Homework 1

Melody Zhao - xz2996

Due: Friday Oct 7th 11:59pm ET

In this homework we'll do some data exploration and perform a hypothesis test.

Instructions

Follow the comments below and fill in the blanks (____) to complete.

When completed,

1. Replace Name and UNI in the first cell and filename
2. Kernel -> Restart & Run All to run all cells in order
3. Print Preview -> Print (Landscape Layout) -> Save to pdf
4. Post pdf to GradeScope

Environment Setup

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt

sns.set_style('darkgrid')

%matplotlib inline
```

Part 1: Data Exploration

One data science task, and a common one used for data science interviews, is to predict defaults on loans.
We're going to load a subset of a common loan dataset and explore some of the features.

Here is a brief description of the features included:

- **purpose**: The purpose of the loan, such as: credit_card, debt_consolidation, etc.
- **annual_inc**: Annual income of the borrower
- **home_ownership**: The borrower's relationship with their primary residence
- **loan_amnt**: The amount of money applied for
- **outcome**: The result of the loan: paid off or default

In [5]:
```
# 1. (1pt) Load the data from ../data/loan_data_subset.csv into the variabl
#    using the column 'id' as the index with index_col='id'
#    note: use the default separator ','

df = pd.read_csv('../data/loan_data_subset.csv',index_col='id')
```

In [6]:
```
# 2. (1pt) Using .shape, how many rows and columns does the dataset have?

print(f'dataframe has {df.shape[0]} rows and {df.shape[1]} columns.')
```

```
dataframe has 1000 rows and 5 columns.
```

In [7]:
```
# 3. (1pt) Display the first 3 rows of the dataset using .head()

df.head(3)
```

Out[7]:

id	purpose	annual_inc	home_ownership	loan_amnt	outcome
id0	credit_card	40000	MORTGAGE	7875	paid off
id1	debt_consolidation	47000	MORTGAGE	9325	paid off
id2	debt_consolidation	28264	RENT	10600	paid off

In [8]:
```
# 4. (1pt) Print out the first 3 rows of the numeric feature columns includ
#    (3 rows x 2 columns)

df.loc['id0':'id2', ['annual_inc','loan_amnt']]
```

Out[8]:

id	annual_inc	loan_amnt
id0	40000	7875
id1	47000	9325
id2	28264	10600

In [9]:
```
# 5. (1pt) Print out the first 3 rows of the the categorical feature column
#    (3 rows x 3 columns)
df.loc['id0':'id2', ['purpose','home_ownership','outcome']]
```

Out[9]:

id	purpose	home_ownership	outcome
id0	credit_card	MORTGAGE	paid off
id1	debt_consolidation	MORTGAGE	paid off
id2	debt_consolidation	RENT	paid off

In [10]:
```
# 6. (1pt) Display all columns for rows with id from id100 to id102 inclusi
#    We should see 3 rows, 5 columns

