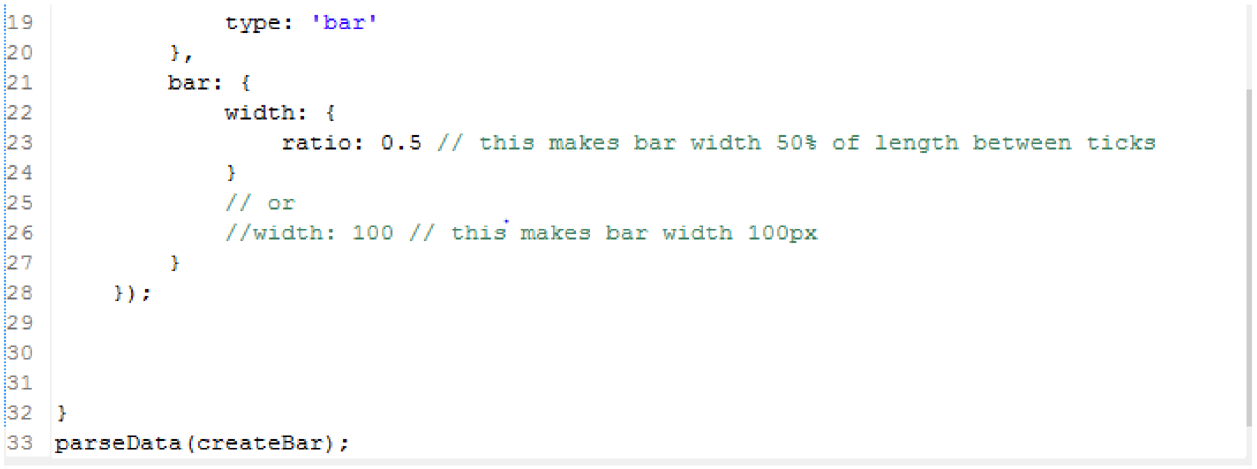
```
In [140]:  for i in lis:
                v=i.PAYMENT_DATE.unique().tolist()
                list3=[]
                for j in v:
                    c=i[i.PAYMENT_DATE==j].PAYMENT_AMOUNT.sum()
                    list3.append(c)
                a=i.CUSTOMER.unique().tolist();
                d2=pd.DataFrame({str(a[0]) : list3})
                result = pd.concat([d1, d2], axis=1)
                d1=result
```

3) createbar.js



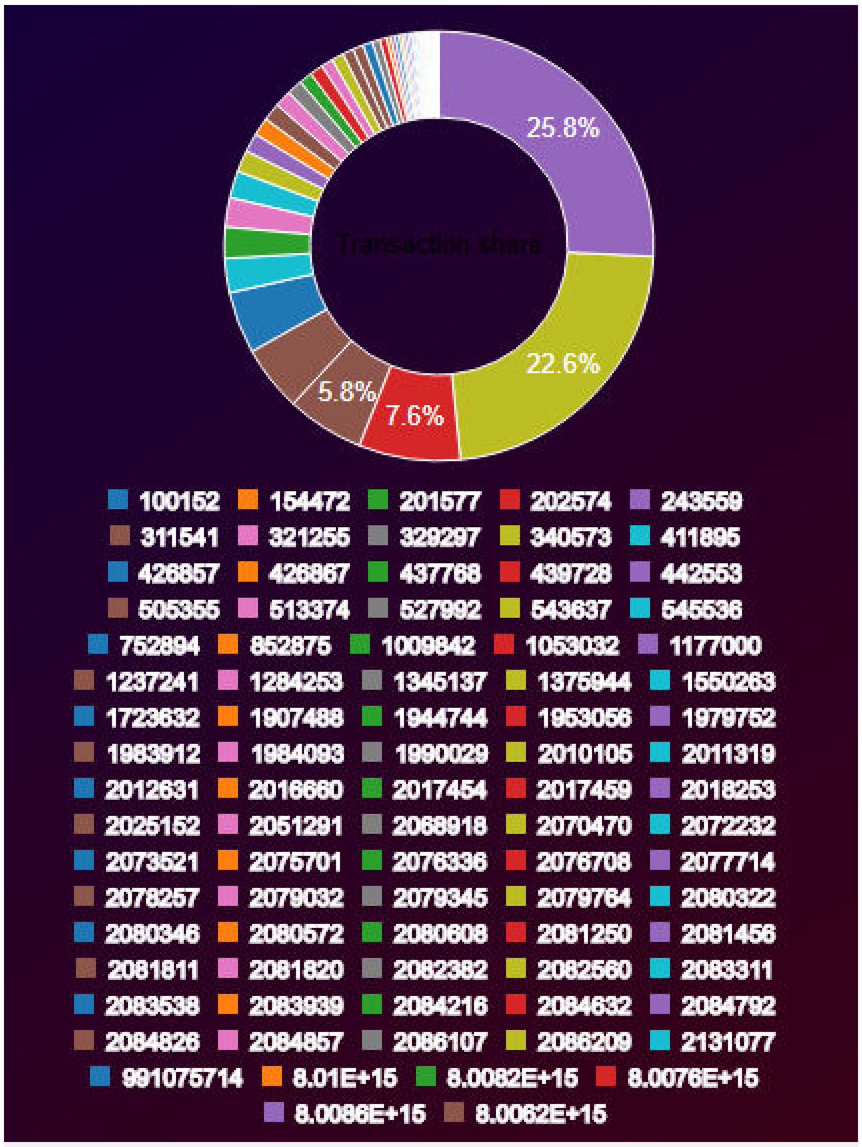
```
1  function parseData(createBar){
2      Papa.parse("train.csv",{
3          download:true,
4          complete:function(results){
5              createBar(results.data);
6          }
7      });
8  }
9  function createBar(data){
10 
11     var chart = c3.generate({
12         data: {
13             columns: [
14     ['neft',1351.0,37.0,7.0,1.0,80.0],
15     ['cc',2087.0,365.0,33.0,17.0,30.0],
16     ['dd',569.0,370.0,42.0,9.0,09.0],
17     ['rtgs',208.0,320.0,20.0,19.0,5.0],
18                 ],
19             type: 'bar'
20         },
```

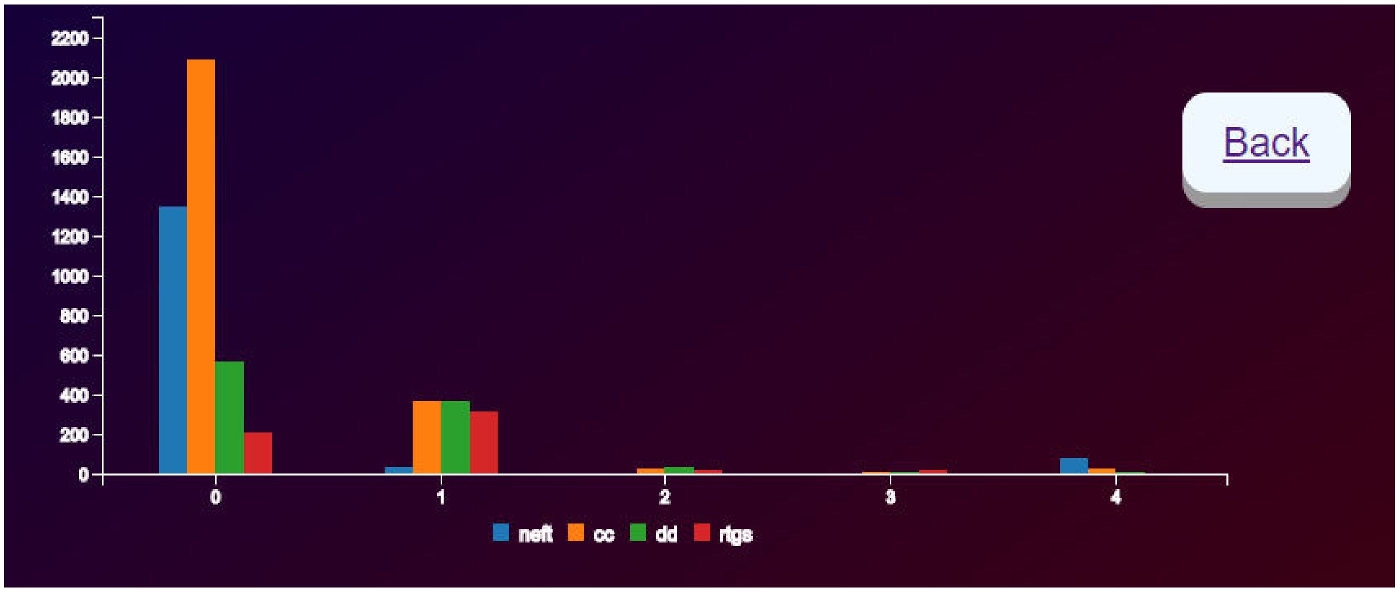
Replace the data coming from backend from line 14 to 17.

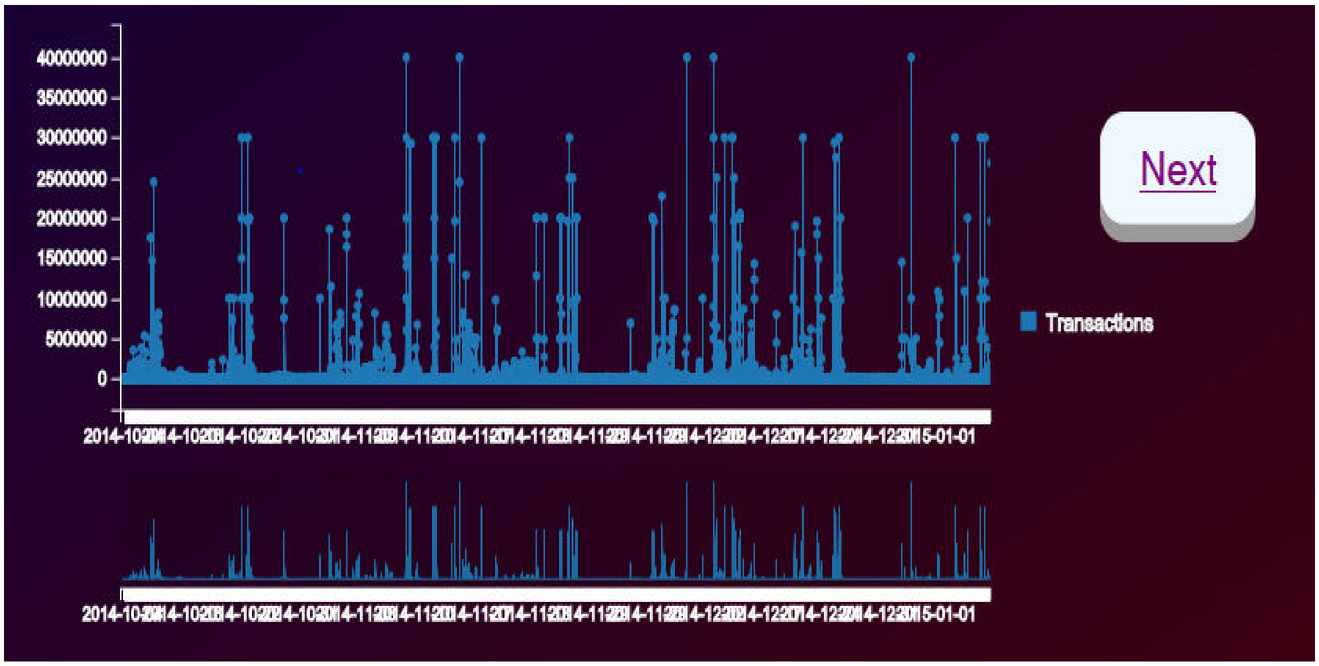


```
19             type: 'bar'
20         },
21         bar: {
22             width: {
23                 ratio: 0.5 // this makes bar width 50% of length between ticks
24             }
25             // or
26             //width: 100 // this makes bar width 100px
27         }
28     });
29 
30 
31 
32 }
33 parseData(createBar);
```

<u>Results from D3.js</u>




2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
neft
cc
dd
rtgs
Back

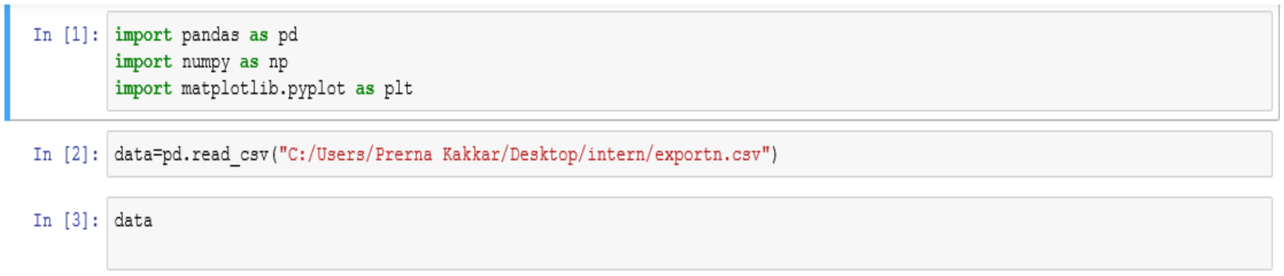

40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
Transactions
Next
2015-01-01

Data Analysis

Data analysis, also known as **analysis of data** or **data analytics**, is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.

Data mining is a particular data analysis technique that focuses on modeling and knowledge discovery for predictive rather than purely descriptive purposes, while business intelligence covers data analysis that relies heavily on aggregation, focusing on business information. In statistical applications data analysis can be divided into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data and CDA on confirming or falsifying existing hypotheses. Predictive analytics focuses on application of statistical models for predictive forecasting or classification, while text analytics applies statistical, linguistic, and structural techniques to extract and classify information from textual sources, a species of unstructured data. All are varieties of data analysis.

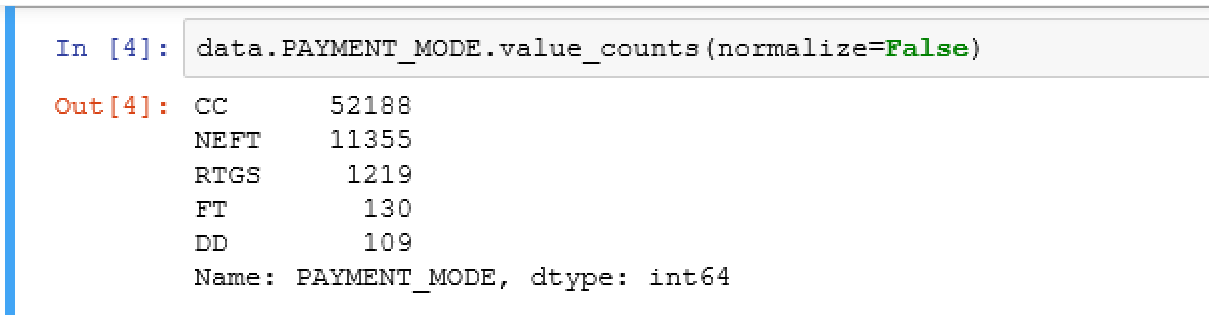
Data integration is a precursor to data analysis, and data analysis is closely linked to data visualization and data dissemination. The term *data analysis* is sometimes used as a synonym for data modeling.



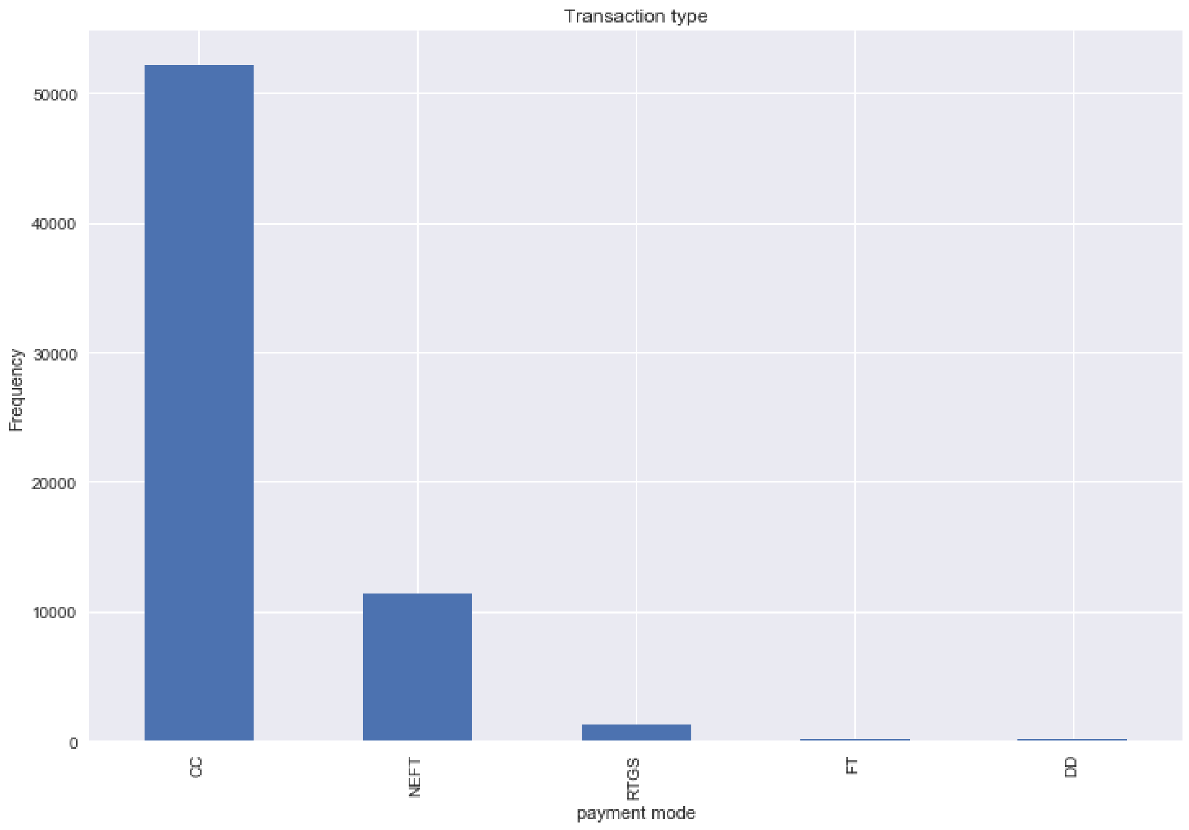
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: data=pd.read_csv("C:/Users/Prerna Kakkar/Desktop/intern/exportn.csv")

In [3]: data
```

Replace the path of csv file you want to do data analysis in In[2].

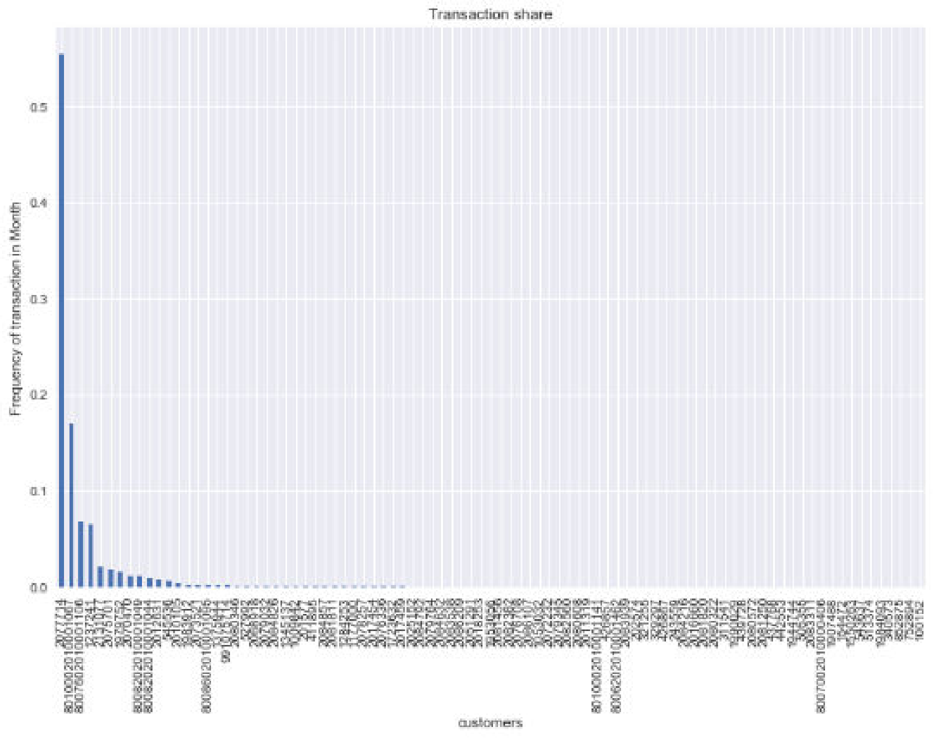
Results:



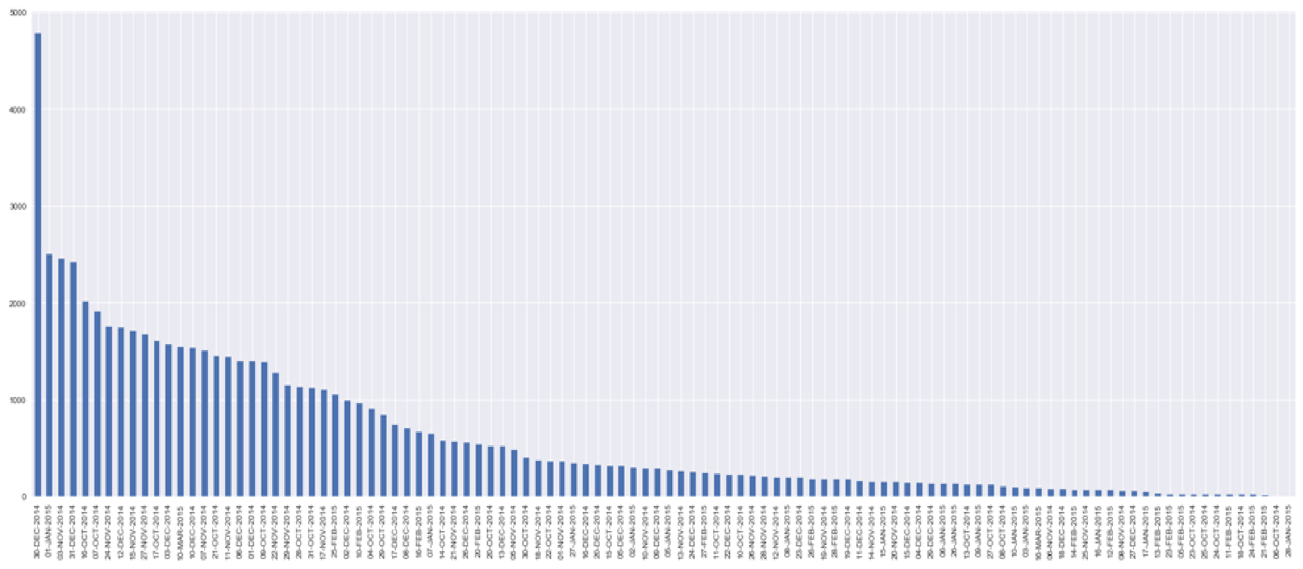
```
In [4]: data.PAYMENT_MODE.value_counts(normalize=False)

Out[4]: CC      52188
        NEFT    11355
        RTGS     1219
        FT        130
        DD        109
        Name: PAYMENT_MODE, dtype: int64
```



In [8]:
```
plt.figure(1,figsize=(12,8))
data.CUSTOMER.value_counts(normalize=True).plot(kind='bar')
plt.xlabel('customers')
plt.ylabel('Frequency of transaction in Month')
plt.title('Transaction share')
plt.show()
```



In [9]:
```
plt.figure(1,figsize=(30,10))
data.PAYMENT_DATE.value_counts(normalize=False).plot(kind='bar')
plt.show()
```

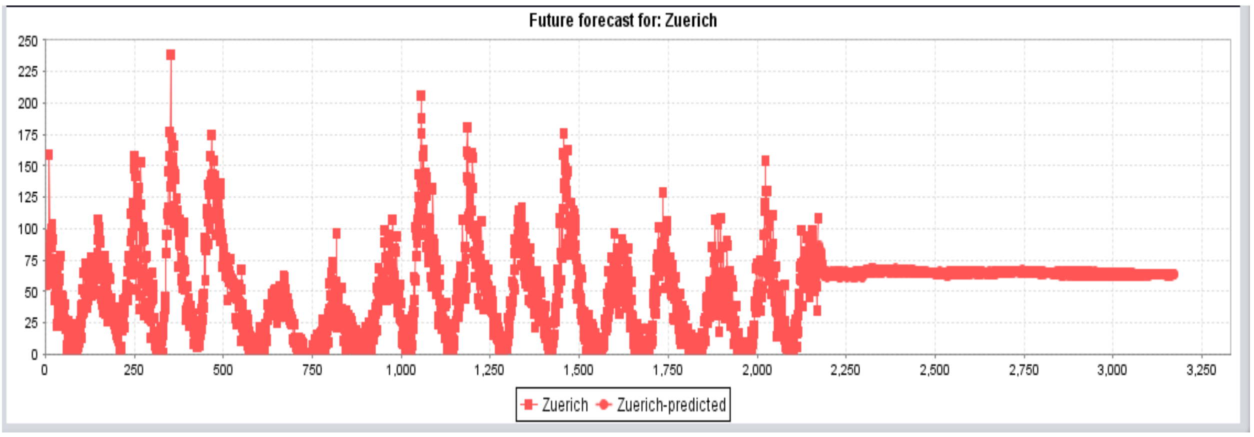


Research

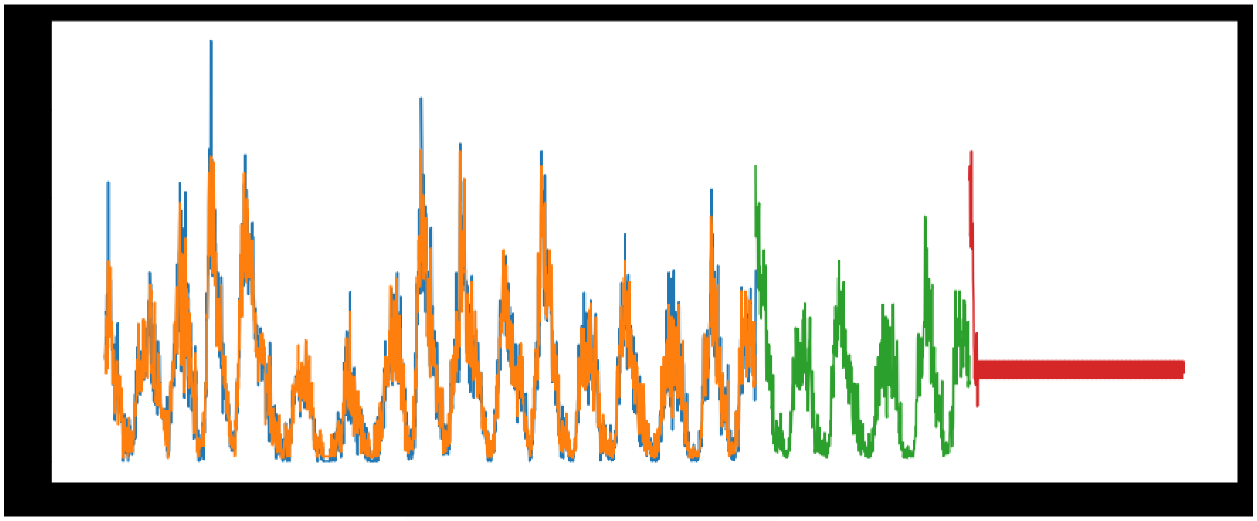
1) Data Mining Techniques:

Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

Results:



(Without feature extraction)



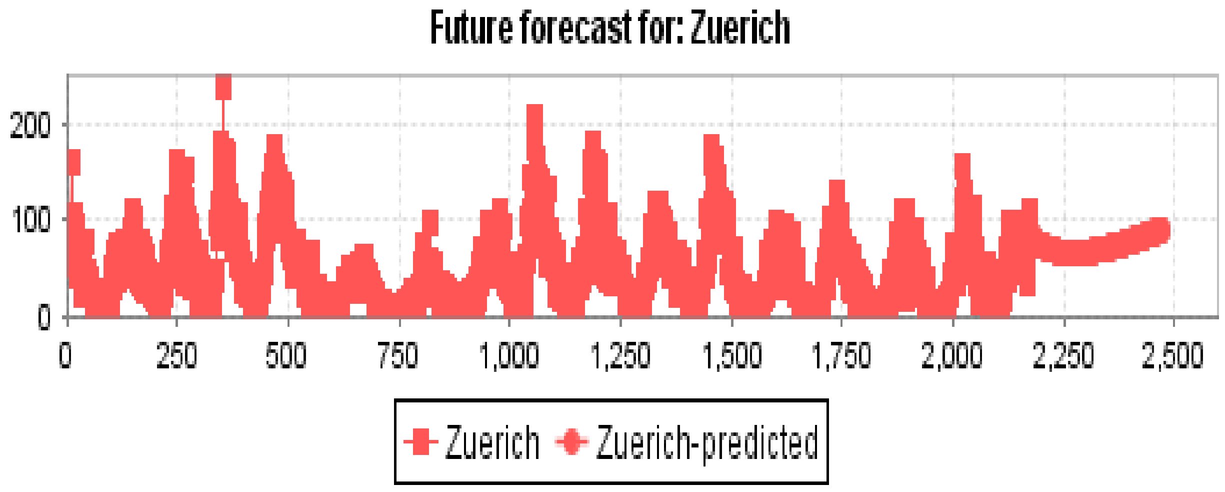
(With feature extraction)

Random forest was not performing good because of rolling mean and as can be seen from the graph we are getting a straight line. This can be improved by better feature selection so as to get better results in time series.

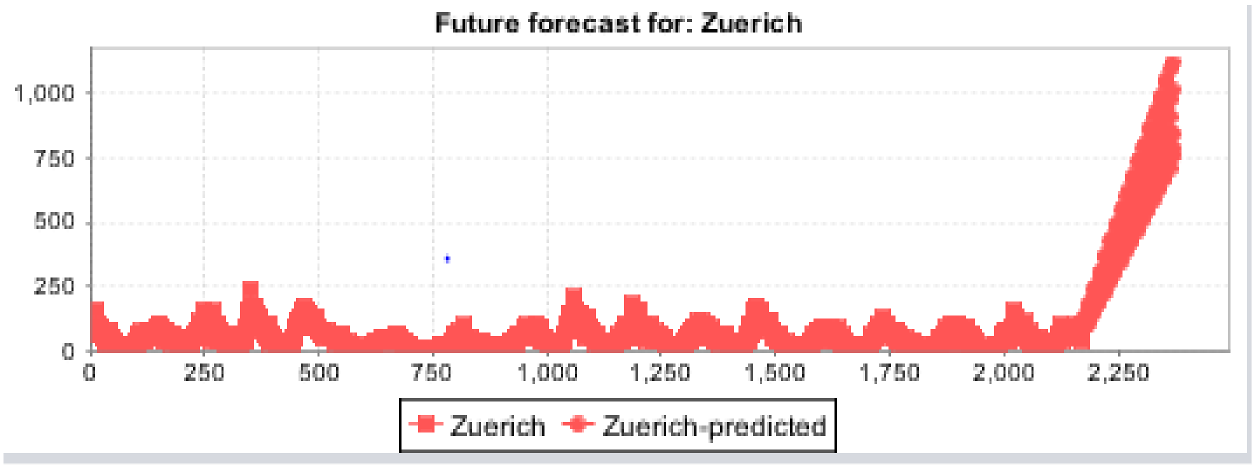
2) **Support Vector Machine:** In machine learning, **support vector machines** (**SVMs**, also **support vector networks**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are not labeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is called **support vector clustering** and is often used in industrial applications either when data are not labeled or when only some data are labeled as a preprocessing for a classification pass.

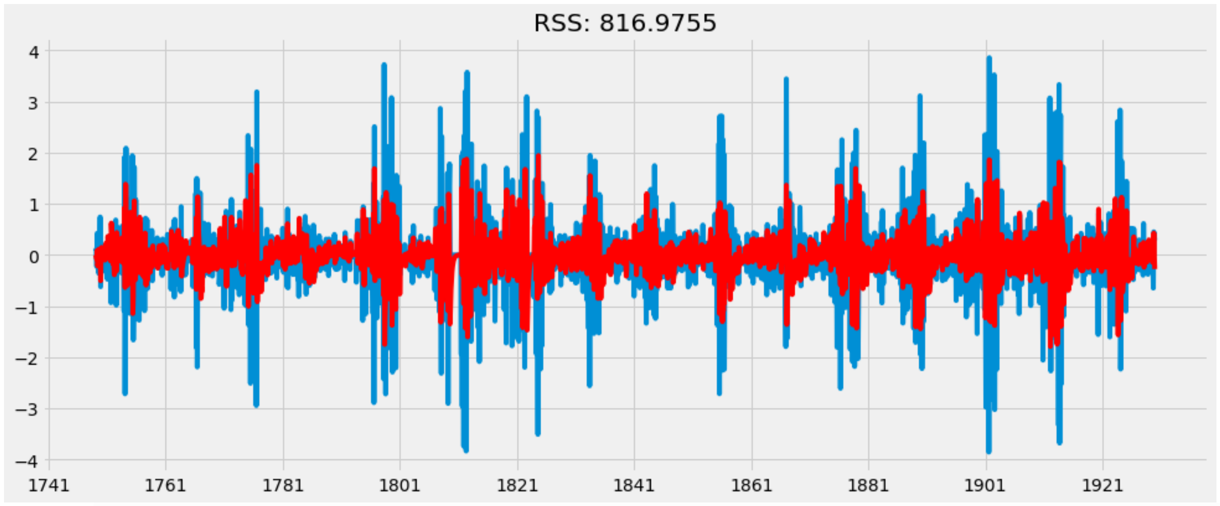


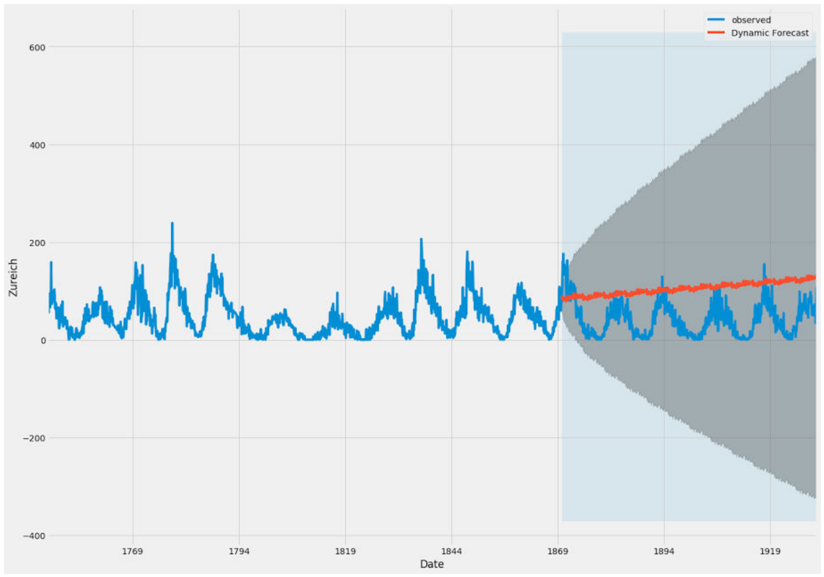
3) Holt-Winters algorithm: Triple Exponential Smoothing, also known as the Holt-Winters method, is one of the many methods or algorithms that can be used to forecast data points in a series, provided that the series is “seasonal”, i.e. repetitive over some period.



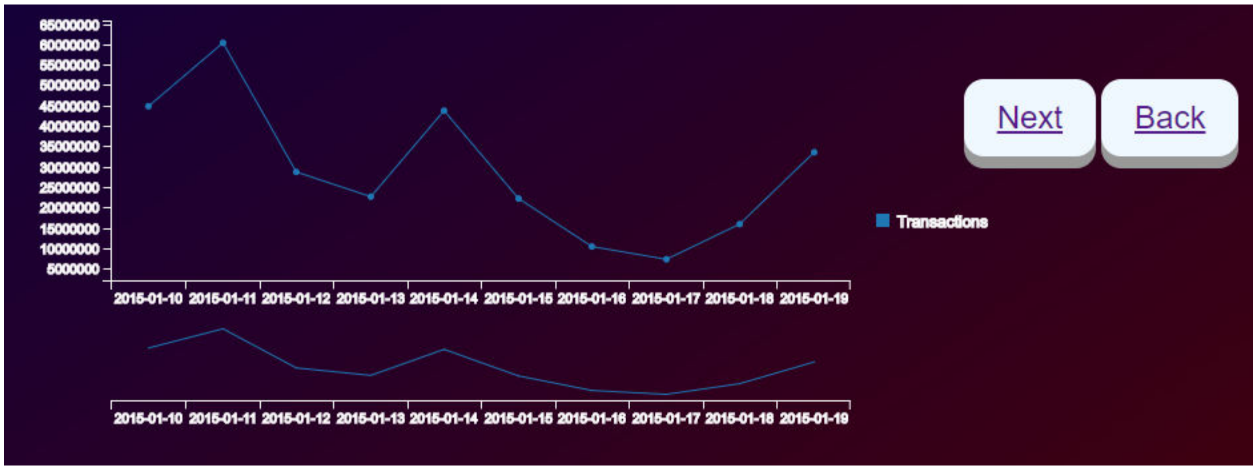
The value is going on increasing therefore it was not used.

4) Arima: In statistics and econometrics, and in particular in time series analysis, an **autoregressive integrated moving average (ARIMA)** model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.


RSS: 816.9755
4
3
2
1
0
−1
−2
−3
−4
1741
1761
1781
1801
1821
1841
1861
1881
1901
1921


observed
Dynamic Forecast
600
400
200
0
−200
−400
Zureich
1769
1794
1819
1844
1869
1894
1919
Date

Backend



Forecasted result

The library has two classes :

1) Forecast
2) SeriesHandler (Optional)

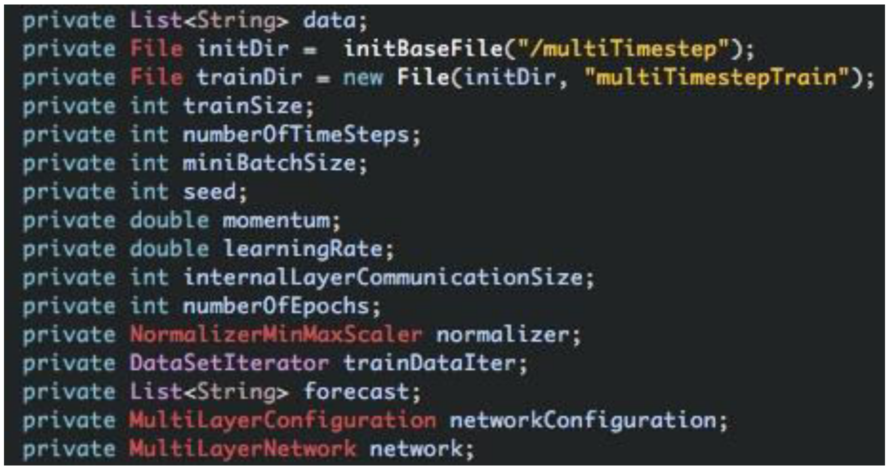
Requirements :

Java archives :

1) dl4j-examples-0.8-SNAPSHOT
2) dl4j-examples-0.8-SNAPSHOT-bin

Forecast.java

This class examines a single dimension String List (Strings should represent Decimal or Integral values) and predicts future values to be added to the list.



```java
private List<String> data;
private File initDir =  initBaseFile("/multiTimestep");
private File trainDir = new File(initDir, "multiTimestepTrain");
private int trainSize;
private int numberOfTimeSteps;
private int miniBatchSize;
private int seed;
private double momentum;
private double learningRate;
private int internalLayerCommunicationSize;
private int numberOfEpochs;
private NormalizerMinMaxScaler normalizer;
private DataSetIterator trainDataIter;
private List<String> forecast;
private MultiLayerConfiguration networkConfiguration;
private MultiLayerNetwork network;
```

Data Members:

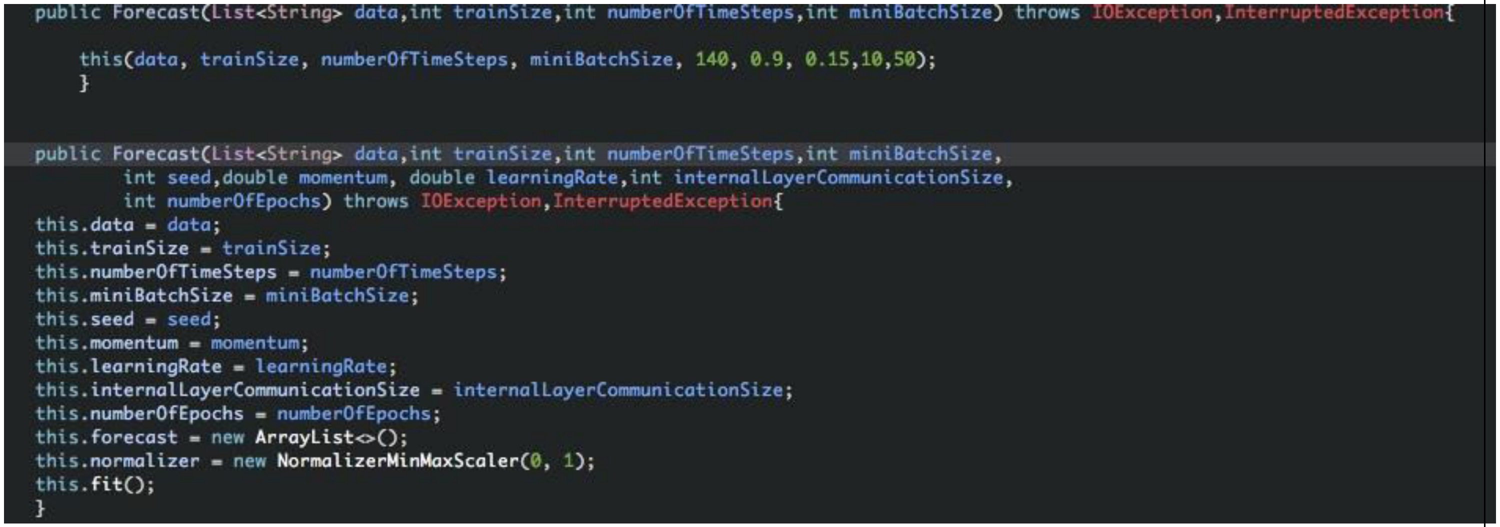
Data Member	About
List<String> data	Stores Training data
File initDir	Root directory for better data management
File trainDir	A single folder to hold training files created by code
int trainSize	Number of training samples
int numberOfTimeSteps	Number of Time Steps (required by algorithm). Should be a factor of trainSize
int miniBatchSize	Smallest Batch Size (required by algorithm). Should be a factor of numberOfTimeSteps
int seed	Used to fix initial weights of neutrons
double momentum	Momentum of optimisation algorithm
double learningRate	Learning rate of neural Network
int internalLayerCommuncationSize	Output size of first layer and input size of second layer
int numberOfEpochs	Number of Epochs
NormalizerMinMaxScaler normaliser	Normaliser used for data’s pre-processing
DataSetIterator trainDataIter	DeepLearning4j networks requires an iterator to train. DataSetIterator is provided by DL4j
List<String> forecast	Stores forecasted Results

Data Member	About
MultiLayerCofiguration networkConfiguration	network configuration object
MultiLayerNetwork network	network object

Member Functions

Function	Access	Arguments	Return Type
Forecast()	public	List<String> data int trainSize int numberOfTimeSteps int miniBatchSize	Constructor
Forecast()	public	List<String> data int trainSize int numberOfTimeSteps int miniBatchSize int seed double momentum double learningRate int internalLayerCommunicationSiz e int numberOfEpochs	Constructor
buildNetwork()	private	-	void
createIndArrayFromStringList()	private	List<jString> rawStrings int startIndex int length	INDArray
fit()	private	-	void
getForecast()	public	-	List<String>
initBaseFile()	private	String	File
predict()	private	-	void
prepNormalizer()	private	-	void
prepTraingIterator()	private	-	void
prepTrainingFiles()	private	-	void

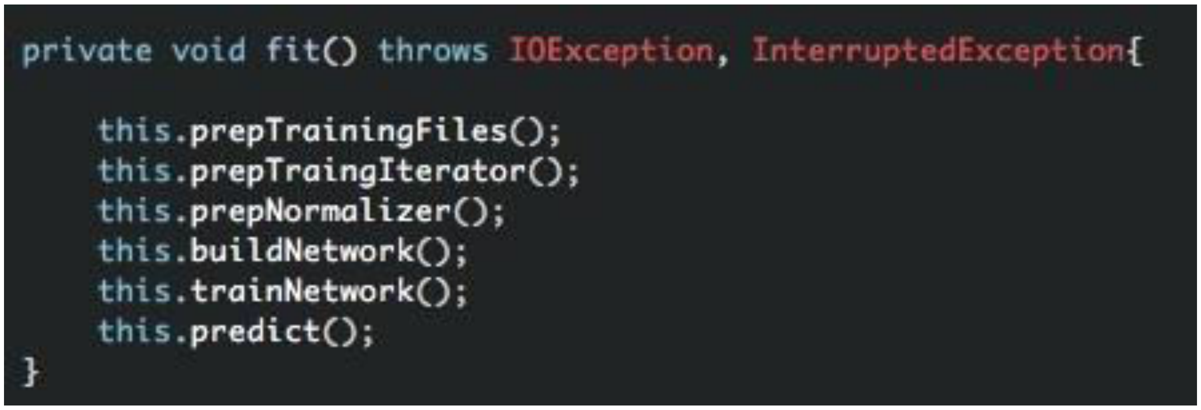
Function	Access	Arguments	Return Type
trainNetwork()	private	-	void



```
public Forecast(List<String> data,int trainSize,int numberOfTimeSteps,int miniBatchSize) throws IOException,InterruptedException{

    this(data, trainSize, numberOfTimeSteps, miniBatchSize, 140, 0.9, 0.15,10,50);
    }


public Forecast(List<String> data,int trainSize,int numberOfTimeSteps,int miniBatchSize,
        int seed,double momentum, double learningRate,int internalLayerCommunicationSize,
        int numberOfEpochs) throws IOException,InterruptedException{
this.data = data;
this.trainSize = trainSize;
this.numberOfTimeSteps = numberOfTimeSteps;
this.miniBatchSize = miniBatchSize;
this.seed = seed;
this.momentum = momentum;
this.learningRate = learningRate;
this.internalLayerCommunicationSize = internalLayerCommunicationSize;
this.numberOfEpochs = numberOfEpochs;
this.forecast = new ArrayList<>();
this.normalizer = new NormalizerMinMaxScaler(0, 1);
this.fit();
}
```

Forecast()

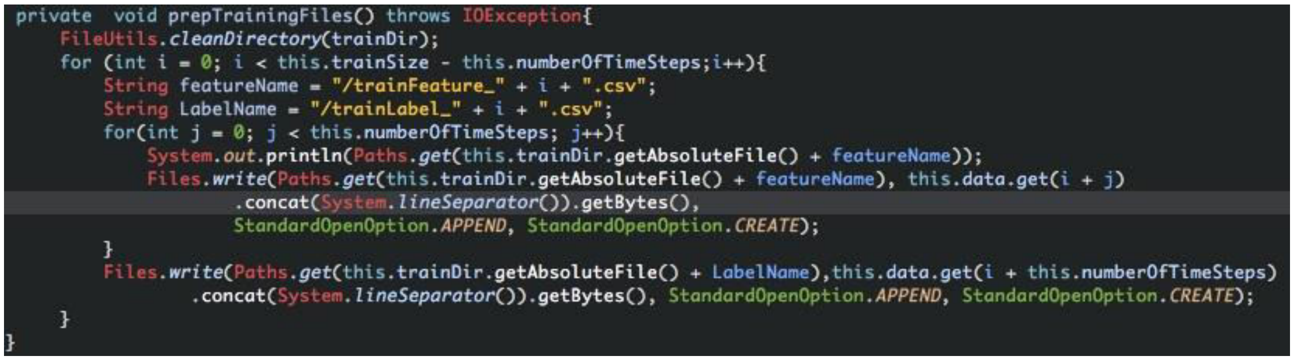
Sets values of data members passed to the constructer and calls fit() to predict the data.



```
private void fit() throws IOException, InterruptedException{

    this.prepTrainingFiles();
    this.prepTraingIterator();
    this.prepNormalizer();
    this.buildNetwork();
    this.trainNetwork();
    this.predict();
}
```

fit()

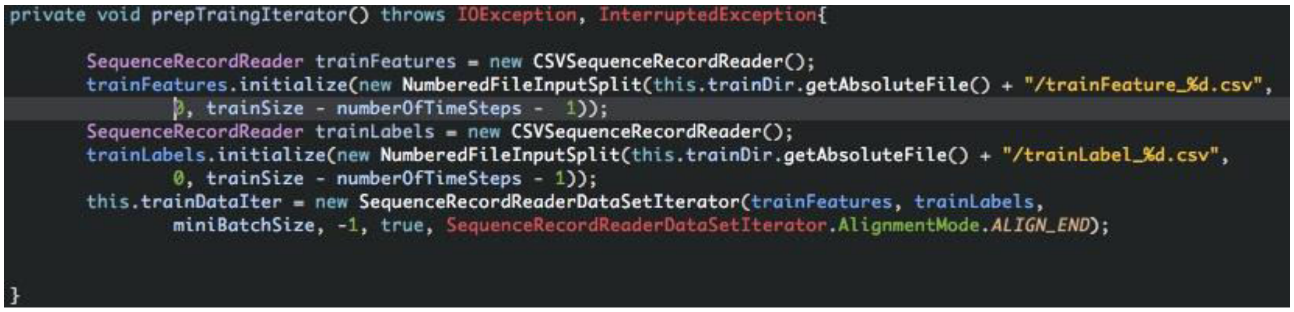
Calls relevant functions in sequential order to perform prediction



```
private  void prepTrainingFiles() throws IOException{
    FileUtils.cleanDirectory(trainDir);
    for (int i = 0; i < this.trainSize - this.numberOfTimeSteps;i++){
        String featureName = "/trainFeature_" + i + ".csv";
        String LabelName = "/trainLabel_" + i + ".csv";
        for(int j = 0; j < this.numberOfTimeSteps; j++){
            System.out.println(Paths.get(this.trainDir.getAbsoluteFile() + featureName));
            Files.write(Paths.get(this.trainDir.getAbsoluteFile() + featureName), this.data.get(i + j)
                    .concat(System.lineSeparator()).getBytes(),
                    StandardOpenOption.APPEND, StandardOpenOption.CREATE);
        }
        Files.write(Paths.get(this.trainDir.getAbsoluteFile() + LabelName),this.data.get(i + this.numberOfTimeSteps)
                .concat(System.lineSeparator()).getBytes(), StandardOpenOption.APPEND, StandardOpenOption.CREATE);
    }
}
```

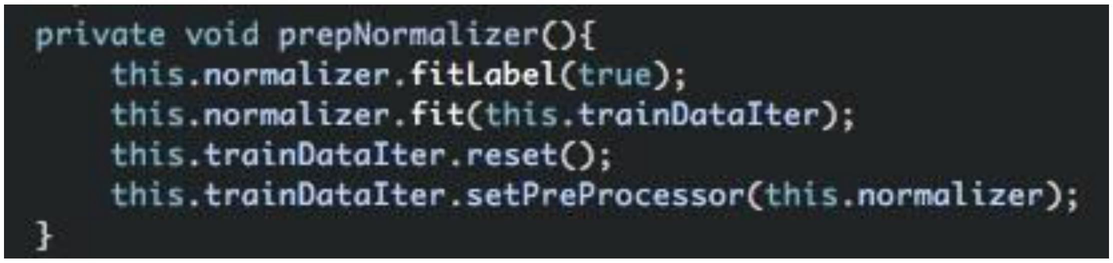
<u>prepTrainingFiles()</u>

- cleanDirectory() deletes any files present in the trainDir directory
- Two types of rifles are created, labels and features.
- Features can be treated as input and corresponding label id their output



```
private void prepTraingIterator() throws IOException, InterruptedException{

        SequenceRecordReader trainFeatures = new CSVSequenceRecordReader();
        trainFeatures.initialize(new NumberedFileInputSplit(this.trainDir.getAbsoluteFile() + "/trainFeature_%d.csv",
                0, trainSize - numberOfTimeSteps -  1));
        SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
        trainLabels.initialize(new NumberedFileInputSplit(this.trainDir.getAbsoluteFile() + "/trainLabel_%d.csv",
                0, trainSize - numberOfTimeSteps - 1));
        this.trainDataIter = new SequenceRecordReaderDataSetIterator(trainFeatures, trainLabels,
                miniBatchSize, -1, true, SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);


}
```

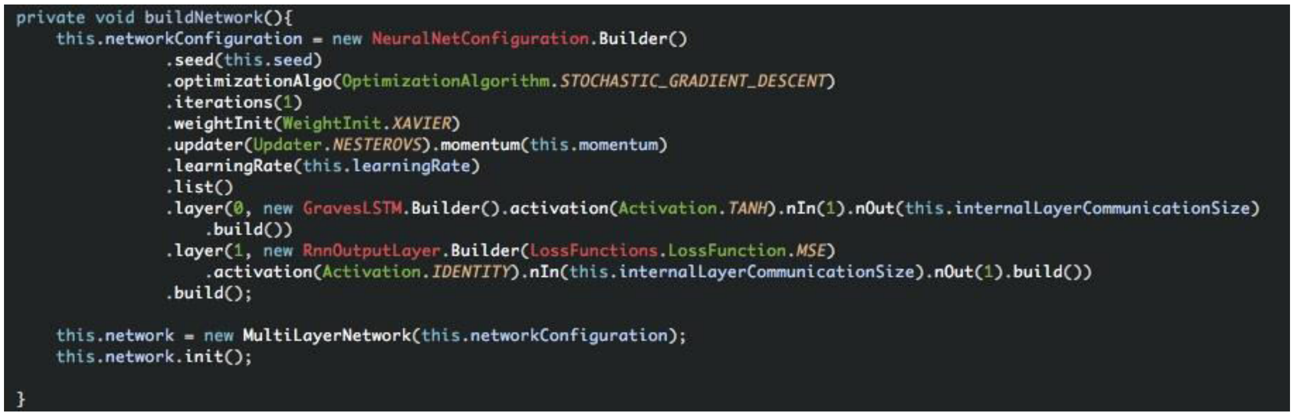
- The functions saves the files in trainDir.
- SequenceRecordReader reads a sequence of files. We need to enter the initial file number to final file number
- TrainDataIterator keeps a record of both, label sequence reader and feature sequence reader and passes both accordingly to the model



```
private void prepNormalizer(){
    this.normalizer.fitLabel(true);
    this.normalizer.fit(this.trainDataIter);
    this.trainDataIter.reset();
    this.trainDataIter.setPreProcessor(this.normalizer);
}
```

<u>prepNormalizer()</u>

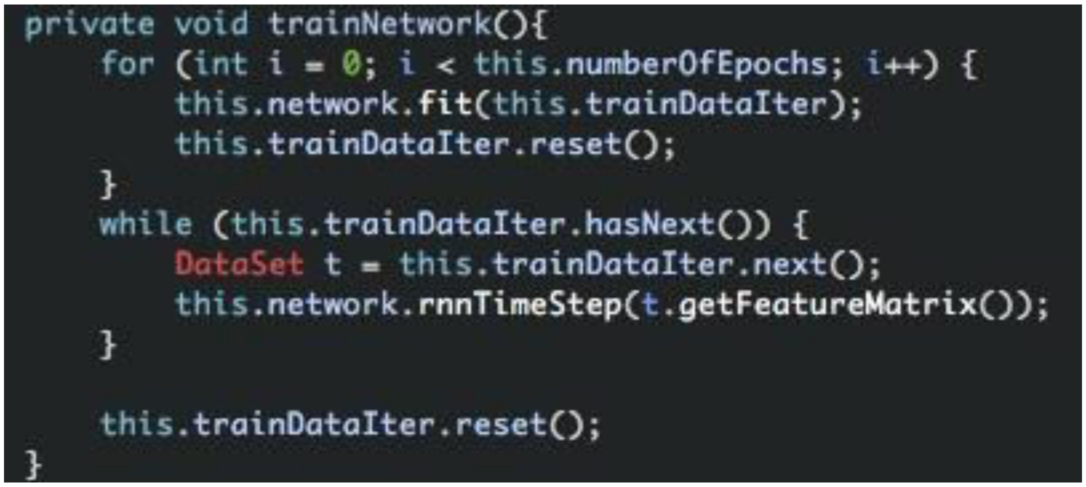
- Creates a data normaliser. We use labels to fit
- trainDatIter is needed to be reset to 0 index as normaliser takes it to EOF
- after fitting normaliser, we set preprocessor to the training data



```java
private void buildNetwork(){
    this.networkConfiguration = new NeuralNetConfiguration.Builder()
                .seed(this.seed)
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
                .iterations(1)
                .weightInit(WeightInit.XAVIER)
                .updater(Updater.NESTEROVS).momentum(this.momentum)
                .learningRate(this.learningRate)
                .list()
                .layer(0, new GravesLSTM.Builder().activation(Activation.TANH).nIn(1).nOut(this.internalLayerCommunicationSize)
                    .build())
                .layer(1, new RnnOutputLayer.Builder(LossFunctions.LossFunction.MSE)
                    .activation(Activation.IDENTITY).nIn(this.internalLayerCommunicationSize).nOut(1).build())
                .build();

    this.network = new MultiLayerNetwork(this.networkConfiguration);
    this.network.init();

}
```

<u>buildNetwork()</u>

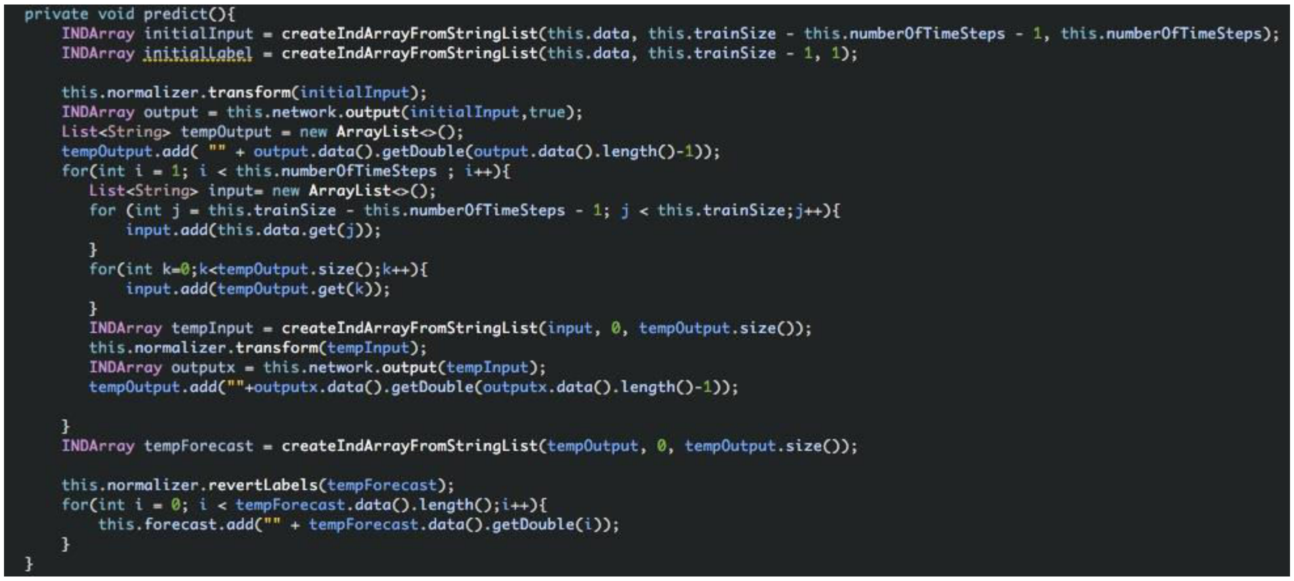
- Creates network configuration
- We create one LSTM layer, and one output layer
- A new network is created and initialised



```java
private void trainNetwork(){
    for (int i = 0; i < this.numberOfEpochs; i++) {
        this.network.fit(this.trainDataIter);
        this.trainDataIter.reset();
    }
    while (this.trainDataIter.hasNext()) {
        DataSet t = this.trainDataIter.next();
        this.network.rnnTimeStep(t.getFeatureMatrix());
    }

    this.trainDataIter.reset();
}
```

<u>trainNetwork()</u>

- Trains networks for specified epochs
- trainDataIter needs to be reset after each epoch
- rnnTimeStep is a requirement from the algorithm



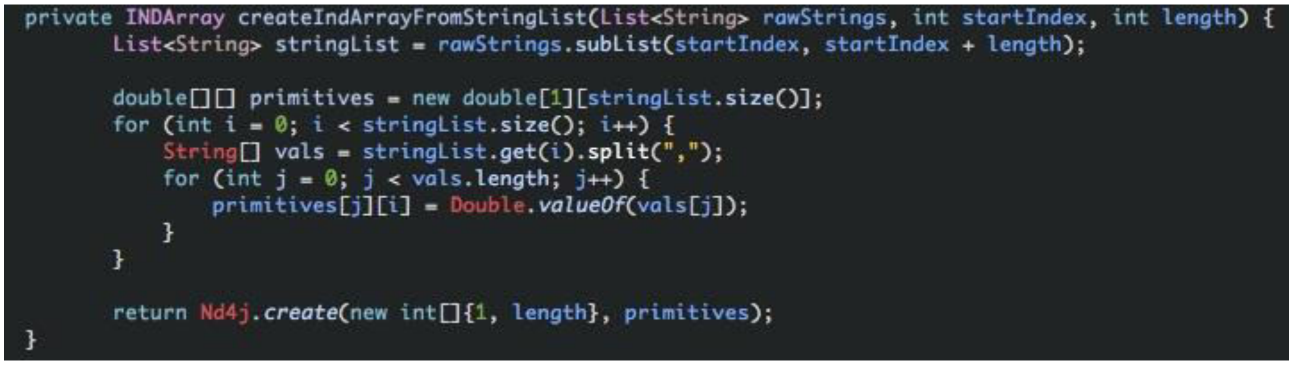
```java
private void predict(){
    INDArray initialInput = createIndArrayFromStringList(this.data, this.trainSize - this.numberOfTimeSteps - 1, this.numberOfTimeSteps);
    INDArray initialLabel = createIndArrayFromStringList(this.data, this.trainSize - 1, 1);

    this.normalizer.transform(initialInput);
    INDArray output = this.network.output(initialInput,true);
    List<String> tempOutput = new ArrayList<>();
    tempOutput.add( "" + output.data().getDouble(output.data().length()-1));
    for(int i = 1; i < this.numberOfTimeSteps ; i++){
       List<String> input= new ArrayList<>();
       for (int j = this.trainSize - this.numberOfTimeSteps - 1; j < this.trainSize;j++){
           input.add(this.data.get(j));
       }
       for(int k=0;k<tempOutput.size();k++){
           input.add(tempOutput.get(k));
       }
       INDArray tempInput = createIndArrayFromStringList(input, 0, tempOutput.size());
       this.normalizer.transform(tempInput);
       INDArray outputx = this.network.output(tempInput);
       tempOutput.add(""+outputx.data().getDouble(outputx.data().length()-1));

    }
    INDArray tempForecast = createIndArrayFromStringList(tempOutput, 0, tempOutput.size());

    this.normalizer.revertLabels(tempForecast);
    for(int i = 0; i < tempForecast.data().length();i++){
        this.forecast.add("" + tempForecast.data().getDouble(i));
    }
}
```

<u>predict()</u>

- This function dynamically creates input from previous outputs to achieve prediction
- We need to normalise each input and revert each output

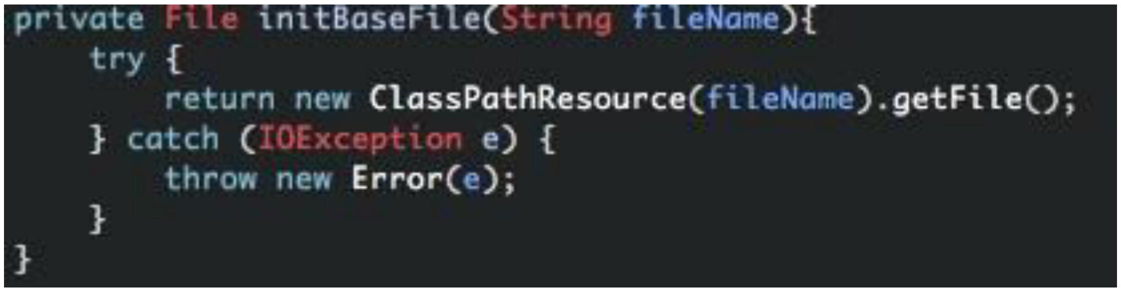
- initial inputs are from the last feature file
- We find label for the input features and use this label as an input for next prediction
- If there are *k* features in a given input, for next prediction we use *k-1* values from these and one output value as input
- This is done till be have a feature of purely predicted values. Predicting further from these can result in magnified errors

<u>createIndArrayStringList()</u>



```java
private INDArray createIndArrayFromStringList(List<String> rawStrings, int startIndex, int length) {
        List<String> stringList = rawStrings.subList(startIndex, startIndex + length);

        double[][] primitives = new double[1][stringList.size()];
        for (int i = 0; i < stringList.size(); i++) {
            String[] vals = stringList.get(i).split(",");
            for (int j = 0; j < vals.length; j++) {
                primitives[j][i] = Double.valueOf(vals[j]);
            }
        }

        return Nd4j.create(new int[]{1, length}, primitives);
}
```

- INDArray is an interface provided by DL4j. It is used as an input to train the model and hence, this function is required
- This function takes a String List and returns corresponding INDArray



```java
private File initBaseFile(String fileName){
    try {
        return new ClassPathResource(fileName).getFile();
    } catch (IOException e) {
        throw new Error(e);
    }
}
```

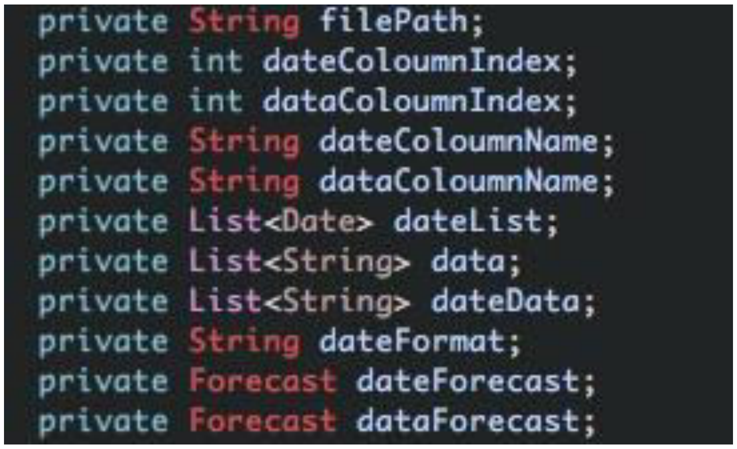
<u>initBaseFile()</u>

- Initiates directory of given name as root. Required for cleaner file handling

SeriesHandler.java

This handles any univariate time series. It takes a cvs path, date data column name and data column name as input.

Data Members

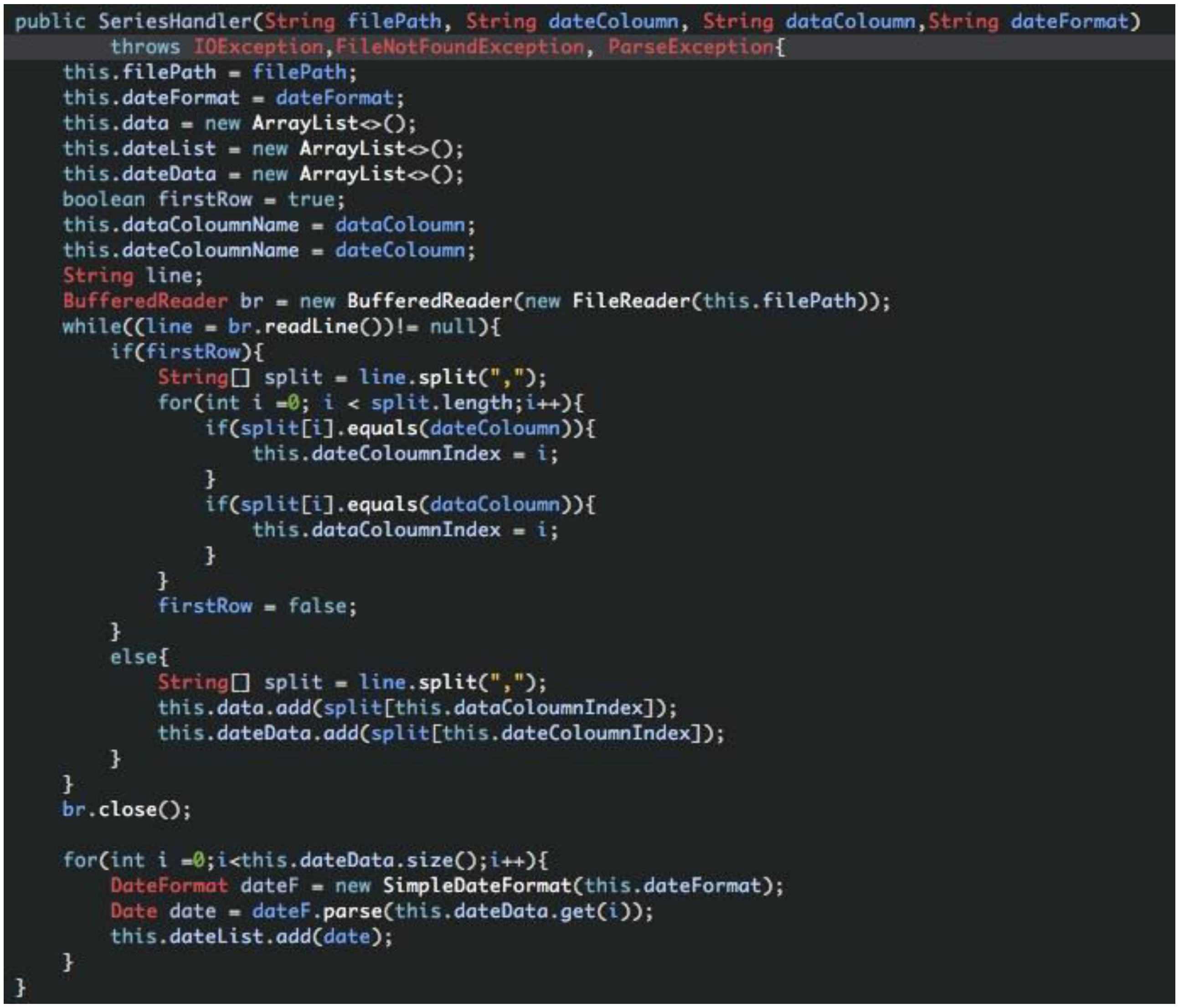


```
private String filePath;
private int dateColoumnIndex;
private int dataColoumnIndex;
private String dateColoumnName;
private String dataColoumnName;
private List<Date> dateList;
private List<String> data;
private List<String> dateData;
private String dateFormat;
private Forecast dateForecast;
private Forecast dataForecast;
```

Data Member	**About**
String filePath	Path to cvs file
int dateColoumnIndex	Index of column containing dates
int dataColoumnIndex	Index of column containing data
String dateColoumnName	Name of column containing dates
String dataColoumnName	Name of column containing data
List<Date> dateList	List of Dates in csv
List<String> data	List of data in csv
List<String> dateData	List of Dates in csv in String format
String dateFormat	Format in which dates are found in csv
Forecast dateForecast	Forecast object for dates
Forecast dataForecast	Forecast object for data

Member Functions

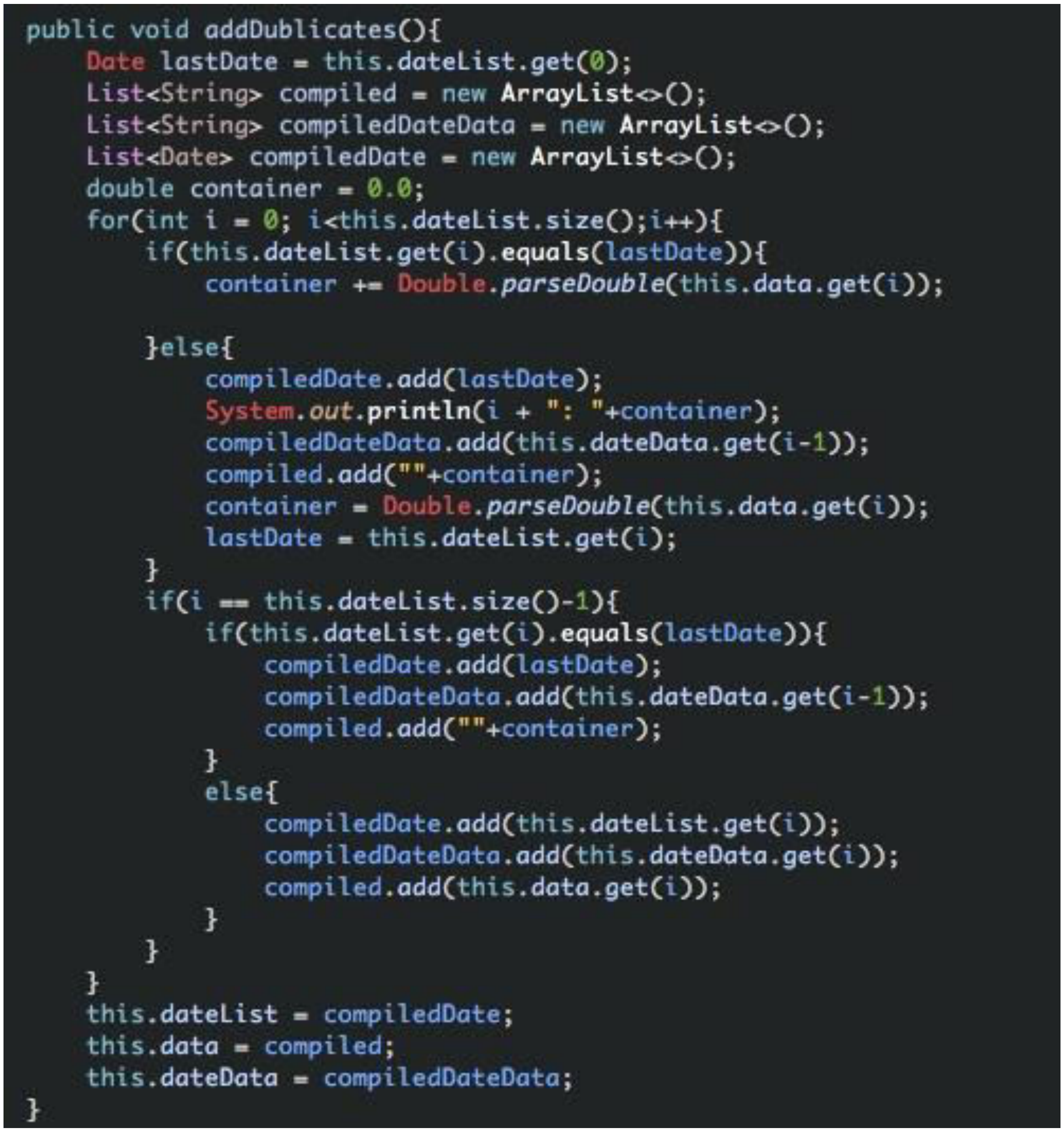
Function	Access	Arguments	Return Type
SeriesHandler()	Public	String filePath String dateColoumn String dataColoumn String dateFormat	Constructor
addDublicates()	Public	-	void
createAndSaveForecast()	Public	String savePath int trainSize int numberOfTimeSteps int miniBatchSize	void
createAndSaveForecast()	Public	String savePath int trainSize int numberOfTimeSteps int miniBatchSize int seed double momentum double learningRate int internalLayerCommunicationSize int numberOfEpochs	void
saveCsv	private	String savePath List<String> forecastedData List<String> finalDates	void



```java
public SeriesHandler(String filePath, String dateColoumn, String dataColoumn,String dateFormat)
        throws IOException,FileNotFoundException, ParseException{
    this.filePath = filePath;
    this.dateFormat = dateFormat;
    this.data = new ArrayList<>();
    this.dateList = new ArrayList<>();
    this.dateData = new ArrayList<>();
    boolean firstRow = true;
    this.dataColoumnName = dataColoumn;
    this.dateColoumnName = dateColoumn;
    String line;
    BufferedReader br = new BufferedReader(new FileReader(this.filePath));
    while((line = br.readLine())!= null){
        if(firstRow){
            String[] split = line.split(",");
            for(int i =0; i < split.length;i++){
                if(split[i].equals(dateColoumn)){
                    this.dateColoumnIndex = i;
                }
                if(split[i].equals(dataColoumn)){
                    this.dataColoumnIndex = i;
                }
            }
            firstRow = false;
        }
        else{
            String[] split = line.split(",");
            this.data.add(split[this.dataColoumnIndex]);
            this.dateData.add(split[this.dateColoumnIndex]);
        }
    }
    br.close();

    for(int i =0;i<this.dateData.size();i++){
        DateFormat dateF = new SimpleDateFormat(this.dateFormat);
        Date date = dateF.parse(this.dateData.get(i));
        this.dateList.add(date);
    }
}
```

<u>SeriesHandler()</u>

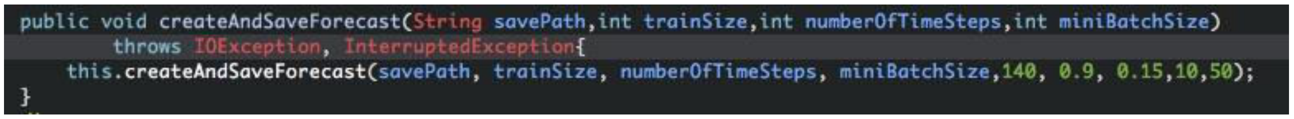
- Extracts dates and data from provided file (via given file path)



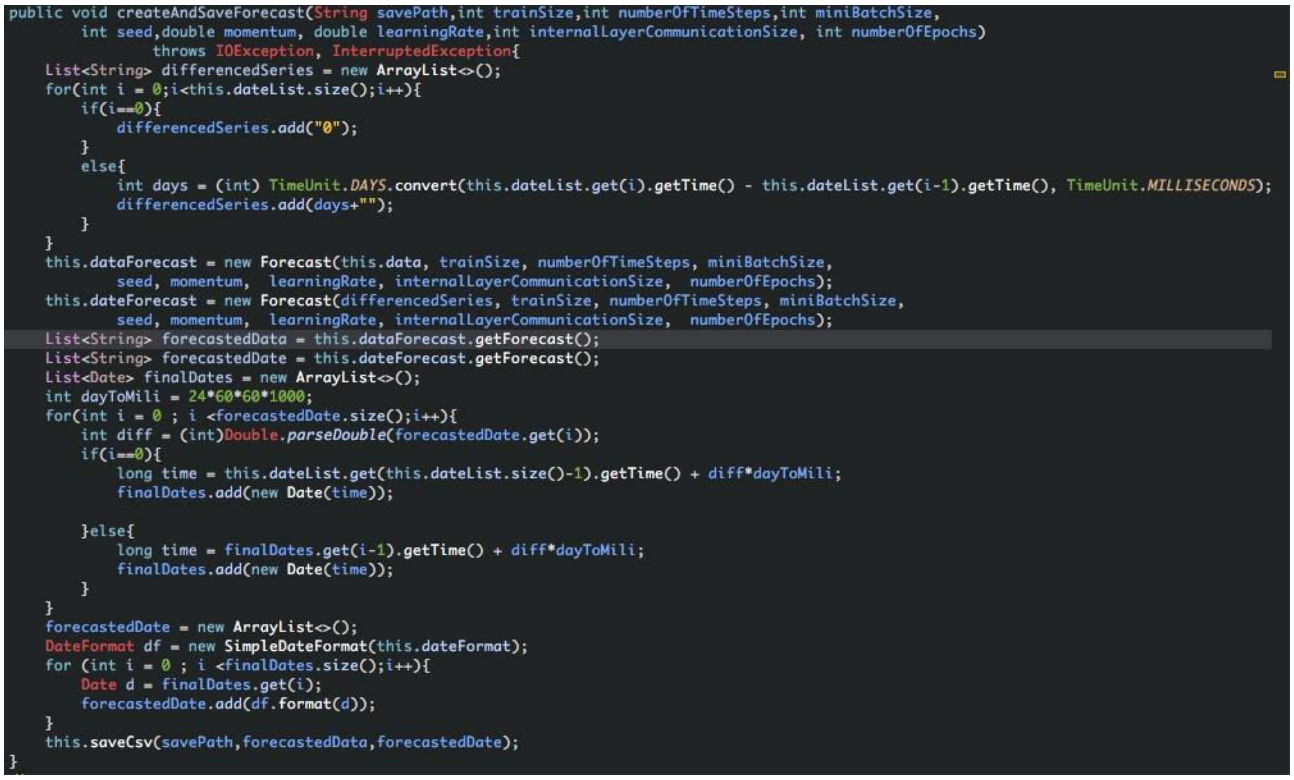
```
public void addDublicates(){
    Date lastDate = this.dateList.get(0);
    List<String> compiled = new ArrayList<>();
    List<String> compiledDateData = new ArrayList<>();
    List<Date> compiledDate = new ArrayList<>();
    double container = 0.0;
    for(int i = 0; i<this.dateList.size();i++){
        if(this.dateList.get(i).equals(lastDate)){
            container += Double.parseDouble(this.data.get(i));

        }else{
            compiledDate.add(lastDate);
            System.out.println(i + ": "+container);
            compiledDateData.add(this.dateData.get(i-1));
            compiled.add(""+container);
            container = Double.parseDouble(this.data.get(i));
            lastDate = this.dateList.get(i);
        }
        if(i == this.dateList.size()-1){
            if(this.dateList.get(i).equals(lastDate)){
                compiledDate.add(lastDate);
                compiledDateData.add(this.dateData.get(i-1));
                compiled.add(""+container);
            }
            else{
                compiledDate.add(this.dateList.get(i));
                compiledDateData.add(this.dateData.get(i));
                compiled.add(this.data.get(i));
            }
        }
    }
    this.dateList = compiledDate;
    this.data = compiled;
    this.dateData = compiledDateData;
}
```

addDublicates()

- This method identifies transactions that have taken place on a single day and adds them
- This is done as we only need to forecast the total transaction amount at a given time point
- It saves the unique dates only



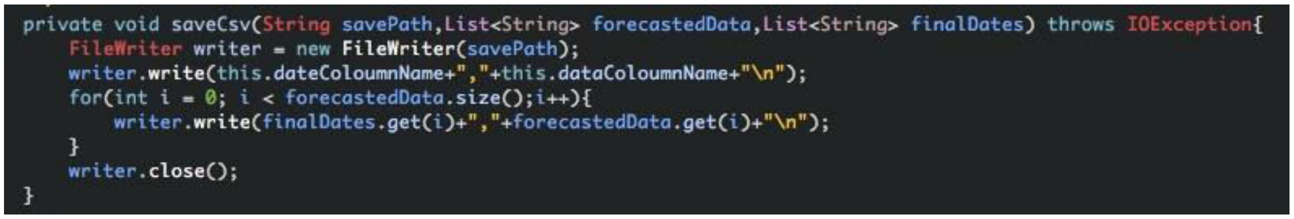
```java
public void createAndSaveForecast(String savePath,int trainSize,int numberOfTimeSteps,int miniBatchSize)
        throws IOException, InterruptedException{
    this.createAndSaveForecast(savePath, trainSize, numberOfTimeSteps, miniBatchSize,140, 0.9, 0.15,10,50);
}
```



```java
public void createAndSaveForecast(String savePath,int trainSize,int numberOfTimeSteps,int miniBatchSize,
        int seed,double momentum, double learningRate,int internalLayerCommunicationSize, int numberOfEpochs)
            throws IOException, InterruptedException{
    List<String> differencedSeries = new ArrayList<>();
    for(int i = 0;i<this.dateList.size();i++){
        if(i==0){
            differencedSeries.add("0");
        }
        else{
            int days = (int) TimeUnit.DAYS.convert(this.dateList.get(i).getTime() - this.dateList.get(i-1).getTime(), TimeUnit.MILLISECONDS);
            differencedSeries.add(days+"");
        }
    }
    this.dataForecast = new Forecast(this.data, trainSize, numberOfTimeSteps, miniBatchSize,
            seed, momentum,  learningRate, internalLayerCommunicationSize,  numberOfEpochs);
    this.dateForecast = new Forecast(differencedSeries, trainSize, numberOfTimeSteps, miniBatchSize,
            seed, momentum,  learningRate, internalLayerCommunicationSize,  numberOfEpochs);
    List<String> forecastedData = this.dataForecast.getForecast();
    List<String> forecastedDate = this.dateForecast.getForecast();
    List<Date> finalDates = new ArrayList<>();
    int dayToMili = 24*60*60*1000;
    for(int i = 0 ; i <forecastedDate.size();i++){
        int diff = (int)Double.parseDouble(forecastedDate.get(i));
        if(i==0){
            long time = this.dateList.get(this.dateList.size()-1).getTime() + diff*dayToMili;
            finalDates.add(new Date(time));

        }else{
            long time = finalDates.get(i-1).getTime() + diff*dayToMili;
            finalDates.add(new Date(time));
        }
    }
    forecastedDate = new ArrayList<>();
    DateFormat df = new SimpleDateFormat(this.dateFormat);
    for (int i = 0 ; i <finalDates.size();i++){
        Date d = finalDates.get(i);
        forecastedDate.add(df.format(d));
    }
    this.saveCsv(savePath,forecastedData,forecastedDate);
}
```

createAndSaveForecast()

- This method uses Forecast object to forecast data
- The Date data is differenced first and then the differenced series is predicted
- The predicted difference series is then added back to last date to get predicted dates
- The method saves this predicted data into the file passed



```java
private void saveCsv(String savePath,List<String> forecastedData,List<String> finalDates) throws IOException{
    FileWriter writer = new FileWriter(savePath);
    writer.write(this.dateColoumnName+","+this.dataColoumnName+"\n");
    for(int i = 0; i < forecastedData.size();i++){
        writer.write(finalDates.get(i)+","+forecastedData.get(i)+"\n");
    }
    writer.close();
}
```

saveCsv()

- This method takes the predicted data and dates as input and writes them into the specified CSV