Predicting Aggregate Interference

Carson Slater

Data Preparation and EDA

The following code is the result of triage for what we thought was important for reproducibility of all the research that went into this paper. Other R scripts such as `analysis0.R`, `analysis1.R`, `analysis2.R` contain the messy, yet authentic code that was the outpouring of this modeling process. When someone is attempting to discover something they have quick ideas they don't wanna forget so they just keep coding them up and their organization takes a backseat to discovery and inquiry.

Data Preparation

```
save <- FALSE

tuning_data <- read.csv(here::here("Data", "MamdaniTuningData_1-6.csv"),
                        header = TRUE)

distance <- as.vector(as.matrix(tuning_data[ ,c("tx1_distance",
                                                "tx2_distance",
                                                "tx3_distance")]))

input_psd <- as.vector(as.matrix(tuning_data[ ,c("tx1_psd",
                                                 "tx2_psd",
                                                 "tx3_psd")]))

scaled_psd <- as.vector(as.matrix(tuning_data[ ,c("tx1_scaled_psd",
                                                  "tx2_scaled_psd",
                                                  "tx3_scaled_psd")]))

aggregate_data <- cbind.data.frame(rep(tuning_data$rad_loc_idx, 3),
                                   distance,
                                   input_psd,
```

```
                                                    scaled_psd)
```

We found it difficult to find real-world data for this kind of problem, so we have simulated data. To make our model robust, we decided to simulate random noise in our outcome of interest, `scaled_psd` (recieved power spectral density). We opted to assume a gaussian probability distribution with a mean of zero and a priori illicited variance. So to find the variance, we know the noise density (variance) at room temp is -174 dBm/Hz. We can convert to dBW/200MHz by using the following formula

$$\sqrt{|-174 + 10\log_{10}(2 \times 10^8) - 30|} = 10.99953$$

. We take the square root of the absolute value and then we have the standard deviation to parameterize the distribution of noise for the outcome of interest. So the noise is $\mathcal{N}(0, 10.99)$.

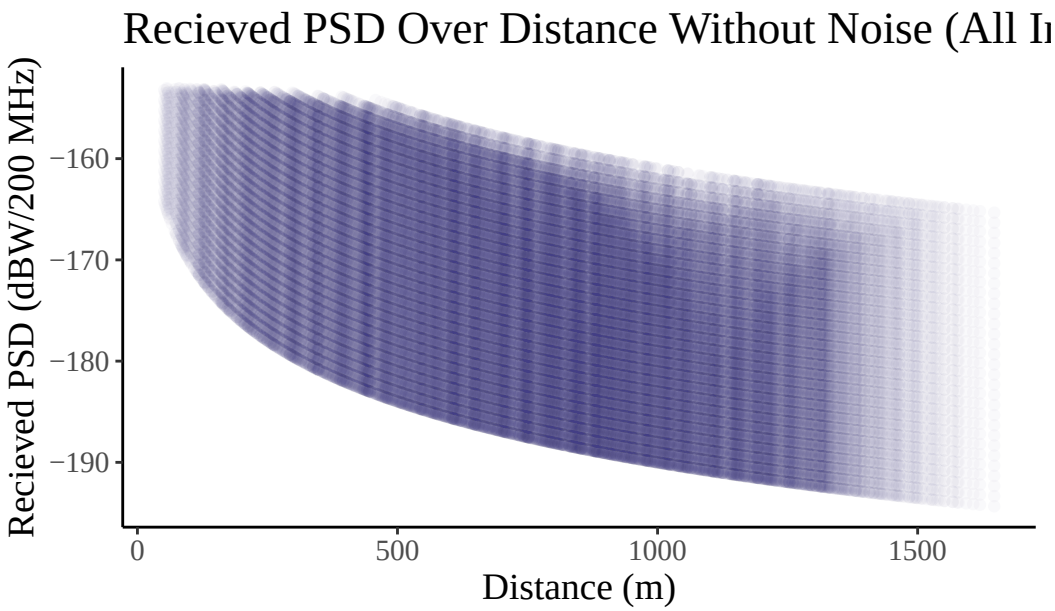
```
# generating potential transformation for the `distance` variable
# see analyis2.R for other transformations we tried
aggregate_data <- aggregate_data |>
  mutate(transform5 = distance^(-1/1000))

# adding noise
set.seed(613)

sigma <- sqrt(abs(-174 + 10*log10(200e6) - 30))

noise <- rnorm(nrow(aggregate_data), 0, sigma)

noise_aggregate_data <- aggregate_data

noise_aggregate_data$scaled_psd <- noise_aggregate_data$scaled_psd + noise
```

