HarvardX Data Science Final Project: A Classification Task Using Machine Learning Techniques

Gretta Digbeu

July 28th, 2019

A.Introduction

This machine learning exercise is the final component in the capstone course of the Harvard edX Data Science online professional certificate program. Students pursuing the verified track are asked to apply predictive machine learning techniques using a publicly available dataset to solve a problem of their choice.

We chose to explore the 1996 *Adult* dataset, also known as the *Census Income* dataset, available on the UCI's Machine Learning Repository. The data was extracted by Silicon Valley researchers (Ronny Kohavi and Barry Becker) from the 1994 Current Population Survey (CPS) data held in the Census database. It is already partitioned into training and testing sets containing 32,561 and 16,281 observations respectively (30,162 and 15,060 after removing unknowns), and includes 14 attributes. The variable of interest, gross income, was discretized into two ranges with a threshold of $50,000. The data is therefore ideal for a multivariate classification exercise with the goal of predicting this binary outcome variable.

Before exploring the data for trends and patterns, let us present a breakdown of the attributes contained therein:

attribute	type	description
age	continuous	Age in years
workclass	categorical	Class of worker
final_weight	continuous	Demographic weight
education_level	categorical	Highest level of education achieved
education_num	continuous	Total years of education
marital_status	categorical	Marital status
occupation	categorical	Type of occupation
relationship	categorical	Relationship of survey responder to the head of household
race	categorical	Ethnic origin
sex	binary	Biological sex
capital_gain	continuous	Profits made from the sale of real estate, investments and personal property
capital_loss	continuous	Losses incurred when capital assets decrease in value
hours_per_week	continuous	Average number of hours worked per week
native_country	categorical	Country of origin
income_category	categorical	Income in relation to 50k

We note that there are 14 attributes, 6 of which are continuous and 8 are categorical. This includes the *weight* feature assigned to each observation by the researchers at the Population Division of the Census Bureau. These weights are controlled to independent estimates of the civilian non-institutional population of the US, using 3 sets of controls: a single cell estimate of the population 16 and up for each state; controls for Hispanic origin by age and sex; and controls by race, age and sex. People with similar demographic characteristics should therefore have similar weights, with one important caveat: because the CPS sample is a collection of 51 separate state samples, each with its own probability of selection, such similarities only apply to observations

from the same state.

B.Exploratory Data Analysis

In order to identify trends and patterns in the census data, we must conduct some exploratory data analysis. Our findings will guide the methods employed in this exercise, and help us determine whether any transformations must be made to the data. These include but are not limited to: removing attributes highly correlated with other features, applying log transformations or scaling/standardizing the values of continuous attributes, and removing attributes with very few unique values or close to zero variation in our outcome variable.

I.Data at a Glance

After importing the training and testing sets separately and removing the unknowns, we assign the column names listed above. Note that for ease of understanding, these variable names differ slightly from the column names listed on the UCI Machine Learning Repository. Next, we examine the structure of each subset to ensure that they are structurally identical because we want to explore these trends over the entire sample, as opposed to examining the training and testing sets separately. The results are as follows:

Training Set

```
## 'data.frame':    30162 obs. of  15 variables:
##  $ age            : num  39 50 38 53 28 37 49 52 ...
##  $ workclass      : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4 4 4 4 6 ...
##  $ final_weight   : int  77516 83311 215646 234721 338409 284582 160187 209642 ...
##  $ education_level: Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 ...
##  $ education_num  : int  13 13 9 7 13 14 5 9 ...
##  $ marital_status : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 ...
##  $ occupation     : Factor w/ 14 levels "Adm-clerical",..: 1 4 6 6 10 4 8 4 ...
##  $ relationship   : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 ...
##  $ race           : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 ...
##  $ sex            : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 ...
##  $ capital_gain   : num  2174 0 0 0 ...
##  $ capital_loss   : num  0 0 0 0 0 0 0 0 ...
##  $ hours_per_week : int  40 13 40 40 40 40 16 45 ...
##  $ native_country : Factor w/ 41 levels "Cambodia","Canada",..: 39 39 39 39 5 39 23 39 ...
##  $ income_category: Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 ...
##  - attr(*, "na.action")= 'omit' Named int  15 28 39 52 62 70 78 94 ...
##   ..- attr(*, "names")= chr  "15" "28" "39" ...
```

Testing Set

```
## 'data.frame':    15060 obs. of  15 variables:
##  $ age            : num  25 38 28 44 34 63 24 55 ...
##  $ workclass      : Factor w/ 8 levels "Federal-gov",..: 4 4 2 4 4 6 4 4 ...
##  $ final_weight   : int  226802 89814 336951 160323 198693 104626 369667 104996 ...
##  $ education_level: Factor w/ 16 levels "10th","11th",..: 2 12 8 16 1 15 16 6 ...
##  $ education_num  : int  7 9 12 10 6 15 10 4 ...
##  $ marital_status : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 3 3 5 3 5 3 ...
##  $ occupation     : Factor w/ 14 levels "Adm-clerical",..: 7 5 11 7 8 10 8 3 ...
##  $ relationship   : Factor w/ 6 levels "Husband","Not-in-family",..: 4 1 1 1 2 1 5 1 ...
##  $ race           : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 3 5 5 3 5 5 5 5 ...
##  $ sex            : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 2 ...
##  $ capital_gain   : num  0 0 0 7688 ...
##  $ capital_loss   : num  0 0 0 0 0 0 0 0 ...
```

```
##  $ hours_per_week : int  40 50 40 40 30 32 40 10 ...
##  $ native_country : Factor w/ 40 levels "Cambodia","Canada",..: 38 38 38 38 38 38 38 38 ...
##  $ income_category: Factor w/ 2 levels "<=50K.",">50K.": 1 1 2 2 1 2 1 1 ...
##  - attr(*, "na.action")= 'omit' Named int  5 7 14 20 23 36 66 76 ...
##   ..- attr(*, "names")= chr  "5" "7" "14" ...
```

Looking at the structure of each subset reveals that the levels of our variable of interest (income_category), are syntactically different. The levels in the testing set have an unnecessary period at the end of each expression.

We therefore recode the levels in the testing data so they match the training set:

Moreover, a quick look at the relationship between the **education_level** and **education_num** variables reveals that the latter does not correspond to the total number of years of education. Rather, it is simply a numeric representation of the **education_level** attribute:

education_level	education_num
Preschool	1
1st-4th	2
5th-6th	3
7th-8th	4
9th	5
10th	6
11th	7
12th	8
HS-grad	9
Some-college	10
Assoc-voc	11
Assoc-acdm	12
Bachelors	13
Masters	14
Prof-school	15
Doctorate	16

We can therefore remove the **education_num** feature, as it is a redundant variable that provides no additional information about our population. We remove it from both the testing and training data frames.

We can now proceed to bind the rows and explore the full dataset. Excluding the unknowns, our census sample has **45,222** observations.

```
## 'data.frame':    45222 obs. of  14 variables:
##  $ age            : num  39 50 38 53 28 37 49 52 ...
##  $ workclass      : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4 4 4 4 6 ...
##  $ final_weight   : int  77516 83311 215646 234721 338409 284582 160187 209642 ...
##  $ education_level: Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 ...
##  $ marital_status : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 ...
##  $ occupation     : Factor w/ 14 levels "Adm-clerical",..: 1 4 6 6 10 4 8 4 ...
##  $ relationship   : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 ...
##  $ race           : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 ...
##  $ sex            : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 ...
##  $ capital_gain   : num  2174 0 0 0 ...
##  $ capital_loss   : num  0 0 0 0 0 0 0 0 ...
##  $ hours_per_week : int  40 13 40 40 40 40 16 45 ...
##  $ native_country : Factor w/ 41 levels "Cambodia","Canada",..: 39 39 39 39 5 39 23 39 ...
##  $ income_category: Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 ...
```

First, we look at the prevalence of each class of our outcome variable. What is relative proportion of people

making more than $50,000 a year in our sample?

income_category	n	pct	percentage
<=50K	34014	0.752156	75.2%
>50K	11208	0.247844	24.8%

We note that at 75.4%, the prevalence of people with incomes below or equal to $50,000/year is much higher than that of people making more than $50,000/year. The implication of this uneven prevalence is that when it comes time to assessing an algorithm, instead of taking the balanced accuracy (F1 score) generated from the confusion matrix at face value, we should compute a weighted score using the f_meas() function, adjusting the *Beta* accordingly.

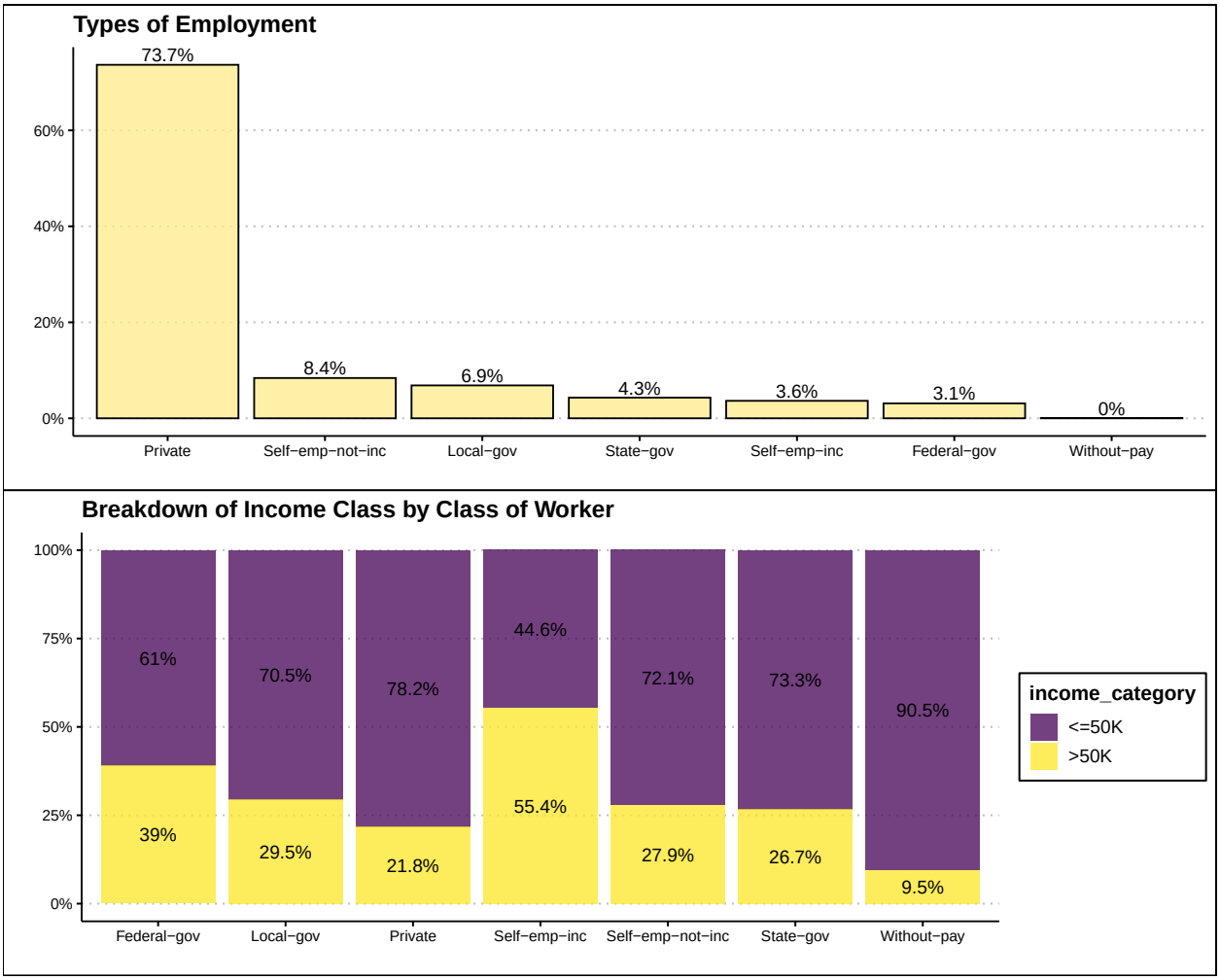
II.Deep Dive

II.1 Categorical Attributes We now proceed to take a deeper dive into a selection of features one by one, starting with the categorical variables. These are **workclass, education_level, marital_status, occupation, relationship, race, sex, and native_country**. These are all factor variables so we take a look at the levels contained in each.

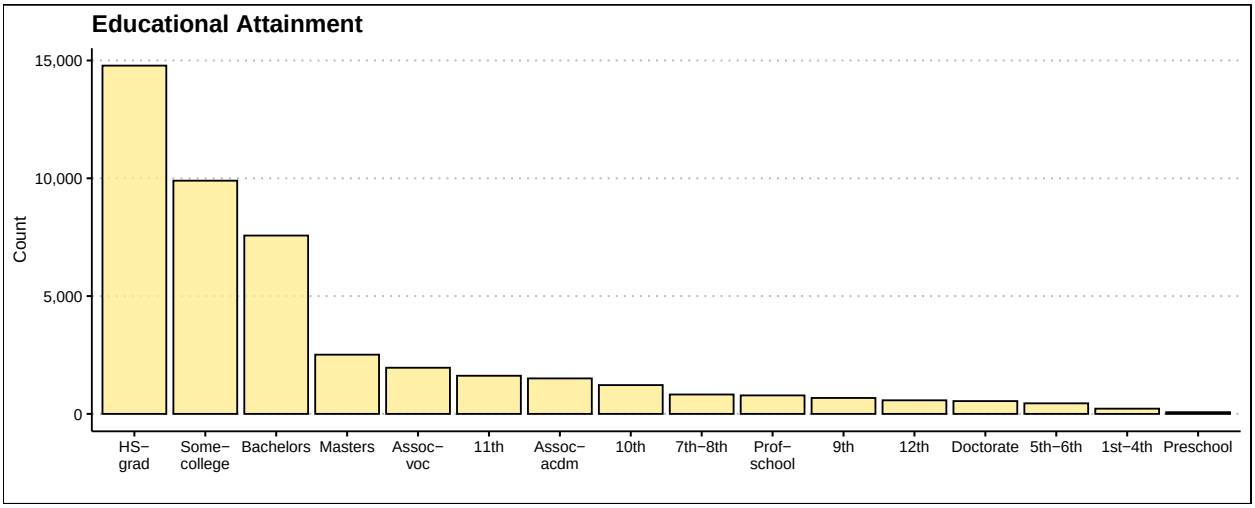
We notice that the variable **work_class** (class of worker) has an unused level ("never-worked"), so we drop it from the data.

```
##      Federal-gov        Local-gov     Never-worked          Private
##             1406             3100                0            33307
##     Self-emp-inc Self-emp-not-inc        State-gov      Without-pay
##             1646             3796             1946               21
```

Upon summarizing the data along this feature, we note that most survey respondents (73.7%) are private sector workers, and that the highest concentration of individuals earning more than $50K/year is found among the incorporated self-employed workers (self-emp-inc).



Next, we look at the **education_level** variable. We see that there is an unnecessary level of detail.

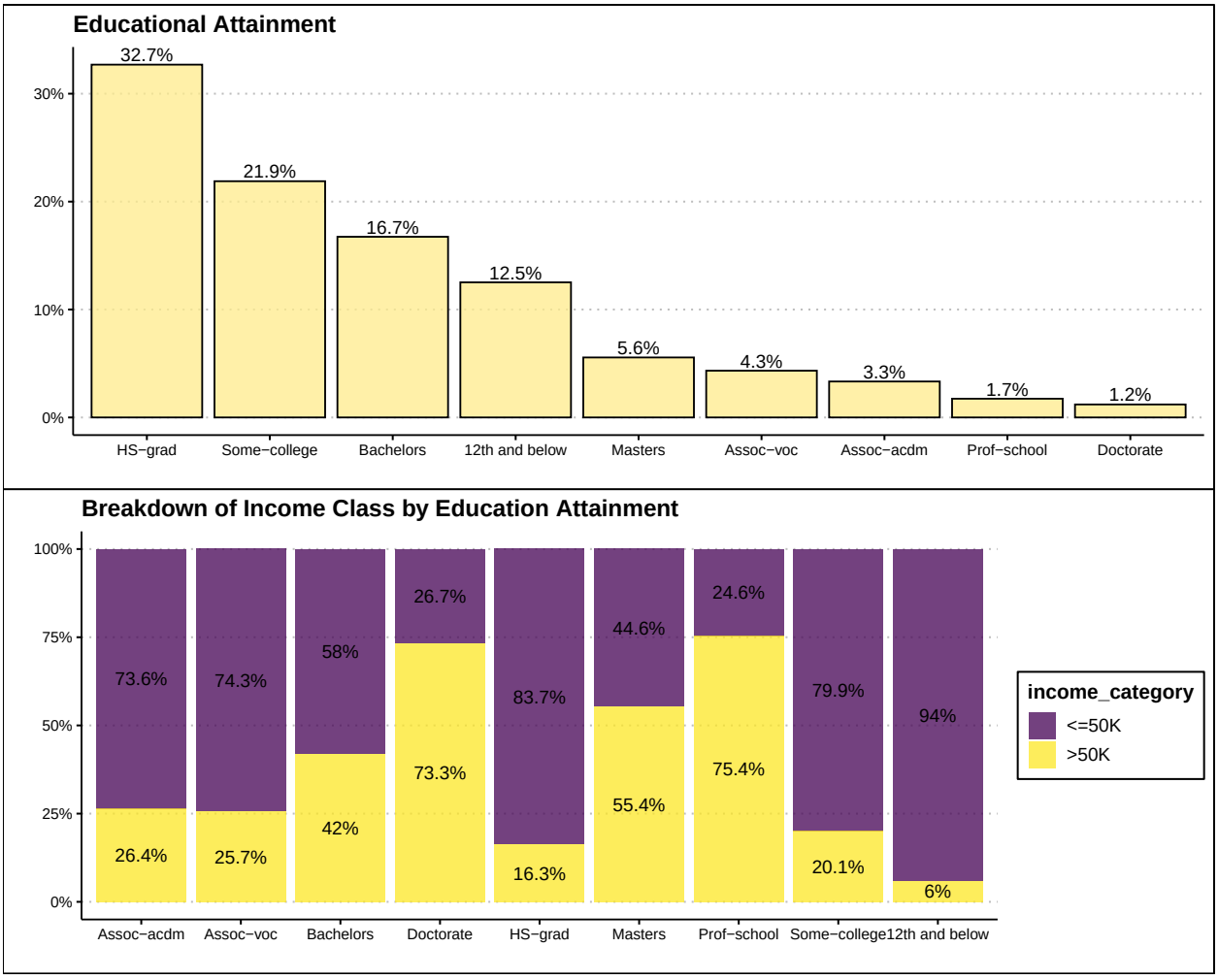


Distinguishing between people who completed 7th-8th grade, versus 10th, versus 12th grade, and so on, would create additional degrees of freedom that are unlikely to add value to our predictive model. We therefore recode the variable so that all levels below *High School Graduate* are combined into a single class which we

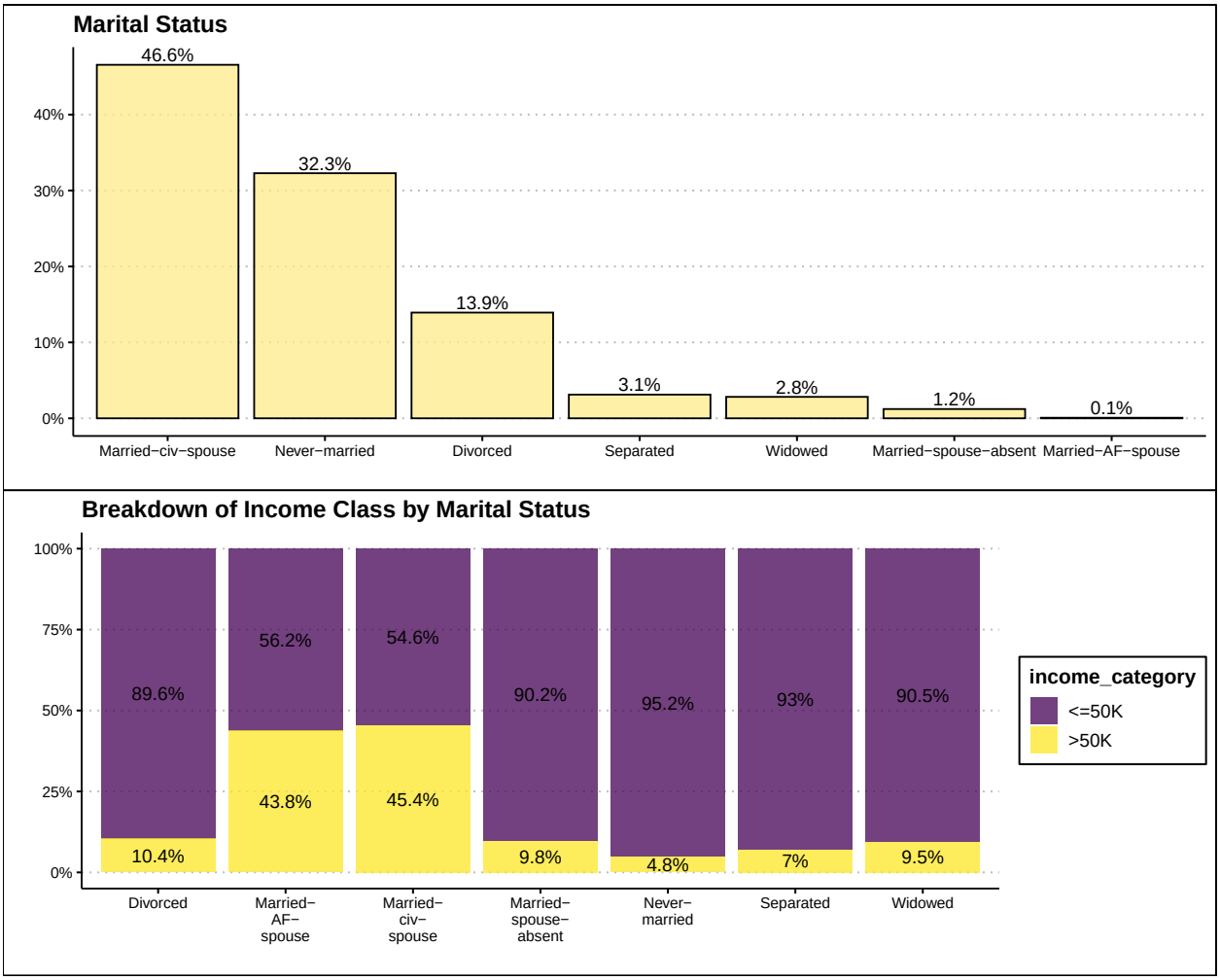
call *12th and below*.

```
## [1] "Assoc-acdm"     "Assoc-voc"      "Bachelors"      "Doctorate"
## [5] "HS-grad"        "Masters"        "Prof-school"    "Some-college"
## [9] "12th and below"
```

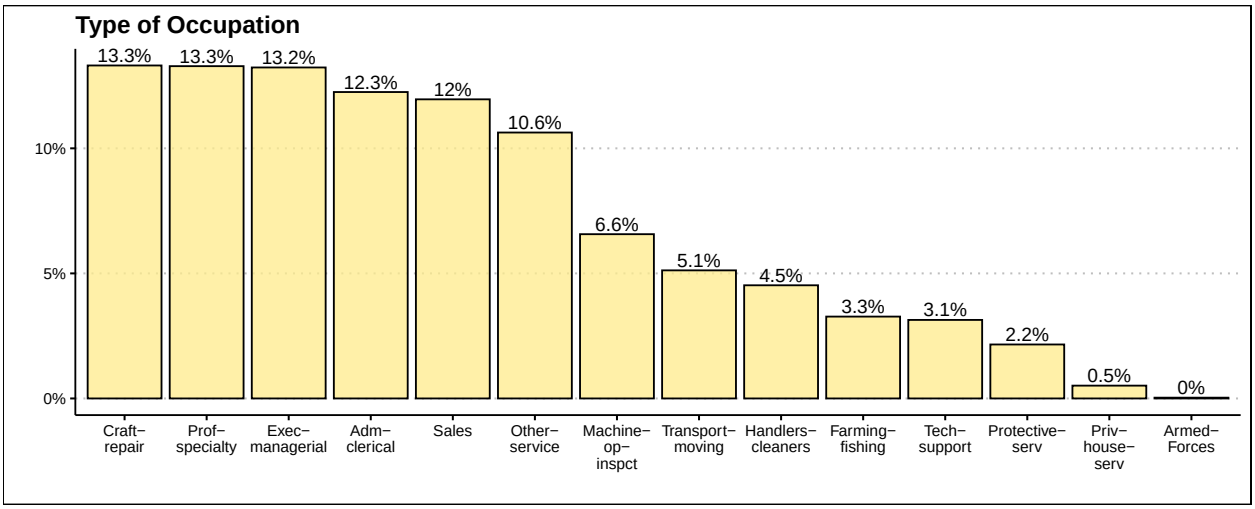
The resulting summary shows that high school graduates are the most common type of worker (32.7%), followed by people who completed some college (21.9%) and those who obtained a Bachelor's degree. We also see that the smallest proportion of people making more than \$50K/year is found among those with educational attainment of 12th grade and below, while the highest proportion (75.4%) is found among those with professional degrees beyond a Bachelor's degree (*Prof-school*). This is to be expected.

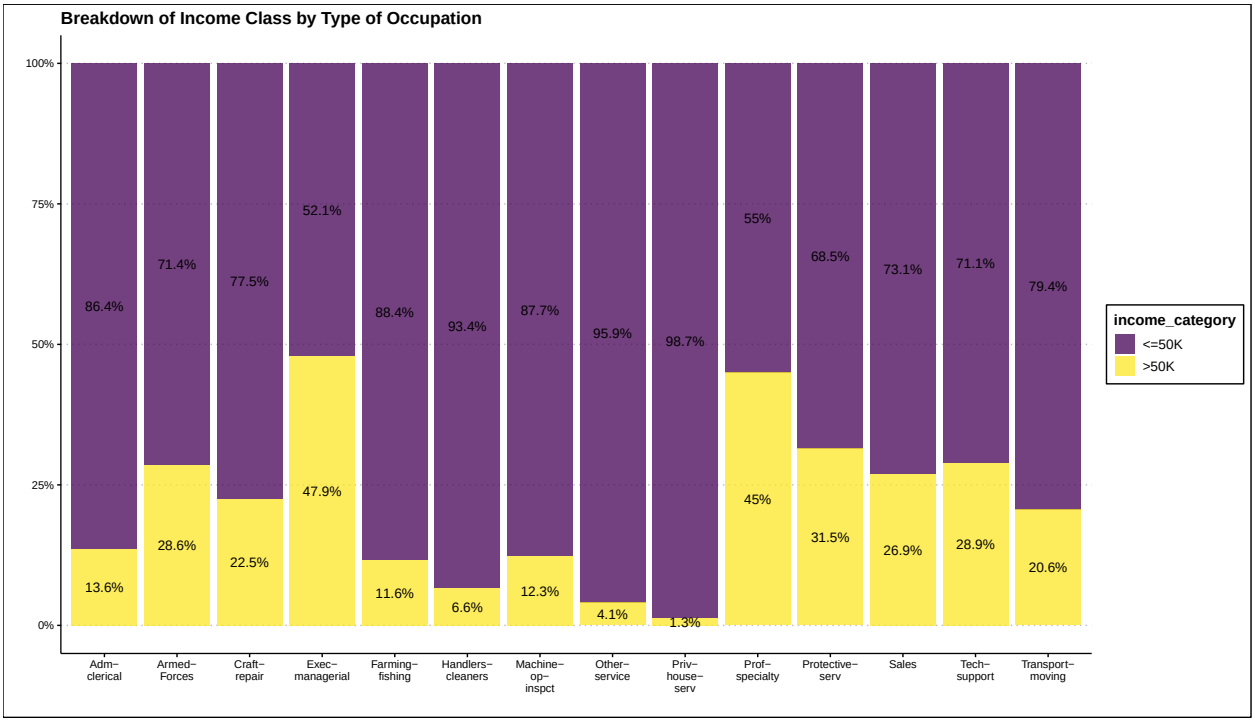


Next, we examine the **marital_status** variable. We find that civilian married individuals make up the greatest proportion of the data (46.6%), and that these same people are more likely to be earning more than \$50K/year (45.4%).

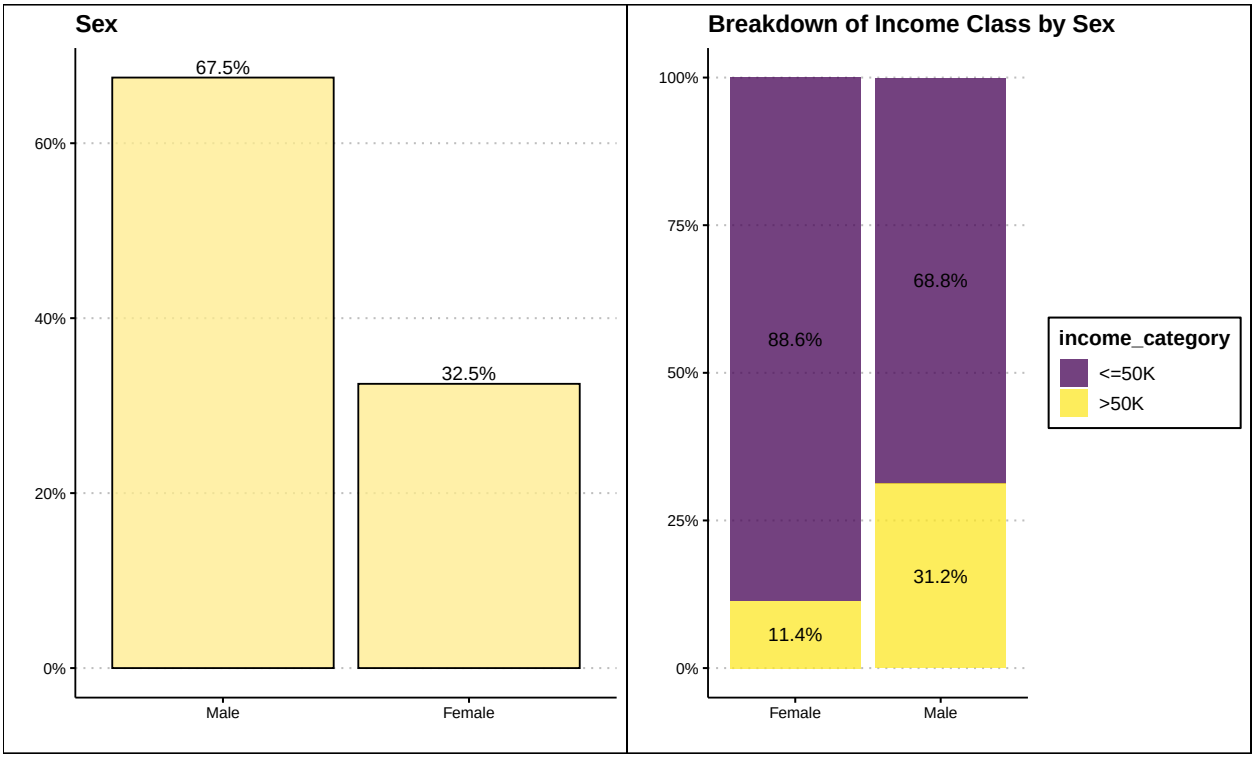


Looking at the **occupation** variable, we note that occupations are fairly evenly distributed in the sample, and that, unsurprisingly, individuals in managerial roles (*Exec-managerial*) and those with a white-collar specialty (*Prof-specialty*) are most likely to make above \$50K/year, with 47.9% and 45% of respondents in that income category respectively.





The **sex** variable reveals that the majority of survey responders (67.5%) are male, and that men are more likely than women to earn more than \$50K/year (32.1% versus 11.4%).



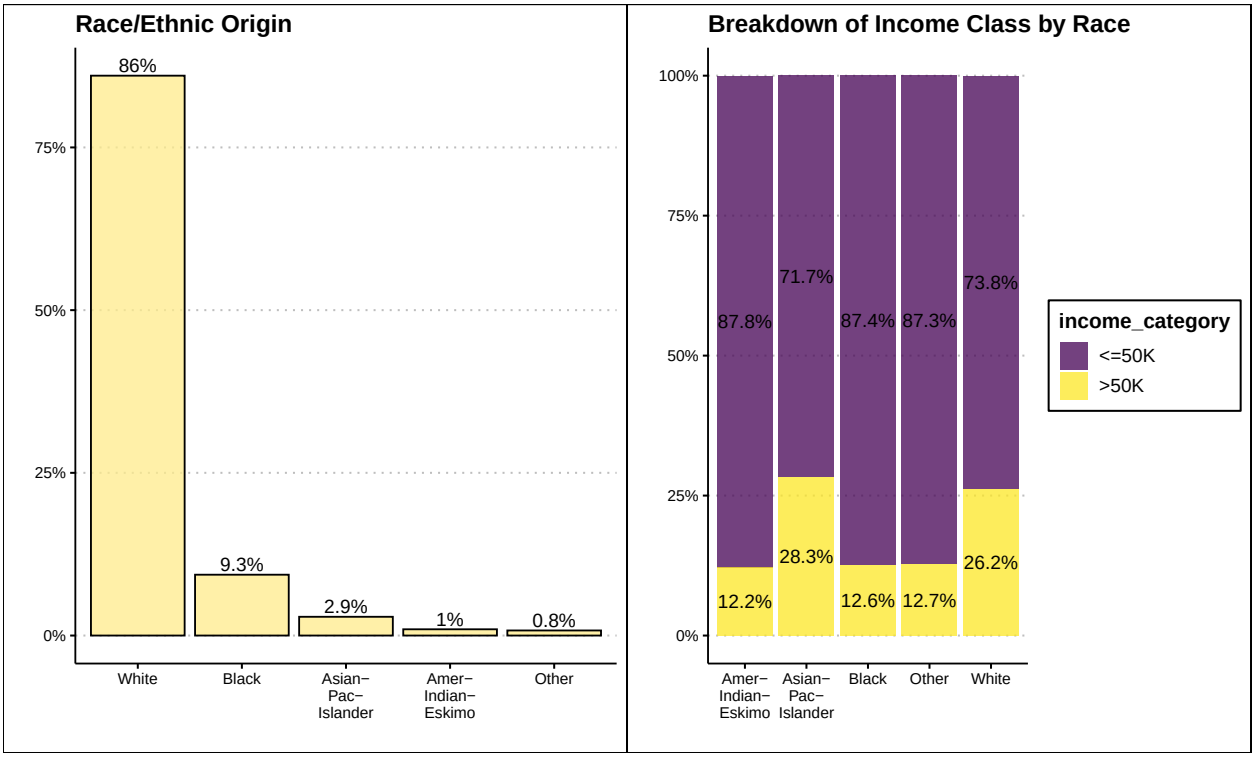
We skip over the **relationship** variable because the metadata does not provide enough information on what it means. While census bureau documentation explains that the **relationship** question denotes each

household member's relationship to the primary householder (the person who owns or rents the housing unit), the variable in our dataset does not contain a class which is the equivalent of "self". The levels and their frequencies are as follows:

```
##         Husband  Not-in-family Other-relative       Own-child       Unmarried
##           18666           11702            1349            6626            4788
##            Wife
##            2091
```

The absence of a class equivalent to "self" suggests that the UCI sample excludes primary householders altogether, yet the fact that most survey respondents are male (67.5%) and "husband" is the most common class of the **relationship** variable suggests otherwise. Another possibility is that **relationship** actually denotes the role of the individual in the household. This would explain the high prevalence of males and "husband" as the relationship type; however we do not have enough information to ascertain our hypothesis.

We move on to examine variability along the **race** variable, and note that an overwhelming majority (86%) of the survey respondents are white, and that the rate of people earning more than \$50K/year is highest among the Asian/Pacific Islander ethnic group (28.3%), followed by caucassians at 26.2%.



Finally, we examine the structure of the **native_country** variable. A summary of this categorical feature shows that it contains 41 different levels, most of which have very few observations.

```
##  [1] "Cambodia"                  "Canada"
##  [3] "China"                     "Columbia"
##  [5] "Cuba"                      "Dominican-Republic"
##  [7] "Ecuador"                   "El-Salvador"
##  [9] "England"                   "France"
## [11] "Germany"                   "Greece"
## [13] "Guatemala"                 "Haiti"
## [15] "Holand-Netherlands"        "Honduras"
## [17] "Hong"                      "Hungary"
```

```
## [19] "India"                    "Iran"
## [21] "Ireland"                  "Italy"
## [23] "Jamaica"                  "Japan"
## [25] "Laos"                     "Mexico"
## [27] "Nicaragua"                "Outlying-US(Guam-USVI-etc)"
## [29] "Peru"                     "Philippines"
## [31] "Poland"                   "Portugal"
## [33] "Puerto-Rico"              "Scotland"
## [35] "South"                    "Taiwan"
## [37] "Thailand"                 "Trinadad&Tobago"
## [39] "United-States"            "Vietnam"
## [41] "Yugoslavia"
```

Because keeping so many levels as an input is likely to add unnecessary degrees of freedom and bring down our model's performance, we collapse countries of origin into regional categories. We assign these categories to a new variable, **native_region**:

```
Asia <- c("Cambodia", "India", "Laos", "Thailand",
          "Vietnam", "Hong", "Iran", "China",
          "Japan", "Philippines", "Taiwan")

Europe <- c("France", "Italy", "Poland", "Scotland",
            "Germany", "Portugal", "Yugoslavia", "England",
            "Greece","Holand-Netherlands", "Hungary", "Ireland")

North_America <- c("Outlying-US(Guam-USVI-etc)",
                   "Canada", "United-States", "Puerto-Rico")

Latin_America_Carrib <- c("Columbia", "Ecuador", "Guatemala",
                          "Honduras", "Cuba", "El-Salvador", "Haiti",
                          "Jamaica", "Mexico", "Peru", "Trinadad&Tobago",
                          "Dominican-Republic", "Nicaragua")

Unknown <- c("South")

data_all <- data_all %>%
  mutate(native_region = case_when(native_country %in% Asia ~ "Asia",
                                   native_country %in% Europe ~ "Europe",
                                   native_country %in% North_America ~"North America",
                                   native_country %in% Latin_America_Carrib
                                   ~ "Latin America Carrib", native_country %in%
                                     Unknown ~ "Unknown"))
```

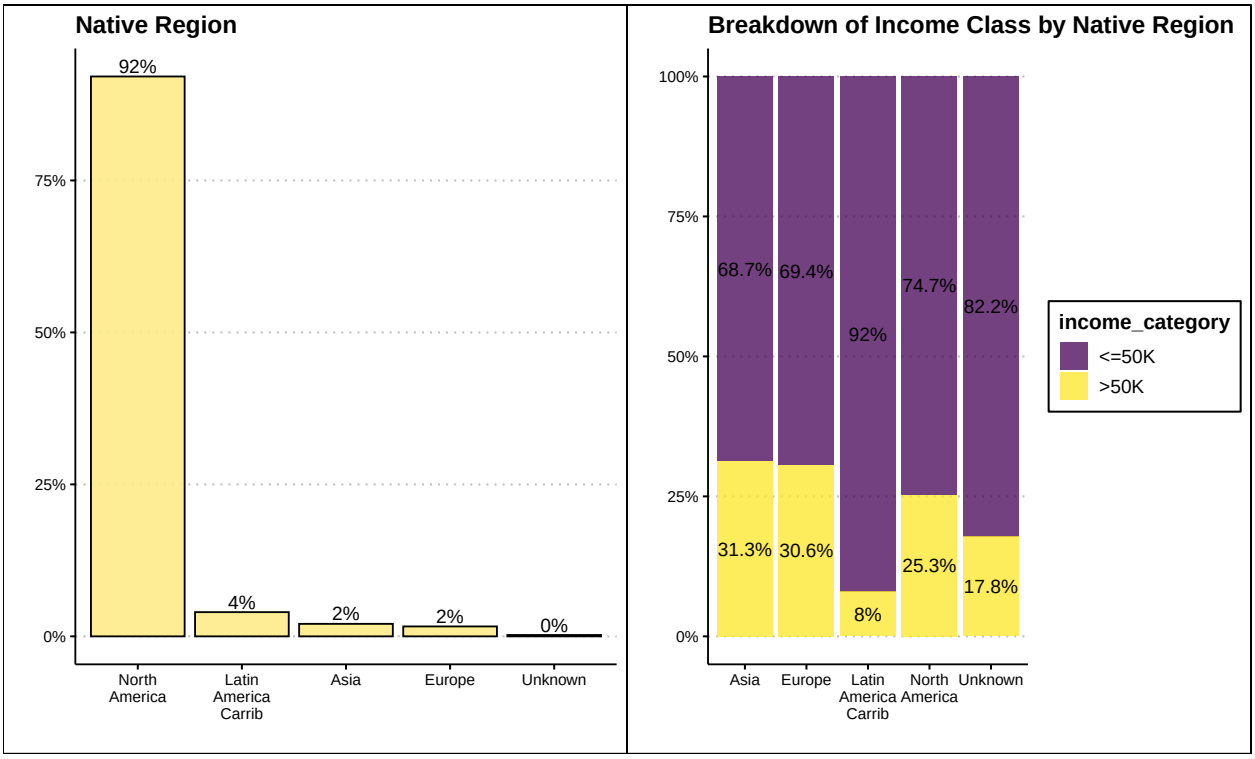
Our **native_region** feature will replace the **native_country** variable. It has the following values:

```
## [1] "Asia"                 "Europe"               "Latin America Carrib"
## [4] "North America"        "Unknown"
```

Here, "Unknown" corresponds to the entries with the ambiguous value of "South".

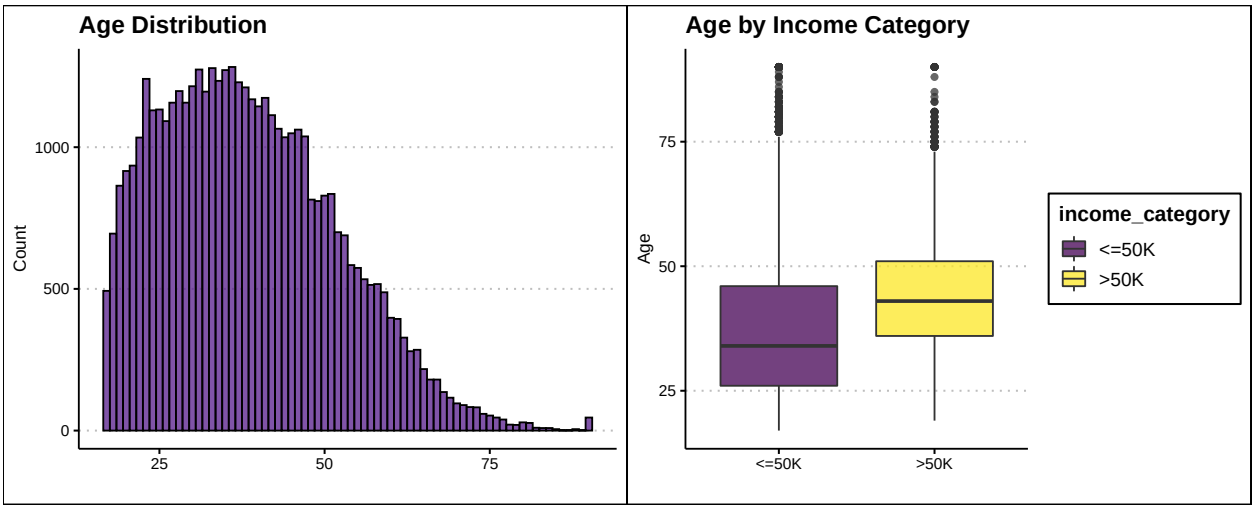
```
## [1] 0
```

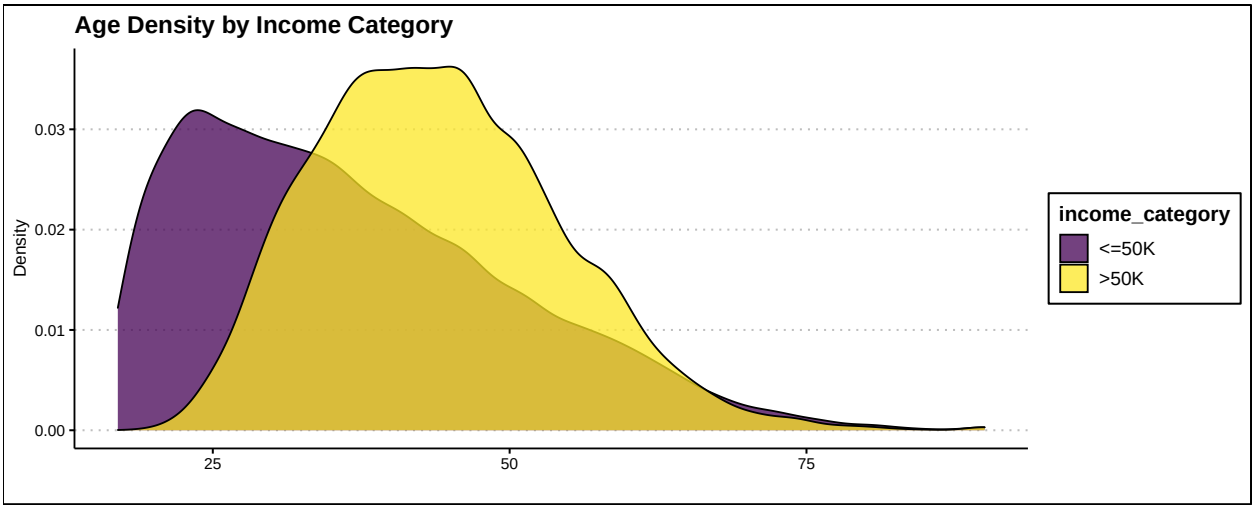
```
## [1] "Asia"                 "Europe"               "Latin America Carrib"
## [4] "North America"        "Unknown"
```



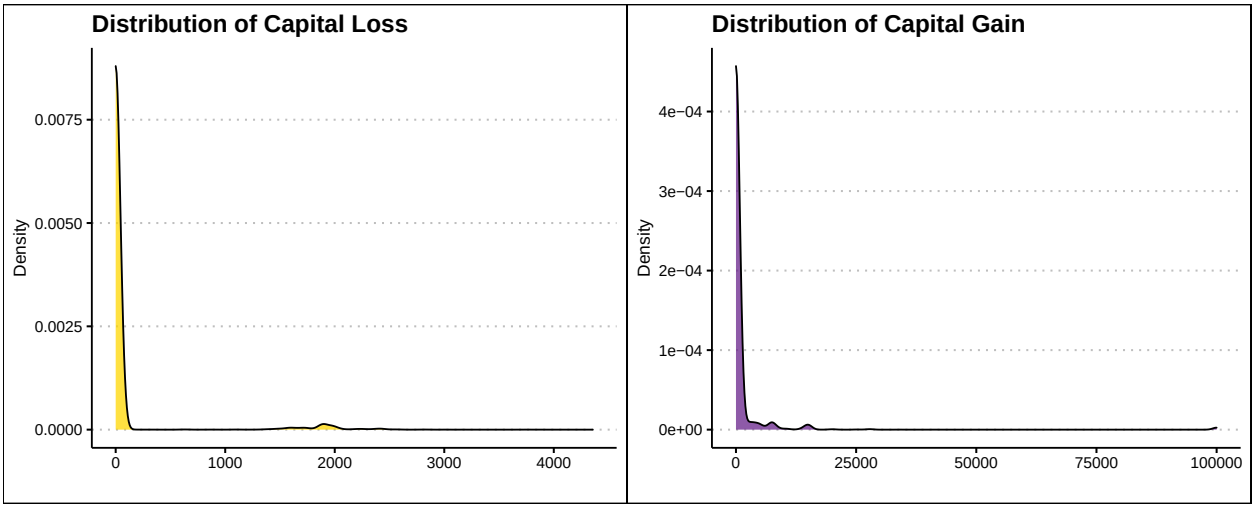
II.2 Continuous Attributes We move on to examine the distribution of values along our continuous predictors. We see that 50% of survey respondents are between the ages of 28 and 47, and and that the age variable has a right-skewed distribution, as there is a higher concentration of individuals below the median age than above it. This helps to explain the low prevalence of individuals earning at least \$50K/year (the median income in the U.S in 1994 was \$52,942). We also see from looking at boxplots and density plots of age along our two income categories that the age distribution of people earning more than \$50K/year is older and and has less variability than that of people earning below that threshold.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   28.00   37.00   38.55   47.00   90.00
```





Finally, we examine the distribution of values contained in the **capital_loss** and the **capital_gain** variables. Density plots of these attributes suggest that the values are heavily skewed towards zero.



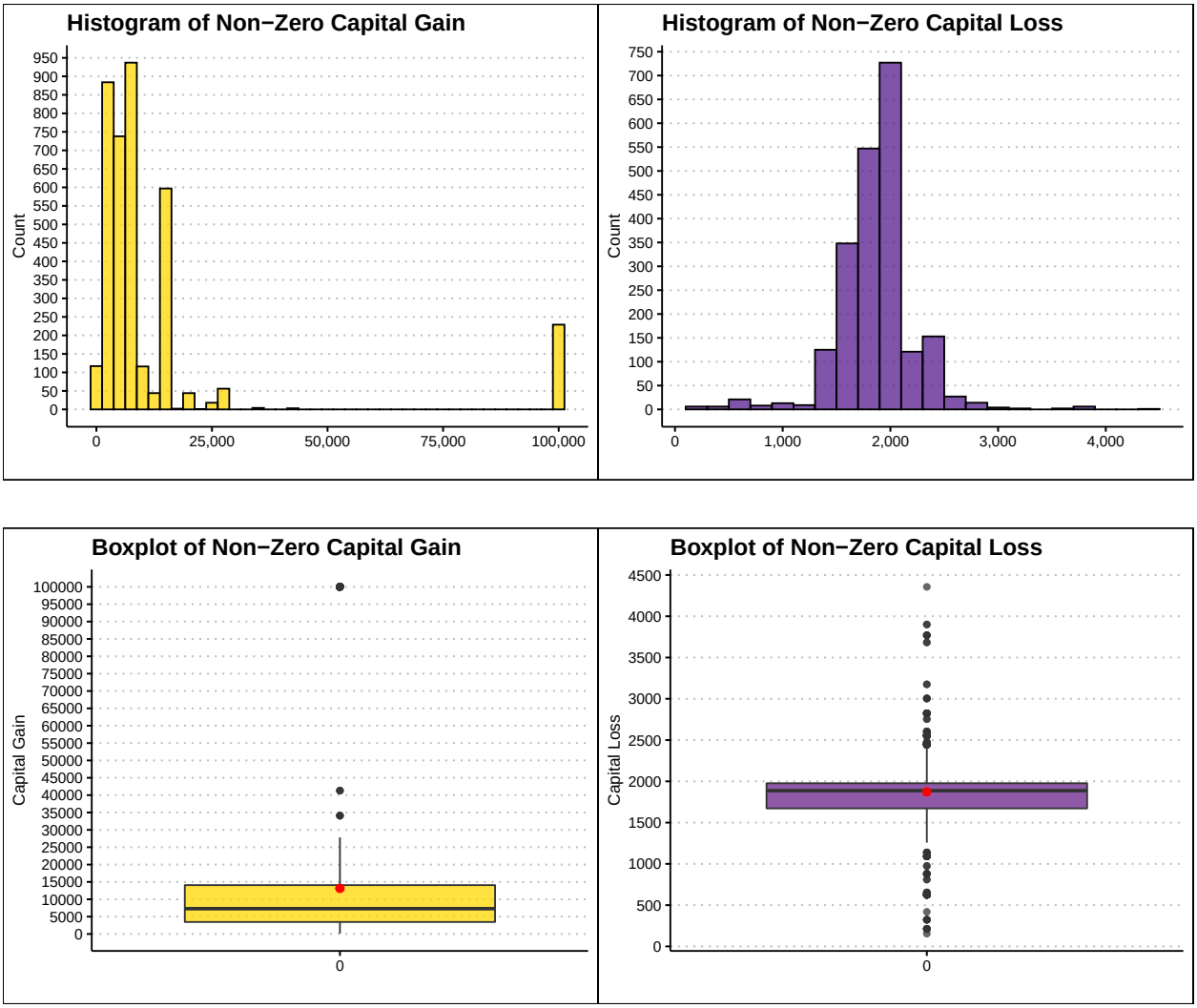
We perform a summary computation to ascertain this and see that there are indeed very few instances of values that differ from zero. 91.6% of the survey respondents have no capital gain, and and 95.3% have no capital loss.

Such a high prevalence of zeros in a continuous variable is likely to distort our predictions. We must therefore create bins so that we can treat both **capital_gain** and **capital_loss** as categorical features in our predictive model. In order to compute these bins, we first explore the distribution of the non-zero values for each variable. We compute the quartiles and get the below results:

	Capital_Gain	Capital_Loss
0%	114	155
25%	3464	1672
50%	7298	1887
75%	14084	1977
100%	99999	4356

We note that the interquartile range for **capital_gain** (10,620) is significantly larger than the IQR for **capital_loss** (305). Then, we create histograms and boxplots of the non-zeros values of **capital_gain** and **capital_loss** to further investigate the distribution of these attributes before creating cutoff values for our

bins.



We see that the distribution of non-zero values of **capital_gain** is right-skewed, with most pepple having a capital gain between $2,500 and $27,500 - the most common value being $7,500 - and that there is a handful of outliers with a capital gain of $100,000. The distribution for **capital_loss** on the other hand, is more symmetric, with most values falling between $1,400 and $2,400 - the most common value being $2,000 - and a large number of outliers above and below this window.

We then create the **cap_gain** variable which has the following levels for the corresponding ranges of **capital_gain**:

- Values equal to or below the first quartile of non-zero values ($3,464): Low
- Values between the first and third quartile of non-zero values ($3,464 to $14,084): Medium
- Values above the third quartile of non-zero values ($14,084): High

We do the same for **capital_loss** and create the **cap_loss** variable which has following levels:

- Values equal to or below the first quartile of non-zero values ($1,672): Low
- Values between the first and third quartile of non-zero values ($1,672 to $1,977): Medium
- Values above the third quartile of non-zero values ($1,977): High

The code looks like this:

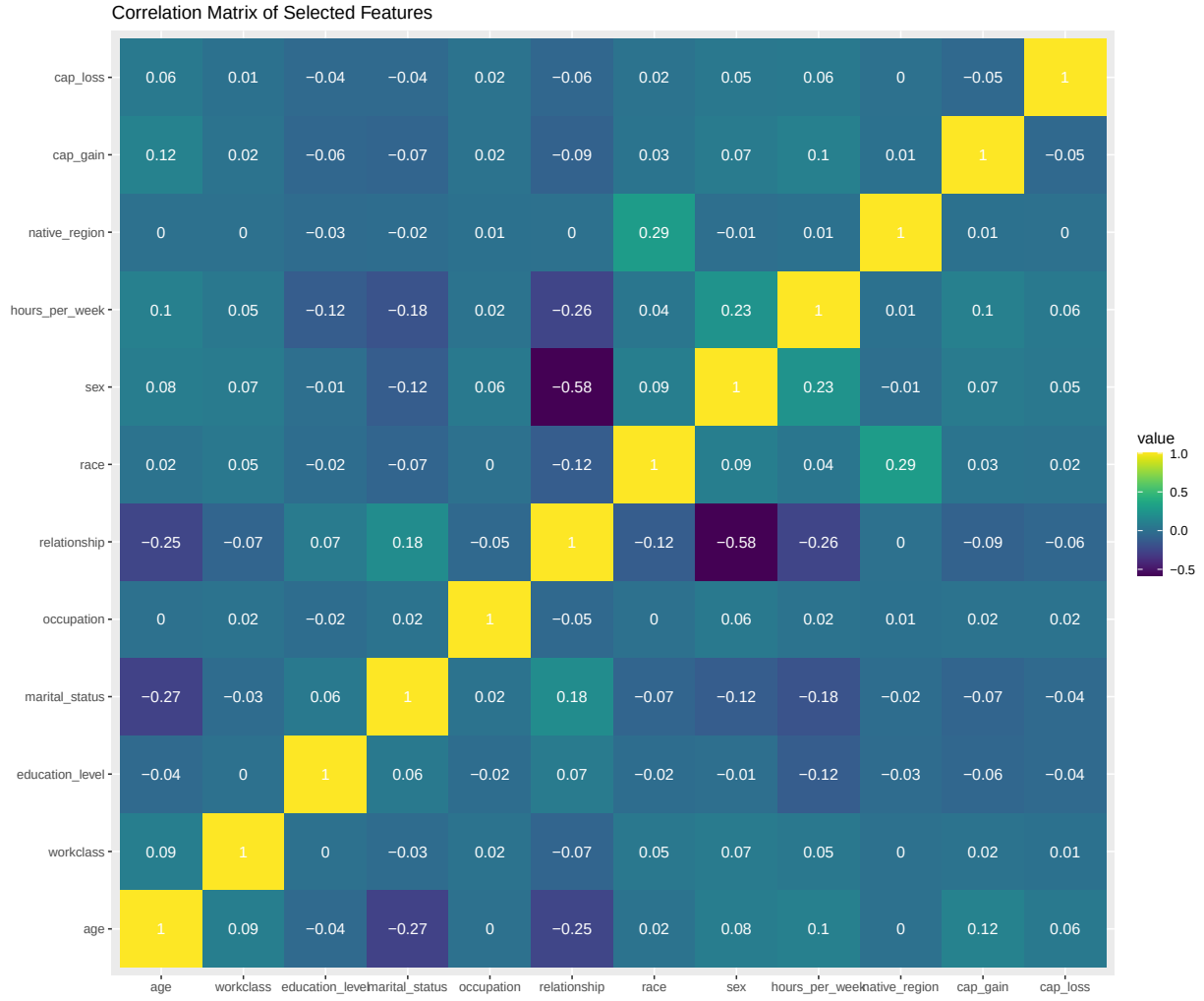
```
data_all <- data_all %>%
  mutate(cap_gain = ifelse(capital_gain <= 3464, "Low",
                           ifelse(capital_gain > 3464 & capital_gain < 14084,
                                  "Medium", "High")))


data_all$cap_gain <- factor(data_all$cap_gain,
                            ordered = TRUE,
                            levels = c("Low", "Medium", "High"))

data_all <- data_all %>%
  mutate(cap_loss = ifelse(capital_loss <= 1672, "Low",
                           ifelse(capital_gain > 1672 & capital_gain < 1977,
                                  "Medium", "High")))

data_all$cap_loss <- factor(data_all$cap_loss,
                            ordered = TRUE,
                            levels = c("Low", "Medium", "High"))
```

After making these modifications to our features, we proceed to conduct a correlation analysis in order to assess how interrelated they are. This is important because in order to build an efficient predictive model, it is best to only include variables that **uniquely** explain some amount of variance in the outcome. Because correlations are most easily computed on numeric variables, we convert all the values of our categorical variables to numeric levels after subsetting our data to exclude the outcome variable (**income_category**), the demographic **weight** variable, and the variables we no longer wish to retain after making our modifications: the continous **capital_gain** and **capital_loss** features, and the **native_country** feature. The correlation matrix does not suggest any strong correlation between any of our features, so we can retain of all the selected features in training a model to predict the income brackets.



C.Paritionning and Training

We begin by subsetting our dataset to retain only the following features: **age**, **workclass**, **education_level**, **marital_status**, **occupation**, **relationship**, **race**, **sex**, **hours_per_week**, **native_region**, **cap_gain**, and **cap_loss**. Finally, we convert the **cap_gain** and **cap_loss** features to numeric for both models because they are ordinal. Finally, we inspect the dataframe structure on last time to ensure that our categorical variables as stored as factors. The code looks like this:

```
## 'data.frame':    45222 obs. of  13 variables:
##  $ age            : num  39 50 38 53 28 37 49 52 ...
##  $ workclass      : Factor w/ 7 levels "Federal-gov",..: 6 5 3 3 3 3 3 5 ...
##  $ education_level: Factor w/ 9 levels "Assoc-acdm","Assoc-voc",..: 3 3 5 9 3 6 9 5 ...
##  $ marital_status : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 ...
##  $ occupation     : Factor w/ 14 levels "Adm-clerical",..: 1 4 6 6 10 4 8 4 ...
##  $ relationship   : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 ...
##  $ race           : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 ...
##  $ sex            : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 ...
##  $ hours_per_week : int  40 13 40 40 40 40 16 45 ...
##  $ income_category: Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 ...
##  $ native_region  : Factor w/ 5 levels "Asia","Europe",..: 4 4 4 4 3 4 3 4 ...
##  $ cap_gain       : Ord.factor w/ 3 levels "Low"<"Medium"<..: 1 1 1 1 1 1 1 1 ...
```

```
##  $ cap_loss       : Ord.factor w/ 3 levels "Low"<"Medium"<..: 1 1 1 1 1 1 1 1 ...
```

We then partition the data into training and testing sets to which we allocate 90% and 10% of the data respectively. The resulting training set has 40,699 observations and the testing set has 4,523 observations. We make sure to set a seed so that each time the program is run, the partition remains the same. Our selected classification models in order of preference are: support vector machines, gradient boosted trees, and logistic regression. As mentionned in the introduction, we will be using balanced accuracy (F1 Score, computed with the F_meas function of the caret package) to optimize the models and assess their performance.

We begin by training a logistic regression model because it is one of the most simple and commonly used supervised machine learning algorithms for classification tasks, is easy to implement, and is often used as a baseline for binary classification. The model performs binary classification by estimating a linear relationship between the features and the outcome variable, and converting the resulting logit values to probabilities between zero and 1 using the sigmoid function. These probabilities are then mapped to one of the two outcome classes using a pre-determined probability threshold of 0.5. While we would have liked to change the threshold value due to the unbalanced nature of our outcome (high prevalence of people earning <=50K), the caret package does not allow for parameter tuning.

We choose a Beta value of 0.9 for our balanced accuracy (F1 score) computation using the F_meas function. Beta is used to assign relative weights to precision and recall, and a value below 1 signifies that we wish to assign more weight to precision over recall. This beta will be fixed for all of our models in order to allow for comparability.

We train the logistic model on our training set using all of our selected features, using the controlTrain() function to specify a reseampling scheme of 20-fold cross validation. This generates 20 non-overlapping, independent random samples from our training set, which are used to generate a cross-validated F1 score once the predictions are applied to our testing set.

We train two logistic models: one with the categorical features as they are, and one with one hot encoding of our nominal categorical features, because we are curious about the claim that logistic regression performs better with continuous, binary, and ordinal features. By one hot encoding we mean creating dunmmy variables for each level of our nominal features such that each of these variables has only two possible values: zero and one. For instance, **marital_status** is split into the following variables: “Divorced” “Married-AF-spouse” “Married-civ-spouse” “Married-spouse-absent” “Never-married” “Separated” “Widowed”, each of which is binary.

We proceed to one-hot encode the rest of the nominal features for the second iteration of our logistic classifier. These are: **workclass**, **education_level**, **marital_status**, **occupation**, **relationship**, **race**, and **native_region**. The code looks like this:

```
ohe_features = c("workclass", "education_level", "marital_status", "occupation",
                 "relationship", "race", "native_region")

#Training Set
dummies_train <- dummyVars(~ workclass + education_level + marital_status +
                             occupation + relationship + race + native_region,
                           data = training)


ohe_dummies_train <- as.data.frame(predict(dummies_train, newdata = training))
training_ohe <- cbind(training[,-c(which(colnames(training) %in% ohe_features))],
                      ohe_dummies_train)

training_ohe$cap_gain <- as.numeric(training_ohe$cap_gain)
training_ohe$cap_loss <- as.numeric(training_ohe$cap_loss)
```

```r
#Testing Set
dummies_test <- dummyVars(~ workclass + education_level + marital_status +
                            occupation + relationship + race + native_region,
                          data = testing)

ohe_dummies_test <- as.data.frame(predict(dummies_test, newdata = testing))
testing_ohe <- cbind(testing[,-c(which(colnames(testing) %in% ohe_features))],
                     ohe_dummies_test)

testing_ohe$cap_gain <- as.numeric(testing_ohe$cap_gain)
testing_ohe$cap_loss <- as.numeric(testing_ohe$cap_loss)


# Define cross validation parameters
train_control <- trainControl(method = "cv", number = 20, p = .8)

# Train the model on training set
logit_fit1 <- train(income_category ~ .,
                    data = training,
                    trControl = train_control,
                    method = "glm",
                    family=binomial())


#Fit the model on testing set
predictions1 <- predict(logit_fit1, testing)

#Compute balanced accuracy
BA_1 <- F_meas(data = predictions1, reference = testing$income_category, beta = .9)

#TRAINING ATTEMPT 2: WITH ONE HOT ENCODED VARIABLES

logit_fit2 <- train(income_category ~ .,
                    data = training_ohe,
                    trControl = train_control,
                    method = "glm",
                    family=binomial())

#Fit the model on testing set
predictions2 <- predict(logit_fit2, testing_ohe)

#Compute balanced accuracy
BA_2 <- F_meas(data = predictions2, reference = testing_ohe$income_category, beta = .9)

#Add results to a table
results <- data.frame(Method = c("Simple Logistic Regression",
                                 "Logistic Regression With One-Hot Encoding"),
                      Balanced_Accuracy = c(BA_1, BA_2))
```

After training each model we generate predictions for the testing set and obtain the below results:

Method	Balanced_Accuracy
Simple Logistic Regression	0.9003631
Logistic Regression With One-Hot Encoding	0.9003646

As the results table shows, one hot encoding on produces a very negligible improvement in our logistic model's performance.

The second model we train is the support vector machines (SVMs) algorithm. This supervised, non-probabilistic classifier has become very popular for its ability to generate very high accuracy levels while using a minimal amount of computational power. It works by partioning the feature space into two or more groups with decision boundaries (which take the shape of n-dimensional hyperplanes, n denoting the number of features used) that separate the data into our outcome classes. The algorithm iterates through several decision bounderies and selects the one that maximizes the margin between the points in each class, using a regularization parameter that strikes a balance between expected loss and margin maximization. What we call support vectors is the collection of points that lie closest to the hyperplane and therefore influence its position and orientation. Support vector machines are especcially effective for highly dimensional data (i.e. data with many features) because they make use of kernels to enlarge the feature space and accommodate a non-linear boundary between the outcome classes. We use the *svm* function from the e107 package to train this model.

We scale the values of our continuous features before implementing the algorithm so that each feature's contribution to the distance calculations is appromixately proportionate. We then train the model using the same cross validation parameters specified for the logistic regression. The code looks like this:

```
#2) SUPPORT VECTOR MACHINES#
  ############################

  #Scale the numeric features in both the training and the teting set

  training_svm <- training
  training_svm$age <- scale(training_svm$age)
  training_svm$hours_per_week <- scale(training_svm$hours_per_week)

  testing_svm <- testing
  testing_svm$age <- scale(testing_svm$age)
  testing_svm$hours_per_week <- scale(testing_svm$hours_per_week)

  str(training_svm)
  str(testing_svm)

  #Train the model using same cross validation parameters as before
  train_control <- trainControl(method = "cv", number = 20, p = .8)

  library(kernlab)

  #grid <- expand.grid(C = seq(0,2,.25))

  svm_linear <- train(income_category ~ .,
                      data = training_svm,
                      trControl = train_control,
                      method = "svmLinear",
                      tuneLength = 10)

  warnings()

  #Generate predictions
  predictions3 = predict(svm_linear, testing_svm)

  #Compute balanced accuracy of linear support vector machine
```

```
BA_3 <- F_meas(data = predictions3, reference = testing_svm$income_category, beta = .9)

#Add results to a table
results <- bind_rows(results,
                     data.frame(Method = "Support Vector Machines with Linear Kernel",
                                Balanced_Accuracy = BA_3))
```

Our results are disapointing, given the amount of time required to train the SVM model. We find that not only does it take significantly longer to train, but it performs slightly worse than logistic regression.

Method	Balanced_Accuracy
Simple Logistic Regression	0.9003631
Logistic Regression With One-Hot Encoding	0.9003646
Support Vector Machines with Linear Kernel	0.8929411

The final model we fit is a non-probabilistic, tree-learning model using the Extreme Gradient Boosting (XGBoost) package. Used for both regression and classification problems, Extreme Gradient Boosting produces predictions by assembling several weak prediction models, and allows for optimization through an arbitrary differentiable loss function. Extreme Gradient Boosted Trees perform binary recursive paritionning of the predictor space, using cross validation (bootstrap aggregation by default) to iterate through a collection of decision trees. Feature selection is random for each tree, and the algortihm selects the combination of paritions that minmize the Gini index (or residual sum of squares for continuous outcomes). It then averages the conditional expectation of these cross-validated decision trees to obtain final predictions for each observation.

What makes it so appealing - in our case, much more appealing than the random forest algorithm - is its ability to perform parallel computations on a single machine with speed, while being compuationally inexpensive (Random Forest caused the RStudio session to crash.) Moroever, Extreme Gradient Boosting is known to perform well when the training data contains a mix of categorical and numeric features, as is the case with our dataset. Lastly, the algorithm possesses a large number of hyperparameters desinged to prevent overfitting.

There are a few preparatory steps to undertake before training. First, we must use the one-hot encoded versions of our training and testing sets because the XGBoost model only accepts numeric inputs. Then, we separate the outcome variable from the feature data, convert both training and testing feature sets into matrices, and reattach the labels to the feature sets in a xgb.DMatrix object, which is specific to the XGBoost framework. The code looks like this:

```
#Convert sex feature (still coded as a factor) to numeric
  training_ohe$sex <- as.numeric(training_ohe$sex)
  testing_ohe$sex <- as.numeric(testing_ohe$sex)

#Separate target variable (income_category) from training and testing sets
# (one hot encoded) and convert to numeric

tr_labels <- training_ohe$income_category
tr_labels <- as.numeric(tr_labels) -1
str(tr_labels)

ts_labels <- testing_ohe$income_category
ts_labels <- as.numeric(ts_labels) -1
str(ts_labels)


training_ohe <- training_ohe %>% select(-income_category)
testing_ohe <- testing_ohe %>% select(-income_category)
```

```
#Convert training and testing sets into matrices then reattach to labels
xgtrain <- matrix(as.numeric(unlist(training_ohe)),nrow=nrow(training_ohe))
xgtest <- matrix(as.numeric(unlist(testing_ohe)),nrow=nrow(testing_ohe))

str(xgtrain)
length(tr_labels)

str(xgtest)
length(ts_labels)

xgtrain <- xgb.DMatrix(data = xgtrain, label = tr_labels)
xgtest <- xgb.DMatrix(data = xgtest, label = ts_labels)
```

Finally, we specify our tuning parameters. XGBoost has a long list of tuning parameters meant to optimize performance and control for overfitting. We select only only the few we deem necessary for the purpose of this simple exercize.

- objective = estimation function, default being linear regression; in our case we select logistic regression for binary classification, which returns predicted probabilities
- nrounds = number of learning iterations (number of trees in the case of classification)
- max.depth = the maximum number of partitions in a tree; the more splits, the more information is captured
- eta = the learning rate, i.e. degree to which the weight of each consecutive tree is reduced
- min_child_rate = minimum Hessian weight required for a new partition; when min is reached, partitions stop
- gamma = the minimum loss reduction required to make an additional partition
- colsample_bytree = the proportion of features supplied to a tree

We store these in a list:

```
params <- list(booster = "gbtree",
               objective = "binary:logistic",
               max_depth = seq(2,15, 1),
               eta = seq(0.05, 0.3, .05),
               min_child_weight = c(1, 3, 5, 7),
               gamma = c(0, 0.4, 0.1),
               colsample_bytree = seq(0.3,0.8,0.1))
```

Next, we use XGBoost's built in cross-validation module which allows us to quickly determine the ideal number of iterations (*nrounds*) without performing a grid search. We specify a maximum of 250 trees, so the module computes a cross validation error (which is an estimate for the test error) for every nround up to 250. We choose a resampling scheme of 10-fold cross validation to make the implemetation less computationally costly.

```
#Find the best number of iterations (trees) using cross validation module
  set.seed(0)
  xgboost_ideal <- xgb.cv(data = xgtrain,
                          nrounds = 250,
                          nfold = 10,
                          params = params,
                          verbose = TRUE,
                          maximize = F,
                          stratified = T,
                          print_every_n = 10,
```

```
                        early_stopping_rounds = 20,
                        eval_metric="error")
```

Here, *stratified* indicates whether the sampling of cross validation folds should be stratified by the value of the outcome label; *print_every_n* specifies the intervals for the printing of results, and *early_stopping_rounds* terminates training when performance, measured by the testing error, does not improve after the specifed number of rounds.

```
#Look at the results (note: CV test error is an optimistic estimate of the actual test error)
  xgboost_ideal
```

```
## ##### xgb.cv 10-folds
##     iter train_error_mean train_error_std test_error_mean test_error_std
##        1        0.2247503    0.0170427387       0.2233714    0.017740616
##        2        0.2157711    0.0137217314       0.2159513    0.014269964
##        3        0.2192329    0.0158009291       0.2180151    0.015289449
##        4        0.2130055    0.0108493980       0.2123145    0.012558632
##        5        0.2149739    0.0097015756       0.2138132    0.010573645
## ---
##      246        0.1501893    0.0006197078       0.1519202    0.003473468
##      247        0.1501811    0.0006347362       0.1517482    0.003471907
##      248        0.1502167    0.0006777153       0.1516744    0.003528497
##      249        0.1502056    0.0006818573       0.1516252    0.003612086
##      250        0.1501568    0.0006454866       0.1516253    0.003536133
## Best iteration:
##  iter train_error_mean train_error_std test_error_mean test_error_std
##   231         0.150749    0.0004684346       0.1516006    0.003570704
```

The results indicate that the ideal number of training iterations is 240, so we train the model using an *nrounds* of 231.

```
#Train model using ideal number of iterations found above

  xgboost_fit <- xgb.train(data = xgtrain,
                           nrounds = 231,
                           nfold = 10,
                           params = params,
                           verbose = TRUE,
                           maximize = F,
                           print_every_n = 10,
                           early_stopping_rounds = 20,
                           watchlist = list(val=xgtest,train=xgtrain),
                           eval_metric="error")
```

Finally, we generate the predictions for the testing set and convert the resulting probabilites into an outcome class, and compute the balanced accuracy which we add to our results table.The code looks like this:

```
 #Generate predictions, and convert to binary by using threshold of .5
  predictions4 = predict(xgboost_fit, xgtest)
  predictions4 <- ifelse (predictions4 > 0.5,1,0)

  #Compute balanced accuracy of extreme gradient boosting
  ts_labels <- as.factor(ts_labels)
  predictions4 <- as.factor(predictions4)
  BA_4 <- F_meas(data = predictions4, reference = ts_labels, beta = .9)
```

```r
#Add results to a table
results <- bind_rows(results,
                     data.frame(Method = "Gradient Boosted Tree",
                                Balanced_Accuracy =  BA_4))
```

Looking at at the results table, we see that the F-1 score for the XBBoost algorithm is slightly lower than that achieved with logistic regression, and marginally higher than what was achieved with support vector machines. Overall, our results defy our expectations (we expected much more variation in our F-1 scores) and higlight some key lessons about machine learning model selection. First, the fact that performance is similarly high across all three models is a testament to the success of discriminative models (lazy learners) with classification tasks. Neither logistic regression, support vector machines, nor gradient boosted decision trees attempt to estimate the joint probability distribution between every single feature and the outcome, but rather simply optimize the conditional probabilities of our outcome. Second, the fact that logistic regression slightly outperforms the other models illustrates how probabilisitc classifiers are generally more robust to error propagation.

Method	Balanced_Accuracy
Simple Logistic Regression	0.9003631
Logistic Regression With One-Hot Encoding	0.9003646
Support Vector Machines with Linear Kernel	0.8929411
Gradient Boosted Tree	0.8996816

A.Conclusion

In this capstone project, we conducted a machine learning classification task on the UCI's Machine Learning Repository's 1996 *Adult* dataset, with income level (>$50 or >=$50k) as our variable of interest. After rejoining the already partitionned training and testing sets (for a total 45,222 observations), we carried out exploratory analysis which allowed us to perform important feature extraction tasks. This included dropping a redundant attribute (**education_num**), collapsing two of the categorical features (**education_level** and **native_country**), and discretizing two of our continuous features (**capital_gain** and **capital_loss**). We trained three discriminative classifiers (logistic regression, support vector machines, and gradient boosted decision trees) which we assessed using balanced accuracy scores (slightly discounting the weight of recall) due to the imbalanced nature of our outcome classes. We found that while all three models performed well on our validation sets, logistic regression generated the best results. This probabilistic classifier was the easiest to train, had the fastest implementation, and achieved the highest balanced accuracy score (0.9). Given more time, possible improvements on this classification exercise would include: 1. oversampling, undersampling, and/or generating synthetic samples to correct for the class imbalance; 2. partitionning with several different seeds to resample the training data, and assessing the bias/variance tradeoff across each of the classifiers.