1. 49999 New York taxi trips

To drive a yellow New York taxi, you have to hold a "medallion" from the city's *Taxi and Limousine Commission*. Recently, one of those changed hands for over one million dollars, which shows how lucrative the job can be.

But this is the age of business intelligence and analytics! Even taxi drivers can stand to benefit from some careful investigation of the data, guiding them to maximize their profits. In this project, we will analyze a random sample of 49999 New York journeys made in 2013. We will also use regression trees and random forests to build a model that can predict the locations and times when the biggest fares can be earned.

Let's start by taking a look at the data!

In [99]:

```
# Loading the tidyverse
library(tidyverse)

# Reading in the taxi data
taxi <- read_csv("datasets/taxi.csv")

# Taking a look at the first few rows in taxi
head(taxi)
```

```
Parsed with column specification:
cols(
  medallion = col_character(),
  pickup_datetime = col_datetime(format = ""),
  pickup_longitude = col_double(),
  pickup_latitude = col_double(),
  trip_time_in_secs = col_double(),
  fare_amount = col_double(),
  tip_amount = col_double()
)
```

medallion	pickup_datetime	pickup_longitude	pickup_lat
4D24F4D8EF35878595044A52B098DFD2	2013-01-13 10:23:00	-73.94646	40.77273
A49C37EB966E7B05E69523D1CB7BE303	2013-01-13 04:52:00	-73.99827	40.74041
1E4B72A8E623888F53A9693C364AC05A	2013-01-13 10:47:00	-73.95346	40.77586
F7E4E9439C46B8AD5B16AB9F1B3279D7	2013-01-13 11:14:00	-73.98137	40.72473
A9DC75D59E0EA27E1ED328E8BE8CD828	2013-01-13 11:24:00	-73.96800	40.76000
19BF1BB516C4E992EA3FBAEDA73D6262	2013-01-13 10:51:00	-73.98502	40.76341

In [100]:
```
library(testthat)
library(IRkernel.testthat)

run_tests({
    test_that("Test that tidyverse is loaded", {
        expect_true( "package:tidyverse" %in% search(),
            info = "The tidyverse package should be loaded using library().")
    })

    test_that("Read in data correctly.", {
        expect_is(taxi, "tbl_df",
            info = 'You should use read_csv (with an underscore) to read "data
sets/taxi.csv" into taxi.')
    })

    test_that("Read in data correctly.", {
        taxi_temp <- read_csv('datasets/taxi.csv')
        expect_equivalent(taxi, taxi_temp,
            info = 'taxi should contain the data in "datasets/taxi.csv".')
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 55.954 0.363 2695.436 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

2. Cleaning the taxi data

As you can see above, the `taxi` dataset contains the times and price of a large number of taxi trips. Importantly we also get to know the location, the longitude and latitude, where the trip was started.

Cleaning data is a large part of any data scientist's daily work. It may not seem glamorous, but it makes the difference between a successful model and a failure. The `taxi` dataset needs a bit of polishing before we're ready to use it.

In [101]:
```
# Renaming the location variables,
# dropping any journeys with zero fares and zero tips,
# and creating the total variable as the log sum of fare and tip
taxi <- taxi %>%
   rename(long = pickup_longitude, lat = pickup_latitude)  %>%
   filter(fare_amount > 0 | tip_amount > 0) %>%
   mutate(total = log(fare_amount + tip_amount) )
```

In [102]:
```
run_tests({
    test_that("rename lat", {
        expect_true(!is.null(taxi$lat),
            info = "The taxi data frame does not contain a variable called la
t. You need to rename pickup_latitude.")
    })
    test_that("rename long", {
        expect_true(!is.null(taxi$long),
            info = "The taxi data frame does not contain a variable called lon
g. You need to rename pickup_longitude.")
    })
    test_that("total exists", {
        expect_true(!is.null(taxi$total),
            info = "The taxi data frame does not contain a variable called tot
al. You need to create this as the logarithm (use the log() function) of the s
um of fare_amount and tip_amount.")
    })
    test_that("Modified data correctly.", {
        taxi_temp <- read_csv('datasets/taxi.csv') %>%
            rename(long = pickup_longitude, lat = pickup_latitude)  %>%
            filter(fare_amount > 0 | tip_amount > 0) %>%
            mutate(total = log(fare_amount + tip_amount) )
        expect_equivalent(taxi, taxi_temp,
            info = 'The taxi dataframe has not been modified correctly. See if
you can find something is wrong with your code.')
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 56.095 0.363 2695.577 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

3. Zooming in on Manhattan

While the dataset contains taxi trips from all over New York City, the bulk of the trips are to and from Manhattan, so let's focus only on trips initiated there.

In [103]:
```
# Reducing the data to taxi trips starting in Manhattan
# Manhattan is bounded by the rectangle with
# latitude from 40.70 to 40.83 and
# longitude from -74.025 to -73.93
taxi <- taxi  %>%
    filter(between(lat,  40.70, 40.83) &
           between(long, -74.025, -73.93))
```

In [104]:
```
run_tests({
  test_that("The correct number of rows have been filtered away", {
      expect_equal(45766, nrow(taxi),
      info = "It seems you haven't filter away the taxi trips outside of Manha
ttan correctly.")
  })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 56.225 0.367 2695.71 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

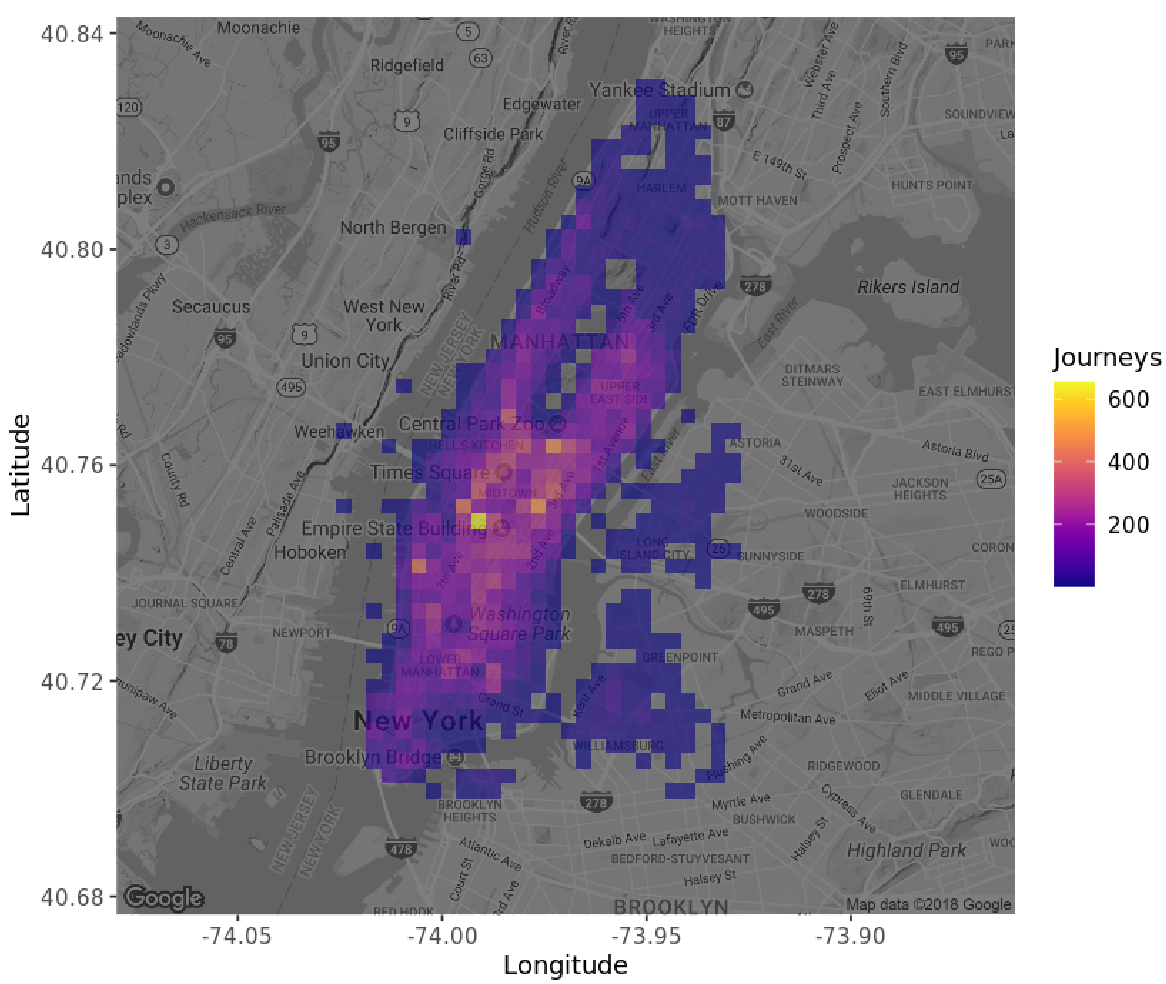
4. Where does the journey begin?

It's time to draw a map! We're going to use the excellent `ggmap` package together with `ggplot2` to visualize where in Manhattan people tend to start their taxi journeys.

In [105]:
```
# Loading in ggmap and viridis for nice colors
library(ggmap)
library(viridis)

