1. Book questions (Use R for all regressions and plots)

   None// //

2. LightGBM:

   Download the data for airbnb for Los Angeles, CA from `http://insideairbnb.com/get-the-data/`. Variable definitions can be found at `https://docs.google.com/spreadsheets/d/1iWCNJcSutYqpULSQHlNyGI edit?pli=1#gid=1322284596`

   This is the exact same data from your last HW

   (a) Keep all but the first 4 variables in the analysis. fix the following:
      1. Create a new variable indicating missing values.
      2. Make the date into a usable variable.
      3. Create a variable that is a binary indicator for if they have a license.
      4. Make sure all the variables are the right type (i.e. factors).

   (b) Remove observations with missing price (DV).

   (c) Impute the missing values using mean/mode or a random forest.

   (d) Create train and test data set.

   (e) Make the model matrices.

   (f) Run a LightGBD model with the default parameters. Record the MSE to compare to later models.

   (g) Look at the residuals. How should they be interpreted.

   (h) Do a grid search over the following parameters to optimize LightGBD:
      1. learning rate
      2. max depth
      3. number of leaves
      4. number of iterations

      Please note, this is a full grid search, not an one at a time like we did with XGBoost. So it will take you around 90 minutes to run.

   (i) Run the model with the optimal hyper-parameter values.

   (j) Compare the test MSE to the default model. How much improvement did you achieve in percentage terms.

   (k) Compare the MSE to the XGBoost model last week. Is there an improvement?

   (l) Calculate the residuals. Do they look appropriate. If not, what impact does that have?

   (m) Calculate the importance matrix. Explain the what 3 variables you believe are the most important and why? Are there any variables you think can be dropped from the model?

   (n) Plot the first three trees for the model. Do the first three trees show the same variables to be important as the importance matrix.

   (o) Plot the multi-tree plot. Does this also validate the importance matrix?

3. Use data from NY Taxi and Limousine Commission from January 2023 High-Volume For-Hire Services (HVFHS) to run the following regressions and diagnostics. The example code is at the end of the questions.

(a) Download the data from: `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.
The data dictionary is at the bottom of the webpage. Get only the January 2023 file for HVFHS.

(b) Take a samples of size 10,000 from the larger data set. (you may choose to delete the larger file after if you like.)

(c) Frequently we want to include different time elements in our ML models. For example, time of day, day of week, wait times all impact the usage and pricing of ride share. Use the R functions to break the timestamp into parts. Create a variable for:

- date
- day of the week
- time of day (hour)
- wait time after calling a ride share
- Create a dummy variable for Uber. Make this a factor!

(d) Plot some of the variables in a scatterplot matrix.

(e) You can DROP the following after making the time variables: **You may choose to keep PULocation for trees where we would not for regression.**

- Dispatching_base_num
- Pickup_datetime
- DropOff_datetime
- PULocationID
- DOLocationID
- originating_base_num
- request_datetime
- on_scene_datetime
- access_a_ride_flag
- wav_request_flag
- wav_match_flag

4. Run a neural network on the taxi data.
Since we did not cover neural networks at length in class, we are only going to fit a very simple network. Just like gradient boosting, neural networks have lots of hyper parameters that need to be adjusted for the best fit. We are going to use the defaults.

(a) Open use the same taxi data to predict Uber as last week. Drop the two ride share variables.

(b) Normalize the data. This is where we rescale all of the data range from 0-1.

(c) Run a neural net with only 2 hidden layers. (this may take a bit). You may need to increase the stepmax in order to get convergence.

(d) Plot your neural net.

(e) Run the test data through you training models for predictions. Plot the predictions. Do they look reasonable?