Data Visualization

In this our exploratory data analysis, I observed there is a non-linear relationship between `distance` and the `scaled_psd`.

```
viridis_pal(option = "D")(12)
#  [1] "#440154FF" "#482173FF" "#433E85FF" "#38598CFF" "#2D708EFF" "#25858EFF"
#  [7] "#1E9B8AFF" "#2BB07FFF" "#51C56AFF" "#85D54AFF" "#C2DF23FF" "#FDE725FF"

# No noise
if (save == TRUE) pdf(file = "Figures/rawdata.pdf", height = 5, width = 7)
aggregate_data |>
```

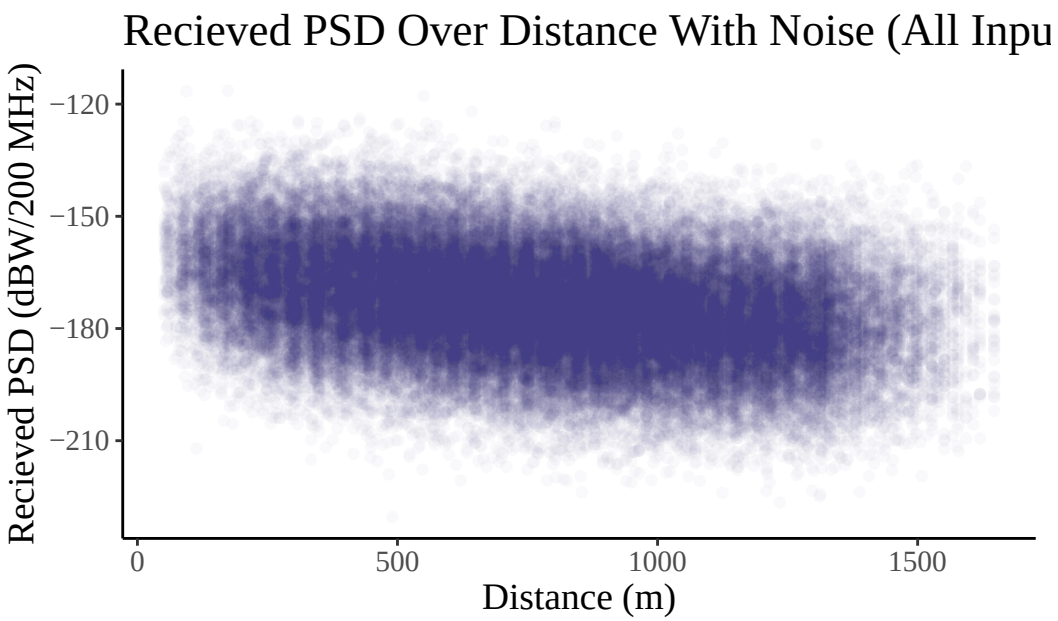
```
  ggplot(aes(distance, scaled_psd)) +
  geom_point(color = "#433E85FF", alpha = 0.03) +
  labs(title = "Recieved PSD Over Distance Without Noise (All Input PSD)",
       caption = "Raw simulated data plotted before any added noise without the distance v
  xlab(TeX("Distance (m)")) +
  ylab("Recieved PSD (dBW/200 MHz)") +
  theme(text=element_text(family="Times New Roman", size=14),
        plot.caption = element_text(hjust=0.5))
```



