Lab 8: Define and Solve an ML Problem of Your Choosing

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

In this lab assignment, you will follow the machine learning life cycle and implement a model to solve a machine learning problem of your choosing. You will select a data set and choose a predictive problem that the data set supports. You will then inspect the data with your problem in mind and begin to formulate a project plan. You will then implement the machine learning project plan.

You will complete the following tasks:

1. Build Your DataFrame
2. Define Your ML Problem
3. Perform exploratory data analysis to understand your data.
4. Define Your Project Plan
5. Implement Your Project Plan:
    - Prepare your data for your model.
    - Fit your model to the training data and evaluate your model.
    - Improve your model's performance.

Part 1: Build Your DataFrame

You will have the option to choose one of four data sets that you have worked with in this program:

- The "census" data set that contains Census information from 1994: `censusData.csv`
- Airbnb NYC "listings" data set: `airbnbListingsData.csv`
- World Happiness Report (WHR) data set: `WHR2018Chapter2OnlineData.csv`
- Book Review data set: `bookReviewsData.csv`

Note that these are variations of the data sets that you have worked with in this program. For example, some do not include some of the preprocessing necessary for specific models.

Load a Data Set and Save it as a Pandas DataFrame

The code cell below contains filenames (path + filename) for each of the four data sets available to you.

Task: In the code cell below, use the same method you have been using to load the data using `pd.read_csv()` and save it to DataFrame `df`.

You can load each file as a new DataFrame to inspect the data before choosing your data set.

In [2]:
```
# File names of the four data sets
adultDataSet_filename = os.path.join(os.getcwd(), "data", "censusData.csv")
airbnbDataSet_filename = os.path.join(os.getcwd(), "data", "airbnbListingsData.csv"
WHRDataSet_filename = os.path.join(os.getcwd(), "data", "WHR2018Chapter2OnlineData.
bookReviewDataSet_filename = os.path.join(os.getcwd(), "data", "bookReviewsData.csv


df = pd.read_csv(adultDataSet_filename, header=0) # YOUR CODE HERE

df.head(10)
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	rac
0	39.0	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Whi
1	50.0	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Whi
2	38.0	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Whi
3	53.0	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Blac
4	28.0	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Blac
5	37.0	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	Whi
6	49.0	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Blac
7	52.0	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	Whi
8	31.0	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	Whi
9	42.0	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Whi

Part 2: Define Your ML Problem

Next you will formulate your ML Problem. In the markdown cell below, answer the following questions:

1. List the data set you have chosen.
2. What will you be predicting? What is the label?
3. Is this a supervised or unsupervised learning problem? Is this a clustering, classification or regression problem? Is it a binary classificaiton or multi-class classifiction problem?
4. What are your features? (note: this list may change after your explore your data)

5. Explain why this is an important problem. In other words, how would a company create value with a model that predicts this label?

1. The adult census data set.
2. I'll be predicting whether an adult makes more or less than/equal to 50K. The label is the income_binary column.
3. This is a binary classification supervised learning problem.
4. My features are all of the other columns except for my label column. However, by simply inspecting the data we can remove a few columns from our features list. The ones that we can remove are: fnlwgt (because this is a value used to create the census which won't affect our label), education (because it's made redundant by education-num), and relationship (because we get the same information from marital-status).
5. Companies can create value with a model that predicts this label in many different ways. It can be used to figure out what customer demographic people fall into, which helps companies figure out who their target audience is (who is more likely to purchase their products), or who would be more likely to make a purchase if offered a discount. This can also be used by companies to figure out who should be contacted about loans or credit cards. An advising company or an educational company could use this to figure out who would be most likely to use/benefit from their services.

Part 3: Understand Your Data

The next step is to perform exploratory data analysis. Inspect and analyze your data set with your machine learning problem in mind. Consider the following as you inspect your data:

1. What data preparation techniques would you like to use? These data preparation techniques may include:
   - addressing missingness, such as replacing missing values with means
   - finding and replacing outliers
   - renaming features and labels
   - finding and replacing outliers
   - performing feature engineering techniques such as one-hot encoding on categorical features
   - selecting appropriate features and removing irrelevant features
   - performing specific data cleaning and preprocessing techniques for an NLP problem
   - addressing class imbalance in your data sample to promote fair AI
2. What machine learning model (or models) you would like to use that is suitable for your predictive problem and data?

- Are there other data preparation techniques that you will need to apply to build a balanced modeling data set for your problem and model? For example, will you need to scale your data?

3. How will you evaluate and improve the model's performance?

- Are there specific evaluation metrics and methods that are appropriate for your model?

Think of the different techniques you have used to inspect and analyze your data in this course. These include using Pandas to apply data filters, using the Pandas `describe()` method to get insight into key statistics for each column, using the Pandas `dtypes` property to inspect the data type of each column, and using Matplotlib and Seaborn to detect outliers and visualize relationships between features and labels. If you are working on a classification problem, use techniques you have learned to determine if there is class imbalance.

Task: Use the techniques you have learned in this course to inspect and analyze your data. You can import additional packages that you have used in this course that you will need to perform this task.

Note: You can add code cells if needed by going to the **Insert** menu and clicking on **Insert Cell Below** in the drop-drown menu.

In [3]:
```
# YOUR CODE HERE
df.drop(columns=['fnlwgt', 'education', 'relationship'], inplace=True)
print(df.shape)
print(df.columns.tolist())
```

```
(32561, 12)
['age', 'workclass', 'education-num', 'marital-status', 'occupation', 'race', 'sex_s
elfID', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income_
binary']
```

In [4]:
```
# Display the data types of the dataframe's columns
df.dtypes
```

Out[4]:
```
age                float64
workclass           object
education-num        int64
marital-status      object
occupation          object
race                object
sex_selfID          object
capital-gain         int64
capital-loss         int64
hours-per-week     float64
native-country      object
income_binary       object
dtype: object
```

In [5]:
```
# Inspecting the age, hours-per-week, and capital-gain/loss columns
# to see if there are outliers that need to be handled
print(df['age'].describe())
print(df['hours-per-week'].describe())
print(df['capital-gain'].describe())
print(df['capital-loss'].describe())
```

```
count    32399.000000
mean        38.589216
std         13.647862
min         17.000000
25%         28.000000
50%         37.000000
75%         48.000000
max         90.000000
Name: age, dtype: float64
count    32236.000000
mean        40.450428
std         12.353748
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
Name: hours-per-week, dtype: float64
count    32561.000000
mean       615.907773
std       2420.191974
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      14084.000000
Name: capital-gain, dtype: float64
count    32561.000000
mean        87.303830
std        402.960219
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       4356.000000
Name: capital-loss, dtype: float64
```

In [7]:
```
# Handling outliers by creating new columns that we got by winsorizing the age, hou
import scipy.stats as stats
df['age_values'] = stats.mstats.winsorize(df['age'], limits=[0.01, 0.01])
df['hours-per-week_values'] = stats.mstats.winsorize(df['hours-per-week'], limits=[
df['capital-gain_values'] = stats.mstats.winsorize(df['capital-gain'], limits=[0.01
df['capital-loss_values'] = stats.mstats.winsorize(df['capital-loss'], limits=[0.01
df.head()
```

Out[7]:

	age	workclass	education-num	marital-status	occupation	race	sex_selfID	capital-gain	capital-loss
0	39.0	State-gov	13	Never-married	Adm-clerical	White	Non-Female	2174	0
1	50.0	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	White	Non-Female	0	0
2	38.0	Private	9	Divorced	Handlers-cleaners	White	Non-Female	0	0
3	53.0	Private	7	Married-civ-spouse	Handlers-cleaners	Black	Non-Female	0	0
4	28.0	Private	13	Married-civ-spouse	Prof-specialty	Black	Female	0	0

In [8]:

```
# Checking to see if there are changes between the original columns and their winds
# If there are values otehr than zero, it means some changes happened
print((df['age'] - df['age_values']).unique())
print((df['hours-per-week'] - df['hours-per-week_values']).unique())
print((df['capital-gain'] - df['capital-gain_values']).unique())
print((df['capital-loss'] - df['capital-loss_values']).unique())
```

```
[ 0. nan  1. 12.  2.  3. 10.  4.  5.  6.  7.  8.  9.]
[ 0. nan -6. -7. -2. -3. -4. -1. -5.]
[0]
[   0   62  199  226  359  435   71  397  372  412   21   22  278  567
  194  225  464  258 1024  251  844  579   77  169  149  623  302 2376
  266  509 1790 1703  287  100  477 1920  221  487  183  774  492]
```

In [9]:

```
# Find which columns contain missing values
nan_count = np.sum(df.isnull(), axis = 0)
print(nan_count)
```

```
age                       162
workclass                1836
education-num               0
marital-status              0
occupation               1843
race                        0
sex_selfID                  0
capital-gain                0
capital-loss                0
hours-per-week            325
native-country            583
income_binary               0
age_values                  0
hours-per-week_values       0
capital-gain_values         0
capital-loss_values         0
dtype: int64
```

In [10]:

```
# Storing the True/False series that indicate whether a row has missing values in a
df['age_na'] = df['age'].isnull()
df['hours-per-week_na'] = df['hours-per-week'].isnull()
df.head()
```

Out[10]:

	age	workclass	education-num	marital-status	occupation	race	sex_selfID	capital-gain	capital-loss
0	39.0	State-gov	13	Never-married	Adm-clerical	White	Non-Female	2174	0
1	50.0	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	White	Non-Female	0	0
2	38.0	Private	9	Divorced	Handlers-cleaners	White	Non-Female	0	0
3	53.0	Private	7	Married-civ-spouse	Handlers-cleaners	Black	Non-Female	0	0
4	28.0	Private	13	Married-civ-spouse	Prof-specialty	Black	Female	0	0

In [11]:

```
# Filling the missing values of the age and hours-per-week columns with their mean
df['age'].fillna(value=df['age'].mean(), inplace=True)
df['hours-per-week'].fillna(value=df['hours-per-week'].mean(), inplace=True)
```

In [12]:

```
# Checking to see whether they still contain missing values
print(np.sum(df['age'].isnull(), axis = 0))
print(np.sum(df['hours-per-week'].isnull(), axis = 0))
```

```
0
0
```

In [13]:
```
# Finding all of the columns who have a data type of 'object' and adding them to a
# list called to_encode to be one-hot encoded later
to_encode = df.select_dtypes(include='object').columns.tolist()
print(to_encode)

# Removes income_binary columns from the list of columns to one-hot encode
to_encode.remove('income_binary')
print(to_encode)
```

```
['workclass', 'marital-status', 'occupation', 'race', 'sex_selfID', 'native-countr
y', 'income_binary']
['workclass', 'marital-status', 'occupation', 'race', 'sex_selfID', 'native-countr
y']
```

In [14]:
```
# Display the number of unique values each column in to_encode has
df[to_encode].nunique()
```

Out[14]:
```
workclass          8
marital-status     7
occupation        14
race               5
sex_selfID         2
native-country    41
dtype: int64
```

In [15]:
```
# One-Hot Encoding The Columns
from sklearn.preprocessing import OneHotEncoder  # Imports OneHotEncoder from sklea

# Create the encoder:
enc = OneHotEncoder(sparse_output=False)

# Apply the encoder:
# 'enc.fit_transform() fits the encoder to the data and transforms the data into on
# The results are saved to the 'df_enc' DataFrame
df_enc = pd.DataFrame(enc.fit_transform(df[to_encode]))

# enc.get_feature_names() reinstates the original column names.
df_enc.columns = enc.get_feature_names_out(['workclass', 'marital-status', 'occupat
df_enc.head(10)
```

Out[15]:

	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0
5	0.0	0.0	0.0	1.0	0.0
6	0.0	0.0	0.0	1.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	1.0	0.0
9	0.0	0.0	0.0	1.0	0.0

10 rows × 80 columns

In [16]:

```
# Removing the columns that were one-hot encoded from the df and joining the two Da
df.drop(columns=to_encode, inplace=True)
df = pd.concat([df, df_enc], axis=1)
df.columns
```

Out[16]:
```
Index(['age', 'education-num', 'capital-gain', 'capital-loss',
       'hours-per-week', 'income_binary', 'age_values',
       'hours-per-week_values', 'capital-gain_values', 'capital-loss_values',
       'age_na', 'hours-per-week_na', 'workclass_Federal-gov',
       'workclass_Local-gov', 'workclass_Never-worked', 'workclass_Private',
       'workclass_Self-emp-inc', 'workclass_Self-emp-not-inc',
       'workclass_State-gov', 'workclass_Without-pay', 'workclass_nan',
       'marital-status_Divorced', 'marital-status_Married-AF-spouse',
       'marital-status_Married-civ-spouse',
       'marital-status_Married-spouse-absent', 'marital-status_Never-married',
       'marital-status_Separated', 'marital-status_Widowed',
       'occupation_Adm-clerical', 'occupation_Armed-Forces',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Farming-fishing', 'occupation_Handlers-cleaners',
       'occupation_Machine-op-inspct', 'occupation_Other-service',
       'occupation_Priv-house-serv', 'occupation_Prof-specialty',
       'occupation_Protective-serv', 'occupation_Sales',
       'occupation_Tech-support', 'occupation_Transport-moving',
       'occupation_nan', 'race_Amer-Indian-Inuit', 'race_Asian-Pac-Islander',
       'race_Black', 'race_Other', 'race_White', 'sex_selfID_Female',
       'sex_selfID_Non-Female', 'native-country_Cambodia',
       'native-country_Canada', 'native-country_China',
       'native-country_Columbia', 'native-country_Cuba',
       'native-country_Dominican-Republic', 'native-country_Ecuador',
       'native-country_El-Salvador', 'native-country_England',
       'native-country_France', 'native-country_Germany',
       'native-country_Greece', 'native-country_Guatemala',
       'native-country_Haiti', 'native-country_Holand-Netherlands',
       'native-country_Honduras', 'native-country_Hong',
       'native-country_Hungary', 'native-country_India', 'native-country_Iran',
       'native-country_Ireland', 'native-country_Italy',
       'native-country_Jamaica', 'native-country_Japan', 'native-country_Laos',
       'native-country_Mexico', 'native-country_Nicaragua',
       'native-country_Outlying-US(Guam-USVI-etc)', 'native-country_Peru',
       'native-country_Philippines', 'native-country_Poland',
       'native-country_Portugal', 'native-country_Puerto-Rico',
       'native-country_Scotland', 'native-country_South',
       'native-country_Taiwan', 'native-country_Thailand',
       'native-country_Trinadad&Tobago', 'native-country_United-States',
       'native-country_Vietnam', 'native-country_Yugoslavia',
       'native-country_nan'],
      dtype='object')
```

In [17]:
```
# Converting the income_binary column from values like >50K or <=50K to 1s and 0s (
df['income_binary'] = df['income_binary'].str.strip()
df['income_binary'] = df['income_binary'].map({'>50K': 1, '<=50K': 0})
df['income_binary'].value_counts()
```

Out[17]:
```
0    24720
1     7841
Name: income_binary, dtype: int64
```

In [19]:
```
# Looking at the correlations of features with my label using a correlation matrix
corr_matrix = round(df.corr(),5)
corrs = corr_matrix['income_binary']
```

```
corrs_sorted = corrs.sort_values(ascending=False)
print(corrs_sorted.to_string())
```

```
income_binary                              1.00000
marital-status_Married-civ-spouse          0.44470
capital-gain                               0.34756
capital-gain_values                        0.34756
education-num                              0.33515
age                                        0.23310
age_values                                 0.22931
hours-per-week                             0.22840
sex_selfID_Non-Female                      0.21598
occupation_Exec-managerial                 0.21486
hours-per-week_values                      0.20630
occupation_Prof-specialty                  0.18587
capital-loss_values                        0.15141
capital-loss                               0.15053
workclass_Self-emp-inc                     0.13947
race_White                                 0.08522
workclass_Federal-gov                      0.05937
native-country_United-States               0.03447
workclass_Local-gov                        0.03309
workclass_Self-emp-not-inc                 0.03002
occupation_Protective-serv                 0.02812
occupation_Tech-support                    0.02570
occupation_Sales                           0.02369
native-country_India                       0.02066
native-country_Iran                        0.01512
native-country_Japan                       0.01494
workclass_State-gov                        0.01484
native-country_Taiwan                      0.01402
native-country_Philippines                 0.01231
native-country_Germany                     0.01222
native-country_France                      0.01208
marital-status_Married-AF-spouse           0.01206
native-country_Canada                      0.01164
native-country_England                     0.01139
native-country_Italy                       0.01127
race_Asian-Pac-Islander                    0.01054
native-country_Cambodia                    0.00721
native-country_Yugoslavia                  0.00696
native-country_Hong                        0.00343
native-country_nan                         0.00304
native-country_China                       0.00291
native-country_Cuba                        0.00283
native-country_Greece                      0.00245
native-country_Scotland                    0.00041
native-country_Hungary                    -0.00047
hours-per-week_na                         -0.00164
native-country_Ireland                    -0.00206
native-country_Holand-Netherlands         -0.00312
native-country_Thailand                   -0.00408
age_na                                    -0.00409
native-country_Poland                     -0.00410
native-country_South                      -0.00474
occupation_Armed-Forces                   -0.00504
native-country_Ecuador                    -0.00672
native-country_Laos                       -0.00713
native-country_Trinadad&Tobago            -0.00766
```

```
native-country_Honduras                     -0.00766
workclass_Never-worked                      -0.00826
native-country_Portugal                     -0.01047
native-country_Outlying-US(Guam-USVI-etc)   -0.01168
workclass_Without-pay                       -0.01168
occupation_Craft-repair                     -0.01258
native-country_Peru                         -0.01273
native-country_Haiti                        -0.01290
native-country_Jamaica                      -0.01371
native-country_Nicaragua                    -0.01376
native-country_Vietnam                      -0.01765
native-country_Puerto-Rico                  -0.01879
native-country_Guatemala                    -0.02013
native-country_Columbia                     -0.02062
native-country_El-Salvador                  -0.02084
occupation_Transport-moving                 -0.02148
native-country_Dominican-Republic           -0.02304
race_Amer-Indian-Inuit                      -0.02872
race_Other                                  -0.03183
occupation_Priv-house-serv                  -0.03712
marital-status_Married-spouse-absent        -0.04253
occupation_Farming-fishing                  -0.05192
native-country_Mexico                       -0.06290
marital-status_Widowed                      -0.06438
occupation_Machine-op-inspct                -0.06940
marital-status_Separated                    -0.07439
workclass_nan                               -0.07820
workclass_Private                           -0.07853
occupation_nan                              -0.07858
occupation_Handlers-cleaners                -0.08727
race_Black                                  -0.08909
occupation_Adm-clerical                     -0.08999
marital-status_Divorced                     -0.12699
occupation_Other-service                    -0.15635
sex_selfID_Female                           -0.21598
marital-status_Never-married                -0.31844
```

Part 4: Define Your Project Plan

Now that you understand your data, in the markdown cell below, define your plan to implement the remaining phases of the machine learning life cycle (data preparation, modeling, evaluation) to solve your ML problem. Answer the following questions:

- Do you have a new feature list? If so, what are the features that you chose to keep and remove after inspecting the data?
- Explain different data preparation techniques that you will use to prepare your data for modeling.
- What is your model (or models)?
- Describe your plan to train your model, analyze its performance and then improve the model. That is, describe your model building, validation and selection plan to produce a model that generalizes well to new data.

- As mentioned above, I decided to remove the 'fnlwgt', 'education', and 'relationship' features before cleaning and prepping the data because they were either irrelevant or redundant. I chose to keep all of the other features, and have new features that were added as a result of filling in missing values and one-hot encoding.

- I performed feature engineering (specifically one-hot encoding columns with an object data type, which converts them to a numeric input the model can process, in addition to converting my income_binary column into a binary indicator, a column of 0s and 1s where 0 indicates the adult makes <=50K and 1 indicates the adult makes >50K), data exploration (using the pandas built-in describe() method to look at the data distribution of numerical columns to figure out whether they have outliers that need to be handled and using the pandas corr() method to look at the correlation between columns/features in the DataFrame and my label), and data cleaning (removing outliers by winsorizing the age, hours-per-week, and capital-gain/loss columns and then handling missing values in those columns by replacing missing values with their mean).

- My model is a logistic regression model.

- In order to train my model, I plan on splitting the data and creating training/testing data sets using my DataFrame and scikit-learn. Then I'll train and fit a Logistic Regression model (using the scikit-learn LogisticRegression class) on the training data and test the trained model on the test data. After that, I'll train different Logistic Regression classifiers with different hyperparameter values (specifically the C inverse of regularization strength hyperparameter) and use the accuracy and log loss results, that I'll plot using seaborn, to analyze performance and determine which value of C yields the best results in terms of accuracy and should be used for my model. Once I've looked at those results I'll also perform a grid search to try and identify the optimal value of C and evaluate both model's predictions using a confusion matrix and plotting and comparing the precision-recall curves (ROC) and AUC. I'll then test to see whether my model can be improved by using the SelectKBest feature selection method and looking at the AUC values produced by a model trained with only the best features.

Part 5: Implement Your Project Plan

Task: In the code cell below, import additional packages that you have used in this course that you will need to implement your project plan.

In [20]:
```
# YOUR CODE HERE
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
```

Task: Use the rest of this notebook to carry out your project plan.

You will:

1. Prepare your data for your model.
2. Fit your model to the training data and evaluate your model.
3. Improve your model's performance by performing model selection and/or feature selection techniques to find best model for your problem.

Add code cells below and populate the notebook with commentary, code, analyses, results, and figures as you see fit.

In [22]:
```
# Creates Labeled Examples from the Data Set
y = df['income_binary']
X = df.drop(columns = 'income_binary', axis=1)
```

In [23]:
```
# Prints out the number of examples and features so I can understand the data
print("Number of examples: " + str(X.shape[0]))
print("\nNumber of Features:" + str(X.shape[1]))
```