(f) Calculate the confusion matrix.
You are welcome to fit other nueral nets to see if you can improve the accuracy - and yes, you easily can. They take a long time to run and I am not going to test you on the issue.

```
LA <- read.csv("C:/Users/tonyk/Downloads/listings (2).csv")
View(LA)

#### Create variable for if they have a license
LA$license[LA$license == ""] <- NA

### keep what you need
la2<- LA[,c(5:12, 14:17)]
la2$license1<- as.factor(is.na(LA$license))


#make dates usable
library(lubridate)
la2$last_review_date <- ymd(LA$last_review)

### Make a variable for if any row has a missing value
la2$missing<-as.factor(complete.cases(la2))

### make missing a variable by column
## make a matrix of missing values
miss0<- as.data.frame(is.na(la2) )
## make it numeric
miss1<-lapply(miss0, as.numeric)


#### Missing Data
library(randomForest)
set.seed(982465)

##Remove missing DV before imputing.
noy<-is.na(la2$price)
la3<-la2[which(noy==0),]

### make sure IV are right class (factors)
sapply(la3,class)
la3[,1]<- as.factor(la3[,1])
la3[,2]<- as.factor(la3[,2])
la3[,5]<- as.factor(la3[,5])
la3[,13]<- as.factor(la3[,13])
la3[,14]<- as.factor(la3[,14])
```

```
### Impute with mean/mode
la3.mean<- na.roughfix(la3)



## reorder to make y first, not needed just easier.
la3.mean<-la3.mean[,c(6,1:5,7:15)]

#### make factors into numbers

i <- sapply(la3.mean, is.factor)
dict <- lapply(la3.mean[i], \(x) setNames(seq_len(nlevels(x)), levels(x)))
la3.mean[i] <- lapply(la3.mean[i], as.integer)




## set up data
y_train <- (train[,1])
y_test <- (test[,1])
### lgb dataframe types helps it run fast.
train_lgb <- lgb.Dataset(train[,2:ncol(train)],label=y_train, categorical_feature=c(2,5))
test_lgb <- lgb.Dataset.create.valid(train_lgb,test[,2:ncol(train)],label = y_test)


#base untuned lightgbn
light_gbn_base <- lgb.train(
params = list(
objective = "regression",
metric = "l2"
),
data = train_lgb
)


yhat_fit_base <- predict(light_gbn_base,train[,2:ncol(train)])
yhat_predict_base <- predict(light_gbn_base,test[,2:ncol(train)])

rmse_fit_base <- RMSE(y_train,yhat_fit_base)
rmse_predict_base <- RMSE(y_test,yhat_predict_base)




#################
#grid search
#create hyperparameter grid
num_leaves =seq(24,44,2)
```

```
max_depth = round(log(num_leaves) / log(2),0)
num_iterations = seq(400,500,100)
early_stopping_rounds = round(num_iterations * .1,0)

hyper_grid <- expand.grid(max_depth = max_depth,
num_leaves =num_leaves,
num_iterations = num_iterations,
early_stopping_rounds=early_stopping_rounds,
learning_rate = seq(.30, .50, .025))

hyper_grid <- unique(hyper_grid)
rmse_fit <- NULL
rmse_predict <- NULL

for (j in 1:nrow(hyper_grid)) {
set.seed(32123)
light_gbn_tuned <- lgb.train(
params = list(
objective = "regression",
metric = "l2",
max_depth = hyper_grid$max_depth[j],
num_leaves =hyper_grid$num_leaves[j],
num_iterations = hyper_grid$num_iterations[j],
early_stopping_rounds=hyper_grid$early_stopping_rounds[j],
learning_rate = hyper_grid$learning_rate[j]
#feature_fraction = .9
),
valids = list(test = test_lgb),
data = train_lgb
)

yhat_fit_tuned <- predict(light_gbn_tuned,train[,2:ncol(train)])
yhat_predict_tuned <- predict(light_gbn_tuned,test[,2:ncol(train)])

rmse_fit[j] <- RMSE(y_train,yhat_fit_tuned)
rmse_predict[j] <- RMSE(y_test,yhat_predict_tuned)
cat(j, "\n")
}


min(rmse_fit)
min(rmse_predict)
hyper_grid[which.min(rmse_fit),]
hyper_grid[which.min(rmse_predict),]



rmse_diff <- rmse_fit - rmse_predict
```

```
rmse_models <- data.frame(rmse_fit,rmse_predict, rmse_diff)
rmse_models_sort <- rmse_models[order(rmse_diff),]
#11:42
set.seed(32123)
light_gbn_final <- lgb.train(
params = list(
objective = "regression",
metric = "l2",
max_depth = 5,
num_leaves =30,
num_iterations = 400,
early_stopping_rounds=40,
learning_rate = .35
#feature_fraction = .8
),
valids = list(test = test_lgb),
data = train_lgb
)

yhat_fit_final <- predict(light_gbn_final,train[,2:ncol(train)])
yhat_predict_final <- predict(light_gbn_final,test[,2:ncol(train)])

rmse_fit_final<- RMSE(y_train,yhat_fit_final)
rmse_predict_final <- RMSE(y_test,yhat_predict_final)

plot(y_test,yhat_predict_final, main='LightGBM of LA AirBnB', xlab='actual', ylab='predicted')
abline(lm(yhat_predict_final~y_test))

lgb_imp <- lgb.importance(light_gbn_final)

lgb.plot.importance(lgb_imp)


r <- y_test - yhat_predict_final

sum(abs(r) <= rmse_predict_final) / length(y_test)
sum(abs(r) <= 2 * rmse_predict_final) / length(y_test)



##############################################
library(ggplot2)
library(reshape2)
```

```
library(arrow)

# Read in data- you will need to change the path
trips <- read_parquet('C:/Users/tonyk/Downloads/fhvhv_tripdata_2023-01 (1).parquet')
#take a sample
set.seed(39756820)
trip1<-trips[sample(1:nrow(trips), 10000),]

# Optional, remove the larger data set to save memory
# rm(trips)

#make dates usable
trip1$date <- as.Date( format(as.POSIXct(trip1$pickup_datetime,
format = '%m/%d/%Y %H:%M:%S'),
format = '%m/%d/%Y'), "%m/%d/%y")

#create weekday variable
library(lubridate)
trip1$weekday <- as.factor(wday(trip1$date))

#create an hour of the day variable
trip1$hour <- as.numeric(format(as.POSIXct(trip1$pickup_datetime,
format = '%m/%d/%Y %H:%M:%S'),
format = "%H"))
# Create a wait time variable. pmax forces neagtive wait times to be 0 as that is impossible.
trip1$wait<- pmax(as.numeric(difftime(trip1$pickup_datetime, trip1$request_datetime)),0)

# remove negative fares, which are errors
trip1$base_passenger_fare<-pmax(trip1$base_passenger_fare,0)

## Recode Uber or lyft
library(car)
trip1$Uber<- as.numeric( trip1$hvfhs_license_num=='HV0003')

# Optional:subset to only the variables used. Drop id variables,
# drop access_a_ride_flag and wav_match_flag
trip1.1<- trip1[,-c(1:9,22:24)]


# Optional: plot the variables
library(GGally)
ggpairs(trip1.1, columns= 1:5)



set.seed(27514)
mm<- model.matrix(~. -1, data=trip1.2)
library(scales)
```

```
# drop the rideshare variables
mm<- mm[,c(2:11, 15:21)]
## rescale all the variables
mm2<- apply(mm, 2, rescale)
## train and test datasets
samp <- sample(c(TRUE, FALSE),
nrow(mm),
replace=TRUE,
prob=c(0.7,0.3))
train <-  mm2[samp,]
test <- mm2[!samp,]

library(neuralnet)
# nueral network with 2 hidden nodes
nn_train1 <- neuralnet(Uber1~., data=train, hidden=c(2), linear.output=F, rep=1)
# on test data
nn_test1 <- neuralnet(Uber1~., data=test, hidden = c(5),
linear.output = F)

# Do not run! nueral network with 2 layers, 3 hidden nodes and then 2
# nn_train2 <- neuralnet(Uber1~., data=train, hidden=c(3,2), linear.output=F, rep=3, stepmax=1

plot(nn_train1, rep = 'best')

set.seed(42132) # set the random seed for reproducibility
# Compute fitted values from the training data
predictions_train <- predict(nn_train1, newdata = train)
# Test the neural networks out of sample performance
predictions_test <- predict(nn_train1, newdata = test)

p.test<-round(predictions_test,0 )
cm<- table(p.test, test[,1])
cm
```