Income Classification Project

Luke Buaas

2024-06-21

Introduction:

In today's data-driven world, our ability to quantify circumstances using data and then predict outcomes has reached unprecedented levels. This has been proved to be extremely useful in various domains including daily life, business, and government operations. One particularly valuable application of data-driven predictions is identifying individuals who earn less than $50,000 USD annually based on their census data. This capability can help individuals make informed decisions to enhance their earning potential and quality of life, while also providing crucial insights for various stakeholders, such as government entities.

In this project we will aim to leverage multiple machine learning models to classify an individual's income into two catergories: less than or equal to $50,000 USD, and greater than $50,000 USD. To do this we will analyze a diverse data set derived from census data, including many attributes such as age, education, occupation, martial status, and so on. By analyzing these features, we will build and compare multiple machine learning models that can accurately forecast income levels.

For this project the primary objectives are:

1) Data Cleansing & Analysis: We will begin by downloading the data and examining the relationship between all variables and income. Also, we will transform the data to make sure it is suitable for all applicable machine learning models.

2) Model Development: In this step we will implement various machine learning models, including Random Forest, XGBoost, KNN, Decision Trees, and an Ensemble of the best models.

3) Model Evaluation: Finally we will evaluate the models and judge their effectiveness using multiple metrics, such as accuracy, balanced accuracy, and F1 score.

Through this project, we will demonstrate the power of different machine learning models in classifying socio-economic data, and provide a strong framework that can be used for similar classification tasks.

Loads data & packages

```
if (!require(tidyverse)) install.packages('tidyverse')
library(tidyverse)
if (!require(caret)) install.packages('caret')
library(caret)
if (!require(factoextra)) install.packages('factoextra')
library(factoextra)
if (!require(randomForest)) install.packages('randomForest')
library(randomForest)
if (!require(data.table)) install.packages('data.table')
library(data.table)
if (!require(knitr)) install.packages('knitr')
library(knitr)
if (!require(rpart)) install.packages('rpart')
library(rpart)


# Load the data
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"

# Sets names to the values from the website (https://archive.ics.uci.edu/dataset/2/adult)
column_names <- c("age", "workclass", "fnlwgt", "education", "education_num", "marital_status",
                  "occupation", "relationship", "race", "sex", "capital_gain", "capital_loss",
                  "hours_per_week", "native_country", "income")

# Load the data & replaces NA with "?"
data <- read.csv(url, header = FALSE, col.names = column_names, na.strings = "?")
sum(is.na(data))
```

```
## [1] 0
```

Before beginning to fit models to the data, it is essential to thoroughly examine the variables present in the dataset to gain a deeper understanding of the data. This invloves examining the relationship between the available variables and our target variable, income. For this analysis, we will convert most of our variables to factors. This conversion ensures that our data is accurately represented in our graphs and is compatible with the machine learning algorithms we will use to predict income.

Data Exploration, Cleaning, & Analysis

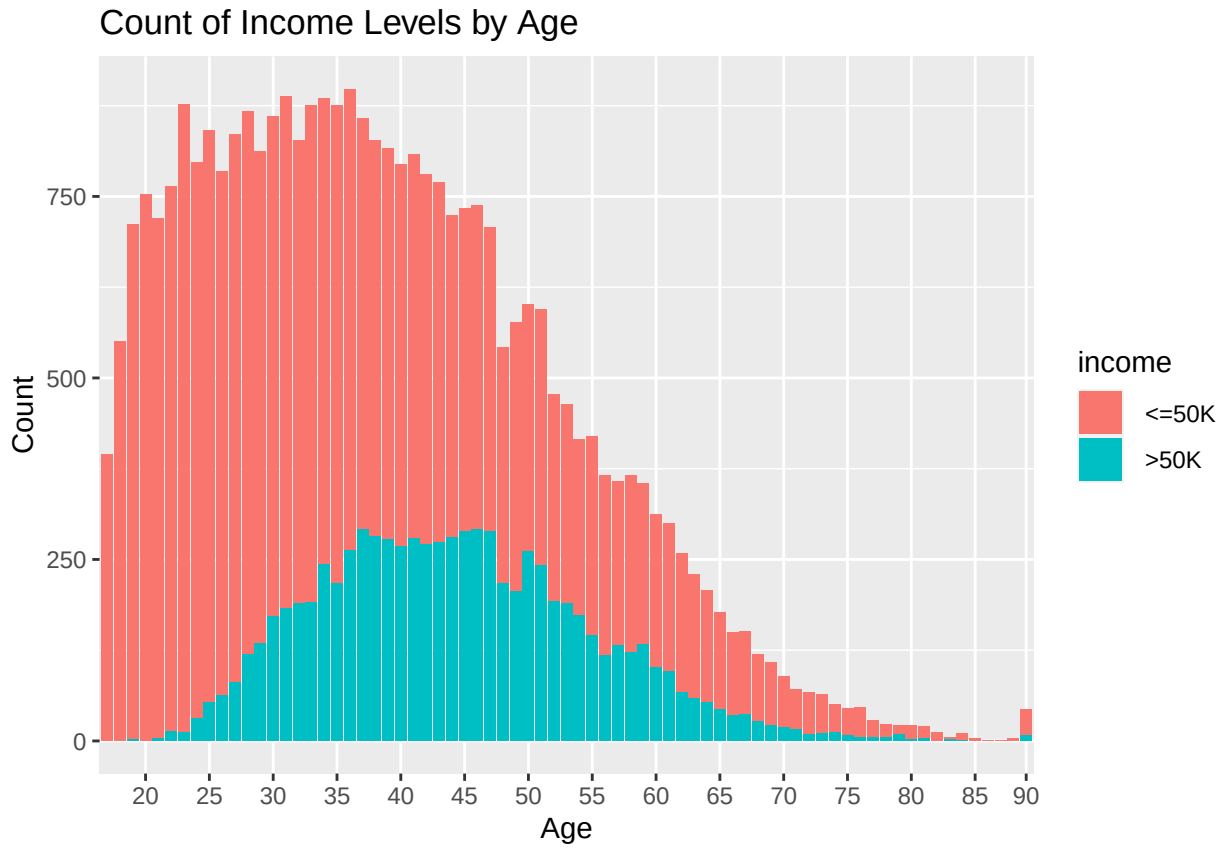
Above/Under 50K count

```
data %>%
  group_by(income) %>%
  summarize(count = n()) %>%
  kable()
```

income	count
<=50K	24720
>50K	7841

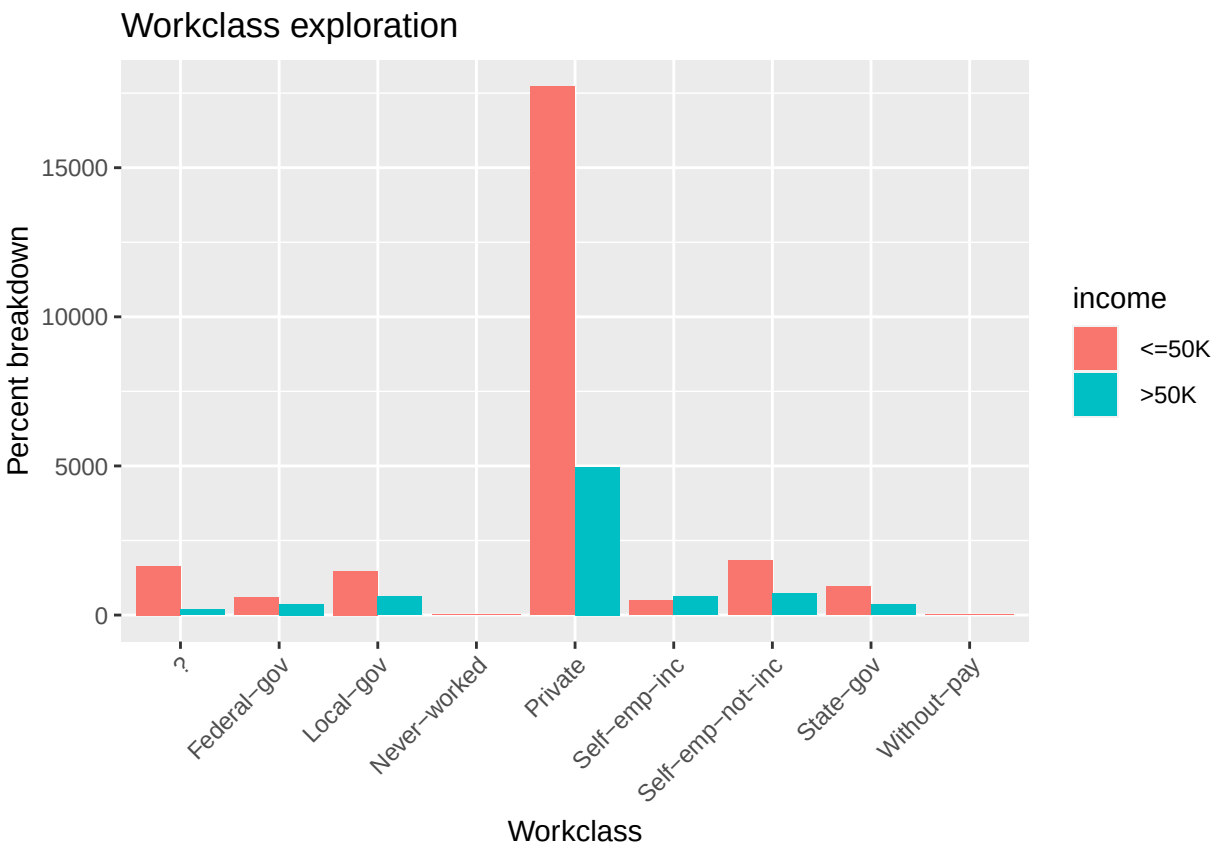
Age exploration

```
# Age exploration
data %>%
  ggplot(aes(x=as.factor(age), fill=income)) +
  geom_bar(position = "stack") +
  labs(x = "Age", y = "Count", title = "Count of Income Levels by Age") +
  scale_x_discrete(breaks = seq(0, 110, by = 5))
```



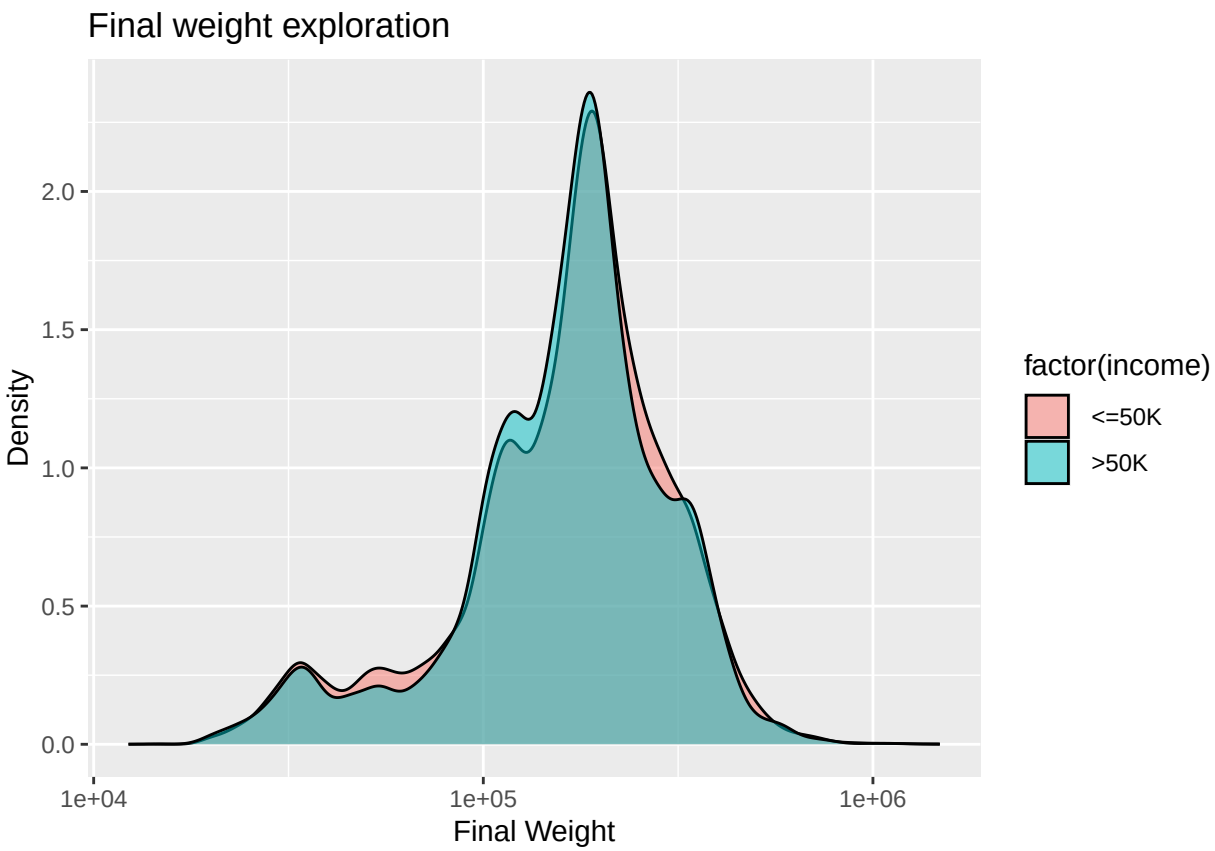
Workclass Exploration

```
# Workclass exploration
data %>%
  ggplot(aes(x=workclass, fill=income)) +
  geom_bar(position = "dodge") +
  labs(x = "Workclass", y = "Percent breakdown", title = "Workclass exploration") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



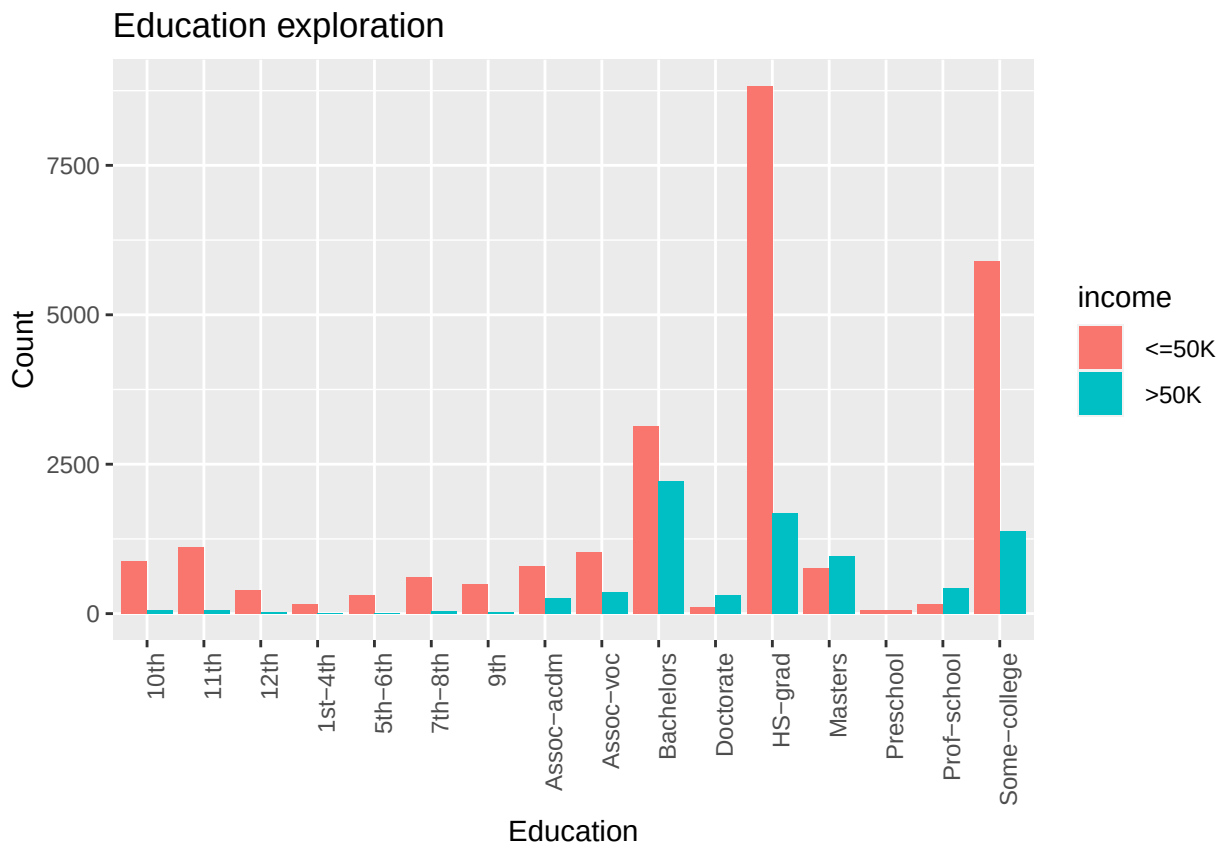
Final Weight Exploration

```
# fnlwgt exploration
data %>%
  ggplot(aes(x = fnlwgt, fill = factor(income))) +
  geom_density(alpha = 0.5) +
  scale_x_log10() +
  labs(x = "Final Weight", y = "Density", title = "Final weight exploration")
```



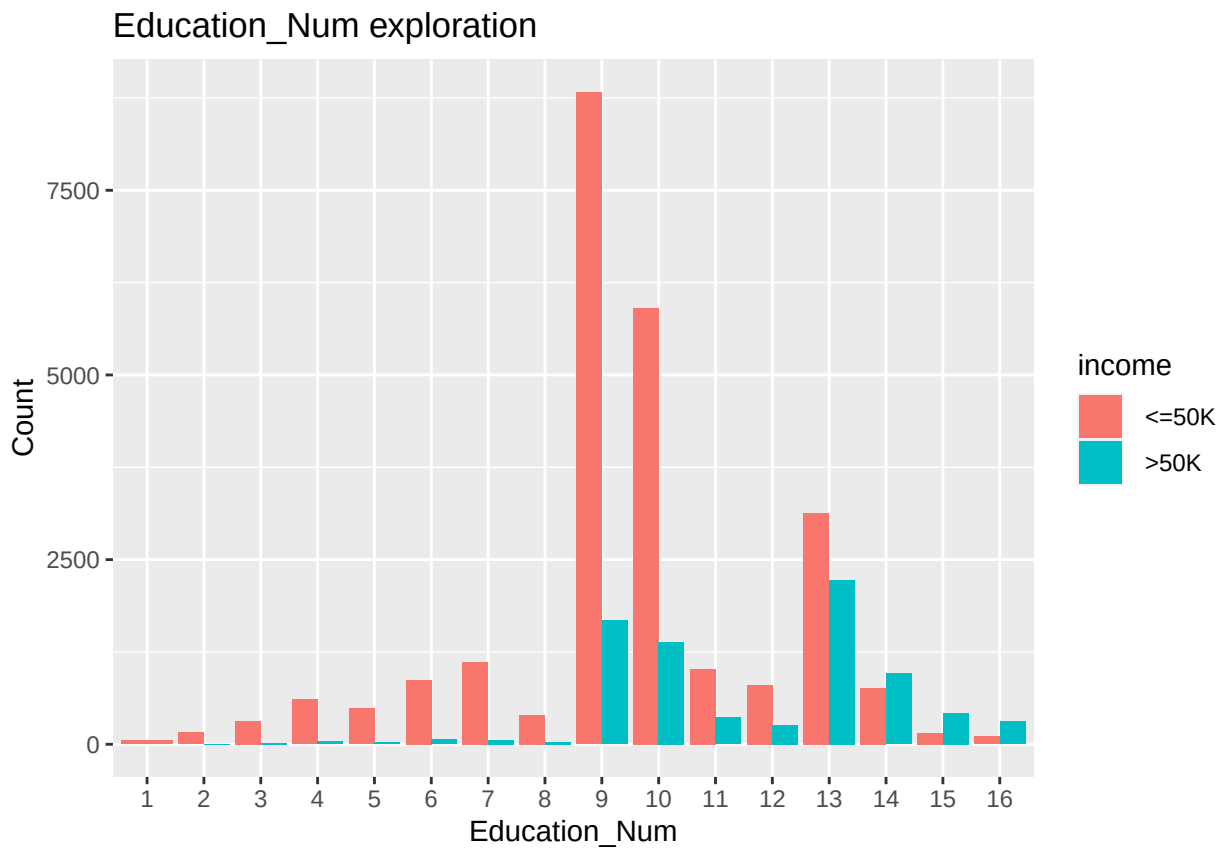
Education Exploration

```
# education exploration
data %>%
  ggplot(aes(x=education, fill=income)) +
  geom_bar(position = "dodge") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Education", y = "Count", title = "Education exploration")
```



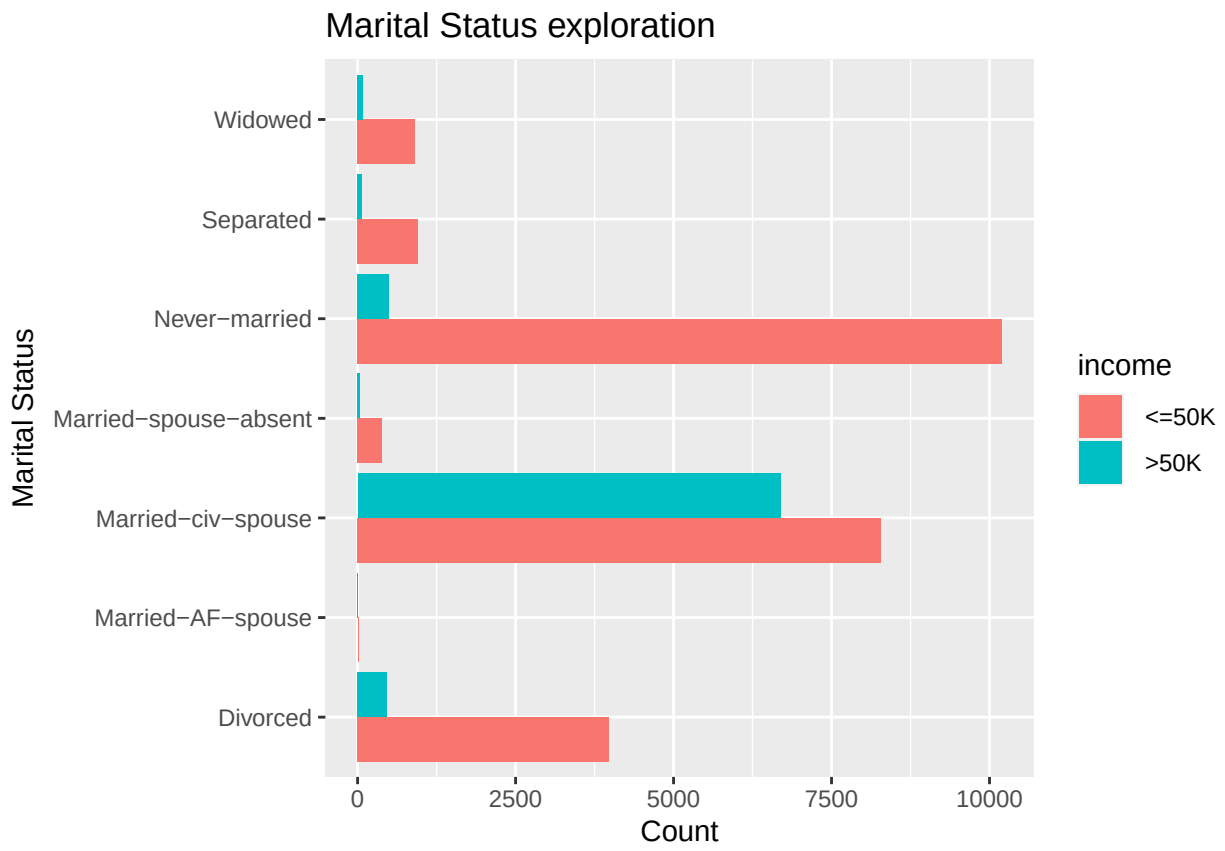
Education Number Explaination

```
# education-num exploration
data %>%
  ggplot(aes(x=as.factor(`education_num`), fill=income)) +
  geom_bar(position = "dodge") +
  labs(x = "Education_Num", y = "Count", title = "Education_Num exploration")
```



Marital Status Exploration

```
# marital-status exploration
data %>%
  ggplot(aes(x=as.factor(`marital_status`), fill=income)) +
  geom_bar(position = "dodge") +
  coord_flip() +
  labs(x = "Marital Status", y = "Count", title = "Marital Status exploration")
```

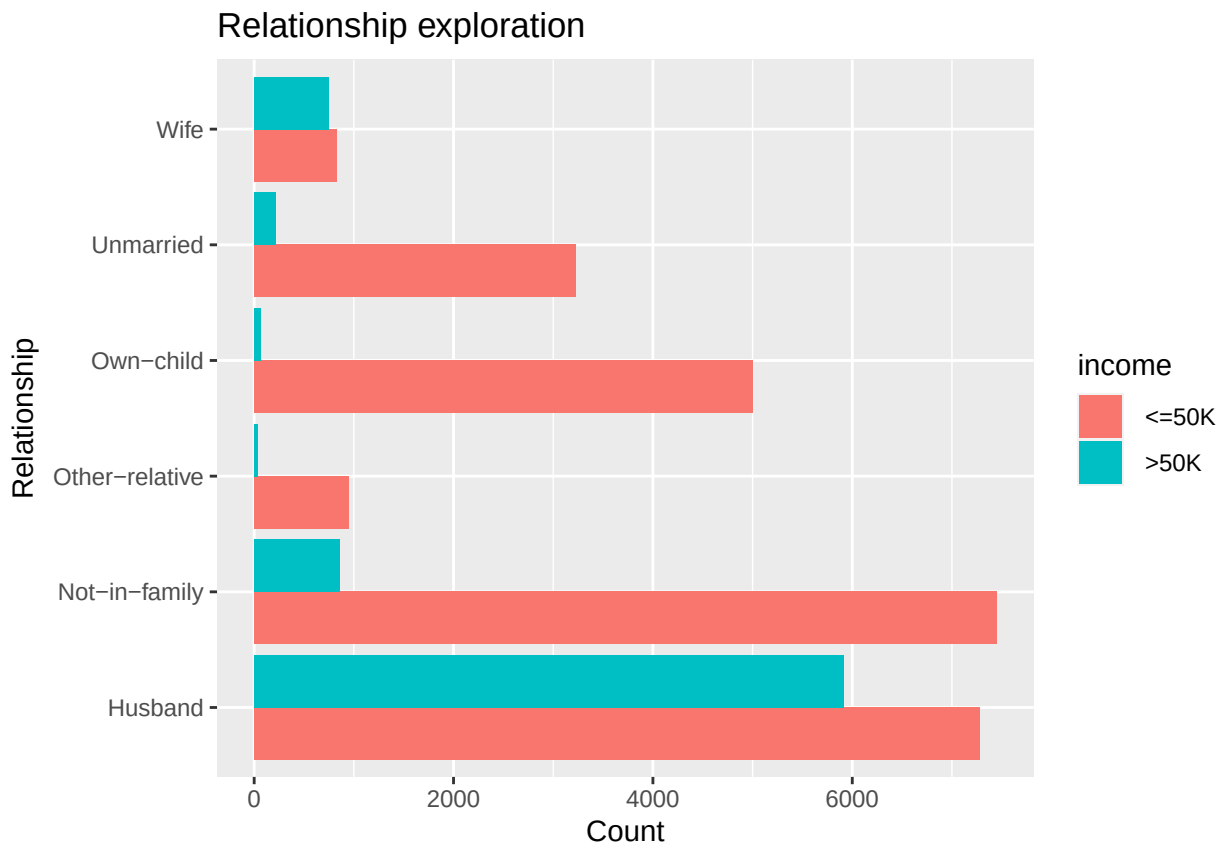


Occupation Exploration

```
# occupation exploration
data %>%
  ggplot(aes(x=as.factor(occupation), fill=income)) +
  geom_bar(position = "dodge") +
  coord_flip() +
  labs(x = "Occupation", y = "Count", title = "Occupation exploration")
```

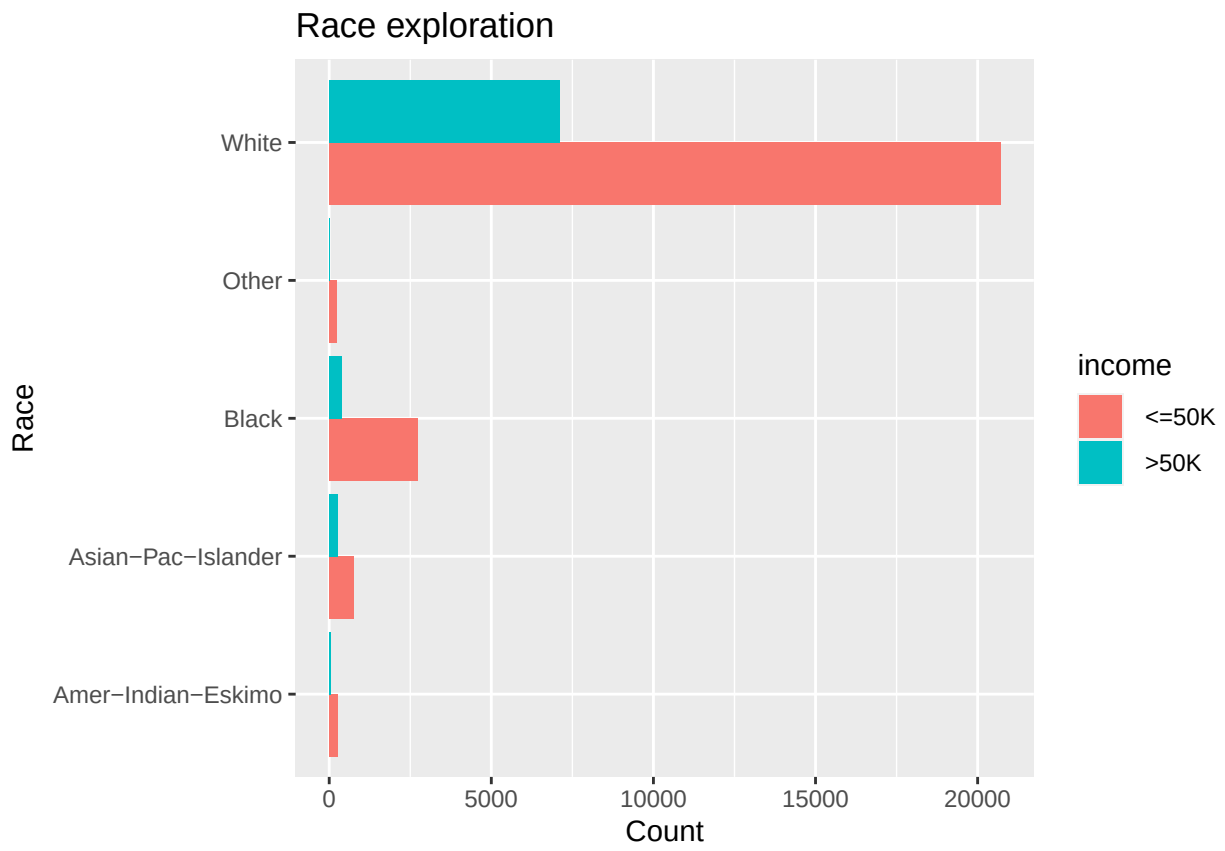
Relationship Exploration

```
# relationship exploration
data %>%
  ggplot(aes(x=as.factor(relationship), fill=income)) +
  geom_bar(position = "dodge") +
  coord_flip() +
  labs(x = "Relationship", y = "Count", title = "Relationship exploration")
```



Race Exploration

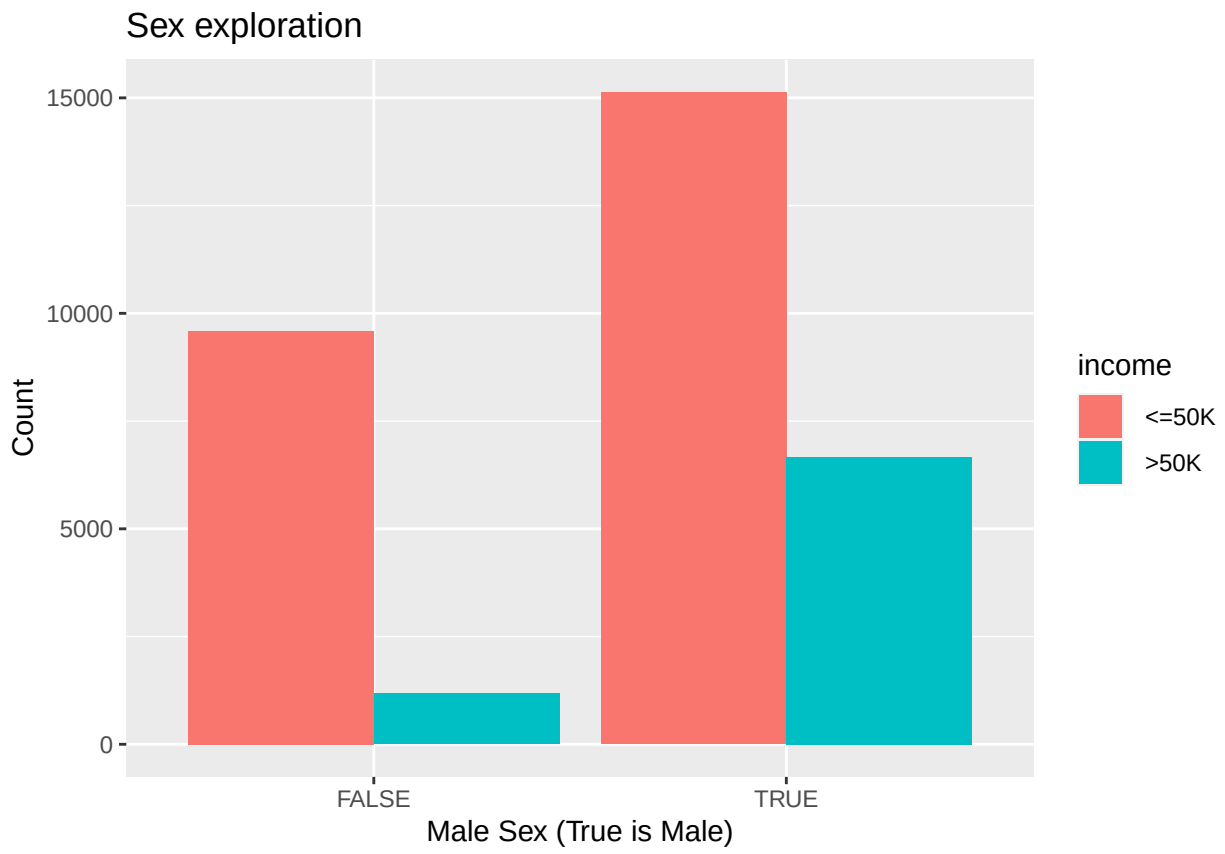
```
# race exploration
data %>%
  ggplot(aes(x=as.factor(race), fill=income)) +
  geom_bar(position = "dodge") +
  coord_flip() +
  labs(x = "Race", y = "Count", title = "Race exploration")
```



Sex Exploration

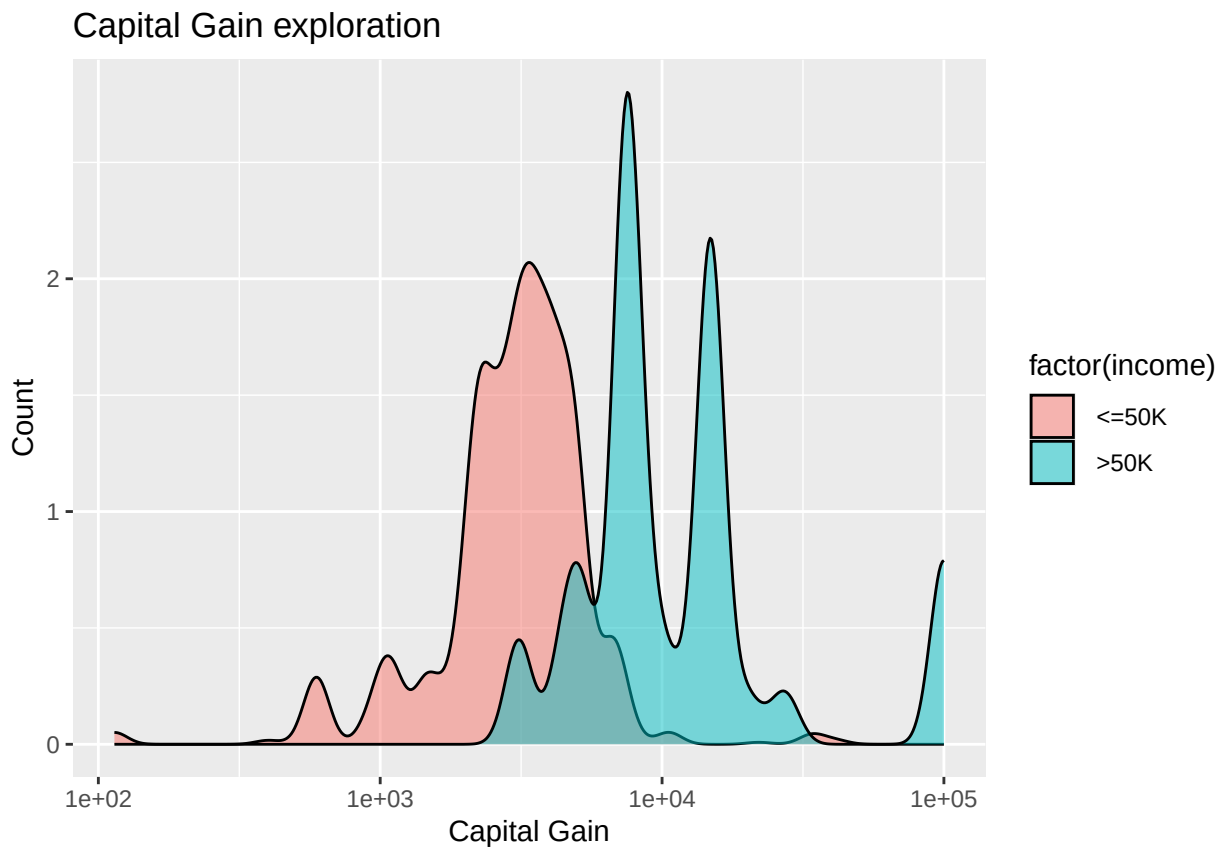
```
# sex (M is 1 F is 0) exploration
data <- data %>%
  mutate(sex = as.factor(sex),
         sex = ifelse(sex == " Male",TRUE,FALSE)) %>%
  rename(sex_M = sex)

data %>%
  ggplot(aes(x=sex_M, fill=income)) +
  geom_bar(position = "dodge") +
  labs(x = "Male Sex (True is Male)", y = "Count", title = "Sex exploration")
```



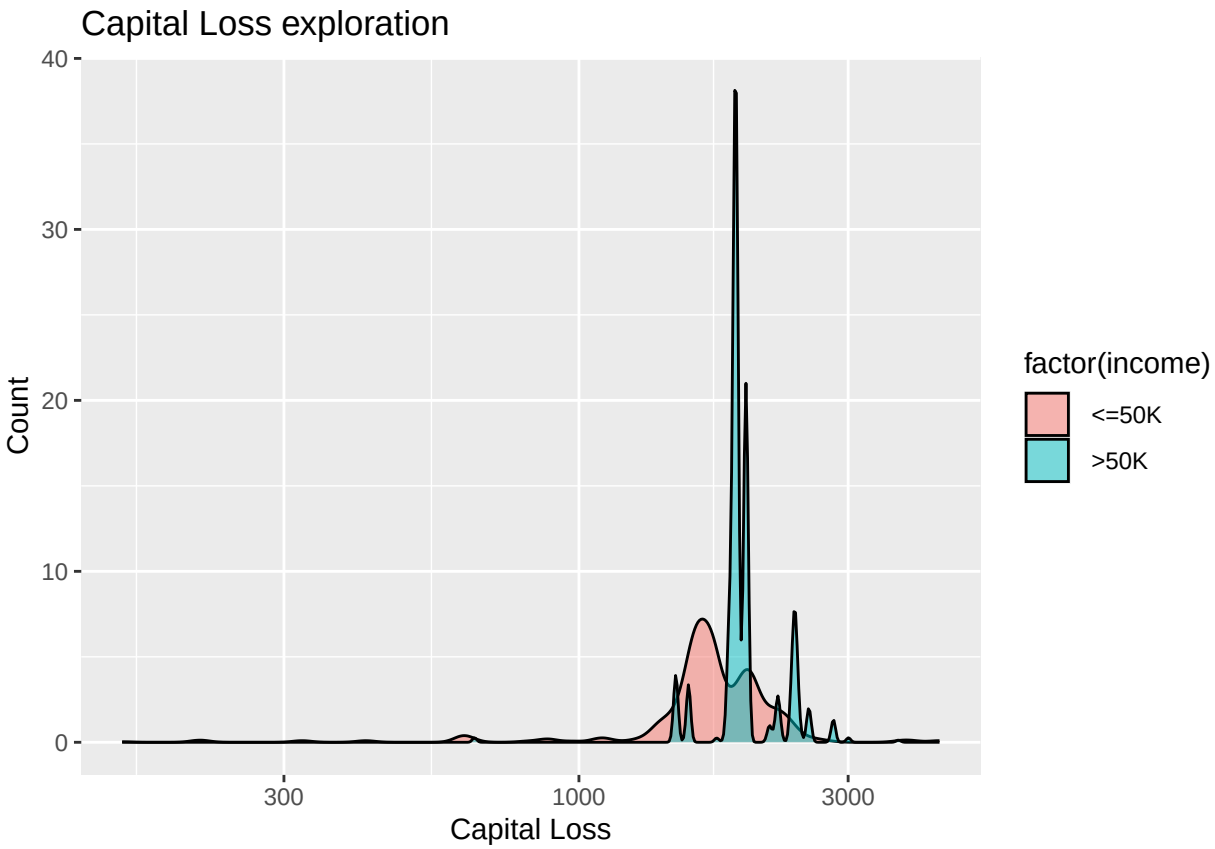
Capital Gain Exploration

```
# capital-gain exploration
data %>%
  ggplot(aes(x = `capital_gain`, fill = factor(income))) +
  geom_density(alpha = 0.5) +
  scale_x_log10() +
  labs(x = "Capital Gain", y = "Count", title = "Capital Gain exploration")
```



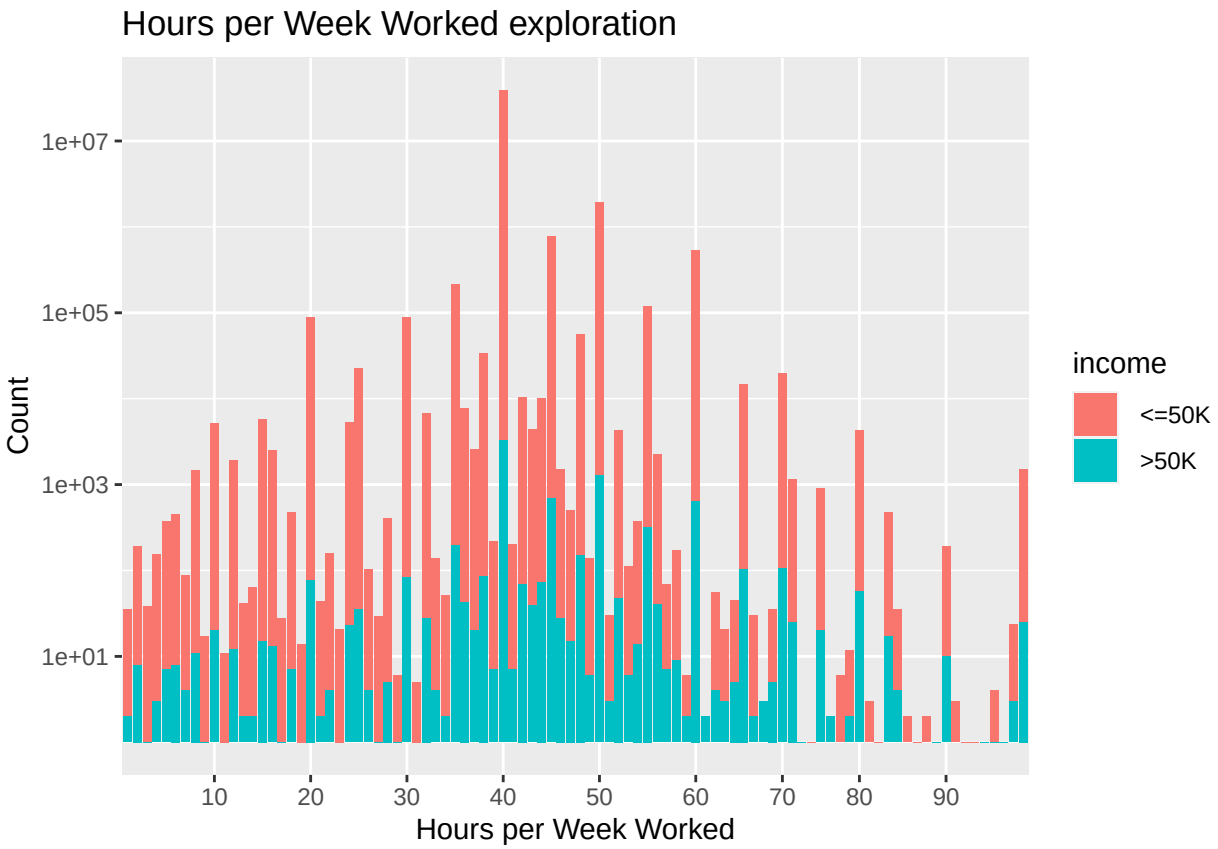
Capital Loss Exploration

```
# capital-loss exploration
data %>%
  ggplot(aes(x=`capital_loss`, fill = factor(income))) +
  geom_density(alpha = 0.5) +
  scale_x_log10() +
  labs(x = "Capital Loss", y = "Count", title = "Capital Loss exploration")
```



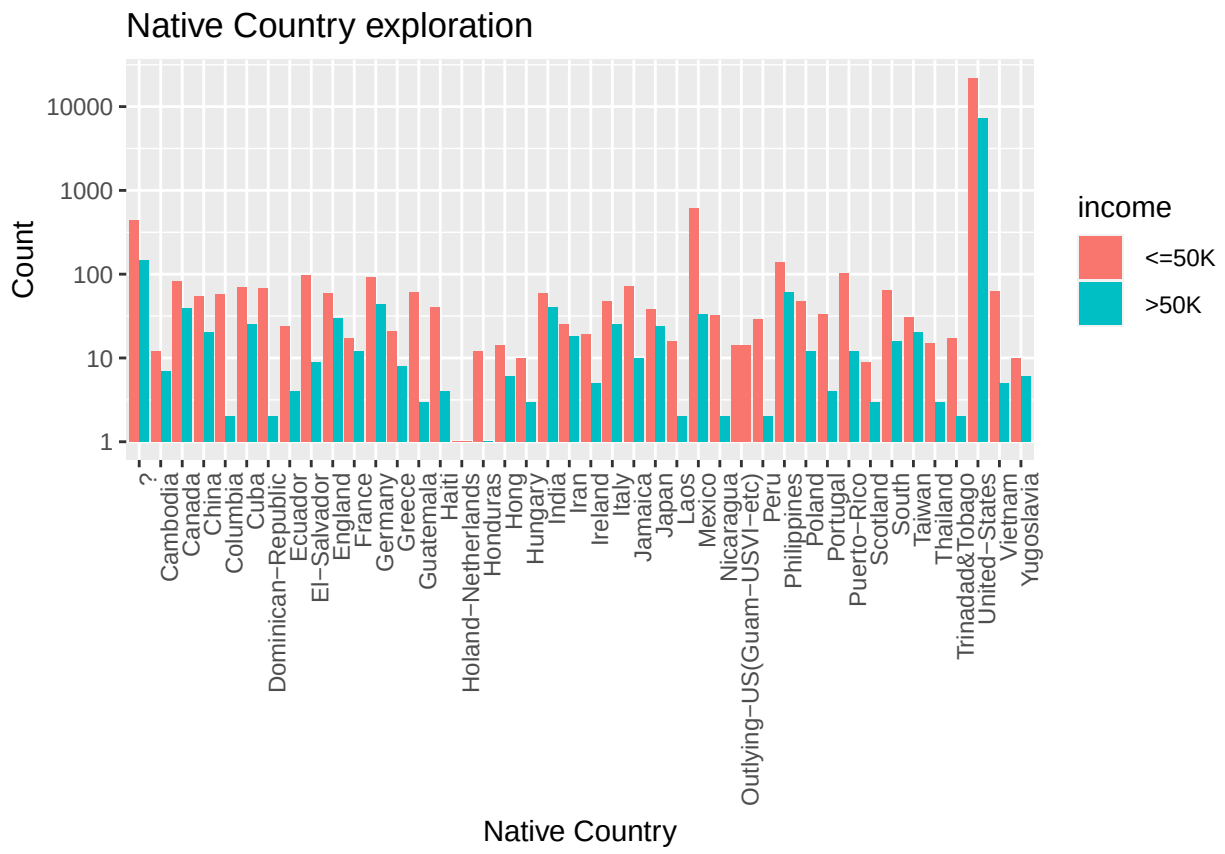
Hour per Week Worked Exploration

```
# hours-per-week exploration
data %>%
  ggplot(aes(x=as.factor(`hours_per_week`), fill=income)) +
  geom_bar(position = "stack") +
  scale_y_log10() +
  scale_x_discrete(breaks = seq(0, 100, by = 10)) +
  labs(x = "Hours per Week Worked", y = "Count", title = "Hours per Week Worked exploration")
```



Native Country Exploration

```
#native-country exploration
data %>%
  ggplot(aes(x=`native_country`, fill=income)) +
  geom_bar(position = "dodge") +
  scale_y_log10() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Native Country", y = "Count", title = "Native Country exploration")
```



Data cleansing

```
# Changes data to factor for analysis
data <- data %>%
  mutate(age = as.factor(age),
         workclass = as.factor(workclass),
         education = as.factor(education),
         education_num = as.factor(education_num),
         `marital_status`= as.factor(`marital_status`),
         occupation = as.factor(occupation),
         relationship = as.factor(relationship),
         race = as.factor(race),
         hours_per_week = as.factor(hours_per_week),
         `native_country` = as.factor(`native_country`))

# Removes Final Weight variable
data <- data %>%
  select(-fnlwgt)

# Makes income a factor
data <- data %>%
  mutate(income = as.factor(income))

# Saves original income names
```

```r
orig_income_names <- unique(data$income)

# Edits factor level names for income to make them more machine learning friendly
levels(data$income) <- make.names(levels(data$income))
```

The graphs above indicate that all variables, except for final weight, exhibit substantial variation across the two income groups. The graph for final weight shows significant overlap between the two income groups, which suggests a similar distribution of final weight values across both income groups. Given this observation, we will remove the final weight variable from our dataset during the model building process.

Additionally, after reviewing the above graphs, it appears beneficial to approach this problem using multiple models, since each variable has a different amount of variation across income groups. Specifically, we will implement Random Forest, XGBoost, KNN, and Decision Tree models. Following the development of these models, we will create an Ensemble model, providing a fifth potential option for predicting income levels.

To properly evaluate how our models handle new data, we will split the dataset into a training and test set. Given the dataset's size of approximately 32,500 rows, we will allocate 80% of the data for training and the remaining 20% for testing. This split ensures that our models have ample data for training while preserving enough test data for accurate testing and evaluation.

Splits data into training and test sets

```r
# Creates data partition
set.seed(1)
test_index <- createDataPartition(data$income, times = 1, p = 0.2, list = FALSE)

train_df <- data[-test_index,]
test_df_1 <- data[test_index,]

### Ensure that the test data only contains data present in the train_df ###

test_df <- test_df_1 %>%
  semi_join(train_df, by = c("age", "workclass", "education", "occupation", "native_country"))

### Adds removed test rows to training data ###

removed_data <- anti_join(test_df_1,test_df)
train_df <- rbind(train_df,removed_data)

rm(removed_data, test_df_1, test_index, url, column_names)
```

Now that we have created the testing and training datasets, we will move on to creating our 5 machine learning models.

Random Forest Model

```
#################################################################
# Evaluates data based off of the training set using the
# Random Forest model
#################################################################
# Define the training control & 3-fold cross-validation & sets binary classifier model
set.seed(5)
train_control <- trainControl(method = "cv", number = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)

# Creates rfm machine learning model
rf_fit <- train(income ~ ., data = train_df, method = "rf",
                trControl = train_control, tuneLength = 5, metric = "ROC")

predict_rfm <- predict(rf_fit, newdata = test_df)

# Checks accuracy, precision (PPV), recall (sensitivity/TPR), prevalence, and balanced accuracy
# (avg of specificity and sensitivity)
RF_confusionMatrix <- confusionMatrix(data = predict_rfm, reference = test_df$income)


#F_1 score b=1 default
RF_F_1 <- F_meas(data = predict_rfm, reference = test_df$income)



# Tabulates model results
results_table <- tibble(`Machine Learning Model` = "Random Forest",
                        Accuracy = RF_confusionMatrix$overall["Accuracy"],
                        `Balanced Accuracy`= RF_confusionMatrix$byClass["Balanced Accuracy"],
                        `F_1 Score` = RF_F_1,
                        Sensitivity = RF_confusionMatrix$byClass["Sensitivity"],
                        Specificity = RF_confusionMatrix$byClass["Specificity"],
                        PPV = RF_confusionMatrix$byClass["Pos Pred Value"],
                        `<=50K Prevalence`= RF_confusionMatrix$byClass["Prevalence"])

results_table %>% kable()
```

Machine Learning Model	Accuracy	Balanced Accuracy	F_1 Score	Sensitivity	Specificity	PPV	<=50K Prevalence
Random Forest	0.8604119	0.7770786	0.9108543	0.9363562	0.617801	0.8867046	0.7615977

In the above code, we created the Random Forest model with 3-fold cross-validation. To evaluate the effectiveness of this model, we will focus primarily on balanced accuracy and the F1 score. Since the dataset is imbalanced between the two income groups, with income less than or equal to $50,000 USD being much more prevalent, these metrics provide a more accurate assessment of model performance compared to accuracy alone, which can be skewed by unbalanced datasets. Next we will create an XGBoost model.

XGBoost Model

```
#################################################################
# Evaluates data based off of the training set using the
# XGBoost model
#################################################################
# Set up training control for xgb
set.seed(5)
train_control <- trainControl(method = "cv", number = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)

#Creates model & predictions
xgb_fit <- train(income ~ ., data = train_df, method = "xgbTree",
                 trControl = train_control, metric = "ROC",
                 verbosity = 0)

predict_xgb <- predict(xgb_fit, newdata = test_df)

# Checks accuracy, precision (PPV), recall (sensitivity/TPR), prevalence, and balanced accuracy
# (avg of specificity and sensitivity)
XGB_confusionMatrix <- confusionMatrix(data = predict_xgb, reference = test_df$income)


#F_1 score b=1 default
XGB_F_1 <- F_meas(data = predict_xgb, reference = test_df$income)

# Tabulates model results
results_table <- rbind(
  results_table,
  tibble(`Machine Learning Model` = "XGBoost model",
         `Balanced Accuracy`= XGB_confusionMatrix$byClass["Balanced Accuracy"],
         Accuracy = XGB_confusionMatrix$overall["Accuracy"],
         `F_1 Score` = XGB_F_1,
         Sensitivity = XGB_confusionMatrix$byClass["Sensitivity"],
         Specificity = XGB_confusionMatrix$byClass["Specificity"],
         PPV = XGB_confusionMatrix$byClass["Pos Pred Value"],
         `<=50K Prevalence`= XGB_confusionMatrix$byClass["Prevalence"]))

results_table %>% kable()
```

Machine Learning Model	Accuracy	Balanced Accuracy	F_1 Score	Sensitivity	Specificity	PPV	<=50K Prevalence
Random Forest	0.8604119	0.7770786	0.9108543	0.9363562	0.6178010	0.8867046	0.7615977
XGBoost model	0.8803828	0.7982824	0.9240322	0.9552035	0.6413613	0.8948311	0.7615977

After incorporating the XGBoost model into our results table, we can see that it performs slightly better on this dataset compared to the Random Forest model. Specifically, we noted improvements across the three main evaluation metrics: accuracy, balanced accuracy, and F1 score. These enhancements indicate that the XGBoost model provides a more accurate and balanced classification of the income groups.

Given these promising results, we will now proceed to develop a K-Nearest Neighbors (KNN) model to further explore its potential for this classification task.

K-Nearest Neighbors (KNN) Model

```
#################################################################
# Evaluates data based off of the training set using the
# KNN Model
#################################################################
# Define the training control using 3-fold cross-validation
set.seed(5)

train_control <- trainControl(method = "cv", number = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)

tune <- data.frame(k = seq(1, 20, by = 2))

# Makes KNN model
knn_fit <- train(income ~ ., data = train_df, method = "knn", tuneGrid = tune,
                 trControl = train_control, metric = "ROC")
knn_fit$bestTune
```

```
##     k
## 10 19
```

```
# Predicts with KNN model
predict_knn <- predict(knn_fit, newdata = test_df)

# Checks accuracy, precision (PPV), recall (sensitivity/TPR), prevalence, and balanced accuracy
# (avg of specificity and sensitivity)
KNN_confusionMatrix <- confusionMatrix(data = predict_knn, reference = test_df$income)


#F_1 score b=1 default
KNN_F_1 <- F_meas(data = predict_knn, reference = test_df$income)

# Tabulates model results
results_table <- rbind(
  results_table,
  tibble(`Machine Learning Model` = "KNN",
         Accuracy = KNN_confusionMatrix$overall["Accuracy"],
         `Balanced Accuracy`= KNN_confusionMatrix$byClass["Balanced Accuracy"],
         `F_1 Score` = KNN_F_1,
         Sensitivity = KNN_confusionMatrix$byClass["Sensitivity"],
         Specificity = KNN_confusionMatrix$byClass["Specificity"],
         PPV = KNN_confusionMatrix$byClass["Pos Pred Value"],
         `<=50K Prevalence`= KNN_confusionMatrix$byClass["Prevalence"]))

results_table %>% kable()
```

Machine Learning Model	Accuracy	Balanced Accuracy	F_1 Score	Sensitivity	Specificity	PPV	<=50K Prevalence
Random Forest	0.8604119	0.7770786	0.9108543	0.9363562	0.6178010	0.8867046	0.7615977
XGBoost model	0.8803828	0.7982824	0.9240322	0.9552035	0.6413613	0.8948311	0.7615977
KNN	0.8743499	0.7949212	0.9198514	0.9467359	0.6431065	0.8944516	0.7615977

When comparing the K-Nearest Neighbors (KNN) model to the previous two, we can see that it performs slightly worse on this dataset compared to the XGBoost model but still better than the Random Forest Model. When examining accuracy, balanced accuracy, and F1 scoreof the KNN model, the results seem to fall in between the previous two models. This means while this model is not the best fit for this dataset, it is still a strong and viable option for classifying the income groups.

Now we will develop a Decision Tree (Rpart) model to continue to explore further classification models.

Decision Tree (Rpart) Model

```
#################################################################
# Evaluates data based off of the training set using the
# Decision Tree Model
#################################################################
# Define the training control & tuning parameters using 3-fold cross-validation
set.seed(5)

train_control <- trainControl(method = "cv", number = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)

tune <- expand.grid(cp = seq(0, 0.05, by = 0.01))

# Makes Decision Tree model
rpart_fit <- train(income ~ ., data = train_df, method = "rpart", tuneGrid = tune,
                   trControl = train_control, metric = "ROC")

# Predicts with KNN model
predict_rpart <- predict(rpart_fit, newdata = test_df)

# Checks accuracy, precision (PPV), recall (sensitivity/TPR), prevalence, and balanced accuracy
# (avg of specificity and sensitivity)
RPART_confusionMatrix <- confusionMatrix(data = predict_rpart, reference = test_df$income)


#F_1 score b=1 default
RPART_F_1 <- F_meas(data = predict_rpart, reference = test_df$income)

# Tabulates model results
results_table <- rbind(
  results_table,
  tibble(`Machine Learning Model` = "Decision Tree",
```

```
          Accuracy = RPART_confusionMatrix$overall["Accuracy"],
          `Balanced Accuracy`= RPART_confusionMatrix$byClass["Balanced Accuracy"],
          `F_1 Score` = RPART_F_1,
          Sensitivity = RPART_confusionMatrix$byClass["Sensitivity"],
          Specificity = RPART_confusionMatrix$byClass["Specificity"],
          PPV = RPART_confusionMatrix$byClass["Pos Pred Value"],
          `<=50K Prevalence`= RPART_confusionMatrix$byClass["Prevalence"]))

results_table %>% kable()
```

Machine Learning Model	Accuracy	Balanced Accuracy	F_1 Score	Sensitivity	Specificity	PPV	<=50K Prevalence
Random Forest	0.8604119	0.7770786	0.9108543	0.9363562	0.6178010	0.8867046	0.7615977
XGBoost model	0.8803828	0.7982824	0.9240322	0.9552035	0.6413613	0.8948311	0.7615977
KNN	0.8743499	0.7949212	0.9198514	0.9467359	0.6431065	0.8944516	0.7615977
Decision Tree	0.8635323	0.7809256	0.9128818	0.9388145	0.6230366	0.8883432	0.7615977

When examining the Decision Tree (Rpart) model, we can see that it performs similarly to the Random Forest model, but with slight improvements across the three main evaluation metrics: accuracy, balanced accuracy, and F1 score. While this indicate that the Decision Tree (Rpart) model may provides a more accurate and balanced classification of the income groups than the Random Forest model, it still preforms worse than the XGBoost and K-Nearest Neighbors (KNN) models.

Finally, we will create an Ensemble model using the previous 4 models.

Ensemble Model

```
#####################################################################
# Creates an Ensemble of the models.
# Then evaluates the data based off of the training set
#####################################################################

# Calculates ensemble
set.seed(5)

#1 = "X...50K" (<=50K)
ensemble <- cbind(rfm = predict_rfm == "X...50K", XGBoost = predict_xgb == "X...50K",
                  KNN = predict_knn == "X...50K",rpart = predict_rpart == "X...50K")

# Predicts "X...50K" (<=50K), X..50K (>50K)
ensemble_predict <- ifelse(rowMeans(ensemble) >= .5, "X...50K", "X..50K")

# Checks accuracy, precision (PPV), recall (sensitivity/TPR), prevalence, and balanced accuracy
# (avg of specificity and sensitivity)
ensemble_confusionMatrix <- confusionMatrix(data = as.factor(ensemble_predict),
                                            reference = test_df$income)
```

```r
#F_1 score b=1 default
ENSEMBLE_F_1 <- F_meas(data = as.factor(ensemble_predict), reference = test_df$income)

# Tabulates model results
results_table <- rbind(
  results_table,
  tibble(`Machine Learning Model` = "RFM, XGB, KNN, & Rpart Ensemble",
         Accuracy = ensemble_confusionMatrix$overall["Accuracy"],
         `Balanced Accuracy`= ensemble_confusionMatrix$byClass["Balanced Accuracy"],
         `F_1 Score` = ENSEMBLE_F_1,
         Sensitivity = ensemble_confusionMatrix$byClass["Sensitivity"],
         Specificity = ensemble_confusionMatrix$byClass["Specificity"],
         PPV = ensemble_confusionMatrix$byClass["Pos Pred Value"],
         `<=50K Prevalence`= ensemble_confusionMatrix$byClass["Prevalence"]))

results_table %>% kable()
```

Machine Learning Model	Accuracy	Balanced Accuracy	F_1 Score	Sensitivity	Specificity	PPV	<=50K Prevalence
Random Forest	0.8604119	0.7770786	0.9108543	0.9363562	0.6178010	0.8867046	0.7615977
XGBoost model	0.8803828	0.7982824	0.9240322	0.9552035	0.6413613	0.8948311	0.7615977
KNN	0.8743499	0.7949212	0.9198514	0.9467359	0.6431065	0.8944516	0.7615977
Decision Tree	0.8635323	0.7809256	0.9128818	0.9388145	0.6230366	0.8883432	0.7615977
RFM, XGB, KNN, & Rpart Ensemble	0.8743499	0.7787360	0.9209838	0.9614859	0.5959860	0.8837560	0.7615977

Lastly, when evaluating the Ensemble model, it appears that this model is not the best fit for this dataset. While the F1 score is the second highest among the five models, balanced accuracy score is the second lowest. This suggests that although the Ensemble model captures some important aspects of the data, there are other models that perform better overall. We will discuss these findings further in the conclusion.

Conclusion:

In this analysis, we explored various machine learning models to predict income levels. By iteratively creating and evaluating each model, we identified each of their strengths and weaknesses. Now to rank these models, we will primarily use the balanced accuracy score, as it is the average of sensitivity and specificity. This score provides a more comprehensive measure of a model's performance by accounting for both the true positive rate (sensitivity) and the true negative rate (specificity), therefore giving a balanced view of how well the model can distinguish between the two classes, regardless of the unbalanced nature of the dataset. In cases where models have similar balanced accuracy scores, we will use the F1 score to further differentiate and determine the best-performing model.

Given the above, the ranking of the models is as follows:

1) XGBoost model: Balanced accuracy of 0.798 & F1 of 0.924

2) K-Nearest Neighbors (KNN) model: Balanced accuracy of 0.795 & F1 of 0.920

3) Decision Tree (Rpart) model: Balanced accuracy of 0.781 & F1 of 0.913

4) Ensemble model: Balanced accuracy of 0.778 & F1 of 0.921

5) Random Forest model: Balanced accuracy of 0.777 & F1 of 0.911

For this project, while the models are all viable options, the best fit for this dataset seems to be the XGBoost model. This model not only has the highest balanced accuracy, it also has the highest F1 score, and the highest regular accuracy. While this is a viable and strong model, for the purpose of future iterations, there is still potential to increase the accuracy of this model. I will list 2 potential ways to improve this model further:

1) For future iterations of this model, creating new features or transforming existing ones can provide the model with more relevant information. For example, by completing a feature importance analysis on our current models, we might be able to identify which features are more strongly correlated with income. This insight can guide the creation of new features or the transformation of existing ones, helping the model capture complex relationships in the data more effectively.

2) Hyper-parameter Tuning: While the current model may already be tuned, more extensive hyper-parameter optimization and further use of cross-validation could potentially yield better model performance.

While there may be room for improvement, the current XGBoost model is a strong foundation for predicting income class. Also, the insights gained from this model can inform future research and models, helping to increase the speed and efficiency of model creation. Overall, this is a very strong classification model, and can currently be used to accurately predict an individual's income level.