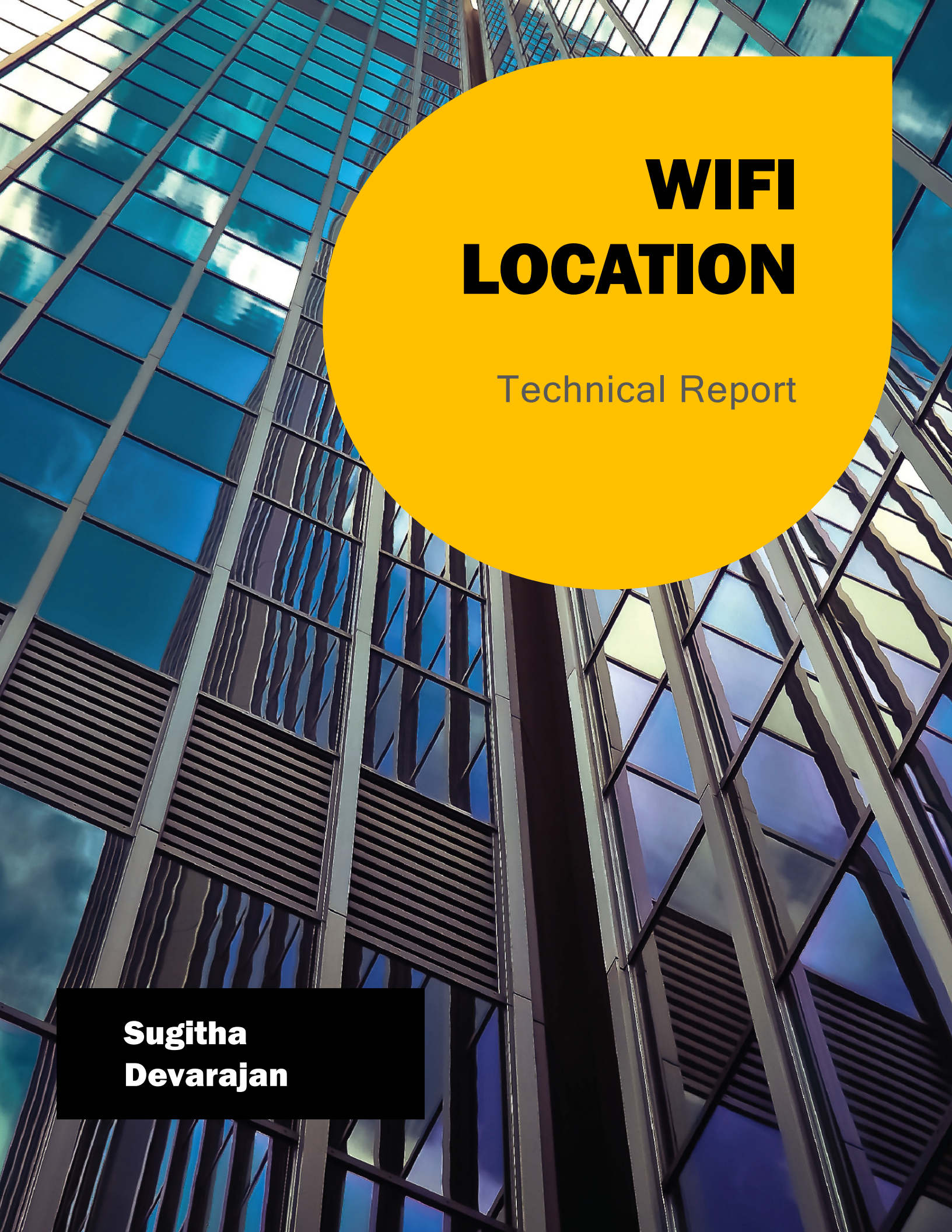


WIFI LOCATION

Technical Report

Sugitha Devarajan

Objective:

Use "WIFI fingerprinting" data to determine a person's location in indoor spaces. WIFI fingerprinting uses the signals from multiple WIFI hotspots within the building to determine the location.

Evaluate multiple machine learning models to see which produces the best result to determine the location.

Understanding Domain:

Indoor localization is still an open problem mainly due to the loss of GPS signal in indoor environments.