Raw simulated data plotted before any added noise without the distance variable transf

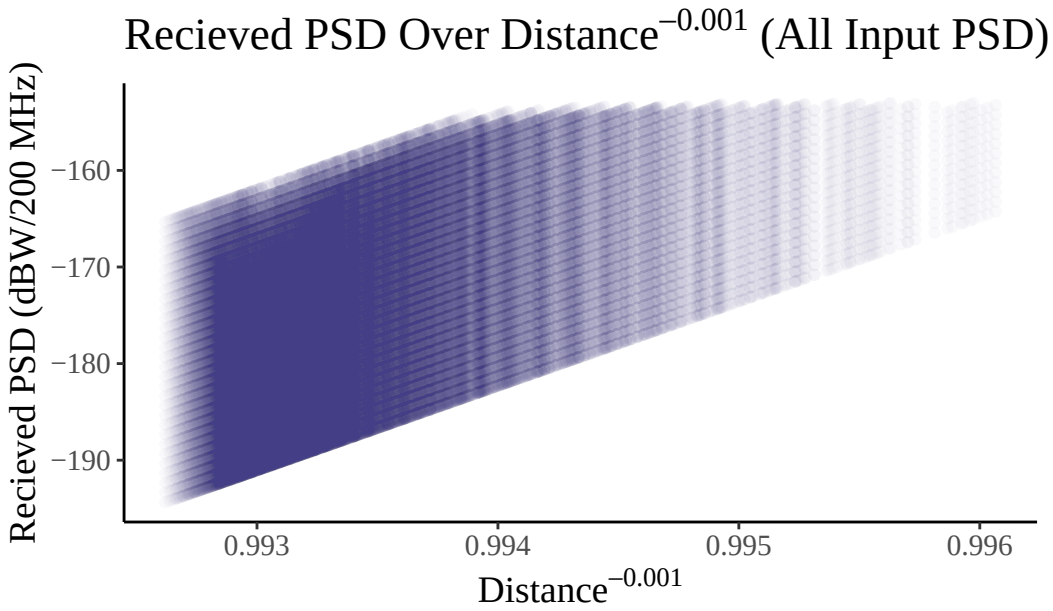
```
if (save == TRUE) dev.off()

 # Jitter (noise) added
if (save == TRUE) pdf(file = "Figures/noise_rawdata.pdf", height = 5, width = 7)
noise_aggregate_data |>
  ggplot(aes(distance, scaled_psd)) +
  geom_point(color = "#433E85FF", alpha = 0.03) +
  labs(title = "Recieved PSD Over Distance With Noise (All Input PSD)",
       caption = "Raw simulated data plotted with simulated noise without the distance var
  xlab("Distance (m)") +
  ylab("Recieved PSD (dBW/200 MHz)") +
  theme(text=element_text(family="Times New Roman", size=14),
```

```
        plot.caption = element_text(hjust=0.5))
```



