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 [2]:

```
# Loading the tidyverse
# .... YOUR CODE FOR TASK 1 ....
library(tidyverse)
# Reading in the taxi data
taxi <-read_csv("datasets/taxi.csv")
head(taxi)
# Taking a look at the first few rows in taxi
# .... YOUR CODE FOR TASK 1 ....
```

```
-- Attaching packages --------------------------------------- tidyverse 1.
3.0 --
v ggplot2 3.3.2     v purrr   0.3.4
v tibble  3.0.3     v dplyr   1.0.0
v tidyr   1.1.0     v stringr 1.4.0
v readr   1.3.1     v forcats 0.5.0
-- Conflicts ------------------------------------------ tidyverse_conflict
s() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
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
4D24F4D8EF35878595044A52B098DFD2	2013-01-13 10:23:00	-73.94646	40.772
A49C37EB966E7B05E69523D1CB7BE303	2013-01-13 04:52:00	-73.99827	40.740
1E4B72A8E623888F53A9693C364AC05A	2013-01-13 10:47:00	-73.95346	40.775
F7E4E9439C46B8AD5B16AB9F1B3279D7	2013-01-13 11:14:00	-73.98137	40.724
A9DC75D59E0EA27E1ED328E8BE8CD828	2013-01-13 11:24:00	-73.96800	40.760
19BF1BB516C4E992EA3FBAEDA73D6262	2013-01-13 10:51:00	-73.98502	40.763

In [3]:

```
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 "datasets/tax
i.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".')
    })
})
```

```
Attaching package: 'testthat'

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

    matches

The following object is masked from 'package:purrr':

    is_null

The following object is masked from 'package:tidyr':

    matches


<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: 2.468 0.165 74.511 0.006 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 [4]:

```
# 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(lat=pickup_latitude)%>%
   rename(long=pickup_longitude)%>%
   filter(fare_amount>0|tip_amount>0)%>%
   mutate(total=log(fare_amount+tip_amount))
taxi
```

medallion	pickup_datetime	long	lat	trip_
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
5F2EFC03B544635C9B0E7A4AA4FF9AC3	2013-01-13 12:53:00	-73.97295	40.79527	0
8DEB70907D00AA1D7FF5E2683240549B	2013-01-13 07:59:00	-73.96577	40.76530	120
E15F7CCB808DD15E0496D830D3DEDECE	2013-01-13 08:09:00	-73.94768	40.77507	720
0B3D3D51C78E944F68DC04209E86D5F7	2013-01-13 12:53:00	-73.98457	40.72488	180
3C16CFAD2B12F3508F7211C37F8F8B8F	2013-01-13 13:07:00	-73.97068	40.78506	360
5888835B75CF97CBE738A58484B12A6D	2013-01-13 12:31:00	-74.00164	40.74414	660
58A836AD639867DB949723BA2A941734	2013-01-13 13:37:00	-73.98800	40.74960	420
C01368C76C8DEFA6C678CF0D54AE87F0	2013-01-13 13:31:00	-73.99309	40.74755	600
F86299BD8DF9B0C90A4E20AD2A44EAF7	2013-01-13 14:31:00	-73.96832	40.78688	480
00D07524C1482FF5A3CB0932BE29003E	2013-01-13 13:02:00	-73.95131	40.81007	144(
2232F3F8B6124B1947AEF62509A97DAE	2013-01-13 16:16:00	-73.98534	40.72374	420
85045374BBBFA1376CE5D64DED56B010	2013-01-13 13:14:00	-73.91682	40.76967	120(
70F5983D94E1BA0B217E9F8F447D4382	2013-01-13 16:30:00	-73.97288	40.78684	300
3AC3AB85F6E59CB391253FA638B854AA	2013-01-13 13:24:00	-73.98768	40.71979	660

medallion	pickup_datetime	long	lat	trip_
60DC999A38D953EE86AB81C23C9480E8	2013-01-13 16:03:00	-73.98846	40.73724	780
084ACA045413975E5FB960ECBD9C6588	2013-01-13 15:58:00	-73.96040	40.76165	480
AC496252AF3119662DEDBFD24A70E83B	2013-01-13 16:08:00	-73.98025	40.75132	660
F2773B3D7BC9C2A18B942644F43DD33D	2013-01-13 14:45:00	-74.01029	40.71158	660
9E800FF6BCF49308A6BC0678ED27499D	2013-01-13 17:29:00	-73.97753	40.74235	420
ED9B6E969C35E16314E6863A95D83B5F	2013-01-13 17:48:00	-73.99191	40.73557	600
4FBA078630428EFA5EEEF2E67A293464	2013-01-13 15:14:00	-73.87447	40.77406	180(
FDD65AC3468B2A12AF0D3577D64FBF76	2013-01-13 19:15:00	-73.99110	40.69207	180(
135A8EDC6EBD6F4397B7A6D281C75AB0	2013-01-13 16:59:00	-73.97753	40.76359	240
131949A57D5717A3E7112404CCDFF27B	2013-01-13 17:36:00	-73.97504	40.75832	360
...	...	...	...	...
0CE65EC42F3ABEB0BCC08F274703DDE7	2013-12-03 11:58:00	-73.98287	40.74546	420
126A5559920CCC1F4AA7A7DFB137D328	2013-12-05 22:12:00	-74.00825	40.73770	540
221F94E1FB935807635C1045866796CC	2013-12-03 12:31:00	-73.95893	40.77771	420
0C4726D4E2AF94BF8FE2D23EFDA20917	2013-12-05 22:03:00	-73.99159	40.74441	102(
A6DB36B570BD59E08FBE76086C5EE662	2013-12-03 13:36:00	-73.98994	40.74703	120(
60C3AD7179183044E91FF3B9B90C1CAA	2013-12-03 11:50:00	-73.99038	40.72965	600
A910CF5A84FFE5F10B8CCBC06CC5F944	2013-12-05 22:42:00	-73.99249	40.74290	540
037CB433AB5895649F7A9EF37F767EF1	2013-12-05 22:49:00	-73.96532	40.77145	540
798452AE1E4F97CE2B491F9590861FEA	2013-12-03 14:26:00	-73.95112	40.78284	900
204BAB16D3382C5A5711068B28E624C2	2013-12-03 14:12:00	-73.97968	40.76131	780

medallion	pickup_datetime	long	lat	trip_
F3BA458FFB70903630ABF3332CB983F1	2013-12-03 14:39:00	-73.94083	40.79271	102(
5221BCE45F9FD34B709EB0882885B7AB	2013-12-05 23:21:00	-73.94922	40.80278	102(
2B3EFA3719EA953A656D5041750787EB	2013-12-03 13:43:00	-73.78947	40.64633	336(
B65ECEC404246E0A0370540B5FE24AAB	2013-12-05 23:40:00	-73.98241	40.75505	210(
8220D3BEF8BCF3C5DB50C66F32D9AB61	2013-12-05 23:50:00	-73.98628	40.74044	900
87CDD10EC56CB55A6F92D872A16AECCD	2013-12-03 14:32:00	-73.98254	40.76761	240(
758FE3823459A49A24051D260415E866	2013-12-05 22:42:00	-73.98413	40.72923	132(
554389035D554151A222468199443C39	2013-12-06 00:35:00	-73.94059	40.71187	540
1AABA654E5ABF4F25333847479904504	2013-12-03 13:55:00	-73.99451	40.74100	480
0C4CEF33F6F0D06E62988C22B6F53983	2013-12-05 22:46:00	-74.01471	40.71081	126(
16183FABEA1B2C53821A44CBFA0C1907	2013-12-03 16:10:00	-73.94863	40.77665	240
7D5EBEEF1F35996553AE89392DF504C8	2013-12-06 01:30:00	-73.99052	40.75630	360
82BA7115B2866F14CC4364621C71D050	2013-12-03 17:24:00	-73.95631	40.76751	120
6E2BE266388543BEBD9DAE67DE2EF745	2013-12-03 17:23:00	-74.00461	40.71924	660
162A967C11C5A4839059F1B1C9868C33	2013-12-03 15:56:00	-73.96405	40.77710	600
FB84C95C217D345556E3B14EA6D63E5C	2013-12-03 16:46:00	-73.99992	40.71914	660
1A507DEE7AD80F346106A3016A388038	2013-12-03 17:06:00	-73.99654	40.76344	660
D45DCD8D59D2C02E5630FAC6BF9B9F96	2013-12-06 06:53:00	-73.94276	40.79025	420
CCDD7C317BBF35D4585CB9BC4F4299A5	2013-12-03 16:53:00	-73.95743	40.78273	126(
90D244D17A03926D69448F687C1424A2	2013-12-03 17:37:00	-73.96688	40.79355	420