```
Number of examples: 32561

Number of Features:91
```

In [24]:
```
# Creates training and testing data sets out of the labeled examples
# Using 0.20 for test_size because this is a large dataset and I plan on doing cros
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_st
```

In [25]:
```
# Prints out the dimensions of the training and test data sets
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(26048, 91)
(6513, 91)
(26048,)
(6513,)
```

In [30]:
```
# Recieved an error that the LR model didn't fully converge, scaling the data to tr
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [31]:
```
# Creates a function that takes the training/testing data sets and a hyperparameter
def train_test_LR(X_train, y_train, X_test, y_test, c=1):

    model = LogisticRegression(class_weight='balanced', C=c, max_iter=1000) # Added
    model.fit(X_train, y_train)
    probability_predictions = model.predict_proba(X_test)
```

```
    class_label_predictions = model.predict(X_test)

    l_loss = log_loss(y_test, probability_predictions)
    acc_score = accuracy_score(y_test, class_label_predictions)

    return l_loss, acc_score
```

In [32]:
```
# Trains a Logistic Regression classifier with the default value for C (c=1)
train_test_LR(X_train, y_train, X_test, y_test, c=1)
```

Out[32]:
```
(0.3925212120710344, 0.8074619990787656)
```

In [37]:
```
# Creates a list of twenty values of C where each element has a value of 10^i for i
cs = [10**i for i in range(-10,10)]
cs
```

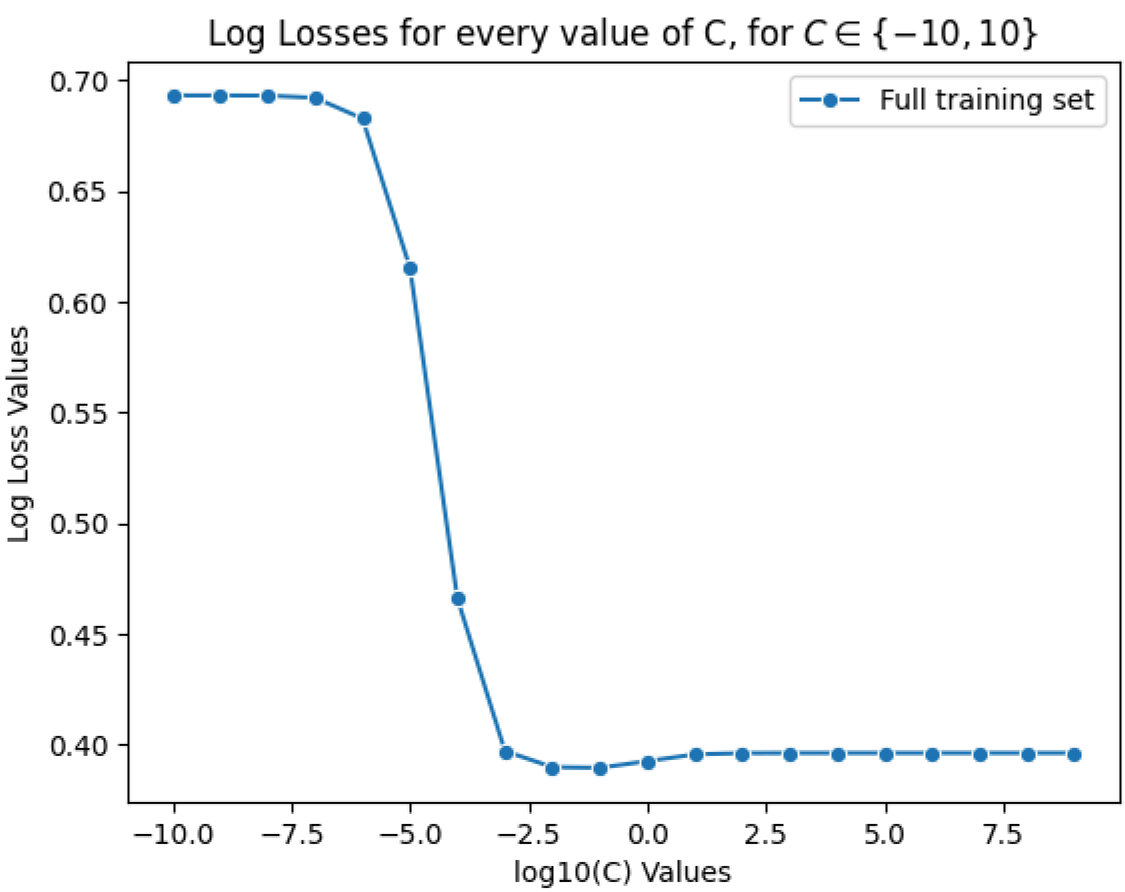
Out[37]:
```
[1e-10,
 1e-09,
 1e-08,
 1e-07,
 1e-06,
 1e-05,
 0.0001,
 0.001,
 0.01,
 0.1,
 1,
 10,
 100,
 1000,
 10000,
 100000,
 1000000,
 10000000,
 100000000,
 1000000000]
```

In [34]:
```
# Loops over every value in cs and trains a different Logistic Regression model for
# Prints out the accuracy and log scores for each model and saves them to a list to
log_losses = []
acc_scores = []
for c in cs:
    log_reg_model = train_test_LR(X_train, y_train, X_test, y_test, c=c)
    log_losses.append(log_reg_model[0])
    acc_scores.append(log_reg_model[1])
print(log_losses)
print(acc_scores)
```

```
[0.6931463882137531, 0.6931392574180235, 0.6930679811267422, 0.6920644450732234, 0.6
826975061234828, 0.6153258714143267, 0.4659386780343021, 0.3971328402009288, 0.38982
30784018512, 0.38953710411345194, 0.3925212120710344, 0.39559077972784124, 0.3961446
238382511, 0.39620477649843333, 0.39621084417438035, 0.3962114514711414, 0.396211512
20610895, 0.3962115182796624, 0.39621151888702016, 0.3962115189477445]
[0.7194841087056656, 0.7194841087056656, 0.7194841087056656, 0.7980961154613849, 0.7
985567326884692, 0.8002456625211116, 0.8019345923537541, 0.8053124520190389, 0.80776
90772301551, 0.8077690772301551, 0.8074619990787656, 0.8086903116843237, 0.808690311
6843237, 0.8086903116843237, 0.8086903116843237, 0.8086903116843237, 0.8086903116843
237, 0.8086903116843237, 0.8086903116843237, 0.8086903116843237]
```

In [35]:
```python
# Reformats the values in cs so that it's easy to read in our plot
cs_log10 = np.log10(cs)
```