df.loc['id100':'id102']
```

Out[10]:

id	purpose	annual_inc	home_ownership	loan_amnt	outcome
id100	credit_card	75000	RENT	10000	paid off
id101	other	72000	RENT	3000	paid off
id102	debt_consolidation	79000	RENT	16000	paid off

In [117]:
```
# 7. (3pt) Display annual_inc and home_ownership columns for the 3 rows wit
#    We should see 3 rows, 2 columns
df[['annual_inc','home_ownership']].sort_values(by=['annual_inc'], ascendin
```

Out[117]:

id	annual_inc	home_ownership
id768	367000	MORTGAGE
id201	334000	OWN
id419	310000	MORTGAGE

In [181]:
```
# 8. (3pt) What is the mean annual_inc for rows with:
#        (loan_amnt greater than the median loan_amnt) and
#        (outcome of 'paid off') and
#        (home_ownership of 'MORTGAGE' or 'OWN')
loan_amnt_median = df.loan_amnt.median()
mean_annual_inc = df.loc[(df.loan_amnt > loan_amnt_median) & (df.outcome ==
# Print the mean annual income found with precision of 2
print(f'{mean_annual_inc = :0.2f}')
```

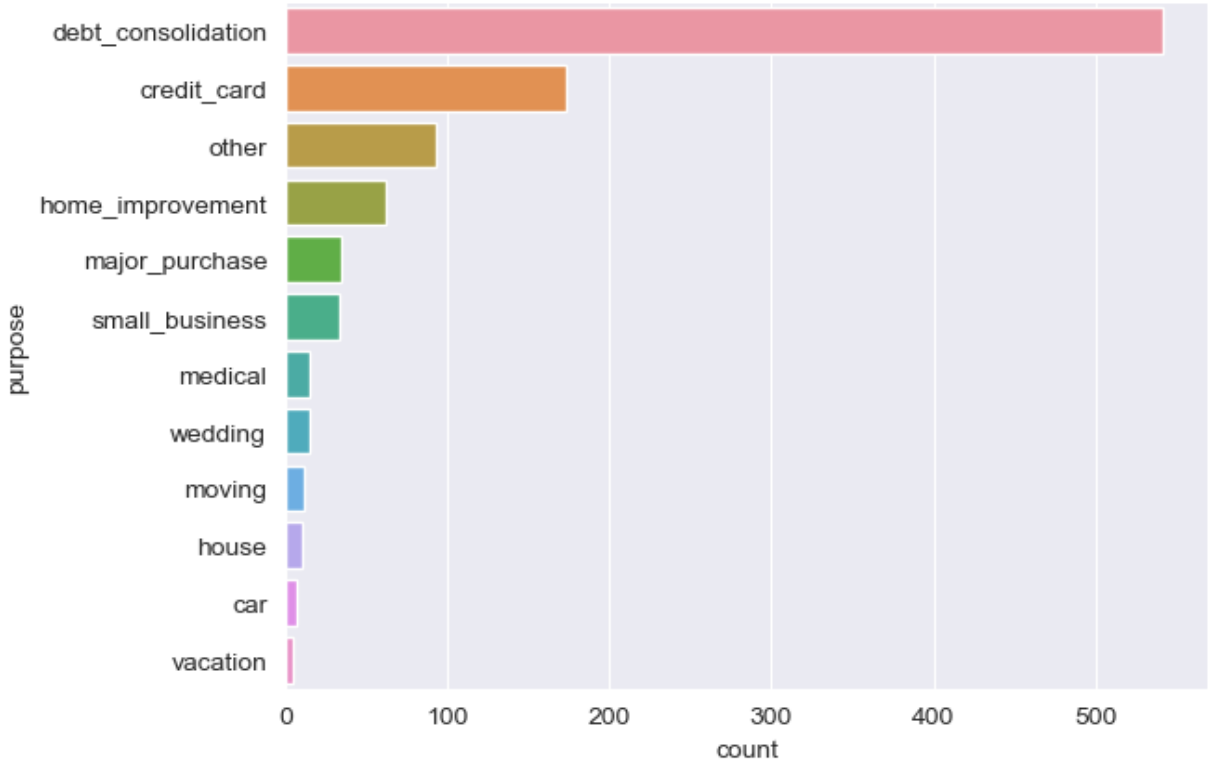
```
mean_annual_inc = 98223.29
```

In [93]:
```
# 9. (1pt) Calculate frequencies of the different values seen in column 'pu
#    Store in purpose_counts.

purpose_counts = df.purpose.value_counts()

print(purpose_counts)
```

```
debt_consolidation    542
credit_card           173
other                  93
home_improvement       62
major_purchase         34
small_business         33
medical                15
wedding                15
moving                 12
house                  10
car                     7
vacation                4
Name: purpose, dtype: int64
```

In [179]:
```
# 10. (3pt) Plot the frequency of each of the categories seen in the 'purpo
#   Order the bars using the purpose_counts.index, generated in the cell ab
#      which is sorted by frequency by default. (use the order= argument in
#   Because there are many values, and some of the labels are long,
#      place 'purpose' on the y-axis instead of the x-axis (use y= instead

sns.countplot(
    y = 'purpose',
    data = df,
    order = purpose_counts.index
)
```

Out[179]:
```
<AxesSubplot: xlabel='count', ylabel='purpose'>
```



In [28]:
```
# 11. (2pt) What is the mean loan_amnt for each category in purpose?
#     Use groupby()
#     Sort the resulting series by value ascending (default)