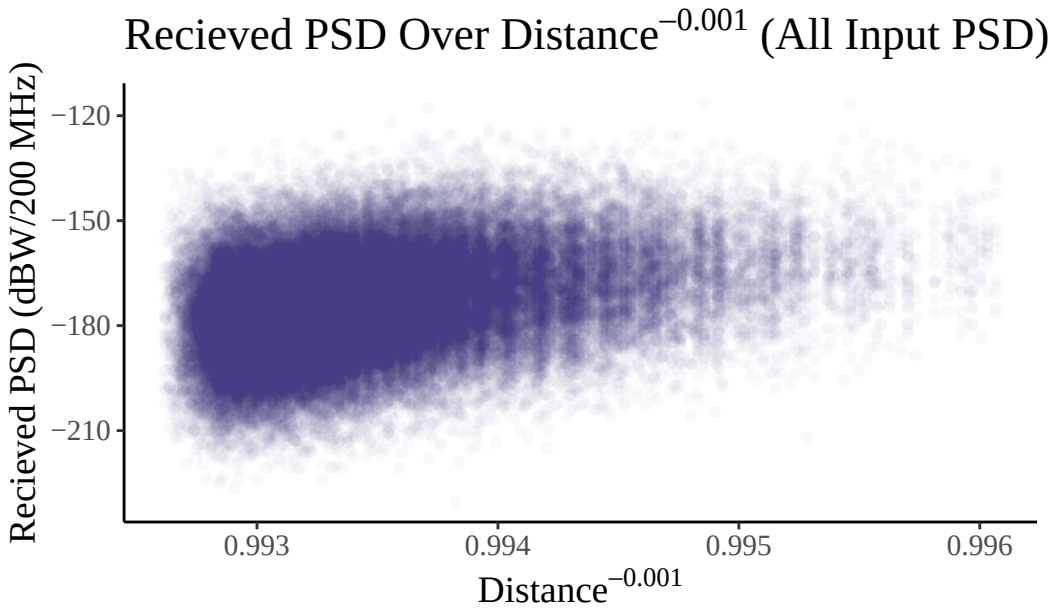
Raw simulated data plotted with simulated noise without the distance variable transfo

```
if (save == TRUE) dev.off()

# No noise + transformed distance
if (save == TRUE) pdf(file = "Figures/tfdata.pdf", height = 5, width = 7)
aggregate_data |>
  ggplot(aes(transform5, scaled_psd)) +
  geom_point(color = "#433E85FF", alpha = 0.03) +
  labs(title = TeX("Recieved PSD Over Distance$^{-0.001}$ (All Input PSD)"),
       caption = "Data plotted before any added noise with the Distance variable transform
  xlab(TeX("Distance$^{-0.001}$")) +
  ylab("Recieved PSD (dBW/200 MHz)") +
  theme(text=element_text(family="Times New Roman", size=14),
        plot.caption = element_text(hjust=0.5))
```



Data plotted before any added noise with the Distance variable transformed.

```
if (save == TRUE) dev.off()

# Noise + transformed distance
if (save == TRUE) pdf(file = "Figures/noise_tfdata.pdf", height = 5, width = 7)
noise_aggregate_data |>
  ggplot(aes(transform5, scaled_psd)) +
  geom_point(color = "#433E85FF", alpha = 0.03) +
  labs(title = TeX("Recieved PSD Over Distance$^{-0.001}$ (All Input PSD)"),
       caption = "Data plotted after adding noise and transforming the distiance variable.
  xlab(TeX("Distance$^{-0.001}$")) +
  ylab("Recieved PSD (dBW/200 MHz)") +
  theme(text=element_text(family="Times New Roman", size=14),
        plot.caption = element_text(hjust=0.5))
```



Data plotted after adding noise and transforming the distiance variable.

```
if (save == TRUE) dev.off()
```

Modeling

We create a `parsnip` linear model object.

```
lm_mod <-
  parsnip::linear_reg() |>
  parsnip::set_engine("lm") |>
  parsnip::set_mode("regression")
```

