Appendix

EDA

```
# add some libraries
library(ggplot2)
library(dplyr)
```

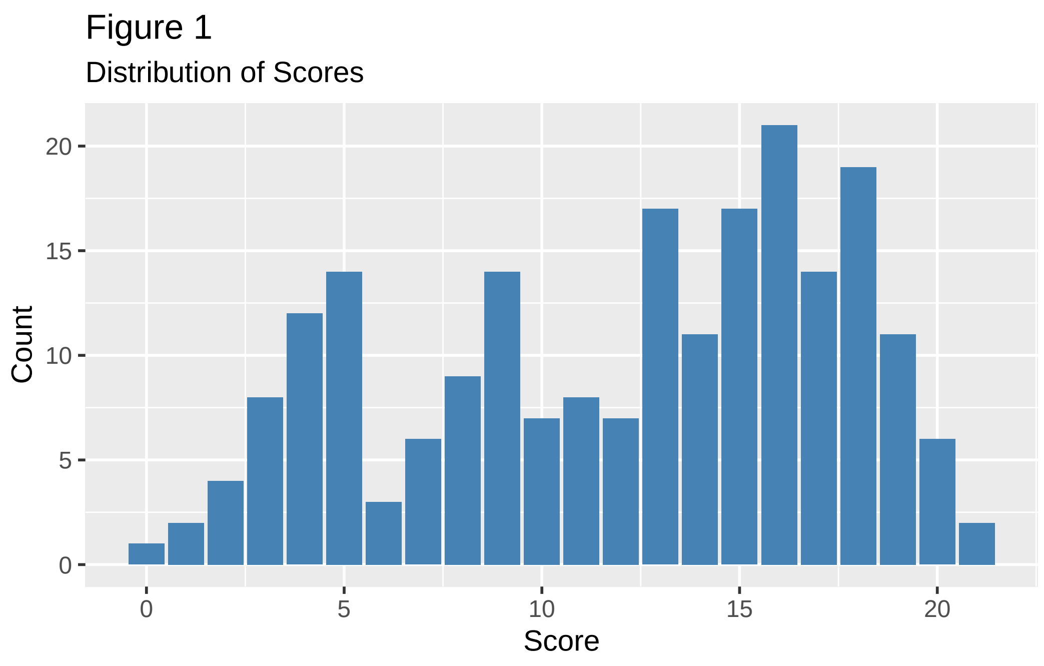
```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
# Load the survey ratings data
survey_response <- readr::read_csv('https://raw.githubusercontent.com/yawgmoth/HanabiData/re
```

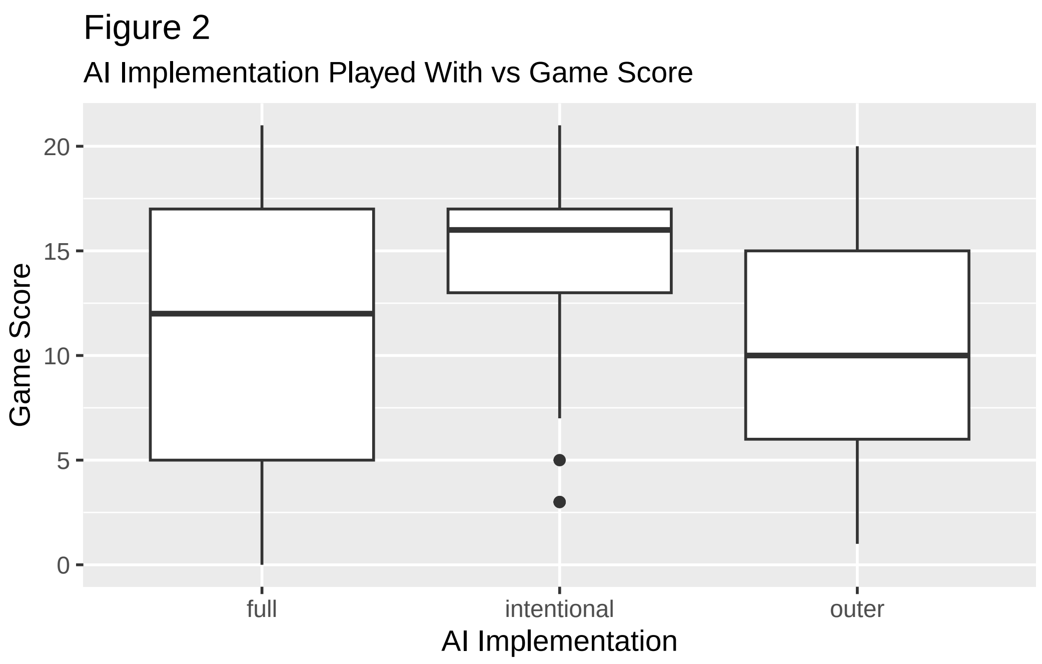
```
Rows: 240 Columns: 14

-- Column specification --------------------------------------------------------
Delimiter: ","
chr (5): id, ai, age, gamer, publish
dbl (9): deck, score, boardgameexp, hanabiexp, recent, maxscore, intention, ...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#dim(survey_response)
# count rows with NA entries
#sum(rowSums(is.na(survey_response)) > 0) #27 rows with missing data


# Preview the data
# summary(survey_response)

# cleanup and remove NAs
survey_complete <- survey_response[complete.cases(survey_response),]
survey_complete$ai <- survey_complete$ai %>% as.factor()
survey_complete$gamer <- survey_complete$gamer %>% as.factor()
survey_complete $age <- survey_complete$age %>% as.factor()
#dim(survey_complete)
#summary(survey_complete)


# plot distribution of scores
ggplot(survey_complete, aes(x = score)) + geom_bar(bins = 26, fill = "steelblue") + labs(titl
```

```
Warning in geom_bar(bins = 26, fill = "steelblue"): Ignoring unknown
parameters: `bins`
```



```
# plot AI implementation vs score
ggplot(survey_complete, aes(x = ai, y = score)) +
  geom_boxplot() + labs(title = "Figure 2", subtitle = "AI Implementation Played With vs Game
```

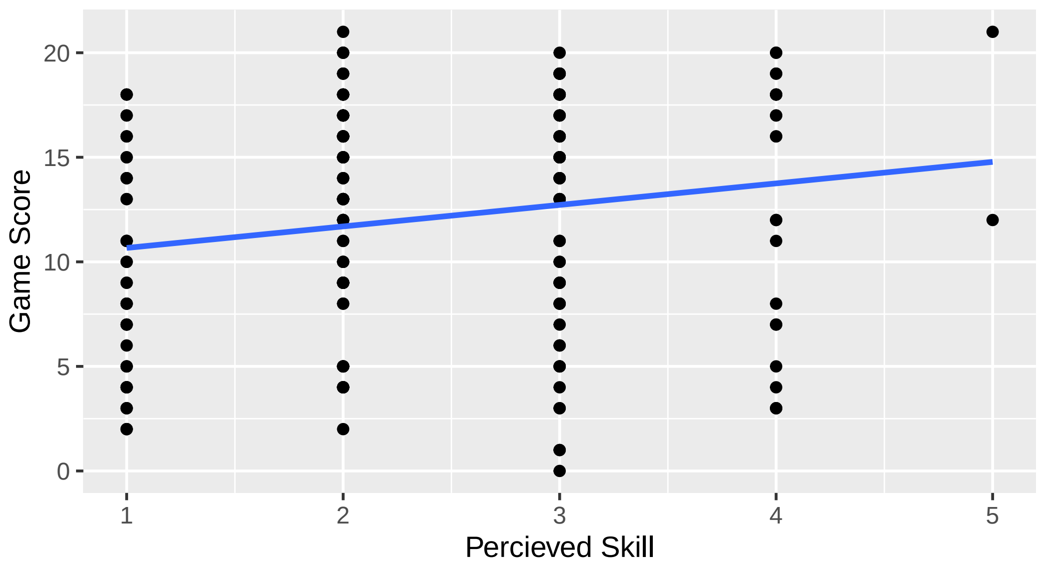


```
# plot percieved skill vs score
ggplot(survey_complete, aes(x = skill, y = score)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) + labs(title = "Figure 3", subtitle = "Perceived Ski
```

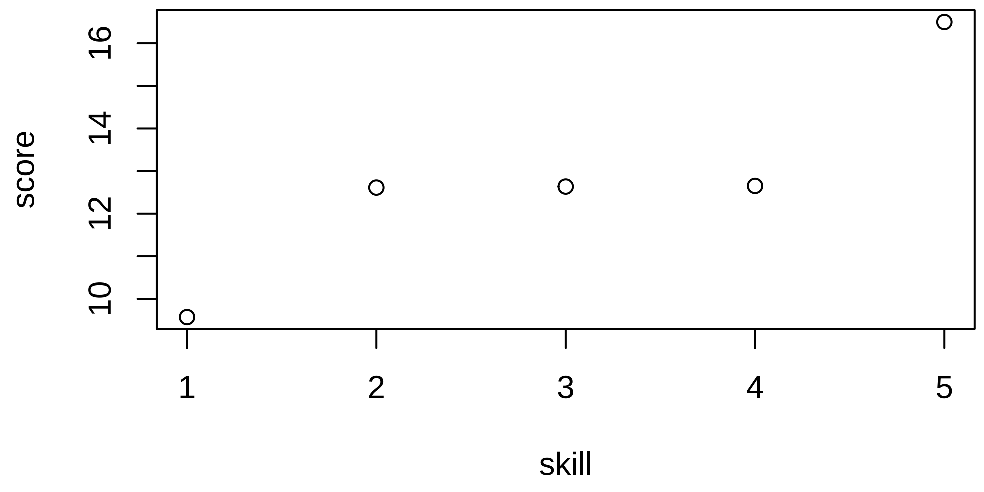
```
`geom_smooth()` using formula = 'y ~ x'
```

Figure 3

Perceived Skill vs Game Score



```
# aggregate score by perceived skill of AI
score_skill <- aggregate(score ~ skill, data = survey_complete, FUN = mean)
#score_skill

# aggregated score by whether they liked the AI
score_like <- aggregate(score ~ like, data = survey_complete, FUN = mean)
#score_like

# aggregate score by how intentional they thought the AI was
score_intention <- aggregate(score ~ intention, data = survey_complete, FUN = mean)
#score_intention

# aggregate score by which AI was used
score_ai <- aggregate(score ~ ai, data = survey_complete, FUN = mean)
#score_ai

# plot aggregated score by perceived skill
plot(score_skill)
```



```
# plot aggregated score by AI implementation
# plot(score_ai)
```

ANOVA and Tukey

```
# anova with just AI

model <- aov(score ~ ai, survey_complete)
#summary(model)

# need AI and skill to be factors
survey_complete$ai <- factor(survey_complete$ai)
survey_complete$skillf <- factor(survey_complete$skill)

# anove with both AI and skill plus interaction
model2 <- aov(score ~ skillf + ai + skill:ai, survey_complete)
#summary(model2)

# anova with just skill (as factor)
model3 <- aov(score ~ skillf, survey_complete)
#summary(model3)
```

```r
# anova with just AI (as factor)
model4 <- aov(score ~ ai, survey_complete)
summary(model4)
```

```
             Df Sum Sq Mean Sq F value   Pr(>F)    
ai            2    589  294.66   11.19 2.42e-05 ***
Residuals   210   5531   26.34                     
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Tukey test compares all pairwise options from the ANOVA
tukey_results <- TukeyHSD(model4)
print(tukey_results)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = score ~ ai, data = survey_complete)

$ai
                        diff       lwr       upr     p adj
intentional-full   3.1428571  1.064447  5.221268 0.0012846
outer-full        -0.7467532 -2.747293  1.253786 0.6528219
outer-intentional -3.8896104 -5.921657 -1.857564 0.0000309
```

```r
perceived_skill <- survey_complete$skill
final_score <- survey_complete$score

# Perform Spearman's rank correlation test
#result <- cor.test(perceived_skill, final_score, method = "spearman")

# Print the full result
#print(result)
```

Spearman’s Rank

```r
perceived_skill <- survey_complete$skill
final_score <- survey_complete$score

# Perform Spearman's rank correlation test
result <- cor.test(perceived_skill, final_score, method = "spearman")
```

```
Warning in cor.test.default(perceived_skill, final_score, method = "spearman"):
Cannot compute exact p-value with ties
```

```r
# Print the full result
print(result)
```

```

    Spearman's rank correlation rho

data:  perceived_skill and final_score
S = 1297231, p-value = 0.004373
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.1945487
```

```r
perceived_like <- survey_complete$like
final_score <- survey_complete$score

# Perform Spearman's rank correlation test
result <- cor.test(perceived_like, final_score, method = "spearman")
```

```
Warning in cor.test.default(perceived_like, final_score, method = "spearman"):
Cannot compute exact p-value with ties
```

```r
# Print the full result
print(result)
```

```

    Spearman's rank correlation rho

data:  perceived_like and final_score
S = 1470958, p-value = 0.2077
alternative hypothesis: true rho is not equal to 0
sample estimates:
       rho
0.08668117
```

Neural Network

```r
install.packages("keras3")
#Load the necessary libraries
library(tidyverse)
library(dplyr)
library(keras3)

features_data <- read_csv("features.zip", "features.csv")
features_complete <- features_data[complete.cases(features_data),]

# remove first row, it has text labels
first_row <- features_complete[1, ]
# data frame with all the 1s and 0s
df_num <- as.data.frame(lapply(features_complete[-1, ], as.numeric))

#df_num_complete <- complete.cases(df_num)

# calculate number of samples we have
num_samples = nrow(df_num)

# Split the dataset into training and testing sets
# df_num is a data frame: 192731 x 393
# all data is one-hot-encoded categorical, all 1 or 0
split_ratio <- 0.8
num_train_samples <- floor(num_samples * split_ratio)
train_data <- df_num[1:num_train_samples, ]
test_data <- df_num[(num_train_samples + 1):num_samples, ]

# Prepare the data for training
# train_data is a data frame: 154184 x 393
# 373 columns are features, the last 20 are the "action" we are predicting
train_features <- as.matrix(train_data[, 1:373])
train_labels <- as.matrix(train_data[, 374:393])
test_features <- as.matrix(test_data[, 1:373])
test_labels <- as.matrix(test_data[, 374:393])

# Build the neurla network model
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = ncol(train_features)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = ncol(train_labels), activation = "softmax") # Use 'sigmoid' if multi-la
```

```
# Compile the model
model %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy", # Use 'binary_crossentropy' for multi-label with sigmoi
  metrics = c("accuracy")
)

# Print the model summary
summary(model)

# Train the model
history <- model %>% fit(
  x = train_features,
  y = train_labels,
  epochs = 50,
  batch_size = 32,
  validation_split = 0.2
)

# Display results

history
plot(history)


predictions <- model %>% predict(test_features)

eval_result <- model %>% evaluate(
  x = test_features,
  y = test_labels
)

cat("Test accuracy:", eval_result$accuracy, "\n")
cat("Test loss:", eval_result$loss, "\n")

first_row[373 + which.max(predictions[1,])]

first_row[373 + which.max(predictions[2,])]
```

XGBoost

```r
#install.packages("xgboost")
library(tidyverse)
library(xgboost)
library(caret)
unzip("features.zip")
features_df <- read_csv("features.csv", col_names = TRUE)

View(features_df)

dim(features_df)

# drop na
features_df <- features_df %>% drop_na()

# aim to predict player actions based on current game state (20 possibilities; hint specific
# model will only be applicable for games w/ totalpoints >= 20 (as data is only from games wh
# last 20 cols of features are player actions (1-hot encoded), which is what we want to predi


# converting the potential actions (binary cols) to multilabel factor 0-19 since xgb is 0 ind
y <- apply(features_df[ , tail(names(features_df), 20)], MARGIN = 1, which.max) - 1

action_cols <- tail(names(features_df), 20)

# class labels
action_map <- data.frame(
    class_id = 0:19,
    action_col = action_cols
)
action_map

features_df$label <- y

set.seed(111)
training_rows <- sample(seq_len(nrow(features_df)), size = 0.8 * nrow(features_df))
train_df <- features_df[training_rows, ]
test_df <- features_df[-training_rows, ]

training_input <- xgb.DMatrix(data = as.matrix(train_df[, 1:373]), label= train_df$label)
testing_input <- xgb.DMatrix(data = as.matrix(test_df[,1:373]), label= test_df$label)

# now building first model (maybe hyperparameter tune later?)
```

```r
parameters <- list(
    booster = "gbtree",
    objective = "multi:softprob",
    num_class = 20,
    eval_metric = "mlogloss",
    max_depth = 6,
    min_child_weight = 1,
    gamma = 1,
    eta = 0.1,
    subsample = 0.7,
    colsample_bytree = 0.7
)

watchlist <- list(
  train = training_input,
  eval = testing_input
)


model1 <- xgb.train(
  params = parameters,
  data = training_input,
  nrounds = 1000,
  watchlist = watchlist,
  early_stopping_rounds = 30,
  print_every_n = 30,
  nthread = parallel::detectCores(),
  tree_method = "hist"
)

importance_matrix <- xgb.importance(
  feature_names = colnames(train_df[, 1:373]),
  model = model1
)
print(importance_matrix[1:20, ])
xgb.plot.importance(
  importance_matrix = importance_matrix[1:20, ],
  main = "top 20 feature importance"
)

pred_probs <- predict(model1, testing_input)
pred_probs_matrix <- matrix(pred_probs, nrow = length(pred_probs) / 20, ncol = 20, byrow = Tl
pred_class <- max.col(pred_probs_matrix) - 1

test_df$pred_classes <- pred_class
View(test_df)

conf_mat <- confusionMatrix(
  data = factor(test_df$pred_classes),
  reference = factor(test_df$label)
)
conf_mat

f1_scores_per_class <- conf_mat$byClass[, "F1"]
macro_f1 <- mean(f1_scores_per_class)
macro_f1


top_k_accuracy_xgb <- function(pred_matrix, true_labels, k) {
  top_k_preds <- apply(pred_matrix, 1, function(x) order(x, decreasing = TRUE)[1:k])
  correct <- sapply(1:length(true_labels), function(i) {
    (true_labels[i] + 1) %in% top_k_preds[, i]
  })
  mean(correct)
}

top_k_accuracy_xgb(pred_probs_matrix, test_df$label, 3)
```