WLAN Fingerprint-based positioning systems are based on the **Received Signal Strength Indicator** (RSSI) value. Commonly, two phases are needed: calibration and operation. In the calibration phase, a radio map of the area where the users should be detected is constructed. Later, during the operational phase, a user obtains the signal strength of all visible access points of the WLAN that can be detected from his/her position and creates a test sample. This sample is sent to the server to be compared with the training samples of the radio map. Basically, the user’s location corresponds to the position associated with the most similar sample in the radio map.

One of the major advantages of the WLAN fingerprint-based methods is that they do not require the installation of any additional hardware since they use the existing WLAN

infrastructure. Therefore, the location of the user can be obtained without additional infrastructures and costs. However, WLANs were not natively designed to support a positioning function. Considering the existing obstacles introduced by the indoor environment (including reflections and multi path interference) the spread of radio signal in indoor environments is very hard to predict.

Data:

- It covers a surface of 108703m2
  including 3 buildings with 4 or 5 floors depending on the building.
- The number of different places (reference points) appearing in the database is 933.
- 21049 sampled points have been captured: 19938 for training/learning and 1111 for validation/testing.
- Dataset independence has been assured by taking Validation (or testing) samples 4 months after Training ones.
- The number of different wireless access points (WAPs) appearing in the database is 520.
- Data were collected by more than 20 users using 25 different models of mobile devices (some users used more than one model).

Data variables

001-520 RSSI levels - The most important information for WLAN fingerprinting comparison purposes are the WAPs detected and their RSSI level values. In the proposed database, this information represents the 98% of the data given in each record (520 vector positions out of 529) as a

520-element vector of integer values. These values represent the RSSI levels whereas the WAP identifiers (MAC addresses) are linked to the vector positions. The MAC addresses are coded as strings, and the RSSI levels correspond to negative integer values6 measured in dBm, where −100dBm is equivalent to a very weak signal, whereas 0dBM means that the detected WAP has an extremely good signal and 100 when WAP was not detected.
A total number of 520 WAPs appear in the database and the 520-element vector from each record contains the raw intensity levels of the detected WAPs from a single WiFi scan. Obviously, not all the WAPs are detected in each scan.

521-523 - Real world coordinates of the sample points
524 BuildingID
525 SpaceID
526 Relative position with respect to SpaceID
527 UserID
528 PhoneID
529 Timestamp

Preprocess data

- WAP001-520 are all int type
- Longitude and latitude are num type
- BuiidingID, Spaceid, relative position, userid, and phone id are all int type
- Timestamp needs conversion to date – converted to date and using Lubridate split the timestamp to year, day, month, week, weekday, hour and minute. Converting timestamp to int type.

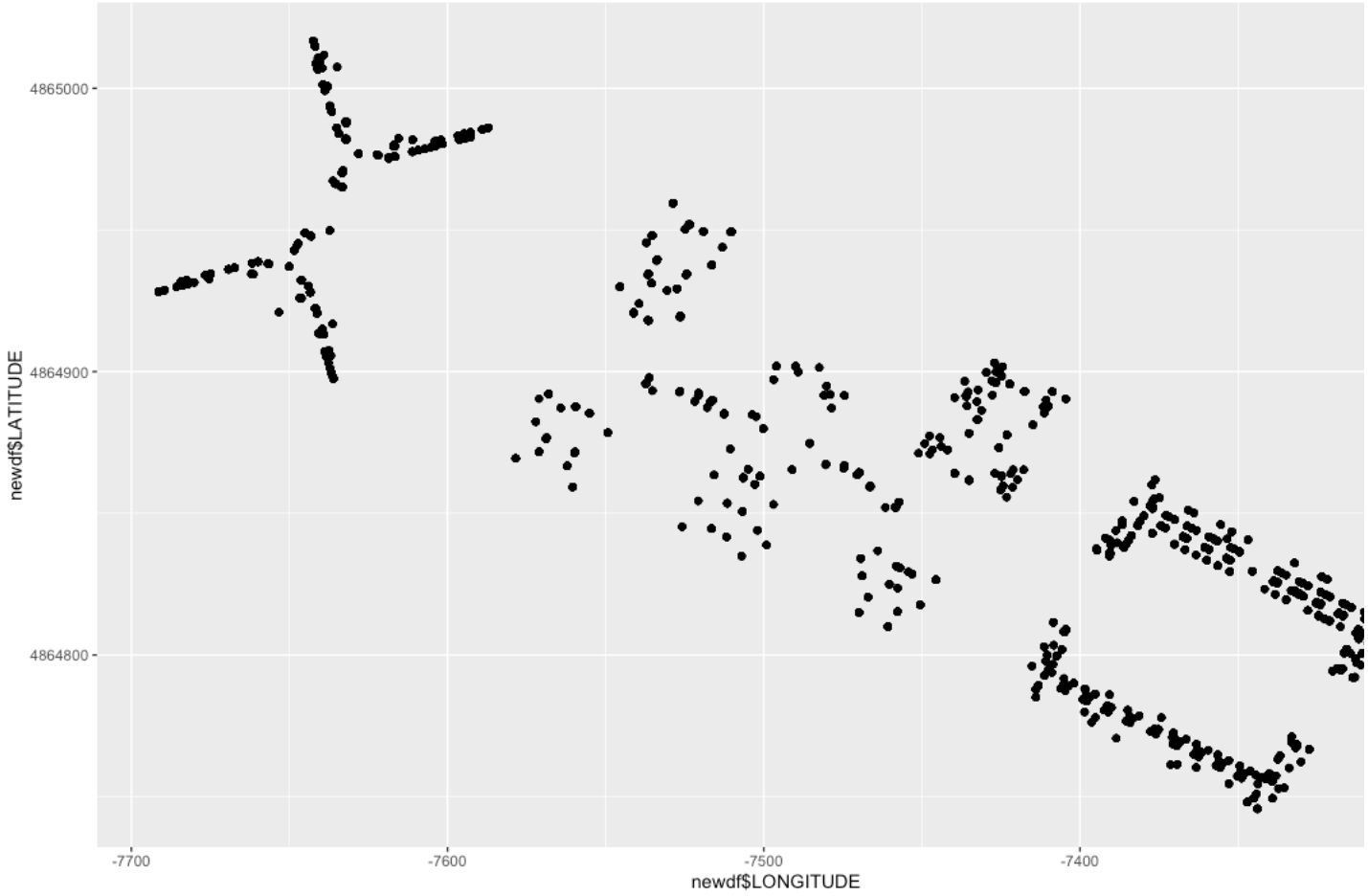
- No missing values - sum(is.na(newdf))
  [1] 0
- Moving all the columns from 521 to front of the data set so it will be easy to model.
- Summary of all columns except WAPs

```
> colnew<-c("month","day","week","weekDay","LONGITUDE","LATITUDE","FLOOR","BUILDINGID","SPACEID","RELATIVEPOSITION","USERID","PHONEID")
> summary(newdf[colnew])
     month            day             week          weekDay         LONGITUDE       LATITUDE          FLOOR         BUILDINGID       SPACEID     
 Min.   :5.000   Min.   : 4.00   Min.   :22.00   Min.   :0.0000   Min.   :-7691   Min.   :4864746   Min.   :0.000   Min.   :0.000   Min.   :  1.0  
 1st Qu.:6.000   1st Qu.:20.00   1st Qu.:24.00   1st Qu.:0.0000   1st Qu.:-7595   1st Qu.:4864821   1st Qu.:1.000   1st Qu.:0.000   1st Qu.:110.0  
 Median :6.000   Median :20.00   Median :25.00   Median :0.0000   Median :-7423   Median :4864852   Median :2.000   Median :1.000   Median :129.0  
 Mean   :5.925   Mean   :18.84   Mean   :24.52   Mean   :0.6551   Mean   :-7464   Mean   :4864871   Mean   :1.675   Mean   :1.213   Mean   :148.4  
 3rd Qu.:6.000   3rd Qu.:20.00   3rd Qu.:25.00   3rd Qu.:0.0000   3rd Qu.:-7359   3rd Qu.:4864930   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:207.0  
 Max.   :6.000   Max.   :31.00   Max.   :25.00   Max.   :4.0000   Max.   :-7301   Max.   :4865017   Max.   :4.000   Max.   :2.000   Max.   :254.0  
 RELATIVEPOSITION     USERID          PHONEID     
 Min.   :1.000    Min.   : 1.000   Min.   : 1.00  
 1st Qu.:2.000    1st Qu.: 5.000   1st Qu.: 8.00  
 Median :2.000    Median :11.000   Median :13.00  
 Mean   :1.833    Mean   : 9.068   Mean   :13.02  
 3rd Qu.:2.000    3rd Qu.:13.000   3rd Qu.:14.00  
 Max.   :2.000    Max.   :18.000   Max.   :24.00
```