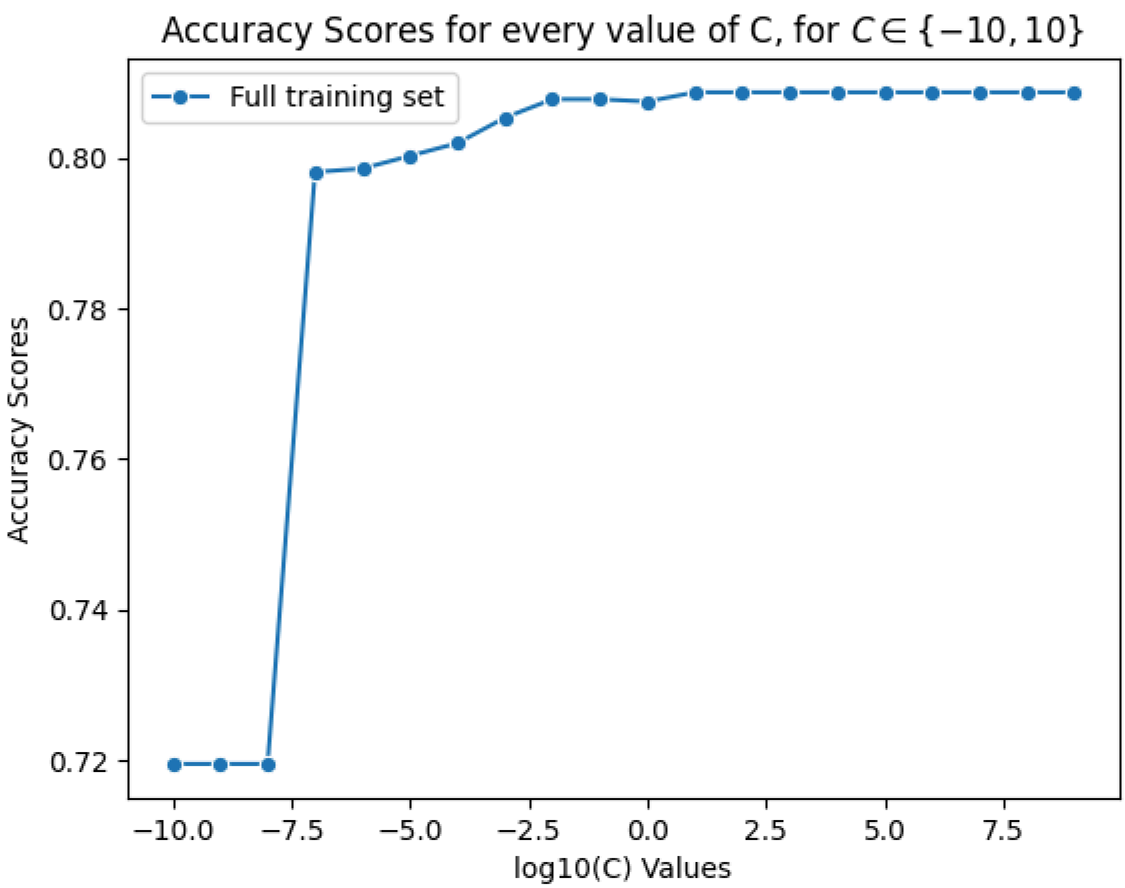
In [36]:
```python
# Creates and plots a seaborn lineplot demonstrating log loss for every hyperparame
# C is on the x-axis, log loss is on the y-axis
fig = plt.figure()
ax = fig.add_subplot(111)

p = sns.lineplot(x=cs_log10, y=log_losses, marker='o', label = 'Full training set')

plt.title('Log Losses for every value of C, for $C\in\{-10, 10\}$')
ax.set_xlabel('log10(C) Values')
ax.set_ylabel('Log Loss Values')
plt.show()
```



Log Loss Analysis

The value of C that yields the best results in terms of loss is 0.1. It's the 10th point on the lineplot and it has the lowest log loss value, therefore, it's the best.

In [38]:
```
# Creates and plots a seaborn lineplot demonstrating accuracy for every hyperparame
# C is on the x-axis, accuracy is on the y-axis
fig = plt.figure()
ax = fig.add_subplot(111)

p = sns.lineplot(x=cs_log10, y=acc_scores, marker='o', label = 'Full training set')

plt.title('Accuracy Scores for every value of C, for $C\in\{-10, 10\}$')
ax.set_xlabel('log10(C) Values')
ax.set_ylabel('Accuracy Scores')
plt.show()
```



Accuracy Analysis

The value of C that yields the best results in terms of accuracy is 10. It's the 12th point on the lineplot and it's the highest point the plot gets to before plateauing, and since larger C values mean weaker regularization (which means a more complex model with a higher risk of overfitting) and accuracy isn't increasing the higher we go with C values after 10, performance isn't improving either. Therefore, 10 is the C value that yields the best results.

Start of GridSearchCV Process

In [47]:
```
# Imports necessary inputs
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall_curv
```

In [39]:
```
# Trains a new model
model_default = LogisticRegression(C=1.0, max_iter=1000)
model_default.fit(X_train, y_train)
```

Out[39]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [40]:
```
# Tests the new model on the X_test test set
proba_predictions_default = model_default.predict_proba(X_test)[:, 1].tolist() # Ma

class_label_predictions_default = model_default.predict(X_test) # Make predictions
```

In [43]:
```
# Uses a confusion matrix to evaluate accuracy
c_m = confusion_matrix(y_test, class_label_predictions_default, labels=[True, False
```

In [44]:
```
# Creates a param_grid dictionary that contains 10 possible hyperparameter values f
cs_1 = [10**i for i in range(-5,5)]
param_grid = {'C': cs_1}
param_grid
```

Out[44]:
```
{'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
```

In [45]:
```
# Uses GridSearchCV to search over the different hyperparameter C values
print('Running Grid Search...')

model = LogisticRegression(max_iter=1000)

grid = GridSearchCV(model, param_grid, cv=5) # Runs a grid search with 5-fold cross

grid_search = grid.fit(X_train, y_train)

print('Done')
```

```
Running Grid Search...
Done
```

In [46]:
```
# Retrieves the hyperparameter C value that had the best score
best_C = grid_search.best_params_['C']
best_C
```

Out[46]:
```
1
```

Precision-Recall Curve Comparison

In [49]:
```
# Creates a LogisticRegression model with the best hyperparameter C value found usi
model_log_loss = LogisticRegression(C=0.1, max_iter=1000)
```

```
model_accuracy = LogisticRegression(C=10, max_iter=1000)
```

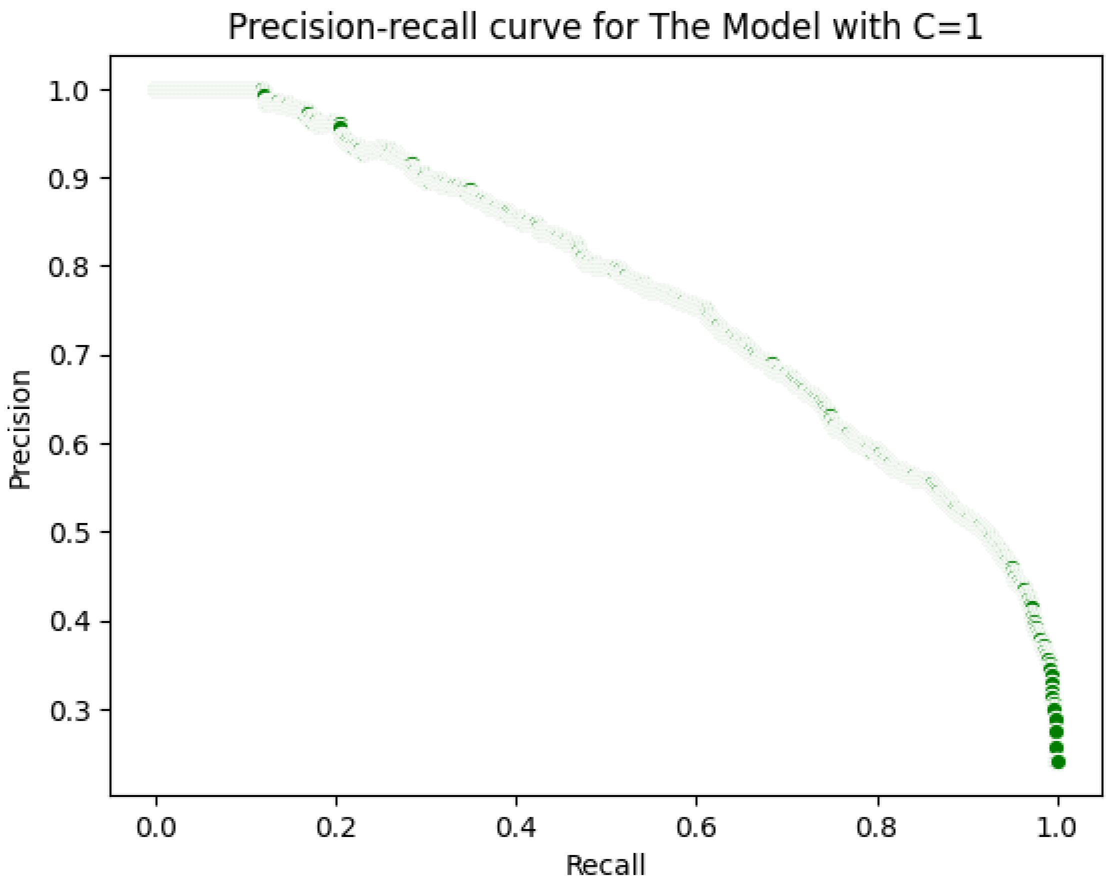
In [51]:
```
# Fits the models to the training data
model_log_loss.fit(X_train, y_train)
model_accuracy.fit(X_train, y_train)
```