df.groupby('purpose')['loan_amnt'].mean().sort_values(ascending=True)
```

Out[28]:
```
purpose
moving                 4933.333333
car                    5542.857143
medical                6666.666667
vacation               7700.000000
wedding                9153.333333
other                  9758.064516
major_purchase        11732.352941
home_improvement      12114.516129
credit_card           12776.589595
debt_consolidation    14440.221402
house                 14717.500000
small_business        15344.696970
Name: loan_amnt, dtype: float64
```

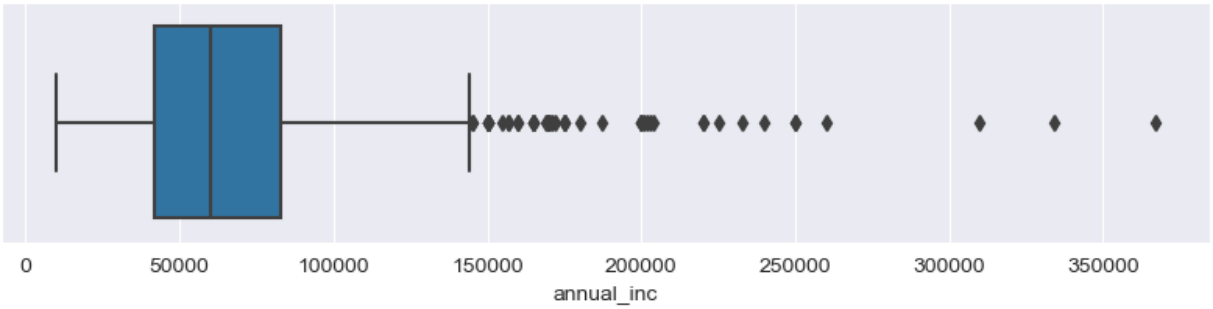
In [30]:
```
# 12. (1pt) Display the summary statistics of annual_inc using .describe()
#     Round all values to the hundredths place (precision of 2) using .roun

df.annual_inc.describe().round(2)
```

Out[30]:
```
count      1000.00
mean      68158.89
std       40271.75
min       10000.00
25%       42000.00
50%       60000.00
75%       83000.00
max      367000.00
Name: annual_inc, dtype: float64
```

In [62]:
```
# 13. (2pt) There appears to be a fairly large difference between mean and
#   Print out the absolute difference in mean annual_inc and median annual_
#   To calculate the absolute value, use np.abs()

annual_inc_mean = df.annual_inc.mean()

annual_inc_median = df.annual_inc.median()

print(f'absolute difference = {np.abs(annual_inc_mean - annual_inc_median):
```

```
absolute difference = 8158.89
```

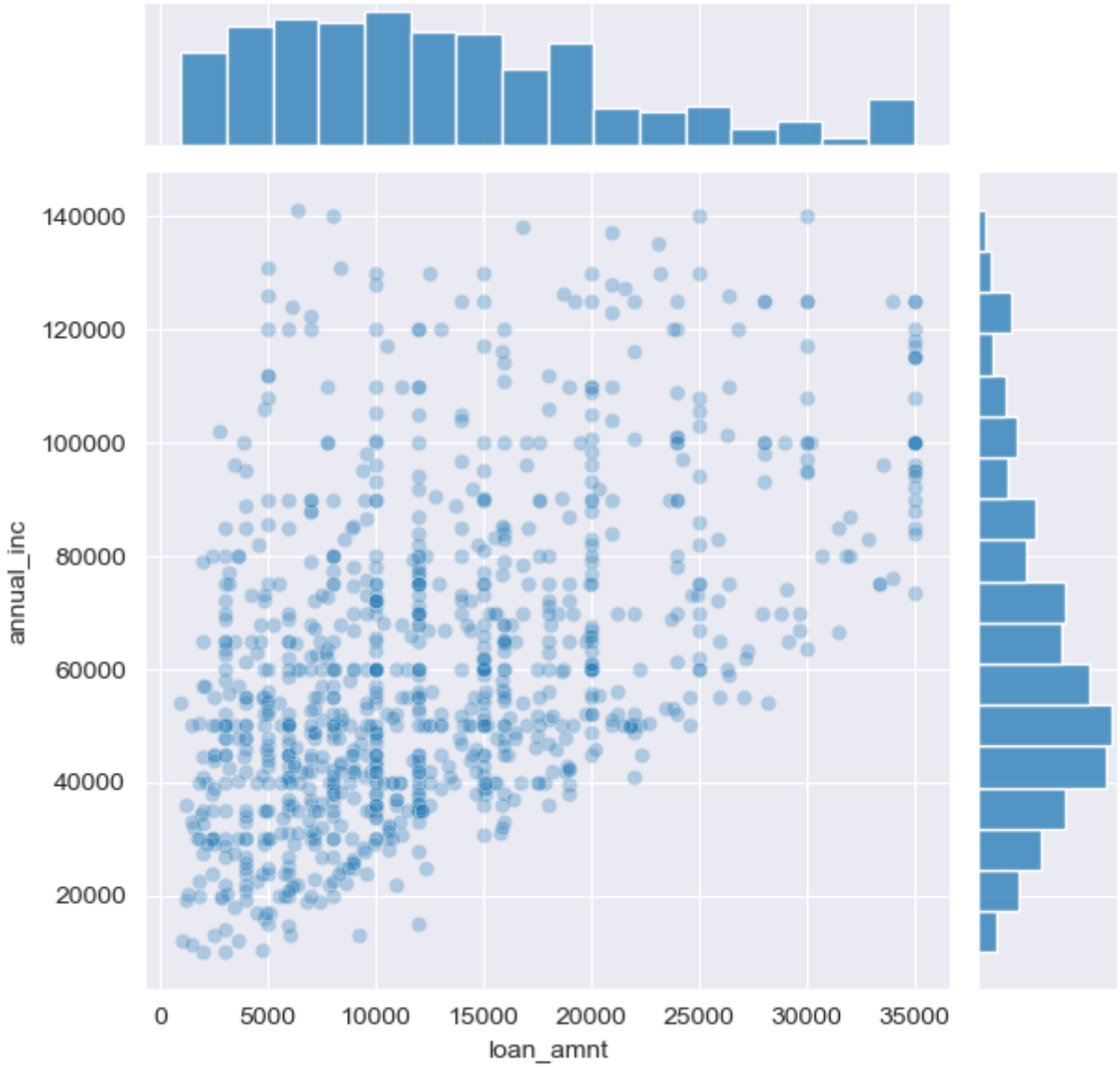
In [56]:
```
# 14. (2pt) Display a boxplot of annual_inc using sns.boxplot.