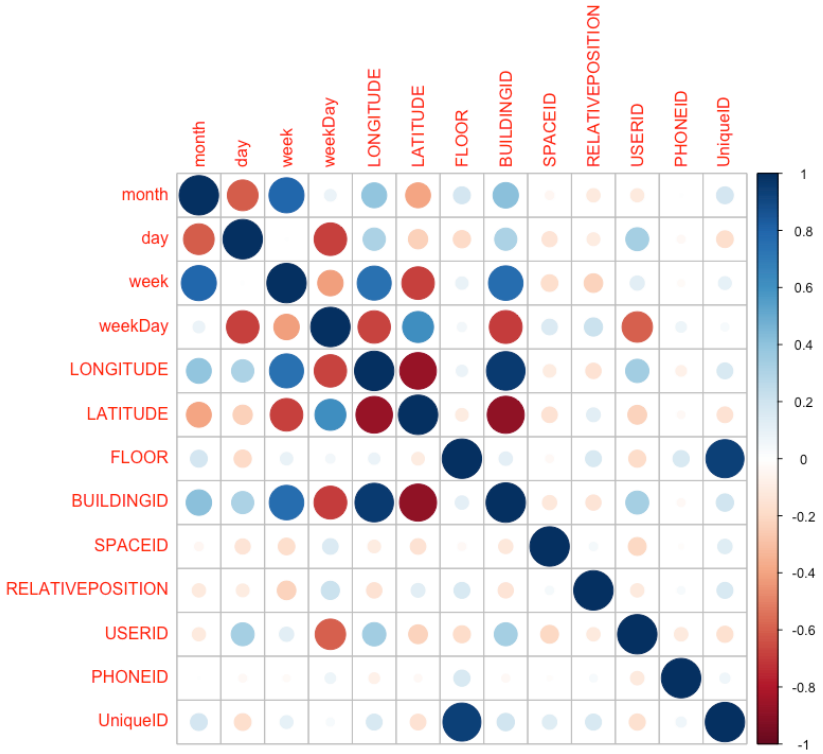
- There are duplicates
  > sum(duplicated(newdf))
  [1] 754
- Removing all columns and rows with only 100
  newdf <- newdf[, colSums(newdf != 100) > 0]
  newdf <- newdf[rowSums(newdf[,13:477] != 100) > 0,]
- Combine floor/building id/space id /relative position and convert to factor
- Check for duplicates in the unique id.
  > sum(table(newdf$UniqueID)-1)
  [1] 19062
  > length(unique(newdf$UniqueID))
  [1] 875
- Drop all the columns except all the WAPs, Building ID, and dependent variable.
- Filter the data by building ids and delete the building ID.
- Now data is ready for modeling

Exploratory data analysis with R

When you plot the Longitude and Latitude using ggplot geom_point() you can view the building layout

