CS 480 HW #2

1. Effort level.

a. The homework took me 12 hours.

b. N\A

c. Used ChatGPT as a tool to look up syntax and documentation for Scikit-learn and related machine learning libraries.

2. Paired t-test.

a. 95% confidence interval.

Below, I provide the code for constructing the 95% confidence interval.

```
from scipy.stats import norm, ttest_rel, t
import numpy as np
# 2a
f = open('./data.txt', 'r')
A = []
B = []
for line in f:
  line = line.strip().split()
A.append(float(line[0]))
B.append(float(line[1]))


a = np.array(A)
b = np.array(B)
```

```
mean_a = np.mean(a)
mean_b = np.mean(b)

std_a = np.std(a,ddof=1)
std_b = np.std(b,ddof=1)

n = len(a)
print(n)
alpha = 0.05
tci_a = t.interval(1-alpha, n-1, mean_a, std_a/np.sqrt(n))
tci_b = t.interval(1-alpha, n-1, mean_b, std_b/np.sqrt(n))

print('Algorithm A CI:')
print(tci_a)
print()
print('Algorithm B CI:')
print(tci_b)
```

The output of the Python script is reported.

Algorithm A CI: (np.float64(2.3434452353035717), np.float64(3.8232214313630952))

Algorithm B CI: (np.float64(1.5268346967576174), np.float64(2.9731653032423826))

Based on the reporting, we can say that two confidence intervals overlap. However, we cannot say that one algorithm is better than the other since they do indeed overlap, and a paired t-test is required to determine if one is better than the other in a statistically meaningful way.

b. Two-sided paired t-test for null-hypothesis.

The code for computing the t-test is as follows.

```
print(ttest_rel(a, b).confidence_interval)
```

This outputs the following result.

TtestResult(statistic=np.float64(2.277867258047101), pvalue=np.float64(0.043699584804600185), df=np.int64(11))

Since the standard t-distribution with 11 degrees of freedom and 95% confidence is known to be 2.201, and so interval being [-2.201, 2.201], we know that our t-stat of 2.278 goes out of this interval, so we are safe to reject the null hypothesis and say that one algorithm has a statistically difference in performance compared to the other. From this experiment, we can see that two-sided paired t-test to find a statistically significant difference between two algorithms is concrete and deterministic while comparing two separate confidence intervals does not really allow us to conclude anything, even if they do overlap as they did in this case.

c. Report p-value for testing done in b.

The p-value as reported in part b is 0.044. We know that the two algorithms are statistically different because our p-value falls below our 0.05 significance threshold. This is another way to determine if the results of two models are statistically different from each other.

d. Describe pseudocode for bootstrapping method for this problem.

The pseudocode for bootstrapping is as follows.

Obtain F1 score for each fold.

Sort them in increasing order.

Compute the top .25 quantile.

Compute the bottom .25 quantile.

The CI is [F1 - top quantile, F1 + lower quantile]

e. Implement and perform bootstrapping paired test with computed p-value.

3. Linear regression.

a. Plot year versus ice days.

Below, I provide the plot of annual ice cover days for Lake Mendota and Lake Monona.

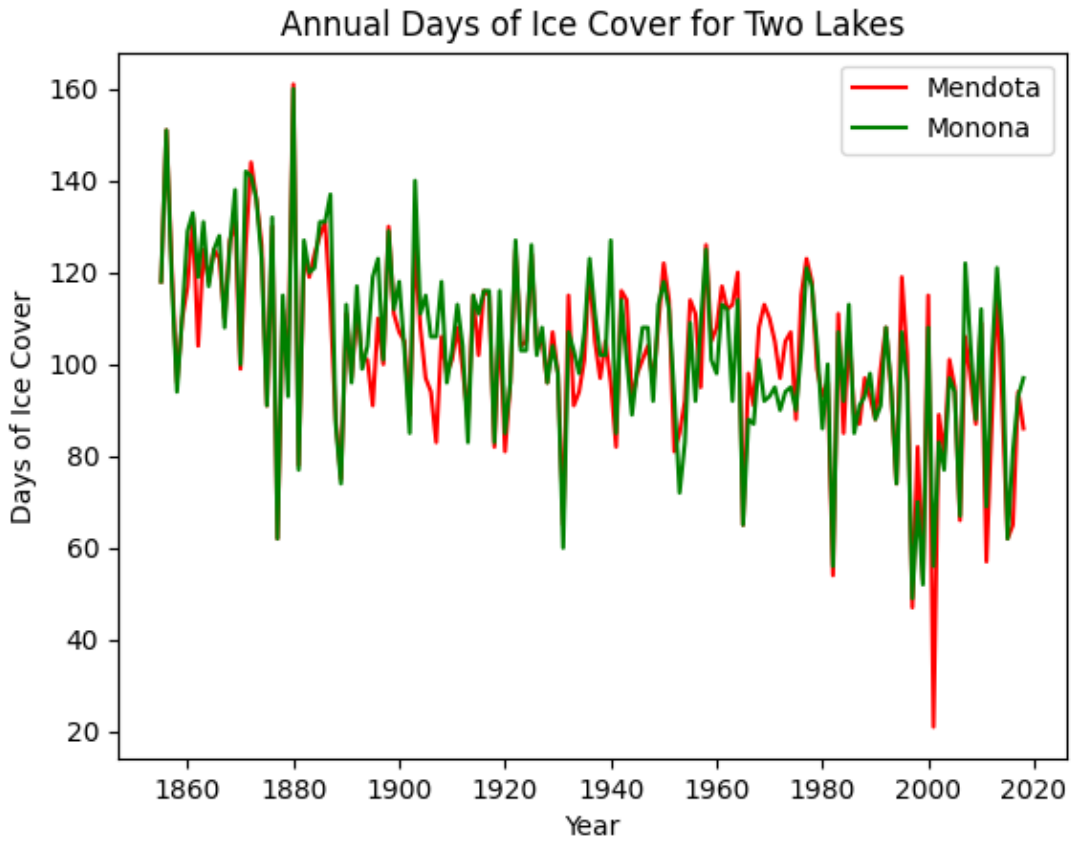

Figure 1: Plot of annual ice cover days for Lake Mendota (red) and Lake Monona (green) using Python.

Below, I provide the plot of the difference in annual days of ice cover between Lake Monona and Lake Mendota (Lake Monona - Lake Mendota).

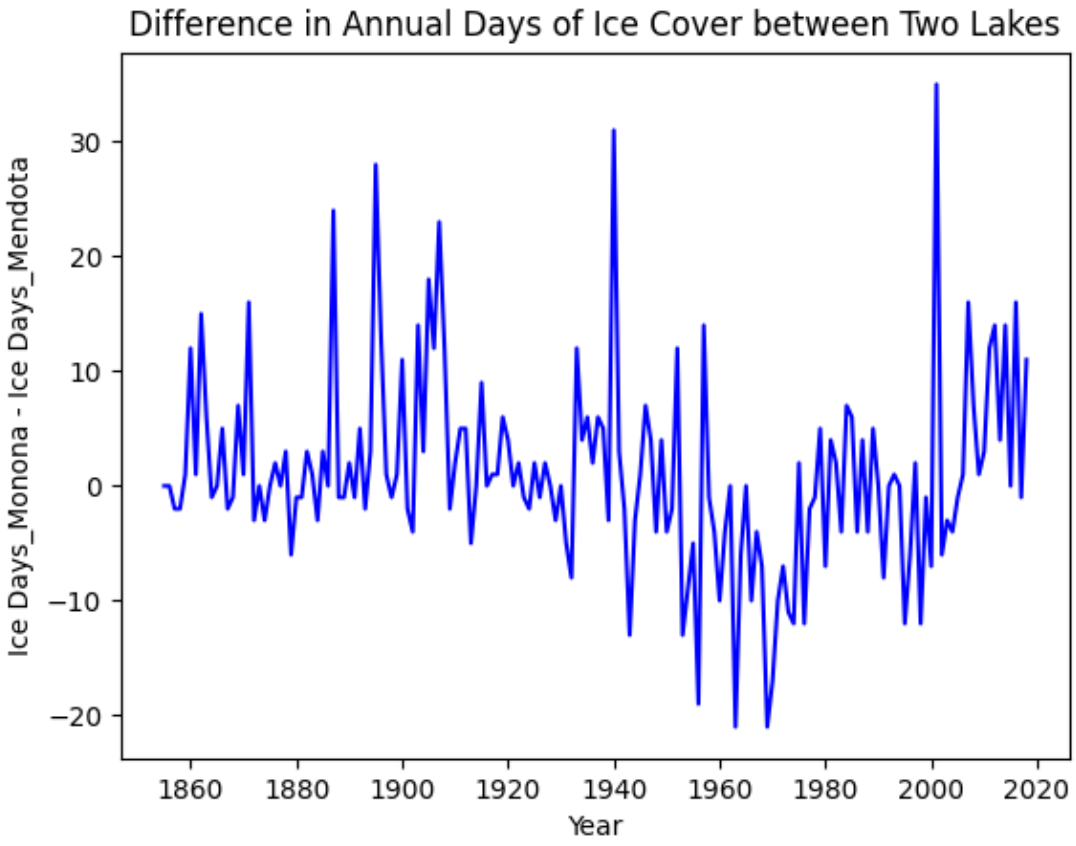

Figure 2: Plot of the difference in annual days of ice cover between Lake Monona and Lake Mendota using Python.

Below is the Python script used to generate the two plots.

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np


# load in csv data

col = 'Days of Ice Cover'
```

```
mendota_df =
pd.read_csv('mendota.csv')
.loc[5:175].dropna(subset=[col]).iloc[::-1].reset_index(drop=True)

monona_df =
pd.read_csv('monona.csv')
.loc[6:176].dropna(subset=[col]).iloc[::-1].reset_index(drop=True)

# 3a
plt.figure(1)
plt.plot([x for x in range(1855,2019)],mendota_df[col], color='red')
plt.plot([x for x in range(1855,2019)],monona_df[col], color='green')
plt.xlabel('Year')
plt.ylabel(col)
plt.title('Annual Days of Ice Cover for Two Lakes')
plt.legend(['Mendota', 'Monona'])
plt.savefig('ice_cover.png')
plt.show()

diff = []
for i,j in zip(monona_df[col],mendota_df[col]):
    diff.append(i-j)

plt.figure(2)
plt.plot([x for x in range(1855,2019)],diff, color='blue')
plt.xlabel('Year')
```

```
plt.ylabel('Ice Days_Monona - Ice Days_Mendota')
plt.title('Difference in Annual Days of Ice Cover between Two Lakes')
plt.savefig('diff.png')
plt.show()
```

b. Split the datasets into training and testing. Compute std and mean for the two lakes respetively.

The results are as follows.

Mendota Mean: 107.19

Mendota STD: 16.74

Monona Mean: 108.48

Monona STD: 18.12

Below is the Python script used to compute the means and STDs.

```
# 3b
split = mendota_df.index[mendota_df['Winter'] == '1970-71'].tolist()[0] + 1
mendota_df_train = mendota_df.iloc[:split]
mendota_df_test = mendota_df.iloc[split:]
split = monona_df.index[monona_df['Winter'] == '1970-71'].tolist()[0] + 1
monona_df_train = monona_df.iloc[:split]
monona_df_test = monona_df.iloc[split:]

mendota_a = np.array(mendota_df_train[col])
monona_a = np.array(monona_df_train[col])

print('Mendota Mean:')
```

```
print(np.mean(mendota_a))
print('Mendota STD:')
print(np.std(mendota_a,ddof=1))
print()
print('Monona Mean:')
print(np.mean(monona_a))
print('Monona STD:')
print(np.std(monona_a,ddof=1))
```

c. Using training sets, train a linear regression model.

Scikit-learn in Python was used for this section.

Below I provide the Python script for training a linear regression model.

```
# 3c
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

lr = LinearRegression()
monona_df_train['ones'] = 1
monona_df_test['ones'] = 1
lr.fit(monona_df_train[['ones','Winter',col]], mendota_df_train[[col]])
mendota_pred = lr.predict(monona_df_test[['ones','Winter', col]])

weights = lr.coef_
print(weights)
intercept = lr.intercept_
```

```
print(intercept)
```

The results are as follows:

Feature weights: [[0., 0.04122457, 0.85295064]]

Intercept:[-64.1827663]

These are the relevant outputs of the Scikit-learn LinearRegression model. One thing to note is that although we are trying to get an intercept in our OLS solution by modeling our features as (1, x, y_{monona}), the LinerRegression model in Scikit-learn does not seem to care about an artificial '1' appended to the feature vector and will in fact produce the same result without the '1' column, so we can get away with just (x, y_{monona}). As reported above, the feature weights which represent (B_0, B_1, B_2) omits the B_0 value; probably because it ignores a column of 1's like I mentioned. Instead, it will report our B_0 (which is our intercept) in its own category called 'intercept'.

d. Mean squared error on the test set.

Below, I provide the Python script to compute the MSE of our predicted values from our Scikit-learn LinearRegression model. The computation of MSE is also provided in the Scikit-learn library.

