

Predicting Satellite Coordinates Using Machine Learning Models

by ISMAIL ESACK DAWOODJEE .

Submission date: 01-Apr-2020 03:42PM (UTC+0800)

Submission ID: 1286841668

File name:

32516_ISMAIL_ESACK_DAWOODJEE__Predicting_Satellite_Coordinates_Using_Machine_Learning_Models_15843_194531242.pdf
(1.06M)

Word count: 10958

Character count: 54154



ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT046-3-M-AML

APPLIED MACHINE LEARNING

APUMF1908DSBA

HAND OUT DATE: 19 MARCH 2020

HAND IN DATE: 31 MARCH 2020

WEIGHTAGE: 60%

Predicting Satellite Coordinates Using Machine Learning Models

NAME: ISMAIL ESACK DAWOODJEE

INSTITUTION: Asia Pacific University of Technology and Innovation

EMAIL: tp054033@mail.apu.edu.my

TP NUMBER: TP054033

LECTURER NAME: Prof. Dr. Mandava Rajeswari

Contents

| | |
|------------------------------------|----|
| 1. Methodology | 2 |
| 2. Metrics and Models | 4 |
| 3. Choosing a Satellite to Study | 5 |
| 4. Data Preparation | 7 |
| 5a. Data Exploration Part 1 | 10 |
| 5b. Data Exploration Part 2 | 17 |
| 6. Data Preprocessing | 22 |
| 7. Non-Linear Regression Model | 25 |
| 8. Support Vector Regression Model | 28 |
| 9. Random Forest Regression | 32 |
| 10. Analysis and Recommendations | 39 |
| 11. Conclusion | 42 |
| Acknowledgements | 42 |
| References | 43 |

1. Methodology

The two datasets (in CSV format) used in this study were provided to participants of the 2020 International Data Analytics Olympiad by the Russian Astronomical Science Centre. The first dataset is the Train dataset, which contains 649913 observations and 15 variables. The second is the Test dataset, which contains 284072 observations and 9 variables. The Train dataset consists of real and simulated coordinates (measured in *km*) and velocities (measured in *km/s*) of 600 satellites in the month of January while the Test dataset consists of only simulated coordinates and velocities of 300 satellites in February 2014. The variable names and descriptions for both datasets are summarized in Table 1 below. Moreover, the first five rows of both Train and Test datasets are displayed in Figure 1.

Table 1: Variable names and descriptions

| Variable Name | Variable Description |
|---|--|
| <i>id</i> | integer id of the measurement (present in both datasets) |
| <i>epoch</i> | datetime in “%Y-%m-%dT%H:%M:%S.%f” format (e.g. 2014-01-23T17:48:23.349) (present in both datasets) |
| <i>sat_id</i> | integer id of the satellite (present in both datasets) |
| $\{x, y, z, V_x, V_y, V_z\}$ | measurements of real coordinates and velocities of the satellite (target variables; not present in the Test dataset) |
| $\{x_{sim}, y_{sim}, z_{sim}, V_{xsim}, V_{ysim}, V_{zsim}\}$ | measurements of simulated coordinates and velocities of the satellite (present in both datasets) |

| train.head() | | | | | | | | | | | | | | | |
|--------------|-----------|-------------------------|---------------|---------------|---------------|---------------|-----------|-----------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|
| | id | epoch | sat_id | x | y | z | Vx | Vy | Vz | x_sim | y_sim | z_sim | Vx_sim | Vy_sim | Vz_sim |
| 0 | 0 | 2014-01-01 00:00:00.000 | 0 | -8856.823863 | 13117.780146 | -20728.353233 | -0.908303 | -3.808436 | -2.022083 | -8843.131454 | 13138.221690 | -20741.615306 | -0.907527 | -3.804930 | -2.024133 |
| 1 | 1 | 2014-01-01 00:49:43.000 | 0 | -10567.672384 | 1619.746066 | -24451.813271 | -0.302590 | -4.272617 | -0.612796 | -10555.500066 | 1649.289367 | -24473.089556 | -0.303704 | -4.269816 | -0.616468 |
| 2 | 2 | 2014-01-01 01:33:29.001 | 0 | -10578.684043 | -10180.467460 | -24238.280849 | 0.277435 | -4.047522 | 0.723155 | -10571.858472 | -10145.939908 | -24271.169776 | 0.274880 | -4.046788 | 0.718768 |
| 3 | 3 | 2014-01-01 02:20:09.001 | 0 | -9140.251857 | -20651.437460 | -20720.381279 | 0.715600 | -3.373762 | 1.722115 | -9149.620794 | -20618.200201 | -20765.019094 | 0.712437 | -3.375202 | 1.718306 |
| 4 | 4 | 2014-01-01 03:09:52.002 | 0 | -6719.092336 | -28929.061629 | -14938.907957 | 0.992507 | -2.519732 | 2.344703 | -6729.358857 | -28902.271436 | -14992.399986 | 0.989382 | -2.522618 | 2.342237 |

| test.head() | | | | | | | | | | | | | | |
|-------------|-----------|---------------|-------------------------|---------------|---------------|--------------|---------------|---------------|---------------|--|--|--|--|--|
| | id | sat_id | epoch | x_sim | y_sim | z_sim | Vx_sim | Vy_sim | Vz_sim | | | | | |
| 0 | 3927 | 1 | 2014-02-01 00:01:45.162 | -13366.891347 | -14236.753503 | 6386.774555 | 4.333815 | -0.692764 | 0.810774 | | | | | |
| 1 | 3928 | 1 | 2014-02-01 00:22:57.007 | -7370.434039 | -14498.771520 | 7130.411325 | 5.077413 | 0.360959 | 0.313402 | | | | | |
| 2 | 3929 | 1 | 2014-02-01 00:44:08.862 | -572.098654 | -13065.289498 | 7033.794876 | 5.519106 | 2.012830 | -0.539412 | | | | | |
| 3 | 3930 | 1 | 2014-02-01 01:05:20.697 | 6208.945257 | -9076.852425 | 5548.296900 | 4.849212 | 4.338955 | -1.869600 | | | | | |
| 4 | 3931 | 1 | 2014-02-01 01:26:32.542 | 10768.200284 | -2199.706707 | 2272.014862 | 1.940505 | 6.192887 | -3.167724 | | | | | |

Figure 1: The first 5 rows of Train and Test datasets. It should be noted that only 300 out of 600 satellites from Train are present in Test. For instance, Test does not contain Satellite 0.

The *epoch* variable, as used in the astronomical literature, refers to the moment in time that the elements of a space object, such as its coordinates and velocities, are specified. This should not be confused with the number of epochs of training done for an ML model. The set of *simulated* coordinates and velocities were obtained using the less accurate SGP4 physics-based simulation model while the set of *real* coordinates and velocities were obtained using a more accurate (but unspecified) simulation model. To illustrate this

information using an example satellite, the x - and x_{sim} -coordinates of Satellite 1 from both Train and Test datasets are plotted in Figure 2 below.



Figure 2: Plots of true and simulated x-coordinates for Satellite 1

The green plot displays the real x -coordinates of Satellite 1 in January 2014, obtained from the Train dataset. The blue and orange plots are the simulated x_{sim} -coordinates in January 2014 (obtained from the Train dataset) and February 2014 (obtained from the Test dataset), respectively.

Several findings can be deduced by inspection from Figure 2 and from plots of other kinematic states as well (the set of x, y, z, V_x, V_y, V_z can be collectively called the *kinematic states*). Firstly, satellite kinematics display a “seasonal” pattern, which makes sense from a physical point of view, because a full orbit around the Earth occurs over a fixed period of time. Secondly, there are no cyclical patterns displayed by the data. To distinguish between cyclical and seasonal patterns, an extract from the book *Forecasting: Principles and Practice* says “If the fluctuations are not of a fixed frequency then they are cyclic; if the frequency is unchanging and associated with some aspect of the calendar, then the pattern is seasonal” (Hyndman and Athanasopoulos, 2018, p.31). Since the fluctuations occur over a fixed orbital period, the pattern should be seasonal. Finally, the simulated coordinates x_{sim} were seen to be initially accurate in predicting true/real coordinates but becomes increasingly inaccurate at future epochs. These findings suggest applying classical forecasting techniques, namely Seasonal Holt-Winters and Seasonal ARIMA methods. However, the focus of this study is on developing machine learning models, so applying classical forecasting methods can be done as an extension to this paper.

Based on the plots and the above information, three approaches can be taken:

1. Ignore the simulated states (coordinates and velocities) and directly predict real states for February using training data of real states and epochs in January. This is equivalent to simply forecasting/extrapolating the green curve into the future.

Predictor variables $\{epoch, x\} \rightarrow$ ML model \rightarrow Output $\{\hat{x}\}$

2. Use both real and simulated states in January as training variables to fit the ML model, which can then predict the real states for February.

Predictor variables $\{epoch, x, x_{sim}\} \rightarrow$ ML model \rightarrow Output $\{\hat{x}\}$

3. Prepare the data so that the true errors (e.g. $e_x = x_{sim} - x$) plus the simulated states in January become predictor variables for the ML model, which can then produce the output ML-modified errors (e.g. \hat{e}_x) for February. From the ML-modified errors, the real states in February can be retrieved via $\hat{x} = x_{sim} - \hat{e}_x$. This is similar to the approach used by Peng and Bai (2019).

Predictor variables $\{epoch, e_x, x_{sim}\} \rightarrow$ ML model \rightarrow Output $\{\hat{e}_x\} \rightarrow \{\hat{x}\}$

When combined with the three ML models considered, there are a total of 9 unique ways of tackling the problem of predicting satellite trajectories. Once again, not all combinations will be explored in this paper, investigating only 3 combinations and leaving 6 for further research. In particular, Approach 1 will be used for fitting a Non-Linear Regression model, while Approach 3 will be used for Support Vector Regression and Random Forest Regression models. Furthermore, only the x -coordinate will be modelled since the ML models can be readily extended to include the other kinematics states.

2. Metrics and Models

The three metrics that were chosen to evaluate the ML models are RMSE (Root Mean Square Error), SMAPE (Symmetric Mean Absolute Percentage Error) and MAPRE (Mean Absolute Percentage Residual Error). The RMSE is a commonly used metric in regression models, defined by the equation below:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{x}_i - x_i)^2},$$

where \hat{x}_i is the predicted value, x_i is the actual value and n is the number of observations in the dataset.

The SMAPE metric is less well-known, but it is specialized for use in forecasting applications. It is defined by the equation below:

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^n \frac{|\hat{x}_i - x_i|}{|\hat{x}_i| + |x_i|},$$

where the terms in the equation are the same as those in RMSE.

The MAPRE is a metric introduced by Peng and Bai (2018), in which they used the concept of MAPE (Mean Absolute Percentage Error) to define the MAPRE as follows:

$$MAPRE = 100\% \frac{\sum_{i=1}^n |e_x - \hat{e}_x|}{\sum_{i=1}^n |e_x|},$$

where e_x is the true error in the x -coordinate and \hat{e}_x is the ML-predicted error. The authors claimed that the learning performance of the ML model is directly quantified by this metric.

The three machine learning models that will be studied in this paper are Non-Linear Regression, Support Vector Regression and Random Forest Regression.

Non-Linear Regression - When faced with a periodic pattern in the data, an intuitive mathematical function for fitting it would be the Fourier series. It is known from mathematics that any periodic pattern that is continuous (need not be differential) can be approximated by a Fourier series. However, based on the

periodic pattern observed from Figure 2 in the previous section, a simple sinusoidal curve would be good enough to model the data. Also, due to the slight inclination seen from the figure, a combination of the sine function together with an added linear trend was considered for the regression model. Modelling the x -coordinate then turns out as follows:

$$x(t) = A \sin\left(\frac{2\pi}{T}t + \phi\right) + mt + c,$$

where A is the amplitude, T is the period, ϕ is the phase shift, m and c are the slope and intercept of the linear trend, t is the independent time variable and $x(t)$ is the time-dependent x -coordinate. For this model, the built-in `nls` or non-linear least squares function in R will be used to fit the above equation.

Support Vector Regression (SVR) - Support vector machines are a robust class of machine learning models that have been preferred by researchers for many years until the advent of artificial neural networks. Part of what makes them robust is their random sampling during cross-validation of the dataset, making the model more generalizable and applicable to new observations. SVMs are usually used for classification but they can also be extended for regression problems, such as the one considered in this study. For regression, the parameters of the SVR are the `kernels` (can be radial/Gaussian, linear, polynomial, or sigmoid), the maximum margin of error `epsilon` (the higher the epsilon, the higher the model's tolerance to errors), and the cost or regularization parameter `C` (the higher the C, the fewer the amount of margin violations above `epsilon`). These parameters should be tuned to obtain the most optimal model possible. The standard `svm` function from the machine learning library `e1071` will be used to implement the SVR model.

Random Forest Regression (RFR) - Similar to SVMs, random forests and decision trees are used mainly for classification tasks but can be extended to include regression as well. Random forests use bootstrap aggregation of many decision trees trained on randomly sampled data to obtain a model with lower variance (compared to one decision tree trained on the entire dataset). The parameters that will be tuned for the RFR are: the number of trees to be grown (`ntree`), the number of variables to split at each branch of a tree (`mtry`), and the maximum number of terminal nodes (or leaves) a tree can have (`maxnodes`). The `randomForest` function from the `randomForest` library will be used to implement the RFR model.

3. Choosing a Satellite to Study

Out of the 300 satellites in the Train dataset, one satellite was chosen for implementing the 3 models outlined in the previous section. This satellite was chosen on the basis of having the largest number of observations, so as to provide the most training data for the models.

Read in the original datasets. This may take some time because the two files are very large. There are no external libraries required for this section.

```
train <- read.csv("train.csv")
test  <- read.csv("test.csv")
```

Check the dimensions of the datasets.

```
dim(train)

## [1] 649912      15

dim(test)

## [1] 284071      9
```

The Train dataset has 649912 observations and 15 variables while the Test dataset has 284071 observations and 9 variables.

Check the variable names of the datasets.

```
names(train)

## [1] "id"      "epoch"   "sat_id"  "x"       "y"       "z"       "Vx"      "Vy"
## [9] "Vz"      "x_sim"   "y_sim"   "z_sim"   "Vx_sim"  "Vy_sim"  "Vz_sim"

names(test)

## [1] "id"      "sat_id"  "epoch"   "x_sim"   "y_sim"   "z_sim"   "Vx_sim"  "Vy_sim"
## [9] "Vz_sim"
```

The six excluded variables are the true coordinates and velocities, not present in the Test dataset.

Check the structure of the datasets.

```
str(train)

## 'data.frame': 649912 obs. of 15 variables:
## $ id    : int 0 1 2 3 4 5 6 7 8 9 ...
## $ epoch : Factor w/ 649222 levels "2014-01-01T00:00:00.000",...: 1 402 1051 1742 2408 3106 3775 4455
## $ sat_id: int 0 0 0 0 0 0 0 0 0 ...
## $ x     : num -8856 -10568 -10579 -9148 -6719 ...
## $ y     : num 13118 1620 -10180 -20651 -28929 ...
## $ z     : num -20728 -24452 -24238 -20720 -14939 ...
## $ Vx    : num -0.908 -0.303 0.277 0.716 0.993 ...
## $ Vy    : num -3.81 -4.27 -4.05 -3.37 -2.52 ...
## $ Vz    : num -2.022 -0.613 0.723 1.722 2.345 ...
## $ x_sim : num -8843 -10556 -10572 -9150 -6729 ...
## $ y_sim : num 13138 1649 -10146 -20618 -28902 ...
## $ z_sim : num -20742 -24473 -24271 -20765 -14992 ...
## $ Vx_sim: num -0.908 -0.304 0.275 0.712 0.989 ...
## $ Vy_sim: num -3.8 -4.27 -4.05 -3.38 -2.52 ...
## $ Vz_sim: num -2.024 -0.616 0.719 1.718 2.342 ...

str(test)

## 'data.frame': 284071 obs. of 9 variables:
## $ id    : int 3927 3928 3929 3930 3931 3932 3933 3934 3935 3936 ...
## $ sat_id: int 1 1 1 1 1 1 1 1 1 ...
## $ epoch : Factor w/ 284047 levels "2014-02-01T00:00:01.882",...: 14 164 310 463 606 761 906 1051 119
## $ x_sim : num -13367 -7370 -572 6209 10768 ...
## $ y_sim : num -14237 -14499 -13065 -9077 -2200 ...
## $ z_sim : num 6387 7130 7034 5548 2272 ...
## $ Vx_sim: num 4.33 5.08 5.52 4.85 1.94 ...
## $ Vy_sim: num -0.693 0.361 2.013 4.339 6.193 ...
## $ Vz_sim: num 0.811 0.313 -0.539 -1.87 -3.168 ...
```

Although, the epoch column should be in datetime format, it is given as a Factor object. This issue will be addressed in the data preparation stage.

Note how many observations there are for each satellite.

```
obs <- table(train$sat_id)
head(obs, n = 5)

##
##    0     1     2     3     4
##  958 2108 417  354 1210
```

Re-order the number of observations in decreasing order, and subtract 1 because R starts counting at 1 (but sat_id starts counting from 0).

```
sats <- order(obs, decreasing = TRUE) - 1
head(sats, n = 5)

## [1] 372 186 429 321 529
```

Next, obtain the five satellites with the highest number of observations.

```
five_highest <- obs[head(sats, n = 5) + 1]
five_highest

##
## 372 186 429 321 529
## 6320 6204 6074 5673 5665
```

Since Satellite 372 has the highest number of observations at 6320, this satellite was chosen.

```
sat_372 <- train[train$sat_id == 372, ]
```

Reset its index.

```
rownames(sat_372) = NULL
```

Finally, the data for Satellite 372 was written into the home directory as a stand-alone dataset.

```
write.csv(x = sat_372, file = "sat_372.csv", row.names = FALSE)
```

4. Data Preparation

The libraries used for data preparation were first installed and loaded before preparation on the Satellite 372 dataset.

The “lubridate” library is for handling datetimes while the others are standard libraries for handling functions, models and mappings. To list, the libraries required for this section are `lubridate`, `purrr`, `caret` and `dplyr`.

```
# install.packages("lubridate")
library(lubridate)

# install.packages("purrr")
library(purrr)

# install.packages("caret")
library(caret)

# install.packages("dplyr")
library(dplyr)
```

Read in the Satellite 372 data.

```
df <- read.csv("sat_372.csv")

dim(df)

## [1] 6320   15

sum(is.na(df))

## [1] 0

str(df)

## 'data.frame': 6320 obs. of 15 variables:
## $ id    : int 746351 746352 746353 746354 746355 ...
## $ epoch : Factor w/ 6320 levels "2014-01-01T00:00:00.000",...
## $ sat_id: int 372 372 372 372 372 372 372 372 372 ...
## $ x     : num -7865 -8412 -8142 -7118 -5479 ...
## $ y     : num -3130 -220 2712 5396 7619 ...
## $ z     : num 2908 3197 3176 2860 2296 ...
## $ Vx   : num -2.245 -0.322 1.569 3.208 4.455 ...
## $ Vy   : num 6.61 7.01 6.72 5.86 4.57 ...
## $ Vz   : num 1.04 0.317 -0.412 -1.061 -1.572 ...
## $ x_sim: num -7860 -8408 -8139 -7114 -5475 ...
## $ y_sim: num -3132 -221 2713 5398 7621 ...
## $ z_sim: num 2905 3195 3174 2859 2296 ...
## $ Vx_sim: num -2.248 -0.323 1.569 3.209 4.455 ...
## $ Vy_sim: num 6.61 7.01 6.73 5.86 4.58 ...
## $ Vz_sim: num 1.041 0.319 -0.411 -1.06 -1.571 ...
```

Satellite 372 data has 6320 observations and 15 variables, with no evidence of missing values. Checking the structure indicates that the epoch column, which should consist of timestamps, is read as a Factor instead. Thus, this variable is converted into the POSIXct datetime format:

```
# set option to convert fractional seconds, up to 3 decimal places
op <- options(digits.secs = 3)

# convert epoch into datetime
```

```

df["epoch"] <- map(df["epoch"], ymd_hms)

class(df$epoch)

```

```
## [1] "POSIXct" "POSIXt"
```

Now that epoch is in proper datetime format, the time difference between two epochs can be calculated. This is important to see because it shows how often the coordinate and velocity data were sampled from the simulations.

```

num_obs <- dim(df)[1]

# calculate time difference between two successive epochs
time_diff <- df[2:num_obs, "epoch"] - df[1:num_obs-1, "epoch"]

length(time_diff) # 6319 observations, as opposed to 6320 in the dataset

## [1] 6319

# the first element shouldn't be there because there is no epoch 0 before epoch 1
# so fill that entry with the next element
time_diff <- prepend(time_diff, time_diff[1])

df["delta_t"] <- time_diff

```

Next, the absolute record of time, starting from January 1 00:00:00 as being the zero point in time, can be calculated and added as another column. Absolute time will be a useful additional predictor variable for training machine learning models.

```

df["abs_time"] <- df$epoch - df[1,"epoch"]

head(df$abs_time)

## Time differences in secs
## [1] 0.000 423.869 847.738 1271.606 1695.475 2119.344

```

In order to follow the approach used by Peng and Bai (2019), which is Approach 3 described in the Methodology section, six error variables were added for each of the six kinematic states.

```

df["e_x"] = df["x"] - df["x_sim"]
df["e_y"] = df["y"] - df["y_sim"]
df["e_z"] = df["z"] - df["z_sim"]
df["e_Vx"] = df["Vx"] - df["Vx_sim"]
df["e_Vy"] = df["Vy"] - df["Vy_sim"]
df["e_Vz"] = df["Vz"] - df["Vz_sim"]

```

When using Approach 3, these error columns will be the target variables.

While exploring the `delta_t` variable, an anomaly was detected. For the purpose of data preparation, the steps in removing this anomaly was added to this section although the same steps were done in the subsequent Data Exploration section.

```

"
# delete anomaly at ID 750171 (please refer to Section 5a: Data Exploration 1)
df <- filter(df, df$id != 750171)
anomaly_index <- as.numeric(rownames(df[df$id == 750172,]))
df[anomaly_index, 'delta_t'] <- df[anomaly_index-1, 'delta_t']
"

```

Finally, the dataset can be split into Train and Test components. The Test data was taken from the last 7 days while the Train dataset was taken from the first 25 days. This is because 7 days is the minimum time period in which response to imminent threats, such as collisions between space objects, can be initiated to avoid that danger (Peng and Bai, 2019).

```

# split data into Train and Test. Test is last 7 days
# Train is from days 1:24 and Test is from days 25:31
train <- filter(df, as.numeric(unlist(map(df['epoch'], day))) < 25)
test  <- filter(df, as.numeric(unlist(map(df['epoch'], day))) >= 25)

```

Export Prepared, Train and Test datasets to the home directory.

```

write.csv(x = df, file = "sat_372_prepared.csv", row.names = FALSE)
write.csv(x = train, file = "sat_372_train.csv", row.names = FALSE)
write.csv(x = test, file = "sat_372_test.csv", row.names = FALSE)

```

5a. Data Exploration Part 1

Load the libraries used for data exploration, read in the dataset and change epoch class to datetime. The libraries required for this section are `lubridate` and `ggplot2`.

```

library(lubridate)
library(ggplot2)

df <- read.csv("sat_372_prepared.csv")

# change epoch class to POSIXct
df$epoch <- as.POSIXct(df$epoch, tz = "UTC")

# set ggplot black and white background theme
theme_set(theme_bw())

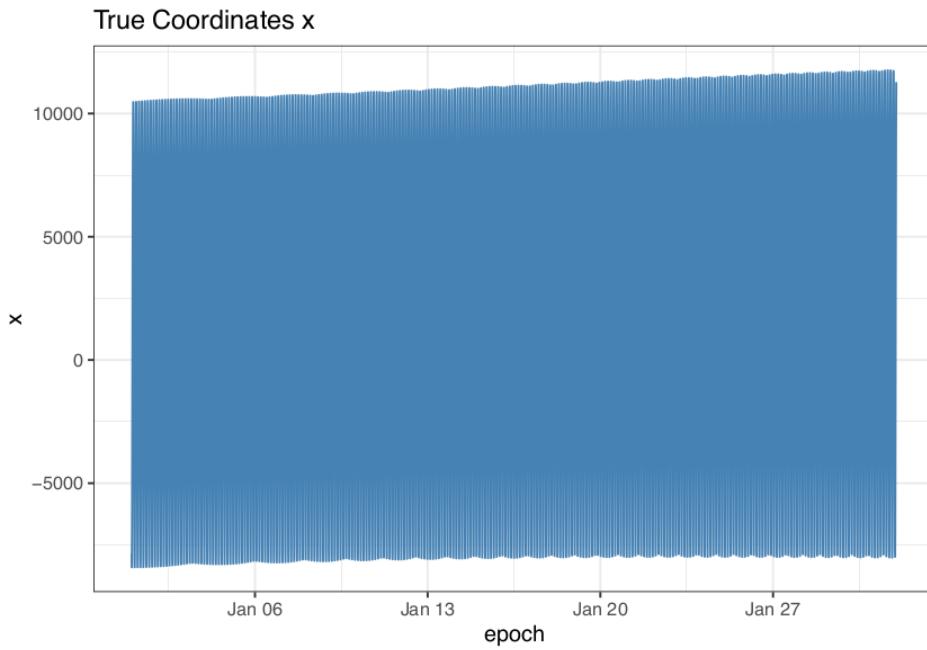
```

Firstly, all 6320 observations for the variable `x` were plotted over 31 days to see how it varies.

```

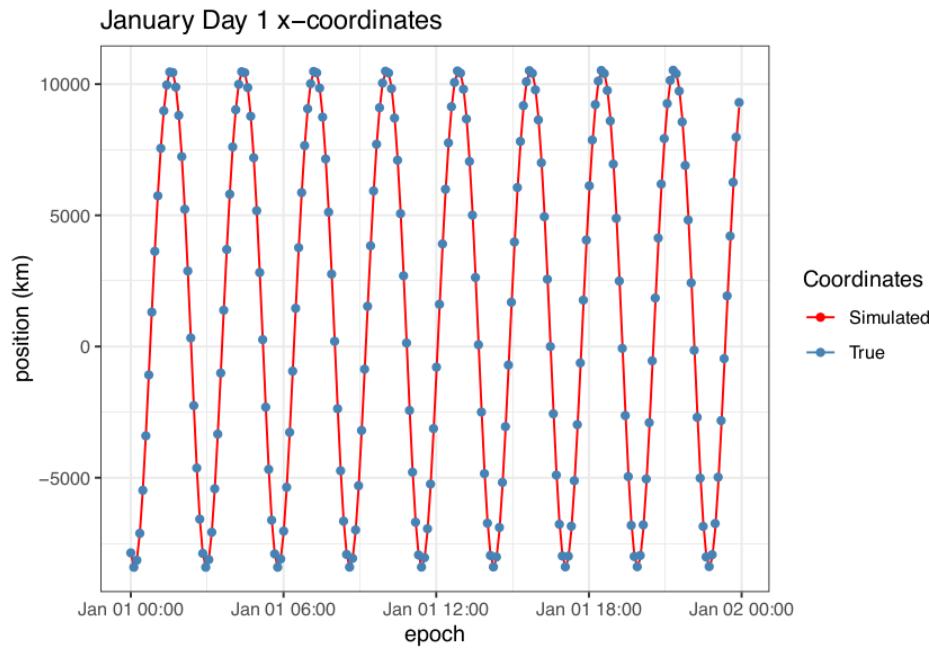
ggplot(
  aes(x = epoch),
  data = df
) + geom_line(aes(y = x), color = "steelblue") +
  labs(title = "True Coordinates x")

```



The plot above indicates a somewhat periodic pattern, but it is difficult to see the detailed features of the graph, so only the first 204 observations will be plotted in subsequent visualizations. There are 6320 observations over 31 days, so one day is approximately 204 observations, assuming equal time interval between observations. Moreover, to see the comparison between the true and simulated coordinates, both of them were plotted on the same figure.

```
ggplot(
  aes(x = epoch),
  data = head(df, 204)
) + geom_line(aes(y = x_sim, colour = "Simulated")) +
  geom_point(aes(y = x, colour = "True")) +
  scale_colour_manual(values = c("red", "steelblue")) +
  labs(title = "January Day 1 x-coordinates", colour = "Coordinates") +
  ylab("position (km)")
```



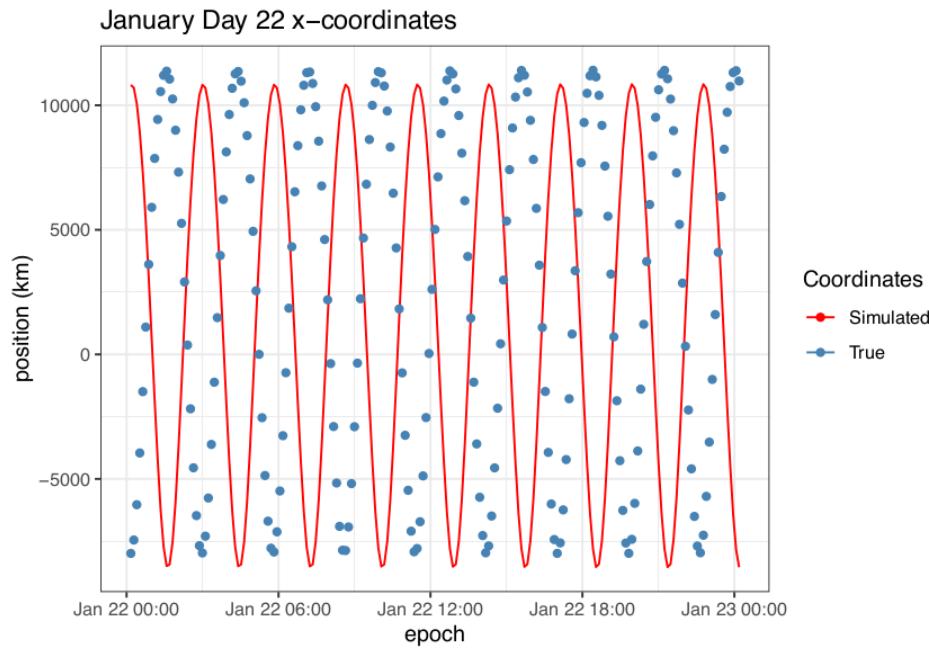
At first glance, it looks as if the simulated values are very good at approximating the true coordinate. To see if this accuracy persists, the same plot on Day 22 was done.

```

start = 204 * 21
stop = 204 * 22

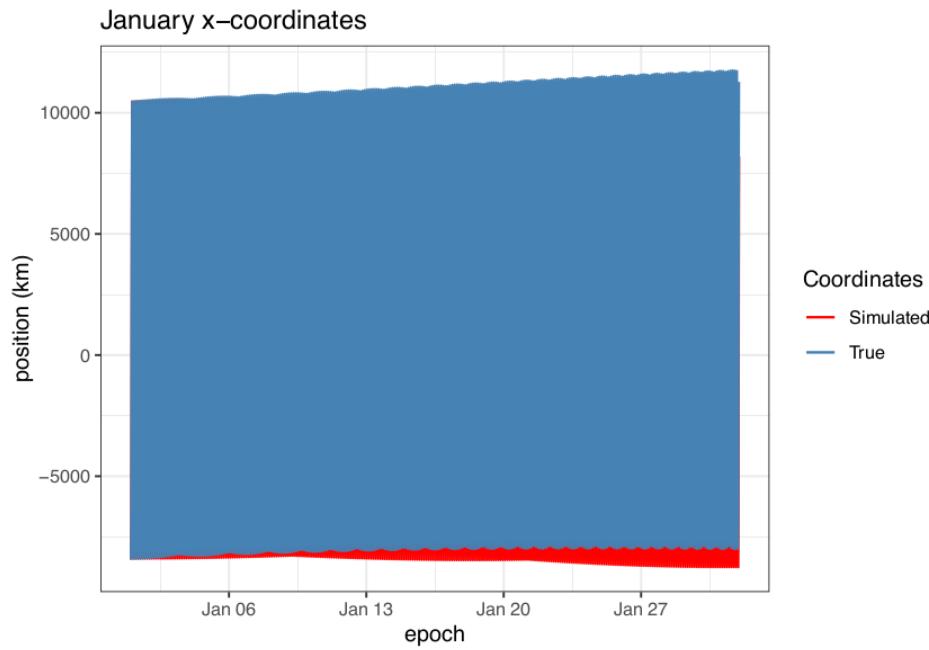
ggplot(
  aes(x = epoch),
  data = df[start:stop,]
) + geom_line(aes(y = x_sim, colour = "Simulated")) +
  geom_point(aes(y = x, colour = "True")) +
  scale_colour_manual(values = c("red", "steelblue")) +
  labs(title = "January Day 22 x-coordinates", colour = "Coordinates") +
  ylab("position (km)")

```



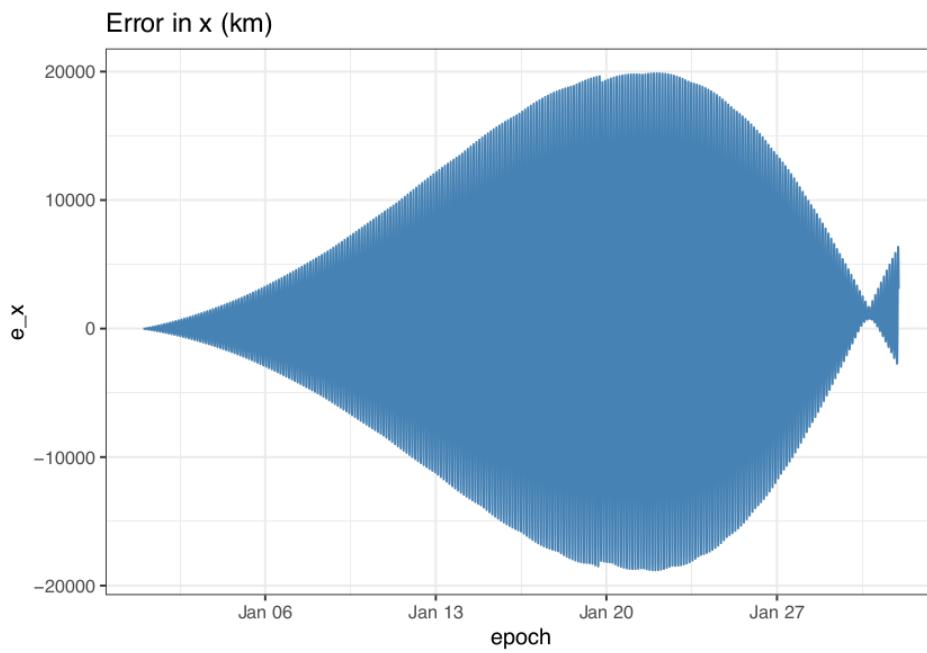
A clear lag (or phase shift) between the true and simulated coordinates can be observed. Moreover, the overall plot for the entire 31 days was plotted.

```
ggplot(
  aes(x = epoch),
  data = df
) + geom_line(aes(y = x_sim, colour = "Simulated")) +
  geom_line(aes(y = x, colour = "True")) +
  scale_colour_manual(values = c("red", "steelblue")) +
  labs(title = "January x-coordinates", colour = "Coordinates") +
  ylab("position (km)")
```



Again, it is somewhat difficult to make out the details, but the fitting capacity of the simulated values on the true coordinates becomes poorer as time passes. This brings the discussion to visualizing the errors themselves, for instance, the error in x given by the variable $e_x = x - x_{sim}$.

```
ggplot(
  aes(x = epoch),
  data = df
) + geom_line(aes(y = e_x), color = "steelblue") +
  labs(title = "Error in x (km)")
```



The e_x variable looks like a spade-shaped object - in which the error starts out small and increases (this was expected) but peaks around Day 21-23 and then decreases to a small value at Day 31 (this was unexpected). After this, the error increases over Day 31. Visualizing the other errors also indicates the same spade-shaped pattern, albeit with the peaks occurring on different days.

Thus, it can be presumed that the simulated coordinates lag behind the true coordinates by some phase shift, which then corrects itself over 30 days. In fact, the plot above indicates two kinds of errors: the error in predicting the true coordinate, and the failure to do so at the correct epochs.

Similar visualizations can be done for the remaining five kinematic states.

Next, the anomaly mentioned in Section 2: Data Preparation needs to be dealt with. This anomaly was first discovered when exploring the `delta_t` variable.

```
table(df$delta_t)

##
## 0.00099927520751953    423.867999792099    423.868000030518
##          1                  191                 1298
## 423.868999958038    423.869000196457
##          3979                851
```

One observation has a `delta_t` value significantly different from the others (approximately 0.001s). Look at that observation:

```

filter(df, df$delta_t < 1)

##      id          epoch sat_id      x      y      z      Vx
## 1 750172 2014-01-19 17:46:18.68    372 8581.831 -7292.922 -1281.879 -3.698617
##           Vy      Vz   x_sim   y_sim   z_sim   Vx_sim   Vy_sim   Vz_sim
## 1 -3.662911 1.957172 804.142 9700.123 -2179.557 6.068693 -0.1743378 -1.948784
##      delta_t abs_time     e_x     e_y     e_z     e_Vx     e_Vy     e_Vz
## 1 0.0009999275 1619179 7777.689 -16993.05 897.6774 -9.76731 -3.488573 3.905956

```

This anomaly is at ID 750172. Looking at the two observations on either side of it could provide more information.

```

filter(df, df$id %in% c(750170:750174))

##      id          epoch sat_id      x      y      z      Vx
## 1 750170 2014-01-19 17:39:14.809    372 9931.653 -5577.948 -2074.1662 -2.656222
## 2 750171 2014-01-19 17:46:18.680    372 8581.835 -7292.919 -1281.8812 -3.698615
## 3 750172 2014-01-19 17:46:18.680    372 8581.831 -7292.922 -1281.8792 -3.698617
## 4 750173 2014-01-19 17:53:22.549    372 6811.510 -8648.768 -427.0761 -4.632076
## 5 750174 2014-01-19 18:00:26.418    372 4678.618 -9541.137 449.8514 -5.397970
##           Vy      Vz   x_sim   y_sim   z_sim   Vx_sim   Vy_sim
## 1 -4.389228 1.768332 -4192.864 8385.487 -286.5656 5.386392 3.3003315
## 2 -3.662913 1.957171 -1765.593 9413.773 -1283.1297 5.978466 1.5441521
## 3 -3.662911 1.957172 804.142 9700.123 -2179.5566 6.068693 -0.1743378
## 4 -2.693422 2.060363 3319.096 9289.577 -2920.2035 5.734835 -1.7275924
## 5 -1.475909 2.057826 5619.019 8269.602 -3469.9314 5.069336 -3.0410557
##           Vz_sim   delta_t abs_time     e_x     e_y     e_z     e_Vx
## 1 -2.412996 4.238690e+02 1618755 14124.5175 -13963.44 -1787.600627 -8.042614
## 2 -2.259280 4.238690e+02 1619179 10347.4273 -16706.69 1.248568 -9.677081
## 3 -1.948784 9.999275e-04 1619179 7777.6890 -16993.05 897.677397 -9.767310
## 4 -1.532216 4.238690e+02 1619603 3492.4143 -17938.35 2493.127342 -10.366911
## 5 -1.054605 4.238690e+02 1620026 -940.4005 -17810.74 3919.782802 -10.467306
##           e_Vy     e_Vz
## 1 -7.6895595 4.181328
## 2 -5.2070655 4.216451
## 3 -3.4885735 3.905956
## 4 -0.9658295 3.592579
## 5  1.5651471 3.112430

```

From inspecting the `epoch` and `abs_time` variables, it seems as though the simulation was updated within a millisecond, because while x_{sim} changes by a lot (from -1765.593 to 804.142), x changes very little (from 8581.835 to 8581.831) within that millisecond.

The solution to this anomaly was to keep the updated observation (which is ID 750172) and delete the old observation (ID 750171).

```

df <- filter(df, df$id != 750171)

# replace the 1 millisecond value with the delta_t value before it
anomaly_index <- as.numeric(rownames(df[df$id == 750172,]))
df[anomaly_index, 'delta_t'] <- df[anomaly_index-1, 'delta_t']

```

Other than this anomaly, coordinate data was seen to be sampled at equally spaced intervals of 7.06 minutes.

```

mean(df$delta_t) / 60

## [1] 7.064479

sd(df$delta_t) / 60

## [1] 7.073848e-06

```

Finally, it is suggested for the reader to rerun the code chunk from Section 4: Data Preparation to take into account this anomaly before splitting the dataset into Train and Test.

5b. Data Exploration Part 2

This section explores the correlation between variables as well as testing for normality and deducing the orbit type of Satellite 372. The libraries required for this section are `Hmisc` and `corrplot`.

```

#install.packages("Hmisc")
library("Hmisc")

#install.packages("corrplot")
library("corrplot")

df <- read.csv("sat_372_prepared.csv")

df$epoch <- as.POSIXct(df$epoch, tz = "UTC")

```

Confirm that the anomaly detected from Section 3: Data Exploration is absent.

```

dim(df)

## [1] 6320   23

```

The dataset should have 6319 observations. Otherwise, the anomaly is still present.

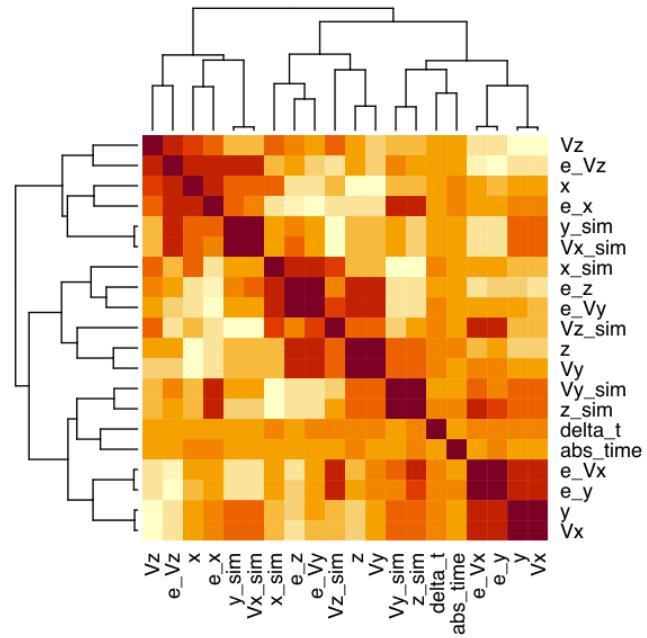
Firstly, a correlation matrix between all the relevant variables was calculated and rounded to three significant figures. After this, a heatmap (including the dendrogram) and correlation plot was graphed to visualize it.

```

corrrmat <- round(rcorr(as.matrix(df[c(4:23)]))$r, 3)
#corrrmat

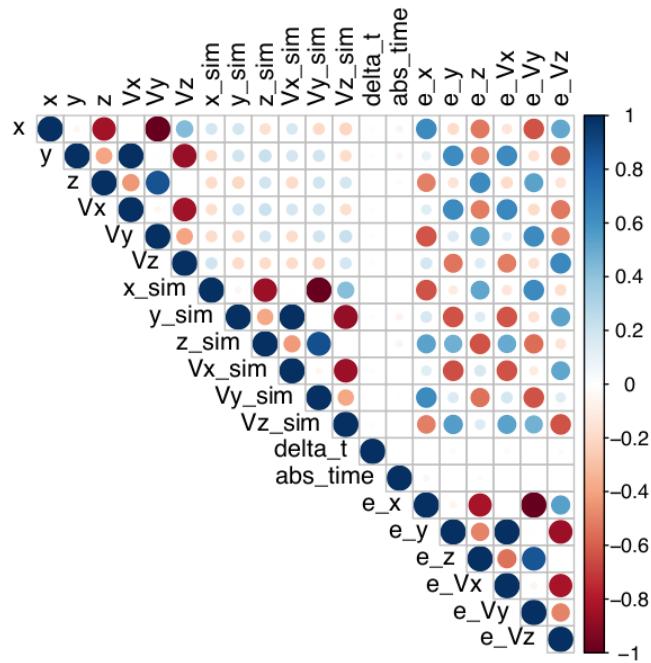
# visualize a heatmap of the correlation matrix
heatmap(x = corrrmat, symm = TRUE)

```



Another visualization of the correlation matrix:

```
corrplot(corrmat, type = "upper", tl.col = "black",
         sig.level = 0.01, insig = "blank")
```



The above plot indicates a stronger correlation between simulated states and errors, as opposed to the weaker correlation between simulated states and true states. This justifies the use of Approach 3 over Approach 2 (discussed in Section 2: Methodology) in treating the errors as the target variables instead of true states.

Next, the normality of each variable was tested using the Shapiro-Wilk's test for the first 5000 observations.

```
# Shapiro-Wilk's test for the first 5000 observations
for (i in c(4:23)) {
  print(names(df)[i])
  print(shapiro.test(x = df[1:5000, i]))
}

## [1] "x"
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.90162, p-value < 2.2e-16
##
## [1] "y"
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.90234, p-value < 2.2e-16
##
## [1] "z"
```

```

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.89957, p-value < 2.2e-16  

##  

## [1] "Vx"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.90567, p-value < 2.2e-16  

##  

## [1] "Vy"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.89357, p-value < 2.2e-16  

##  

## [1] "Vz"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.91224, p-value < 2.2e-16  

##  

## [1] "x_sim"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.8998, p-value < 2.2e-16  

##  

## [1] "y_sim"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.90252, p-value < 2.2e-16  

##  

## [1] "z_sim"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.90062, p-value < 2.2e-16  

##  

## [1] "Vx_sim"  

##  

## Shapiro-Wilk normality test  

##  

## data: df[1:5000, i]  

## W = 0.90446, p-value < 2.2e-16

```

```

## [1] "Vy_sim"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.89515, p-value < 2.2e-16
##
## [1] "Vz_sim"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.90856, p-value < 2.2e-16
##
## [1] "delta_t"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.0030214, p-value < 2.2e-16
##
## [1] "abs_time"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.9549, p-value < 2.2e-16
##
## [1] "e_x"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.9779, p-value < 2.2e-16
##
## [1] "e_y"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.9717, p-value < 2.2e-16
##
## [1] "e_z"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.97776, p-value < 2.2e-16
##
## [1] "e_Vx"
##
## Shapiro-Wilk normality test
##

```

```

## data: df[1:5000, i]
## W = 0.97683, p-value < 2.2e-16
##
## [1] "e_Vy"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.97773, p-value < 2.2e-16
##
## [1] "e_Vz"
##
## Shapiro-Wilk normality test
##
## data: df[1:5000, i]
## W = 0.97591, p-value < 2.2e-16

```

All the variables indicate a significant p-value, which means they are all non-normal. Thus, for the purpose of scaling data, min-max normalization should be used instead of standardization.

Finally, the orbit type of Satellite 372 can be determined by calculating its radius of orbit.

```

radius <- sqrt( (df$x)^2 + (df$y)^2 + (df$z)^2 )

summary(radius)

##      Min. 1st Qu. Median   Mean 3rd Qu.   Max.
##     8166    9158   10366   10248   11323   12027

```

The satellite orbit radius ranges from a minimum of 8166 km to a maximum of 12027 km, with a mean radius of 10248 km. This orbit is greater than the 2000 km threshold for Low Earth Orbit but less than the 35786 km lower limit for High Earth Orbit. Hence, Satellite 372 is an MEO (Medium Earth Orbit) object.

6. Data Preprocessing

Following the Shapiro-Wilk's normality test done in the previous section, standardization will not be done. Instead, min-max normalization will be applied to the predictor variables, which are the simulated states. Moreover, the target variables should not be scaled.

The library required for this section is `caret`.

```

library(caret)

train <- read.csv("sat_372_train.csv")
test  <- read.csv("sat_372_test.csv")

```

To prevent data leakage, which means making use of data from sources other than the training data, the min-max normalization was only fitted to the Train dataset.

```

# the fitted normalization parameters from the Train dataset
prep_params <- preProcess(train[c(10:15,17)], method = c("range"))
prep_params

```

```

## Created from 4894 samples and 7 variables
##
## Pre-processing:
##   - ignored (0)
##   - re-scaling to [0, 1] (7)

```

Using these parameters, both Train and Test data was normalized and updated using min-max scaling.

```

train_norm <- predict(prep_params, train)
test_norm <- predict(prep_params, test)

```

To check if the variables are properly normalized, call on the `summary` function:

```
summary(train_norm)
```

```

##      id          epoch        sat_id       x
## Min. :746351  2014-01-19 17:46:18.680: 2  Min. :372  Min. :-8412
## 1st Qu.:747574 2014-01-01 00:00:00.000: 1  1st Qu.:372  1st Qu.:-4532
## Median :748798 2014-01-01 00:07:03.868: 1  Median :372  Median : 2844
## Mean   :748798 2014-01-01 00:14:07.737: 1  Mean   :372  Mean   : 2148
## 3rd Qu.:750021 2014-01-01 00:21:11.605: 1  3rd Qu.:372  3rd Qu.: 8783
## Max.  :751244 2014-01-01 00:28:15.474: 1  Max.  :372  Max.  :11502
## (Other)          :4887
##      y          z          Vx          Vy
## Min. :-9874.6  Min. :-4157.7  Min. :-6.133092  Min. :-5.60307
## 1st Qu.:-6839.7 1st Qu.:-3267.5 1st Qu.:-4.156911 1st Qu.:-4.24357
## Median : 212.5 Median : -990.4 Median : 0.114163 Median : -0.88744
## Mean   : 165.4 Mean   : -742.7 Mean   : 0.007607 Mean   : -0.00145
## 3rd Qu.: 7172.4 3rd Qu.: 1778.6 3rd Qu.: 4.174043 3rd Qu.: 4.19625
## Max.  :10491.9 Max.  : 3207.6 Max.  : 6.227322 Max.  : 7.31246
##
##      Vz          x_sim        y_sim        z_sim
## Min. :-2.604052 Min. :0.0000  Min. :0.0000  Min. :0.0000
## 1st Qu.:-1.566498 1st Qu.:0.1856  1st Qu.:0.1500  1st Qu.:0.1283
## Median : 0.088180 Median :0.5551  Median :0.4927  Median :0.4456
## Mean   : -0.001493 Mean   :0.5278  Mean   :0.4931  Mean   :0.4714
## 3rd Qu.: 1.570320 3rd Qu.:0.8713  3rd Qu.:0.8353  3rd Qu.:0.8133
## Max.  : 2.352936 Max.  :1.0000  Max.  :1.0000  Max.  :1.0000
##
##      Vx_sim        Vy_sim        Vz_sim        delta_t
## Min. :0.0000  Min. :0.0000  Min. :0.0000  Min. : 0.001
## 1st Qu.:0.1573 1st Qu.:0.1032  1st Qu.:0.1874  1st Qu.:423.869
## Median :0.5007 Median :0.3883  Median :0.5265  Median :423.869
## Mean   :0.4955 Mean   :0.4436  Mean   :0.5127  Mean   :423.782
## 3rd Qu.:0.8321 3rd Qu.:0.7816  3rd Qu.:0.8386  3rd Qu.:423.869
## Max.  :1.0000  Max.  :1.0000  Max.  :1.0000  Max.  :423.869
##
##      abs_time       e_x          e_y          e_z
## Min. :0.0000  Min. :-18807.21  Min. :-19405.76  Min. :-7386.94
## 1st Qu.:0.2501 1st Qu.:-4742.68  1st Qu.:-5477.33  1st Qu.:-2216.65
## Median :0.5001 Median : 89.99  Median : -93.76  Median : -27.28
## Mean   :0.5001 Mean   : 477.50  Mean   : 130.10  Mean   : -179.91
## 3rd Qu.:0.7502 3rd Qu.: 5842.16  3rd Qu.: 5572.86  3rd Qu.: 1812.12

```

```

## Max. :1.0000  Max. : 19866.85  Max. : 19793.91  Max. : 7005.75
##
##      e_Vx          e_Vy          e_Vz
## Min. :-12.281838  Min. :-12.019945  Min. :-4.591500
## 1st Qu.: -3.374420 1st Qu.: -3.351361 1st Qu.: -1.222402
## Median : -0.181315 Median : -0.009739 Median : 0.074519
## Mean   :  0.007732 Mean   : -0.001714 Mean   : -0.001723
## 3rd Qu.:  3.292516 3rd Qu.:  3.297733 3rd Qu.: 1.265914
## Max.  : 11.964720 Max.  : 12.777061 Max.  : 4.476471
##
## summary(test_norm)

##      id           epoch       sat_id      x
## Min. :751245 2014-01-25 00:06:29.865: 1  Min. :372  Min. :-8018
## 1st Qu.:751601 2014-01-25 00:13:33.733: 1  1st Qu.:372  1st Qu.:-4023
## Median :751958 2014-01-25 00:20:37.602: 1  Median :372  Median : 3589
## Mean   :751958 2014-01-25 00:27:41.470: 1  Mean   :372  Mean   : 2709
## 3rd Qu.:752314 2014-01-25 00:34:45.339: 1  3rd Qu.:372  3rd Qu.: 9468
## Max.  :752670 2014-01-25 00:41:49.209: 1  Max.  :372  Max.  :11755
## (Other)          :1420
##      y           z           Vx          Vy
## Min. :-9875.2  Min. :-4030.9  Min. :-6.087401  Min. :-5.07677
## 1st Qu.:-7282.8 1st Qu.:-3056.8 1st Qu.:-4.268321 1st Qu.:-4.05188
## Median :-688.6  Median :-691.6  Median :-0.132249 Median :-1.02339
## Mean   :-496.6  Mean   :-524.1  Mean   : 0.002576  Mean   : 0.01123
## 3rd Qu.: 6241.1 3rd Qu.: 1996.7 3rd Qu.: 4.254049 3rd Qu.: 3.96956
## Max.  : 9393.0  Max.  : 3327.0  Max.  : 6.471717  Max.  : 7.29461
##
##      Vz          x_sim        y_sim        z_sim
## Min. :-2.727405  Min. :-0.006586  Min. :-0.001688  Min. : 0.02956
## 1st Qu.:-1.512639 1st Qu.: 0.175288 1st Qu.: 0.130283 1st Qu.: 0.17023
## Median : 0.334431 Median : 0.554313 Median : 0.457731 Median : 0.50256
## Mean   :-0.002398 Mean   : 0.524977 Mean   : 0.460655 Mean   : 0.51582
## 3rd Qu.: 1.543000 3rd Qu.: 0.874699 3rd Qu.: 0.788289 3rd Qu.: 0.86170
## Max.  : 1.989109 Max.  : 1.004034 Max.  : 0.946567 Max.  : 1.02429
##
##      Vx_sim        Vy_sim        Vz_sim      delta_t
## Min. :0.01816  Min. :0.01346  Min. :-0.007712  Min. :423.9
## 1st Qu.:0.14910 1st Qu.:0.11269 1st Qu.: 0.188485 1st Qu.:423.9
## Median :0.48310 Median :0.40378 Median : 0.557162 Median :423.9
## Mean   :0.49773 Mean   :0.44479 Mean   : 0.510327 Mean   :423.9
## 3rd Qu.: 0.84356 3rd Qu.:0.78218 3rd Qu.: 0.833230 3rd Qu.:423.9
## Max.  : 1.01507 Max.  : 0.98437 Max.  : 0.933053 Max.  :423.9
##
##      abs_time      e_x          e_y          e_z
## Min. :1.000  Min. :-16062  Min. :-17766.5  Min. :-5971.3
## 1st Qu.:1.073 1st Qu.: -2616  1st Qu.: -3163.5  1st Qu.: -1796.9
## Median :1.146  Median :  996  Median :  326.0  Median : -306.0
## Mean   :1.146  Mean   : 1093  Mean   : 138.5  Mean   : -289.8
## 3rd Qu.:1.219 3rd Qu.: 5162  3rd Qu.: 4298.9  3rd Qu.:  941.7
## Max.  :1.291  Max.  : 17437  Max.  : 15442.8  Max.  : 6266.3
##
##      e_Vx          e_Vy          e_Vz

```

```

## Min.   :-11.87655   Min.   :-10.293136   Min.   :-3.567586
## 1st Qu.: -1.88445   1st Qu.: -2.500300   1st Qu.: -0.935988
## Median :  0.21829   Median :  0.150848   Median : -0.093711
## Mean   : -0.02484   Mean   : -0.003895   Mean   :  0.008938
## 3rd Qu.:  2.83788   3rd Qu.:  2.109473   3rd Qu.:  0.830561
## Max.   :  8.73853   Max.   : 11.278976   Max.   :  4.185510
##

```

All simulated states in the Train dataset are now within the interval [0,1]. For the Test dataset, some simulated states contain negative values, which is the result of fitting the normalization parameters on the Train dataset only.

The normalized data was then exported to the Home Directory.

```

write.csv(x = train_norm, file = "sat_372_normalized_train.csv", row.names = FALSE)
write.csv(x = test_norm, file = "sat_372_normalized_test.csv", row.names = FALSE)

```

7. Non-Linear Regression Model

When implementing a non-linear fit, there is no need to use normalized data. Therefore, the non-normalized prepared datasets, including the splitted Train and Test, were loaded. The libraries required for this section are `caret` and `Metrics`.

```

library(caret)

#install.packages("Metrics")
library(Metrics)

##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
## 
##     precision, recall

train <- read.csv("sat_372_train.csv")
test  <- read.csv("sat_372_test.csv")

```

One problem arose while implementing the non-linear regression. Fitting on all obervations in the Train dataset results in non-sensical values for the model parameters. After trying out various amounts of obervations, the last 979 observations of the Train dataset were used to fit the model because it gave one of the lowest residual standard errors. Moreover, it resulted in statistically significant values for the model parameters.

```

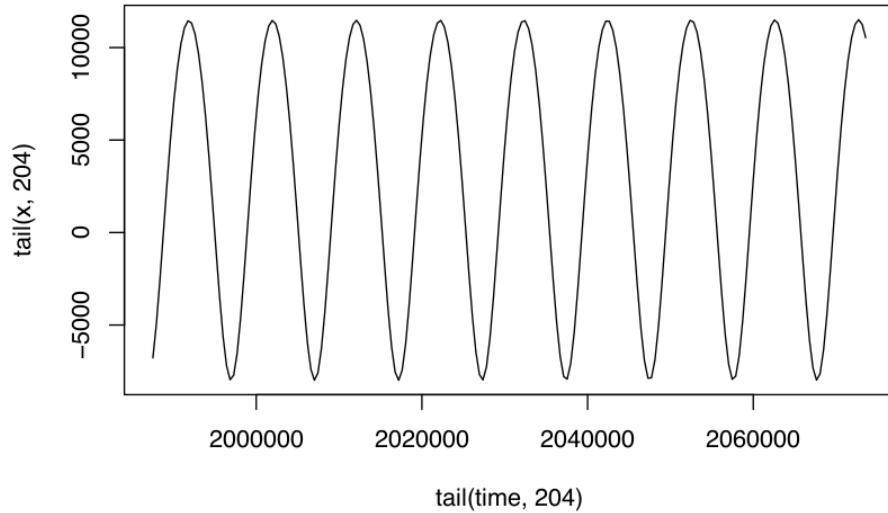
# train model on the last 1/5th of the Train data
len <- ceiling(dim(train)[1] / 5)
df  <- tail(train, len)

time <- df[, "abs_time"]
x    <- df[, "x"]
sine_func <- x ~ A*sin(2*pi*time/t + phi) + m*time + c

```

Fitting the `nls` function requires the user to provide starting values. Some of these values were obtained by inspecting a plot of the final day (Day 31) in the Train dataset and approximating them by eye.

```
plot(x = tail(time, 204), y = tail(x, 204), type = "l")
```



From the above plot, the x-coordinate varies between -7000 to 12000, so the amplitude `A` was estimated as 10000. Likewise, the time difference between every other peak (or trough) appears to be separated by 20000 seconds, so the period `t` of the curve was estimated to be 10000. The phase shift `phi` and intercept `c` was just provided as 1 while the very gentle upward slope `m` was given the small positive starting value 0.01. These parameters need not be accurate as the model will automatically find the best fitted parameters.

After inspection, the non-linear regression model was implemented.

```
nls_model <- nls(formula = sine_func,
                   data = data.frame(time, x),
                   start = list(A = 10000, t = 10000, phi = 1, m = 0.01, c = 1))
```

The model details, summary and coefficients can be viewed below.

```
nls_model
```

```
## Nonlinear regression model
##   model: x ~ A * sin(2 * pi * time/t + phi) + m * time + c
##   data: data.frame(time, x)
##       A      t      phi      m      c
## 10000 10000     1 0.01 1.00
```

```

## 9.582e+03 1.011e+04 1.400e+01 3.881e-04 1.829e+03
## residual sum-of-squares: 351749287
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 3.846e-07

summary(nls_model)

##
## Formula: x ~ A * sin(2 * pi * time/t + phi) + m * time + c
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## A    9.582e+03  2.715e+01  352.85 < 2e-16 ***
## t    1.011e+04  3.854e-01 26232.81 < 2e-16 ***
## phi  1.400e+01  4.429e-02   316.16 < 2e-16 ***
## m    3.881e-04  1.604e-04     2.42   0.0157 *
## c    1.829e+03  2.999e+02      6.10  1.53e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 600.9 on 974 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 3.846e-07

coefficients(nls_model)

##           A             t             phi            m            c
## 9.581631e+03 1.011001e+04 1.400285e+01 3.881093e-04 1.829344e+03

param <- as.numeric(coefficients(nls_model))
param

## [1] 9.581631e+03 1.011001e+04 1.400285e+01 3.881093e-04 1.829344e+03

```

These parameters define an upward sloping sine curve, which can be extrapolated over the epochs from the Test dataset.

```

predict_state <- function(state, parameters, data) {

  # `state` is the desired kinematic state; must be char(x,y,z,Vx,Vy,Vz)
  # `parameters` are the model parameters obtained from fitting nls
  # `data` can be the Test data, but can also be observations from Train

  t <- data[, "abs_time"]
  k <- data[, state]
  k_hat <- param[1] * sin(2*pi*t / param[2] + param[3]) + param[4]*t + param[5]

  return(k_hat)
}

```

```

}

x_hat_nls <- predict_state(state = "x", parameters = param, data = test)

```

The performance of the model was tested on three metrics: RMSE, SMAPE and MAPRE. RMSE and SMAPE are already part of the `Metrics` library in R but the MAPRE metric needs to be manually defined.

```

mapre <- function(actual, predicted) {

  metric <- 100 * (sum(abs(actual - predicted))) / (sum(abs(actual)))

  return(metric)

}

```

Note that the following usage of the MAPRE metric does not conform with the definition presented in Peng and Bai (2019) because it uses coordinates instead of errors. Moreover, R's definition of the SMAPE metric confines the output within the interval [0, 2], so the resulting metric needs to be divided by 2 and multiplied by 100% to give the standard percentage value.

```

evaluate_model <- function(true, estimated) {

  r <- rmse(actual = true, predicted = estimated)
  s <- smape(actual = true, predicted = estimated) * 100/2
  m <- mapre(actual = true, predicted = estimated)

  eval = c(r,s,m)
  names(eval) <- c("RMSE", "SMAPE", "MAPRE")

  return(eval)

}

nls_eval = evaluate_model(true = test$x, estimated = x_hat_nls)
nls_eval

##      RMSE      SMAPE      MAPRE
## 1699.58460  20.05577  20.64286

```

Similar models can be built for the remaining kinematic states, but will not be done in this study.

8. Support Vector Regression Model

For implementing the SVR model, normalized data needs to be used. The libraries required for this section are `ggplot2`, `Metrics` and `e1071`. Additionally, the user-created `mapre` and `evaluate_model` functions from the previous section need to be activated.

```

library(ggplot2)
library(e1071)

```

```

##  

## Attaching package: 'e1071'  

##  

## The following object is masked from 'package:Hmisc':  

##  

##     impute  

library(Metrics)  

train <- read.csv("sat_372_normalized_train.csv")  

test  <- read.csv("sat_372_normalized_test.csv")  

func <- e_x ~ abs_time + x_sim + y_sim + z_sim + Vx_sim + Vy_sim + Vz_sim

```

The dependent variable to be predicted is the error in `x` given by `e_x`. The predictor variables are the simulated kinematic states. Initially, the default SVR model without tuning was implemented and the error metrics calculated.

```

svr_default <- svm(formula = func, data = train, scale = FALSE)
summary(svr_default)

```

```

##  

## Call:  

## svm(formula = func, data = train, scale = FALSE)  

##  

##  

## Parameters:  

##   SVM-Type:  eps-regression  

##   SVM-Kernel: radial  

##       cost:  1  

##      gamma: 0.1428571  

##    epsilon: 0.1  

##  

##  

## Number of Support Vectors:  4892  

e_x_hat_svr1 <- predict(svr_default, test)
svr_default_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr1)
svr_default_eval

```

| | RMSE | SMAPE | MAPRE |
|----|------------|----------|----------|
| ## | 6982.30680 | 89.79596 | 99.30569 |

The results are not very promising compared to the non-linear regression model. However, more optimal values of the hyperparameters `epsilon` and `cost` were found through a grid search.

```

"  

#Tuning the hyperparameters epsilon and C  

set.seed(123)
svr_default_tuned <- tune(method = svm, train.x = func, data = train,

```

```

ranges = list(epsilon = seq(0,1,0.1), cost = 2^(0:9)))
plot(svr_default_tuned)
summary(svr_default_tuned)

opt_model = svr_default_tuned$best.model
summary(opt_model)

e_x_hat_svr1 <- predict(opt_model, test)
svr_default_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr1)
svr_default_eval #cost=512, gamma=0.25, epsilon=0
""

## [1] "\n#Tuning the hyperparameters epsilon and C\nset.seed(123)\nsvr_default_tuned <- tune(method

svr_default_tuned <- svm(formula = func, data = train, scale = FALSE, gamma = 0.25, epsilon = 0, cost =
summary(svr_default_tuned)

##
## Call:
## svm(formula = func, data = train, gamma = 0.25, epsilon = 0, cost = 512,
##      scale = FALSE)
##
##
## Parameters:
##   SVM-Type:  eps-regression
##   SVM-Kernel: radial
##     cost: 512
##     gamma: 0.25
##     epsilon: 0
##
##
## Number of Support Vectors:  4894

e_x_hat_svr1_tuned <- predict(svr_default_tuned, test)
svr_default_tuned_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr1_tuned)
svr_default_tuned_eval

##          RMSE        SMAPE       MAPRE
## 13007.87177    69.08859   213.90692

```

The grid search takes a long time to run because it searches over a space of 110 (11 by 10) possible combinations of epsilon and C. To improve computational time, a localized approach to the grid search can be taken to reduce the time required. Also, the evaluation is not much of an improvement either. The SMAPE score decreases, which indicates better forecasting accuracy, but the RMSE and MAPRE increased more than twice as compared to the untuned model. This was thought to be due to grid search's inherent random sampling when carrying out 10-fold cross-validation, which is unsuitable for time series or time-dependent data in chronological order.

The next SVR model will use the linear kernel instead of the default radial one.

```

svr_linear <- svm(formula = func, data = train, scale = FALSE, kernel = "linear")
summary(svr_linear)

##
## Call:
## svm(formula = func, data = train, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##   SVM-Type:  eps-regression
##   SVM-Kernel: linear
##   cost:      1
##   gamma:    0.1428571
##   epsilon:   0.1
##
##
## Number of Support Vectors:  4894

e_x_hat_svr2 <- predict(svr_linear, test)
svr_linear_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr2)
svr_linear_eval

##          RMSE      SMAPE      MAPRE
## 6965.13117  81.39365  100.10938

```

These results are a slight improvement over the radial kernel model. This time, the model tuning will also consider the kernel hyperparameter in addition to `epsilon` and `cost`.

```

"
#Tuning the hyperparameters epsilon and C

set.seed(123)
svr_tuned <- tune(method = svm, train.x = func, data = train,
                    ranges = list(epsilon = seq(0,1,0.5),
                                  cost = 2^(seq(0,10,5)),
                                  kernel = c('radial', 'linear', 'sigmoid')))
#plot(svr_tuned)
summary(svr_tuned)

opt_model = svr_tuned$best.model
summary(opt_model)

e_x_hat_svr1 <- predict(opt_model, test)
svr_tuned_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr1)
svr_default_eval #cost=512, gamma=0.25, epsilon=0

# Retune more finely
svr_retuned <- tune(method = svm, train.x = func, data = train, scale = FALSE,
                     ranges = list(epsilon = seq(0,0.5,0.25),
                                   cost = 2^(seq(5,10,2.5)),
                                   kernel = 'radial'))
summary(svr_retuned)

```

```

opt_model = svr_retuned$best.model
summary(opt_model)

e_x_hat_svr1 <- predict(opt_model, test)
svr_retuned_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr1)
svr_retuned_eval
"

## [1] "\n#Tuning the hyperparameters epsilon and C\n\nset.seed(123)\nnsvr_tuned <- tune(method = svm, t

RMSE and MAPRE improved but this came at the cost of a higher SMAPE. Finally, a sigmoid kernel SVR
model was fitted without tuning.

svr_sigmoid <- svm(formula = func, data = train, scale = FALSE, kernel = "sigmoid")
summary(svr_sigmoid)

##
## Call:
## svm(formula = func, data = train, kernel = "sigmoid", scale = FALSE)
##
##
## Parameters:
##   SVM-Type:  eps-regression
##   SVM-Kernel:  sigmoid
##     cost:  1
##     gamma:  0.1428571
##    coef.0:  0
##    epsilon:  0.1
##
##
## Number of Support Vectors:  4894

e_x_hat_svr3 <- predict(svr_sigmoid, test)
svr_sigmoid_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_svr3)
svr_sigmoid_eval

##      RMSE      SMAPE      MAPRE
## 6999.82413  92.25037  99.48947

```

The untuned sigmoid SVR model performed slightly worse than the untuned linear SVR model across all metrics. Hyperparameter tuning improved the SMAPE metric but worsened the other metrics. Overall, the SVR models, even after tuning the hyperparameters, did not perform as well as the non-linear fit, most probably due to random sampling during cross-validation.

9. Random Forest Regression

The third and final model is the Random Forest Regression (RFR) model. Usually, decision trees and random forests are used in classification tasks, but they can be extended to regression problems as well. The libraries required for this section are `randomForest` and `caret`. Moreover, the user-created functions `mapre` and `evaluate_model` in Section: Non-Linear Regression need to be activated. Since scaling is not necessary for random forests, the unscaled datasets were used.

```

# install.packages("randomForest")
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##   combine

## The following object is masked from 'package:ggplot2':
##   margin

library(caret)
library(Metrics)

train <- read.csv("sat_372_train.csv")
test  <- read.csv("sat_372_test.csv")

```

The random forest model requires a formula (given below) `formula`, the number of trees `ntree`, and the number of variables that are randomly selected to be partitioned at each split in a tree `mtry`. The default values of `ntree = 500` trees and (7 variables/3) `mtry = 2` variables at each split were used initially.

```

func <- e_x ~ abs_time + x_sim + y_sim + z_sim + Vx_sim + Vy_sim + Vz_sim

set.seed(123)
rf_default <- randomForest(formula = func, data = train)
print(rf_default)

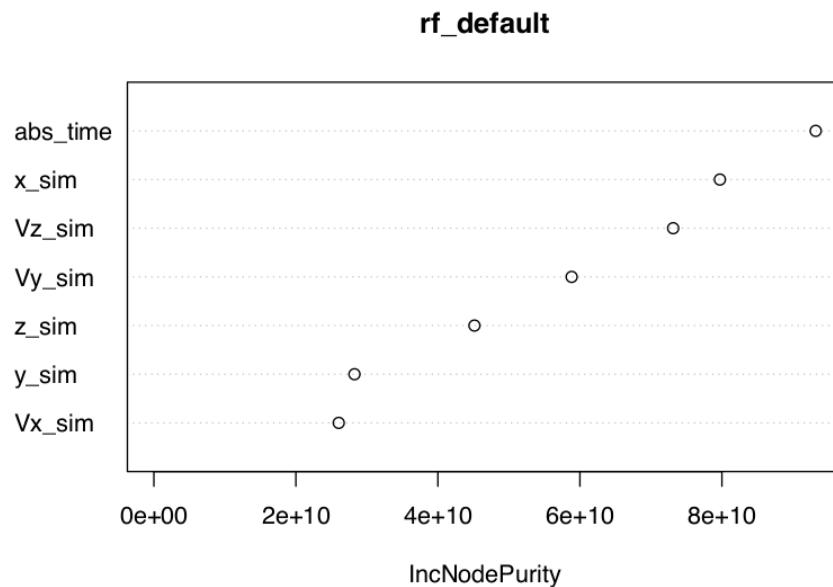
##
## Call:
##   randomForest(formula = func, data = train)
##   Type of random forest: regression
##   Number of trees: 500
##   No. of variables tried at each split: 2
##   Mean of squared residuals: 33221.23
##   % Var explained: 99.96

attributes(rf_default)

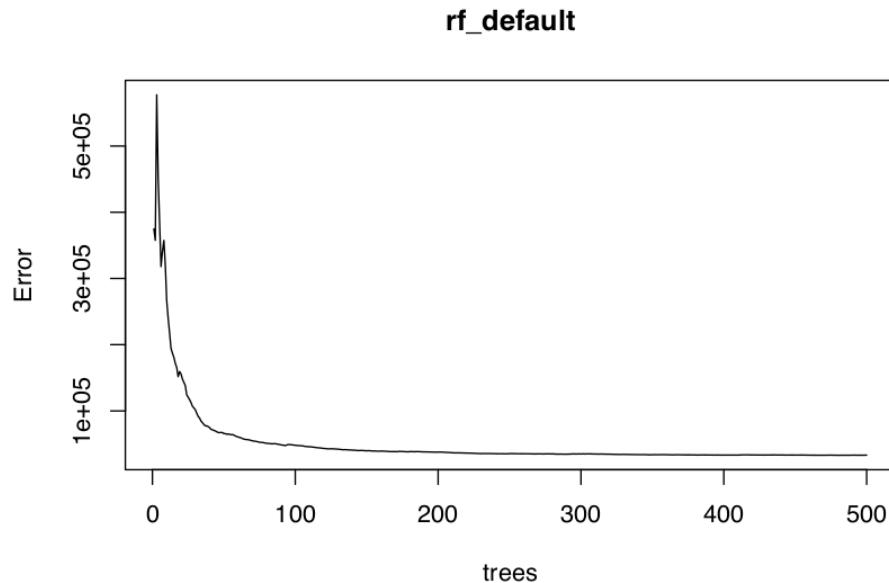
## $names
## [1] "call"          "type"          "predicted"      "mse"
## [5] "rsq"           "oob.times"     "importance"    "importanceSD"
## [9] "localImportance" "proximity"     "ntree"         "mtry"

```

```
## [13] "forest"          "coefs"           "y"                  "test"  
## [17] "inbag"            "terms"  
##  
## $class  
## [1] "randomForest.formula" "randomForest"  
  
varImpPlot(rf_default)
```



```
plot(rf_default)
```



An noteworthy observation from the Variable Importance Plot is that the variable `abs_time` was seen to be most important out of the 7 variables considered, with `Vz_sim` and `x_sim` having similar importance. Moreover, the model performance levels off after around 100 to 200 trees. After fitting the model, predictions were made.

```
e_x_hat_rf1 <- predict(rf_default, test)
rf_default_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_rf1)
rf_default_eval

##      RMSE      SMAPE      MAPRE
## 9277.1817  57.7785 143.4618
```

The results display a similar pattern to the tuned SVR models, with a lower SMAPE score but higher RMSE and MAPRE.

Now the most optimal parameters of the regression model will be found through grid search tuning. Tuning the random forest involves tuning the number of trees `ntree`, the number of variable splits `mtry`, and the maximum number of nodes `maxnodes`.

```
""
# This code finds the best value of `mtry`.
set.seed(123)
tuneGrid <- expand.grid(.mtry = c(1: 7))
trControl <- trainControl(method = 'timeslice',
                           number = 5,
                           search = 'grid',
```

```

initialWindow = 1000,
horizon = 200,
fixedWindow = TRUE)
rf_mtry <- train(func,
  data = train,
  method = 'rf',
  metric = 'RMSE',
  tuneGrid = tuneGrid,
  trControl = trControl,
  importance = TRUE,
  nodesize = 10,
  ntree = 250)
print(rf_mtry) # 7 is the best
best_mtry <- rf_mtry$bestTune$mtry
best_mtry
""

## [1] "\n# This code finds the best value of `mtry`.\nset.seed(123)\ntuneGrid <- expand.grid(.mtry = c

```

The best value of `mtry` was found to be 7. The next grid search looks at the best value of `maxnodes`.

```

"#
# This code finds the best value of `maxnodes`.
set.seed(123)
tuneGrid <- expand.grid(.mtry = 7)
trControl <- trainControl(method = 'timeslice',
  number = 5,
  search = 'grid',
  initialWindow = 1200,
  horizon = 1000,
  fixedWindow = TRUE)

store_maxnode <- list()
tuneGrid <- expand.grid(.mtry = 7)
for (maxnodes in c(5: 15)) {
  set.seed(123)
  rf_maxnode <- train(func,
    data = train,
    method = 'rf',
    metric = 'RMSE',
    tuneGrid = tuneGrid,
    trControl = trControl,
    importance = TRUE,
    nodesize = 10,
    maxnodes = maxnodes,
    ntree = 250)
  current_iteration <- toString(maxnodes)
  store_maxnode[[current_iteration]] <- rf_maxnode
}
results_maxnode <- resamples(store_maxnode)
summary(results_maxnode) # 15 is the best
""

## [1] "\n# This code finds the best value of `maxnodes`.\nset.seed(123)\ntuneGrid <- expand.grid(.mtry =

```

Using 15 as the best value for `maxnodes` and 7 for `mtry`, the following code finds the best value for the number of trees `ntree`.

```
""
# This code finds the best value of `ntree` .
set.seed(123)
tuneGrid <- expand.grid(.mtry = 7)
trControl <- trainControl(method = 'timeslice',
                           number = 5,
                           search = 'grid',
                           initialWindow = 1200,
                           horizon = 1000,
                           fixedWindow = TRUE)

store_maxtrees <- list()
for (ntree in c(50, 100, 150, 200, 250, 300, 350, 400, 500)) {
  set.seed(123)
  rf_maxtrees <- train(func,
                        data = train,
                        method = 'rf',
                        metric = 'RMSE',
                        tuneGrid = tuneGrid,
                        trControl = trControl,
                        importance = TRUE,
                        nodesize = 10,
                        maxnodes = 15,
                        ntree = ntree)
  key <- toString(ntree)
  store_maxtrees[[key]] <- rf_maxtrees
}
results_tree <- resamples(store_maxtrees)
summary(results_tree) # choose 200 trees even though 500 is the best
""

## [1] "\n# This code finds the best value of `ntree` .\nset.seed(123)\ntuneGrid <- expand.grid(.mtry = "
```

The results indicate very little improvement in error when using 500 trees over 50 trees. Hence, to be on the safe side, use 200 trees as the best `ntree` value. In summary, `mtry = 7`, `maxnodes = 15` and `ntree = 200` will be used to build the optimal random forest regression model.

```
func <- e_x ~ abs_time + x_sim + y_sim + z_sim + Vx_sim + Vy_sim + Vz_sim

set.seed(123)
rf_optimal <- randomForest(formula = func, data = train, ntree = 200, mtry = 7, nodesize = 15)
print(rf_optimal)

##
## Call:
##   randomForest(formula = func, data = train, ntree = 200, mtry = 7,      nodesize = 15)
##   Type of random forest: regression
##   Number of trees: 200
##   No. of variables tried at each split: 7
##
```

```

##      Mean of squared residuals: 75088.15
##          % Var explained: 99.91

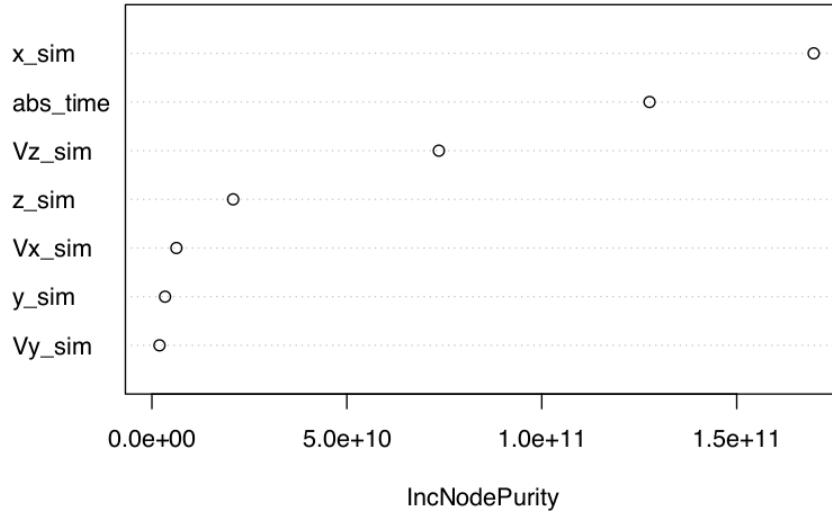
attributes(rf_optimal)

## $names
## [1] "call"           "type"          "predicted"      "mse"
## [5] "rsq"            "oob.times"     "importance"    "importanceSD"
## [9] "localImportance" "proximity"     "ntree"         "mtry"
## [13] "forest"         "coefs"         "y"              "test"
## [17] "inbag"          "terms"
##
## $class
## [1] "randomForest.formula" "randomForest"

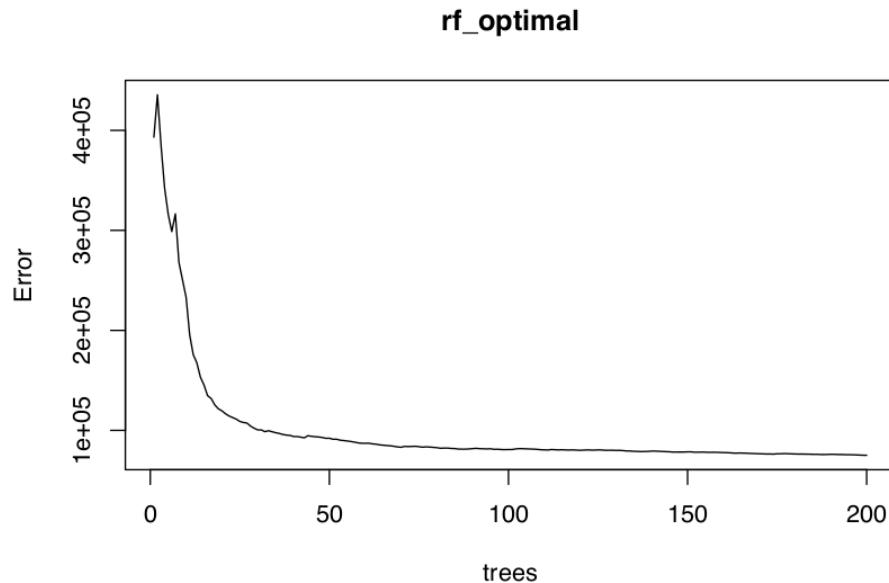
varImpPlot(rf_optimal)

```

rf_optimal



```
plot(rf_optimal)
```



In the optimal random forest model, the most important variable was found to be `x_sim`, followed by `abs_time`, while the last three variables `Vx_sim`, `y_sim` and `Vy_sim` were seen to be of low (and roughly similar) importance.

```
e_x_hat_rf_opt <- predict(rf_optimal, test)
rf_optimal_eval <- evaluate_model(true = test$e_x, estimated = e_x_hat_rf_opt)
rf_optimal_eval

##          RMSE      SMAPE      MAPRE
## 8902.21409  56.95911 136.05419
```

Evaluating the optimized model results in better performance, since all three metrics are lower than the un-optimized model.

10. Analysis and Recommendations

So far, all the models that were built were trained on the Train dataset and the evaluated results were obtained by testing the models on the Test dataset. This misses out a crucial aspect of machine learning, namely, validation of the models. Validation can be done through comparing the performance metrics across both Train and Test datasets, in order to determine whether overfitting or underfitting occurs.

Here, a concise tabular summary of the model performances across the three metrics, tested only on Test data, was constructed as shown below:

```

# the evaluation vector of the tuned linear SVR model was not saved,
# so recreate it here
svr_default_retuned_eval <- c(11310.53790, 69.57217, 187.43788)
names(svr_default_retuned_eval) <- c("RMSE", "SMAPE", "MAPRE")

model_names <- c("Non-Linear Regression", "SVR Radial", "SVR Radial Tuned",
                 "SVR Radial Retuned", "SVR Linear", "SVR Sigmoid",
                 "Random Forest", "Random Forest Tuned")

model_performances <- data.frame(nls_eval, svr_default_eval, svr_default_tuned_eval,
                                   svr_default_retuned_eval, svr_linear_eval,
                                   svr_sigmoid_eval, rf_default_eval, rf_optimal_eval)

colnames(model_performances) <- model_names

# take transpose of the table
model_performances <- t(model_performances)
model_performances

##          RMSE    SMAPE    MAPRE
## Non-Linear Regression 1699.585 20.05577 20.64286
## SVR Radial           6982.307 89.79596 99.30569
## SVR Radial Tuned     13007.872 69.08859 213.90692
## SVR Radial Retuned   11310.538 69.57217 187.43788
## SVR Linear            6965.131 81.39365 100.10938
## SVR Sigmoid           6999.824 92.25037 99.48947
## Random Forest         9277.182 57.77850 143.46176
## Random Forest Tuned  8902.214 56.95911 136.05419

```

Note that these results were obtained only for the `x` coordinate. Similar steps to model building may be done for the other five kinematic states. After putting the Test results in tabular form above, a simple validation of each model can be done by testing the machine learning models on the Train dataset, and then comparing the difference in performance.

```

# the evaluation vectors will have the subscript `v` when tested on Train data

# non-linear regression model on Train data
x_hat_nls_v <- predict_state(state = "x", parameters = param, data = train)
nls_eval_v = evaluate_model(true = train$x, estimated = x_hat_nls_v)

# SVR radial model on Train data
e_x_hat_svr1_v <- predict(svr_default, train)
svr_default_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_svr1_v)

# SVR radial tuned model on Train data
e_x_hat_svr1_tuned_v <- predict(svr_default_tuned, train)
svr_default_tuned_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_svr1_tuned_v)

# Unfortunately, the grid search results of the retuned/fine-tuned SVR radial model was not saved.
# Also, running the grid search took a long time, so the retuned model will be skipped

# SVR linear model on Train data
e_x_hat_svr2_v <- predict(svr_linear, train)

```

```

svr_linear_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_svr2_v)

# SVR sigmoid model on Train data
e_x_hat_svr3_v <- predict(svr_sigmoid, train)
svr_sigmoid_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_svr3_v)

# Random forest model on Train data
e_x_hat_rf1_v <- predict(rf_default, train)
rf_default_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_rf1_v)

# Random forest tuned model on Train data
e_x_hat_rf_opt_v <- predict(rf_optimal, train)
rf_optimal_eval_v <- evaluate_model(true = train$e_x, estimated = e_x_hat_rf_opt_v)

```

A similar table for the model performances on Train data was then constructed below:

```

model_names_v <- model_names <- c("Non-Linear Regression", "SVR Radial", "SVR Radial Tuned",
                                    "SVR Linear", "SVR Sigmoid",
                                    "Random Forest", "Random Forest Tuned")

model_performances_v <- data.frame(nls_eval_v, svr_default_eval_v,
                                     svr_default_tuned_eval_v, svr_linear_eval_v,
                                     svr_sigmoid_eval_v, rf_default_eval_v, rf_optimal_eval_v)

colnames(model_performances_v) <- model_names_v

model_performances_v <- t(round(model_performances_v, 3))
model_performances_v

##          RMSE   SMAPE   MAPRE
## Non-Linear Regression 6822.119 43.781 73.554
## SVR Radial            9096.560 91.975 99.986
## SVR Radial Tuned     9130.702 79.291 101.923
## SVR Linear             73243133.412 99.989 914069.164
## SVR Sigmoid            9106.748 95.027 100.263
## Random Forest          87.408  1.904  0.585
## Random Forest Tuned    180.977  2.967  1.444

```

A comparison of the two tables can be made by calculating the percentage difference in error between them, defined in the code chunk below. If the percentage difference is positive, then the model has been overfitted because the error in the Test dataset is higher than the error in the Train dataset, and vice versa.

```

test_eval <- model_performances[-4,]
train_eval <- model_performances_v

# calculate percentage difference in error
perc_diff <- round((test_eval - train_eval) * 100 / train_eval, 2)
perc_diff

##          RMSE   SMAPE   MAPRE
## Non-Linear Regression -75.09 -54.19 -71.94
## SVR Radial           -23.24 -2.37 -0.68

```

```

## SVR Radial Tuned      42.46  -12.87   109.87
## SVR Linear           -99.99  -18.60   -99.99
## SVR Sigmoid          -23.14   -2.92    -0.77
## Random Forest        10513.65 2934.59 24423.38
## Random Forest Tuned  4818.98 1819.75  9322.04

```

negative means underfit, positive means overfit

Of course, this is a naive way of validation, since it does not take multiple training epochs into account and simply looks at one particular instance of a model at one time. However, it does give a general ballpark indicator of how serious the overfit/underfit is. For instance, the SVR Linear kernel model indicates a severe case of underfitting while the Tuned Random Forest model indicates the opposite. The models that appear to be generalizable across both datasets are the SVR Radial and SVR Sigmoid models. In addition, the tuned models seem to overfit much more than the corresponding un-tuned models. The underfit in the non-linear regression model is due to some quirk of the `nls` function not being able to fit on large amounts of data, but fits very well on about 1000 observations. Finally, the metric that reacts the least to overfitting/underfitting is the SMAPE metric, due to its low percentage difference. This could be due to SMAPE being a specialized metric for time-dependent data.

From this analysis, the best model was chosen to be the non-linear regression model due to its simplicity and interpretability as well as its performance on the Test dataset. Due to the time-dependent nature of the datasets, the best metric to measure performance would be the SMAPE metric.

11. Conclusion

To recap, this study looked at data on satellite kinematics and used three machine learning models to predict the x -coordinates of one satellite. This satellite (Satellite 372) was chosen on the basis of having the highest number of observations. Based on the periodic pattern observed in the coordinate data, the first model proposed was a non-linear sine curve regression with an additive linear trend. The second and third models were Support Vector Regression and Random Forest Regression. The Non-Linear Regression model was observed to have the best performance out of the three, most probably due to fitting the correct periodic function. The other two models performed poorly. This was thought to be due to random sampling, which is not appropriate for data arranged in chronological order.

There are several improvements and extensions that could be made to this study. Firstly, only a small subset of the given dataset was used. Out of 600 satellites, only one was studied. Moreover, out of the six kinematic states, only the x -coordinate (and its error) were predicted. Also, the true states for the Test dataset were not provided to competitors, so the evaluation of the proposed models was restricted to the Train dataset. Thus, this study effectively looked at only 1/3600 of the entire dataset.

Secondly, an in-depth exploration and discussion on the other 5 kinematic states could be done. These states were explored but were not included and discussed within the study. Also, the validation of the proposed models were unsatisfactory at best, so better validation techniques that are different from randomly sampled cross-validation (and especially designed for time-dependent data), could be studied and applied instead.

Thirdly, only three machine learning models were proposed in this study. Gaussian process regression and the multilayer perceptron could be explored as an extension. In addition, classical forecasting techniques such as Seasonal Holt-Winters and Seasonal ARIMA models could be quite ideal in forecasting/predicting satellite coordinates, even if they are not machine learning models.

Acknowledgements

The author thanks the Russian Astronomical Science Center (and the hosts, partners and organizers of the International Data Analytics Olympiad) for providing the data sets used in this study.

References

- Peng, H. and Bai, X. (2018). Improving orbit prediction accuracy through supervised machine learning. *Advances in Space Research*. 61 (10). p.pp. 2628–2646.
- Peng, H. and Bai, X. (2019). Machine Learning Approach to Improve Satellite Orbit Prediction Accuracy Using Publicly Available Data. *Journal of the Astronautical Sciences*.
- Hyndman, R.J., & Athanasopoulos, G. (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. [Online] Available at otexts.com/fpp2/. [Accessed 3 March 2019].

Predicting Satellite Coordinates Using Machine Learning Models

GRADEMARK REPORT

FINAL GRADE

47 /100

GENERAL COMMENTS

Instructor

Brave attempt to try an uncommon domain which requires much time to understand the domain and data preprocessing.

Systematic approach to the problem.

Very good analysis.

As rightfully pointed out success of non linear regression model may be attributed to the periodic nature of the data and to correct choosing a nonlinear periodic function to estimate it.

As for over or underfit, it is a good observation. However, this may not be a metric to compare performance. This observation is useful to identify the phenomena and use several of the ways to remedy it.

Appreciate several insights presented. An example is the data leakage problem while referring to min max normalisation.

Overall, very good work and clear presentation.

Keep up the good learning approach to reach greater heights.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

METHODS (5%)

79 / 100

Methods (5%)

| | |
|----------------------|---|
| DISTINCTION (100) | Highly appropriate methods are selected and clearly articulated with proper justification. |
| DISCTINCTION (79) | Methods are well argued and justified. |
| MERIT (69) | Methods are explained and appropriate for the project. |
| PASS (59) | Appropriate methods are broadly outlined, but details are not clear. |
| FAIL (49) | The methods are either not appropriate for the project or is poorly articulated suggesting deficits in understanding. |
| FAIL (39) | Missing methods or not relevant information is given. |

DATASET PREP (5%)

79 / 100

Dataset Preparation (5%)

| | |
|----------------------|---|
| DISTINCTION (100) | A very good dataset preparation has been showcased and reported appropriately and fully with no deficiencies. |
| DISCTINCTION (79) | A good dataset preparation has been showcased and reported appropriately with very minor deficiencies. |
| MERIT (69) | A reasonable effort shown in the dataset preparation with all required sections with minor deficiencies. |
| PASS (59) | A weak effort shown in the dataset preparation where some required sections are reported with minor deficiencies. |
| FAIL (49) | A poor effort shown in the dataset preparation where very few required sections are reported with major deficiencies. |
| FAIL (39) | There is no or very poor evidence for dataset preparation found in the report. |

MODEL IMP (20%)

79 / 100

Model Implementation (20%)

| | |
|----------------------|---|
| DISTINCTION (100) | Highly appropriate and suitable ML algorithms are used to build the models and the outputs are clearly explained. |
| DISCTINCTION (79) | ML algorithms are well selected, models are built along with the outputs are clearly explained. |
| MERIT (69) | The model building is done using the ML algorithm and outputs are explained. Most components are clear and can be understood. |
| PASS (59) | The model building is satisfactory with the reasonable ML algorithm and with output explanation. Some details are not clear. |
| FAIL | The model building is either not appropriate for the intended problem or is poorly |

| | |
|--------------|---|
| (49) | implemented suggesting deficits in understanding. |
| FAIL (39) | Missing model building part or irrelevant application of the ML algorithms with mismatching explanation of the model outputs. |

| | |
|------------------------|----------|
| MODEL VALID (10%) | 79 / 100 |
| Model Validation (10%) | |

| | |
|----------------------|--|
| DISTINCTION (100) | Highly appropriate and suitable model validation and evaluation of the model results and interpretation its significance. |
| DISCTINCTION (79) | Model validations are well done, model results are evaluated and interpreted with its significance. |
| MERIT (69) | The model validations are done and outputs are evaluated with its significance. Most components are clear and can be understood. |
| PASS (59) | The model validations are satisfactorily done and outputs are evaluated with its significance. Some details are not clear. |
| FAIL (49) | The model validations are either not appropriate for the intended problem or is poorly done suggesting deficits in understanding. |
| FAIL (39) | Missing model validation part or irrelevant application of the model validations with mismatching evaluation of the model outputs. |

| | |
|----------------------------------|----------|
| ANALYSIS (15%) | 79 / 100 |
| Analysis & Recommendations (15%) | |

| | |
|----------------------|---|
| DISTINCTION (100) | Outstanding evidence of results, critical analysis and discussion/recommendations. |
| DISCTINCTION (79) | Very good results, critical analysis and discussion/ recommendations are provided. |
| MERIT (69) | Appropriate analysis and discussion/ recommendations of results are provided, but still need improvement. |
| PASS (59) | Satisfactory results with discussion/ recommendations, but no critical analyses are provided. |
| FAIL (49) | There are some discussion/ recommendations on the results, but it is not satisfactory. |
| FAIL (39) | Little or no evidence of analysis or discussion/ recommendations. |

| | |
|-----------------|----------|
| CONCLUSION (5%) | 79 / 100 |
| Conclusion (5%) | |

| | |
|----------------------|--|
| DISTINCTION (100) | The conclusion is perfect, and address all the objectives that are fully met, with outstanding implications. |
| DISCTINCTION (79) | The conclusion is very good, and shows objectives are met. There are good implications. |
| MERIT (69) | Good conclusion which shows the fulfilment of objectives, but still could be improved. |
| PASS | The conclusion is satisfactory, but does not address fully achievement of objectives. |

(59)

FAIL
(49) There are some conclusion, but not very relevant. It does not show the fulfilment of the objectives.

FAIL
(39) There is no or very little evidence of conclusion.