#  To make a wide plot, use plt.subplots with 1 row, 1 column of axes and a
fig,ax = plt.subplots(1,1, figsize=(10,2))

#  Plot a boxplot of annual_inc using sns.boxplot() and ax with annual_inc
sns.boxplot(x = df.annual_inc,ax = ax);
```



In [57]:
```
# 15. (1pt) We'll remove some of records with the highest annual_inc, treat
#    What is the 95th percentile of annual_inc? (use .percentile() from num
#    Eg. Where is the cutoff where we remove extremely high values but keep

annual_inc_95 = df.annual_inc.quantile(.95, interpolation = 'linear')

print(f'95th percentile of annual_inc: {annual_inc_95:0.2f}')
```

```
95th percentile of annual_inc: 141195.95
```

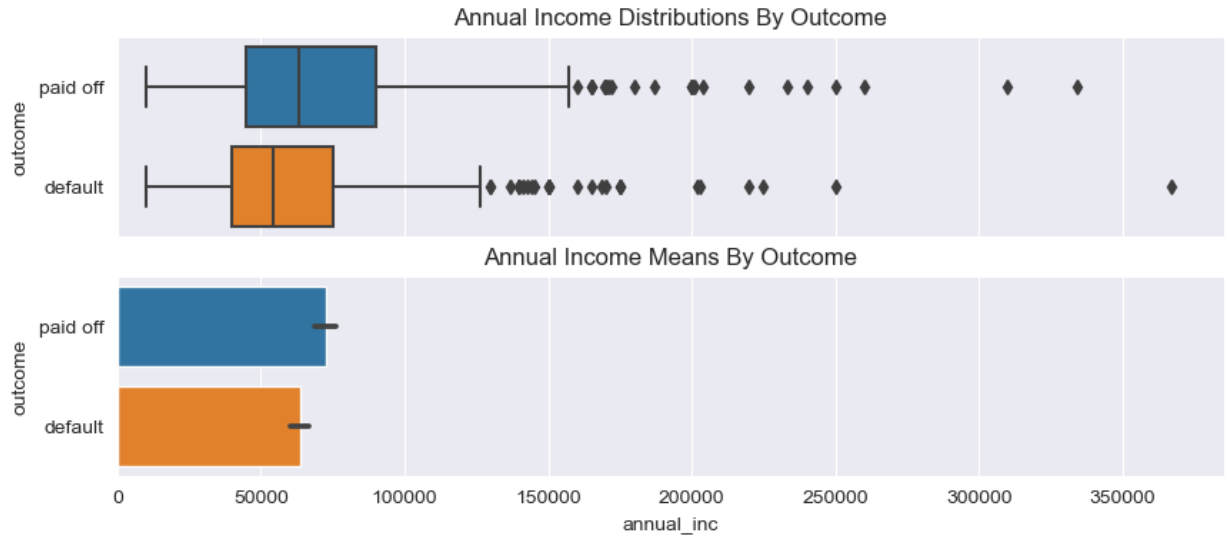
In [141]:
```
# 16. (3pt) Plot loan_amnt (x-axis) against annual_inc (y-axis) using sns.j
#     Only include rows where annual_inc < annual_inc_95
#     Set alpha=0.3 to add transparency to markers
new = df.loc[df.annual_inc < annual_inc_95, 'annual_inc']
sns.jointplot(x = df.loan_amnt, y = new , alpha = 0.3);
```



In [159]:
```
print(df.loan_amnt);
```