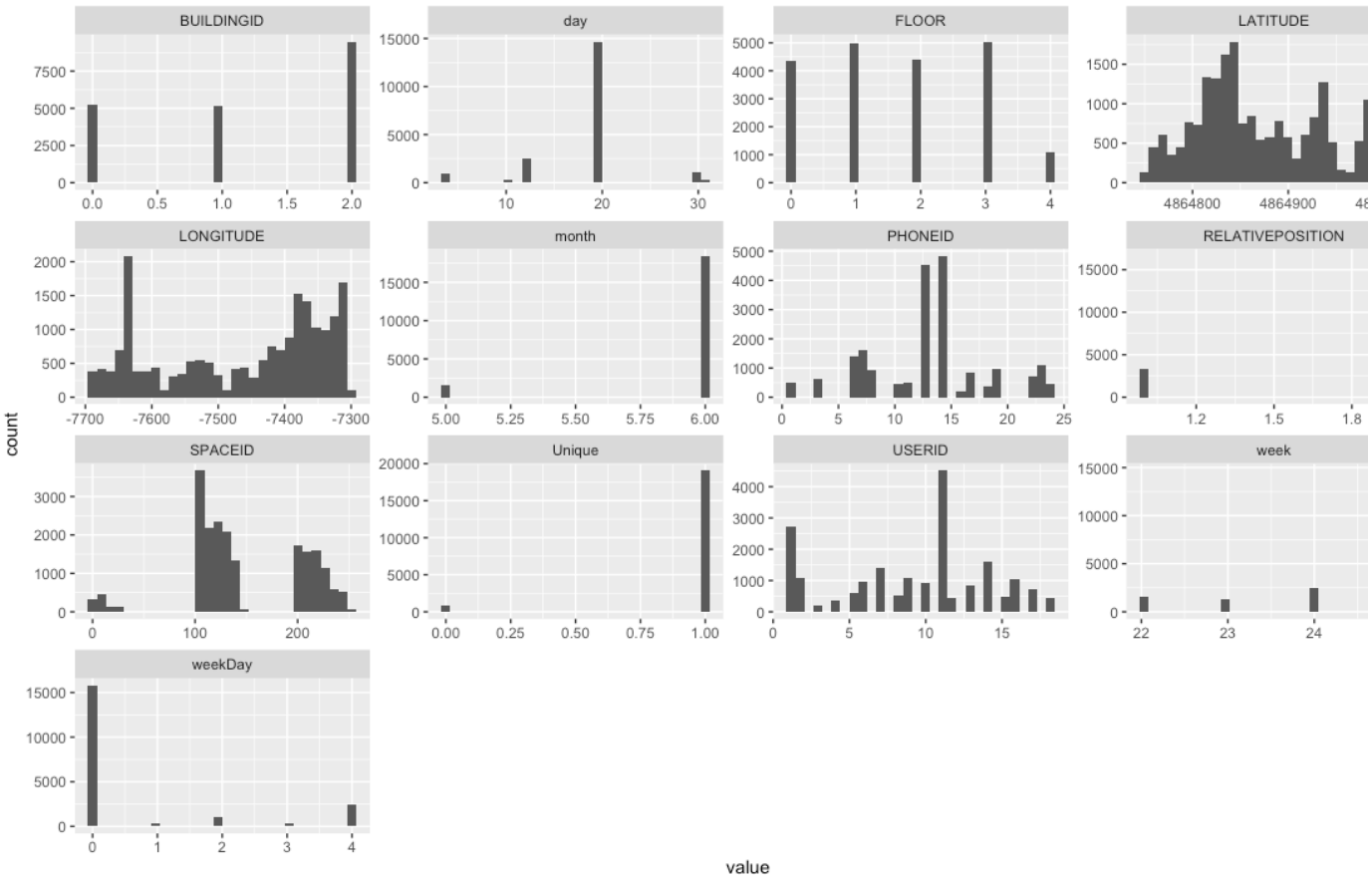


Plotting the correlation of certain columns shows the positive and negative relationship. We found unique id has strong positive correlation with Floor.



Basic Histogram

- Building 2 has more data
- Day 20/week 25/weekday 0/month 6 has max data
- Floor 4 has least data
- Relative position 2 has more data.



Model

- Decision Tree
  Decision Trees are versatile Machine Learning algorithm that can perform both classification and regression tasks. They are very powerful algorithms, capable of fitting complex datasets. Besides, decision trees are fundamental components of random forests, which are among the most potent Machine Learning algorithms available today.
  Advantageous of Decision Trees
  - Easy Interpretation
  - Making prediction is fast
  - Easy to identify important variables
  - Handless missing data

    - One of the drawbacks is to can have high variability in performance.
    - Recursive portioning- basis can achieve maximum homogeneity within the new partition.

- C5.0

While there are numerous implementations of decision trees, one of the most well-known is the C5.0 algorithm. The C5.0 algorithm has become the industry standard for producing decision trees, because it does well for most types of problems directly out of the box. Compared to more advanced and sophisticated machine learning models (e.g., Neural Networks and Support Vector Machines), the decision trees under the C5.0 algorithm generally perform nearly as well but are much easier to understand and deploy.

