Recipe User-Interaction Analysis

Name(s): Jessica Hung, Samantha Lin

Website Link: https://jesshung323.github.io/recipe-user-interaction-analysis/

Code

In [1]:
```
import pandas as pd
import numpy as np
import os

import plotly.express as px
pd.options.plotting.backend = 'plotly'
```

File Uploads

In [2]:
```
recipes =pd.read_csv('food_data/RAW_recipes.csv')
```

In [3]:
```
interactions = pd.read_csv('food_data/RAW_interactions.csv')
```

In [4]:
```
recipes_copy = recipes.copy()
```

In [5]:
```
interactions['recipe_id'].dtype
```

Out[5]:
```
dtype('int64')
```

Introduction

When people search for recipes online and skim through the webpage for each one, it is natural for one to pay extra attention on the amount of time needed to complete the dish and how well this recipe has been rated by previous users because afterall, you want to make something that tastes good and is not a waste of your time.

Our project tries to analyze the relationship between the time (in minutes) a recipe takes and its corresponding rating, and answer the question:

What is the relationship between the cooking time and average rating of recipes?

Perhaps a longer recipe will receive lower rating because as one completes the recipe that takes a super long time, and realizes the outcome is not as “wow” as expected, then the user may be more likely to leave a low rating. While at the same time, a recipe that takes less time may be less sophisticated in developing flavors and presentation, hence the rating for these recipes may also be lower. Which one is more intuitive for you?

__Dataset Brief Overview__

Number of Rows: There are two datasets required for this project:

1. RAW_recipes.csv has 83782 rows
2. RAW_interactions.csv has 731927 rows
3. Relevant Columns: 'minutes', 'rating', 'n_steps', 'id', 'review', 'recipe_id'

Column Descriptions:

1. 'minutes': cooking time, in minutes, the recipe requires; type int.
2. 'rating': rating given to the recipe by a user; type int.
3. 'n_steps': number of steps a recipe has; type int.
4. 'id': recipe’s ID in RAW_recipes.csv; type int.
5. 'review': a review left by the user of the recipe; type object.
6. 'recipe_id': recipe’s ID in RAW_interactions.csv; type int.

Cleaning and EDA

In [6]:
```
# Merges the two datasets together by 'id' and 'recipe_id'.
# Then creates a new groupby dataframe with average rating,
# and merge with new_merged
new_merged = recipes.merge(interactions, left_on = "id",
                           right_on = 'recipe_id',
                           how = 'left')
new_merged['rating'] = new_merged['rating'].replace(0, np.NaN)
recipe_average_rating = new_merged.groupby('id')['rating'].mean()
new_merged = new_merged.merge((recipe_average_rating).to_frame(),
                              left_on = 'id',
                              right_index = True,
                              how = 'left',
```

```
                                  suffixes=('_original', '_average'))
merged_copy = new_merged.copy()
```

In [7]:
```
# Strips '[' and ']' from 'nutrition' column because they
# are string values.
new_merged['nutrition'] = new_merged['nutrition'].apply(
    lambda x: x.strip('[').strip(']')
)
```

In [8]:
```
# Creates new columns by splitting the string in 'nutrition'
# column
new_merged[['calories',
            'total fat',
            'sugar',
            'sodium',
            'protein',
            'saturated fat',
            'carbohydrate']] = new_merged.nutrition.str.split(pat= ', ',
                                                              expand=True)
new_merged[['calories',
            'total fat',
            'sugar',
            'sodium',
            'protein',
            'saturated fat',
            'carbohydrate']] = new_merged[['calories',
            'total fat',
            'sugar',
            'sodium',
            'protein',
            'saturated fat',
            'carbohydrate']].applymap(lambda x: float(x))
```

In [9]:
```
new_merged.columns
```

Out[9]:
```
Index(['name', 'id', 'minutes', 'contributor_id', 'submitted', 'tags',
       'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
       'n_ingredients', 'user_id', 'recipe_id', 'date', 'rating_original',
       'review', 'rating_average', 'calories', 'total fat', 'sugar', 'sodiu
m',
       'protein', 'saturated fat', 'carbohydrate'],
      dtype='object')
```