The Model (Without Noise)

```
set.seed(613)
mod_split <- aggregate_data |>
  initial_split(
    prop = 0.8
  )

mod_test <- testing(mod_split)
```

```r
mod_train <- training(mod_split)

# `tidymodels` Procedure for Fitting Linear Model.
my_recipe <- recipe(scaled_psd ~ input_psd + transform5,
                    data = aggregate_data)

baked_recipe <- bake(prep(my_recipe), new_data = NULL)

lm_wkflow <- workflow() |>
  add_model(lm_mod) |>
  add_recipe(my_recipe)

# Specifying Model Metrics
my_metrics <- metric_set(mape, rsq, rmse)

# Fit the model (not with v-fold CV)
lm_fit <- fit(lm_wkflow,
              data = mod_train)

lm_pred_test <- augment(lm_fit,
                        new_data = mod_test)

my_metrics(lm_pred_test,
           truth = scaled_psd,
           estimate = .pred)
# # A tibble: 3 x 3
#   .metric .estimator .estimate
#   <chr>   <chr>          <dbl>
# 1 mape    standard    0.000834
# 2 rsq     standard    1.00
# 3 rmse    standard    0.00197

# THE MODEL ------------------------------------------------------------------
lm_fit |>
  tidy()
# # A tibble: 3 x 5
#   term        estimate  std.error statistic p.value
#   <chr>          <dbl>      <dbl>     <dbl>   <dbl>
# 1 (Intercept) -8800.   0.0146      -602420.       0
# 2 input_psd       1.00 0.00000105   949632.       0
# 3 transform5   8740.   0.0147       594317.       0
```

```r
m1 <- lm(scaled_psd ~ input_psd + transform5,
         data = mod_train)
summary(m1)
#
# Call:
# lm(formula = scaled_psd ~ input_psd + transform5, data = mod_train)
#
# Residuals:
#        Min         1Q     Median         3Q        Max
# -0.0202641 -0.0009867  0.0004971  0.0014579  0.0021630
#
# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept) -8.800e+03  1.461e-02 -602420   <2e-16 ***
# input_psd    1.000e+00  1.053e-06  949632   <2e-16 ***
# transform5   8.740e+03  1.471e-02  594317   <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.001997 on 56853 degrees of freedom
# Multiple R-squared:      1,   Adjusted R-squared:      1
# F-statistic: 6.005e+11 on 2 and 56853 DF,  p-value: < 2.2e-16
```

Cross Validation (No Noise)

```r
# Creating CV Folds to Confirm Model Metrics
lm_folds <- vfold_cv(mod_train, v = 10, repeats = 5)

# Fitting the Model (with CV)

lm_folds_fitted <- lm_wkflow |>
  fit_resamples(resamples = lm_folds,
                metrics = my_metrics)

# collect_metrics(lm_folds_fitted, summarize = F) |>
#   filter(.metric == "mape") |>
#   arrange(desc(.estimate)) |>
#   kable()

collect_metrics(lm_folds_fitted, summarize = T) |>
```

```
  select(.metric, mean, n, std_err) |>
  kable()
```

.metric	mean	n	std_err
mape	0.0008335	50	1.4e-06
rmse	0.0019968	50	7.0e-06
rsq	1.0000000	50	0.0e+00

Benchmarking

```
# Benchmark ------------------------------------------------------------------
columns <- c("Expression",
             "Min. Runtime",
             "Med. Runtime",
             "Itr/Sec",
             "Memory Allocated")

bnch <- bench::mark(tidymodels = lm_wkflow |>
                      fit_resamples(resamples = lm_folds, metrics = my_metrics),
                    check = FALSE)
# Warning: Some expressions had a GC in every iteration; so filtering is
# disabled.
bnch |>
  select(expression, min,
         median, `itr/sec`,
         mem_alloc) |>
  knitr::kable(digits = 3,
               col.names = columns) |>
  kableExtra::kable_styling(font_size = 10,
                            latex_options = "HOLD_position")
```

Expression	Min. Runtime	Med. Runtime	Itr/Sec	Memory Allocated
tidymodels	1.58s	1.58s	0.634	925MB

The Model (With Noise)

```
# Performing Model Training

set.seed(613)
```

```r
noise_mod_split <- noise_aggregate_data |>
  initial_split(
    prop = 0.8
  )

noise_mod_test <- testing(noise_mod_split)
noise_mod_train <- training(noise_mod_split)

# `tidymodels` Procedure for Fitting Linear Model.
noise_my_recipe <- recipe(scaled_psd ~ input_psd + transform5,
                    data = aggregate_data)

noise_baked_recipe <- bake(prep(my_recipe), new_data = NULL)

noise_lm_wkflow <- workflow() |>
  add_model(lm_mod) |>
  add_recipe(noise_my_recipe)

# Specifying Model Metrics
noise_my_metrics <- metric_set(mape, rsq, rmse)

# Fit the model (not with v-fold CV)
noise_lm_fit <- fit(lm_wkflow,
              data = noise_mod_train)

noise_lm_pred_test <- augment(noise_lm_fit,
                        new_data = noise_mod_test)

noise_my_metrics(noise_lm_pred_test,
           truth = scaled_psd,
           estimate = .pred)
# # A tibble: 3 x 3
#   .metric .estimator .estimate
#   <chr>   <chr>          <dbl>
# 1 mape    standard       5.11
# 2 rsq     standard       0.401
# 3 rmse    standard      11.0

# THE MODEL
noise_lm_fit |>
  tidy()
```

```
# # A tibble: 3 x 5
#   term         estimate std.error statistic p.value
#   <chr>           <dbl>     <dbl>     <dbl>   <dbl>
# 1 (Intercept) -8883.     80.2         -111.       0
# 2 input_psd       0.995   0.00578      172.       0
# 3 transform5   8824.     80.7          109.       0

m2 <- lm(scaled_psd ~ input_psd + transform5,
         data = noise_mod_train)
summary(m2)
#
# Call:
# lm(formula = scaled_psd ~ input_psd + transform5, data = noise_mod_train)
#
# Residuals:
#     Min      1Q  Median      3Q     Max
# -48.539  -7.400   0.037   7.353  43.427
#
# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept) -8.883e+03  8.019e+01  -110.8   <2e-16 ***
# input_psd    9.951e-01  5.781e-03   172.1   <2e-16 ***
# transform5   8.824e+03  8.073e+01   109.3   <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 10.96 on 56853 degrees of freedom
# Multiple R-squared:  0.4116,  Adjusted R-squared:  0.4116
# F-statistic: 1.989e+04 on 2 and 56853 DF,  p-value: < 2.2e-16
```

Cross Validation (With Noise)

```
# Creating CV Folds to Confirm Model Metrics
noise_lm_folds <- vfold_cv(noise_mod_train, v = 10, repeats = 5)

# Fitting the Model (with CV)

noise_lm_folds_fitted <- noise_lm_wkflow |>
  fit_resamples(resamples = noise_lm_folds,
                metrics = my_metrics)
```

```
# collect_metrics(noise_lm_folds_fitted, summarize = F) |>
#   filter(.metric == "mape") |>
#   arrange(desc(.estimate)) |>
#   kable()

collect_metrics(noise_lm_folds_fitted, summarize = T) |>
  select(.metric, mean, n, std_err) |>
  kable()
```

.metric	mean	n	std_err
mape	5.0867431	50	0.0067476
rmse	10.9642469	50	0.0136898
rsq	0.4116061	50	0.0013397

Benchmarking

```
# Benchmark ----------------------------------------------------------------
columns <- c("Expression",
             "Min. Runtime",
             "Med. Runtime",
             "Itr/Sec",
             "Memory Allocated")

bnch <- bench::mark(tidymodels = noise_lm_wkflow |>
                      fit_resamples(resamples = noise_lm_folds, metrics = my_metrics),
                    check = FALSE)
# Warning: Some expressions had a GC in every iteration; so filtering is
# disabled.
bnch |>
  select(expression, min,
         median, `itr/sec`,
         mem_alloc) |>
  knitr::kable(digits = 3,
               col.names = columns) |>
  kableExtra::kable_styling(font_size = 10,
                            latex_options = "HOLD_position")
```

Expression	Min. Runtime	Med. Runtime	Itr/Sec	Memory Allocated
tidymodels	1.61s	1.61s	0.622	925MB