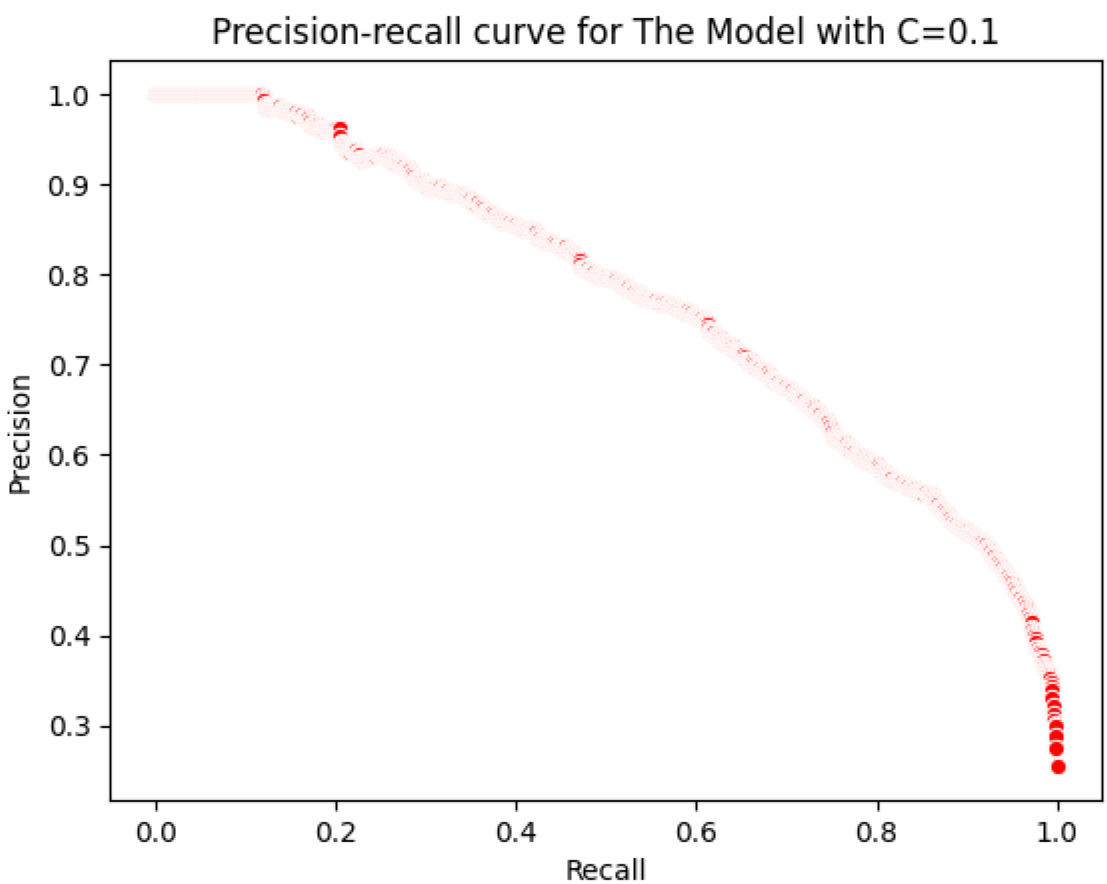
Out[51]:
```
LogisticRegression

LogisticRegression(C=10, max_iter=1000)
```

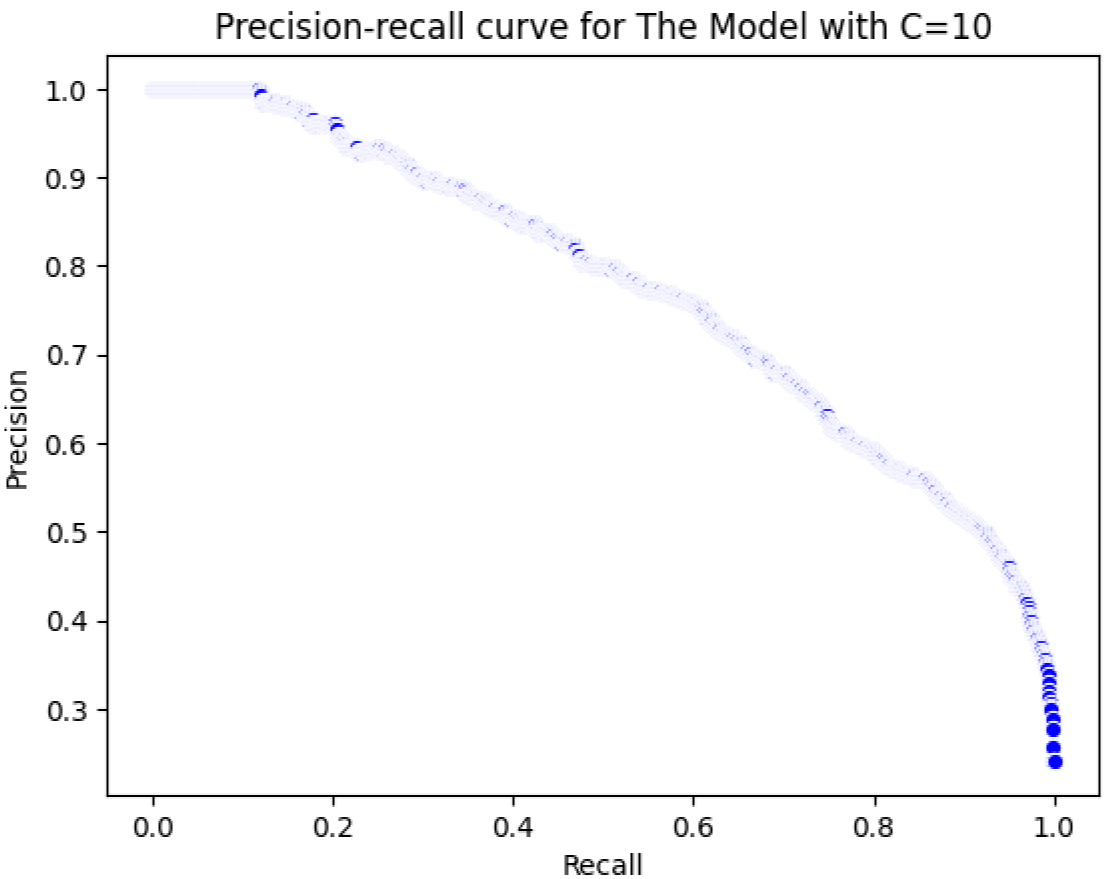
In [52]:
```
# Creates predictions on the test data using the models created with the best C val
proba_predictions_log_loss = model_log_loss.predict_proba(X_test)[:, 1].tolist()
proba_predictions_accuracy = model_accuracy.predict_proba(X_test)[:, 1].tolist()
```

In [53]:
```
# Compute precision-recall pairs for all the models
precision_default, recall_default, thresholds_default = precision_recall_curve(y_te
precision_log_loss, recall_log_loss, thresholds_log_loss = precision_recall_curve(y
precision_accuracy, recall_accuracy, thresholds_accuracy = precision_recall_curve(y
```

In [54]:
```
# Plots Seaborn lineplots to visualize the precision-recall curve for all models
# Recall is on the x-axis, Precision is on the y-axis

# default plot
fig = plt.figure()
ax = fig.add_subplot(111)

sns.lineplot(x=recall_default, y=precision_default, marker = 'o', color='green')

plt.title("Precision-recall curve for The Model with C=1")
plt.xlabel("Recall")
plt.ylabel("Precision")

# log loss plot
fig1 = plt.figure()
ax1 = fig1.add_subplot(111)

sns.lineplot(x=recall_log_loss, y=precision_log_loss, marker = 'o', color='red')

plt.title("Precision-recall curve for The Model with C=0.1")
plt.xlabel("Recall")
plt.ylabel("Precision")

# accuracy plot
fig1 = plt.figure()
ax1 = fig1.add_subplot(111)

sns.lineplot(x=recall_accuracy, y=precision_accuracy, marker = 'o', color='blue')

plt.title("Precision-recall curve for The Model with C=10")
plt.xlabel("Recall")
plt.ylabel("Precision")

plt.show()
```


Precision-recall curve for The Model with C=1
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0


Precision-recall curve for The Model with C=0.1
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0

