MTH6139 Time Series

Coursework 1

true

March 2024

Contents

1. The project and dataset

1.1 What is the project about?

The goal of this project is to forecast UK spending for the upcoming year using the Prophet algorithm. This will be accomplished by decomposing the seasonal components of UK spending and illustrating the trend. In addition, we will delve deeper into the Prophet library by demonstrating how to apply the 'Holiday' filter to a dataset.

1.2 Getting to know the dataset we work with

Finding the appropriate data is the first and most important step to make. The data I chose to work with is daily non-seasonally adjusted UK spending using debit and credit cards between 2020 and 2024. Source:

https://www.ons.gov.uk/economy/economicoutputandproductivity/output/datasets/ukspendingoncreditanddebitcards

Firstly, let's examine the data

```
#loading the dataset
uk_spending=read.csv('uk_spending_dataset.csv')
head(uk_spending)
```

```
##            Date.Aggregate
## 1 13 Jan 2020;93.97322222
## 2 14 Jan 2020;92.79815004
## 3 15 Jan 2020;92.67428904
## 4 16 Jan 2020;93.02360652
## 5 17 Jan 2020;93.13415493
## 6 20 Jan 2020;94.80196668
```

```r
str(uk_spending)
```

```
## 'data.frame':    1049 obs. of  1 variable:
##  $ Date.Aggregate: chr  "13 Jan 2020;93.97322222" "14 Jan 2020;92.79815004" "15 Jan 2020;92.67428904
```

The original dataframe has a combined variable called Date.Aggregate and contains 1049 observations. The date represents when transactions occurred, while the aggregate column shows the average index of transaction magnitude across different spending categories.

1.3 Reformating the dataset

For compatibility with the Prophet functions, we need to reformat the dataframe. It should have two columns named 'ds' and 'y', which contain the date and numeric value respectively. The ds column should follow the format YYYY-MM-DD for a date, or YYYY-MM-DD HH:MM:SS for a timestamp.

```r
# splitting the combined column into two columns using the strsplit function
uk_spending_split=strsplit(uk_spending$Date.Aggregate, ";")

# creating two new columns using the sapply function
uk_spending$ds=sapply(uk_spending_split, function(x) x[1])
uk_spending$y=sapply(uk_spending_split, function(x) x[2])

# converting columns into the right datatype and droping the combined column
uk_spending$ds=as.Date(uk_spending$ds, format = "%d %b %Y")
uk_spending$y=as.numeric(uk_spending$y)
uk_spending$Date.Aggregate=NULL


str(uk_spending)
```

```
## 'data.frame':    1049 obs. of  2 variables:
##  $ ds: Date, format: "2020-01-13" "2020-01-14" ...
##  $ y : num  94 92.8 92.7 93 93.1 ...
```

```r
head(uk_spending)
```

```
##           ds        y
## 1 2020-01-13 93.97322
## 2 2020-01-14 92.79815
## 3 2020-01-15 92.67429
## 4 2020-01-16 93.02361
## 5 2020-01-17 93.13415
## 6 2020-01-20 94.80197
```

Remark: You could notice that our data misses a few datapoints. This prevents us from using the astsa library, as it requires a time series object.

However, one significant advantage of the Prophet algorithm is its ability to handle missing data, a common issue in real-world scenarios. Each data point is associated with its actual date value, allowing the algorithm to work effectively even with gaps in the dates.

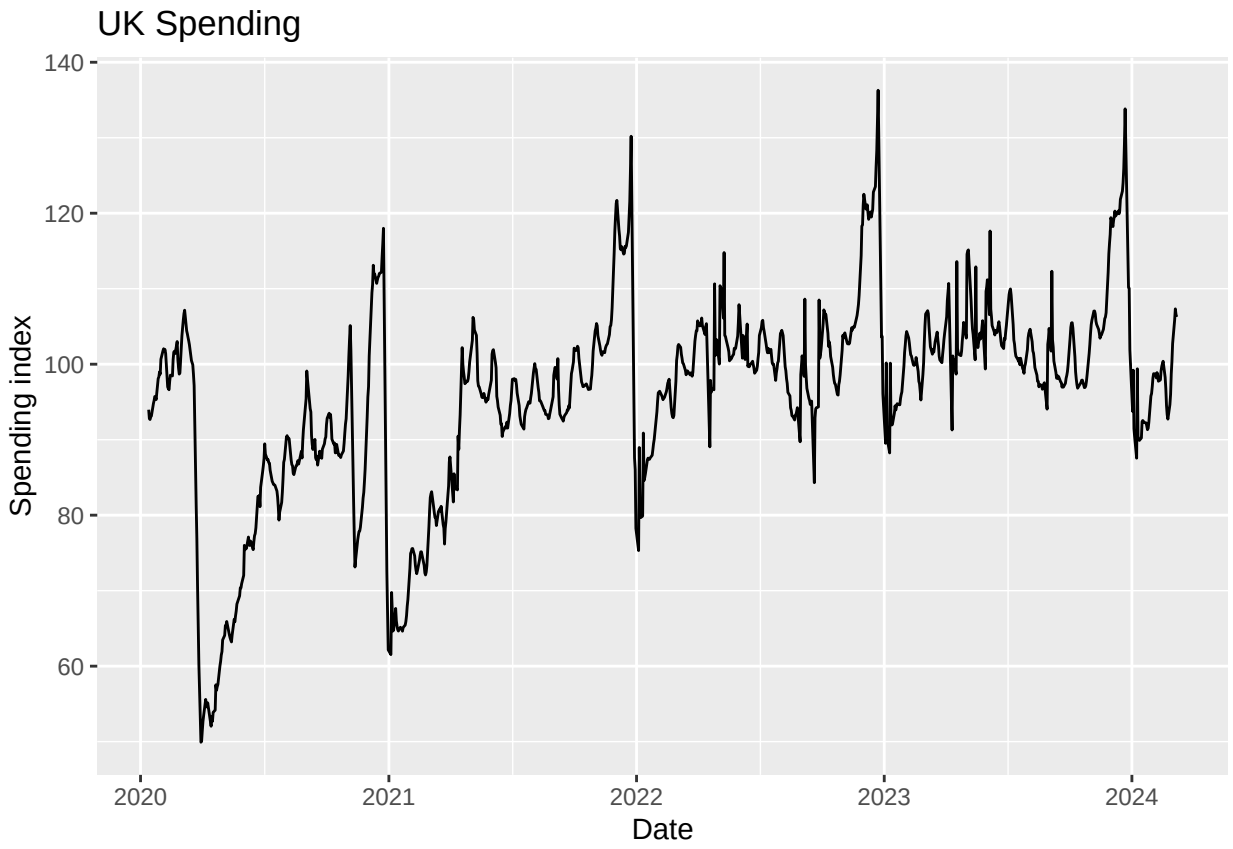
1.4 Data set visualisation

Before proceeding with Prophet, let's gain a better understanding of the dataset by completing simple visualizations using ggplot2.

```r
# loading the library
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
#plotting the dataframe
ggplot(uk_spending, aes(x=ds, y=y)) +
  geom_line() +
  labs(title="UK Spending",
       x="Date",
       y="Spending index")
```



The graph clearly shows both the prominent trend and daily and yearly seasonality. It also appears that the variance increases over time, likely due to inflation. So, presenting the data analysis with a log transformation could be valuable to mitigate this heteroscedastic influence.

2. Using Prophet for forecasting

2.1 Meeting the coursework minimun requirements

In this section, we perform the basic Prophet forecasting.

We use the daily.seasonality = TRUE and yearly.seasonality = TRUE arguments to instruct Prophet to model seasonality on a daily and yearly basis. This is because we expect the data to exhibit varying behaviour throughout the year, while also maintaining similar yearly patterns.

```
#applying prophet modelling
library(prophet)
```

```
## Loading required package: Rcpp
```

```
## Warning: package 'Rcpp' was built under R version 4.2.3
```

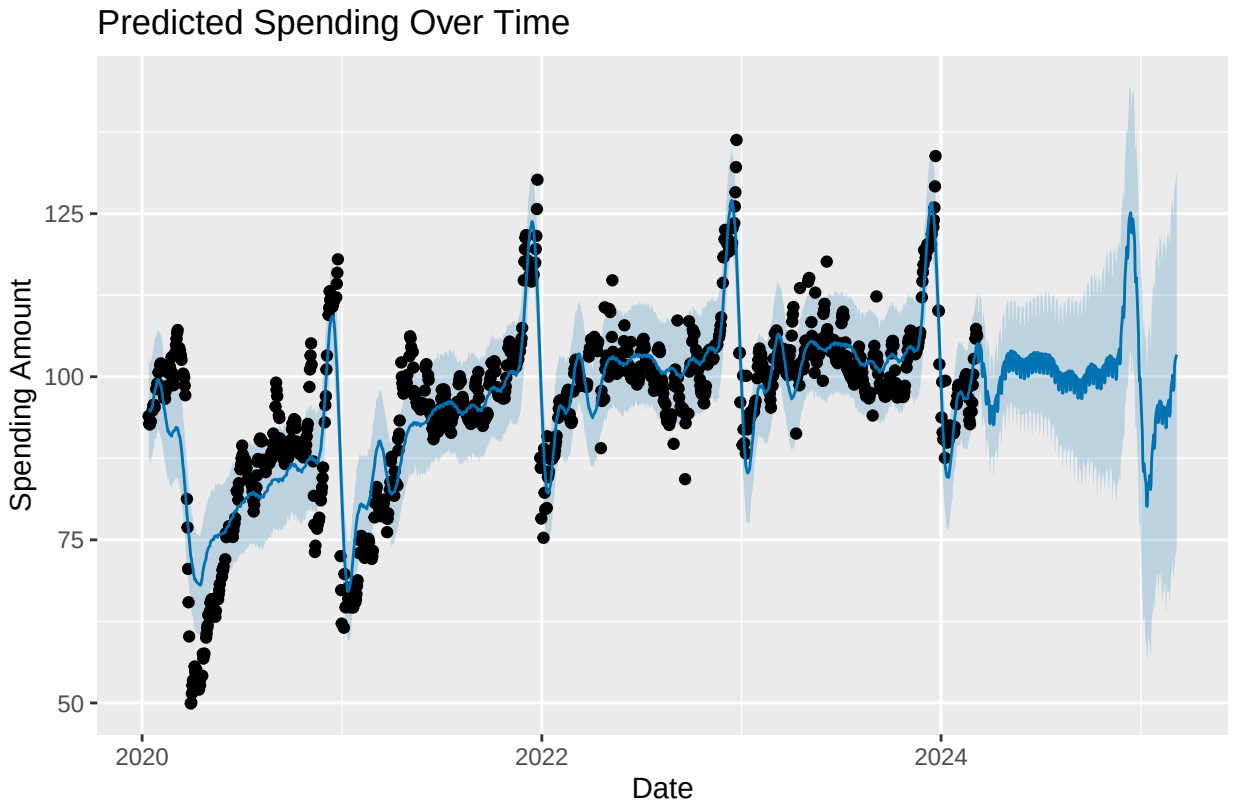
```
## Loading required package: rlang
```

```
## Warning: package 'rlang' was built under R version 4.2.3
```

```
prophet_spending=prophet(uk_spending, daily.seasonality  = TRUE, yearly.seasonality = TRUE)
future_spending=make_future_dataframe(prophet_spending, 365, freq = 'day')
predicted_spending = predict(prophet_spending, future_spending)
```

Here, make_future_dataframe is a function that creates a DataFrame with future dates for our forecast. The frequency is set to 'day' so that to indicate daily predictions.

```
plot(prophet_spending, predicted_spending)+
labs(title = "Predicted Spending Over Time",
     x = "Date",
     y = "Spending Amount")
```

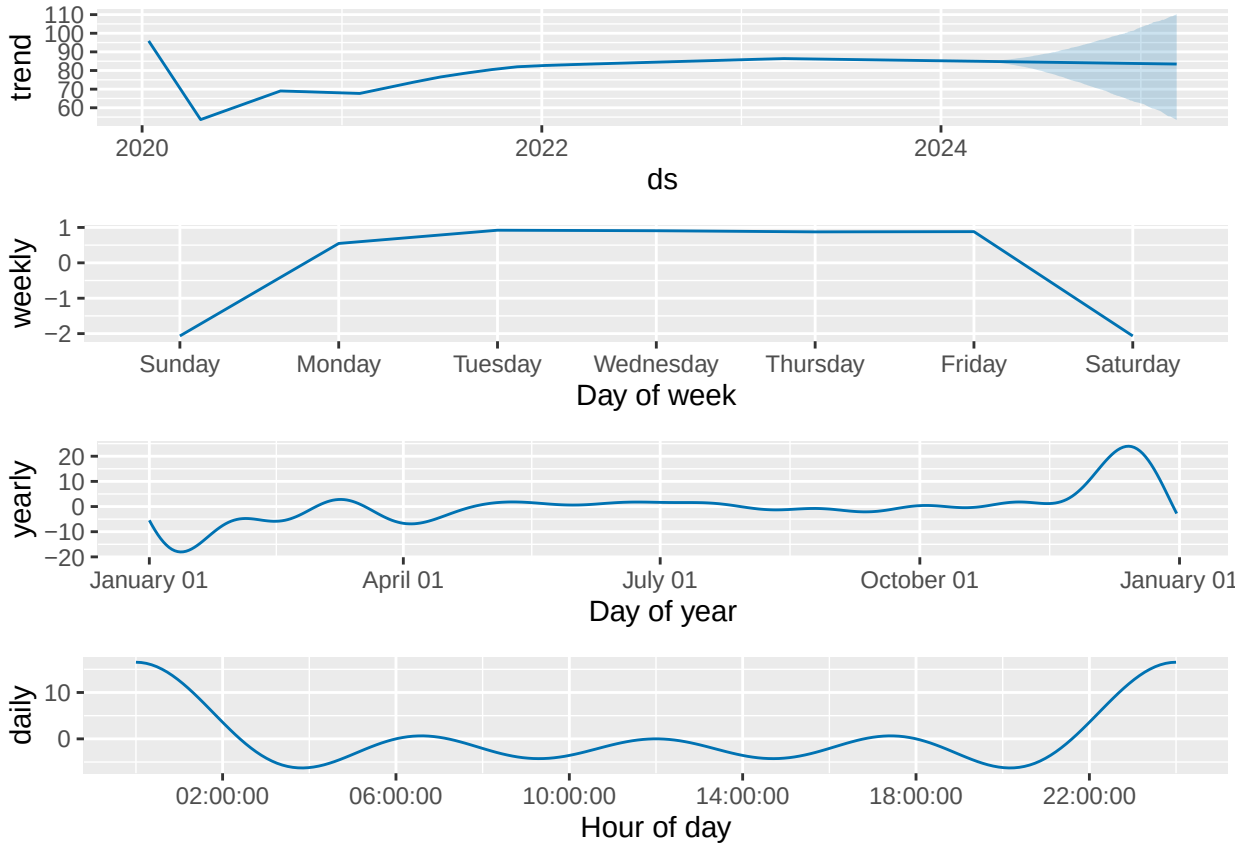


In this graph, black dots represent actual UK spending data points, showing variability in spending across different days. The graph also depicts the model's spending predictions for the following year, based on trend and seasonality components. The shaded area represents the uncertainty intervals of these predictions, which increase as we move further into the future.

Spending appears to fluctuate in a regular pattern, corresponding to yearly cycles. There is also a general upward trend in spending over time. However, the predicted values suggest that spending will not significantly increase in 2024.

We call prophet_plot_components function to see what the model components are.

```
uk_spending_components=prophet_plot_components(prophet_spending, predicted_spending)
```



Here are some observations from the components results:

1. A sharp decline in UK spending is noticeable around 2020, reflecting the impact of Covid on the country. The trend appears to level off in after.
2. The days of the week contribute similarly to the overall trend. However, Saturday and Sunday negatively impact it as banks do not process transactions over the weekend.
3. Day of the year component depict some interesting patterns. Firstly, the spending reaches its peak during the Christmas period. Another expenditure growth is between February and March, which the time of mid season sales. Both of two rises are followed by the decrease in expenses. Both peaks are followed by a decrease in expenditure.
4. The hour of the day component exhibits a significant increase around midnight, which aligns with the time most transactions are processed.

2.2 Log-Prophet Forecasting

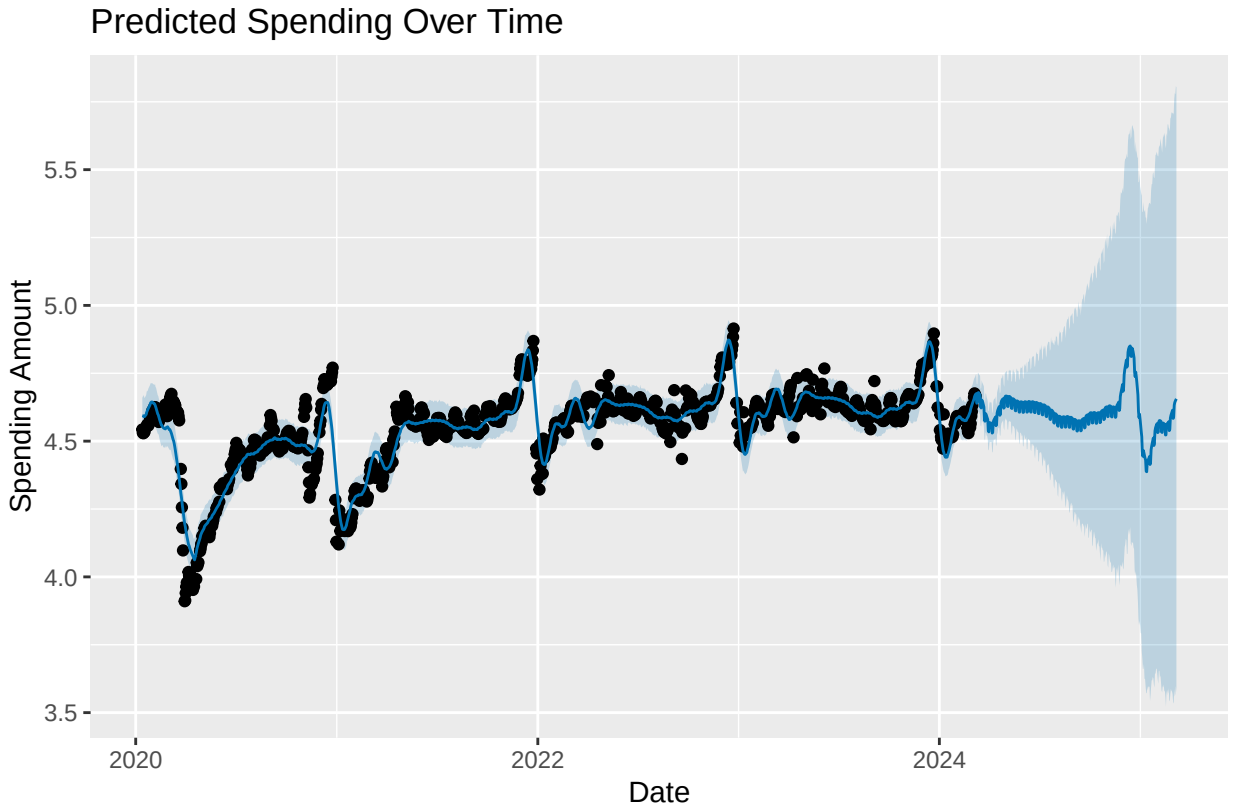
As previously noted, data variance increases over time. In such situations, heteroscedasticity can lead to lower precision, increasing the likelihood that the coefficient estimates deviate from the correct population value.

To address this issue, we can apply a log-transformation to the data and then perform forecasting again

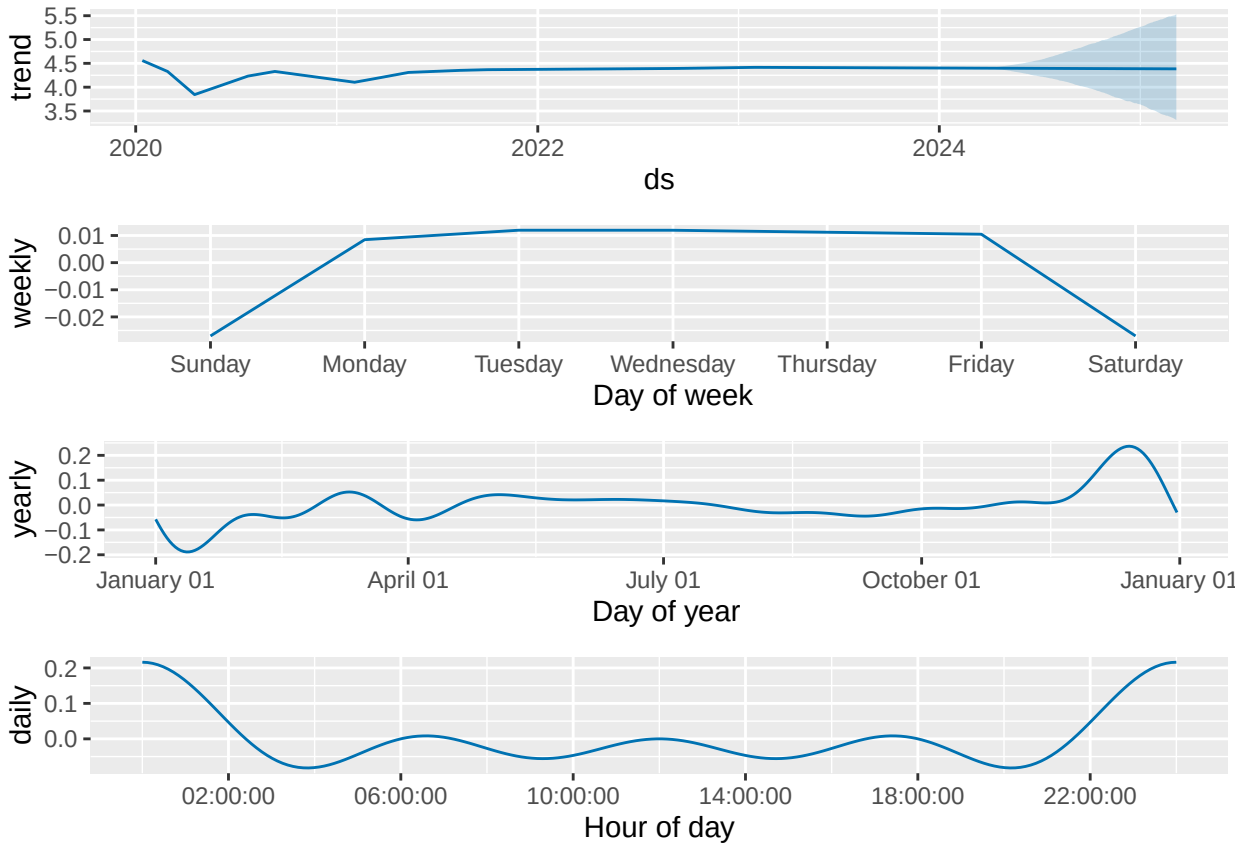
```
# log-transform the 'y' values to stabilise variance
uk_spending$y=log(uk_spending$y)

#follow the forecasting precudure using prophet library as before
log_spending=prophet(uk_spending, daily.seasonality = TRUE, yearly.seasonality = TRUE)
future_log_spending=make_future_dataframe(log_spending, periods = 365)
predicted_log_spending=predict(log_spending, future_log_spending)
```

```r
# plot the forecast with original scale values
plot(log_spending, predicted_log_spending) +
labs(title="Predicted Spending Over Time",
     x="Date",
     y="Spending Amount")
```




```r
log_spending_components=prophet_plot_components(log_spending, predicted_log_spending)
```



From the plot, it's clear that we managed to stabilise the variance using the log-transformation.

Now, our trend component appears smoother and is better adjusted for the initial heteroscedasticity.

2.3 Adding Holiday component

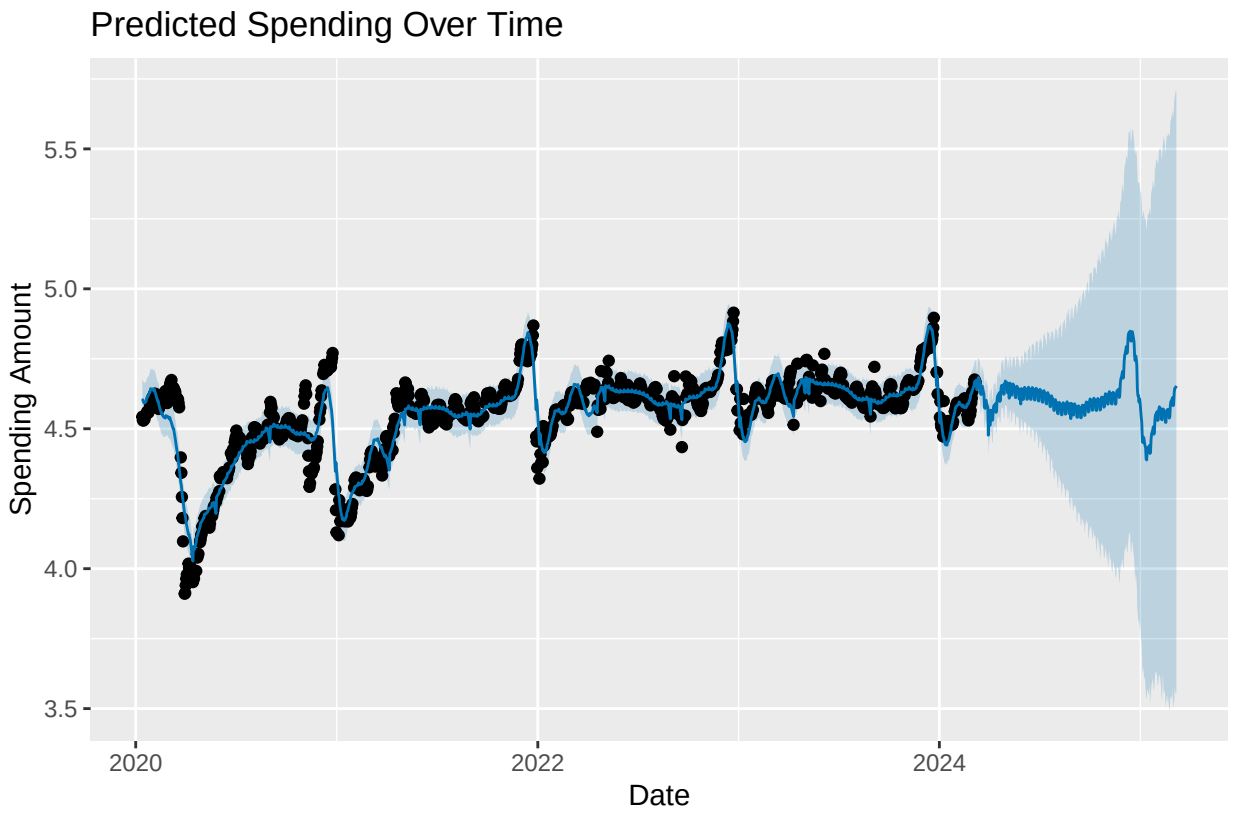
Lastly, we will examine log-transformed data by adding Holiday component as we assume that people spend more during holidays.

```
# define holidays
holidays=data.frame(
  holiday='UKPublicHoliday',
  ds=as.Date(c('2020-01-01', '2020-04-10', '2020-04-13', '2020-05-08',
                '2020-05-25', '2020-08-31', '2020-12-25', '2020-12-26',
                '2020-12-28', '2021-01-01', '2021-04-02', '2021-04-05',
                '2021-05-03', '2021-05-31', '2021-08-30', '2021-12-25',
                '2021-12-26', '2021-12-27', '2021-12-28', '2022-01-01',
                '2022-01-03', '2022-04-15', '2022-04-18', '2022-05-02',
                '2022-06-02', '2022-06-03', '2022-08-29', '2022-09-19',
                '2022-12-25', '2022-12-26', '2022-12-27', '2023-01-02',
                '2023-04-07', '2023-04-10', '2023-05-01', '2023-05-08',
                '2023-08-28', '2023-12-25', '2023-12-26', '2024-01-01',
                '2024-03-29', '2024-04-01', '2024-05-06', '2024-05-27',
                '2024-08-26', '2024-12-25', '2024-12-26'
                )),
  lower_window = 0,
  upper_window = 1
)
```

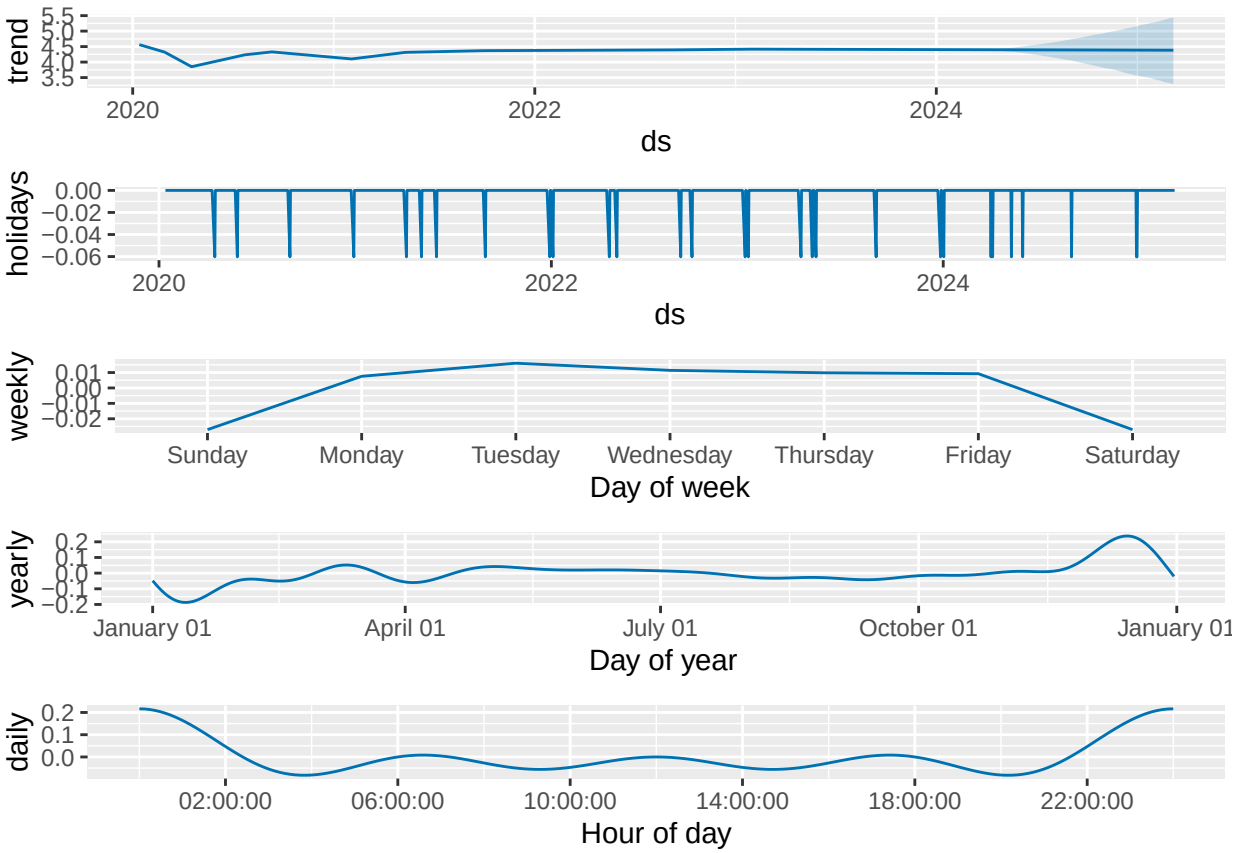
```r
holiday_adjusted_spending=prophet(uk_spending, daily.seasonality = TRUE, yearly.seasonality = TRUE, holi
future_holiday_adjusted_spending=make_future_dataframe(holiday_adjusted_spending, periods = 365)
predicted_holiday_adjusted_spending=predict(holiday_adjusted_spending, future_holiday_adjusted_spending

predicted_holiday_adjusted_spending=predict(holiday_adjusted_spending, future_holiday_adjusted_spending

plot(holiday_adjusted_spending, predicted_holiday_adjusted_spending) +
labs(title="Predicted Spending Over Time",
     x="Date",
     y="Spending Amount")
```




```r
prophet_plot_components(holiday_adjusted_spending, predicted_holiday_adjusted_spending)
```



Here, by adding Holiday component, we are achieving more precise estimation.