```
# 3d
mse = mean_squared_error(mendota_df_test[[col]], mendota_pred)
print(mse)
```

The result of this script is reported below.

Mean sqaure error: 124.26409483990123

This is a fairly nice MSE since it is the square of the mean error value. It means our linear regression model is on average about 11 days off the mark every year.

e. Train a linear regression model using Monona.

Below, I provide the Python script for retraining our linear regression model on our new feature vector which just omits y_{monona}.

```
# 3e


lr2 = LinearRegression()
lr2.fit(monona_df_train[['ones', 'Winter']],mendota_df_train[[col]])
mendota_pred2 = lr2.predict(monona_df_test[['ones', 'Winter']])
mse2 = mean_squared_error(mendota_df_test[[col]], mendota_pred2)


weights2 = lr2.coef_
intercept2 = lr2.intercept_


print('Feature weights: ' + str(weights2))
print('Intercept:' + str(intercept2))
print('Mean sqaure error: ' + str(mse2))
```

I report the newly trained linear model below.

Feature weights: [[0., -0.15629877]]

Intercept:[406.11105985]

Mean sqaure error: 418.14436543672855

As seen above, the MSE is much higher when the linear model only has the year and cannot reference y_{monona} in order to predict $y_{mendota}$. This intuitively makes sense. Remember that the first element of the feature weights which should be intercept is simply reported in a separate variable called 'intercept' in Scikit-learn's LinerRegression

model.

i. Interpret the sign of γ_1.

γ_1 can be interpreted as the slope of our x variable. Since it is negative, we can say that it derived a directly negative correlation between the year and the number of ice days. In other words, as the years go by and 'increases', the number of ices days decrease. Even more simply put, our model predict a warming trend.

ii. Assessing a viewpoint formed from the model.

The stated viewpoint could be wrong because looking at our model again, it can be seen that the strongest weight $B_2 = 0.85$ is much greater than the $B_1 = 0.04$ that the analysts are referring to. B_2 is the weigh of feature y_{monona} on label $y_{mendota}$ and since they have a strong correlation to each other and are trending downwards together, there is little weight on B_1 which is the number of years since feature y_{monona} is doing all the 'heavy lifting'. In fact, it is more accurate to say that B_1 is zero-ish weight, and it was a coincidence that it happened to be barely positive as a part of the OLS solution.

4. Maximum likelihood estimation for linear regression.

a. Likelihood of the ith data point.

$Laplace(\mu, b) \rightarrow f(z|\mu, b) = \frac{1}{2b} e^{-\frac{|z-u|}{b}}$

$Laplace(w^T x, 1) \rightarrow f(z|w^T x, 1) = \frac{1}{2} e^{-|z-u|}$

$P(y_i|x_i w) = f(y_i|w^T x_i, 1)$

$= \frac{1}{2} e^{-|y_i - w^T x_i|}$

b. Log-likelihood of the dataset.

$L = P(y_1..., y_n|x_1, ..., x_n, w)$

$= \prod_{i=1}^{n} P(y_i|x_i, w)$

$= \prod_{i=1}^{n} (\frac{1}{2} e^{-|y_i - w^T x_i|})$

$\log(L) = \log(\prod_{i=1}^{n} (\frac{1}{2} e^{-|y_i - w^T x_i|}))$

$= \sum_{i=1}^{n} \log(\frac{1}{2} e^{-|y_i - w^T x_i|})$

$= n \log(\frac{1}{2}) + \sum_{i=1}^{n} \log(e^{-|y_i - w^T x_i|})$

$= n \log(\frac{1}{2}) + \sum_{i=1}^{n} (-|y_i - w^T x_i|)$

$= n \log(\frac{1}{2}) - \sum_{i=1}^{n} |y_i - w^T x_i|$

$w^{MLE} = \arg\max_w \sum_{i=1}^{n} (\log(\frac{1}{2}) - |y_i - w^T x_i|)$

$w^{MLE} = \arg\min_w \sum_{i=1}^{n} |y_i - w^T x_i|$

From our derivation, the optimization problem is to find the value of w that would minimize $\sum_{i=1}^{n} |y_i - w^T x_i|$.

This is not equivalent to the least squares problem because our derivation above shows we are minimizing the residuals of the points to a line which is consistent with our study of linear regression. In least squares however, you want to take the zero derivative of a convex, quadratic function in order to find the global minimum, and the derived w^{MLE} from finding this is, in general, not the same problem as minimizing residuals.

c. Maximum likelihood estimator w under the above model.

$w^{MLE} = \arg\min_w \sum_{i=1}^{n} |y_i - w^T x_i|$

$w^{MLE} = \min_w (|3 - w(1)| + |5 - w(1)| + |6 - w(1)|)$

$w^{MLE} = \min_w (|3 - w| + |5 - w| + |6 - w|)$

Then, let $f(w) = |3 - w| + |5 - w| + |6 - w|$. Below, I provide the plot of $f(w)$.

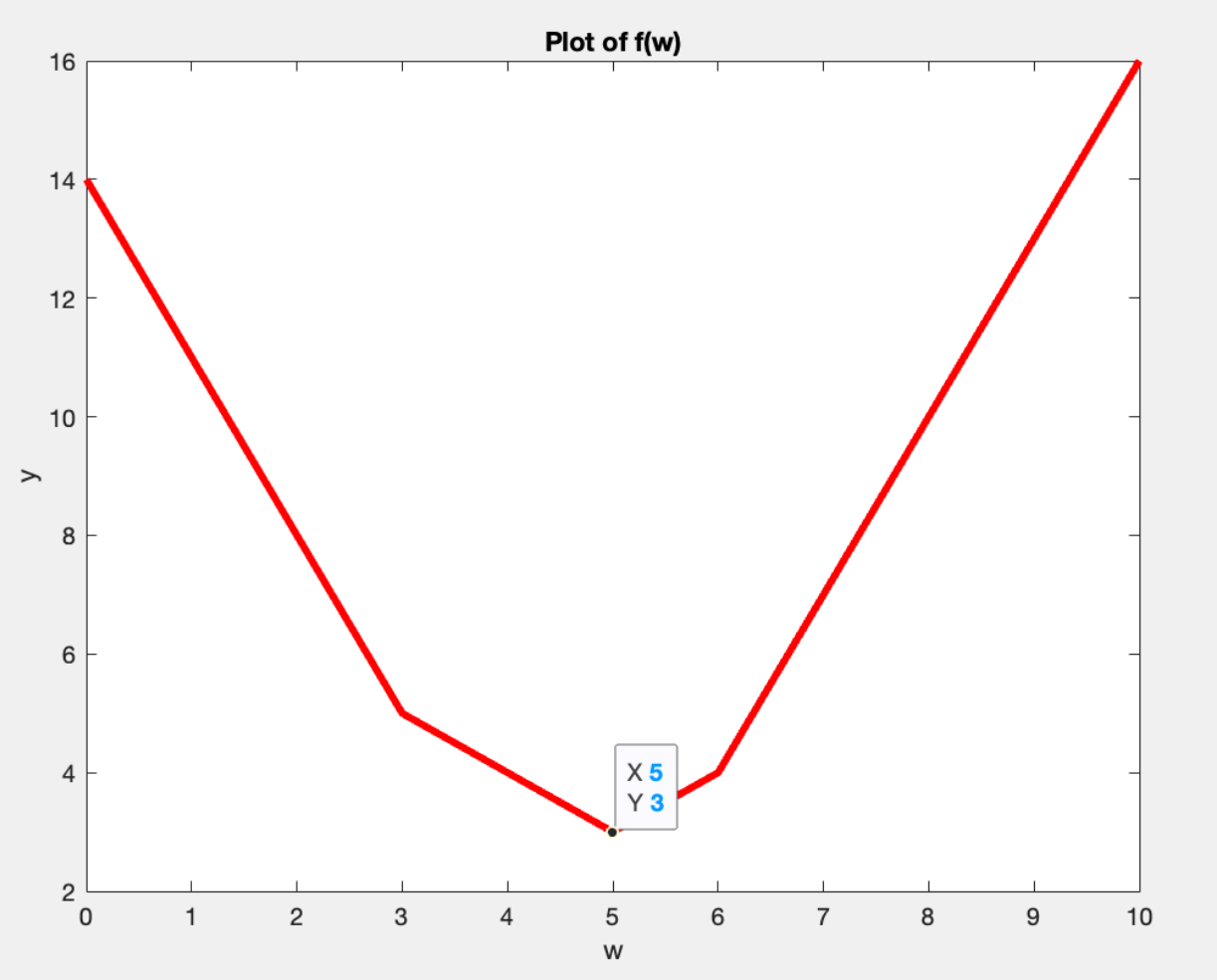

Figure 3: Plot of $f(w)$ in MATLAB. The clicked point at (5,3) is the global minimum of $f(w)$.

From the plot above, we can see that the function $f(w)$ is minimized when $w = 5$, so we have $w^{MLE} = \min_w(|3 - w| + |5 - w| + |6 - w|) = 5$.