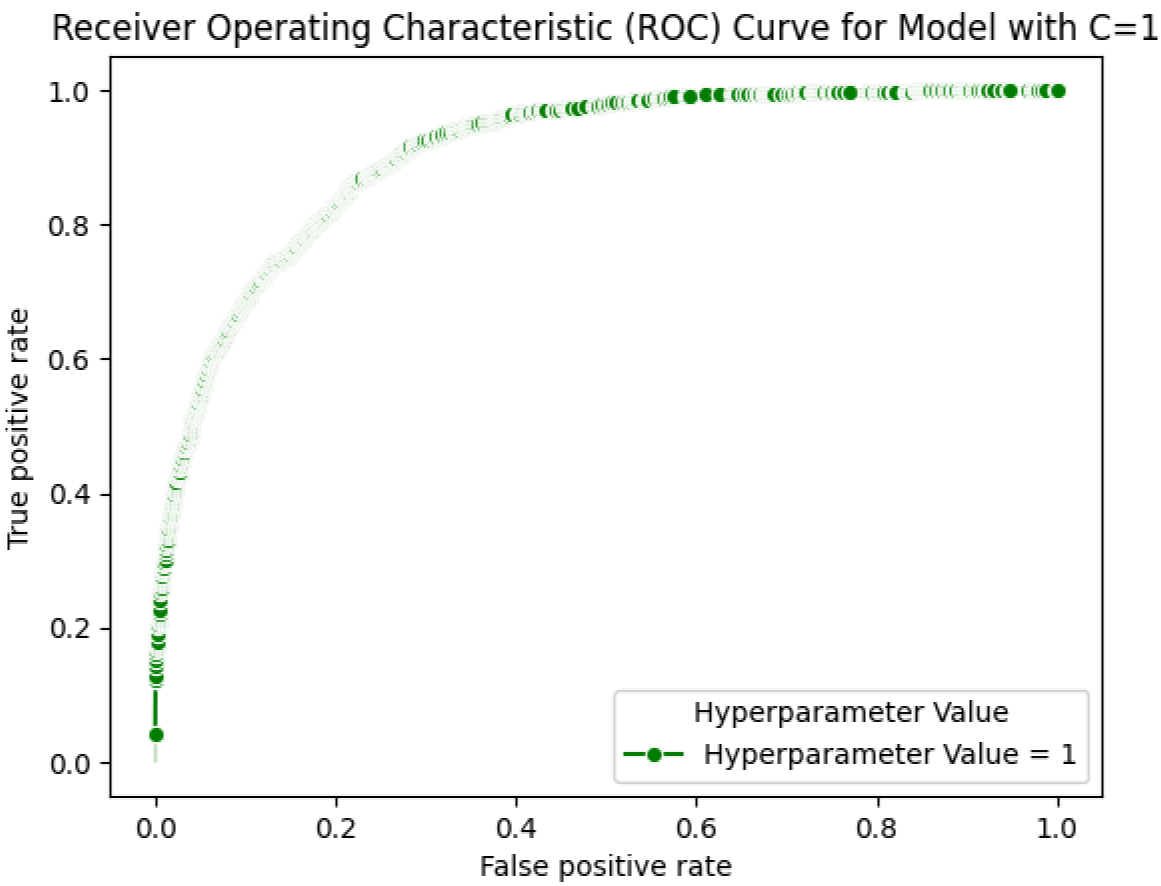


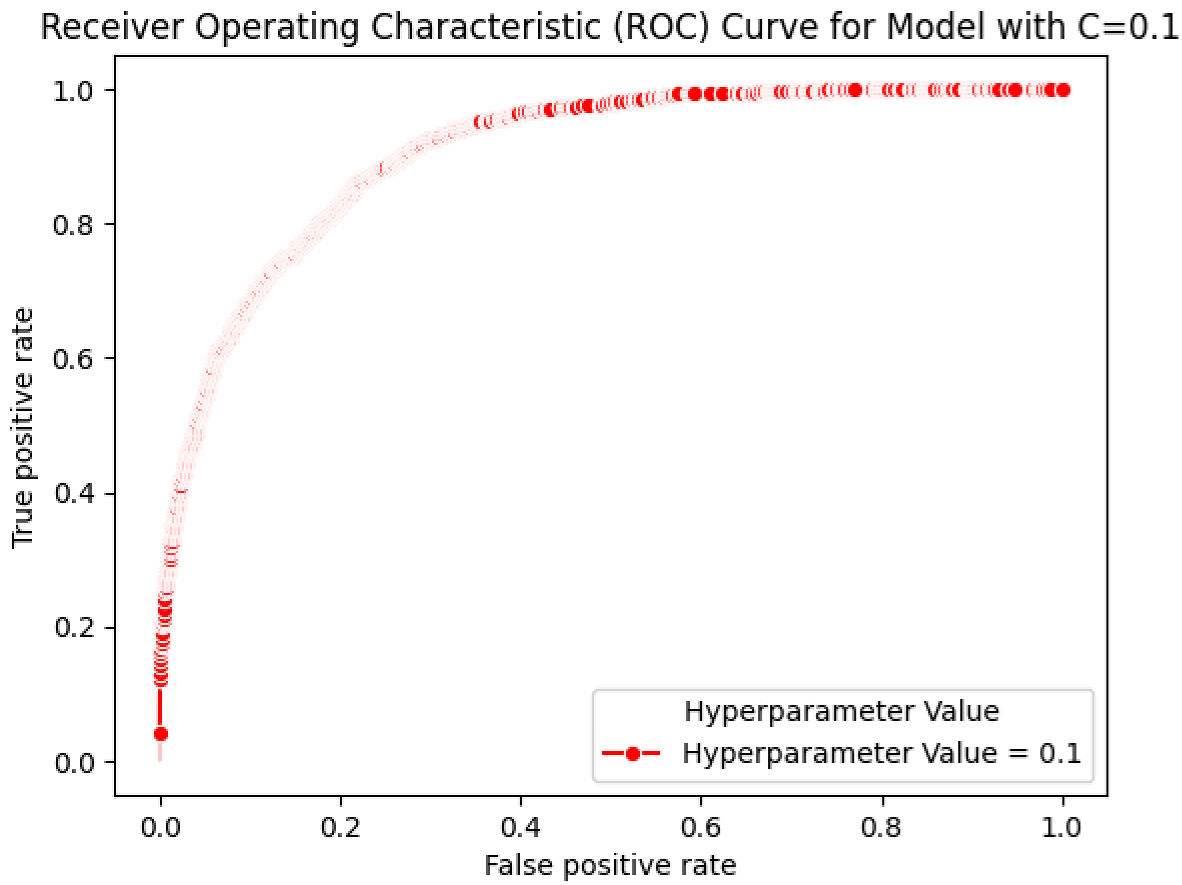
ROC Curve and AUC Comparison

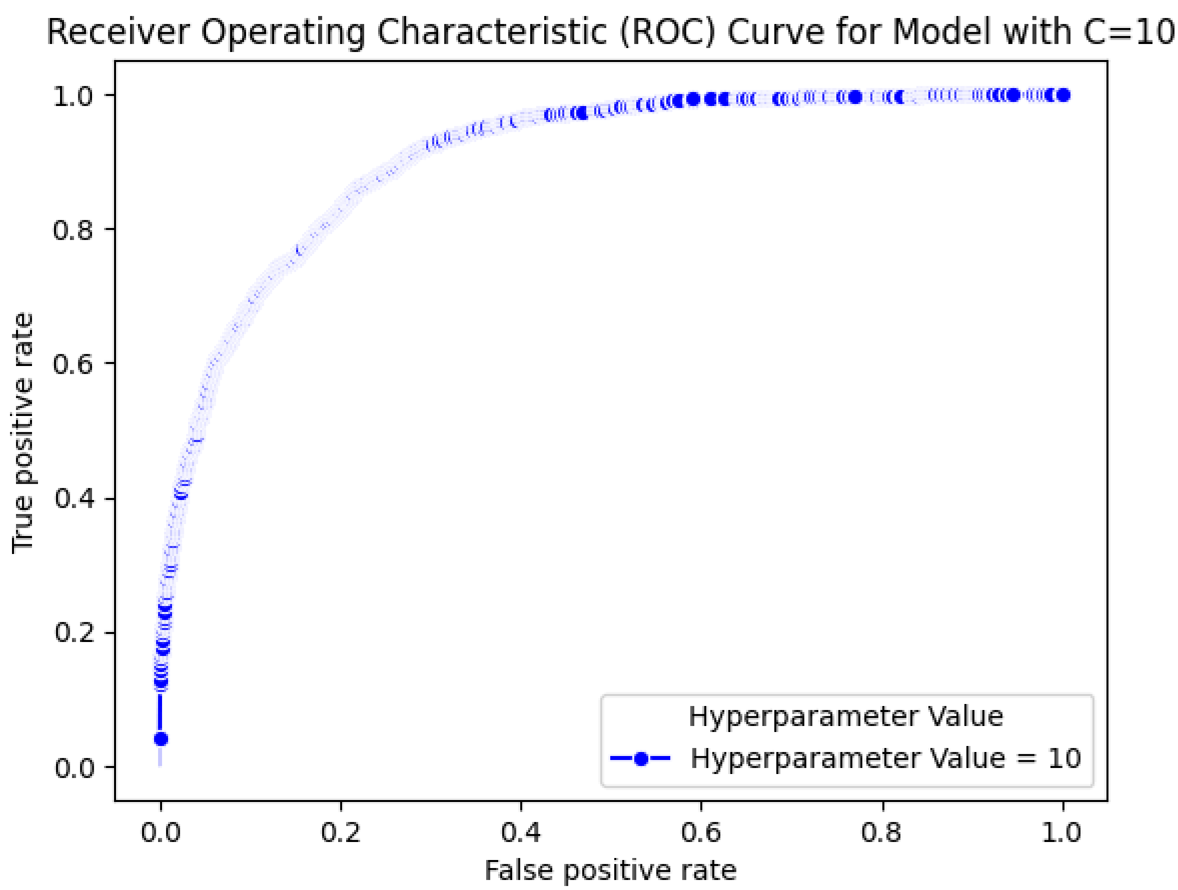
In [55]:
```
# Records the true positive and false positive rates for all models
fpr_default, tpr_default, thresholds_default = roc_curve(y_test, proba_predictions_
fpr_log_loss, tpr_log_loss, thresholds_log_loss = roc_curve(y_test, proba_predictio
fpr_accuracy, tpr_accuracy, thresholds_accuracy = roc_curve(y_test, proba_predictio
```

In [57]:
```
# Plots the ROC Curves for all models

fig = plt.figure()
ax = fig.add_subplot(111)
sns.lineplot(x=fpr_default, y=tpr_default, marker = 'o', color='green', label='Hype
plt.title("Receiver Operating Characteristic (ROC) Curve for Model with C=1")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.legend(title='Hyperparameter Value', loc='lower right')

fig1 = plt.figure()
ax1 = fig1.add_subplot(111)
sns.lineplot(x=fpr_log_loss, y=tpr_log_loss, marker = 'o', color='red', label='Hype
plt.title("Receiver Operating Characteristic (ROC) Curve for Model with C=0.1")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.legend(title='Hyperparameter Value', loc='lower right')

fig1 = plt.figure()
ax1 = fig1.add_subplot(111)
```

```
sns.lineplot(x=fpr_accuracy, y=tpr_accuracy, marker = 'o', color='blue', label='Hyp
plt.title("Receiver Operating Characteristic (ROC) Curve for Model with C=10")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.legend(title='Hyperparameter Value', loc='lower right')

plt.show()
```




Receiver Operating Characteristic (ROC) Curve for Model with C=0.1
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
Hyperparameter Value
Hyperparameter Value = 0.1


Receiver Operating Characteristic (ROC) Curve for Model with C=10
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
Hyperparameter Value
Hyperparameter Value = 10

In [58]:
```
# Computes the area under the receiver operating characteristic (ROC) curve for bot
auc_default = auc(fpr_default, tpr_default)
auc_log_loss = auc(fpr_log_loss, tpr_log_loss)
auc_accuracy = auc(fpr_accuracy, tpr_accuracy)

print(auc_default)
print(auc_log_loss)
print(auc_accuracy)
```

```
0.9048565524308586
0.905312896306255
0.9050023557802656
```

The model trained using the best C value from the log_loss plot resulted in the highest AUC value, which means that it's the best model.

In [62]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

selector = SelectKBest(f_classif, k=10)
selector.fit(X, y)

filter = selector.get_support()
top_10_features = X.columns[filter]

print("Best 10 features:")
print(top_10_features)

X_reduced = X[top_10_features]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X[top_10_features])

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran

model_log_loss_new = LogisticRegression(C=0.1, max_iter=2000)
model_log_loss_new.fit(X_train, y_train)

proba_predictions_log_loss_new = model_log_loss_new.predict_proba(X_test)[:, 1].tol

fpr, tpr, thresholds = roc_curve(y_test, proba_predictions_log_loss_new)
auc_result = auc(fpr, tpr)
print("AUC:", auc_result)
```

```
Best 10 features:
Index(['age', 'education-num', 'capital-gain', 'hours-per-week', 'age_values',
       'capital-gain_values', 'marital-status_Married-civ-spouse',
       'marital-status_Never-married', 'sex_selfID_Female',
       'sex_selfID_Non-Female'],
      dtype='object')
AUC: 0.8927434754933163
```

Changing the model to be trained off of only the best top 10 features ended up decreasing the AUC value, which means the model was less effective at distinguishing between whether an adult made more or less than 50K.

In [ ]:

```

```