- C5.0 algorithm is a successor of C4.5 algorithm also developed by Quinlan (1994)
- Gives a binary tree or multi branches tree
- Uses Information Gain (Entropy) as its splitting criteria.
- C5.0 pruning technique adopts the Binomial Confidence Limit method.
- n a case of handling missing values, C5.0 allows to whether estimate missing values as a function of other attributes or apportions the case statistically among the results.

- RF

Random forests are based on a simple idea: 'the wisdom of the crowd'. Aggregate of the results of multiple predictors gives a better prediction than the best individual predictor. A group of

predictors is called an **ensemble**. Thus, this technique is called **Ensemble Learning**.

In earlier tutorial, you learned how to use **Decision trees** to make a binary prediction. To improve our technique, we can train a group of **Decision Tree classifiers**, each on a different random subset of the train set. To make a prediction, we just obtain the predictions of all individuals trees, then predict the class that gets the most votes. This technique is called **Random Forest**.

- KNN

K-Nearest Neighbor or K-NN is a Supervised Non-linear classification algorithm. K-NN is a non-parametric algorithm i.e. it doesn't make any assumption about underlying data or its distribution. It is one of the simplest and widely used algorithm which depends on it's k value (Neighbors) and finds it's applications in many industries like finance industry, healthcare industry etc.

Theory

In the KNN algorithm, K specifies the number of neighbors, and its algorithm is as follows:

- Choose the number K of neighbor.
- Take the K Nearest Neighbor of unknown data point according to distance.
- Among the K-neighbors, Count the number of data points in each category.
- Assign the new data point to a category, where you counted the most neighbors.

The euclidian distance used for calculating the distance between k neighbors and some of the variables have different magnitudes, so standardization is important.

Some of the popular application examples are

- Recommendation system
- Loan Approval
- Anamoly Detection
- Text Categorization
- Finance
- Medicine

Kappa Score

Kappa Score compares an Observed Accuracy with an Expected Accuracy. Observed Accuracy is simply the number of instances that were classified correctly. Expected Accuracy is defined as the accuracy that any random classifier would be expected to achieve.

Steps to model:

1. Split the data using create data partition in to train and test for all building types
2. Add a control of 10 folds
3. Set seed
4. Train the model using all 4 models
5. Predict using test data
6. Find the accuracy and kappa

1. Run Decision Tree

Building 0

```
> postResample(treeb0.unique.predicted,
wifib0.testing$UniqueID)
  Accuracy      Kappa
0.03550296 0.02967035
```

Building 1

```
> postResample(treeb1.unique.predicted,
wifib1.testing$UniqueID)
  Accuracy      Kappa
0.07289596 0.05673889
```

Building 2

```
> postResample(treeb2.unique.predicted,
wifib2.testing$UniqueID)
  Accuracy      Kappa
0.01829925 0.01127450
```

2. Run C5.0

Building 0

```
> postResample(prediction_C50_b0, wifib0.testing$UniqueID)
 Accuracy     Kappa
0.7100592 0.7088794
```

Building 1

```
> postResample(prediction_C50_b1, wifib1.testing$UniqueID)
 Accuracy     Kappa
```

0.8018555 0.8006759

Building 2

> postResample(prediction_C50_b2, wifib2.testing$UniqueID)

Accuracy Kappa

0.7477574 0.7469829

3. Run KNN

Building 0

> postResample(prediction_KNN_b0, wifib0.testing$UniqueID)

Accuracy Kappa

0.5404339 0.5385734

Building 1

> postResample(prediction_KNN_b1, wifib1.testing$UniqueID)

Accuracy Kappa

0.6620278 0.6599821

Building 2

> postResample(prediction_KNN_b2, wifib2.testing$UniqueID)

Accuracy Kappa

0.6318622 0.6307197

4. Random forest

RF fit plot with training data

Building 0

```
> postResample(preds_b0_rf, wifib0.testing$UniqueID)
 Accuracy     Kappa
0.7666009 0.7656493
```

