In [1]:
```
import pandas as pd
import numpy as np
import os
import re
%matplotlib inline
import matplotlib.pyplot as plt
```

Final Exam

**PSTAT 134/234 (Spring 2023)

Data description: Fraudulent transactions

A credit card company wants to know whether a set of variables $x_1, \ldots, x_p$ have an impact on the probability of a given transaction being fraudulent. To understand the relationship between these predictor variables and the probability of a transaction being fraudulent, the company can perform logistic regression where the response is defined as:

$$y = \begin{cases} 1, & \text{if transaction is fraudulent.} \\ 0, & \text{otherwise.} \end{cases}$$

Question 1: Read Data into Python

1. Unzip folder `DataFinalExam` in your working directory.
2. Use regular expressions to filter data sets that match the conditions:
    - Starting with pattern 'Data'
    - From years: 2020, 2021, 2022
    - From months: January, February, March and April
3. Concatenate all files in a single data frame
4. Set Y to be column 'Y' and X as the remaining columns of the merged data frame.

In [2]:
```python
folder_path = 'DataFinalExam'

# Define the regular expression pattern for the desired filenames
pattern = r'^Data-20(20|21|22)-(01|02|03|04)-\d{2}.csv$'   ##Starting by 'Da

# Initialize an empty list to store the data frames
dfs = []

# Iterate over the files in the folder
for file_name in os.listdir(folder_path):
    # Check if the file matches the desired pattern
    if re.match(pattern, file_name):
        # Read the CSV file and append the data frame to the list
        file_path = os.path.join(folder_path, file_name)
        df = pd.read_csv(file_path)
        dfs.append(df)
        print(file_name)

# Concatenate the data frames into a single data frame
merged_df = pd.concat(dfs)

Y = merged_df['Y']
X = merged_df.drop('Y', axis=1)
```

```
Data-2020-04-01.csv
Data-2021-02-01.csv
Data-2021-01-01.csv
Data-2020-02-01.csv
Data-2021-04-01.csv
Data-2022-03-01.csv
Data-2022-02-01.csv
Data-2021-03-01.csv
Data-2022-04-01.csv
Data-2022-01-01.csv
Data-2020-03-01.csv
Data-2020-01-01.csv
```

Question 2

In this problem, we use convex optimization to train a logistic regression model with regularization. We are given data $(x_i, y_i), i = 1, \ldots, n$. The $x_i \in \mathbf{R}^p$ are feature vectors, while the $y_i \in \{0, 1\}$ are associated boolean classes.

The goal is to construct a linear classifier $\hat{y} = 1\left[x^T \beta > 0\right]$, which is 1 when $x^T \beta$ is positive and 0 otherwise. We model the posterior probabilities of the classes given the data linearly, with

$$\log \frac{\Pr(Y=1 \mid X=x)}{\Pr(Y=0 \mid X=x)} = x^T \beta$$

This implies that

$$\Pr(Y=1 \mid X=x) = \frac{\exp\left(x^T \beta\right)}{1+\exp(x^T \beta)}, \quad \Pr(Y=0 \mid X=x) = \frac{1}{1+\exp(x^T \beta)}.$$

We fit β by maximizing the log-likelihood of the data:

$$\ell(\beta) = \sum_{i=1}^{n} y_i x_i^T \beta - \log\left(1+\exp\left(x_i^T \beta\right)\right)$$

Because ℓ is a concave function of β, this is a convex optimization problem.

Question 2a:

1. Use gradient descent to create function `Update_Beta` that uses the data as input, to obtain the optimal value of β.
   - At each iteration of your algorithm the function should keep track of the maximum update, $Max_{update} = \|\beta_{new} - \beta\|_\infty$ and the mean absolute error defined as, $error = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{n}$
2. Use your function to estimate β.

Hint: Use the fact that maximizig the concave function $f(x)$ is equivalent to minimizing the convex function $-f(x)$.

In [3]:
```
#mean absolute error
def mean_absolute_error(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))

#logistic function
def logistic(x):
    return 1 / (1 + np.exp(-x))

#update beta
def Update_Beta(X, y, alpha=0.01, max_iteration=5000):
    n = X.shape[0]
    p = X.shape[1]
    beta = np.random.rand(p)
    errors = []
    max_updates = []

    #calculate for each iteration
    for i in range(max_iteration):
        beta_old = beta.copy()
        y_pred = logistic(X @ beta)
        gradient = (X.T @ (y - y_pred)) / n
        beta += alpha * gradient
        update = np.max(np.abs(beta - beta_old))
        max_updates.append(update)
        error = mean_absolute_error(y, y_pred)
        errors.append(error)

    return beta, errors, max_updates

beta, errors, max_updates = Update_Beta(X.values, Y.values)
print(beta)
```

```
[ 2.38069286  0.80673558 -0.61882833  0.37275741 -0.30683578 -0.75108397
  0.44609841 -0.56558719  0.12661906  0.555374    0.03034404  0.26648263
 -0.03108634  0.24360686 -0.31379374 -0.72230631 -0.51081921  0.00387857
  0.5564099   0.44455403]
```

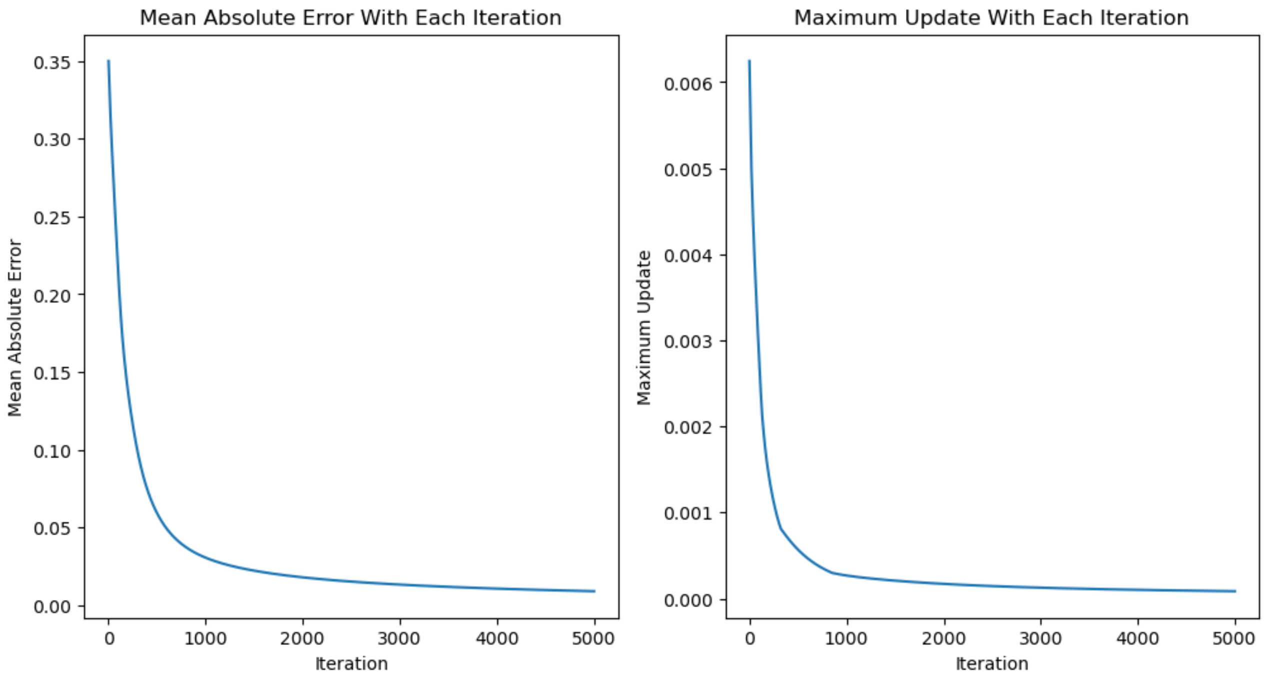
Question 2b:

Use diagnostic plots to assess the convergence of your algorithm.

In [4]:
```
#plot MAE
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(errors)
plt.title('Mean Absolute Error With Each Iteration')
plt.xlabel('Iteration')
plt.ylabel('Mean Absolute Error')

#Plot MU
plt.subplot(1, 2, 2)
plt.plot(max_updates)
plt.title('Maximum Update With Each Iteration')
plt.xlabel('Iteration')
plt.ylabel('Maximum Update')

plt.show()
```



As we can see the mean absolute error converges towards 0 with each iteration. This model is trying to minimize error with each iteration of the gradient descent algorithm with the beats being updated to minimize the absolute error. This convergence means the model is getting close to the actual values.

As we can see the maximum update with each iteration is also getting lower with each update. The maximum update represents the maximum change in the perameters (beta) between each iteration. As with each iteration, the peramaters are being updated and as get closer to the actual value which means there is less change.

Question 2c (PSTAT 234)

Suppose we incorporate a regularization term $\lambda \|\beta\|_1$ with $\lambda > 0$, so that the objective function to be maximized is:

$$\ell(\beta) = \sum_{i=1}^{n} y_i x_i^T \beta - \log\big(1 + \exp\big(x_i^T \beta\big)\big) - \lambda \|\beta\|_1$$

With $\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j|$.

1. By using the library `cvxpy`, create the function `Update_Beta_reg` that takes the data and the value of λ as inputs to obtain the optimal value of β.
2. Run a loop that iterates over different values of λ (in between 0.01 and 1), and uses the function that you created (`Update_Beta_reg`) to obtain several solutions for β.
3. What value of λ would you choose based on the average absolute error?
4. How these resultes compare to part 2a?

In [5]:
```
# Fill-in ...
def Update_Beta_reg(X, y, lambda_val):



    ...
    return ...
```

Submission Checklist

1. Save file to confirm all changes are on disk
2. Run *Kernel > Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File > Save and export Notebook as/ > HTML*.
5. Open in Google Chrome and print to PDF.
6. Submit to Gradescope