Jean An

IPDS Final Project

1) Using Python, download the main table from this page:
https://en.wikipedia.org/wiki/Farebox_recovery_ratio

```
import wikipedia, re
from BeautifulSoup import BeautifulSoup

def get_data():
    wiki_search_string = "Farebox recovery ratio"
    wiki_page_title = "Farebox recovery ratio"
    wiki_table_caption = """Ratio of fares to operating costs for
                            public transport systems"""
    parsed_table_data = []

    search_results = wikipedia.search(wiki_search_string)
    for result in search_results:
        if wiki_page_title in result:
            my_page = wikipedia.page(result)
            soup = BeautifulSoup(my_page.html())
            table = soup.find('caption',text=re.compile
                    (r'%s'%wiki_table_caption)).findParent('table')
            rows = table.findAll('tr')
            for row in rows:
                children = row.findChildren(recursive=False)
                row_text = []
                for child in children:
                    clean_text = child.text
                    clean_text = clean_text.split('&#91;')[0]
                    clean_text = clean_text.split('&#160;')[-1]
                    clean_text = clean_text.strip()
                    row_text.append(clean_text)
                parsed_table_data.append(row_text)
    return parsed_table_data
```

2) Write Python code to "clean up" this dataset.

- All numbers are only numbers
- All currencies are converted to USD
- All fractions as floats
- Make a choice how to represent fares and explain your approach

The following code writes all the data we just downloaded from the Wikipedia table into seven separate lists by column, and cleans up the *Ratio* and *Year* columns while doing so. For all the data in *Year* that included two years (e.g. 2017/18) we took the first year in the pair.

```
continent = []
country = []
system = []
ratio = []
raw_fare_system = []
raw_fare = []
year = []

if __name__=="__main__":
    clean_data = get_data()[1:]
    for row in clean_data:
        continent.append(row[0])
        country.append(row[1])
        system.append(row[2])
        ratio.append(float((row[3].split("%"))[0]))
        raw_fare_system.append(row[4])
        raw_fare.append(row[5])
        year.append(int((row[6].split("/"))[0]))
```

The following code cleans up the *Fare System* column. In the original table, there were seventeen distinct fare systems. However, many of which are actually the same, just written is different formats. For example, the most common fare system was "flat rate," but some cells had added descriptions after the term. There were also multiple entries that essentially all described "zone based," but were simply written differently. Therefore, we decided to separate the system into four groups: *flat rate*, *zone based*, *distance based* and *other*.

```
def CleanSystem(raw_fare_system):
    if "Zone" in raw_fare_system:
        return "Zone based"
    if "Flat" in raw_fare_system:
        return "Flat rate"
    if "Distance" in raw_fare_system:
        return "Distance based"
    else:
        return "Other"

fare_system = []
for x in raw_fare_system:
    fare_system.append(CleanSystem(x))
```

We have now taken all the data in `raw_fare_system`, placed every entry into the four distinct buckets we created, and written it into a new list called `fare_system`.

The last column, and also by far the most complicated, that we have to deal with, is the *Fare Rate* column. There are way too many different types of distinct entries in this list, so it would be difficult to clean it up. After studying all the entries, we identified a few criteria that would allow us to clearly distinguish the value we wanted for each entry. First of all, many entries included the price if you pay with cash and if you pay with some version of a pass. In order to get a standard value, we decided to take all cash values only.

Second of all, many of the entries were written in the format such as *US$2.75+*, which essentially tells us that there is a base fare rate. For these, we would remove the + and treat it as the standard fare rate. Then, this leads us to entries that were written in the format such as *US$2–3*, which gives us a range. Since we took the base rate for the previous type of entry, it would only make sense for us to take the low end on this type of entry. Thus, we would cut off the – and everything that comes afterwards.

There were also two entries that included the fare rate of multiple types of transportation, such as bus and train. For these two entries, we decided to take the first number written in the entry, which is the bus fare. Lastly, there were also some other types of entries that required us to create special criterion to clean it up, such as data that included a comma, a slash line, or the word "to" and "from." The following code cleans up the entire `raw_fare` list.

```
def CleanFare(raw_fare):
    if "+ (Bus)" in raw_fare:
        return raw_fare.split("+")[0]
    if "(Bus)" in raw_fare:
        return raw_fare.split("(Bus)")[0]
    if "+" in raw_fare:
        return raw_fare.split("+")[0]
    if "-" in raw_fare:
        return raw_fare.split("-")[0]
    if "From" in raw_fare:
        return re.split("m| /", raw_fare)[1]
    if "cash" in raw_fare:
        return raw_fare.split(" (")[0]
    if " /" in raw_fare:
        return raw_fare.split(" /")[0]
    if " (" in raw_fare:
        return raw_fare.split(" (")[0]
    if "," in raw_fare:
        return raw_fare.split(",")[0]
    if "to" in raw_fare:
        return raw_fare.split("to")[0]
    else:
        return raw_fare

fare_rate = []
for x in raw_fare:
    fare_rate.append(CleanFare(x))
```

After cleaning up the `raw_fare` list and writing it into a new list called `fare_rate`, we then have to convert all of the foreign currencies into US dollars. For this, we used a package called *forex-python* to pull the up-to-date exchange rates from the Internet. The package included the exchange rate for all the foreign currencies we have, except two: New Taiwan dollars and Pakistani rupee.

For New Taiwan dollars, we managed to find another package called *twder* that gave us the exchange rate. However, instead of exchange rate from New Taiwan dollars to US dollars, it gives us the exchange rate from US dollars to New Taiwan dollars. This means that instead of multiplying by the exchange rate, we had to divide by the exchange rate instead. Lastly, for Pakistani rupee, we wouldn't find any package that would allow us to pull the exchange rate from the Internet, so we had to enter it manually.

```
from forex_python.converter import CurrencyRates
import twder
c = CurrencyRates()

def Converter(fare_rate):
    if "EUR" in fare_rate:
        return round((c.get_rate('EUR','USD'))*
                       float(fare_rate.split("EUR")[1]), 2)
    if u'\u20ac' in fare_rate:
        return round((c.get_rate('EUR','USD'))*
                       float(fare_rate.split(u'\u20ac')[1]), 2)
    if u'\xa5' in fare_rate:
        return round((c.get_rate('JPY','USD'))*
                       float(fare_rate.split(u'\xa5')[1]), 2)
    if "HK$" in fare_rate:
        return round((c.get_rate('HKD','USD'))*
                       float(fare_rate.split("HK$")[1]), 2)
    if "SEK" in fare_rate:
        return round((c.get_rate('SEK','USD'))*
                       float(fare_rate.split("SEK")[1]), 2)
    if "SGD" in fare_rate:
        return round((c.get_rate('SGD','USD'))*
                       float(fare_rate.split("SGD")[1]), 2)
    if "CZK" in fare_rate:
        return round((c.get_rate('CZK','USD'))*
                       float(fare_rate.split("CZK")[1]), 2)
    if "CNY" in fare_rate:
        return round((c.get_rate('CNY','USD'))*
                       float(fare_rate.split("CNY")[1]), 2)
    if "C$" in fare_rate:
        return round((c.get_rate('CAD','USD'))*
                       float(fare_rate.split("C$")[1]), 2)
    if "A$" in fare_rate:
        return round((c.get_rate('AUD','USD'))*
                       float(fare_rate.split("A$")[1]), 2)
    if "CHF" in fare_rate:
        return round((c.get_rate('CHF','USD'))*
                       float(fare_rate.split("CHF")[1]), 2)
    if "NT$" in fare_rate:
        return round((float(fare_rate.split("NT$")[1])/
                       float(twder.now('USD')[1])), 2)
    if "PKR" in fare_rate:
        return round((float(fare_rate.split("PKR")[1])/139.10), 2)
    if "US$" in fare_rate:
        return float(fare_rate.split("US$")[1])
    else:
        return "NA"

us_rate = []
for x in fare_rate:
us_rate.append(Converter(x))
```

3) Write the cleaned-up dataset to a local SQLite database.

```
import sqlite3

db_file = "farebox.db"
conn = sqlite3.connect(db_file)

create_table_sql = """CREATE TABLE IF NOT EXISTS farebox_ratio(
                       continent TEXT, country TEXT, system TEXT,
                       ratio REAL, fare_system TEXT, fare_rate
                       TEXT, us_rate REAL, year REAL);"""

cur=conn.cursor()
cur.execute(create_table_sql)

for a, b, c, d, e, f, g, h in zip(continent, country, system,
                                  ratio, fare_system, fare_rate,
                                  us_rate, year):
    sql = """INSERT INTO farebox_ratio(continent, country, system,
                                        ratio, fare_system,
                                        fare_rate, us_rate, year)
                                        VALUES ('%s', '%s',
                                        '%s', '%s', '%s', '%s',
                                        '%s', '%s');""" %(a, b, c,
                                        d, e, f, g, h)
    cur.execute(sql)
    conn.commit()
```

4) Access the SQLite database from R, and create the following plots:

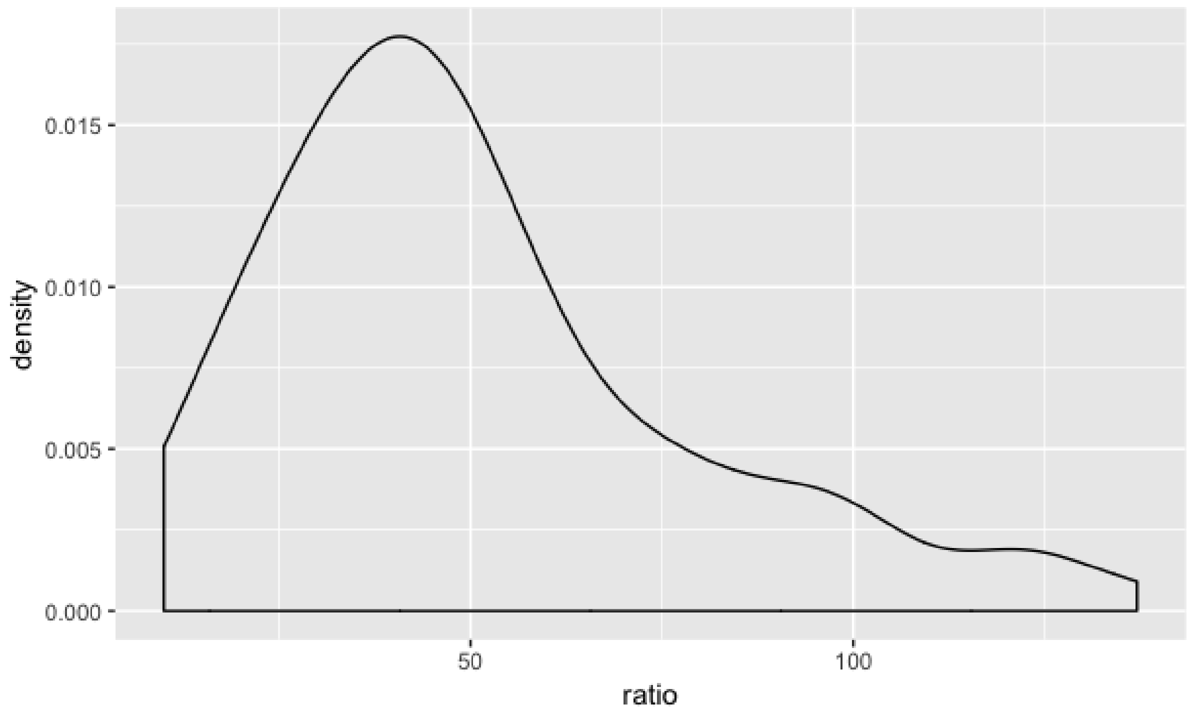
```
library(RSQLite)
library(ggplot2)

con = dbConnect(drv=SQLite(),dbname="farebox.db")
df.farebox <- dbGetQuery(con, "SELECT * FROM farebox_ratio;")
```

- The unconditioned distribution of farebox ratio (called "Ratio" in the Wiki table)

```
ggplot(data = df.farebox, aes(x = ratio)) +
  geom_density()
```

Output:

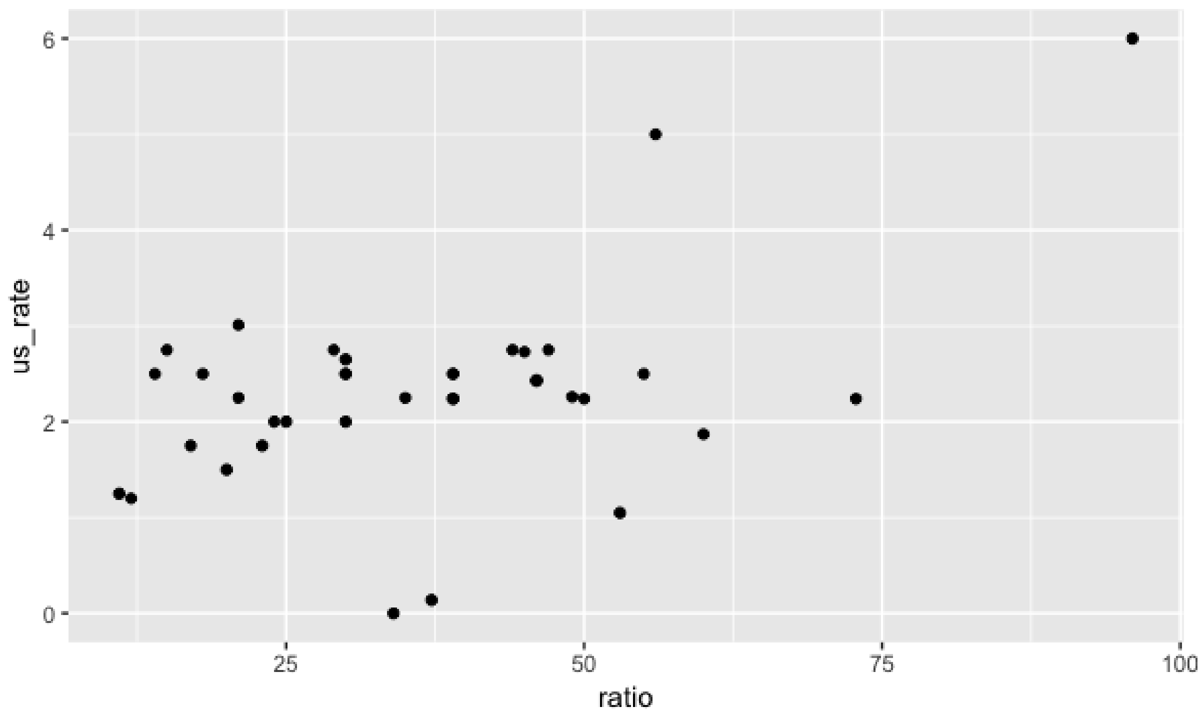


- Scatter plot showing farebox ratio versus fare rates (i.e., what it costs to ride) for flat rate systems

```
df.flat_rate <- dbGetQuery(con, "SELECT * FROM farebox_ratio WHERE
                                  fare_system = 'Flat rate';")

ggplot(data = df.flat_rate, aes(x = ratio, y = us_rate)) +
  geom_point()
```

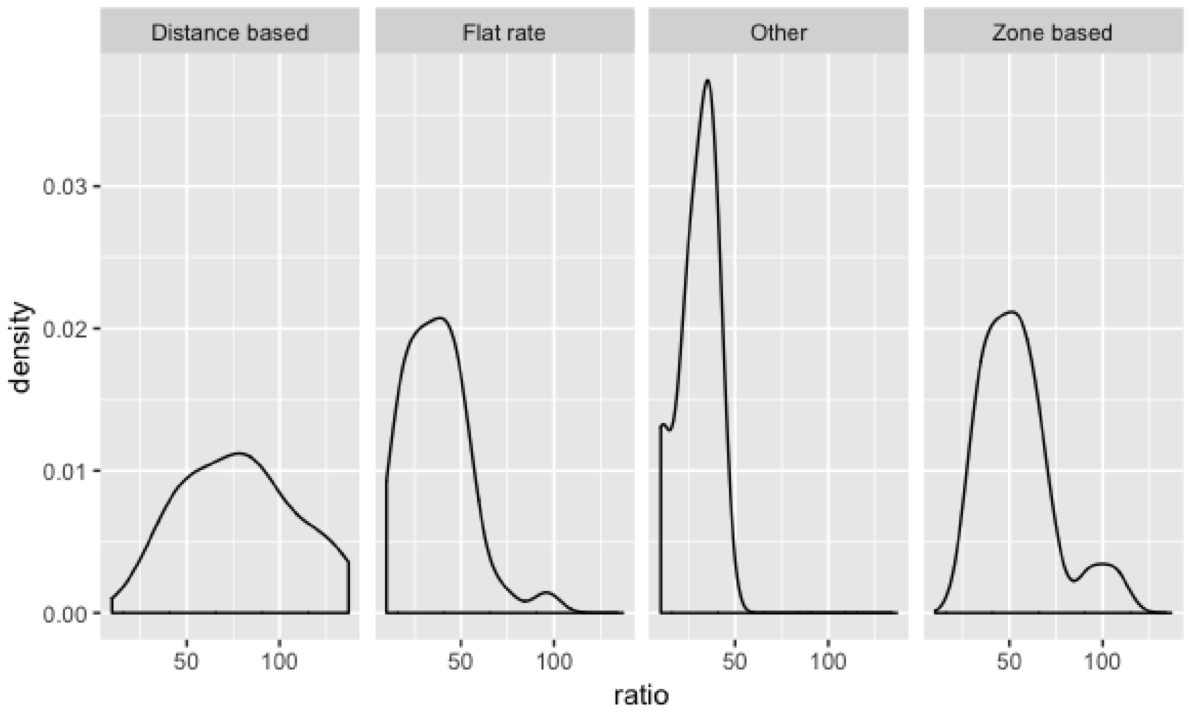
Output:



- Create a facet plot of the distribution of farebox ratios, by fare system i.e., `(facet_wrap(~fare_system, ncol = ...).`

```
ggplot(data = df.farebox, aes(x = ratio)) +
  geom_density() + facet_wrap(~fare_system, ncol = 4)
```

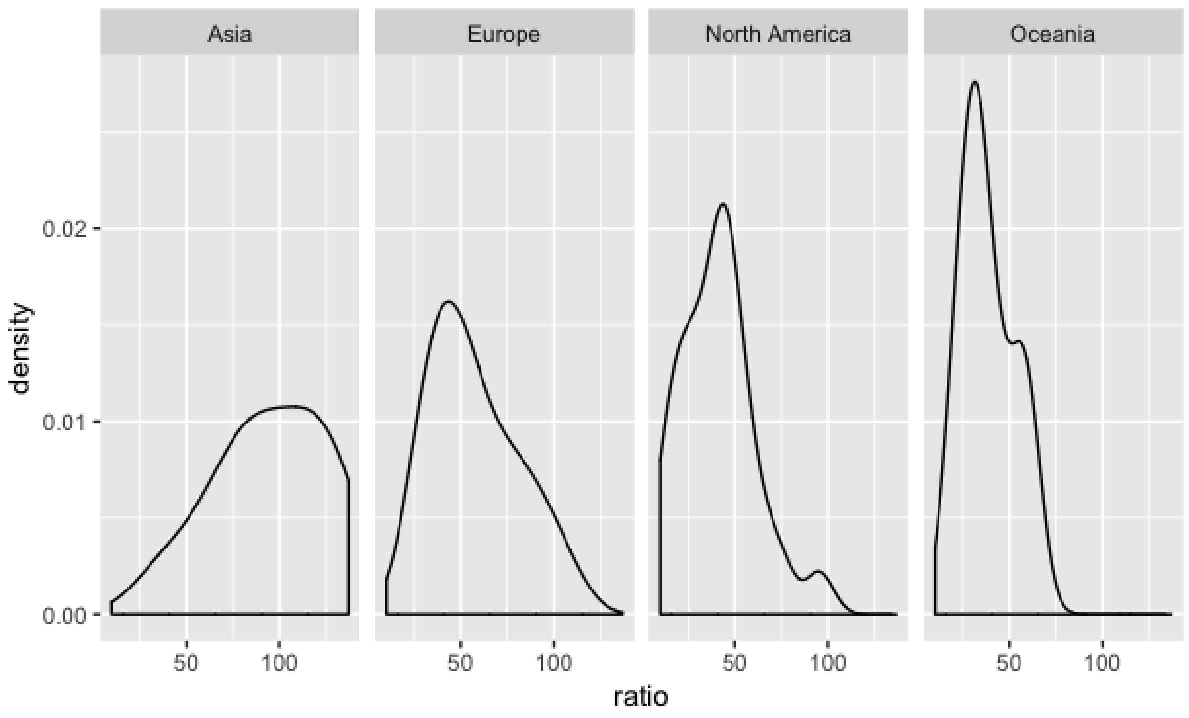
Output:



- Create a faceted plot showing the distribution of farebox ratios by continent (use a histogram or density)

```
ggplot(data = df.farebox, aes(x = ratio)) +
  geom_density() + facet_wrap(~continent, ncol = 4)
```

Output:

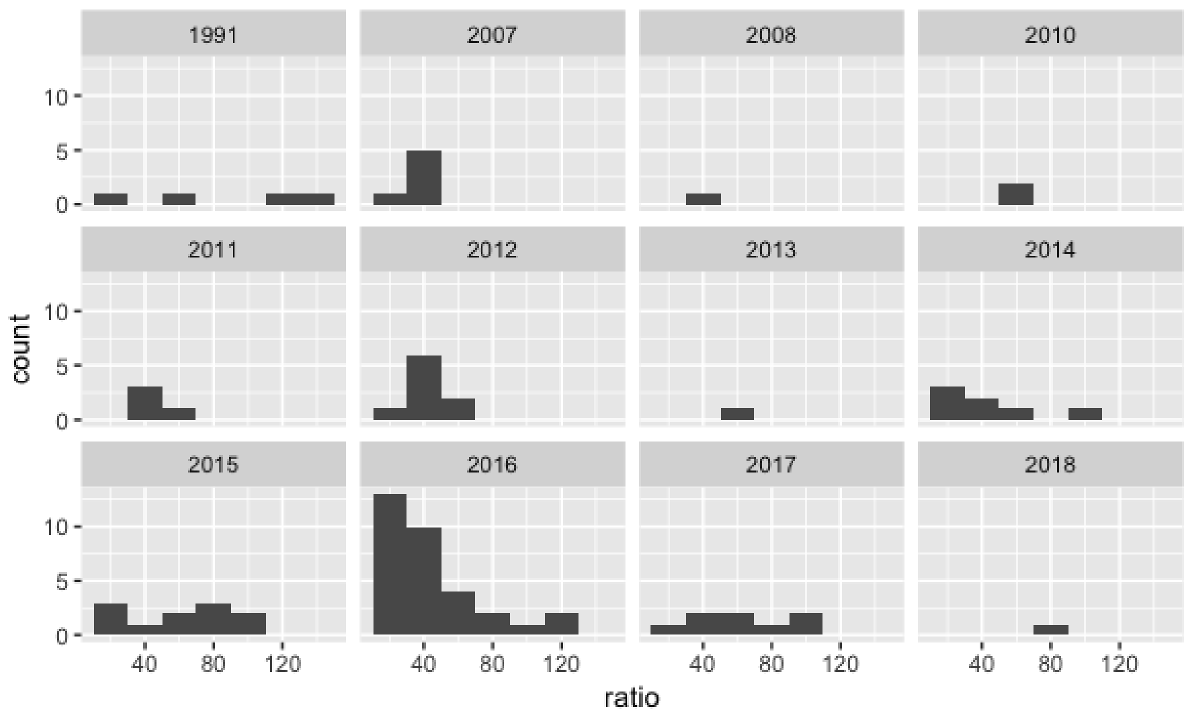


Three Person Team "Bonus" (not actually bonus - must be done) Questions

- Plot the distribution of farebox ratios by data collection year

```
ggplot(data = df.farebox, aes(x = ratio)) +
  geom_histogram(binwidth = 20) + facet_wrap(~year, ncol = 4)
```

Output: (on next page)

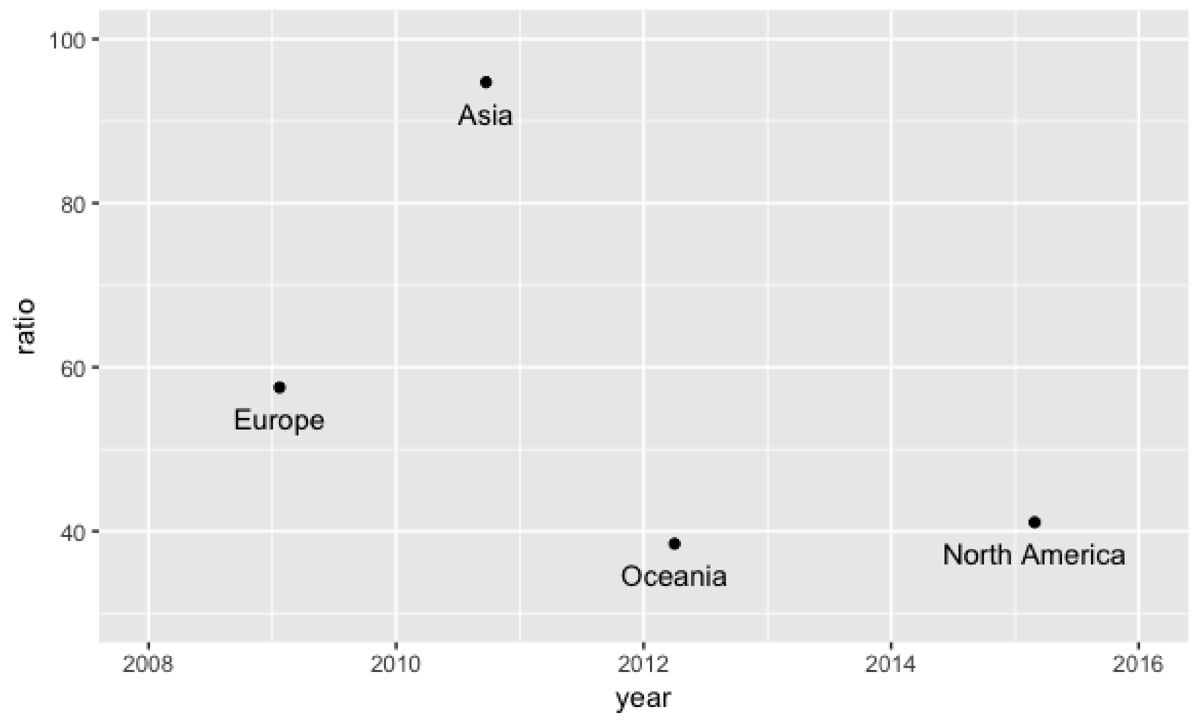


- Scatter plot of mean year by continent versus mean farebox ratio by continent

```
df.continent <- dbGetQuery(con, "SELECT continent, avg(ratio) AS
                                ratio, avg(year) AS year FROM
                                farebox_ratio GROUP BY continent;")

ggplot(data = df.continent, aes(x = year, y = ratio)) +
  geom_point() + geom_text(aes(label = continent), vjust = 2) +
  xlim(2008, 2016) + ylim(30, 100)
```

Output: (on next page)



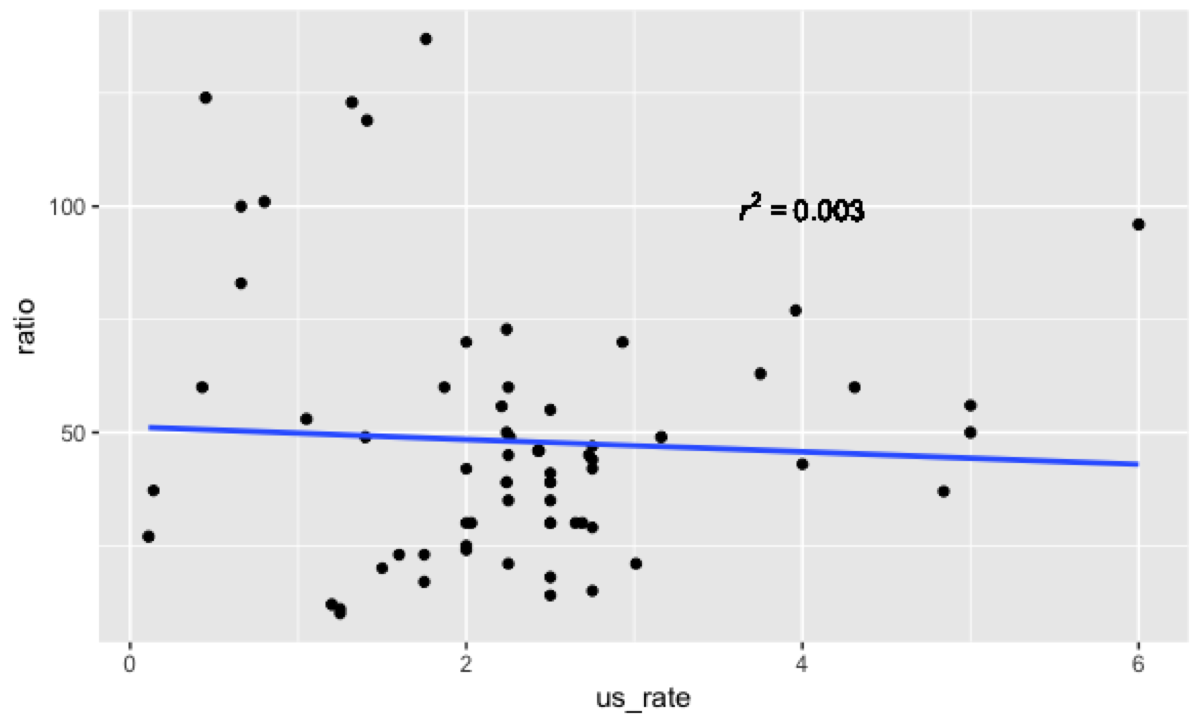
5) Run a prediction model of the fare collection ratio on predictors—what do we learn?

```
df.has_rate <- dbGetQuery(con, "SELECT * FROM farebox_ratio
WHERE us_rate != 'NA';")

lm_eqn <- function(df.has_rate){
  m <- lm(ratio ~ us_rate, df.has_rate);
  eq <- substitute(italic(r)^2~"="~r2,
                   list(r2 = format(summary(m)$r.squared,
digits = 3)))
  as.character(as.expression(eq));
}

ggplot(data = df.has_rate, aes(x = us_rate, y = ratio)) +
  geom_point() + geom_smooth(method = lm, se = FALSE) +
  geom_text(x = 4, y = 100, label = lm_eqn(df.has_rate),
parse = TRUE)
```

Output: (on next page)

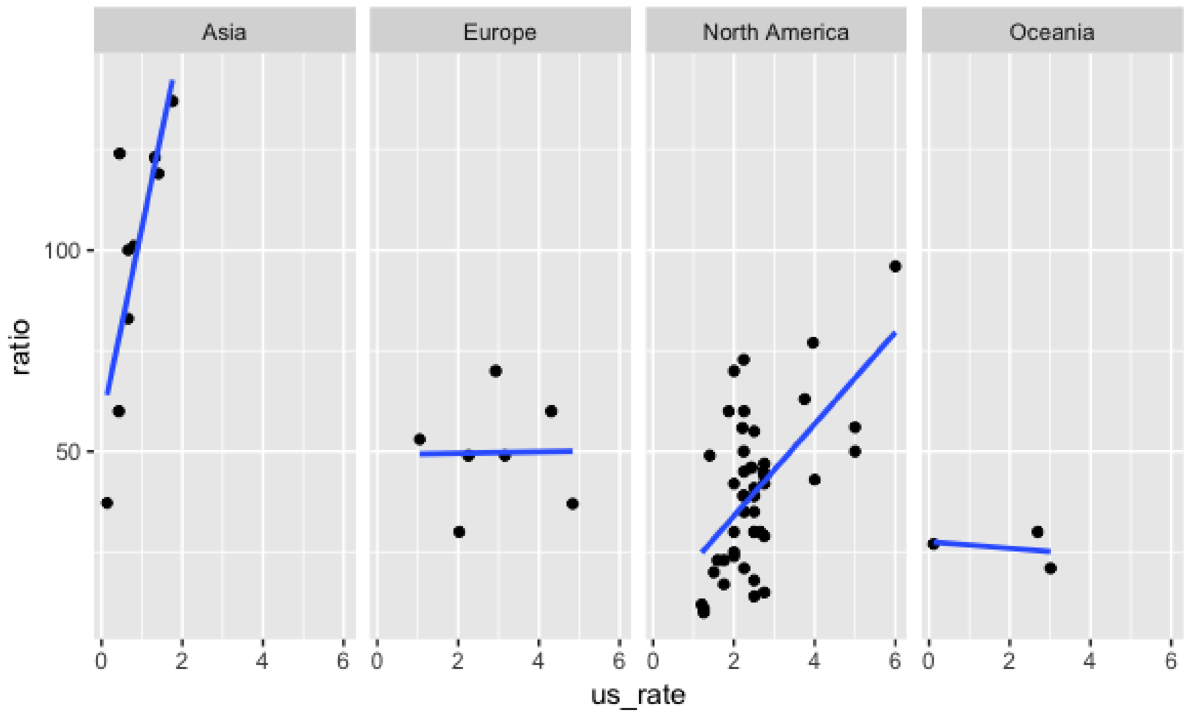


The graph above shows us the relation between the fare rate (standardized to US dollars) and the farebox recovery ratio. Although there was a total of 86 rows in the original table, 19 of which did not have a reported fare rate. Therefore, we ran this regression model using the 67 rows that did have a reported fare rate. As you can see from the linear regression line, which reports an R-squared close to zero, there is essentially no predictability in this model.

In order to dig deeper into the model, we decided to break it down further. The first step we took was to separate the data by continent, and see if it would increase predictability. The following is the code we used to create the facet plot:

```
ggplot(data = df.has_rate, aes(x = us_rate, y = ratio)) +
  geom_point() + geom_smooth(method = lm, se = FALSE) +
  facet_wrap(~continent, ncol = 4)
```

Output: (on next page)



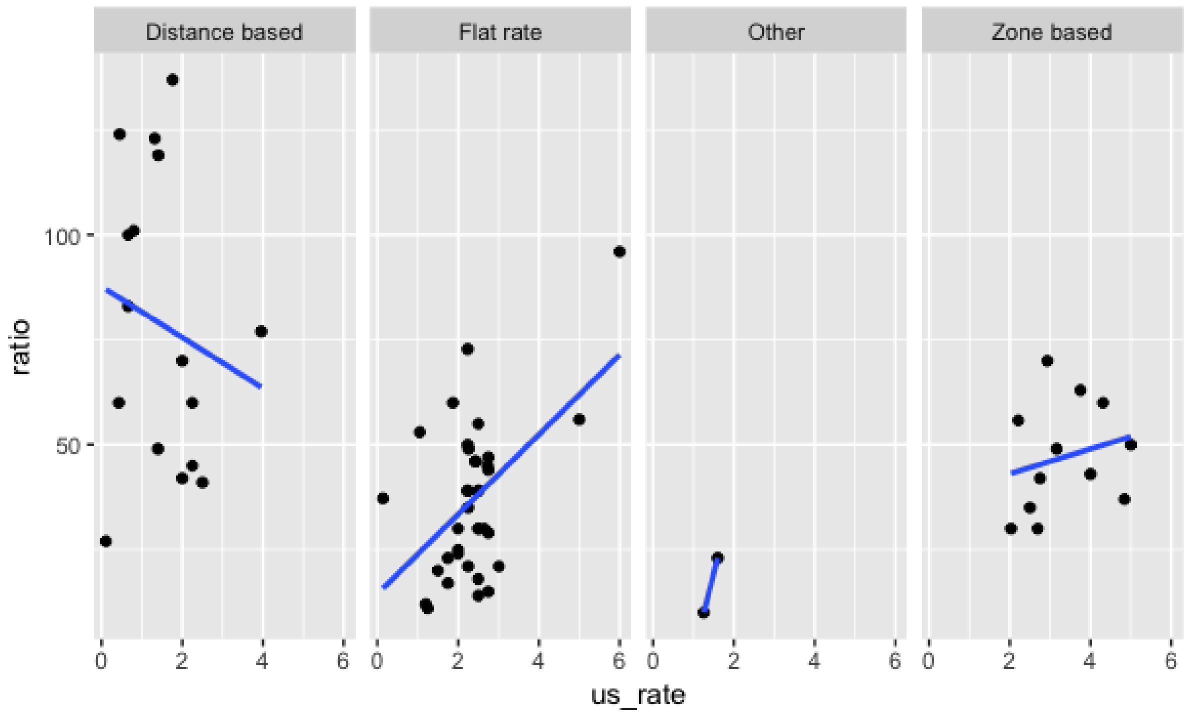
By separating the data by continent, we now begin to see some trend in Asia and North America, while Europe and Oceania still has no predictability. For Asia, it seems like a strong correlation, but we must note that this could very well be caused by the small sample size. If we would want to find out whether or not higher fare rates can actually lead to higher farebox recovery ratios, we would definitely have to include more samples of transportation systems in Asia to be able to draw a conclusion with a higher confidence level.

As for the data in North America, although the trend line suggests that there is a positive correlation between the two factors to some extent, we can still see that the majority of the data points do not fall anywhere close to the trend line. In fact, most fare rates fall around the US$2~3 range, but the corresponding ratio ranges from as low as 10% to as high as 75%. This tells us that the fare rate cannot successfully predict the recovery ratio.

Lastly, we took another step to break down the original data. This time, instead of separating by continent, we looked at how the output is when we separate the data by fare system, namely the four buckets we created. The following code generated the facet plot:

```
ggplot(data = df.has_rate, aes(x = us_rate, y = ratio)) +
  geom_point() + geom_smooth(method = lm, se = FALSE) +
  facet_wrap(~fare_system, ncol = 4)
```

Output: (on next page)



When we break down the data based on fare system, we can clearly see that there still are not any correlations strong enough to have a strong predictability. The zone bases system has basically no correlation; the distance based system has a range of recovery ratio from 25% to 150% and clearly did not correlate with the fare rate. Lastly, the flat rate system returned a result similar to what we saw in the North American sector in the previous graph, where similar fare rates have very different corresponding ratios.

In conclusion, after running the regression model, we have learned that we cannot predict the farebox recovery ratio with the fare rate. Increasing or decreasing the fare rate would most likely have an effect on the recovery ratio, but at least from the data we have, we cannot confidently predict what the impact will be and by how much. One thing worth noting though, is that by converting all the fare rates to US dollars and using that rate to run the regression model, we are essentially assuming that all the systems are operated in the same standard of living, which clearly is not true. Is we could incorporate standard of living into this data set, we might be able to better explain the relationship between the fare rate and the recovery ratio.