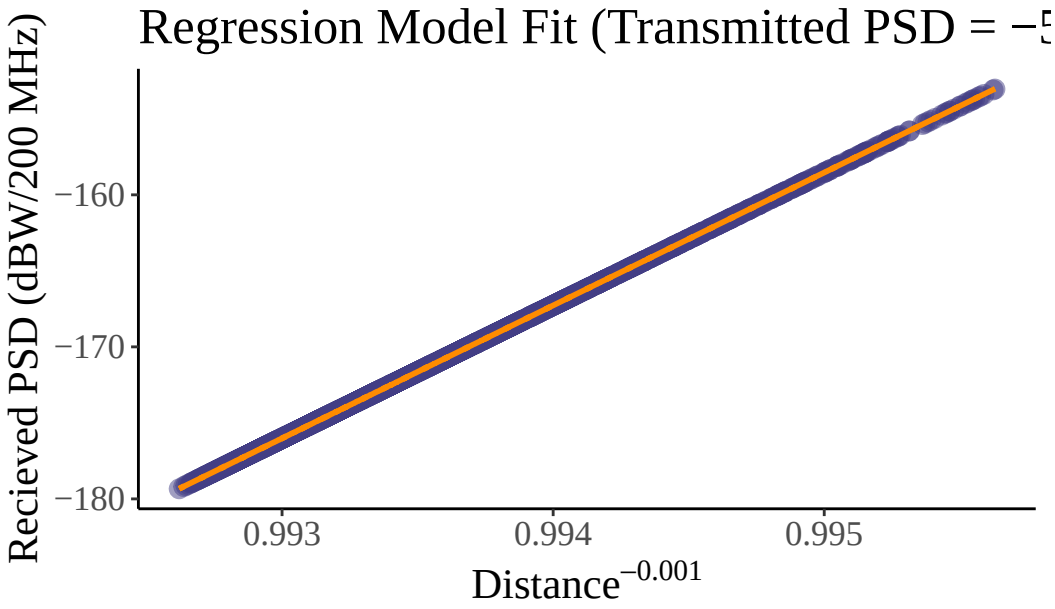
Model Plotting

Making Confidence/Prediction Intervals

```
n <- nrow(lm_pred_test)
Sxx <- sum((aggregate_data$transform5 - mean(aggregate_data$transform5))^2)
xstar <- seq(min(aggregate_data$transform5), max(aggregate_data$transform5), len = 71070)

sigma_m1 <- summary(m1)$sigma
sigma_m2 <- summary(m2)$sigma

ci_m1 <- qt(1 - 0.05/2, n - 2)*sigma_m1*sqrt(1/n + (xstar - mean(aggregate_data$transform5
ci_m2 <- qt(1 - 0.05/2, n - 2)*sigma_m2*sqrt(1/n + (xstar - mean(noise_aggregate_data$tran

pi_m1 <- qt(1 - 0.05/2, n - 2)*sigma_m1*sqrt(1 + 1/n + (xstar - mean(aggregate_data$transf
pi_m2 <- qt(1 - 0.05/2, n - 2)*sigma_m2*sqrt(1 + 1/n + (xstar - mean(noise_aggregate_data$
```

```
# no noise
# predicting on the whole data set
pred <- augment(lm_fit, new_data = aggregate_data)
pred <- pred |> mutate(upper_CI = .pred + ci_m1,
                       lower_CI = .pred - ci_m1,
                       upper_PI = .pred + pi_m1,
                       lower_PI = .pred - pi_m1)

# with noise
# predicting on the whole data set
noise_pred <- augment(noise_lm_fit, new_data = noise_aggregate_data)
noise_pred <- noise_pred |> mutate(upper_CI = .pred + ci_m2,
                                   lower_CI = .pred - ci_m2,
                                   upper_PI = .pred + pi_m2,
                                   lower_PI = .pred - pi_m2)
ipsd55 <- noise_pred |>
  filter(input_psd == -55)

# visualizing the models
if (save == TRUE) pdf(file = "Figures/pred.pdf", height = 5, width = 7)
pred |>
  filter(input_psd == -55) |>
  ggplot(aes(transform5, scaled_psd)) +
    geom_point(color = "#433E85FF", alpha = 0.5, size = 3) +
```

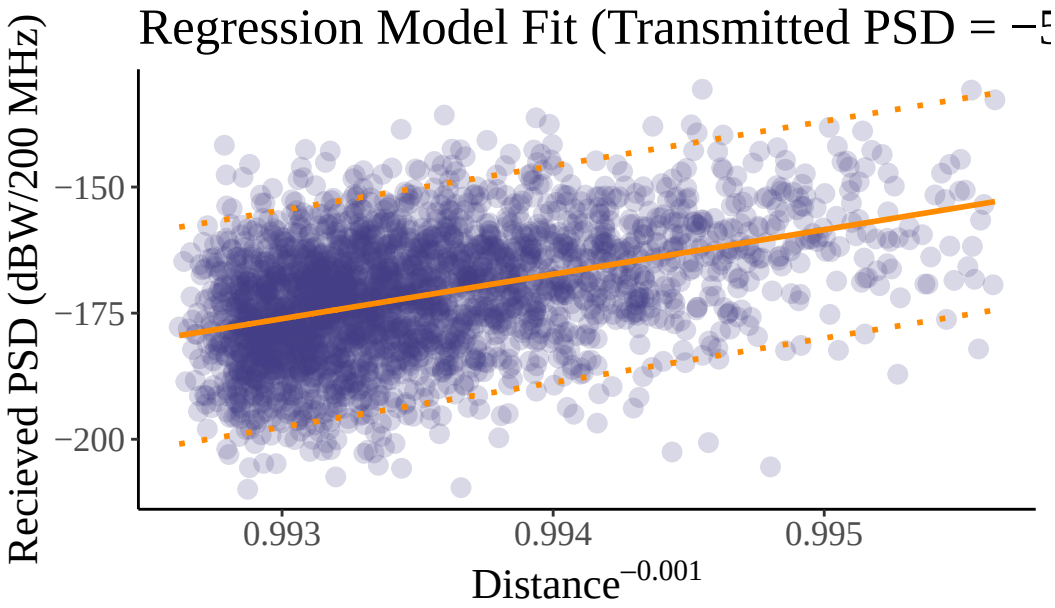
```
    geom_line(aes(transform5, .pred), color = "darkorange", lwd = 1) +
    geom_line(aes(transform5, upper_PI), color = "darkorange", linetype = "dotted", lwd =
    geom_line(aes(transform5, lower_PI), color = "darkorange", linetype = "dotted", lwd =
    labs(title = "Regression Model Fit (Transmitted PSD = -55)",
         caption = "Model fit and prediction interval (no noise added).") +
    xlab(TeX("Distance$^{-0.001}$")) +
    ylab("Recieved PSD (dBW/200 MHz)") +
    theme(text=element_text(family="Times New Roman", size=16),
          plot.caption = element_text(hjust=0.5))
```



Model fit and prediction interval (no noise added).

```
if (save == TRUE) dev.off()

if (save == TRUE) pdf(file = "Figures/noise_pred.pdf", height = 5, width = 7)
noise_pred |>
  filter(input_psd == -55) |>
  ggplot(aes(transform5, scaled_psd)) +
    geom_point(color = "#433E85FF", alpha = 0.2, size = 3) +
    geom_line(aes(transform5, .pred), color = "darkorange", lwd = 1) +
    geom_line(aes(transform5, upper_PI), color = "darkorange", linetype = "dotted", lwd =
    geom_line(aes(transform5, lower_PI), color = "darkorange", linetype = "dotted", lwd =
    labs(title = "Regression Model Fit (Transmitted PSD = -55)",
```

```
      caption = "Model fit and prediction interval with training data (noise added).")
  xlab(TeX("Distance$^{-0.001}$")) +
  ylab("Recieved PSD (dBW/200 MHz)") +
  theme(text=element_text(family="Times New Roman", size=16),
        plot.caption = element_text(hjust=0.5))
```



Model fit and prediction interval with training data (noise added).

```
if (save == TRUE) dev.off()
```