In [10]:
```
# Shows the current histogram distribution of minutes;
# there are extreme outliers.
np.histogram(new_merged['minutes'])
```

Out[10]:
```
(array([234424,      1,      2,      0,      0,      0,      0,      0,
             0,      2]),
 array([      0.,  105120.,  210240.,  315360.,  420480.,  525600.,
         630720.,  735840.,  840960.,  946080., 1051200.]))
```

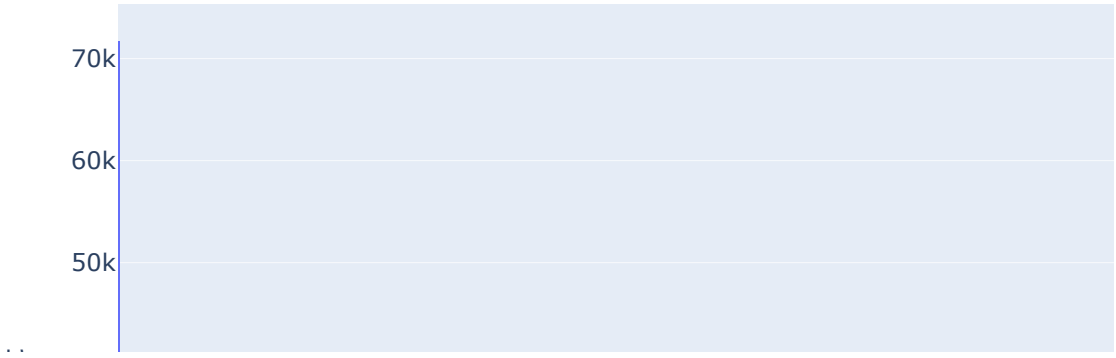
In [11]:
```
# Replaces those 5 extreme outliers' values with NaN
new_merged['minutes'] = np.where(new_merged['minutes'] > 100000,
                                 np.nan, new_merged['minutes'])
```

In [12]:
```
# New histogram distribution; there are still
# quite a few outliers in this one.
np.histogram(new_merged[~new_merged['minutes'].isna()]['minutes'])
```

Out[12]:
```
(array([234198,     63,    128,      8,     15,      4,      1,      3,
            1,      3]),
 array([    0. ,  8641.5, 17283. , 25924.5, 34566. , 43207.5, 51849. ,
        60490.5, 69132. , 77773.5, 86415. ]))
```

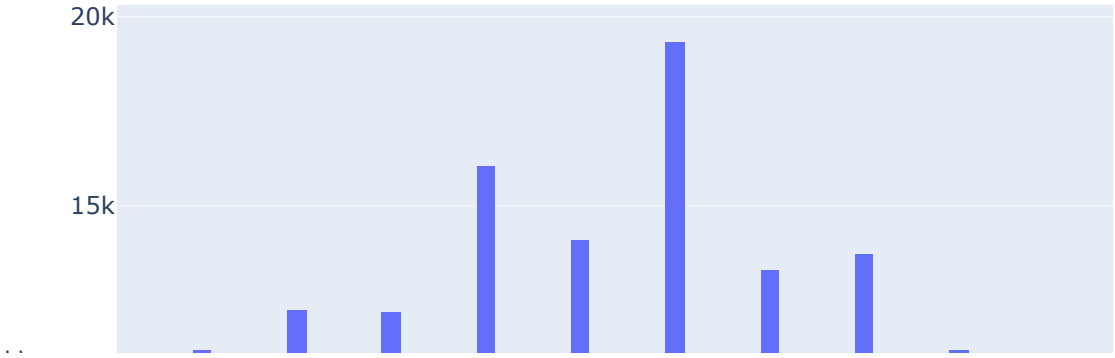
Univariate Analysis Plot 1 - Distribution of Minutes

