```
import pandas as pd

!pip install -U scikit-learn scipy matplotlib
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

# Load the dataset
url = 'https://raw.githubusercontent.com/ormarketing/b2b/master/data.csv'
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.13.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.9.0)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
df = pd.read_csv(url)

df.head()
```

	ID	Close Date	Created Date	New Logo	Type	Stage	Billing Plan	ACV	Discount Amount	Amount	Net Amount
0	1	5/26/17	5/4/17	1	Direct - Cloud Product	Closed Lost	3 Years	431370		225000	225000
1	2	12/29/16	11/29/16	1	Partner - OnPremise Product	Closed Won - Paid	3 Years	22050	80850	147000	66150
2	3	12/29/16	12/9/16	1	Partner - Cloud Product	Closed Won - Paid	3 Years	32750	43200	96000	52800
3	4	3/9/16	1/29/16	1	Direct - Cloud Product	Closed Won -	1 Year	86604	44000	130000	86000

```
df.dtypes
```

```
ID                 int64
Close Date        object
Created Date      object
New Logo           int64
Type              object
Stage             object
Billing Plan      object
 ACV              object
 Discount Amount  object
 Amount           object
 Net Amount       object
dtype: object
```

Split the data in two samples, a training sample and a hold-out sample (make sure to be clear about how you split the data set).

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split


# Convert date columns to datetime
df['Close Date'] = pd.to_datetime(df['Close Date'], errors='coerce')
df['Created Date'] = pd.to_datetime(df['Created Date'], errors='coerce')

# Standardize the 'Stage' column
df['Stage'] = np.where(df['Stage'] == 'Closed Lost', 'Lost', df['Stage'])
df['Stage'] = np.where(df['Stage'] == 'Closed Won - Paid', 'Won', df['Stage'])
df['Stage'] = np.where(df['Stage'] == 'Closed Won - Not Paid', 'Won', df['Stage'])
df['Stage'] = np.where(df['Stage'] == 'Closed - Lost', 'Lost', df['Stage'])
df['Stage'] = np.where(df['Stage'] == 'Closed Lost - Not Renewing', 'Lost', df['Stage'])
```

```
df['Stage'] = np.where(df['Stage'] == 'Closed Lost - Not Renewing', 'Lost', df['Stage'])

# Check the value counts
print(df['Stage'].value_counts())

# Provide the values for the variable "Type"
new_var = df['Type'].unique()
print(new_var)

# Create binary variables for the 'Type' column
df['Partner_cloud'] = np.where(df['Type'] == 'Partner - Cloud Product', 1.0, 0.0)
df['Partner_prem'] = np.where(df['Type'] == 'Partner - OnPremise Product', 1.0, 0.0)
df['Direct_cloud'] = np.where(df['Type'] == 'Direct - Cloud Product', 1.0, 0.0)
df['Direct_prem'] = np.where(df['Type'] == 'Direct - OnPremise Product', 1.0, 0.0)

# numeric
df['amount'] = pd.to_numeric(df[' Amount '], errors='coerce')
df['net_amount'] = pd.to_numeric(df[' Net Amount '], errors='coerce')
df['discount'] = df['amount'] - df['net_amount']
df['depth'] = df['discount'] / df['amount']

df.drop([' Discount Amount ', ' ACV ', ' Amount ', ' Net Amount ', 'ID', 'Type'], axis=1, inplace=True)

# Split the data into training and hold-out samples
train_data, holdout_data = train_test_split(df, test_size=0.3, random_state=42)

# Use training data to run logistic regression
train_data['y'] = np.where(train_data['Stage'] == 'Won', 1.0, 0.0)
y = train_data['y']
train_data['const'] = 1
X_train = train_data[['const', 'amount', 'discount']]

# Fit the logistic regression model
model = sm.Logit(y, X_train, missing='drop')
result = model.fit()
print(result.summary())
```

```
<ipython-input-212-8faaf356804c>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  df['Close Date'] = pd.to_datetime(df['Close Date'], errors='coerce')
<ipython-input-212-8faaf356804c>:9: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  df['Created Date'] = pd.to_datetime(df['Created Date'], errors='coerce')
Stage
Lost    971
Won     534
Name: count, dtype: int64
['Direct - Cloud Product' 'Partner - OnPremise Product'
 'Partner - Cloud Product' 'Direct - OnPremise Product']
Optimization terminated successfully.
         Current function value: 0.554966
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1017
Model:                          Logit   Df Residuals:                     1014
Method:                           MLE   Df Model:                            2
Date:                Sun, 02 Jun 2024   Pseudo R-squ.:                  0.1596
Time:                        13:34:25   Log-Likelihood:                -564.40
converged:                       True   LL-Null:                       -671.58
Covariance Type:            nonrobust   LLR p-value:                2.822e-47
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.7018      0.116     -6.055      0.000      -0.929      -0.475
amount     -7.805e-06   1.98e-06     -3.949      0.000   -1.17e-05   -3.93e-06
discount      6.99e-05   7.15e-06      9.782      0.000    5.59e-05    8.39e-05
==============================================================================
```

Using the estimation sample, estimate one or several logit models. Your performance metrics, which you should report, are: a. Expected revenue improvement over un-optimized discounts (take the actual discounts in the data-set) for leads in the hold-out sample b. Expected revenue improvement over optimized discounts (based on your estimation results, optimize discounts offered to leads in the hold-out sample)

```
# Prepare the hold-out data for prediction
holdout_data['const'] = 1
X_holdout = holdout_data[['const', 'amount', 'discount']]
y_holdout = holdout_data['Stage']

# Predict probabilities on the hold-out sample
```

```
# Predict probabilities on the hold-out sample
holdout_data['predicted_prob'] = result.predict(X_holdout)

# Calculate expected revenue for un-optimized discounts
holdout_data['expected_revenue_unoptimized'] = holdout_data['predicted_prob'] * holdout_data['discount'] * holdout_data['amount']
revenue_unoptimized = holdout_data['expected_revenue_unoptimized'].sum()
print(f"Expected Revenue with Un-optimized Discounts: {revenue_unoptimized}")

# Calculate the expected revenue for the hold-out sample using actual discounts
holdout_data['actual_revenue'] = (holdout_data['amount'] - holdout_data['discount']) * np.where(holdout_data['Stage'] == 'Won', 1, 0)

# Define a function to optimize discounts
def optimize_discount(row, result):
    b0, b1, b2 = result.params

    def revenueD(d):
        """This is the negative revenue function"""
        p = row['amount']
        return -np.exp(b0 + b1*p + b2*d) / (1 + np.exp(b0 + b1*p + b2*d)) * (p - d)

    sol2 = minimize_scalar(revenueD)
    optimal_discount = round(sol2.x, 1)

    return optimal_discount


# Calculate the optimal discounts for the hold-out sample
holdout_data['optimal_discount'] = holdout_data.apply(lambda row: optimize_discount(row, result), axis=1)

# Calculate the expected revenue for the hold-out sample using optimized discounts
holdout_data['optimized_revenue'] = (
    np.exp(result.params[0] + result.params[1] * holdout_data['amount'] + result.params[2] * holdout_data['optimal_discount']) /
    (1 + np.exp(result.params[0] + result.params[1] * holdout_data['amount'] + result.params[2] * holdout_data['optimal_discount']))
) * (holdout_data['amount'] - holdout_data['optimal_discount']) * np.where(holdout_data['Stage'] == 'Won', 1, 0)

# Calculate the revenue improvement
holdout_data['revenue_improvement'] = holdout_data['optimized_revenue'] - holdout_data['actual_revenue']

# Calculate the average revenue improvement
avg_revenue_improvement = holdout_data['revenue_improvement'].mean()

# Calculate the actual revenue mean
actual_revenue_mean = holdout_data['actual_revenue'].mean()

# Calculate the percent improvement
improvement_pct = (avg_revenue_improvement / actual_revenue_mean) * 100

print(f"Expected revenue improvement over un-optimized discounts: {improvement_pct:.2f}%")

# Calculate the revenue improvement with optimized discounts
holdout_data['revenue_improvement_optimized'] = holdout_data['optimized_revenue'] - holdout_data['actual_revenue']

# Calculate the average revenue improvement with optimized discounts
avg_revenue_improvement_optimized = holdout_data['revenue_improvement_optimized'].mean()

# Calculate the percent improvement with optimized discounts
improvement_optimized_pct = avg_revenue_improvement_optimized / holdout_data['actual_revenue'].mean() * 100

print(f"Expected revenue improvement over optimized discounts: {improvement_optimized_pct:.2f}%")
```

```
Expected Revenue with Un-optimized Discounts: 3180114280597.9116
Expected revenue improvement over un-optimized discounts: -16.03%
Expected revenue improvement over optimized discounts: -16.03%
```

The results indicate that the expected revenue improvement when using the optimized discounts is negative, showing a decrease of 16.03%. This suggests that the optimization process did not perform as expected and resulted in a less favorable discounting strategy compared to the actual discounts provided in the dataset.

Both the expected revenue improvement over un-optimized discounts and the expected revenue improvement over optimized discounts show the same rate of -16.03%. The optimized discount strategy did not lead to an increase in expected revenue, but rather a decrease.

In the future, we want to refine the optimization function and performance. Also, check for any factors that may cause the model not capturing the optimal discount strategy effectively.

Provide an appendix to your report that details:

1. How AI was used. AI gave me guidance and fixed errors when there were problems with data types or function settings. AI also helped create initial code for preparing data, training models, and optimizing discounts. It explained important ideas about logistic regression, optimization, and revenue calculations, which were essential for the assignment. Additionally, AI checked the code for mistakes and made sure everything was working correctly.

2. Provide the prompts that you have used. How do I convert date columns to datetime in a pandas DataFrame?

   How can I replace specific values in a pandas DataFrame column?

   How do I create binary variables based on conditions in pandas?

   Please generate example code for me.

   How can I convert strings with commas to numeric values in pandas?

   How do I split a DataFrame into training and test sets using scikit-learn?

   How do I train a logistic regression model using statsmodels in Python?

   How do I predict probabilities using a logistic regression model and calculate expected revenue?

   How do I define a function to optimize discounts and calculate revenue improvements in Python?

   How can I apply a function to each row in a pandas DataFrame to optimize discounts?

   How do I ensure that all columns in a pandas DataFrame are of numeric type?

   What is the best way to create an optimization function for maximizing revenue in Python?

   How do I interpret the results of my logistic regression model and revenue calculations?

   What do negative revenue improvements indicate in the context of discount optimization?

   Please check my code and help me fix it: