A Project Report on

Predictive Maintenance on Delhi Metro Fault Data

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

Devarshi Goswami 1661038

Under the Guidance of

Mr. Alok Kumar Pani

and

Mr. Osho Aditya Saxena

Department of Computer Science and Engineering

School of Engineering and Technology,

CHRIST (Deemed to be University),

Kumbalgodu, Bangalore - 560 074

April-2020

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

School of Engineering and Technology

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that **Devarshi Goswami** has successfully completed the project work entitled "**Predictive Maintenance on Delhi Metro Fault Data**" in partial fulfillment for the award of **Bachelor of Technology** in **Information Technology** during the year **2019-2020**.

Mr. Alok Kumar Pani
Assistant Professor

Mr. Osho Aditya Saxena
Assistant Manager

Dr. K Balachandran
Head

Dr. Iven Jose
Dean

School of Engineering and Technology

Department of Computer Science and Engineering

BONAFIDE CERTIFICATE

It is to certify that this project titled ”Predictive Maintenance on Delhi Metro Fault Data” is the bonafide work of

Name	Register Number
Devarshi Goswami	1661038

Examiners [Name and Signature]	Name of the Candidate :
1.	Register Number :
2.	Date of Examination :

Alok Kumar Pani Engineering CSE <alok.kumar@christuniversity.in>

Regarding Devarshi Goswami

Osho ASaxena <osho.20990@dmrc.org> Fri, Apr 24, 2020 at 5:08 PM
To: alok kumar <alok.kumar@christuniversity.in>

Sir,
I hereby confirm that Devarshi Goswami, Enrollment no, 1661038, student of Christ University CSE deptt, completed his internship at Delhi Metro Rail Corporation, Delhi from 13/01/2020 to 13/03/2020 for a duration of 8 weeks. He was found to be punctual, efficient and insightful in completing the works assigned to him and contributing to the organization.

Please ensure that the candidate obtains the completion certificate from DMRC in person as soon as possible.

Regards,
Osho A. Saxena
AM/IT
DMRC

From: "alok kumar" <alok.kumar@christuniversity.in>
To: "Osho ASaxena" <osho.20990@dmrc.org>
Sent: Friday, April 24, 2020 10:59:54 AM
Subject: Regarding Devarshi Goswami
[Quoted text hidden]

Acknowledgement

I would like to thank CHRIST (Deemed to be University)Vice Chancellor, **Dr. Rev. Fr. Abraham V M**, Pro Vice Chancellor,**Dr. Rev. Fr. Joseph C C**, Director of School of Engineering and Technology, **Dr. Rev. Fr. Benny Thomas** and the Dean **Dr. Iven Jose** for their kind patronage.

I would sincerely like to express my utmost gratitude and appreciation to the Head of the Department of Computer Science and Engineering, School of Engineering and Technology **Dr. K Balachandran**, for giving me this opportunity to take up this project.

I am also extremely grateful to my guide, **Mr. Alok Kumar Pani**, who has supported and helped me to carry out the project. His constant monitoring and encouragement helped me keep up to the project schedule.

I am also extremely grateful to my co-guide, **Mr. Osho Aditya Saxena**, who has supported and helped me to carry out the project. His constant monitoring and encouragement helped me keep up to the project schedule.

Declaration

I, hereby declare that the Project titled "**Predictive Maintenance on Delhi Metro Fault Data**" is a record of original project work undertaken by me for the award of the degree of **Bachelor of Technology** in **Information Technology**. I have completed this study under the supervision of **Mr. Alok Kumar Pani**, Assistant Professor , Department of Computer Science and Engineering and **Mr. Osho Aditya Saxena**,Assistant Manager, Information Technology, DMRC.

I also declare that this project report has not been submitted for the award of any degree, diploma, associateship, fellowship or other title anywhere else. It has not been sent for any publication or presentation purpose.

Place: School of Engineering and Technology, CHRIST (Deemed to be University), Bangalore
Date:

Name	Register Number	Signature
Devarshi Goswami	1661038	

Abstract

Recently, the task of maintaining and working on trains has been greatly improved and leveraged by capability of capturing data in real time and using that data for trend analysis.

The current scenario is that only the fault values are stored in an unstructured format and these values are made accessible to maintainers and manufacturers . It is not available in real time. The proposed framework suggests collection of real time sensor values from various components of the train and build a model that mines for temporal sequences of failures and use that data to forecast future failures via a time series prediction.
The final step of the proposed solution will be to deploy this model and its predictions on the cloud so that all stakeholders can be aware of maintenance metrics of train at all times.
For mining of temporal sequences , we will use an algorithm named Sequential Pattern Discovery using Equivalence classes (SPADE) which makes use of combinatoral properties and effective lattice searching to divide the original problem into fragments which can be then solved with ease right in the main memory via very simple join operations. This allows us to mine for association rules in temporal order that will tell us if a specific component fails due to a failure in some other component of the same train and at which time the failure occurs. For the analysis of trends and their predictions, Auto-regressive Integrated Moving Average (ARIMA) has been used . The prediction supplies information about when and which components of the train should undergo maintenance beforehand and also during regular maintenance events so that all averse conditions and extra costs can be preempted.

Contents

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

.

Item	Description
TRAINSET	The TRAINSET refers to the complete metro rail running on a specefic line that will contain all its unique COMPONENTS (has a prefix of 'TNST' with 3 numbers succeeding it.)
Object Part Code	The COMPONENT unique identification number of a specefic compenent of a train.(It has the prefix 'TC' with 3 numbers succeeding it.)
LN	**L**ine **N**number
P	lag order or the number of lag observations made in the model
D	degree of differencing or the amount of times the raw observations are being differenced
Q	order of moving average or the size of window function

Chapter 1

INTRODUCTION

The transportation industry as of 2017 was worth 75 billion USD and had a growth rate of 5% annually.It is and always has been a key part of the human experience and facilitating businesses all over the world.
Sensor technologies has always been an integral part of the railway networks for essential features like security , traffic control and maintenance measures.And considering Moore's law, these sensors will keep on getting cheaper and evolving into new and better counterparts of themselves everyday . Hence, utilizing this vast amount of data created by these sensors for anything apart from the usages mentioned earlier can become obvious only later on.The possibilities are endless.
The trains are equipped with an abundance of sensors, monitoring many system parameters Those are the variables that can lead to insights regarding maintenance necessity or system performance In the future they shall not only be used for diagnostics, but also to predict impending failures and help prevent them from happening altogether This is known as predictive maintenance, and the main goal of this research project is to develop methods to apply it to the train fleet and prove its feasibility Furthermore, possibilities to automatize the various prediction processes will be researched, as well as an integration of the resulting methodology into a real-world maintenance system will be targeted
Our workflow includes introduction of historical data to train a model and use this trained model on live data to determine weather that machine is operating within standard parameters or not The key aim of our proposed framework is twofold .
Firstly, to mine for temporal sequential patterns in railway component failures across the dense network of metro rails and secondly , to understand this mined temporal data and build a model to predict future failures of components from it. This will assist us to

have a complete understanding of where and when and why any fault in any component occurs so that necessary steps can be promptly taken in order to overcome those.

1.1 Problem Identification

The Delhi Metro Railway network is a fast inter and intra city transit system serving Delhi and the surrounding states within the National Capital Region of India It is by far busiest and largest metro network in India, and second oldest after the Kolkata Metro .It consists of eleven colour-coded normal lines serving 285 stations with a complete length of 391 kilometres (243 mi) Delhi Metro completes over 2,700 journeys every day Even though the network of rails in Delhi metro is gargantuan, there is no concrete predictive maintenance solution Every train is sent to one of large number of workshops for maintenance on a predetermined date monthly This method has drawbacks to it because time and money is being wasted on components that do not need maintenance and critical components that might need maintenance bimonthly or even every week are being overlookedClearly, a metro network that serves 15 million people in our country's capital should not be this impoverished and we need to do something about it

1.2 Problem Formulation

The problem scenario is threefold

1.2.1 Collection of data

A large part of Delhi metro's network is sill not collecting data for analytical purposes and the meagre amount that is being collected is unstructured and fragmented Although, data collection has started in recently , the idea will take time to propagate through the entire network of rails

1.2.2 Understanding data

Since the metro rails log data has traditionally been in a format that is only comprehensible by maintainers for understanding any anomalies during a train's visit to workshops every month, the data needs to be understood and pre-processed in such a way that our algorithms are able to comprehend it and form inferences from it

1.2.3 Researching Algorithms

Understanding which algorithms to use to properly train a general model that will be of use to predict our out of sample future failure cases is of absolute importance That understanding will come from doing case studies of previously deployed predictive maintenance solutions

1.3 Problem Statement & Objectives

The key objectives to tackle the problem domain would be to :

1. deal with large amounts of diagnostic data
2. establish valid truth from data-sets
3. create meaningful features to emphasise underlying effects that indicate failure
4. each train-set may not include same number of components since sequential train-lines may or may not be of the same manufacturer
5. we need to set up labels for each type of diagnostic data
6. finally, define an evaluation procedure to achieve optimal prediction

1.4 Limitations

The work on this project should start from the infantile phase of data collection, which is really slow in a government agency, to finally deploying a fully trained model Hence the scope of the project can be deemed as massive Also, I am only person sanctioned to be working on this along with my external guide's help every now and then Also due to this being an industrial exposure project having a time constraint of only 8 hours per day per workweek for 3 months , deployment of final working model can be a far-fetched expectation What the stakeholders can hope for , although, is for a prototype model trained on whatever data is available at the time being by the end of the project tenure

Chapter 2

RESEARCH METHODOLOGY

Predictive and Preventive maintenance scenarios typically arise around large scale machines, in our case , a metro rail A great predictive maintenance process allows prevention of failures, aid in planning for future resources and also help in reduction of maintenance costs The ultimate goal of predictive maintenance is the ability to seamlessly predict any any every equipment failure (based on sensor metrics) after which corrective maintenance measures are applied.

2.1 Data Collection

The approach to solving the problem statement starts with us procuring the data logs of failure of components across all train lines of Delhi metro.In total there are 9 lines of metro rail connecting all different parts of Delhi and actionable data has been procured for 4 lines LN1 to LN4.

2.2 Data content and thematic analysis

Completely understating the data that we have before starting prepossessing is of absolute necessity before researching on ways to preprocess it and feed it to an algorithm. Lets understand the pre-existing data first. The input data has 7 columns Train Line, Object Part Code , Train Set , Plant Section , Maintenance Plant , Fault Code and Date of failure. A specific component in a train can be identified by the Object Part code which is the component ID and Train Set which is the ID of the train itself. Maintenance plant and Plant Section the ID of the maintenance workshop and an area inside the workshop the data has been recorded in respectively and the Fault code identifies what sort of fault has occurred in the component.

2.3 Case studies

Similar data driven predictive maintenance solutions have been put forward before in other railway and metro railway networks in other countries. The most relevant of those would be by the post graduate students of the university of Darmstadt for a German rail operator Deutsche Bahn AG which will be talked about in detail in successive sections. Another relevant case study would be a paper presented in the Third International Conference on Electronics Communication and Aerospace Technology [ICECA 2019] about how ARIMA can be used on railway sensor data.

2.4 Algorithm Research

Application of temporal sequence mining has not been done before
The occurrence of one fault in a component in heavy machinery or rail can be the resultant of failure of some other component before that or compound effect of many component failures. This can be analysed by using a sequential version of Market Basket Analysis, sometimes called “sequential item-set mining” or “sequential pattern mining”, to introduce a time component to the analysis. When given a series of faults over time, we can determine whether we can find bundles that we expect to be fail simultaneously, and also examine how these failures evolve over time. As mentioned earlier, ARIMA has been used to forecast failure data before and we will draw inspiration from those sources.

Chapter 3

LITERATURE SURVEY AND REVIEW

In [5], For data analysis of the multivariate time series data, linear regression and random forest algorithm are used. First of all, tasks to pre-process are done. This task includes cleaning, integrating, transforming, discretizing, and reducing data. Linear regression model is used to predict the variable dependence on the independent variables observed from the past. The output from this model is then categorized using the learning algorithm of random forest machines to analyze the patterns. Therefore pattern analysis can be performed. Data cleaning is performed in the first step to reduce or delete the date issues which can never be useful in the task of trend analysis. Then, transform and the the rest of the results.Random forest algorithm for classification is applied after transformation and reduction.
Failure prediction is one of the crucial measures for predictive maintenance as it has the potential to avoid failure events and maintenance costs. Mathematical and statistical method modeling for failure predictions has been applied.

The next strategy for predicting failures is the pattern recognition approach[7]. Here, sensor data are collected and analyzed in real-time using pattern recognition approach to predict the occurrence of unusual target events. Infrequent target events are those severe failures which require immediate maintenance attention. The steps in this pattern recognition method for predicting failures are as follows: labelled observation matrix creation from sequence event transformation and hypothesis testing for attribute selection.
Reactive maintenance requires high costs, as it needs maintenance efforts at the last

minute. For predictive maintenance there are essentially two approaches; time guided maintenance and condition-based maintenance. Time-directed maintenance is performed at intervals of difficult times. Predictive or condition-based maintenance is useful in detecting the very beginning of the failure. different schema for predictive maintenance are explored in this article.

In [11], Standard deviation and discrete exceedance are known as the major predictive maintenance technique. Standard deviation is the value by which all the values deviate from the mean. It is used to measure the roughness of the track. In the track, the lower and higher standard deviation value reflects less roughness and high roughness, respectively.It allows the engineers to decide to give the maintenance of which component greater priority. Discrete surpluses have two degrees of surplus.

Level 1 exceedance requires exceedance in alignment, twist, tops etc.

Level 2 exceedance is often labelled for quick visualisation with paint. Maintenance procedures should be triggered if there is any changes in variance.

In [12],Time based maintenance is taken into account. This time-based maintenance program helps to identify the pattern and to track the components effectively. In order to achieve so, in-depth data analysis must first locate the correlation between values, and then reconcile hypothesis correlations with experimental findings.

In [13], A linear regression model is used in this solution . This model helps to find the linear equation coefficients, and thus predicts the dependent variable's value.

Chapter 4

ACTUAL WORK

4.1 Understanding and grouping data

First step: Successive Failures

given a data-set of train-set and train components and their corresponding failure dates , We had to find what unique components a specific train-set had and build a data-frame that consisted concatenated train-set-component numbers along with their fault dates. Then, we found out the difference between each failure date of these Train-set-Component combinations and grouped them by their occurrences. Our final aim for the first phase is to find out which components fail more often of a particular Train-set.

Here, Train-set refers to the engine of a metro and all metro bogies connected to it.

As of now , using the data-set of failures in Train-Lines and Components , we have identified successive fault occurrences in a specific component corresponding to the various Train-Sets (Metro-Bogies).

At first , we have grouped all failure dates corresponding to all combinations of Train-set+Component in a data-frame using groupby function in pandas. This means all successive dates of failures of a particular component of a particular Trains-Set have been listed

Figure Groupby output [Figure B.9]

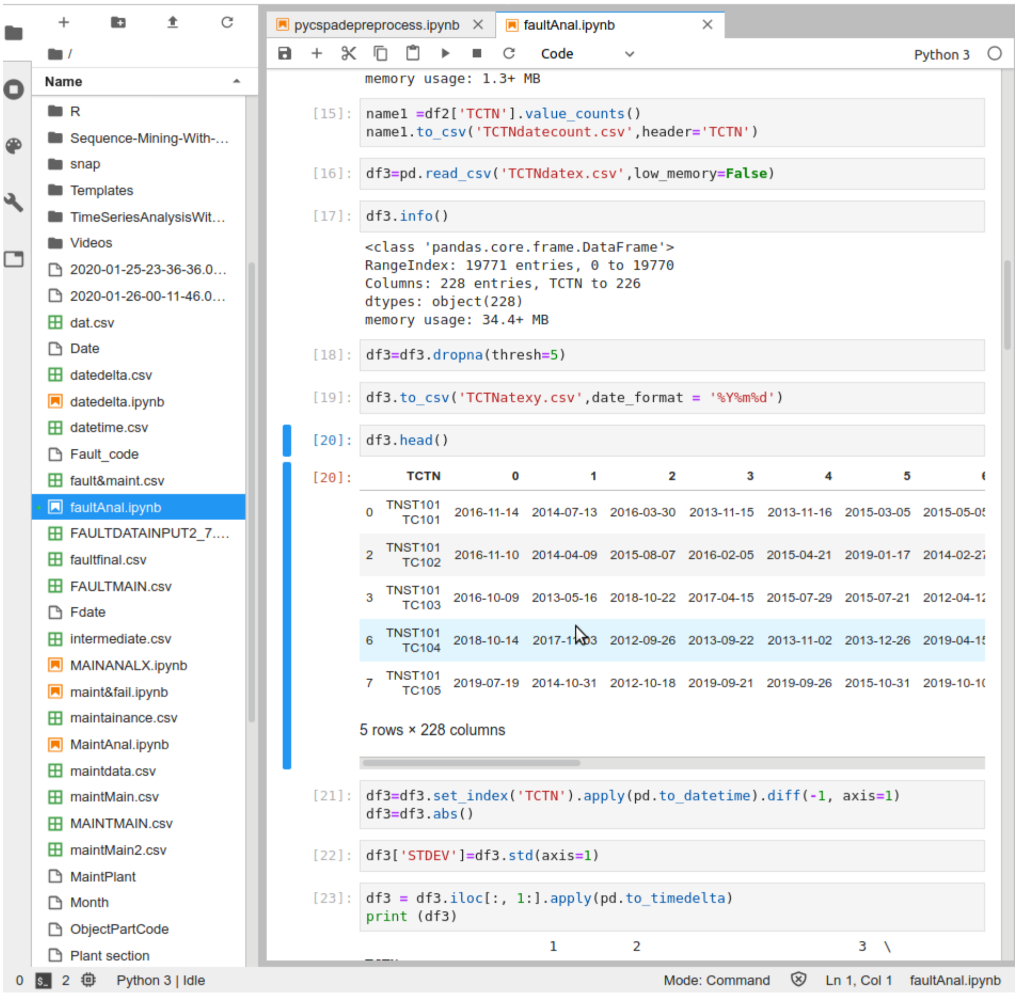


FIGURE 4.1: Groupby output

now using diff operator , we are finding the components of a particular train-set which occur more frequently by subtracting successive date-time values and selecting them. Our goal is to find out which component+Train-set combinations fail more often.

Figure dates between successive occurrences of failures [Figure B.9]

```
[25]: df3 = df3.apply(lambda x: x.dt.days)
      print (df3)
```

```
                     1     2      3      4     5     6    7    8      9    10
\
TCTN
TNST101 TC101      626   866      1    474    61   149  938  354    616  1456
TNST101 TC102      485   182    290   1367  1785   350  280  451  90802     0
TNST101 TC103     1985   555    626      8  1195   864  624   21   1032  1069
TNST101 TC104     1864   361     41     54  1936  1931   58  464     63  1112
TNST101 TC105      743  2529      5   1426  1440  2171  730  355    574  1222
...                ...   ...    ...    ...   ...   ...  ...  ...    ...   ...
TNST344 TC148      159    98  88663      0     0     0    0    0      0     0
TNST344 TC198      153    66    310  88845     0     0    0    0      0     0
TNST344 TC284       90   307  88812      0     0     0    0    0      0     0
TNST344 TC293      153   108  88586      0     0     0    0    0      0     0
TNST345 TC148        8    41  88598      0     0     0    0    0      0     0

                  ...  218   219  220   221  222   223    224  225  226  STDE
V
TCTN              ...
TNST101 TC101     ...  598  1927   41  1841  474  1227  89194    0  NaN   591
1
TNST101 TC102     ...    0     0    0     0    0     0      0    0  NaN   604
0
TNST101 TC103     ...    0     0    0     0    0     0      0    0  NaN   597
4
TNST101 TC104     ...    0     0    0     0    0     0      0    0  NaN   606
2
TNST101 TC105     ...    0     0    0     0    0     0      0    0  NaN   601
0
...               ...  ...   ...  ...   ...  ...   ...    ...  ...  ...
...
TNST344 TC148     ...    0     0    0     0    0     0      0    0  NaN   589
7
TNST344 TC198     ...    0     0    0     0    0     0      0    0  NaN   590
9
TNST344 TC284     ...    0     0    0     0    0     0      0    0  NaN   590
7
TNST344 TC293     ...    0     0    0     0    0     0      0    0  NaN   589
2
TNST345 TC148     ...    0     0    0     0    0     0      0    0  NaN   589
3

[4997 rows x 227 columns]
```

FIGURE 4.2: dates between successive occurrences of failures

Since we know all Train-Sets are sent to the workshop for maintenance on the 15 th of every month, we are creating a dummy data-set to club with these failure dates . Now, we are concentrating on combinations of the failure and maintenance dates:

1. **Failure after Failure** (find out which components are failing more often)
2. **Failure after Maintenance** (find out which components are failing even after repeated
3. **Maintenance after Failure** (not important/ insignificant)
4. **Maintenance after Maintenance** (not important / insignificant)

So, we will merge the failure date data-frame and the dummy maintenance data-frame . After merging the data we will concentrate on these combinations. Before performing this task we are going to associate labels (f and m) with failure and maintenance dates respectively and iterating through the merged data-set of Failure and Maintenance we will find difference between Failure after Failure and Failure after Maintenance dates.

Figure Combinations[Figure B.9]

666	TNST101TC143	2014-05-27	f	316 days 00:00:00.000000000
667	TNST101TC143	2014-05-28	f	1 days 00:00:00.000000000
668	TNST101TC143	2014-05-29	f	1 days 00:00:00.000000000
669	TNST101TC143	2014-06-15	m	
670	TNST101TC143	2014-08-12	f	58 days 00:00:00.000000000
671	TNST101TC143	2016-01-04	f	510 days 00:00:00.000000000
672	TNST101TC143	2016-02-13	f	40 days 00:00:00.000000000
673	TNST101TC143	2016-05-24	f	101 days 00:00:00.000000000
674	TNST101TC143	2017-01-08	f	229 days 00:00:00.000000000
675	TNST101TC143	2017-11-15	f	311 days 00:00:00.000000000
676	TNST101TC143	2018-06-07	f	204 days 00:00:00.000000000
677	TNST101TC143	2019-01-26	f	233 days 00:00:00.000000000
678	TNST101TC143	2019-10-27	f	274 days 00:00:00.000000000
679	TNST101TC143	2019-10-28	f	1 days 00:00:00.000000000

FIGURE 4.3: Combinations

4.2 Implementation of Sequence of Mining

Now we can apply SPADE algorithm for mining temporal frequent patterns on our data-set . This algorithm is used to discover the Sequential Patterns quickly. Current solutions to this problem allow repetitive scans of the database and use complex hash structures with poor loacality. SPADE uses combinatorial properties to break down the original problem into smaller sub-problems that can be solved independently in the main memory using lattice search techniques, and by using simple join operations. We will be using a library called pycspasde for python. This is a python wrapper for the C++ implementation of C-SPADE algorithm by the author, **Mohammed J. Zaki.** The input for the algorithm is of the following format:

```
1 2 4 4 11 37 42
```

```
2 1 2 10 73
2 2 1 72
2 3 3 4 24 77
```

The first number is the sequence index, the second is the event index, the third is the number of elements, followed by the element, space separated. In our case the first index will be the sequence index , the second number will represent the TRAIN-SET , the third will be the number of elements (COMPONENTS) corresponding to that TRAIN-SET followed by the COMPONENTS in a sequential and temporal order. IE We are showing whichever Components of the TRAINS are failing in a sequential manner. This will need some preprocessing and we will show how we are doing it. After we have preprocessed the data , we will apply spade to it. There are a lot of parameters that can be passed to this function. most important ones are:

1. **support**: this is the minimum support level, default to 0 (not excluding anything)
2. **max-gap**: The max number of item-set that can be skipped in a sequence repeated
3. **min-gap**: The min number of item-set that must be skipped in a sequence

Before applying SPADE we will segregate the Data based on the TRAIN-lines(here,"Maint-Plant").
Say, our input data is of this format:

Figure Input data-sheet[Figure B.9]

```python
import pandas as pd
import datetime as dt
import csv
from pycspade.helpers import spade, print_result
```

```python
l=pd.read_csv(r'/home/devarshi/FAULTDATAINPUT2_7.csv',low_memory=False)
```

```python
l
```

	ObjectPartCode	TRAINSET	Fault_code	Plant section	MaintPlant	Date	Month	Year	Fdate	LABEL
0	TC134	TNST115	FC104	PS1	LN1	20	7	2016	20/07/16	f
1	TC117	TNST125	FC104	PS1	LN1	8	1	2013	08/01/13	f
2	TC263	TNST120	FC107	PS1	LN1	20	8	2012	20/08/12	f
3	TC221	TNST109	FC130	PS1	LN1	31	1	2015	31/01/15	f
4	TC212	TNST126	FC130	PS1	LN1	11	1	2013	11/01/13	f
...	...	...	...	...	...	...	...	...	...	...
84563	TC710	TNST250	FC185	PS3	LN1	18	10	2019	18/10/19	f
84564	TC757	TNST307	FC197	PS3	LN1	4	5	2019	04/05/19	f
84565	TC420	TNST278	FC174	PS2	LN2	16	7	2016	16/07/16	f
84566	TC573	TNST302	FC178	PS2	LN2	16	7	2016	16/07/16	f
84567	TC664	TNST298	FC191	PS2	LN2	16	7	2016	16/07/16	f

84568 rows × 10 columns

FIGURE 4.4: input data-sheet

We need to do a few preprocessing tasks before applying spade:

1. Drop all the the unnecessary fields for sequence mining task. All we need is the TRAIN-SET , Components(here,“ObjectPartCode”) corresponding to those TRAIN-SETs and the dates of failure (“Fdate”) .
2. converting the Fdate column to datetime64ns format and bringing to appropriate “
3. Stripping the ‘TC’ part of our Component. Remember, this is the TRAIN-SET, Fdate and Component ID of only Line1 (“LN1”) from the above table.

Remember, this is the TRAIN-SET, Fdate and Component ID of only Line1 (“LN1”) from the above table.

Figure LN-1 processed[Figure B.9]

	TRAINSET	Fdate	ObjectPartCode
0	TNST114	02-Jan-12	101
1	TNST114	02-Jan-13	192
2	TNST114	07-Jan-12	104
3	TNST114	08-Jan-12	115
4	TNST114	08-Jan-12	104
...	...	...	...
18868	TNST342	31-Mar-19	148
18869	TNST342	31-Mar-19	127
18870	TNST342	31-Mar-19	132
18871	TNST342	31-Mar-19	1036
18872	TNST342	31-Mar-19	203

18873 rows × 3 columns

FIGURE 4.5: LN-1 processed

This is how we preprocess the above file to bring it into the format accepted by Pycspade.

Figure preprocessing code[Figure B.9]

```
i=0
for row in inputfile:
    if(i>0):
        out_str='I'
        customer=row[0]
        customer=customer.split('TNST')
        customer=customer[1]
        customer=int(customer)-100
        date=row[1]
        date=date.split('-')
        date=date[2]
        item=row[2]
        item=int(int(item))
        out_str+=str(item)
        seq_event[out_str]=[]
        d[str(customer)+"-"+str(date)]=[]
        if date not in event_id[customer]:
            event_id[customer].append(date)
        outputfile.write(str(customer)+" "+date+" "+out_str+"\n")
        event_id[customer].sort()
    i+=1
```

```
i=0
inputfile = csv.reader(open('/home/devarshi/Desktop/segregatedLines/ln1.csv','r'))
for row in inputfile:
    if(i>0):
        out_str='I'
        customer=row[0]
        customer=customer.split('TNST')
        customer=customer[1]
        customer=int(customer)-100
        date=row[1]
        date=date.split('-')
        date=date[2]
        item=row[2]
        item=int(int(item))
        out_str+=str(item)
        if out_str not in d[str(customer)+"-"+str(date)]:
            d[str(customer)+"-"+str(date)].append(out_str)
        d[str(customer)+"-"+str(date)].sort()
    i+=1
```

FIGURE 4.6: preprocessing code

After doing this we save it as a .txt file which acts as the input for our spade algorithm.

Figure preprocessing output[Figure B.9]

```
1 14 56 101 102 103 104 105 106 113 114 115 118 120 123 125 126 127 132 134 135 143 148 157 159 162 164 166 167 175 179 188 189 192 200 203 211 212 214 216 221 224 240 255 258 268 270 274 293 294 295 310 324
335 361 366 381 382 512
1 15 46 101 102 103 104 105 111 113 114 119 123 126 127 148 149 153 157 159 162 165 166 179 184 188 189 192 201 203 204 211 213 226 239 257 268 270 275 290 293 326 333 343 346 393 412 461 518
1 18 46 101 103 104 105 111 113 114 119 121 123 125 129 131 132 134 135 143 147 148 150 153 157 163 166 180 195 207 211 222 225 240 255 268 270 272 284 293 310 311 357 391 393 402 479 520 562
1 12 51 101 102 103 104 105 107 111 113 114 115 120 121 123 125 129 130 132 143 148 152 153 158 159 165 179 182 185 188 195 198 200 201 206 211 222 235 239 240 255 266 268 270 277 283 305 309 310 319 340 359
382
1 13 56 101 102 103 104 105 108 111 113 114 115 116 123 125 130 132 134 138 143 148 152 153 157 158 159 162 164 166 182 184 188 189 192 195 198 200 201 203 206 211 214 222 240 253 258 266 285 310 312 324 340
346 359 376 401 446 504
1 16 39 101 102 103 105 111 113 114 115 116 118 123 125 126 129 132 134 138 143 148 149 153 157 162 165 188 195 209 211 225 239 264 268 270 290 296 299 402 446 479
1 19 75 101 1012 102 103 1033 1036 104 105 106 1061 1062 108 1094 1108 111 1123 113 114 115 119 120 123 125 126 129 130 131 134 139 143 147 148 149 150 153 157 160 162 164 165 166 167 174 188 200 203 211 213
220 221 222 225 240 270 284 290 293 295 301 309 321 326 342 344 346 353 359 391 446 449 516 519 555 678 769
1 17 33 101 103 104 106 111 113 123 125 126 129 132 134 143 148 149 161 174 188 200 206 211 212 225 252 270 290 293 296 314 340 362 522 562
2 13 52 101 102 103 104 106 111 113 114 115 116 119 122 123 125 129 132 134 135 143 147 148 153 159 160 164 165 166 171 179 188 189 192 193 201 203 206 211 213 220 230 240 253 255 266 270 295 346 359 388 446
469 494
2 16 44 101 102 105 106 113 117 119 122 123 125 127 131 133 134 135 138 143 147 148 161 165 166 167 186 189 194 195 198 203 219 221 229 240 253 255 258 268 270 278 293 359 362 366 440
2 17 47 101 102 103 104 105 111 113 114 115 118 123 125 126 129 133 134 135 143 148 155 159 160 161 164 165 174 188 189 201 203 206 211 212 240 251 255 275 276 293 359 446 452 472 479 501 518 842
2 18 40 101 103 104 111 113 114 115 119 123 124 126 129 132 134 138 143 147 148 149 151 157 159 165 188 200 203 206 220 240 255 276 299 316 344 437 446 466 514 542 609
2 14 54 101 103 104 106 111 113 114 115 118 119 123 125 127 134 147 148 153 154 158 159 161 164 166 179 184 186 189 192 198 200 201 203 206 211 212 213 221 222 225 226 229 235 243 266 272 280 281 295 310 323
335 357 359 381
2 15 51 101 102 103 104 113 114 119 120 122 123 125 126 127 128 129 130 135 148 149 154 159 161 162 165 166 167 174 189 206 212 213 220 226 253 266 270 275 310 314 324 326 339 340 357 362 376 391 418 446 482
520
2 12 43 101 103 104 106 113 115 116 123 125 132 134 139 143 148 158 159 164 165 172 186 188 198 201 203 213 215 221 226 231 239 255 258 270 277 293 294 337 352 357 361 388 391 393
2 19 57 101 103 1036 1041 1045 105 1058 106 1061 108 111 113 114 115 118 119 120 123 125 126 127 134 143 147 148 153 157 163 165 166 180 188 189 195 200 203 206 211 214 218 220 229 251 258 270 274 284 291
293 299 304 309 312 357 449 479 555
3 19 58 101 102 103 1033 1036 104 1041 1045 105 106 111 113 114 119 120 122 123 125 126 130 134 139 143 147 148 153 157 163 164 166 188 189 193 194 200 213 214 216 225 240 247 255 258 260 264 270 278 284 293
301 310 312 326 340 446 582 609 769
3 13 55 101 102 103 104 106 113 116 119 123 125 126 132 134 138 143 148 153 [illegible] 159 161 162 164 166 171 174 182 184 189 192 200 201 203 211 212 220 221 229 239 240 249 253 255 258 266 270 303 324 340 359 361
369 382 391 401 456
3 16 45 101 103 104 106 111 113 114 115 116 118 120 123 126 129 130 131 134 135 136 162 163 164 166 167 180 192 198 200 203 206 211 214 218 220 221 229 247 248 255 293 310 326 361 381 800
3 17 32 101 104 105 106 113 114 118 122 123 125 126 131 134 143 149 152 157 161 163 166 188 200 203 213 220 229 276 310 323 359 376 479
3 15 48 101 102 103 104 105 113 114 116 118 119 122 123 125 127 129 133 134 143 159 161 162 165 166 188 192 199 203 212 213 219 220 221 224 243 253 261 268 270 284 295 317 362 381 401 471 524 531 645
3 18 42 101 103 104 111 113 114 115 119 120 122 123 126 129 132 134 135 143 148 157 159 161 164 166 174 180 200 203 211 212 213 229 231 240 255 256 270 284 293 310 446 499 516
3 14 59 101 102 103 104 106 111 113 114 115 119 120 123 127 129 134 135 138 143 147 148 149 153 157 159 161 164 166 182 188 189 198 203 205 206 211 212 213 214 221 225 229 231 239 243 247 258 261 266 268 270
281 293 295 310 334 335 362 368 412
3 12 49 101 102 103 104 111 113 116 118 119 123 125 132 134 143 148 149 152 154 159 161 166 182 198 211 213 220 239 253 258 260 266 268 270 278 284 292 293 299 305 310 332 345 346 361 385 406 423 430 449
4 13 52 101 102 103 104 105 106 108 111 113 114 116 120 123 125 135 143 147 148 149 153 157 159 161 162 164 166 169 179 188 198 200 201 203 206 211 212 213 214 221 222 231 235 240 270 295 340 344 359 362 382
392 438
4 16 53 101 102 103 105 106 111 113 114 115 120 121 123 125 127 134 135 143 148 157 161 164 166 175 186 194 195 198 200 203 206 211 213 214 224 225 246 257 258 268 270 276 277 293 310 362 391 393 418 452 461
482 486 522
4 12 37 101 102 103 105 111 113 114 115 117 120 123 125 126 129 138 148 157 159 161 162 164 166 169 206 211 213 221 222 239 253 270 271 310 358 373 412 429
4 15 64 101 102 103 104 106 111 113 114 115 120 122 123 125 126 127 134 135 141 143 148 149 153 158 159 160 161 164 166 167 179 188 189 192 200 201 203 204 212 213 214 221 225 226 243 252 255 274 277 286 295
310 313 317 324 333 340 344 362 414 437 452 482 526 540
4 18 41 101 102 103 1032 104 105 111 114 116 119 123 134 143 147 148 149 153 157 161 164 166 174 180 189 200 203 205 211 214 225 255 264 293 309 310 323 326 362 368 390 472
4 17 37 101 102 103 105 106 111 113 115 119 123 134 139 145 148 159 161 166 188 198 200 202 203 212 213 215 224 225 258 275 309 310 313 323 344 446 449 472
4 19 61 101 102 103 1033 104 1041 1045 105 106 1061 1090 1119 113 114 115 1162 118 119 120 122 123 126 133 134 135 141 143 148 153 157 163 165 166 188 189 198 201 203 204 206 210 212 213 214 243 253 268 284
293 326 334 362 433 449 463 504 516 522 555 582 921
4 14 57 101 102 103 104 105 106 111 113 114 115 120 122 123 125 126 134 143 148 149 153 159 161 162 165 166 169 170 188 192 201 203 206 211 212 221 223 239 251 253 255 266 268 270 277 284 323 326 344 355 368
373 381 393 424 461 471 482
5 12 37 101 102 103 104 105 106 113 115 116 121 122 123 125 127 129 133 134 143 148 159 162 186 189 198 206 211 222 247 253 255 270 280 293 310 340 361 395
5 14 66 101 102 103 104 105 106 108 111 113 114 115 119 120 123 125 126 127 129 133 134 135 138 143 147 148 157 159 161 164 165 166 174 179 188 189 192 193 200 201 203 210 211 212 215 220 221 225 229 243 251
```

FIGURE 4.7: preprocessing output

Now since we have 5 lines (“LN1”-“LN5”) in our main input data-set , we have to do the aforementioned steps 5 times after segregation of Lines. This separation of the Lines is done because Each Line in Delhi metro contains separate trains and each train contains separate components.e Now we can apply spade algorithm using the wrapper. We are keeping the support count as 0.5. High support values correspond to commonly-found item-sets that are applicable to many transactions.

Figure SPADE output[Figure B.9]

```python
import csv
import pandas as pd
from pycspade.helpers import spade, print_result
```

```python
resultln1 = spade(filename='dln1.txt', support=0.5, parse=True)
rln1=pd.DataFrame(resultln1)
```

```python
resultln2 = spade(filename='dln2.txt', support=0.5, parse=True)
rln2=pd.DataFrame(resultln2)
```

```python
resultln3 = spade(filename='dln3.txt', support=0.5, parse=True)
rln3=pd.DataFrame(resultln3)
```

```python
resultln4 = spade(filename='dln4.txt', support=0.5, parse=True)
rln4=pd.DataFrame(resultln4)
```

```python
resultln5 = spade(filename='dln5.txt', support=0.5, parse=True)
rln5=pd.DataFrame(resultln5)
```

```python
rln1
```

	nsequences	seqstrm	logger	summary	mined_objects
0	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(101) - [53]
1	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(113) - [49]
2	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(114) - [47]
3	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(134) - [47]
4	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(255) - [76]
5	93	101 -- 53 53 \n113 -- 49 49 \n114 -- 47 47 \n1...	CONF 93 1190 37.1679 5.69892\nargs.MINSUPPORT ...	CONF 93 1190 5.69892 37.1679 530 1 245 18.4493...	(293) - [49]

FIGURE 4.8: SPADE output

4.3 Auto-regressive Integrated Moving Average

After we have applied and analyzed the results of SPADE we move forward to building a time-series of the fault data. We do this by segregation of one TRAINSET+TRAIN COMPONENT from the main fault datasheet.Preferably with the most occurrences of faults. In our case it is TNST101TC101. TNST101 corresponds to the trainset and TC101 is the TrainComponent 101. So TNST101TC101 has a total of 245 fault occurrences. These occurrences are marked by their dates. From the dates, we can derive another parameter of DATEDELTA which is the number of days between each successive occurrence of faults for a given TRAINSET+COMPONENT. This is done because to model a time-series we need a series of data points indexed in time order. How we found the date-delta has been mentioned in previous reports. But we can see that this is not a uniform time series, i.e, the measurements are not properly ordered according to uniform time gaps. Hence we have to impute the values using some distribution or find other ways to model the time-series. In statistics the method of replacing missing data with replacement values is imputation. It is known as ”unit imputation” when replacing a data point component; it is known as ”item imputation” when replacing a data point component. One of the most used imputation methods is an interpolation. It is normal to take measurements at irregular intervals, but most instruments are designed primarily for even-spaced measurements. In the real world, time series may have missing observations, or you might have several series of different frequencies: modeling these as unevenly spaced as well can be useful.

Figure time-series decomposition of TNST101TC101[Figure B.9]

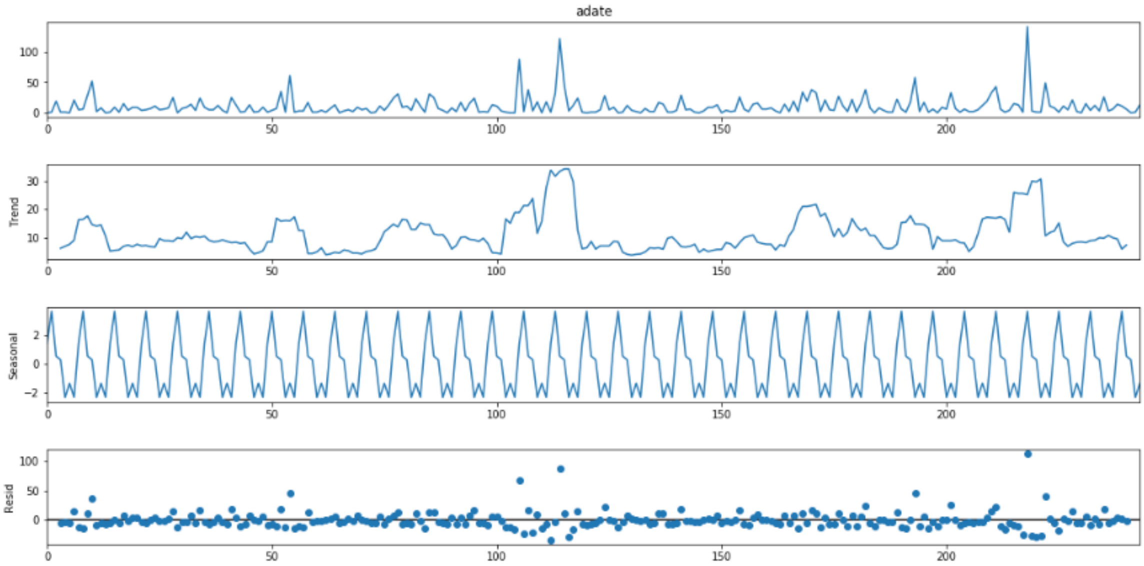


FIGURE 4.9: time-series decomposition of TNST101TC101

Using **seasonal_decompose** method of stats-models we decompose the components of the unevenly spaced time series first. The first graph corresponds to the actual time series and the others are TREND , SEASONALITY and RESIDUAL components respectively.

- **Level**: AVG value in the series under decomposition.
- **Trend**: The uptrend or downtrend , ie increase or decrease in value of series
- **Seasonality**: Cyclical repetitions observed in the series , can be monthly , weekly etc.
- **Resid**: The degree of randomness in the series.

Since this is not showing any visible trends , we will try linearly interpolating the data. In mathematics linear interpolation is a curve fitting method that uses linear polynomials to construct new data points within the range of a discrete set of known data points.

Figure linear interpolation[Figure B.9]

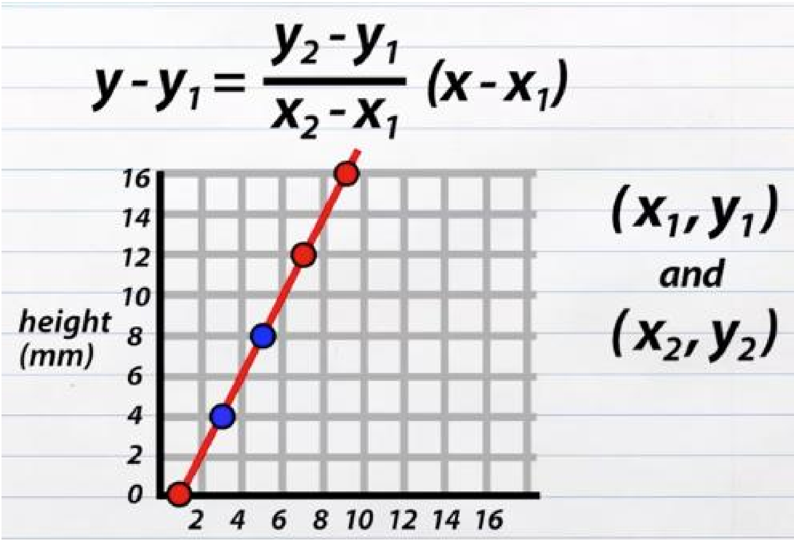


FIGURE 4.10: linear interpolation

After linearly interpolating the data , we can try decomposing the components of the time-series again.

Figure linear interpolation[Figure B.9]

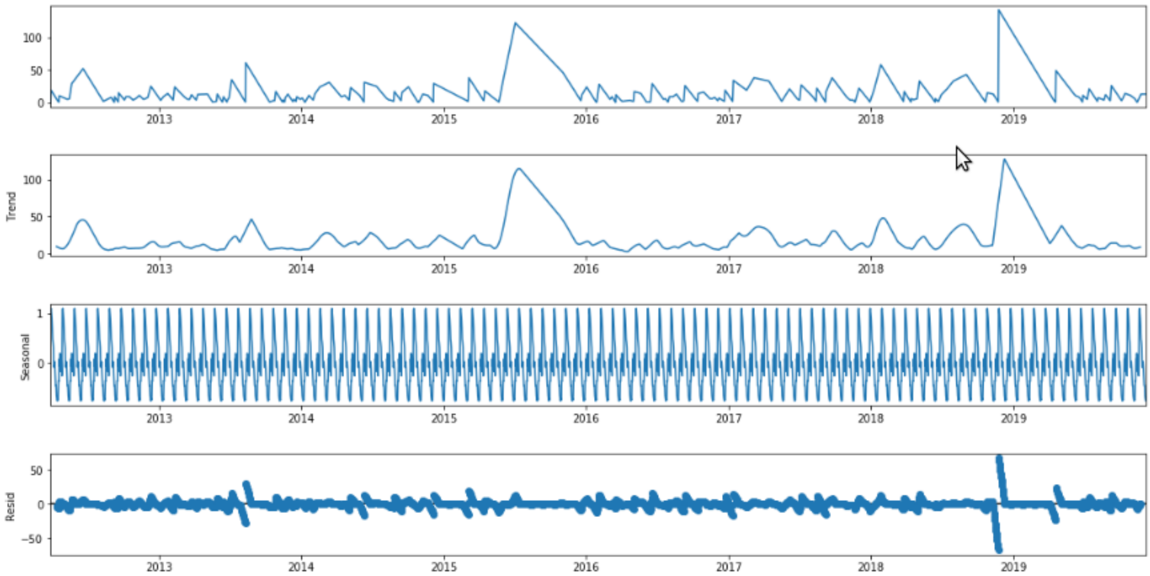


FIGURE 4.11: linear interpolation

Here is the plot of the interpolated data.the red dots are the actual data points and the blue dots correspond to the made up / interpolated data. We can see that doing this gives us a lot more data points which are uniformly distributed . We can fit an ARIMA model seamlessly into this data, but it probably wont give us appropriate results.

Figure interpolated data[Figure B.9]

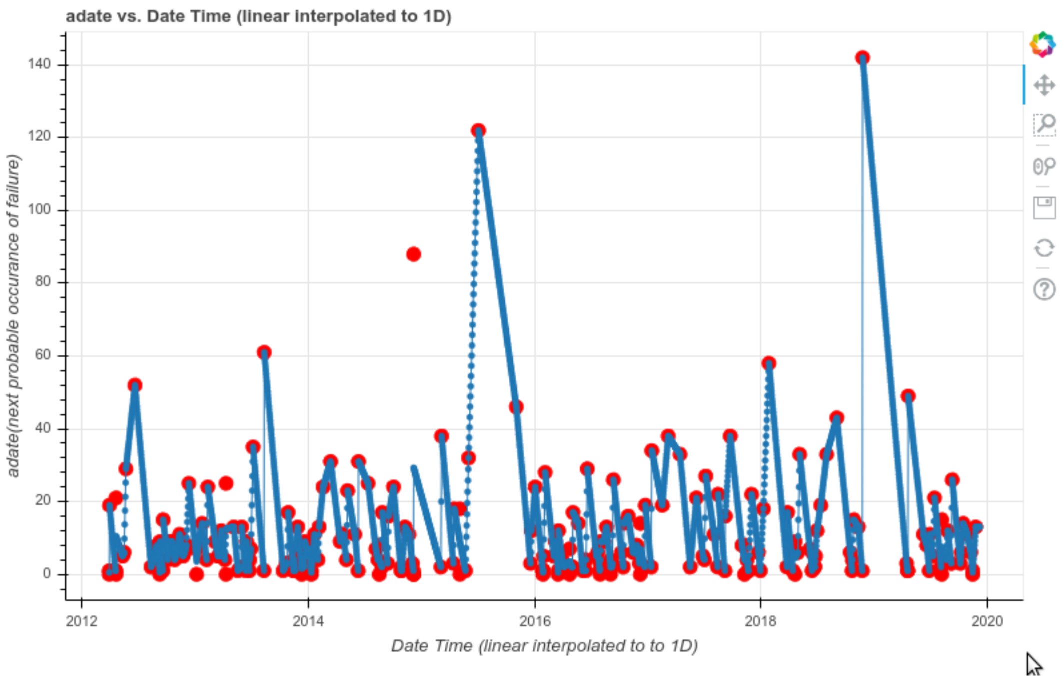


FIGURE 4.12: interpolated data

Still, we cannot find out any trend that seems like it can be modeled.Before doing anything further we can check for auto-correlation. Auto-correlation refers to the degree of correlation between the values of the same variables across different observations in the data. The concept of auto-correlation is most often discussed in the context of time series data in which observations occur at different points in time. An auto-correlation plot is designed to show whether the elements of a time series are positively correlated, negatively correlated, or independent of each other. (The prefix auto means "self"— Auto-correlation refers directly to the correlation between the elements in a time series.) An auto-correlation plot displays the auto-correlation function (acf) meaning on the vertical axis. It can vary between -1 and 1. The horizontal axis of an auto-correlation plot Displays the size of a lag between time series components. For instance, the auto-correlation with lag 2 is the correlation between the elements of the time series and the corresponding elements observed two time periods before.

Figure ACF, PACF before differenciation[Figure B.9]

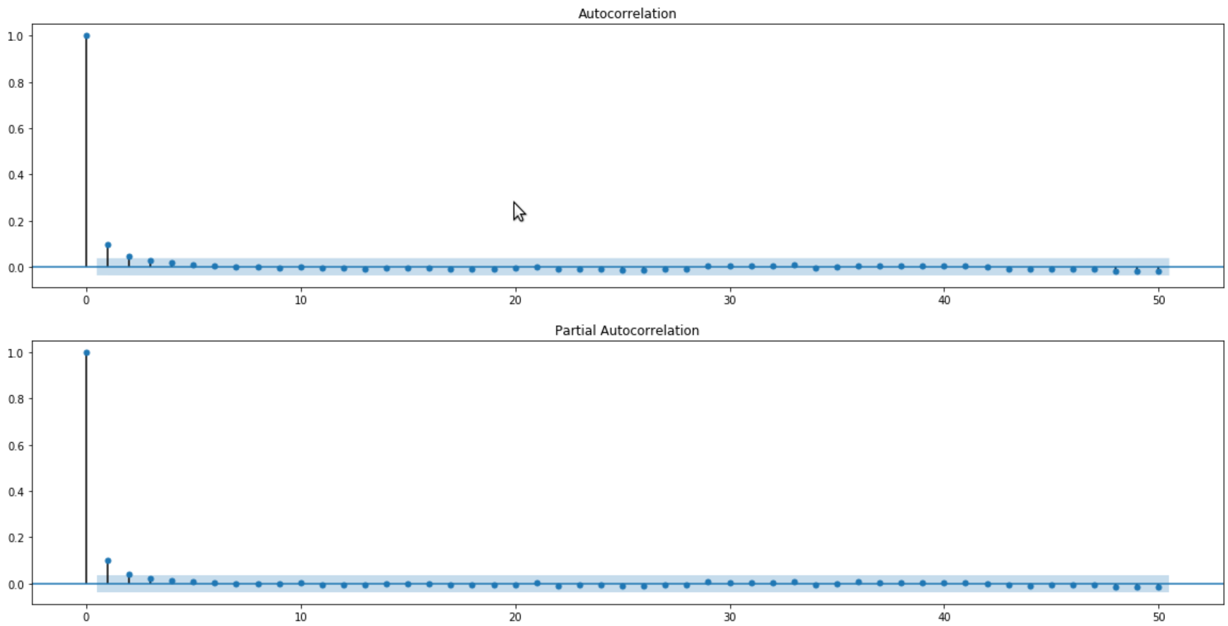


FIGURE 4.13: ACF, PACF before differenciation

The auto-correlation with lag zero is always equal to 1, since this is the auto-correlation between each term and itself. We couldn't see any auto-correlation in the interpolated time-series as well hence we try differencing the data, IE, we consider it as non-stationary and do a few stationarity checks. Time series are stationary unless they have pattern or seasonal impacts. Over time overview statistics measured on the time series are consistent, such as the mean or variance of the observations. Modeling can be easier if a time series is stationary. Methods of statistical modeling assume that the time series is stationary or demands that it be accurate. Statistical tests assume strongly about your results. They can only be used to tell to what degree a null hypothesis can be rejected or not rejected. For a given question to be relevant, the result has to be interpreted. They may also provide a simple test and confirmatory proof that your time series is stationary or non-stationary. Statistical tests assume strongly about your results. They can only be used to tell to what degree a null hypothesis can be rejected or not rejected. For a given question to be relevant, the result has to be interpreted. They may also provide a simple test and confirmatory proof that your time series is stationary or non-stationary.It uses an autoregressive model, maximizing the criterion of knowledge over several different lag values. The test's null hypothesis is that a unit root should represent the time series, because it is not stationary (it has some time-dependent structure); The alternative hypothesis (rejecting the null hypothesis) is that it is constant in the time series.

- **Null Hypothesis (H0)**: If not refused, it implies that the time series has a unit root which means it is non-stationary. It has dependent structure for some time.
- **Alternate Hypothesis (H1)**: The null hypothesis is rejected; it implies that the time series has no unit root which means it is stationary. It has no structure which depends on time.

The p value given by augmented dickey fuller test conveys the following about the stationarity of times-series:

- **p-value greater than 0.05**: If the null hypothesis (H0) is not dismissed, the data has a root unit, and is non-stationary.
- **p-value less than or equal to 0.05**: Reject the null hypothesis (H0), the data is stationary and lacks a unit root.

Figure Augmented Dickey Fuller test[Figure B.9]

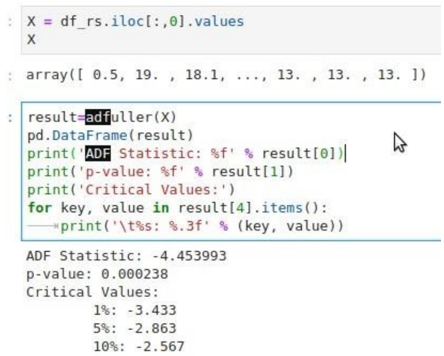


FIGURE 4.14: Augmented Dickey Fuller test

Differentiation is a common and commonly used transformation of data to render the data from time series stationary. It can be used to eliminate time dependency, the so-called temporal dependency, from the sequence. After differentiation we try to find the acf and pacf of interpolated data.

Figure ACF, PACF after differentiation[Figure B.9]

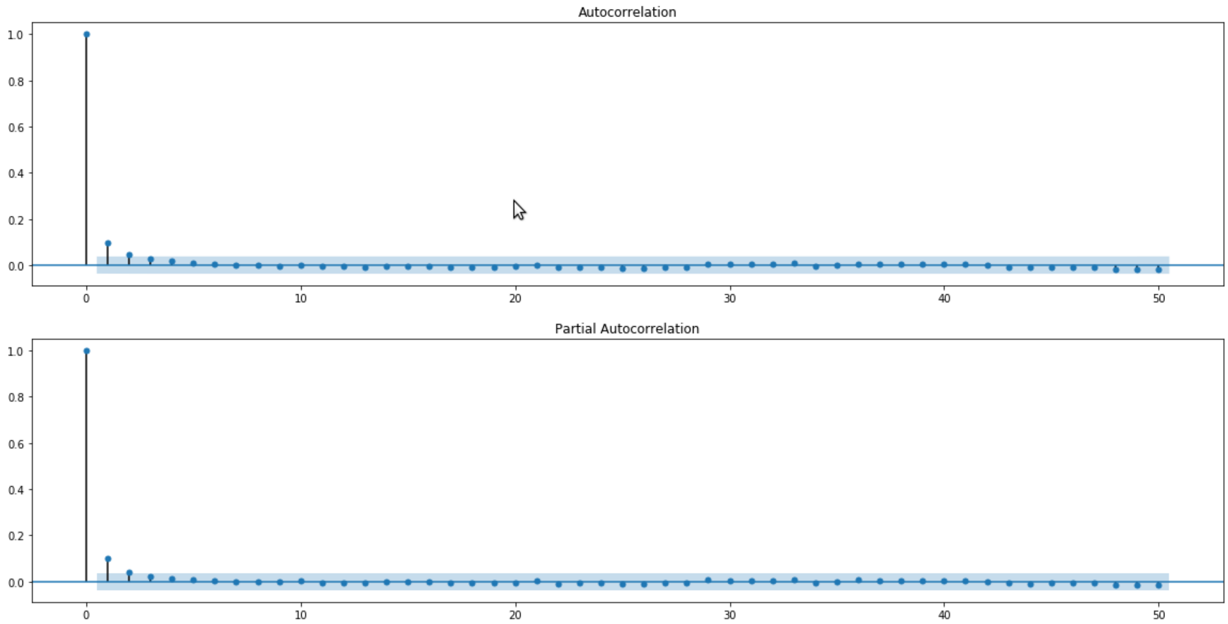


FIGURE 4.15: ACF, PACF after differentiation

Hence we can see here that there is no auto-correlation even after differencing. Hence after all the tests we find that our time-series has no unit root, and in turn that the time series is stationary or does not have time-dependent structure.

So our goal main now is to find out different ways to model /build our time-series which may be more appropriate. Although, we can try fitting and forecasting using ARIMA on the interpolated data for the sake of our understanding. The ARIMA model is a growing, commonly used, statistical method for time series prediction. ARIMA represents an acronym for the combined moving average Auto-Regressive. This is a model class collecting a suite of various time series data from specific standard temporal structures. It is a concisely defined term which describes the main elements of the model itself.

- **AR: Auto-regression**:A model that uses the dependent relationship between an observation and some number of lagged observations.
- **I: Integrated**: The use of differencing of raw observations (e.g. subtracting an observation from observation at the previous time step) to make the time series stationary.
- **MA: Moving Average**:A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The parameters of the ARIMA model are defined as follows: p: The number of lag observations included in the model, also called the lag order. d: The number of times that the raw observations are differenced also called the degree of difference. q: The size of the moving average window, also called the order of moving average. We need to find the best fit of these hyper-parameters for the ARIMA model to give appropriate forecasts.

4.4 Parameter Estimation and model evaluatio

4.4.1 Grid Searching Hyper-parameters

Instead of searching manually for the perfect ARIMA hyper-parameter that approximates our time-series model perfectly, we will apply a different method named grid searching to estimates the perfect p, d and q by building the ARIMA model for a grid (an array of different p ,d and q values) of of input parameters and build models individually for those , test it against a test partition of the same data and check for the parameters giving the least error. The hyper-parameters with the least error is our required p, d and q. The steps for our model evaluation are:

1. Split the dataset into training and test sets.
2. Walk the time steps in the test dataset. repeated
3. Train an ARIMA model.
4. Make a one-step prediction.
5. Store prediction; get and store actual observation.
6. Calculate error score for predictions compared to expected values.

In Python, we can implement this as a new standalone function named $evaluatearimamodel()$ that needs a time series data set as input as well as a tuple with the parameters p, d, and q for the model to be assessed. The data set is divided into two parts: 66 percent for the initial training dataset and 34 percent for the test data set. The test set is iterated every step of the way. Only one iteration offers a guide for making assumptions on new data that you might use. The iterative approach allows for the training of a new ARIMA model every time phase. Each iteration is rendered a prediction, and stored in a list. It is so that all forecasts can be matched with the list of predicted values and a measured error score at the end of the test set. In this case it measures and returns a mean squared error value.

```
# evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error
```

It's fairly easy to determine a suite of parameters.

To iterate the user will define a grid of the parameters p, d, and q ARIMA. For each parameter, a model is built, and its output is evaluated by calling the function $evaluatearimamodel()$ mentioned in the previous section.

The role has to keep track of the lowest observed error score and the configuration which caused it. This can be summarized with a print to standard out at the end of the feature. $evaluate_models()$ function can be implemented as a series of four loops. There are two other dimensions of this. The first is to ensure input data are floating point values (as opposed to integers or strings), as this can cause the ARIMA system to fail. Second, the ARIMA statsmodels method uses numerical optimisation methods internally to find a set of coefficients for the model. Such procedures will fail which may throw an exception in effect. We have to catch these exceptions and skip the configurations which cause a problem. This happens more frequently than you'd have thought.

The complete function to grid search ARIMA hyperparameters is as follows:

```
# evaluate combinations of p, d and q values for an ARIMA model
```

```
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
```

4.4.2 Akaike inforamtion criterion

Another way of validating our model is by using the Akaike Criterion for Details. A commonly used indicator of a statistical model is the Akaike Knowledge Critera (AIC). It quantifies essentially 1) the fitness goodness, and 2) the model's simplicity / parsimony into a single statistic. AIC use is much faster than grid parameter-search. The model with lower AIC is always better.

AIC is used as error evaluation criterion when $pmdarima.arima.auto_arima$ library is used to build ARIMA model. We will use it to further validate our built model.

4.5 Drawbacks

The time-series that we developed form selecting a specific component of a specific train-set had to be imputed because the raw data only contained 245 rows , which is quite less train a time-series forecasting model like ARIMA perfectly to predict future failure occurrences with minimal error. What we have there is not just an irregularly spaced time series but also one that has multiple observations for a single point in time .

4.5.1 Imputing data

Imputed/Interpolated data cannot really capture the trend of the data because of multiple reasons, let us explore those: Error making is simple. Because uniformly spaced time series are typically stored without timestamps (in an array, along with the start time and time interval), it is simple to use the wrong time units — conversions are notoriously prone to error — or to mess up while the data is being registered. It causes bloat. If you have data from several sources of different time resolution, the normal method is to sample as small as possible. You may end up with a time series that is sampled from a sensor every second, sampling every hour. In addition to the practical problems, there are technological reasons to be vigilant when translating unequal data into standard time series includes: Loss of data, and dilution. You lose close-spaced data, and add redundant data points when the data is too sparse.

Figure downsides of imputation[Figure B.9]

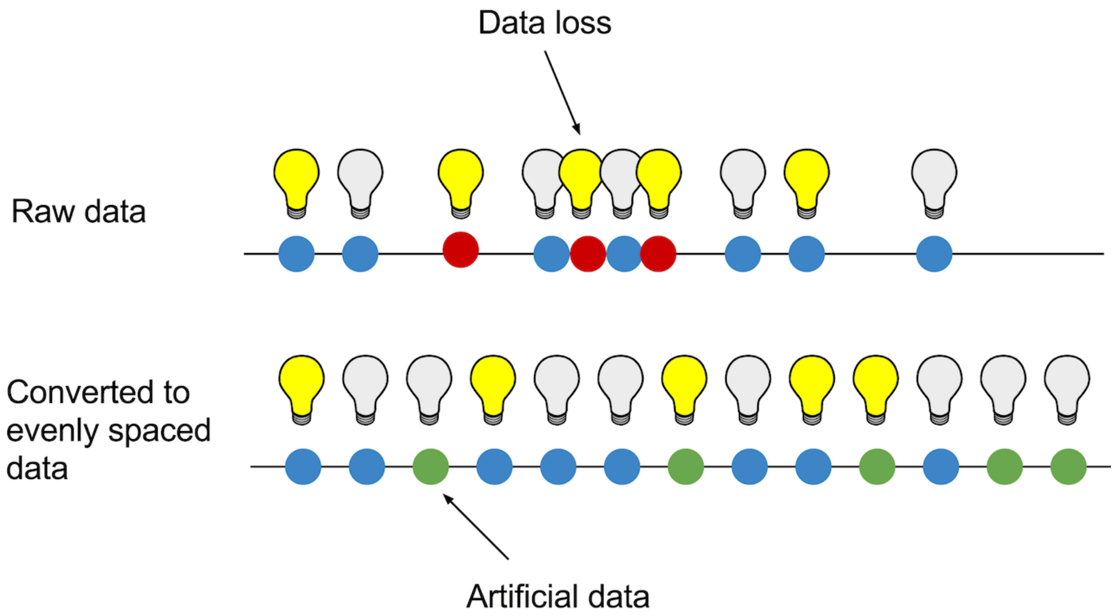


FIGURE 4.16: downsides of imputation

Data on time. The time between the measurements may contain valuable data information. For example, we can confidently assume that the second house is more likely to use an automatic light switch than the first house, based on the frequency and length of the light switching on or off.

Figure downsides of imputation 2[Figure B.9]

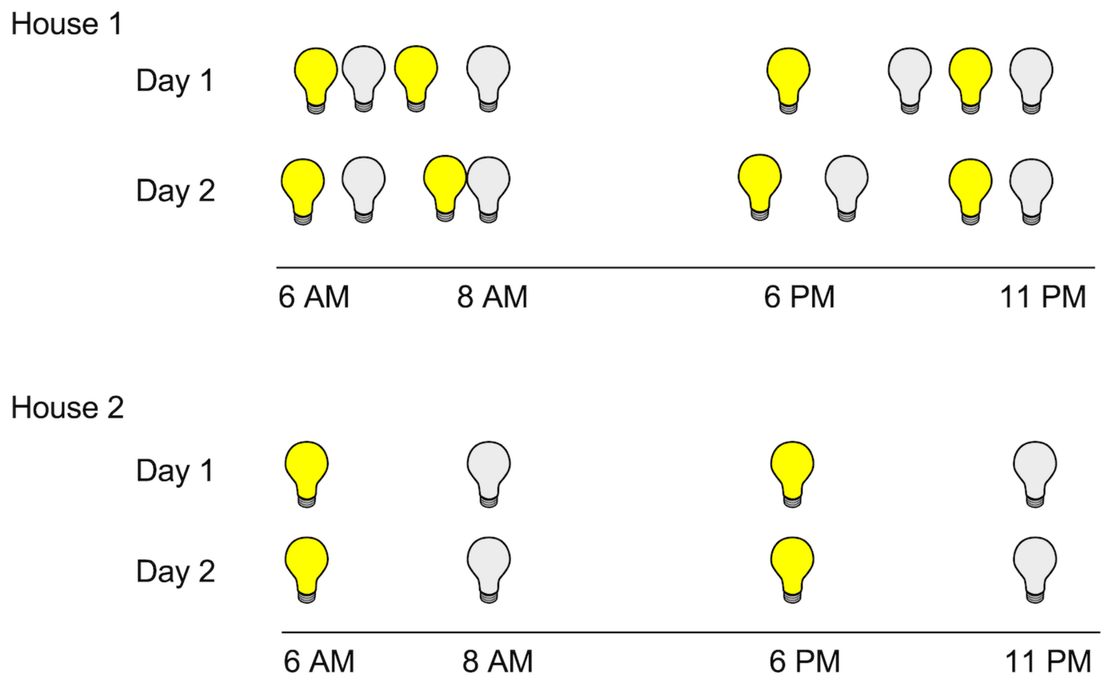


FIGURE 4.17: downsides of imputation 2

4.5.2 Lack of sufficient data and features

A typical time series is a measurement of a variable indexed in time order. And the time-series we have extracted from the data given to us is a really crude version of an actual time series as it contains just the timestamps of when failure has occurred in a specific component of a specific train-set.Also, the maximum number of rows or data-points one train-set-component combinations have is around 245 and less. This makes it really hard to approximate the the data we have with a clear and concise general model which will be capable of predicting future flaws. Hence, only the bare-metal wire-frame of the time-series analysis implementation is documented and further work is obviously necessary. This will be done after the collection of the complete data for lines 5 to 9 of Delhi metro which is still being collected as per official statement.

Chapter 5

RESULTS, DISCUSSIONS AND CONCLUSIONS

The automated system of the data capturing in the railway transportation will help in analyzing the evolution of fault trends and predicting the failure. The proposed system comprises of four phases namely,preprocessing data, sequential association rule mining , training of time-series and prediction using ARIMA model.

5.1 Results & Analysis

5.1.1 SPADE results

We have successfully mined for temporal sequences in our data and now know in a temporal order which faults in components are imminent to appear after which.This helps us find the root cause of some component of the train failing and if we keep that root cause in check, all resultant failures will not occur.

5.1.2 ARIMA

In failure prediction using ARIMA we hit a brick wall at a certain point because of drawbacks that has already been mentioned. These drawbacks will be overcome in time when appropriate data is on our hands. A boilerplate code through which the new data can be fed to get apt results has been developed though and given to seniors for usage when they get their hands on the data that is now being collected.

5.2 Cost Estimation Model

Estimation of all relevant costs cannot be revealed as this is a corporate handled project and some of its parts fall under non disclosure agreement. Although, speculation wise, costs incurred after the full-fledged model is deployed will contain:

1. Cloud Storage and compute to keep the algorithm running at all times
2. Data collection costs incurred for surveyors

5.3 Conclusions

I was able to mine seamlessly for frequent patterns in failure of components that occured in temporal order using Zaki's algorithm , SPADE. In training phase, the time series data of past events are taken into consideration and train the events for extracting features using ARIMA model. The predicted features are considered as the raw features. Since the implementation of the time-series prediction model for faults fell short of necessary data , features and time there has been quite a few shortcomings with regards to time-series analysis.

But since all of the code-base has been shared with managers and guides , as soon as the complete failure data corresponding to every line is collected, this data will be applied to my code and better results of timeseries analysis and prediction will be attained.

5.4 Scope for Future Work

5.4.1 Kalman Filter usage

The easiest way to deal with an irregularly spaced time series with relatively regular "small" gaps is to view it as a regularly spaced time series with missing data. Here, since your smallest gap is 1 day, you can consider it as daily data but with some days missing:

The situation is a little bit different if you have a very large variance in the size of the gaps, for example if you had millisecond-level time stamps but sometimes go a whole year without any observation; in that case it can be handled more efficiently in another way (e.g. by having time-varying matrices in the state space model used by the Kalman filter).

The Kalman filter will allow you to fit an ARIMA model with missing values by computing the likelihood which you can then optimize over the parameters. You can then use that model to forecast. If you need, you can also use the Kalman filter or smoother to get the distribution of the missing values conditional on your data (only past data for the filter, or including future data for the smoother) and parameters.

But you do not need to impute these values first, and doing this is not a preliminary step to an analysis (it is the analysis, you have already picked an ARIMA model at this point).

As for the repeated measures, if it makes sense for the domain you can sum or average those values on a given day. If it doesn't and you have no way to differentiate those records in a given day, you can set up a state space model where the state is, for example, given by:

$$Xt = Xt1 + t \tag{5.1}$$

And the observation equation is:

$$Y(i)t = Xt + (i)t, i = 1, ..., nt \tag{5.2}$$

This would be an ARIMA(1,0,0) model with repeated measures of varying sample sizes depending on the day. The Kalman filter can accommodate state space models with varying observation dimension.

5.4.2 Mine for derived features

Since we fell short on target features to perform our time-series analysis and prediction after seclusion of train-set component combinations we can use a library like featuretools or use intuition to derive one more time dependant feature to perform prediction upon.

Bibliography

[1] Sai Swaroop Ratakonda and Sreela Sasi, *"Seasonal Trend Analysis on Multi-Variate Time Series Data"*, 2018 International Conference on Data Science and Engineering (ICDSE), (2018).

[2] Wissam Sammouri, Etienne Côme, Latifa Oukhellou, Patrice Aknin, and Charles-Eric Fonlladosa, *"Pattern recognition approach for the prediction of infrequent target events in floating train data sequences within a predictive maintenance framework"*., 7th International IEEE Conference on Intelligent Transportation Systems (ITSC), (2014), 918-923.

[3] R B Faiz and S. Singh, *"Time Based Information Analysis of UK Rail Track"*.In 2009 International Conference on Computing, Engineering and Information, (2009), 200-209.

[4] R. Cox and J. S. Turner, "Project Zeus: design of a broadband network and its application on a university campus," Washington Univ., Dept. of Comp. Sci., Technical Report WUCS-91-45, July 30, 1991.

[5] Fuminao Okumura and Hajime Takagi, "Maglev Guideway On the Yamanashi Test Line," *http://www.rtri.or.jp/rd/maglev2/okumura.html,* October 24, 1998.

[6] "AT&T Supplies First CDMA Cellular System in Indonesia," *http://www.att.com/press/1095/951011.nsa.html*, Feb 5, 1996.

Appendix A

Appendix A : Code for various phases of excecution

The code snippets for various phases for project are mentioned here.

- Phase 1: Fault data analysis and finding frequent failures
- Phase 2: Spade for temporal frequent itemset mining
- Phase 3: ARIMA forecasting

A.1 Appendix A Phase 1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import re
import sys
import datetime
data = pd.read_csv('processed3.csv',low_memory=False)
data['Fdate']= pd.to_datetime(data['Fdate'])
data['Fdate'] = data['Fdate'].apply(lambda x: x.date())
data.info()
data
df2 = data.copy()
cols = df2.columns.tolist()
print(cols)
pos =0
print("{:<20}".format('Coulmn Index'),'\tCoulmn Name')
for i in cols:
    print("{:^20}".format(pos),'\t', i)
    pos +=1
p =int(input("chose target index for analysis:- "))
group1 = df2.groupby(cols[p])
print(group1)
for field in cols:
    ((group1.apply(lambda x: x[field].unique())).apply(pd.Series)).to_csv('TCTNdatex.↩
    csv')
    print("File created for",field)
df2.info()
name1 =df2['TCTN'].value_counts()
name1.to_csv('TCTNdatecount.csv',header='TCTN')
df3=pd.read_csv('TCTNdatex.csv',low_memory=False)
df3=df3.dropna(thresh=5)
df3.to_csv('TCTNatexy.csv',date_format = '%Y%m%d')
df3.head()
df3=df3.set_index('TCTN').apply(pd.to_datetime).diff(-1, axis=1)
df3=df3.abs()
df3['STDEV']=df3.std(axis=1)
df3 = df3.iloc[:, 1:].apply(pd.to_timedelta)
print (df3)
df3.to_csv('timedelta.csv')
df3 = df3.apply(lambda x: x.dt.days)
print (df3)
```

A.2 Appendix A Phase 2

```
import pandas as pd
import datetime
r = pd.read_csv('RESULTANT.csv',low_memory=False)
r=r.dropna()
r.dtypes
r['date'] = pd.to_datetime(r['date'])
r=r.drop(['Unnamed: 0'], axis = 1)
 try:
        for i in r.index[0:]:
            if  (r.at[i+1, 'TC'] == r.at[i, 'TC']):
                if (r.at[i+1, 'lbl'] == r.at[i, 'lbl']) & (r.at[i+1, 'lbl'] == 'f'):
                    r.at[i+1, 'datedelta'] = r.at[i+1, 'date'] - r.at[i, 'date']
                elif r.at[i+1, 'lbl'] == 'f':
                    r.at[i+1, 'datedelta'] = r.at[i+1, 'date'] - r.at[i, 'date']
except KeyError:
        print("last key parsed")
r.to_csv('intermediate.csv')
r['month']=pd.DatetimeIndex(r['date']).month
n=r.copy()
g=r.groupby('TC')['month'].nunique().hist()   #train components per month
r.groupby('month')['datedelta'].nunique().hist()   #datedeltas components per month
n=n.sort_values(['month'])
n.groupby('TC')['month'].nunique().reset_index()   #number of months in which each ↩
    TRAINLINE+COMPONENT APPEARS
```

A.3 Appendix A Phase 3

```
min_support=7000
inputfile = csv.reader(open('spade.csv','r'))
outputfile = open('dataset1.txt','w')
event_id=[[] for _ in range(9000)]
d=dict()
seq_event=dict()
i=0
for row in inputfile:
    if(i>0):
        out_str='I'
        customer=row[0]
        customer=customer.split('TNST')
        customer=customer[1]
        customer=int(customer)-100
        date=row[1]
        date=date.split('-')
        date=date[2]
        item=row[2]
        item=int(int(item)/100)
        out_str+=str(item)
        seq_event[out_str]=[]
        d[str(customer)+"-"+str(date)]=[]
        if date not in event_id[customer]:
            print(out_str,event_id[customer])
            event_id[customer].append(date)
        outputfile.write(str(customer)+" "+date+" "+out_str+"\n")
        event_id[customer].sort()
    i+=1

i=0
inputfile = csv.reader(open('spade.csv','r'))
for row in inputfile:
    if(i>0):
        out_str='I'
        customer=row[0]
        customer=customer.split('TNST')
        customer=customer[1]
        customer=int(customer)-100
        date=row[1]
        date=date.split('-')
        date=date[2]
        item=row[2]
        item=int(int(item)/100)
        out_str+=str(item)
        if out_str not in d[str(customer)+"-"+str(date)]:
            d[str(customer)+"-"+str(date)].append(out_str)
        d[str(customer)+"-"+str(date)].sort()
    i+=1

i=0
outputfile = open('data3.txt','w')
```

```
for sequences in d:
    seq=sequences.split('-')
    a=seq[0]
    b=seq[1]
    if i<1000:
        outputfile.write(a+" "+b+" "+str(len(d[sequences])))
    for items in d[sequences]:
        items=items.split('I')
        if i<1000:
            outputfile.write(" "+items[1])
    if i<1000:
        outputfile.write("\n")
    i+=1

from pycspade.helpers import spade, print_result

result = spade(filename='data3.txt', support=0.5, parse=True)
print_result(result)
r=pd.DataFrame(result)
```

Appendix B

Appendix B Graphs and Visualizations

All graphs and visualizations for phases of project are mentioned here.

- Phase 1: Fault data analysis and finding frequent failures
- Phase 2: Spade for temporal frequent itemset mining
- Phase 3: ARIMA forecasting

B.1 Appendix B Section 1

```
[17]: df3.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19771 entries, 0 to 19770
Columns: 228 entries, TCTN to 226
dtypes: object(228)
memory usage: 34.4+ MB

[18]: df3=df3.dropna(thresh=5)

[19]: df3.to_csv('TCTNatexy.csv',date_format = '%Y%m%d')

[20]: df3.head()
```

[20]:

	TCTN	0	1	2	3	4	5	6	7	8	...	217	218	219	220	221	222	223	224	225	226
0	TNST101 TC101	2016-11-14	2014-07-13	2016-03-30	2013-11-15	2013-11-16	2015-03-05	2015-05-05	2014-12-07	2017-07-02	...	2013-01-06	2014-09-14	2013-01-24	2018-05-05	2018-06-15	2013-05-31	2014-09-17	2018-01-26	NaN	NaN
2	TNST101 TC102	2016-11-10	2014-04-09	2015-08-07	2016-02-05	2015-04-21	2019-01-17	2014-02-27	2013-03-14	2012-06-07	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	TNST101 TC103	2016-10-09	2013-05-16	2018-10-22	2017-04-15	2015-07-29	2015-07-21	2012-04-12	2014-08-24	2016-05-09	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	TNST101 TC104	2018-10-14	2017-11-03	2012-09-26	2013-09-22	2013-11-02	2013-12-26	2019-04-15	2013-12-31	2013-11-03	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	TNST101 TC105	2019-07-19	2014-10-31	2012-10-18	2019-09-21	2019-09-26	2015-10-31	2019-10-10	2013-10-30	2015-10-30	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 228 columns

FIGURE B.1: failure dates by trainset and component

```
                1     2      3      4     5     6    7    8      9    10  \
TCTN
TNST101 TC101   626   866      1    474    61   149  938  354    616  1456
TNST101 TC102   485   182    290   1367  1785   350  280  451  90802     0
TNST101 TC103  1985   555    626      8  1195   864  624   21   1032  1069
TNST101 TC104  1864   361     41     54  1936  1931   58  464     63  1112
TNST101 TC105   743  2529      5   1426  1440  2171  730  355    574  1222
...             ...   ...    ...    ...   ...   ...  ...  ...    ...   ...
TNST344 TC148   159    98  88663      0     0     0    0    0      0     0
TNST344 TC198   153    66    310  88845     0     0    0    0      0     0
TNST344 TC284    90   307  88812      0     0     0    0    0      0     0
TNST344 TC293   153   108  88586      0     0     0    0    0      0     0
TNST345 TC148     8    41  88598      0     0     0    0    0      0     0

               ...  218   219  220   221  222   223    224  225  226  STDEV
TCTN           ...
TNST101 TC101  ...  598  1927   41  1841  474  1227  89194    0  NaN   5911
TNST101 TC102  ...    0     0    0     0    0     0      0    0  NaN   6040
TNST101 TC103  ...    0     0    0     0    0     0      0    0  NaN   5974
TNST101 TC104  ...    0     0    0     0    0     0      0    0  NaN   6062
TNST101 TC105  ...    0     0    0     0    0     0      0    0  NaN   6010
...            ...  ...   ...  ...   ...  ...   ...    ...  ...  ...    ...
TNST344 TC148  ...    0     0    0     0    0     0      0    0  NaN   5897
TNST344 TC198  ...    0     0    0     0    0     0      0    0  NaN   5909
TNST344 TC284  ...    0     0    0     0    0     0      0    0  NaN   5907
TNST344 TC293  ...    0     0    0     0    0     0      0    0  NaN   5892
TNST345 TC148  ...    0     0    0     0    0     0      0    0  NaN   5893

[4997 rows x 227 columns]
```

FIGURE B.2: difference between successive failure dates

```
g=r.groupby('TC')['month'].nunique().hist()    #train components per month
```

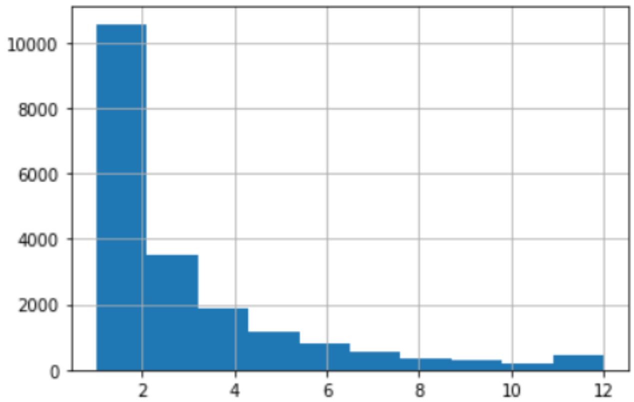


FIGURE B.3: Failures in components per month

```
r.groupby('month')['datedelta'].nunique().hist()    #datedeltas components per month
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22c7c29a90>
```

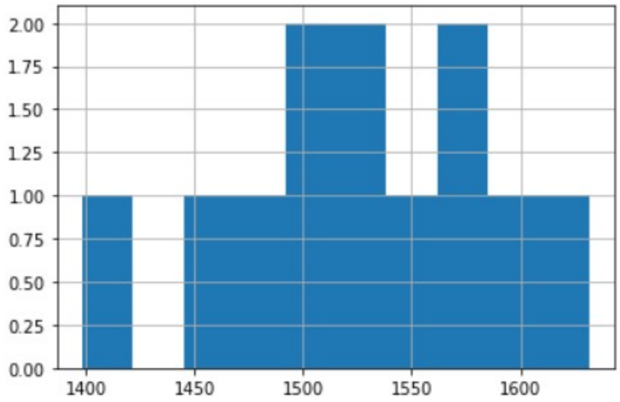
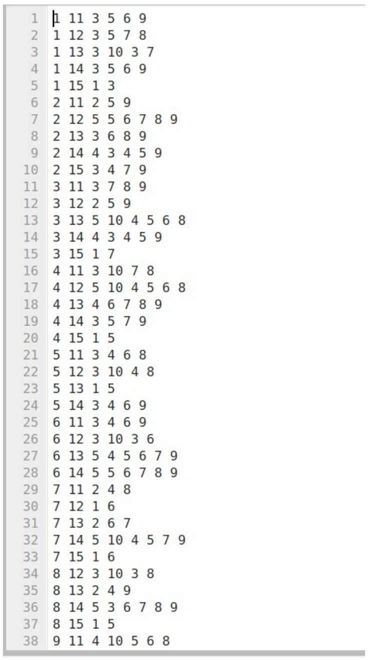


FIGURE B.4: difference in failure dates per month

B.2 Appendix B Section 2

	customer_id	trans_date	tran_amount
1	CS1112	15-Jun-11	56
2	CS1112	19-Aug-11	96
3	CS1112	2-Oct-11	60
4	CS1112	8-Apr-12	56
5	CS1112	24-Jun-12	52
6	CS1112	3-Jul-12	81
7	CS1112	16-Sep-12	72
8	CS1112	15-Dec-12	76
9	CS1112	1-Mar-13	105
10	CS1112	1-Jul-13	36
11	CS1112	13-Nov-13	71
12	CS1112	29-Apr-14	63
13	CS1112	16-Jul-14	90
14	CS1112	4-Dec-14	59
15	CS1112	14-Jan-15	39
16	CS1113	27-May-11	94
17	CS1113	25-Jul-11	57
18	CS1113	23-Oct-11	93
19	CS1113	30-Mar-12	86
20	CS1113	5-Sep-12	67
21	CS1113	8-Oct-12	95
22	CS1113	6-Nov-12	51
23	CS1113	7-Dec-12	75
24	CS1113	6-Mar-13	97

FIGURE B.5: Input to SPADE preprocessor



```
1 11 3 5 6 9
1 12 3 5 7 8
1 13 3 10 3 7
1 14 3 5 6 9
1 15 1 3
2 11 2 5 9
2 12 5 5 6 7 8 9
2 13 3 6 8 9
2 14 4 3 4 5 9
2 15 3 4 7 9
3 11 3 7 8 9
3 12 2 5 9
3 13 5 10 4 5 6 8
3 14 4 3 4 5 9
3 15 1 7
4 11 3 10 7 8
4 12 5 10 4 5 6 8
4 13 4 6 7 8 9
4 14 3 5 7 9
4 15 1 5
5 11 3 4 6 8
5 12 3 10 4 8
5 13 1 5
5 14 3 4 6 9
6 11 3 4 6 9
6 12 3 10 3 6
6 13 5 4 5 6 7 9
6 14 5 5 6 7 8 9
7 11 2 4 8
7 12 1 6
7 13 2 6 7
7 14 5 10 4 5 7 9
7 15 1 6
8 12 3 10 3 8
8 13 2 4 9
8 14 5 3 6 7 8 9
8 15 1 5
9 11 4 10 5 6 8
```

FIGURE B.6: Input to PYCSPADE

B.3 Appendix B Section 3

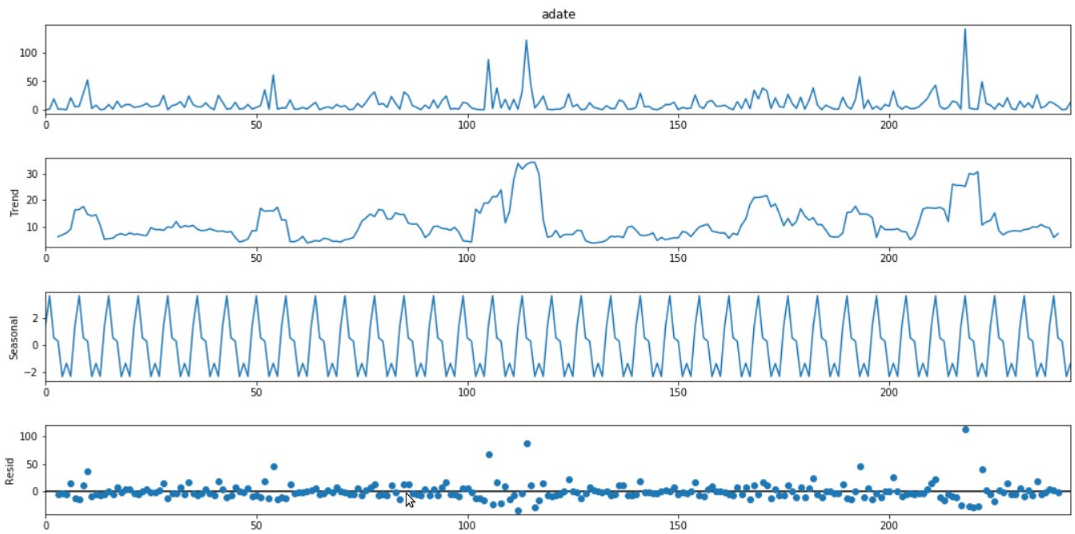


FIGURE B.7: Seasonal Decompose of TNST101TC101

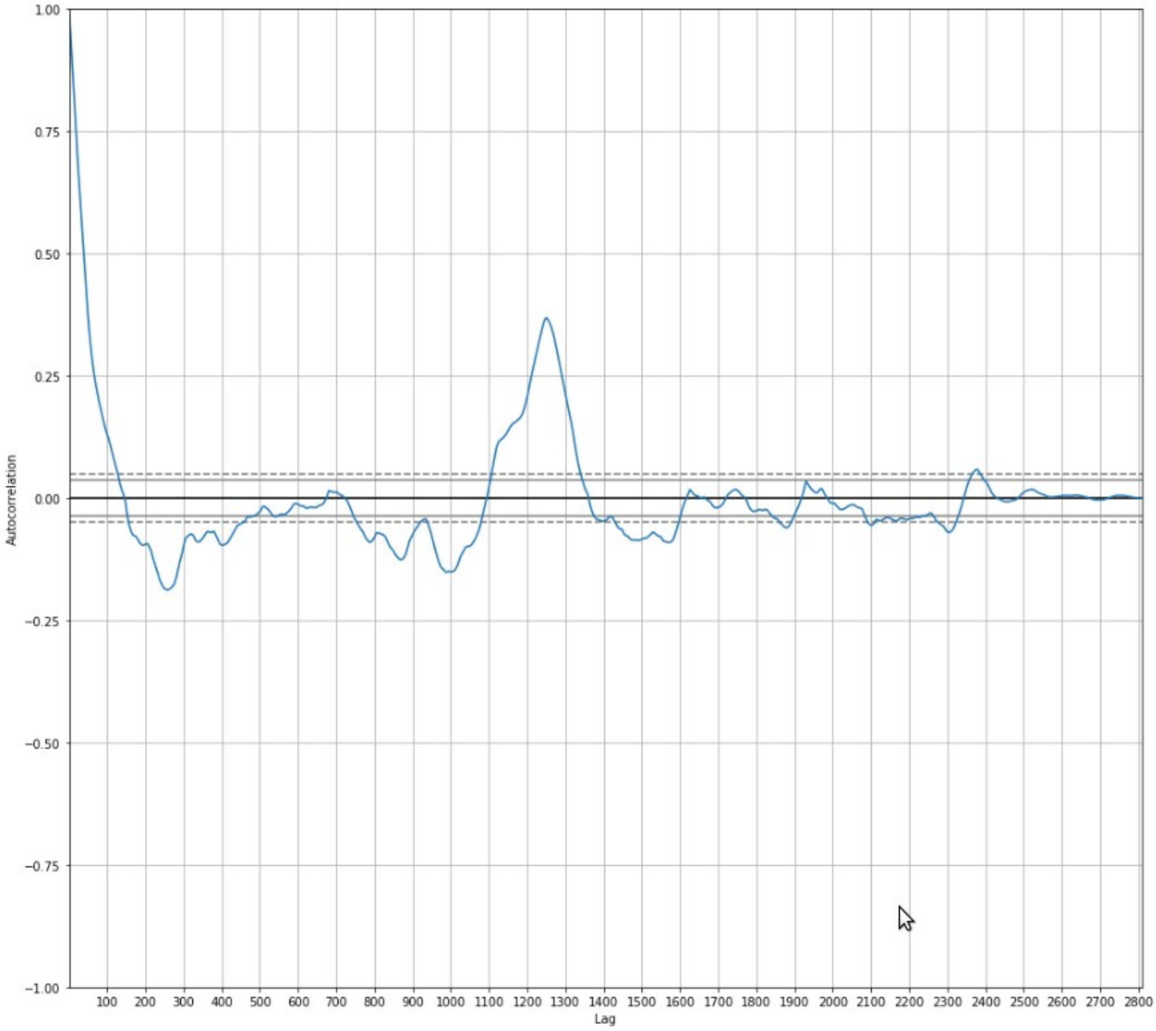


FIGURE B.8: ACF of TNST101TC101

```
# plot
plt.rcParams['figure.figsize'] = [18, 5]
pyplot.plot(predictions, color='red')
pyplot.plot(test)
```

```
[<matplotlib.lines.Line2D at 0x7f1218b47e50>]
```

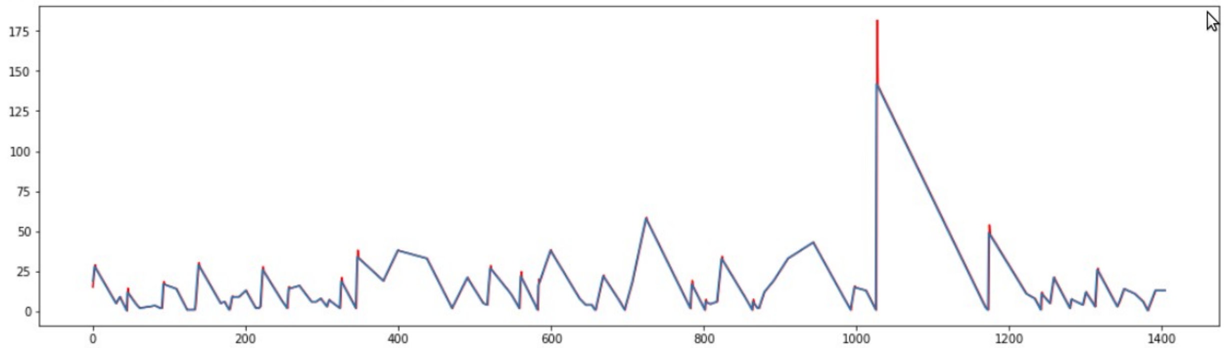


FIGURE B.9: Actual Data chart and predicted chart