In [13]:
```
# Plots the distribution of 'minutes' column without
# the five extreme outliers
fig = px.histogram(new_merged[~new_merged['minutes'].isna()]
                   ['minutes'], x="minutes")
fig.show()
```



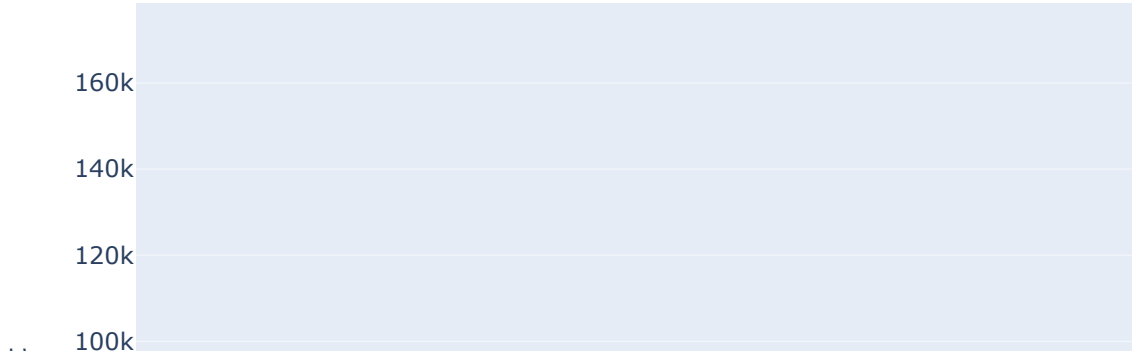
In [14]:
```
# Uses the interquartile range to determine the
# outliers in a copy of new_merged
q75, q25 = np.percentile(merged_copy['minutes'], [75 ,25])
iqr = q75 - q25
outliers_upper_threshold = q75 + 1.5 * iqr
```

```
copy = merged_copy[(merged_copy['minutes'] > 0) &
    (merged_copy['minutes'] <= outliers_upper_threshold)]
fig = px.histogram(copy, x="minutes")
fig.show()
```



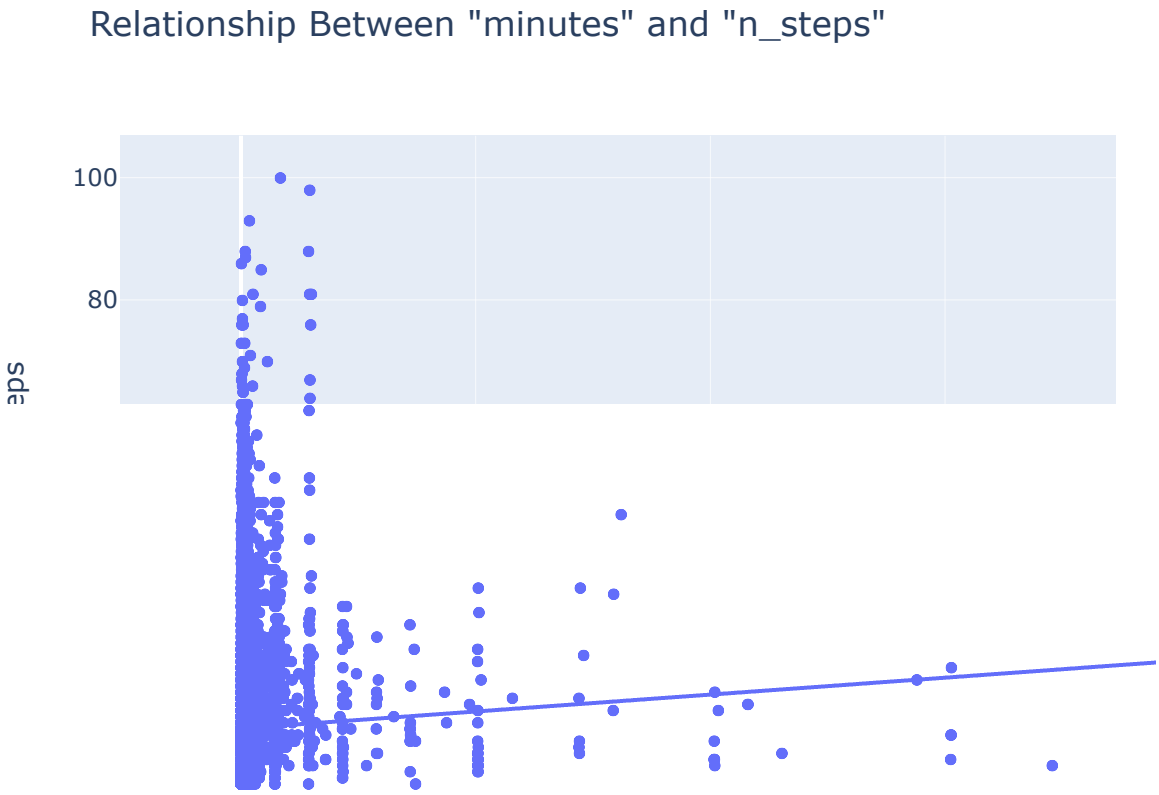
Univariate Analysis Plot 2 - Distribution of Rating

In [15]:
```
# Plots the histogram distribution of
# original rating of the recipes
fig = px.histogram(new_merged, x="rating_original")
fig.show()
```



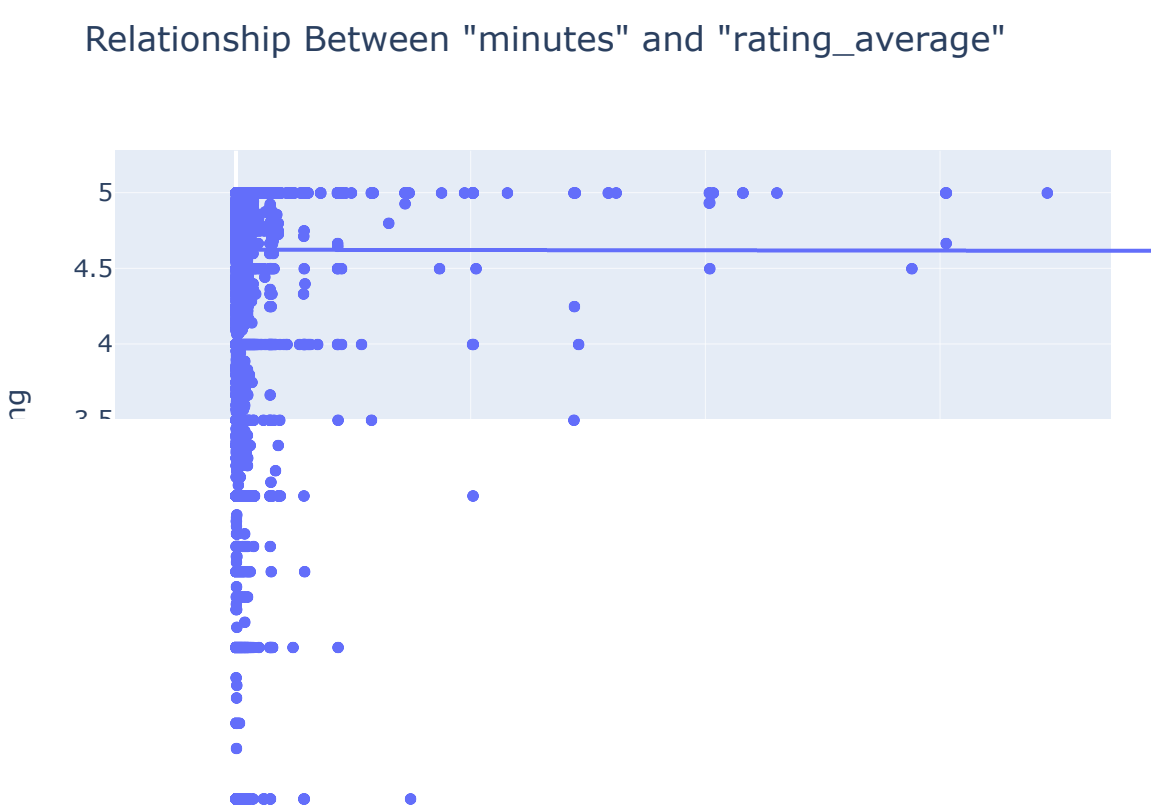
Bivariate Analysis Plot 1 - Scatterplot with Minutes and Number of Steps

In [16]:
```
# Since some recipes are used by more than one user, we grouped the recipes
#by recipe_id first and get the n_steps, minutes, rating_average columns
# Then plot the scatterplot with minutes on x-axis and n_steps on y-axis
same_recipe = new_merged.groupby('recipe_id')[
    ['n_steps', 'minutes', 'rating_average']
].mean()
fig = px.scatter(x=same_recipe['minutes'],
                 y=same_recipe['n_steps'],
                 trendline="ols",
                 labels = {'x': 'Minutes', 'y': 'Number of Steps'},
                 title = 'Relationship Between "minutes" and "n_steps"')
fig.show()
```



Bivariate Analysis Plot 2 - Scatterplot with Minutes and Rating

In [17]:
```
# We use the same grouped dataframe as before to plot the
# scatterplot with minutes on x-axis and rating_average on y-axis
fig = px.scatter(x=same_recipe['minutes'],
                 y=same_recipe['rating_average'],
                 trendline="ols",
                 labels = {'x': 'Minutes', 'y': 'Average Rating'},
                 title = 'Relationship Between "minutes" and "rating_average
fig.show()
```



In [18]:
```
# Collect the 'id' and 'submitted' columns and transform 'submitted'
# to datetime. Then group by the year of each recipe's submission.
copy_date = new_merged[['id', 'submitted']].copy()
copy_date['submitted'] = pd.to_datetime(copy_date['submitted'])
copy_date['submitted year'] = copy_date['submitted'].apply(lambda x: x.year)
copy_date = copy_date.groupby('submitted year')[['submitted']].count()
```

In [19]:
```
copy_date
```

Out[19]:

submitted year	submitted
2008	97468
2009	63015
2010	29897
2011	17181
2012	12700
2013	9870
2014	2534
2015	613
2016	319
2017	521
2018	311

Assessment of Missingness

State whether you believe there is a column in your dataset that is NMAR. Explain your reasoning and any additional data you might want to obtain that could explain the missingness (thereby making it MAR). Make sure to explicitly use the term "NMAR."

We believe that the review column in the interactions dataframe can be NMAR because perhaps the user never really completed the recipe they started on, hence the review would be missing in this case. That being said, if we are able to obtain an additional column indicating whether the user truly finished the recipe, we may be able to determine that the missingness of review is MAR for the reason stated above.

Missingness of 'Minutes' on 'Ratings'

In [20]:
```
# added column of true and false depending on if minutes is missing
merged_mcar = new_merged.copy()
merged_mcar['minutes_missing'] = merged_mcar['minutes'].isna()
merged_mcar
```

Out[20]:

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_step
0	1 brownies in the world best ever	333281	40.0	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	138.4, 10.0, 50.0, 3.0, 3.0, 19.0, 6.0	1
1	1 in canada chocolate chip cookies	453467	45.0	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	595.1, 46.0, 211.0, 22.0, 13.0, 51.0, 26.0	1
2	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
3	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
4	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
...	...	...	...	...	...	...	...	
234424	zydeco ya ya deviled eggs	308080	40.0	37779	2008-06-07	['60-minutes-or-less', 'time-to-make', 'course...	59.2, 6.0, 2.0, 3.0, 6.0, 5.0, 0.0	
234425	cookies by design cookies on a stick	298512	29.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	188.0, 11.0, 57.0, 11.0, 7.0, 21.0, 9.0	
234426	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_step
234427	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	
234428	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	

