Problem Set 3

Programming and Data Management

Question 1 (20 Points)

Learning Objectives used in this question

- Apply fundamental programming principles such as variables, loops, error handling, functions, and control flow to solve programming problems
- Solve interesting and challenging data problems using the R language

Key skills from the course used in this question

- Writing functions
- Variable assignment and recoding
- Data management with dplyr
- Loops and Conditionals
- Data Structures

First, make sure you load in the tidyverse package

Write a function that will take a dataframe/tibble as an argument and a factor variable as another argument and will create dummy variable for that factor and append them to the end of the dataframe. Call the function `create_dummies()`. You cannot use the 'model.matrix()' function or any packages that might actually do this, you must do it from scratch. Make sure you deal with cases where someone doesn't put in a tibble/dataframe as first argument and a factor as second argument. also include a third argument where the user can decide whether or not they want to drop the original variable and just keep the dummies or keep all three. Make this third argument have a default value of TRUE. The tibble you return should only include the original variable (if third argument is TRUE) and the dummies. not any other variables in the data.

If you are not aware, a dummy variable is a variable that can either be true (1) or can be false (0). It cannot take on any other value. I believe we have already gone over this but I want to reiterate it here.

When you have written your function you can use the code below to test that your function works correctly. If you don't get errors and the output looks like it should then you are good to go!

```
set.seed(1234)
test <- tibble(
  sex = sample(c('Male', 'Female', NA), 20, T),
  voted = sample(c('Yes', 'No', NA), 20, T)
)

test <- test %>% mutate(
  sex = parse_factor(
    sex, levels = c('Male', 'Female'), include_na = F
  ),
  voted = parse_factor(voted, levels = c('Yes', 'No'), include_na = F)
)

create_dummies(test, 'sex', T)
create_dummies(test, 'sex', F)
create_dummies(test, 'voted', T)
create_dummies(test, 'voted', F)
```

As an example, if you data looks like this

sex
Female
Female
Male
NA
Male

Then after you run the function like this `create_dummies(test, 'sex', .keep = TRUE)` your data should look like this:

sex	Male_1	Female_2
Female	0	1
Female	0	1
Male	1	0
NA	NA	NA
Male	1	0

Question 2 (10 points)

Learning Objectives used in this question

- Apply fundamental programming principles such as variables, loops, error handling, functions, and control flow to solve programming problems
- Solve interesting and challenging data problems using the R language

Key skills from the course used in this question

- Writing functions
- Variable assignment and recoding
- Loops and Conditionals
- Functionals

The `create_dummies()` function is great it will take a single factor variable and will return the dummy variables we want. But what if we have 20 or 30 or 100 variables we want to create dummies for. Then we have to run this function 20, 30, or 100 times. That seems really inefficient. Write a new function called `create_multiple_dummies()` that will take a dataframe/tibble as the first argument, and the second argument should be a vector of variable names that we want this to operate on. This function should run the `create_dummies()` function we just wrote on each variable in the vector of variables names. Finally it should bind all the results together and return the finished tibble. Use the tibble I created below to do this, you should use the functions from the `purrr` package to make this happen.

Run this code below as a test, if you wrote the function correctly you should get the correct output.

```
set.seed(1234)
test2 <- tibble(
  sex = parse_factor(
    sample(c('Male', 'Female'), 15, T), levels = c('Male', 'Female')
  ),
  age = parse_factor(
    sample(c('18-30', '30-50', '50+'), 15, T),
    levels = c('18-30', '30-50', '50+')
  ),
  income = parse_factor(
    sample(c(' < 25K', '25K < 50K', '50K < 100K', '100K+'), 15, T),
    levels = c(' < 25K', '25K < 50K', '50K < 100K', '100K+')
  )
)

create_multiple_dummies(test2, c('sex', 'age', 'income'), .keep = T)
create_multiple_dummies(test2, c('sex', 'age', 'income'), .keep = F)
```

For example the data looks like this

sex	age	income
Female	30-50	25K < 50K
Female	30-50	50K < 100K
Female	50+	100K+
Female	30-50	50K < 100K
Male	30-50	100K+

So when we run the function `create_multiple_dummies(test2, c('sex','age','income'), .keep = T)` it should look like this

sex	Male_1	Female_2	age	18-30_1	30-50_2	50+_3	income	< 25K_1	25K < 50K_2	50K < 100K_3	100K+_4
Female	0	1	30-50	0	1	0	25K < 50K	0	1	0	0
Female	0	1	30-50	0	1	0	50K < 100K	0	0	1	0
Female	0	1	50+	0	0	1	100K+	0	0	0	1
Female	0	1	30-50	0	1	0	50K < 100K	0	0	1	0
Male	1	0	30-50	0	1	0	100K+	0	0	0	1

Question 3 (15 Points)

Learning Objectives used in this question

- Apply fundamental programming principles such as variables, loops, error handling, functions, and control flow to solve programming problems
- Solve interesting and challenging data problems using the R language

Key skills from the course used in this question

- Writing functions
- Variable assignment and recoding
- Loops and Conditionals
- Functionals

Finally we may want our users to be able to specify that they want to keep some of the original variables and discard others, so re-write the `create_multiple_dummies()` function and call it `create_multiple_dummies2`. It will add the same arguments as the original function but it will also take a fourth argument called `.keeps` which is a vector of TRUE/FALSE/NA that correspond to each variable specified and should have a default value of NULL. For example if we want to keep 'age' but discard 'income' and 'sex' then we should be able to call the function like this `create_multiple_dummies2(df, c('sex','age','income'), .keeps = c(F, T, F))`

Use this code as test code. If you write your function correctly, then this code should give to correct output. Make sure to use the same `test2` data that we created before

```
create_multiple_dummies2(
  test2, c('sex', 'age', 'income'), .keep = T, .keeps = NULL
)
create_multiple_dummies2(
  test2, c('sex', 'age', 'income'), .keep = F, .keeps = NULL
)
create_multiple_dummies2(
  test2, c('sex', 'age', 'income'), .keeps = c(T, T, F)
)
```

For example, the `test2` data looks like this

sex	age	income
Female	30-50	25K < 50K
Female	30-50	50K < 100K
Female	50+	100K+
Female	30-50	50K < 100K
Male	30-50	100K+

Then the a call to `create_multiple_dummies2(test2, c('sex', 'age', 'income'), .keeps = c(T, T, F))` should look like this

sex	Male_1	Female_2	age	18-30_1	30-50_2	50+_3	< 25K_1	25K < 50K_2	50K < 100K_3	100K+_4
Female	0	1	30-50	0	1	0	0	1	0	0
Female	0	1	30-50	0	1	0	0	0	1	0
Female	0	1	50+	0	0	1	0	0	0	1
Female	0	1	30-50	0	1	0	0	0	1	0
Male	1	0	30-50	0	1	0	0	0	0	1

Question 4 (55 Points)

Learning Objectives used in this question

- Solve interesting and challenging data problems using the R language
- Manipulate data in a variety of ways such as rectangle data, recode variables, and creating new variables
- Load data into R, organize it, and prepare it for data analysis

Key skills from the course used in this question

- Data Management
  - Reading and writing data in R
  - Selecting and manipulating rows and columns
  - Grouping and summarizing data
  - Reordering data and re-coding variables
  - Working with strings and variable names
  - Manipulating different data types

Install and load the `nycflights13` packages which contains datasets on flights in the US and load it in. Just use the code below for this. Also store the flights in an object called `df`

```
install.packages('nycflights13')
library(nycflights13)
df <- flights
```

FOR THE REST OF THE QUESTIONS, DO NOT SAVE THE DATASET BACK INTO ITSELF OR ANOTHER OBJECT, JUST WRITE THE CODE SO IT PRINTS OUT THE RESULT. IF YOU DO NEED TO SAVE IT INTO SOMETHING FOR WHATEVER REASON SAVE IT INTO AN OBJECT CALLED `temp`

FOR EXAMPLE DO THIS

```
df %>% arrange(year)
```

NOT THIS

```
df <- df %>% arrange(year)
```

PART A (3 points)

Only select cases where flights took place in March. The month variable is an integer where 1 = January, 2 = February, etc. Do not resave this into itself or a new variable, just run it

PART B (3 points)

Select only cases where the origin airport is either 'JFK' or 'LGA' and the flight was during the winter months (November though February)

PART C (2 points)

The `year` variable is all 2013 so we don't need it, remove it from the data

PART D (2 points)

Reorder the variables so that any with an underscore (`_`) in the name is first followed by anything else

PART E (2 points)

Select only variables from `dep_delay` through `minute`

PART F (3 points)

Select only the variables that are integers

PART G (2 points)

Select only the variables that start with 'arr' or 'dep'

PART H (3 points)

Create a variable called `total_delay` that is the arrival delay plus the departure delay and then order the variables to be departure delay followed by arrival delay, followed by total delay followed by all other variables

PART I (3 points)

Make all the character variables into factors

PART J (7 points)

Create a variable called `distance_bins` with less than 1000 is short, from 1000-2000 inclusive is 'medium' and greater than 2000 is 'long'. Use either if_else() or case_when() to do this. Also once that is done, make it an ordered factor with orders being short, medium, long

PART K (2 points)

Count the number of flights that took off for every day in the dataset

PART L (5 points)

What was the average departure delay for each origin airport for each month

PART M (5 points)

Create a variable from the months for winter (Dec-Feb), spring (mar-may), summer (jun-aug) and fall (sept-Nov). Then figure out the average and median departure delay for each airport for each season of the year. Rename the column of the resulting tibble appropriately

PART N (5 points)

It looks like in the fall flights are more likely to arrive early (they have a negative delay). Let's inspect this further. Create a variable called `was_delayed` that is a 1 if there was a delay with both the arrival and the departure and 0 otherwise. If both arrival and departure are NA, then this variable should also be NA, but if one of the two is valid then we should have a valid integer. save this data in a variable called 'lpm_data'. Also include the `season` variable we created before and make the origin variable a factor

PART O (7 points)

Now run a linear probability model where the dependent variable is our new `was_delayed` dummy variable and the independent variables are the season, the hour, the origin, the air_time, and the distance. First select out these variables and remove all NA values and save it back into `lpm_data`, and only include the airlines (carrier) with the top three number of flights. Also make the origin reference group 'JFK' and the season reference group fall. you can do this with the 'relevel()' function

```
## # A tibble: 16 x 2
##    carrier     n
##    <chr>   <int>
## 1  UA    58665
## 2  B6    54635
## 3  EV    54173
## 4  DL    48110
## 5  AA    32729
## 6  MQ    26397
## 7  US    20536
## 8  9E    18460
## 9  WN    12275
## 10 VX     5162
## 11 FL     3260
## 12 AS      714
## 13 F9      685
## 14 YV      601
## 15 HA      342
## 16 OO       32
```

Install the estimatr package and load it with library

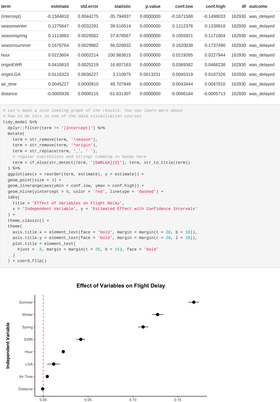
```
install.packages('estimatr')
library(estimatr)
```

Now lets look at a model and see the results

```
library(estimatr)

# run a linear probability model with robust standard errors. You didn't need
# to know this for the course but I am include it here to show you some of what
# you may found data cleaning skills can be for you in the end
model <- lm_robust(
  was_delayed ~ season + hour + origin + air_time + distance,
  data = lpm_data, se_type = 'HC1'
)

# let's take a look at the model
tidy_model <- model %>% broom::tidy() %>% as_tibble()

knitr::kable(tidy_model)
```

term	estimate	std.error	statistic	p.value	conf.low	conf.high	df	outcome
(Intercept)	-0.1584810	0.0064275	-24.794037	0.0000000	-0.1671568	-0.1498053	162930	was_delayed
seasonwinter	0.1275647	0.0032201	39.610514	0.0000000	0.1212376	0.1338918	162930	was_delayed
seasonspring	0.1113863	0.0029562	37.676567	0.0000000	0.1055921	0.1171806	162930	was_delayed
seasonsummer	0.1078764	0.0029562	36.028332	0.0000000	0.1020938	0.1136590	162930	was_delayed
hour	0.0223604	0.0002214	100.963615	0.0000000	0.0219265	0.0227944	162930	was_delayed
originEWR	0.0418010	0.0025219	16.607163	0.0000000	0.0368382	0.0467638	162930	was_delayed
originLGA	0.0116323	0.0026227	3.210875	0.0013231	0.0045319	0.0187326	162930	was_delayed
air_time	0.0045227	0.0000910	49.707948	0.0000000	0.0043444	0.0047010	162930	was_delayed
distance	-0.0009309	0.0000115	-51.631307	0.0000000	-0.0009164	-0.0009713	162930	was_delayed

```
# let's make a nice looking graph of the results. You can learn more about
# how to do this in one of the data visualization courses
tidy_model %>%
  dplyr::filter(term != '(Intercept)') %>%
  mutate(
    term = str_remove(term, 'season'),
    term = str_remove(term, 'origin'),
    term = str_replace(term, '_', ' '),
    # regular expressions and stringr coming in handy here
    term = if_else(str_detect(term, '[EWRLGA]{3}'), term, str_to_title(term))
  ) %>%
  ggplot(aes(x = reorder(term, estimate), y = estimate)) +
  geom_pointrange(size = 2) +
  geom_linerange(aes(ymin = conf.low, ymax = conf.high)) +
  geom_hline(yintercept = 0, color = "red", linetype = 'dashed') +
  labs(
    title = 'Effect of Variables on Flight Delay',
    x = 'Independent Variable', y = 'Estimated Effect with Confidence Intervals'
  ) +
  theme_classic() +
  theme(
    axis.title.x = element_text(face = 'bold', margin = margin(t = 20, b = 10)),
    axis.title.y = element_text(face = 'bold', margin = margin(r = 20, l = 10)),
    plot.title = element_text(
      hjust = .5, margin = margin(t = 25, b = 15), face = 'bold'
    )
  ) + coord_flip()
```




```
# Looks like flights are delayed much more in summer, winter, and spring
# compared to the fall and that Newark has a much higher rate of delays compared
# to the JFK airport.
```