```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install dmba
```

```
Collecting dmba
  Downloading dmba-0.2.4-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from dmba) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from dmba) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from dmba) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from dmba) (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from dmba) (1.6.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from dmba) (1.13.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->dmba) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->dmba) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->dmba) (2025.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->dmba) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->dmba) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->dmba) (1.17.0
Downloading dmba-0.2.4-py3-none-any.whl (11.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 99.7 MB/s eta 0:00:00
Installing collected packages: dmba
Successfully installed dmba-0.2.4
```

```
%matplotlib inline
from pathlib import Path
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import statsmodels.formula.api as smf
# pip install mord
# from mord import LogisticIT
import matplotlib.pylab as plt
import seaborn as sns
from dmba import classificationSummary, gainsChart, liftChart
from dmba.metric import AIC_score
import math
```

```
Colab environment detected.
```

Read the article “Should This Loan be Approved or Denied?”: A Large Dataset with Class Assignment Guidelines

The U.S. SBA was founded in 1953 on the principle of promoting and assisting small enterprises in the U.S. credit market (SBA Overview and History, US Small Business Administration (2015)). Small businesses have been a primary source of job creation in the United States; therefore, fostering small business formation and growth has social benefits by creating job opportunities and reducing unemployment. One way SBA assists these small business enterprises is through a loan guarantee program which is designed to encourage banks to grant loans to small businesses. SBA acts much like an insurance provider to reduce the risk for a bank by taking on some of the risk through guaranteeing a portion of the loan. In the case that a loan goes into default, SBA then covers the amount they guaranteed.

There have been many success stories of start-ups receiving SBA loan guarantees such as FedEx and Apple Computer. However, there have also been stories of small businesses and/or start-ups that have defaulted on their SBA-guaranteed loans. The rate of default on these loans has been a source of controversy for decades. Conservative economists believe that credit markets perform efficiently without government participation. Supporters of SBA-guaranteed loans argue that the social benefits of job creation by those small businesses receiving government-guaranteed loans far outweigh the costs incurred from defaulted loans.

Since SBA loans only guarantee a portion of the entire loan balance, banks will incur some losses if a small business defaults on its SBA-guaranteed loan. Therefore, banks are still faced with a difficult choice as to whether they should grant such a loan because of the high risk of default. One way to inform their decision making is through analyzing relevant historical data such as the SBA case data SBAcase.11.13.17.csv (download the zip data file at the journal site to obtain this data file) with the following Data Dictionary:

```
data_dictionary = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/SBA_DataDictionary.csv')
data_dictionary
```

	Variable Name	Data Type	Description of variable
0	LoanNr_ChkDgt	Text	Identifier – Primary Key
1	Name	Text	Borrower Name
2	City	Text	Borrower City
3	State	Text	Borrower State
4	Zip	Text	Borrower Zip Code
5	Bank	Text	Bank Name
6	BankState	Text	Bank State
7	NAICS	Text	North American Industry Classification System ...
8	ApprovalDate	Date/Time	Date SBA Commitment Issued
9	ApprovalFY	Text	Fiscal Year of Commitment
10	Term	Number	Loan term in months
11	NoEmp	Number	Number of Business Employees
12	NewExist	Text	1 = Existing Business, 2 = New Business
13	CreateJob	Number	Number of jobs created
14	RetainedJob	Number	Number of jobs retained
15	FranchiseCode	Text	Franchise Code 00000 or 00001 = No Franchise
16	UrbanRural	Text	1= Urban, 2= Rural, 0 = Undefined
17	RevLineCr	Text	Revolving Line of Credit : Y = Yes
18	LowDoc	Text	LowDoc Loan Program: Y = Yes, N = No
19	ChgOffDate	Date/Time	The date when a loan is declared to be in default
20	DisbursementDate	Date/Time	Disbursement Date
21	DisbursementGross	Currency	Amount Disbursed
22	BalanceGross	Currency	Gross amount outstanding
23	MIS_Status	Text	Loan Status
24	ChgOffPrinGr	Currency	Charged-off Amount
25	GrAppv	Currency	Gross Amount of Loan Approved by Bank
26	SBA_Appv	Currency	SBA's Guaranteed Amount of Approved Loan
27	New	Number	=1 if NewExist=2 (New Business), =0 if NewExis...
28	Portion	Number	Proportion of Gross Amount Guaranteed by SBA
29	RealEstate	Number	=1 if loan is backed by real estate, =0 otherwise
30	Recession	Number	=1 if loan is active during Great Recession, =...
31	Selected	Number	=1 if the data are selected as training data t...
32	NaN	NaN	NaN
33	Default	Number	=1 if MIS_Status=CHGOFF, =0 if MIS_Status=P I F
34	daysterm	Number	Extra variables generated when creating "Reces...

Next steps: Generate code with data_dictionary | View recommended plots | New interactive sheet

For this case, you need to apply the decision rules and cutoff probability of 0.5 from Section 4.3 to classify the two loans in Table 10 of the article "Should This Loan be Approved or Denied?": A Large Dataset with Class Assignment Guidelines as "higher risk" or "lower risk" for loan approval by writing Python code to reproduce results (not format) in Tables 7(a), 8, 9 of this article using the SBA case data SBAcase.11.13.17.csv. The variable "Selected" indicates which observations are the "training" data and which are the "testing" data (1 = training data to be used to build the model, 0 = testing data to validate the model). Partition the data using this variable.

```
sba_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/SBAcase.csv')
sba_data
```

	Selected	LoanNr_ChkDgt	Name	City	State	Zip	Bank	BankState	NAICS	ApprovalDate	...	ChgOffPrinGr
0	0	1004285007	SIMPLEX OFFICE SOLUTIONS	ANAHEIM	CA	92801	CALIFORNIA BANK & TRUST	CA	532420	15074	...	0
1	1	1004535010	DREAM HOME REALTY	TORRANCE	CA	90505	CALIFORNIA BANK & TRUST	CA	531210	15130	...	0
2	0	1005005006	Winset, Inc. dba Bankers Hill	SAN DIEGO	CA	92103	CALIFORNIA BANK & TRUST	CA	531210	15188	...	0
3	1	1005535001	Shiva Management	SAN DIEGO	CA	92108	CALIFORNIA BANK & TRUST	CA	531312	15719	...	0
4	1	1005996006	GOLD CROWN HOME LOANS, INC	LOS ANGELES	CA	91345	SBA - EDF ENFORCEMENT ACTION	CO	531390	16840	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
2097	1	9893874006	MEGA VIDEO & WIRELESS HIGHLAND	HIGHLAND	CA	92346	UNITI BANK	CA	532230	16838	...	0
2098	1	9901143004	MOVING CONNECTION & CARL'S TRA	EL CAJON	CA	92021	ZIONS FIRST NATIONAL BANK	UT	532120	13530	...	0
2099	0	9903293007	A.J. STUDIO RENTALS, INC.	CAMARILLO	CA	93012	CITY NATIONAL BANK	CA	532120	13531	...	0
2100	1	9925643006	TAHOE MOTION PICTURE RENTALS	SUN VALLEY	CA	91352	CITY NATIONAL BANK	CA	532120	13542	...	0
2101	0	9958873001	EFM ASSOCIATES	LOS ANGELES	CA	90068	WELLS FARGO BANK NATL ASSOC	SD	531110	13557	...	0

2102 rows × 35 columns

Part (a): Review Python documentation: "statsmodels" and example code from the class. Fit a logistic regression model to reproduce results (not format) in Tables 7(a), 8, 9 of this article using the SBA case data SBAcase.11.13.17.csv by using STATMODELS sm.glm or smf.glm.

The logit model produces an estimated probability of being a “1”. Classify as “1” if this estimated probability > cutoff, e.g., 0.5. The following code from Table 10.3 in the example code may be helpful to perform this classification:

Table 10.3 example code:

```
# Table 10.3 example code:
logit = sm.GLM(train_y, train_X, family=sm.families.Binomial())
result = logit.fit()
predictions = result.predict(valid_X)
predictions_nominal = [ 0 if x < 0.5 else 1 for x in predictions]
logit_result = pd.DataFrame({'actual': valid_y,
                             'p(0)': 1 - predictions,
                             'p(1)': predictions,
                             'predicted': predictions_nominal })


# Define the target (dependent) variable 'Default' and the independent variables
dependent_var = 'Default'
independent_vars = ['RealEstate', 'Portion', 'Recession']

# Convert the dependent variable to numeric
sba_data[dependent_var] = pd.to_numeric(sba_data[dependent_var], errors='coerce')

# Split data into training and validation/test sets based on the 'Selected' variable
```

```
train_data = sba_data[sba_data['Selected'] == 1]
test_data = sba_data[sba_data['Selected'] == 0]

# Define the training and validation/test sets
train_X = train_data[independent_vars]
train_y = train_data[dependent_var]

valid_X = test_data[independent_vars]
valid_y = test_data[dependent_var]

# Add a constant to the model (intercept term)
train_X = sm.add_constant(train_X)
valid_X = sm.add_constant(valid_X)

# Fit the logistic regression model
logit_model = sm.GLM(train_y, train_X, family=sm.families.Binomial())
logit_result = logit_model.fit()

# Display the model summary
logit_result.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	Default	**No. Observations:**	1051
Model:	GLM	**Df Residuals:**	1047
Model Family:	Binomial	**Df Model:**	3
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-541.38
Date:	Sun, 23 Feb 2025	**Deviance:**	1082.8
Time:	21:54:19	**Pearson chi2:**	989.
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.1941
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.3931	0.322	4.332	0.000	0.763	2.023
RealEstate	-2.1282	0.345	-6.169	0.000	-2.804	-1.452
Portion	-2.9875	0.539	-5.540	0.000	-4.044	-1.931

```
# Make predictions on the test set
predictions = logit_result.predict(valid_X)

# Classify loans as 1 (high risk) if probability > 0.5
predictions_nominal = [1 if x > 0.5 else 0 for x in predictions]

# Compare predictions with actual outcomes
result_df = pd.DataFrame({
    'actual': valid_y,
    'predicted': predictions_nominal,
    'probability': predictions
})

# Display result summary
print(result_df.head())
```

```
    actual  predicted  probability
0       0          0     0.474852
2       0          0     0.474852
6       1          0     0.048533
7       0          0     0.474852
8       1          0     0.474852
```

```
logit = sm.GLM(train_y, train_X, family=sm.families.Binomial())
result = logit.fit()
predictions = result.predict(valid_X)
predictions_nominal = [ 0 if x < 0.5 else 1 for x in predictions]
logit_result = pd.DataFrame({'actual': valid_y,
                             'p(0)': 1 - predictions,
                             'p(1)': predictions,
                             'predicted': predictions_nominal })
logit_result
```

	actual	p(0)	p(1)	predicted
0	0	0.525148	0.474852	0
2	0	0.525148	0.474852	0
6	1	0.951467	0.048533	0
7	0	0.525148	0.474852	0
8	1	0.525148	0.474852	0
...	...	...	...	...
2094	0	0.525148	0.474852	0
2095	0	0.976400	0.023600	0
2096	0	0.976400	0.023600	0
2099	0	0.730453	0.269547	0
2101	0	0.525148	0.474852	0

Next steps: Generate code with logit_result | View recommended plots | New interactive sheet

Start coding or generate with AI.

Part (b): Refer to Table 8 of the article. Write the estimated equation that associates the outcome variable (i.e., default or not) with predictors RealEstate, Portion, and Recession, in three formats:

(i) The logit as a function of the predictors (see (10.6) of DMBA Chapter 10.2)

(ii) The odds as a function of the predictors (see (10.5) of DMBA Chapter 10.2)

(iii) The probability as a function of the predictors (see (10.2) of DMBA Chapter 10.2)

```
# (i) The logit as a function of the predictors
logit_eq = "logit(Default) = 1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession"

# (ii) The odds as a function of the predictors
odds_eq = "Odds(Default) = exp(1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession)"

# (iii) The probability as a function of the predictors
prob_eq = "P(Default) = 1 / (1 + exp(-(1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession)))"

logit_eq, odds_eq, prob_eq
```

```
('logit(Default) = 1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession',
 'Odds(Default) = exp(1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession)',
 'P(Default) = 1 / (1 + exp(-(1.3931 - 2.1282*RealEstate - 2.9875*Portion + 0.5041*Recession)))')
```

Start coding or generate with AI.

Part (c): Explain why risk indicators in Table 8 were selected using p-values in Table 7(a).

Answer:

Risk indicators such as RealEstate, Portion, and Recession were selected because they have statistically significant p-values ($p < 0.05$). The low p-values for these predictors suggest that they have a strong association with the likelihood of loan default, making them suitable for inclusion in the logistic regression model.

- RealEstate and Portion both have highly significant p-values (close to 0), indicating that loans with real estate backing or a higher portion guaranteed by SBA strongly impact default risk.
- Recession has a p-value of 0.037, which is also below the 0.05 threshold, indicating a significant effect of loans active during a recession on default risk.

Start coding or generate with AI.

Part (d): Interpret parameter (coefficient) estimates of the model in Table 8 with a focus on the odds of default. Answer the following questions by interpreting parameter estimates of the model in Table 8 and specifying odds and probabilities of default for these risk

indicators.

(i) Is a loan backed by real estate more likely or less likely to default (by how much)? Explain using parameter estimates.

(ii) Is a loan active during recession more likely or less likely to default (by how much)? Explain using parameter estimates.

(iii) How much does the portion of a loan guaranteed by SBA increase or decrease the likelihood of default? Explain using parameter estimates.

```
# Coefficients from the logistic regression model
coef_real_estate = -2.1282
coef_portion = -2.9875
coef_recession = 0.5041

# Calculate odds ratios by exponentiating the coefficients
odds_ratio_real_estate = np.exp(coef_real_estate)
odds_ratio_portion = np.exp(coef_portion)
odds_ratio_recession = np.exp(coef_recession)

# Display the results
odds_ratio_real_estate, odds_ratio_recession, odds_ratio_portion
```

```
(0.11905139361332186, 1.6554949043702933, 0.05041331259467938)
```

- RealEstate: Odds Ratio ≈ 0.119. This means that loans backed by real estate are approximately 88.1% less likely to default compared to loans without real estate backing.
- Recession: Odds Ratio ≈ 1.655. Loans active during a recession are approximately 65.5% more likely to default compared to those not active during a recession.
- Portion: Odds Ratio ≈ 0.050. A higher portion of a loan guaranteed by the SBA reduces the odds of default by 95% for each unit increase in the guaranteed portion.

```
Start coding or generate with AI.
```

Part (e): For the California-based example, the final model with the risk indicators in Table 8 is used to estimate the probability of default for the two loan applications. Use Python to predict the probability of default for Carmichael Realty (Loan 1) and SV Consulting (Loan 2). Applying the decision rules and cutoff probability of 0.5 from Section 4.3, how should these two loans be classified as, lower risk (approve) or higher risk (deny)?

```
# Define the input data for the two loan applications:
# Loan 1: Carmichael Realty (Has real estate, 75% of loan guaranteed by SBA, Not recession)
loan1 = pd.DataFrame({
    'const': [1],
    'RealEstate': [1],  # Real estate-backed
    'Portion': [0.75],  # 75% SBA guaranteed ($750,000 out of $1,000,000)
    'Recession': [0]    # Not recession
})

# Loan 2: SV Consulting (No real estate, 40% of loan guaranteed by SBA, Not recession)
loan2 = pd.DataFrame({
    'const': [1],
    'RealEstate': [0],  # No real estate
    'Portion': [0.4],   # 40% SBA guaranteed ($40,000 out of $100,000)
    'Recession': [0]    # Not recession
})

print(loan1)
print(loan2)
```

```
   const  RealEstate  Portion  Recession
0      1           1     0.75          0
   const  RealEstate  Portion  Recession
0      1           0      0.4          0
```

```
# Assuming the logistic regression model has been fitted like this:
logit_model = sm.GLM(train_y, train_X, family=sm.families.Binomial())
logit_result = logit_model.fit()
```

```
# Predict the probability of default for each loan using the fitted model
loan1_prob = logit_result.predict(loan1)[0]
```

```
loan2_prob = logit_result.predict(loan2)[0]

# Classify based on the 0.5 cutoff (higher risk if probability > 0.5, otherwise lower risk)
loan1_classify = 'Higher Risk (Deny)' if loan1_prob > 0.5 else 'Lower Risk (Approve)'
loan2_classify = 'Higher Risk (Deny)' if loan2_prob > 0.5 else 'Lower Risk (Approve)'

print("Carmichael Realty:")
print("Probability:", loan1_prob)
print("Classification:", loan1_classify)
print(" ")
print("SV Consulting:")
print("Probability:", loan2_prob)
print("Classification:", loan2_classify)
```

```
Carmichael Realty:
Probability: 0.048532822357088695
Classification: Lower Risk (Approve)

SV Consulting:
Probability: 0.5493564480854694
Classification: Higher Risk (Deny)
```

Start coding or generate with AI.

Part (f): The default cutoff value of 0.5 is used in conjunction with the probability of default. Compute the threshold that should be used if we want to make a classification based on the odds of default, and the threshold for the corresponding logit.

```
# Define the probability cutoff
probability_cutoff = 0.5

# Calculate the corresponding odds for the given probability cutoff
odds_threshold = probability_cutoff / (1 - probability_cutoff)

# Calculate the logit (natural log of the odds) for the given probability cutoff
logit_threshold = np.log(odds_threshold)

odds_threshold, logit_threshold
```

```
(1.0, 0.0)
```

Start coding or generate with AI.

Part (g): When a “higher risk—more likely to default” loan is misclassified as “lower risk—more likely to pay in full” loan, the misclassification cost is much higher than when a “lower risk—more likely to pay in full” loan is misclassified as “higher risk—more likely to default”. To minimize the expected cost of misclassification, should the cutoff value for classification (which is currently at 0.5) be increased or decreased?

Answer:

When the cost of misclassifying a "higher risk" loan as "lower risk" is much higher than the reverse, the cutoff value should be decreased. By lowering the cutoff, more loans would be classified as higher risk, reducing the chance of underestimating default risk, which helps minimize the high cost associated with misclassifying risky loans as safe.

Thus, to minimize the expected cost of misclassification, the cutoff value should be lowered (< 0.5).

Start coding or generate with AI.

Part (h): (Skip this part. Complete this part after next week's class)

Review Python documentation: sklearn.linear_model.LogisticRegression and example code from the class. Fit a logistic regression model to reproduce results (not format) in Tables 7(a), 8, 9 of this article using the SBA case data SBAcase.11.13.17.csv by using (1) sklearn LogisticRegression() liblinear solver and (2) sklearn LogisticRegression() Default Solver 'lbfgs'. Adjust parameters such as "penalty" and "tol" as needed.

Start coding or generate with AI.

Start coding or generate with AI.