```
id
id0       7875
id1       9325
id2      10600
id3       6000
id4       7900
         ...
id995    18825
id996    11700
id997     7000
id998     7000
id999    19000
Name: loan_amnt, Length: 1000, dtype: int64
```

In [114]:
```
# 17. (5pt) Visualize annual income (annual_inc) by outcome.
#     Outcome takes two values: 'paid off' and 'default'

# NOTE: In all of the below use all rows of df, no longer limiting to df.an

# Here we'll create 2 plots, one that compares the distributions of annual_
# the other comparing the mean of annual_inc by outcome

# Create a subplot with 2 rows and 1 column with figsize of (10,4)
# Use sharex=True to share the x-axis across the two plots
# Capture the return values of plt.subplots() as fig,ax
fig,ax = plt.subplots(2,1,figsize=(10,4), sharex = True)

# On the first axis (ax[0]) use sns.boxplot() to compare the distribution o
#   Place 'annual_inc' on the x-axis and 'outcome' on the y-axis.
sns.boxplot(x = df.annual_inc, y = df.outcome, ax=ax[0])

# Set the title on the first axis ax[0] to be 'Annual Income Distributions
ax[0].set_title ('Annual Income Distributions By Outcome')

# On the second axis (ax[1]) use sns.barplot() to compare the means of annu
#   Place 'annual_inc' on the x-axis and 'outcome' on the y-axis.
sns.barplot(x='annual_inc', y= 'outcome', data = df, ax=ax[1])

# Set the title on the second plot to be 'Annual Income Means By Outcome'
ax[1].set_title('Annual Income Means By Outcome')

# Remove the label on the x-axis of ax[0] using set_xlabel() (as it overlap
ax[0].set_xlabel(None);
```



Part 2: Hypothesis Testing

The plots in the question above indicate a difference in annual_inc by outcome.

Let's test the hypothesis that there is a difference in mean annual_inc for loans with an outcome of 'paid off' vs loans with an outcome of 'default'.

In [129]:
```
# 18. (3pt) Calculate the difference in mean annual_inc between 'paid off'
#     Use: mean_annual_inc_paid_off - mean_annual_inc_default

# Calculate the mean value for each group
mean_annual_inc_paid_off = df.loc[df.outcome == 'paid off', 'annual_inc'].m
mean_annual_inc_default = df.loc[df.outcome == 'default', 'annual_inc'].mea
observed_mean_diff = mean_annual_inc_paid_off - mean_annual_inc_default

# Print the the value of observed_mean_diff with a precision of 2
print(observed_mean_diff.round(2))
```

```
9062.74
```

In [164]:
```
# 19. (5pt) We'll perform a permutation test to see how significant this di
#    by generating 1,000 random permutation samples of mean difference

rand_mean_diffs = []
n_samples = 1000
n_paid_off = len(df[df['outcome']== 'paid off']) # the number of observatio
print(f'{n_paid_off = :d}')

for i in range(n_samples):

    # Get a random permutation of df.annual_inc
    # Use the pandas .sample() function with
    #    sample size the same size as original dataset
    #    sampling without replacement
    #    random_state == i (the index of the loop) for consistency in gradi
    rand_perm = df.annual_inc.sample(frac =1, replace = False, random_state

    # Take the mean of the first n_paid_off random values
    rand_mean_paid_off = rand_perm[:n_paid_off].mean()

    # Take the mean of the remaining random values
    rand_mean_default = rand_perm[n_paid_off:].mean()

    # Append the difference (rand_mean_paid_off - rand_mean_default) to the
    rand_mean_diffs.append(rand_mean_paid_off - rand_mean_default)

# Convert rand_mean_diffs into a numpy array so we can use numpy functions
rand_mean_diffs = np.array(rand_mean_diffs)

# check that we have the correct amount of data by asserting that the lengt
assert rand_mean_diffs.shape[0] == n_samples

# check that we only have one array of differences
assert rand_mean_diffs.ndim == 1

# Display the first three values in rand_mean_diffs so we know when it's do
rand_mean_diffs[:3]
```

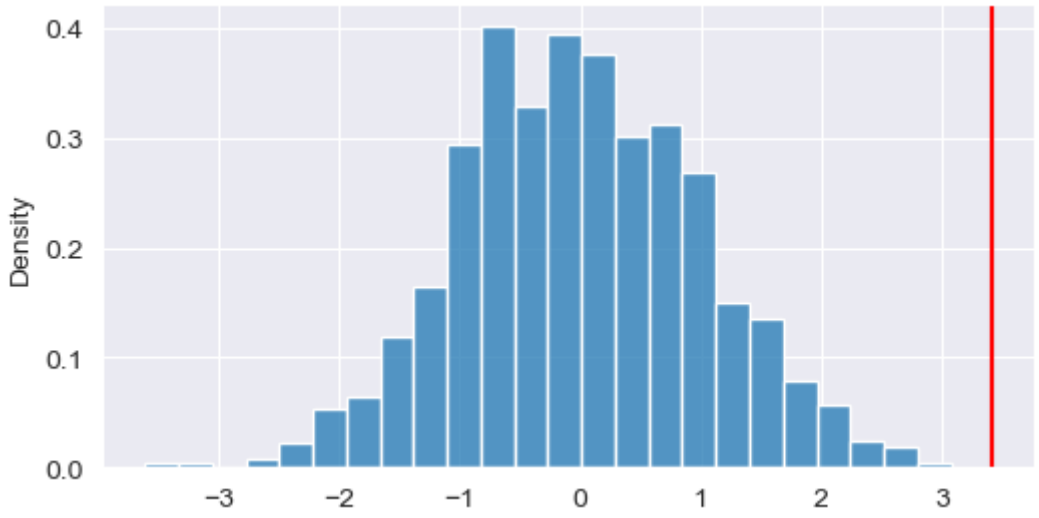
```
n_paid_off = 500
```

Out[164]:
```
array([ 2323.292,  3927.652, -4313.772])
```

In [165]:
```
# 20. (5pt) Before we plot the data, let's transform all values to their z-

# Calculate the sample mean of our rand_mean_diffs using .mean()
mean_rand_mean_diffs = rand_mean_diffs.mean()

# Calculate the sample standard deviation using .std()
std_rand_mean_diffs = rand_mean_diffs.std()

# Transform rand_mean_diffs to rand_mean_diffs_zscore by
#    first subtracting the mean and
#    then dividing by the std dev
rand_mean_diffs_zscore = (rand_mean_diffs - mean_rand_mean_diffs)/std_rand_

# Transform the observed_mean_diff as well by subtracting the mean and divi
observed_mean_diff_zscore = (observed_mean_diff - mean_rand_mean_diffs)/std

# To check our transformation, check that the zscore mean is near 0 and std
print(f'{rand_mean_diffs_zscore.mean() = :0.3f}')
print(f'{rand_mean_diffs_zscore.std()  = :0.3f}')
print(f'{observed_mean_diff_zscore     = :0.3f}')

assert np.abs(rand_mean_diffs_zscore.mean() - 0) < .0001, 'rand_mean_diffs_
assert np.abs(rand_mean_diffs_zscore.std()  - 1) < .0001, 'rand_mean_diffs_
```

```
rand_mean_diffs_zscore.mean() = 0.000
rand_mean_diffs_zscore.std()  = 1.000
observed_mean_diff_zscore     = 3.415
```

In [134]:
```
# 21. (2pt) Plot our observed metric against our samples.

# Use subplots to create a figure with 1 row, 1 columna and figsize of (6,3
fig,ax = plt.subplots(1,1,figsize=(6,3))

# Use seaborn histplot to plot the distribution of rand_mean_diffs_zscore o
ax = sns.histplot(rand_mean_diffs_zscore, stat='density')

# Use ax.axvline() to plot a line at our observed_mean_diff_zscore
# Make the line red using color='r'
ax.axvline(observed_mean_diff_zscore,color='r');
```



In [176]:
```
# 22. (3pt) The plot seems to indicate a real difference in values. What is
#    Calculate a two-tailed p_value using np.abs()
#    Recall that we want the proportion of random samples (rand_mean_diffs_
#      greater than or equal to the absolute value of the observed differen
ad_gt = np.abs(rand_mean_diffs_zscore) >= np.abs(observed_mean_diff_zscore)
p_value = ad_gt.sum()/len(rand_mean_diffs_zscore)

# print the p-value found
p_value
```

Out[176]: 0.001