234429 rows × 26 columns

In [21]:
```
# Gets distributions of each rating by pivoting the table and dividing each
rate_dist = (
    merged_mcar.assign(review_missing = merged_mcar['minutes'].isna())
    .pivot_table(index = 'rating_original', columns = 'minutes_missing', agg
)

rate_dist.columns = ['minutes_missing = False', 'minutes_missing = True']

rate_dist = rate_dist / rate_dist.sum()

rate_dist
```

Out[21]:

rating_original	minutes_missing = False	minutes_missing = True
1.0	0.013082	NaN
2.0	0.010794	NaN
3.0	0.032691	NaN
4.0	0.170045	0.25
5.0	0.773389	0.75

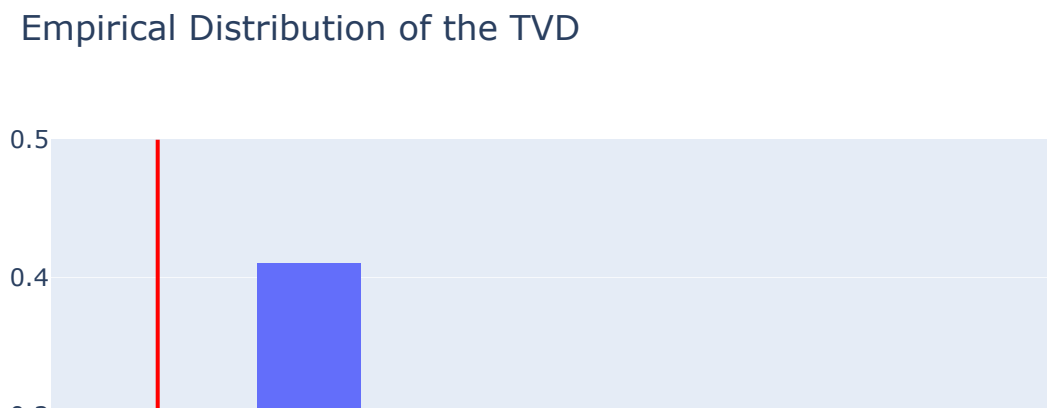
In [22]:
```
# Repeats previous steps (with addition on shuffling the columns)
# with many repetitions to get
# test statistics TVDs we will be testing against our observed TVD
n_repetitions = 100
shuffled = merged_mcar.copy()
tvds = []

for i in range(n_repetitions):
    shuffled['rating_original'] = np.random.permutation(
        shuffled['rating_original'])
    pivoted = (
        shuffled.pivot_table(index = 'rating_original',
                             columns = 'minutes_missing',
                             aggfunc = "size")
```

```
        .apply(lambda x: x / x.sum())
    )

    tvd = pivoted.diff(axis = 1).iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)
```

In [23]:
```
# Finds observed TVD
observed_tvd = rate_dist.diff(axis = 1).iloc[:, -1].abs().sum() / 2
observed_tvd
```

Out[23]:
```
0.051671916094243556
```

In [24]:
```
# Plots distribution of TVDs with observed TVD labeled
fig = px.histogram(pd.DataFrame(tvds), x=0, nbins=15,
                   histnorm='probability',
                   title = 'Empirical Distribution of the TVD')
fig.add_vline(x = observed_tvd, line_color = 'red')
fig.add_annotation(text =
                   f'<span style ="color:red">Observed TVD = \
                   {round(observed_tvd, 2)}</span>',
                   x = 20 * observed_tvd,
                   showarrow = False,
                   y = 0.16)
fig.update_layout(yaxis_range = [0, 0.5])
```



In [25]:
```
# calculated p_value by getting the mean of the number of TVDs
#that was above or equal to the observed tvd
p_value1 = np.mean(np.array(tvds) >= observed_tvd)
p_value1
```

Out[25]: 0.79

We fail to reject the null, hence we conclude that the missingness in the minutes column is not dependent on the rating_original.

Missingness of 'minutes' on 'n_steps'

In [26]:
```
# Adds a column of True or False depending on if minutes is missing
merged_mcar2 = new_merged.copy()
merged_mcar2['minutes_missing'] = merged_mcar2['minutes'].isna()
```

In [27]:
```
# Gets distributions of each minutes_missing for each n_step by pivoting the
# table and dividing each category by n_step total
nstep_dist = (
    merged_mcar2.assign(minutes_missing =
                        merged_mcar2['minutes'].isna())
    .pivot_table(index = 'n_steps',
```

```
                columns = 'minutes_missing',
                aggfunc = 'size')
)

nstep_dist.columns = ['minutes_missing = False',
                      'minutes_missing = True']

nstep_dist = nstep_dist / nstep_dist.sum()

nstep_dist
```

Out[27]:

n_steps	minutes_missing = False	minutes_missing = True
1	0.011462	NaN
2	0.032454	NaN
3	0.049833	NaN
4	0.063645	NaN
5	0.075841	NaN
...	...	...
87	0.000043	NaN
88	0.000043	NaN
93	0.000017	NaN
98	0.000004	NaN
100	0.000013	NaN

84 rows × 2 columns

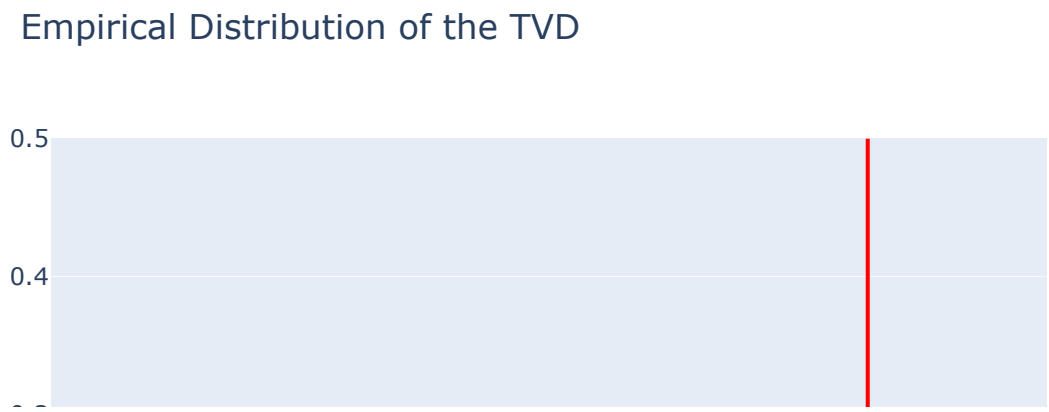
In [28]:
```
# Repeats previous steps (with addition on shuffling the
# n_step column) with many repetitions to get testing tvds
# we will be testing against our observed tvd
n_repetitions = 100
shuffled = merged_mcar2.copy()
tvds2 = np.array([])

for i in range(n_repetitions):
    shuffled['n_steps'] = np.random.permutation(shuffled['n_steps'])
    pivoted = (
        shuffled.pivot_table(index = 'n_steps',
                             columns = 'minutes_missing',
                             aggfunc = "size")
        .apply(lambda x: x / x.sum())
    )
    tvd = pivoted.diff(axis = 1).iloc[:, -1].abs().sum() / 2
    tvds2 = np.append(tvds2, tvd)
```

In [29]:
```
# Calculates the observed TVD
observed_tvd2 = nstep_dist.diff(axis = 1).iloc[:, -1].abs().sum() / 2
```

```
observed_tvd2
```

Out[29]: 0.4347912329795584

In [30]:
```
# Plots distribution of TVDs with observed TVD labeled red
fig = px.histogram(pd.DataFrame(tvds2), x=0, nbins=15,
                   histnorm='probability',
                   title = 'Empirical Distribution of the TVD')
fig.add_vline(x = observed_tvd2, line_color = 'red')
fig.add_annotation(text =
                   f'<span style = "color:red">Observed TVD = \
                   {round(observed_tvd2, 2)}</span>',
                   x = 1.5 * observed_tvd2,
                   showarrow = False, y = 0.16)
fig.update_layout(yaxis_range = [0, 0.5])
```



In [31]:
```
p_value2 = np.mean(np.array(tvds2) >= observed_tvd2)
p_value2
```

Out[31]: 0.03

In this case, we reject the null, hence we conclude that the missingness in the minutes column is dependent on the n_steps column.

Permutation Testing

In [32]:
```
#Creates a new column in the copy of new_merged that shows if a
#recipe has 'minutes' below the median of the dataset.
copy = new_merged.copy()
copy = copy[~copy['review'].isna()]
copy['under median'] = copy['minutes'] <= copy['minutes'].median()
copy
```

Out[32]:

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_step
0	1 brownies in the world best ever	333281	40.0	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	138.4, 10.0, 50.0, 3.0, 3.0, 19.0, 6.0	1
1	1 in canada chocolate chip cookies	453467	45.0	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	595.1, 46.0, 211.0, 22.0, 13.0, 51.0, 26.0	1
2	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
3	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
4	412 broccoli casserole	306168	40.0	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0	
...	...	...	...	...	...	...	...	
234424	zydeco ya ya deviled eggs	308080	40.0	37779	2008-06-07	['60-minutes-or-less', 'time-to-make', 'course...	59.2, 6.0, 2.0, 3.0, 6.0, 5.0, 0.0	
234425	cookies by design cookies on a stick	298512	29.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	188.0, 11.0, 57.0, 11.0, 7.0, 21.0, 9.0	
234426	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_step
234427	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	
234428	cookies by design sugar shortbread cookies	298509	20.0	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0	

234371 rows × 26 columns

In [33]:
```
#Creates two dataframes indicating the lower and
# upper half of the recipes in terms of 'minutes'
lower_half = new_merged[new_merged['minutes'] <=
                        new_merged['minutes'].median()]
upper_half = new_merged[new_merged['minutes'] >
                        new_merged['minutes'].median()]
```

In [34]:
```
#Calculates mean rating_average in lower half
lower_half['rating_average'].mean()
```

Out[34]: 4.692829450173364

In [35]:
```
#Calculates mean rating_average in upper half
upper_half['rating_average'].mean()
```

Out[35]: 4.658544296337589

In [36]:
```
#Calculates the observed difference in rating_average
# in the two samples
observed_stat = lower_half['rating_average'].mean() - \
upper_half['rating_average'].mean()
```

In [37]:
```
#Runs permutation simulation for the test statistic for 100 times.
#Calculate final p-value.
test_stats = []
for _ in range(100):
    copy['under median'] = np.random.permutation(copy['under median'])
    stat = copy[copy['under median'] == True]['rating_average'].mean() - \
    copy[copy['under median'] == False]['rating_average'].mean()
    test_stats.append(stat)

(test_stats >= observed_stat).mean()
```

Out[37]: 0.0

__Null Hypothesis__: In the population, the average rating of recipes with minutes under the median minutes of the dataset and average rating of recipes with minutes above the median minutes come from the same population distribution.

__Alternative Hypothesis__: In the population, the average rating of recipes with minutes under the median minutes of the dataset is higher than the average rating of recipes with minutes above the median minutes.

__Test Statistic__: The difference in means of the average ratings of the two samples of the dataset.

__Significance Level__: 0.05

__p-value__: 0.0

__Conclusion__: We reject the null hypothesis that the two samples are from the same distribution.

We believe, intuitively, that the average ratings of recipes that require fewer minutes will receive higher rating. However, we need to select a threshold to determine how to separate the two samples. Median is the more natural choice because it is more robust against outliers (as we have shown in previous sections, there exists some extreme outliers in the dataset). In addition, the size of the dataset also allows us to generalize the sample median to the population. Using the median, we are able to separate the samples and retrieve the mean of of average ratings of each sample. Since we are trying to prove that average rating of the first sample (sample with lower minutes) is higher than the second, we use signed difference in mean, instead of absolute difference in mean.

In [ ]: