# male-employment-gap-modelling-code

October 10, 2024

[13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import RobustScaler
import math
import itertools
import logging



plot = False
companyNames = ['annapurna', 'chaitanya' ,'muthoot' , 'belstar' , 'satin' ,␣
 ↪'creditaccess' , 'asirvad' , 'spandanassphoorty' , 'bandhan']
finalResults = {}
totalBorrowersMicrofinance = 55000000
```

[14]:
```
# Load the Excel file
file_path = './Microfinance Data 21.xlsx'
excel_data = pd.ExcelFile(file_path)

# Load the data from the first sheet into a DataFrame
df = pd.read_excel(file_path, sheet_name=excel_data.sheet_names[0])


# Set Index of Data Frame to years
df.index = np.arange(1999, 2024)




projections = {}  # Dictionary to store all made projections
```

[15]:
```
# Prototype function that takes in two ARRAYS and a boolean (lower) as inputs␣
 ↪and fits the best polynomial regression model to it,
```

```
# Boolean (lower) is to determine whether to lower degree even further to avoid␣
 ↪overfitting
# Limits degrees to 3 to prevent overfitting

def best_polynomial_fit(X, y, lower):


    best_degree = 1
    best_r2 = -np.inf # Minus Infinity
    best_model = None

    # If statement to determine whether or not to lower degree

    if lower:
      degree = 2
    else:
      degree = 3

    # Run loop to determine how well each polynomial regression model works and␣
 ↪to utilise it

    for degree in range(1, degree+1):
        poly = PolynomialFeatures(degree)
        X_poly = poly.fit_transform(X)
        model = LinearRegression().fit(X_poly, y)
        r2 = r2_score(y, model.predict(X_poly)) # How well regression line␣
 ↪approximates data

        if r2 > best_r2:  # If approximates better switch best model to current␣
 ↪model
            best_r2 = r2
            best_degree = degree
            best_model = model




    return best_degree, best_model, best_r2
```

[16]:
```
def predict(dfKey, title, percent):

    y = df[dfKey].dropna().values.reshape(-1, 1)

    firstYear = df[dfKey].first_valid_index()
    lastYear = df[dfKey].last_valid_index()

    yearsTill2017 = 2017 - firstYear
```

```
  noNegative = True
  previousNum = 0
  numberOfValues = 0
  for num in df[dfKey]:

    if  math.isnan(num):
      continue
    numberOfValues += 1
    if num >= previousNum:

      previousNum = num
    else:
      noNegative = False

  yearsCurrent = np.arange(firstYear, lastYear+1).reshape(-1, 1)
  years_future_local =  np.arange(firstYear, 2031).reshape(-1, 1)



  best_degree, best_model, best_r2 = best_polynomial_fit(yearsCurrent, y,␣
↪False)

  poly = PolynomialFeatures(best_degree)
  future_X_poly = poly.fit_transform(years_future_local)
  projection = best_model.predict(future_X_poly)

  if (percent and projection[-1] >= 100) or␣
↪(df[dfKey][lastYear]>projection[-1] and noNegative):
      best_degree, best_model, best_r2 = best_polynomial_fit(yearsCurrent, y,␣
↪True)

      poly = PolynomialFeatures(best_degree)
      future_X_poly = poly.fit_transform(years_future_local)
      projection = best_model.predict(future_X_poly)


  if not ('Borrowers' in dfKey):
    projections[dfKey] = projection[yearsTill2017:]
  else:
    projections[dfKey] = projection[yearsTill2017-1:]
```

```
    if plot:
        plt.figure(figsize=(12, 6))
        plt.plot(years_future_local, projection, color='blue',␣
 ↪label=f'Projected {title} (Degree {best_degree})')
        plt.scatter(yearsCurrent, y, color='red', label='Actual Data')
        plt.title(title)
        plt.xlabel('Years')
        plt.grid(True)
        plt.legend()
        plt.show()
```

[17]:
```
predict('workingAgesFemaleIndia', 'Number of Working Aged Females India', False)
predict('totalBranches', 'Total Number of MFI Branches', False)
predict('selfEmploymentIndia', 'Percentage of people self employed in India',␣
 ↪True)
predict('mfiMarketshare', 'Microfinance Marketshare of Borrowers', True)
predict('employmentRateWorld', 'World Employment Rate', True)
predict('employmentRateIndia', 'Indian Employment Rate', True)
```

[18]:
```
def calculateChangeBorrowers(companyName) :
    predict(f'{companyName}Borrowers', f'{companyName} Borrowers', False)
    projections[f'{companyName}ChangeBorrowers'] = []


    index = 0
    for i in projections[f'{companyName}Borrowers']:

      if index == 0:
        previous = i
        index += 1
        continue

      current = i

      projections[f'{companyName}ChangeBorrowers'].append(current - previous)

      previous = current


    return projections[f'{companyName}ChangeBorrowers']
```

[19]:
```
for companyName in companyNames:
    predict(f'{companyName}Branches', f'Total Number of {companyName}␣
 ↪Branches', True)
    x = calculateChangeBorrowers(companyName)
```

[20]:
```
def totalBorrowersNeeded(companyName, year):
    n = year - 2017
    companyBranches = projections[f'{companyName}Branches'][6]
    workingAgesFemaleIndiaCurrent = projections['workingAgesFemaleIndia'][6] #␣
↪Starting working aged females india
    workingAgesFemaleIndiaFuture = projections['workingAgesFemaleIndia'][n]
    employmentRateWorld = projections['employmentRateWorld'][n] / 100
    employmentRateIndia = projections['employmentRateIndia'][6] /100 # Static␣
↪Value (2023)
    totalBranches = projections['totalBranches'][6]
    mfiMarketshare = projections['mfiMarketshare'][n] / 100
    selfEmploymentIndia = projections['selfEmploymentIndia'][n] /100


    try:
        if df[f'{companyName}IncomeGen']:
          if df[f'{companyName}IncomeGen'].first_valid_index() <= 2022:
              companyIncomeGeneratingLoanRate =␣
↪(df[f'{companyName}IncomeGen'][-2] + df[f'{companyName}IncomeGen'][-1]) / 2
          else:
              companyIncomeGeneratingLoanRate =␣
↪df[f'{companyName}IncomeGen'][-1]



        if df[f'{companyName}FemalePercentage']:
          if df[f'{companyName}FemalePercentage'].first_valid_index() <= 2022:
              companyFemaleLoanRate =␣
↪(df[f'{companyName}FemalePercentagae'][-2] +␣
↪df[f'{companyName}FemalePercentage'][-1]) / 2
          else:
              companyFemaleLoanRate = df[f'{companyName}FemalePercentagae'][-1]




    except:
        pass


    totalNewJobsNeeded = (workingAgesFemaleIndiaFuture*employmentRateWorld)␣
↪-(workingAgesFemaleIndiaCurrent*employmentRateIndia)
    companyShareMicrofinance = companyBranches/totalBranches
    companyIncomeGeneratingLoanRate = 0.985 # Need to be calculated from data␣
↪using general value for now
    companyFemaleLoanRate = 0.99 # Same for this one
```

```
    totalBorrowersNeeded = totalNewJobsNeeded*selfEmploymentIndia*mfiMarketshare*companyShareMicrofinance/companyFemaleLoanRate/companyIncomeGeneratingLoanRate

    return totalBorrowersNeeded
```

[21]:
```
def analyse(companyName):

    results = {}
    startYear = 2024
    endYear = 2030

    formulaResults = []
    t = 0


    for i in range(endYear - startYear + 1):

      formulaResults.append(totalBorrowersNeeded(companyName, startYear+i))
      t += totalBorrowersNeeded(companyName, startYear+i)/(i+1)

    cumulativeBorrowers = []
    totalBorrowersProjected = 0



    for borrowerYear in projections[f'{companyName}ChangeBorrowers'][7:]:
        totalBorrowersProjected += borrowerYear
        cumulativeBorrowers.append(totalBorrowersProjected*1)



    totalBorrowers2023 = projections[f'{companyName}Borrowers'][6]



    diffBorrowers = np.array(cumulativeBorrowers) - np.array(formulaResults)
    scaledDiffBorrowers = [x / totalBorrowers2023 for x in diffBorrowers]




    x = np.arange(2024, 2031).reshape(-1, 1)

    if plot:
      plt.plot(x, cumulativeBorrowers, label ='Cumulative Change Borrowers')
```

```
    plt.plot(x, formulaResults , '-.', label ='Projected Number of Borrowers␣
 ↪Needed')
    plt.xlabel("Year")
    plt.ylabel("Number of Borrowers")
    plt.legend()
    plt.title(f'Comparison of Projected Number of Borrowers vs Number of␣
 ↪Borrowers Needed to close Female Employment Gap for {companyName}')
    plt.show()

  results['scaledDiffBorrowers'] = scaledDiffBorrowers
  results['diffBorrowers'] =diffBorrowers
  results['borrowersNeeded'] = formulaResults
  return results
```

[22]:
```
# Function to analyze trends for each company
def analyse_trend(values):
    increasing = all(x < y for x, y in zip(values, values[1:]))
    decreasing = all(x > y for x, y in zip(values, values[1:]))

    if increasing:
        return "increasing"
    elif decreasing:
        return "decreasing"
    else:
        return "no clear trend"
```

[ ]:
```

```

[23]:
```
# List of company names to analyze
companyNames = ['annapurna', 'chaitanya', 'muthoot', 'belstar', 'satin',␣
 ↪'creditaccess',
                'asirvad', 'spandanassphoorty', 'bandhan']

sum = {}

# Lists to store positive and negative trend companies
positive_companies = []
negative_companies = []
no_trend_companies = []

# Function to analyze trends for each company
def analyse_trend(values):
    increasing = all(x < y for x, y in zip(values, values[1:]))
    decreasing = all(x > y for x, y in zip(values, values[1:]))

    if increasing:
        return "increasing"
```

```
    elif decreasing:
        return "decreasing"
    else:
        return "no clear trend"

# Loop to analyze companies and maintain the original functionality
for company in companyNames:
    finalResults[company] = analyse(company)

    scaled_list = []
    for i in finalResults[company]["scaledDiffBorrowers"]:
        scaled_list.append(i)

    sum[company] = scaled_list

    # Check the trend and add the company to the appropriate list (new addition)
    trend = analyse_trend(scaled_list)
    if trend == "increasing":
        positive_companies.append(company)
    elif trend == "decreasing":
        negative_companies.append(company)
    else:
        no_trend_companies.append(company)

min_length = min(len(values) for values in sum.values())
index_sums = [0] * min_length
for key, values in sum.items():
    for i in range(min_length):
        index_sums[i] += values[i]
```

[24]:
```
index_sums
```

[24]:
```
[array([-3.92403572]),
 array([-3.09286631]),
 array([-2.17299266]),
 array([-1.1465089]),
 array([0.00390348]),
 array([1.29519353]),
 array([2.74424292])]
```