# Retrieving a stored map object which originally was created by
# manhattan <- get_map("manhattan", zoom = 12, color = "bw")
manhattan <- readRDS("datasets/manhattan.rds")

# Drawing a density map with the number of journey start locations
ggmap(manhattan, darken = 0.5) +
   scale_fill_viridis(option = 'plasma') +
geom_bin2d(data = taxi, aes(x=long, y=lat), bins=60, alpha=0.6) +
   labs(x='Longitude', y='Latitude', fill='Journeys')
```



In [106]:
```
run_tests({

    test_that("Test that ggmap is loaded", {
        expect_true( "package:ggmap" %in% search(),
            info = "The ggmap package should be loaded using library().")
    })
    test_that("Test that viridis is loaded", {
        expect_true( "package:viridis" %in% search(),
            info = "The viridis package should be loaded using library().")
    })

    test_that("Check that geom_bin2d was used", {
        p <- last_plot()
        stat_classes <- as.character(sapply(p$layers, function(layer) {
            class(layer$stat)
        }))

        expect_true("StatBin2d" %in% stat_classes,
            info = "You need to use geom_bin2d correctly to draw the map.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 59.438 0.375 2698.93 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

5. Predicting taxi fares using a tree

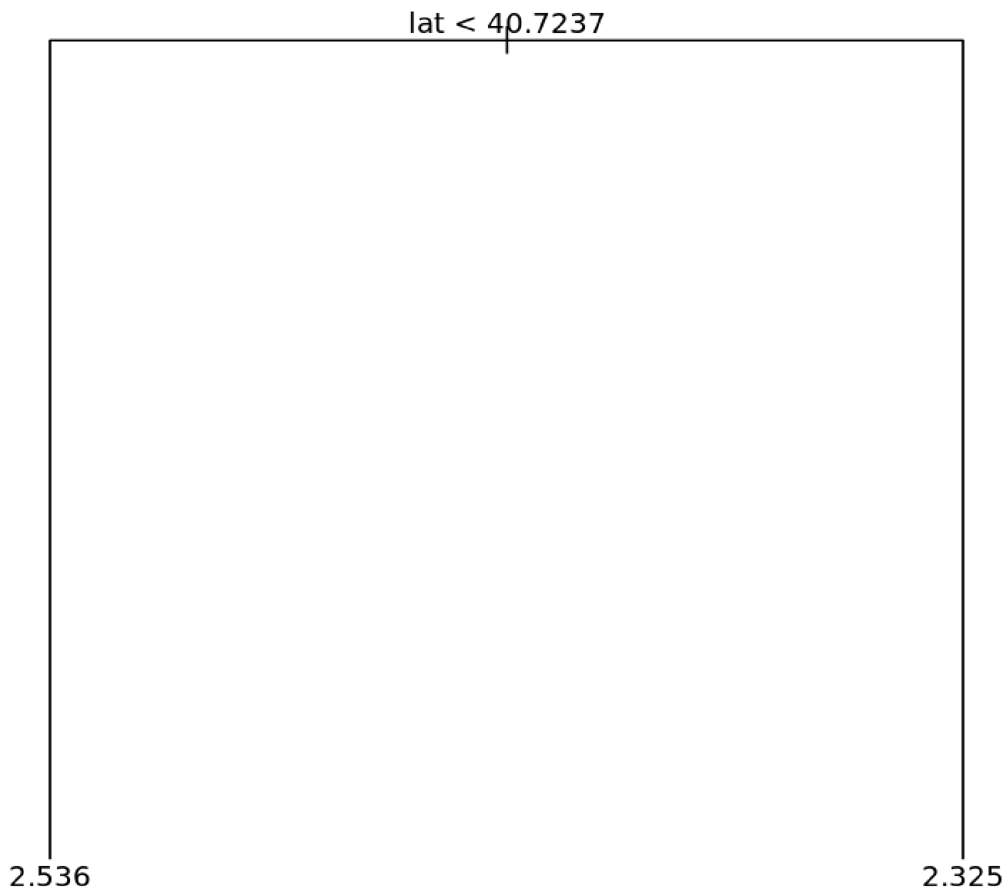
The map from the previous task showed that the journeys are highly concentrated in the business and tourist areas. We also see that some taxi trips originating in Brooklyn slipped through, but that's fine.

We're now going to use a regression tree to predict the `total` fare with `lat` and `long` being the predictors. The `tree` algorithm will try to find cutpoints in those predictors that results in the decision tree with the best predictive capability.

In [107]:
```
# Loading in the tree package
library(tree)

# Fitting a tree to lat and long
fitted_tree <- tree(total ~lat + long, data = taxi)

# Draw a diagram of the tree structure
plot(fitted_tree)
text(fitted_tree)
```



In [108]:
```
run_tests({
    test_that("Test that tree is loaded", {
        expect_true( "package:tree" %in% search(),
            info = "The tree package should be loaded using library().")
    })
  test_that("The tree has been fitted correctly", {
      correctly_fitted_tree <- tree(total ~ lat + long, data = taxi)
      expect_equivalent(fitted_tree, correctly_fitted_tree,
      info = "It seem you didn't fit the tree correctly. Check the hint, it mi
ght help!")
  })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 59.628 0.375 2699.119 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

6. It's time. More predictors.

The tree above looks a bit frugal, it only includes one split: It predicts that trips where `lat < 40.7237` are more expensive, which makes sense as it is downtown Manhattan. But that's it. It didn't even include `long` as `tree` deemed that it didn't improve the predictions. Taxi drivers will need more information than this and any driver paying for your data-driven insights would be disappointed with that. As we know from Robert de Niro, it's best not to upset New York taxi drivers.

Let's start by adding some more predictors related to the *time* the taxi trip was made.

In [109]:

```
# Loading in the lubridate package
library(lubridate)

# Generate the three new time variables
taxi <- taxi %>%
   mutate(hour = hour(pickup_datetime),
           wday = wday(pickup_datetime, label = TRUE),
           month = month(pickup_datetime, label = TRUE))

head(taxi)
```

medallion	pickup_datetime	long	lat	trip_tim
4D24F4D8EF35878595044A52B098DFD2	2013-01-13 10:23:00	-73.94646	40.77273	600
A49C37EB966E7B05E69523D1CB7BE303	2013-01-13 04:52:00	-73.99827	40.74041	840
1E4B72A8E623888F53A9693C364AC05A	2013-01-13 10:47:00	-73.95346	40.77586	60
F7E4E9439C46B8AD5B16AB9F1B3279D7	2013-01-13 11:14:00	-73.98137	40.72473	720
A9DC75D59E0EA27E1ED328E8BE8CD828	2013-01-13 11:24:00	-73.96800	40.76000	240
19BF1BB516C4E992EA3FBAEDA73D6262	2013-01-13 10:51:00	-73.98502	40.76341	540

In [110]:
```
run_tests({
    test_that("Test that lubridate is loaded", {
        expect_true( "package:lubridate" %in% search(),
            info = "The lubridate package should be loaded using library().")
    })
    test_that("hour is correct", {
        expect_equivalent(taxi$hour[1], 10L,
            info = "The `hour` column doesn't seem to be correct. Check the hi
nt for more help.")
    })
    test_that("wday is correct", {
        expect_true(taxi$wday[1] == "Sun",
            info = "The `wday` column doesn't seem to be correct. Check the hi
nt for more help.")
    })
    test_that("month is correct", {
        expect_true(taxi$month[1] == "Jan",
            info = "The `month` column doesn't seem to be correct. Check the h
int for more help.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 59.877 0.379 2699.371 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

7. One more tree!

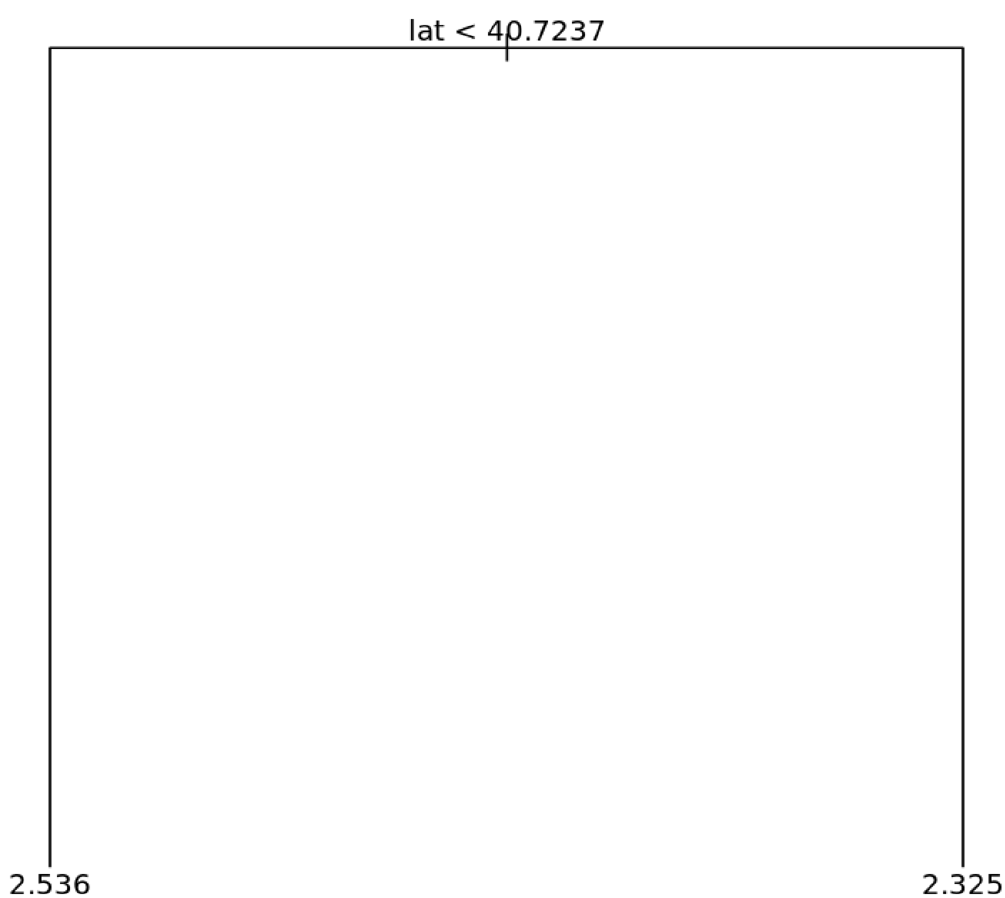
Let's try fitting a new regression tree where we include the new time variables.

In [111]:
```
# Fitting a tree with total as the outcome and
# lat, long, hour, wday, and month as predictors
fitted_tree <- tree(total ~ lat + long + hour + wday + month, data = taxi)

# draw a diagram of the tree structure
plot(fitted_tree)
text(fitted_tree)

# Summarizing the performance of the tree
print(summary(fitted_tree))
```

```
Regression tree:
tree(formula = total ~ lat + long + hour + wday + month, data = taxi)
Variables actually used in tree construction:
[1] "lat"
Number of terminal nodes:  2
Residual mean deviance:  0.3041 = 13910 / 45760
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.61900 -0.37880 -0.04244  0.00000  0.32660  2.69900
```



In [112]:
```
run_tests({
  test_that("The tree has been fitted correctly", {
      correctly_fitted_tree <- tree(total ~ lat + long + hour + wday + month,
 data = taxi)
      expect_equivalent(fitted_tree, correctly_fitted_tree,
      info = "It seem you didn't fit the tree correctly. Check the hint, it mi
ght help!")
  })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 60.09 0.379 2699.584 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

8. One tree is not enough

The regression tree has not changed after including the three time variables. This is likely because latitude is still the most promising first variable to split the data on, and after that split, the other variables are not informative enough to be included. A random forest model, where many different trees are fitted to subsets of the data, may well include the other variables in some of the trees that make it up.

In [113]:
```
# Loading in the randomForest package
library(randomForest)
# Fitting a random forest
fitted_forest <- randomForest(total ~ lat + long + hour + wday + month, data = 
taxi, ntree = 80, sampsize = 10000)

# Printing the fitted_forest object
fitted_forest
```

```
Call:
 randomForest(formula = total ~ lat + long + hour + wday + month,      data = 
taxi, ntree = 80, sampsize = 10000) 
               Type of random forest: regression
                     Number of trees: 80
No. of variables tried at each split: 1

          Mean of squared residuals: 0.2995788
                    % Var explained: 2.84
```

In [114]:
```
run_tests({
    test_that("Test that randomForest is loaded", {
        expect_true( "package:randomForest" %in% search(),
            info = "The randomForest package should be loaded using library
().")
    })
    test_that("ntree is correct.", {
        expect_true(fitted_forest$ntree == 80,
            info = "The ntree argument to randomForest should be ntree = 80 ."
)
    })
    test_that("Check randomForest call was ok", {
        call_string <- paste(deparse(fitted_forest$call), collapse = " ")
        keywords <- c("total", "lat", "long", "hour", "wday", "month",
                      "ntree", "sampsize", "100")
        expect_true(all(str_detect(call_string, keywords)),
            info = "You have not called randomForest correctly. Did you includ
e all the predictors and the right output variable?.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 63.147 0.391 2702.653 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

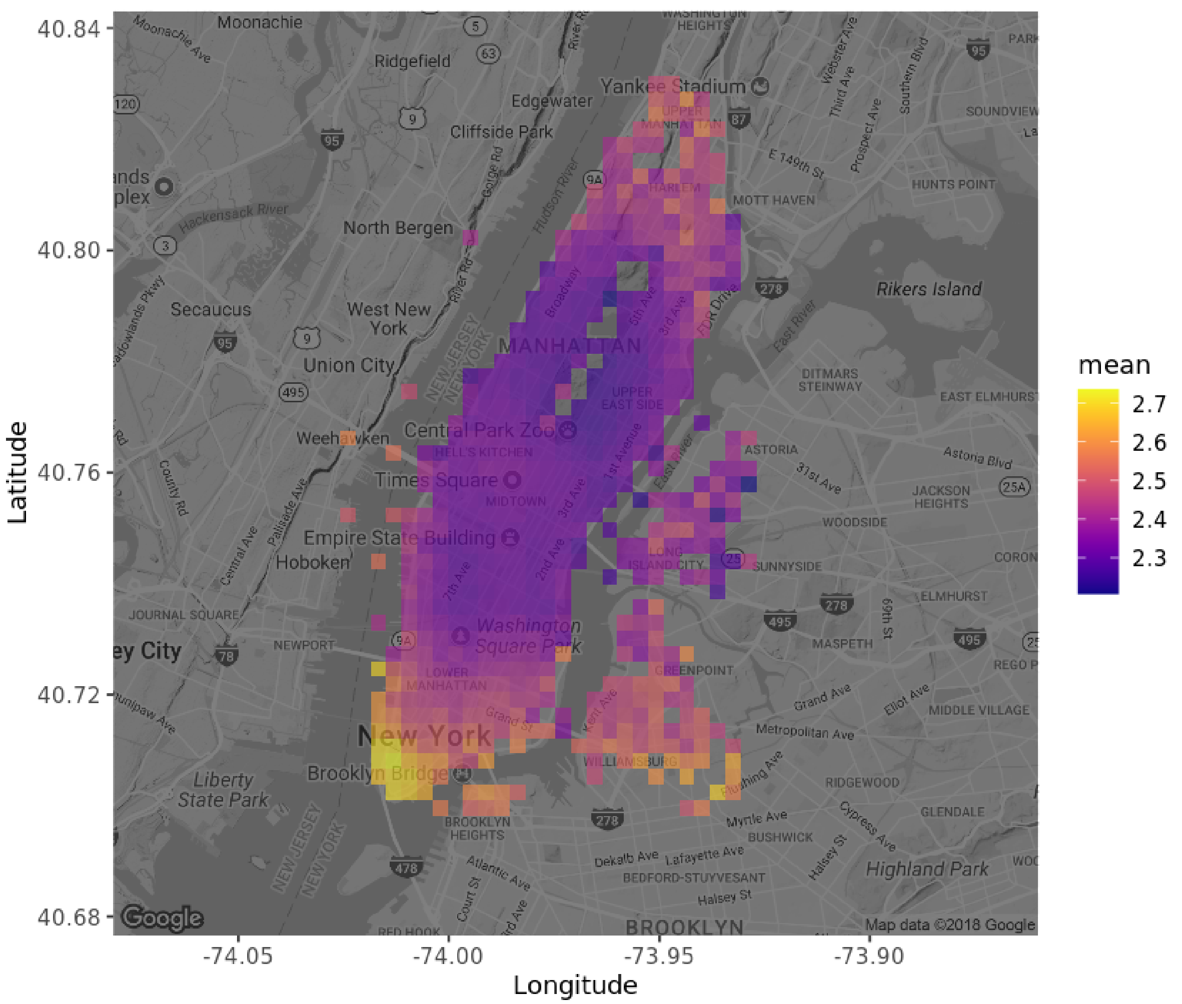
9. Plotting the predicted fare

In the output of `fitted_forest` you should see the `Mean of squared residuals`, that is, the average of the squared errors the model makes. If you scroll up and check the `summary` of `fitted_tree` you'll find `Residual mean deviance` which is the same number. If you compare these numbers, you'll see that `fitted_forest` has a slightly lower error. Neither predictive model is *that* good, in statistical terms, they explain only about 3% of the variance.

Now, let's take a look at the predictions of `fitted_forest` projected back onto Manhattan.

In [115]:
```
# Extracting the prediction from fitted_forest
taxi$pred_total <- fitted_forest$predicted

# Plotting the predicted mean trip prices from according to the random forest
ggmap(manhattan, darken = 0.5) +
   scale_fill_viridis(option = 'plasma') +
   stat_summary_2d(data = taxi, aes(x=long, y=lat, z=pred_total), bins=60, alp
ha=0.6, fun=mean) +
   labs(x='Longitude', y='Latitude', fill='mean')
```



In [116]:
```
run_tests({
    test_that("taxi$pred_total == fitted_forest$predicted", {
        expect_true(all(taxi$pred_total == fitted_forest$predicted),
            info = "You should assign fitted_forest$predicted to taxi$pred_tot
al .")
    })
    test_that("Check that stat_summary_2d was used", {
        p <- last_plot()
        stat_classes <- as.character(sapply(p$layers, function(layer) {
            class(layer$stat)
        }))

        expect_true("StatSummary2d" %in% stat_classes,
            info = "You need to use geom_bin2d correctly to draw the map.")
    })
    test_that("Check that pred_total was used", {
        p <- last_plot()
        p_variables <- unlist(sapply(p$layers, function(layer) {
            as.character(layer$mapping)
        }))
        expect_true(any(str_detect(p_variables, "pred_total")),
            info = "You need to connect pred_total to z in the aes() call corr
ectly.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 65.803 0.403 2705.32 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

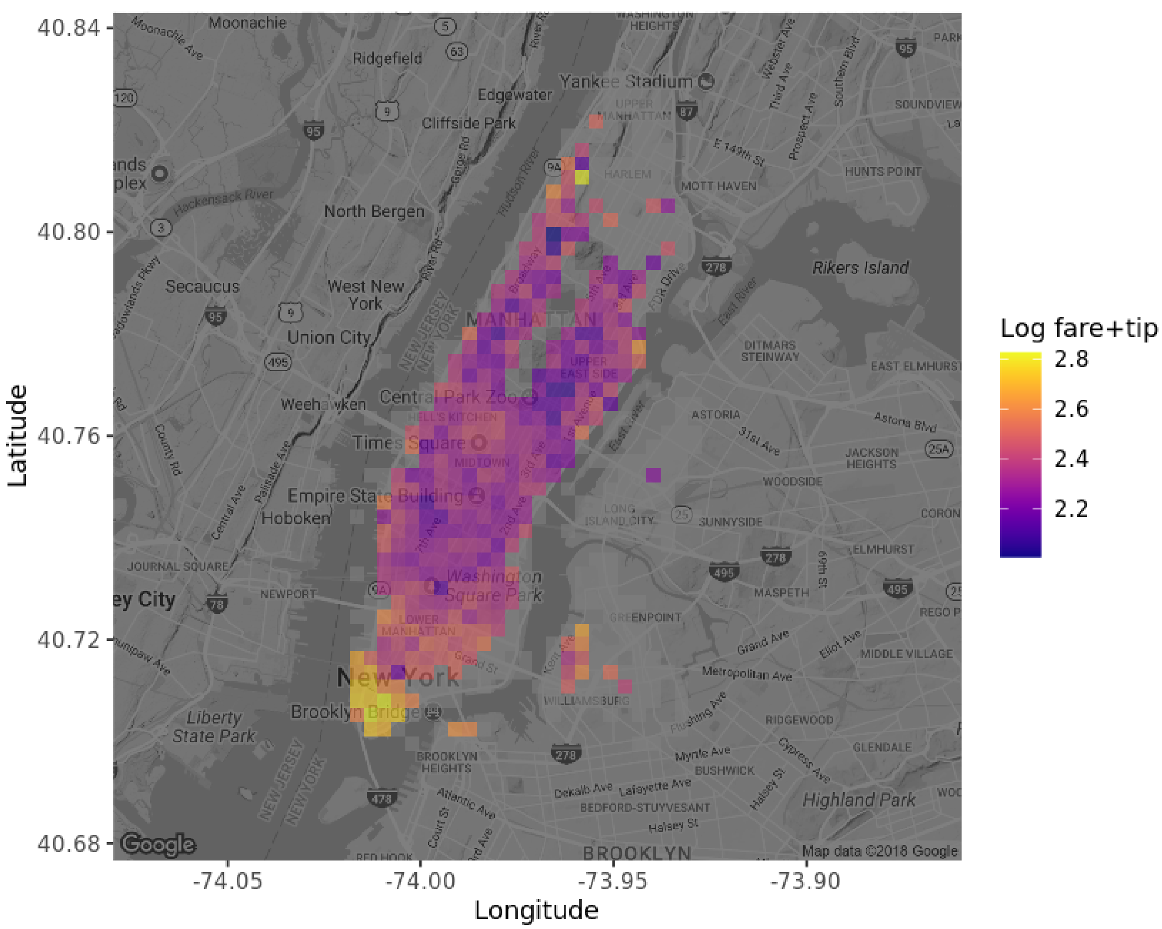
10. Plotting the actual fare

Looking at the map with the predicted fares we see that fares in downtown Manhattan are predicted to be high, while midtown is lower. This map only shows the prediction as a function of `lat` and `long`, but we could also plot the predictions over time, or a combination of time and space, but we'll leave that for another time.

For now, let's compare the map with the predicted fares with a new map showing the mean fares according to the data.

In [117]:
```
# Function that returns the mean *if* there are 15 or more datapoints
mean_if_enough_data <- function(x) {
    ifelse( length(x) >= 15, mean(x), NA)
}

# Plotting the mean trip prices from the data
ggmap(manhattan, darken=0.5) +
    stat_summary_2d(data=taxi, aes(x = long, y = lat, z = total),
                    fun = mean_if_enough_data,
                    alpha = 0.6, bins = 60) +
  scale_fill_viridis(option = 'plasma') +
  labs(x = 'Longitude', y = 'Latitude', fill = 'Log fare+tip')
```



In [118]:
```
run_tests({
    test_that("Check that total was used but not pred_total", {
        p <- last_plot()
        p_variables <- unlist(sapply(p$layers, function(layer) {
            as.character(layer$mapping)
        }))
        expect_true(any(str_detect(p_variables, "total")) &
                    !any(str_detect(p_variables, "pred_total")),
            info = "You need to connect total to z in the aes() call correctl
y. Make sure you are not still using pred_total.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 68.412 0.407 2707.932 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

11. Where do people spend the most?

So it looks like the random forest model captured some of the patterns in our data. At this point in the analysis, there are many more things we could do that we haven't done. We could add more predictors if we have the data. We could try to fine-tune the parameters of `randomForest`. And we should definitely test the model on a hold-out test dataset. But for now, let's be happy with what we have achieved!

So, if you are a taxi driver in NYC, where in Manhattan would you expect people to spend the most on a taxi ride?

In [119]:
```
# Where are people spending the most on their taxi trips?
spends_most_on_trips <- "downtown" # "uptown" or "downtown"
```

In [120]:
```
run_tests({
  test_that("...", {
      expect_true(str_detect(tolower(spends_most_on_trips), "downtown"),
      info = "Well, looking at the plot it looks like people pay more downtow
n.")
  })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 68.441 0.407 2707.96 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```