Building 1

```
> postResample(preds_b1_rf, wifib1.testing$UniqueID)
 Accuracy     Kappa
0.8621604 0.8613265
```

Building 2

```
> postResample(preds_b2_rf, wifib2.testing$UniqueID)
 Accuracy     Kappa
0.8310011 0.8304796
```

Building 0

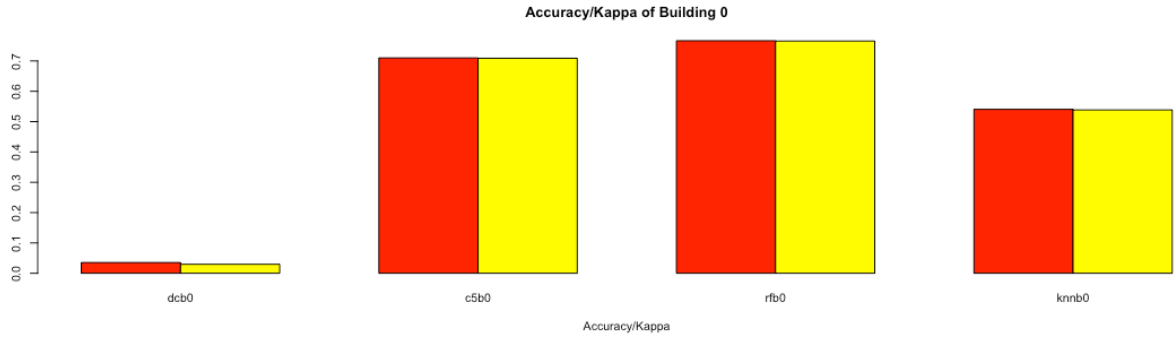
```
> summary(resample_results)

Call:
summary.resamples(object = resample_results)

Models: CART, RF, C5.0, KNN
Number of resamples: 10

Accuracy
           Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
CART 0.01907357 0.02955818 0.03281619 0.03278814 0.03466667 0.05013928    0
RF   0.72105263 0.74472986 0.75032807 0.75429760 0.76783743 0.78048780    0
C5.0 0.65600000 0.69505897 0.69971938 0.69931183 0.70502632 0.75000000    0
KNN  0.49046322 0.51816913 0.53535872 0.53531132 0.55455652 0.57692308    0

Kappa
           Min.    1st Qu.     Median      Mean    3rd Qu.       Max. NA's
CART 0.01262247 0.02393332 0.02734166 0.0271279 0.02914531 0.04439292    0
RF   0.71983028 0.74364101 0.74925779 0.7532285 0.76682126 0.77951135    0
C5.0 0.65451364 0.69374707 0.69842070 0.6980174 0.70375105 0.74893767    0
KNN  0.48831678 0.51612645 0.53338024 0.5333506 0.55268293 0.57511767    0
```

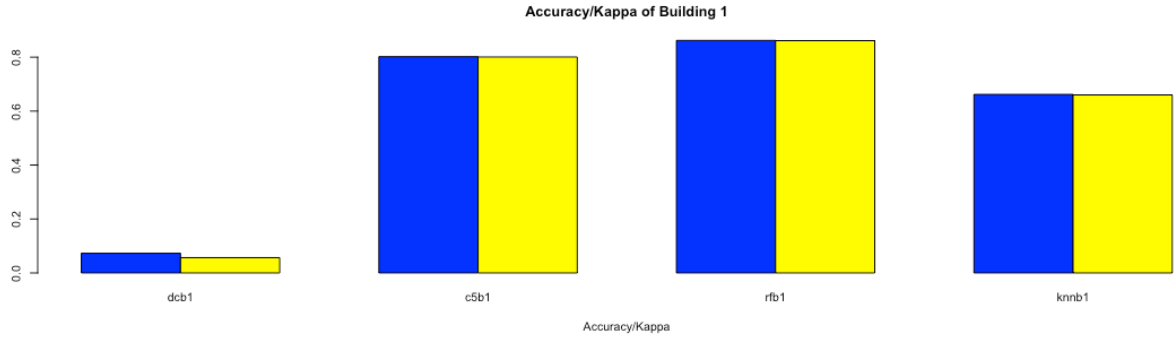
Building 1

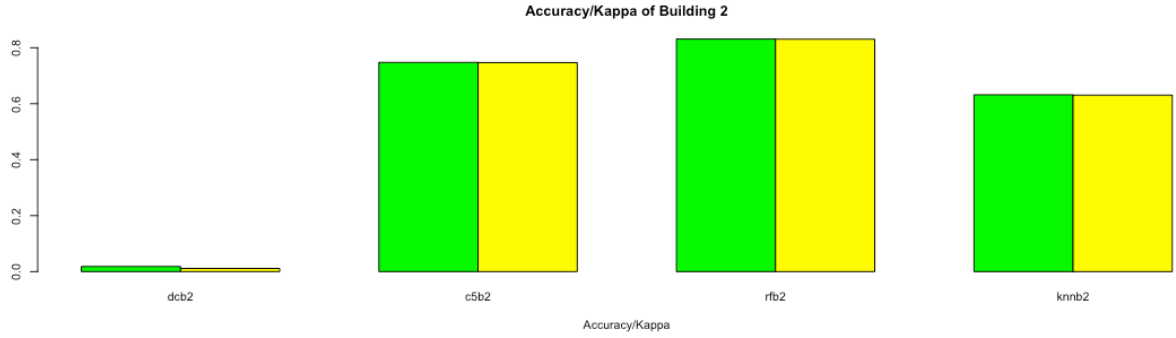
```
> summary(resample_results1)

Call:
summary.resamples(object = resample_results1)

Models: CART, RF, C5.0, KNN
Number of resamples: 10

Accuracy
           Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
CART 0.03252033 0.03435488 0.04813721 0.04510668 0.05102346 0.06233062    0
RF   0.81471390 0.83275073 0.83975182 0.84229565 0.85631659 0.88055556    0
C5.0 0.75068493 0.77068849 0.77870047 0.77788157 0.78709438 0.79722222    0
KNN  0.60000000 0.61801073 0.62371190 0.62472597 0.63438249 0.65155807    0

Kappa
           Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
CART 0.01652893 0.01721202 0.03225826 0.02855826 0.03368253 0.04482815    0
RF   0.81353437 0.83169902 0.83875170 0.84130515 0.85543464 0.87979500    0
C5.0 0.74919583 0.76929454 0.77726645 0.77651412 0.78578239 0.79597699    0
KNN  0.59765038 0.61568333 0.62137952 0.62243172 0.63218297 0.64924427    0
```

Building 2

```
> summary(resample_results2)

Call:
summary.resamples(object = resample_results2)

Models: CART, RF, C5.0, KNN
Number of resamples: 10

Accuracy
           Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
CART 0.01515152 0.02413551 0.02860474 0.02789571 0.03358865 0.03680982    0
RF   0.76586103 0.77692938 0.79043854 0.78970330 0.79753861 0.82202112    0
C5.0 0.66917293 0.69681501 0.71353993 0.70462180 0.71602141 0.72023810    0
KNN  0.55970149 0.57608455 0.60665133 0.60674109 0.63351672 0.66517857    0

Kappa
            Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
CART 0.007633588 0.01850838 0.02248892 0.02170705 0.02782492 0.03064394    0
RF   0.765104743 0.77621025 0.78976156 0.78902504 0.79688559 0.82144430    0
C5.0 0.668126788 0.69586918 0.71262132 0.70368444 0.71512553 0.71934314    0
KNN  0.558268744 0.57473641 0.60540981 0.60548094 0.63234600 0.66413062    0
```


Accuracy/Kappa of Building 0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
dcb0
c5b0
rfb0
knnb0
Accuracy/Kappa


Accuracy/Kappa of Building 1
0.0
0.2
0.4
0.6
0.8
dcb1
c5b1
rfb1
knnb1
Accuracy/Kappa


Accuracy/Kappa of Building 2
0.0
0.2
0.4
0.6
0.8
dcb2
c5b2
rfb2
knnb2
Accuracy/Kappa

I recommend **C5** algorithm for the WIFI location prediction because commutation time was much faster than random forest.

The dataset provided had less unique locations which makes this an imbalanced classification problem where the distribution of data across the known classes is biased or skewed. Maybe with more unique location we could build the possible indoor location model with most accuracy and kappa.