In [5]:

```
run_tests({
    test_that("rename lat", {
        expect_true(!is.null(taxi$lat),
            info = "The taxi data frame does not contain a variable called lat. You nee
d to rename pickup_latitude.")
    })
    test_that("rename long", {
        expect_true(!is.null(taxi$long),
            info = "The taxi data frame does not contain a variable called long. You ne
ed 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 total. You n
eed to create this as the logarithm (use the log() function) of the sum 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: 3.188 0.169 75.244 0.006 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 [6]:

```
# 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))%>%
    filter(between(long,-74.025,-73.93))
```

In [7]:

```
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 Manhattan 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: 3.452 0.181 75.52 0.006 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?

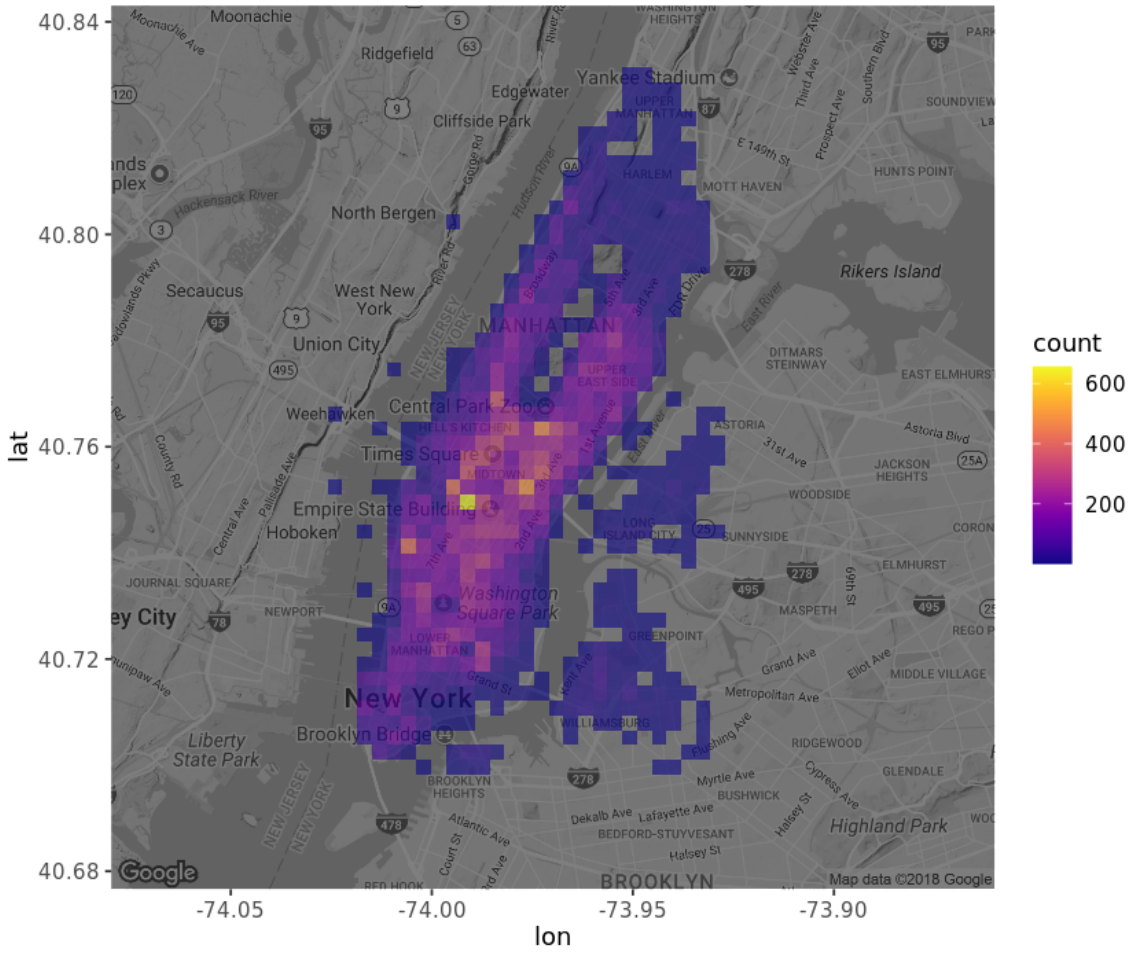
It's time to draw a map! We're going to use the excellent `ggmap` package together with `ggplot2` to visualize where in Manhattan people tend to start their taxi journeys.

In [8]:

```
# Loading in ggmap and viridis for nice colors
# .... YOUR CODE FOR TASK 4 ....

# Retrieving a stored map object which originally was created by
# manhattan <- get_map("manhattan", zoom = 12, color = "bw")
manhattan <- readRDS("datasets/manhattan.rds")
library(ggmap)
library(viridis)
# 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)
   # .... YOUR CODE FOR TASK 4 ....
```

```
Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
Please cite ggmap if you use it! See citation("ggmap") for details.
Loading required package: viridisLite
```


lat
40.84
40.80
40.76
40.72
40.68
lon
-74.05
-74.00
-73.95
-73.90
count
600
400
200
Moonachie
Ridgefield
Edgewater
Yankee Stadium
Cliffside Park
North Bergen
Secaucus
West New York
Union City
Weehawken
Central Park Zoo
Times Square
Empire State Building
Hoboken
Rikers Island
MANHATTAN
Washington Square Park
New York
Brooklyn Bridge
Liberty State Park
BROOKLYN
Highland Park
Google
Map data ©2018 Google

In [9]:

```
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: 8.865 0.24 81.902 0.006 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

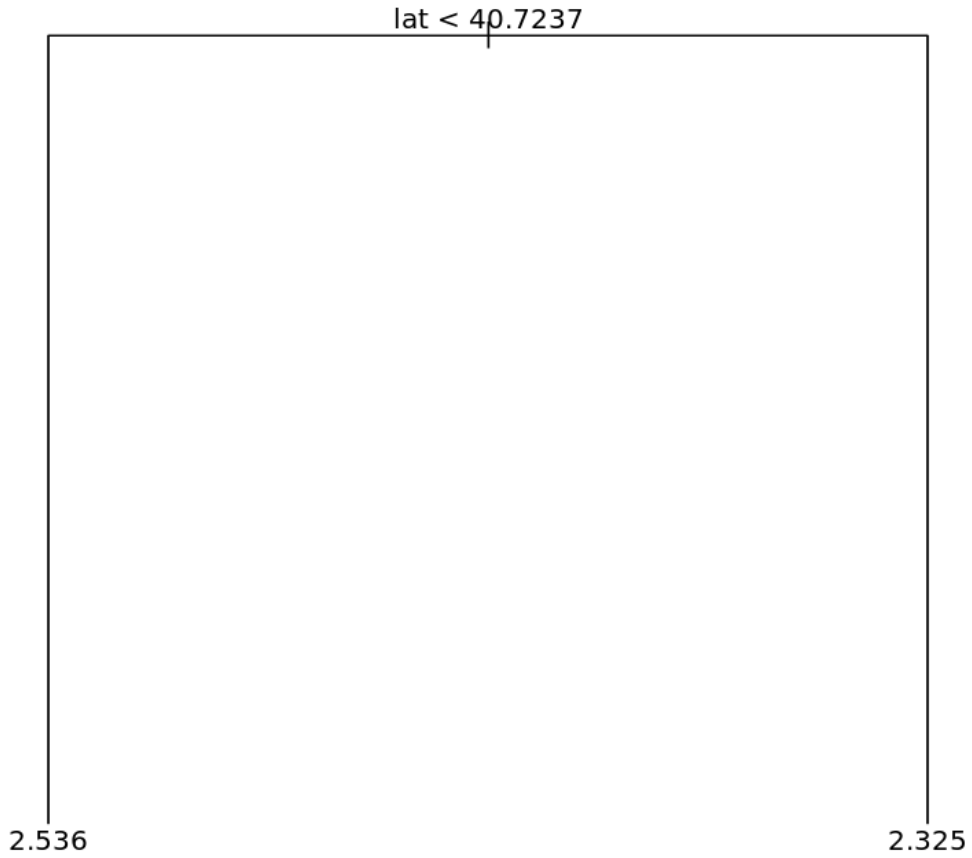
The map from the previous task showed that the journeys are highly concentrated in the business and tourist areas. We also see that some taxi trips originating in Brooklyn slipped through, but that's fine.

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 [10]:

```
# Loading in the tree package
# .... YOUR CODE FOR TASK 5 HERE ....
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)
# .... YOUR CODE FOR TASK 5 HERE ....
```



In [11]:

```
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 might hel
p!")
  })
})
```

```
<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: 9.136 0.248 82.209 0.006 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 [12]:

```
# Loading in the lubridate package
# .... YOUR CODE FOR TASK 6 HERE ....
library(lubridate)
# Generate the three new time variables
taxi <- taxi %>%
    mutate(hour=hour(taxi$pickup_datetime),wday=wday(taxi$pickup_datetime,label=T),mont
h=month(taxi$pickup_datetime,label=T))
```

```
Attaching package: 'lubridate'

The following object is masked from 'package:base':

    date
```

In [13]:

```
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$hour[1], 10L,
            info = "The `hour` column doesn't seem to be correct. Check the hint for mo
re help.")
    })
    test_that("wday is correct", {
        expect_true(taxi$wday[1] == "Sun",
            info = "The `wday` column doesn't seem to be correct. Check the hint for mo
re help.")
    })
    test_that("month is correct", {
        expect_true(taxi$month[1] == "Jan",
            info = "The `month` column doesn't seem to be correct. Check the hint for m
ore 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: 9.655 0.256 82.74 0.006 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 [14]:

```
# 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
# .... YOUR CODE FOR TASK 7 HERE ....
plot(fitted_tree)
text(fitted_tree)
# Summarizing the performance of the tree
summary(fitted_tree)
# .... YOUR CODE FOR TASK 7 HERE ....
```

```
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 [15]:

```
run_tests({
  test_that("The tree has been fitted correctly", {
      correctly_fitted_tree <- tree(total ~ lat + long + hour + wday + month, data = ta
xi)
      expect_equivalent(fitted_tree, correctly_fitted_tree,
      info = "It seem you didn't fit the tree correctly. Check the hint, it might hel
p!")
  })
})
```

```
<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: 10.024 0.26 83.111 0.006 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 [16]:

```
# Loading in the randomForest package
# .... YOUR CODE FOR TASK 8 HERE ....
library(randomForest)
# Fitting a random forest
fitted_forest <- randomForest(total~lat+long+hour+wday+month,data=taxi,ntree=80,sampsiz
e=10000)

# Printing the fitted_forest object
fitted_forest
# .... YOUR CODE FOR TASK 8 HERE ....
```

```
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


Call:
 randomForest(formula = total ~ lat + long + hour + wday + month,      dat
a = 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: 0.2998258
                    % Var explained: 2.76
```

In [17]:

```
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 include 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: 13.378 0.316 86.589 0.006 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.

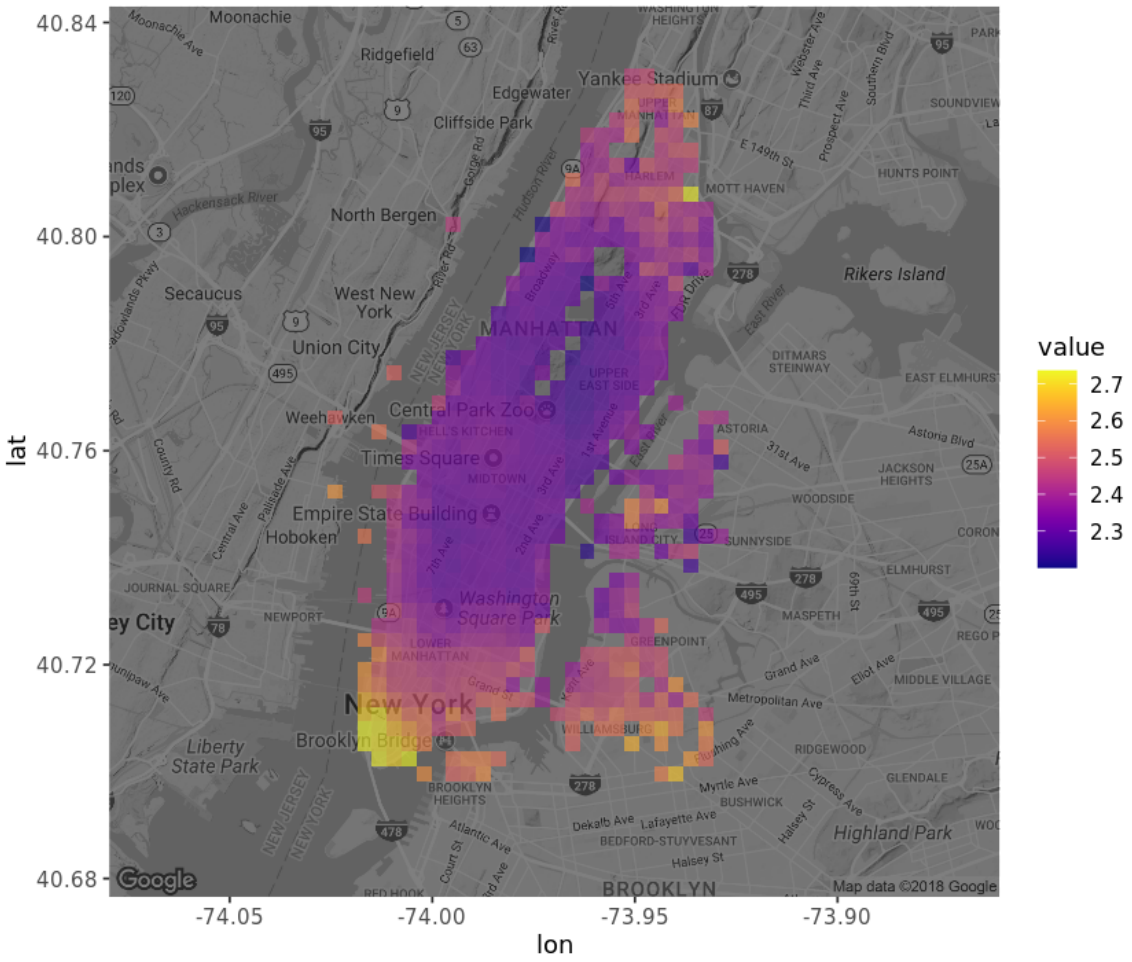
Now, let's take a look at the predictions of `fitted_forest` projected back onto Manhattan.

In [18]:

```
# 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,alpha=0.6,fun=mean)

 # .... COPY CODE FROM TASK 4 AND MODIFY HERE ....
```



In [19]:

```
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_total .")
    })
    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 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: 17.028 0.316 90.319 0.006 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

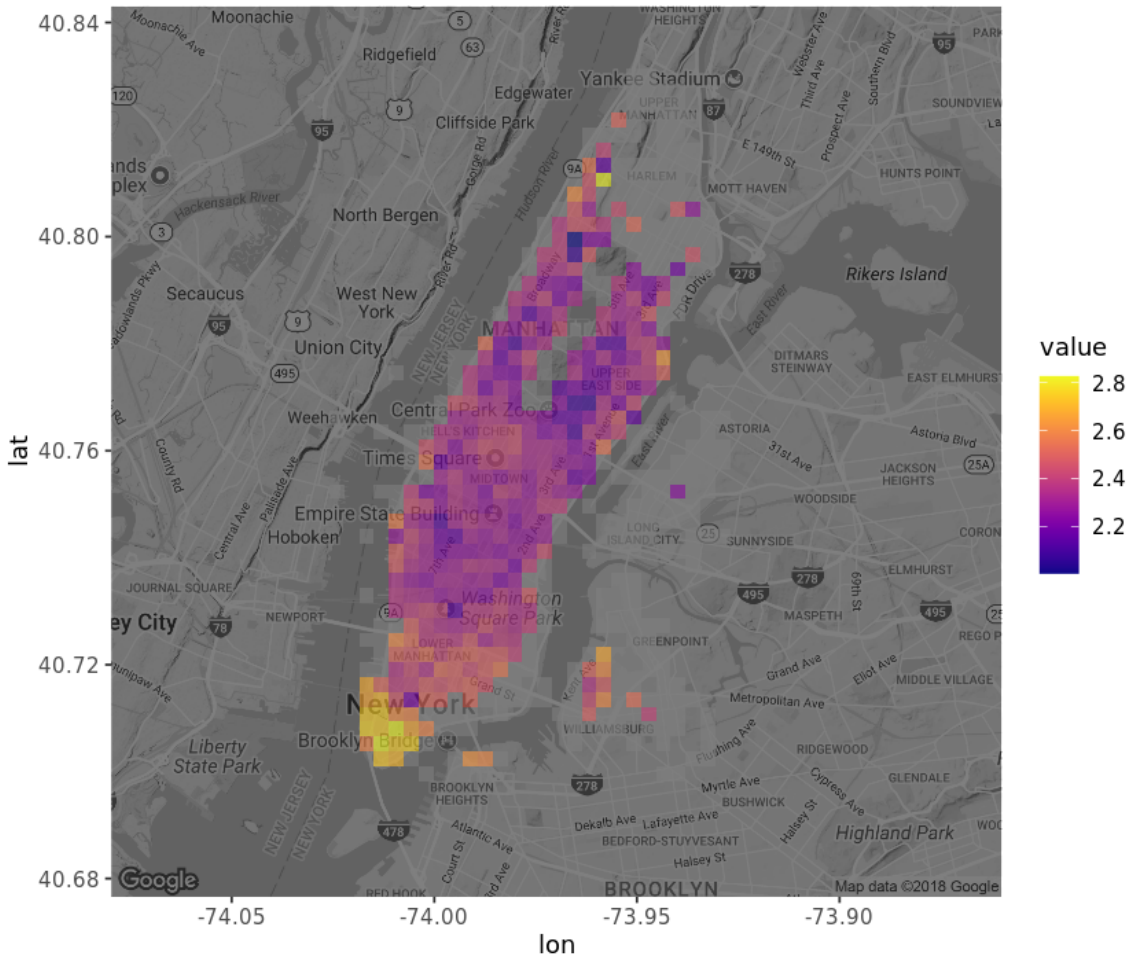
Looking at the map with the predicted fares we see that fares in downtown Manhattan are predicted to be high, while midtown is lower. This map only shows the prediction as a function of `lat` and `long`, but we could also plot the predictions over time, or a combination of time and space, but we'll leave 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 [20]:

```
# 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) +
scale_fill_viridis(option = 'plasma')+
stat_summary_2d(data=taxi,aes(x=long,y=lat,z=total),bins=60,alpha=0.6,fun="mean_if_enou
gh_data")
# .... COPY CODE FROM TASK 9 AND MODIFY HERE ....
```


lat
40.84
40.80
40.76
40.72
40.68
lon
-74.05
-74.00
-73.95
-73.90
value
2.8
2.6
2.4
2.2
Moonachie
Ridgefield
Edgewater
Yankee Stadium
Cliffside Park
North Bergen
Secaucus
West New York
Union City
Weehawken
Central Park Zoo
Times Square
Empire State Building
Hoboken
Washington Square Park
New York
Brooklyn Bridge
Liberty State Park
Rikers Island
MANHATTAN
HARLEM
MOTT HAVEN
HUNTS POINT
ASTORIA
DITMARS STEINWAY
EAST ELMHURST
JACKSON HEIGHTS
WOODSIDE
SUNNYSIDE
ELMHURST
MASPETH
GREENPOINT
MIDDLE VILLAGE
RIDGEWOOD
GLENDALE
BUSHWICK
BROOKLYN HEIGHTS
BEDFORD-STUYVESANT
Highland Park
BROOKLYN
Google
Map data ©2018 Google

In [21]:

```
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 correctly. Make su
re 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: 20.075 0.324 93.381 0.006 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 [22]:

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

In [23]:

```
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 downtown.")
  })
})
```

```
<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: 20.111 0.324 93.416 0.006 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)
```