Data Science – Risk Factor Identification

Chose Home Equity Loan CSV File:

A picture containing text, diagram

Description automatically generated

A picture containing text, screenshot, line, display

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

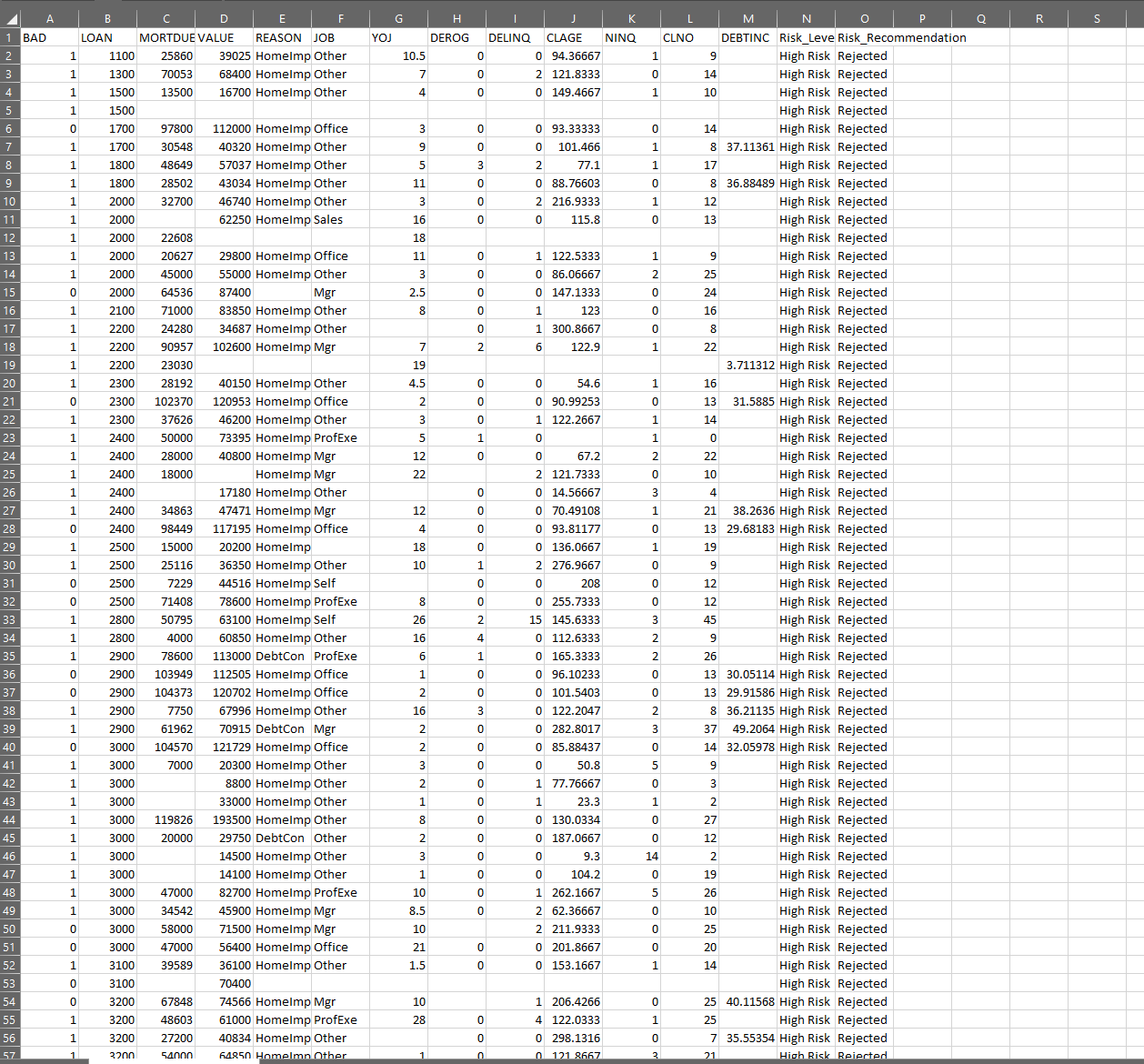
A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

|  |
| --- |
| Python Code: |
| # -\*- coding: utf-8 -\*-  """  Created on Sun May 14 04:45:00 2023  @author: Franc  """  # Imported Libraries  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  #%matplotlib inline  import warnings  warnings.filterwarnings('ignore')  import statsmodels.api as sm  # Load Data  # Data used is the Home Equity Loans (HMEQ)  dat = pd.read\_csv("hmeq.csv")  print(dat.shape) # Prints the (rows, columns) in the console  dat.head(5)  # Creating a correlation matrix, This will identify multicollinearity among the numercial variables.  dat.corr()  # creates a new dataset, df, that contains only the numeric variables  df = dat[['BAD','LOAN','MORTDUE','VALUE','YOJ','DELINQ','NINQ']]  # The second line creates the plot, where the argument kind="scatter" creates the plot without the regression line.  sns.pairplot(df, kind="scatter")  # Plot  plt.show()  plt.scatter(dat['BAD'], dat['LOAN'])  plt.show()  np.corrcoef(dat['BAD'], dat['LOAN'])  print(np.corrcoef(dat['BAD'], dat['LOAN']))  print("Number of Missing Values in the BAD column:", dat['BAD'].isnull().sum()) # number of missing values in 'YOJ' column  print("Number of Missing Values in the LOAN column:", dat['LOAN'].isnull().sum()) # number of missing values in 'DELINQ' column  from scipy.stats import linregress  linregress(dat['BAD'], dat['LOAN'])  print(linregress(dat['BAD'], dat['LOAN']))  pd.crosstab(dat['BAD'], dat['LOAN'])  from scipy.stats import chi2\_contingency  chi2\_contingency(pd.crosstab(dat.BAD, dat.MORTDUE))  # Define a function that will analyze the home equity data and identify our risk factors  def analyze\_loan\_data(data):  # Calculate the default rate for each loan type  default\_rates = data.groupby('LOAN')['BAD'].mean()  # Determine the risk level for each loan type based on the default rate  risk\_levels = pd.cut(default\_rates, bins=[0, 0.1, 0.2, 1], labels=['Low Risk', 'Medium Risk', 'High Risk'])  # Add a new column to the loan data with the risk level for each loan type  data['Risk\_Level'] = data['LOAN'].map(risk\_levels)  return data  # Analyze the loan data and get the results  analyzed\_data = analyze\_loan\_data(dat)  # Apply the risk level recommendations based on the analysis results  risk\_recommendations = [] # Creates an array  for level in analyzed\_data['Risk\_Level']: # Loops in the csv file in the coloumn Risk Level and sees if it is Low, Medium, or High Risk and makes and additional  # Coloumn to put if it approved, needs mangaerial approval, or denied.  if level == 'Low Risk':  risk\_recommendations.append('Approved')  elif level == 'Medium Risk':  risk\_recommendations.append('Need Managerial Approval')  else:  risk\_recommendations.append('Rejected')  analyzed\_data['Risk\_Recommendation'] = risk\_recommendations  '''  analyzed\_data['Risk\_Recommendation'] = ''  analyzed\_data.loc[analyzed\_data['Risk\_Level'] == 'Low', 'Risk\_Recommendation'] = 'Approved'  analyzed\_data.loc[analyzed\_data['Risk\_Level'] == 'Medium', 'Risk\_Recommendation'] = 'Need Managerial Approval'  analyzed\_data.loc[analyzed\_data['Risk\_Level'] == 'High', 'Risk\_Recommendation'] = 'Rejected'  '''  # Write the results to a new CSV file with the new columns added  analyzed\_data.to\_csv('home\_equity\_data\_analysis.csv', index=False)  # Summarize Data of the analysis results and risk level recommendations  print('------- Loan Data Analysis Results -------')  print('Total number of loans:', len(analyzed\_data))  print('Number of loans in each risk category:')  print(analyzed\_data['Risk\_Level'].value\_counts())  print('Risk level recommendations for each loan:')  print(analyzed\_data[['LOAN', 'Risk\_Level', 'Risk\_Recommendation']]) |

Data Science - Trend Prediction

A graph with a red line and green line

Description automatically generated with low confidence

A picture containing text, screenshot, plot, line

Description automatically generated

|  |
| --- |
| Python Code: |
| # -\*- coding: utf-8 -\*-  """  Created on Tue May 9 17:28:57 2023  @author: Franc  """  # Import Libraries / Modules  import datetime  import pandas as pd  import numpy as np  import statistics as stats  import matplotlib.pyplot as plt  from sklearn.linear\_model import Ridge, LinearRegression  from sklearn.preprocessing import PolynomialFeatures  from sklearn.pipeline import make\_pipeline  # Read the CSV file for the data of 6 months  data = pd.read\_csv("Covid19\_6months.csv", low\_memory=False)  data.head()  to\_drop = ['Total\_cases', 'New\_cases'] # remove 'Date' since it is not used  data.drop(to\_drop, inplace=True, axis=1)  print("---- 6 Months ----")  print(data.head())  print()  # Read The CSV File for the full 12months  data\_all = pd.read\_csv("COVID19\_12months.csv", low\_memory=False)  data\_all.head()  to\_drop\_all = ['Total\_cases', 'New\_cases'] # remove 'Date' since it is not used  data\_all.drop(to\_drop\_all, inplace=True, axis=1)  print("---- 12 Months ----")  print(data\_all.head())  print()  # Arrays to hold the information in the CSV Files  date = []  cases = []  all\_date = []  all\_cases = []  # Open File to split the data to store into the arrays  with open("Covid19\_6months.csv","r") as f:  f.readline()  for l in f:  l.strip()  things=l.split(",")  if things[0]:  date.append([datetime.datetime.strptime(things[0],"%m/%d/%Y").timestamp()])  cases.append([float(things[2])])    all\_date.append([datetime.datetime.strptime(things[0],"%m/%d/%Y").timestamp()])  all\_cases.append([float(things[2])])  f.close()  # Open File to split the data to store into the arrays  with open("COVID19\_12months.csv","r") as f:  f.readline()  for l in f:  l.strip()  things=l.split(",")  if things[0]:  all\_date.append([datetime.datetime.strptime(things[0],"%m/%d/%Y").timestamp()])  all\_cases.append([float(things[2])])  f.close()  # Plotting both the Prediction and Actual Graphs  model = make\_pipeline(PolynomialFeatures(3), Ridge())  model.fit(date, cases)  y\_plot = model.predict(date)  all\_y\_plot = model.predict(all\_date)  # Regression Graph of 6 months  plt.title('Covid Cases 01/10/2020-06/30/2020')  plt.plot(date, cases, "b")  plt.plot(date, y\_plot, "g")  plt.plot(all\_date, all\_cases, "r")  plt.legend(["Data Used For Prediction", "Ridge Regression Prediction", "All Data"])  plt.show()  # Comparison Graph of 12 Months  plt.title('Covid Cases 01/10/2020-1/30/2021')  plt.plot(all\_date, all\_cases, "g")  plt.plot(date, cases)  plt.plot(all\_date, all\_y\_plot, "r")  plt.legend(["Data Used For Prediction", "All Data", "Ridge Regression Prediction"])  plt.show() |

Game Development – Dance Challenge

Worked on this code with partner Peter Nguyen!

Song: Rock Ur World -KNOCK2

A screenshot of a video game

Description automatically generated

A screenshot of a video game

Description automatically generated

|  |
| --- |
| Python |
| # -\*- coding: utf-8 -\*-  """  Created on Sun May 14 05:53:29 2023  @author: Franc, Peter  """  #Importing necessary modules and functions  import music, time  from pgzero.builtins import Actor, clock  import pgzrun  from random import randint  #Setting up the game window dimensions  WIDTH = 800  HEIGHT = 600  # Center of the Screen  CENTER\_X = WIDTH / 2  CENTER\_Y = HEIGHT / 2  #Initializing lists  move\_list = []  display\_list = []  # Starting Global Variables  # Individual Player Scores  score1 = 0  score2 = 0  current\_move = 0  count = 4 #Countdown timer value  dance\_length = 4  # Flags  say\_dance = False #Flag to display "Dance!" message  show\_countdown = True #Flag to display countdown  moves\_complete = False #Flag to indicate if all moves have been displayed  game\_over = False #Flag to indicate if the game is over  #Creating actor objects for the dancer and dance move arrows  dancer = Actor("dancer-start")  dancer.pos = CENTER\_X + 5, CENTER\_Y - 40  up = Actor("up")  up.pos = CENTER\_X, CENTER\_Y + 110  right = Actor("right")  right.pos = CENTER\_X + 60, CENTER\_Y + 170  down = Actor("down")  down.pos = CENTER\_X, CENTER\_Y + 230  left = Actor("left")  left.pos = CENTER\_X - 60, CENTER\_Y + 170  # Function to draw the game elements on the screen  def draw():  global game\_over, score1, score2, say\_dance, count, show\_countdown  if not game\_over:  #If the game is not over, draw the dancer, arrows, and scores  screen.clear()  screen.blit("stage", (0, 0))  dancer.draw()  up.draw()  down.draw()  right.draw()  left.draw()    # Tracking Player 1 and 2 Scores  screen.draw.text("Player 1 Score: " + str(score1), color="black", topleft=(10, 10))  screen.draw.text("Player 2 Score: " + str(score2), color="black", topleft=(WIDTH - 135, 10))  # The countdown to the start of the next sequence of dances  if say\_dance:  screen.draw.text("Dance!", color="black", topleft=(CENTER\_X - 65, 150), fontsize=60)  if show\_countdown:  screen.draw.text(str(count), color="black", topleft=(CENTER\_X - 8, 150), fontsize=60)  else:  # If the game is over, only draw the scores and "GAME OVER!" message and song being used.  screen.clear()  screen.blit("stage", (0, 0))  screen.draw.text("Player 1 Score: " + str(score1), color="black", topleft=(10, 10))  screen.draw.text("Player 2 Score: " + str(score2), color="black", topleft=(WIDTH - 135, 10))  screen.draw.text("GAME OVER!", color="black", topleft=(CENTER\_X - 130, 220), fontsize=60)  screen.draw.text("SONG: KNOCK UR WORLD", color="black", topleft=(CENTER\_X - 250, 255), fontsize=60)  # This function is to reset the dancer and arrow images after User presses button.  def reset\_dancer():  global game\_over  if not game\_over:  dancer.image = "dancer-start"  up.image = "up"  right.image = "right"  down.image = "down"  left.image = "left"  # Update the dancer and arrow images based on the moves presented  def update\_dancer(move):  global game\_over  if not game\_over:  if move == 0:  up.image = "up-lit"  dancer.image = "dancer-up"  clock.schedule(reset\_dancer, 0.5)  elif move == 1:  right.image = "right-lit"  dancer.image = "dancer-right"  clock.schedule(reset\_dancer, 0.5)  elif move == 2:  down.image = "down-lit"  dancer.image = "dancer-down"  clock.schedule(reset\_dancer, 0.5)  else:  left.image = "left-lit"  dancer.image = "dancer-left"  clock.schedule(reset\_dancer, 0.5)  # This function will display the set of dance moves on the screen through the actor and highlighting the  # arrows.  def display\_moves():  global move\_list, display\_list, dance\_length, say\_dance, show\_countdown, current\_move  # If there are moves left to display, get the next move from the list  if display\_list:  this\_move = display\_list[0]  display\_list = display\_list[1:]  if this\_move == 0:  update\_dancer(0)  clock.schedule(display\_moves, 1)  elif this\_move == 1:  update\_dancer(1)  clock.schedule(display\_moves, 1)  elif this\_move == 2:  update\_dancer(2)  clock.schedule(display\_moves, 1)  else:  update\_dancer(3)  clock.schedule(display\_moves, 1)  else:  say\_dance = True  show\_countdown = False  # This function is to generate random dance moves for the user to mimic  def generate\_moves():  global move\_list, dance\_length, count, show\_countdown, say\_dance  count = 4  move\_list = []  say\_dance = False  for move in range(0, dance\_length):  #Generate a random move (0: up, 1: right, 2: down, 3: left) and add it to the move list  rand\_move = randint(0, 3)  move\_list.append(rand\_move)  display\_list.append(rand\_move)  show\_countdown = True  countdown()  # This function to implement the countdown before displaying a sequence of dance moves  def countdown():    global count, game\_over, show\_countdown  if count > 1:  count = count - 1  clock.schedule(countdown, 1)  else:  show\_countdown = False  display\_moves()  # Function to move to the next dance move in the sequence  def next\_move():  global dance\_length, current\_move, moves\_complete  if current\_move < dance\_length - 1:  current\_move = current\_move + 1  else:  moves\_complete = True  # Function to handle key release events  def on\_key\_up(key):  global score1, score2, game\_over, move\_list, current\_move, secondplayer  #If it's Player 1's turn, check the released key and update the dancer and scores accordingly  if secondplayer == 1:  if key == keys.UP:  update\_dancer(0)  if move\_list[current\_move] == 0:  score1 = score1 + 1  next\_move()  else:  game\_over = True  elif key == keys.RIGHT:  update\_dancer(1)  if move\_list[current\_move] == 1:  score1 = score1 + 1  next\_move()  else:  game\_over = True  elif key == keys.DOWN:  update\_dancer(2)  if move\_list[current\_move] == 2:  score1 = score1 + 1  next\_move()  else:  game\_over = True  elif key == keys.LEFT:  update\_dancer(3)  if move\_list[current\_move] == 3:  score1 = score1 + 1  next\_move()  else:  game\_over = True  else:  #If it's Player 2's turn, check the released key and update the dancer and scores accordingly  if key == keys.W:  update\_dancer(0)  if move\_list[current\_move] == 0:  score2 = score2 + 1  next\_move()  else:  game\_over = True  elif key == keys.D:  update\_dancer(1)  if move\_list[current\_move] == 1:  score2 = score2 + 1  next\_move()  else:  game\_over = True  elif key == keys.S:  update\_dancer(2)  if move\_list[current\_move] == 2:  score2 = score2 + 1  next\_move()  else:  game\_over = True  elif key == keys.A:  update\_dancer(3)  if move\_list[current\_move] == 3:  score2 = score2 + 1  next\_move()  else:  game\_over = True  # update the game state  def update():  global game\_over, current\_move, moves\_complete, secondplayer  if not game\_over:  #If all moves have been executed, generate a new sequence and switch players  if moves\_complete:  generate\_moves()  moves\_complete = False  current\_move = 0  secondplayer = secondplayer \* -1  #Setting the initial player to 1 and generating the first dance moves  secondplayer = 1  generate\_moves()  music.play("knock2-rock-ur-world-\_ft.-fussy.ogg")  pgzrun.go()  update() |

Job interview Q&A :

Text File in Folder for Submission

Video I recommended to watch for interviewed by Apple Engineer in Python: <https://interviewing.io/mocks/apple-python-count-islands>