AAI590_Profile

April 15, 2024

1 Code for loading the portfolio data from input_data

[12]:
```
#@title 1: Load libraries
import pandas as pd
import numpy as np
import datetime
```

[13]:
```
#@title 2: load data file
df = pd.read_csv('input_data.csv')
```

[14]:
```
#@title 2.1: Review
df.head()
```

[14]:

	name	social_security_number_or_taxpayer_identification_number	address	telephone_number	email	dob	id	employment_status	whether_you_are_employed_by_a_brokerage_firm	annual_income	other_investments	financial_situation	tax_status
0	James Bond	435578	545 Tribe St. Hollywood, CA 90445	2899098876	jamesb@gmail.com	8/10/1966	7	E	N	400000	NaN	NaN	S
1	Penny Love	355666	35566 Sunset Blvd. Los Angeles CA 92333	3102908899	pennylove@gmail.com	7/1/1970	67778	E	N	150000	NaN	NaN	M
2	Que Marshall	48907888	456 James St. Boston MA 02030	2119908876	Qmarshall@gmail.com	4/16/1957	6765555	R	N	300000	Y	NaN	M

```
  investment_experience_and_objectives investment_time_horizon  \
0                                    S                     NaN
1                                    S                     NaN
2                                    M                   short

   liquidity_needs_and_tolerance_for_risk  net_worth trading_experience  \
0                                     NaN    7000000                   M
1                                     NaN     300000                   N
2                                     NaN    3000000                   M

  financial_knowledge  Logic networth
0                   G             NaN
1                   M             NaN
2                   G             NaN
```

1.1 ETL Process

[15]:
```
#@title 3.1: Check for Nan values, replace with 0
df.fillna(0, inplace=True)
```

[16]:
```
#@title 3.2: Remove any rows that dob = 0
df = df[df['dob'] != 0]
```

[17]:
```
#@title 3.3: Covert the dob field
# Convert dob to datetime
df['dob'] = pd.to_datetime(df['dob'], format='%m/%d/%Y')

# Calculate age
today = datetime.datetime.now()
df['age'] = today.year - df['dob'].dt.year

# Remove ages under 21
df = df[df['age'] >= 21]

# Assign logic for age categories
def categorize_age(age):
    if 21 <= age <= 29:
        return 0
    elif 30 <= age <= 39:
        return 1
    elif 40 <= age <= 49:
        return 2
    elif 50 <= age <= 59:
        return 3
    elif 60 <= age <= 68:
        return 4
```

```
    else:
        return 5

df['age_category'] = df['age'].apply(categorize_age)
```

[18]:
```
#@title 3.4: Covert 'employment_status' to 0 and 1
df['employment_status'] = df['employment_status'].apply(lambda x: 0 if x == 'U'␣
 ↪else 1)
```

[19]:
```
#@title 3.5: Convert 'whether_you_are_employed_by_a_brokerage_firm' to 0 and 1
df['whether_you_are_employed_by_a_brokerage_firm'] =␣
 ↪df['whether_you_are_employed_by_a_brokerage_firm'].apply(lambda x: 0 if x ==␣
 ↪'N' else 1)
```

[20]:
```
#@title 3.6: Convert 'investment_experience_and_objectives' to a numeric value␣
 ↪N=0, S=1, M=2
df['investment_experience_and_objectives'] =␣
 ↪df['investment_experience_and_objectives'].apply(lambda x: 0 if x == 'N'␣
 ↪else 1 if x == 'S' else 2)
```

[21]:
```
#@title 3.7: Covert 'investment_time_horizon' to S=0, M=1, L=2
df['investment_time_horizon'] = df['investment_time_horizon'].apply(lambda x: 0␣
 ↪if x == 'short' else 1 if x == 'medium' else 2)
```

[22]:
```
#@title 3.8: Convert 'trading_experience' N=0, S=1, M=2 (N=none, S=some, M=much)
df['trading_experience'] = df['trading_experience'].apply(lambda x: 0 if x ==␣
 ↪'N' else 1 if x == 'S' else 2)
```

[23]:
```
#@title 3.9: Covert 'tax_status' S=0, M=1, D=2, W=3
df['tax_status'] = df['tax_status'].apply(lambda x: 0 if x == 'S' else 1 if x␣
 ↪== 'M' else 2 if x == 'D' else 3)
```

[24]:
```
#@title 3.10: Convert 'financial_knowledge' N=0, M=1, G=2 (N=none, M= medium,␣
 ↪G= good)
df['financial_knowledge'] = df['financial_knowledge'].apply(lambda x: 0 if x ==␣
 ↪'N' else 1 if x == 'M' else 2)
```

[25]:
```
#@title 3.11: Calculate the 'Logic networth' if 'net_worth' <=100000 then = 0,␣
 ↪etc.
def categorize_net_worth(net_worth):
    if net_worth <= 100000:
        return 0
    elif 100000 < net_worth <= 200000:
        return 1
    elif 200000 < net_worth <= 300000:
        return 2
    elif 300000 < net_worth <= 400000:
```

```
        return 3
    elif 400000 < net_worth <= 500000:
        return 5
    elif 500000 < net_worth <= 600000:
        return 6
    elif 600000 < net_worth <= 700000:
        return 7
    elif 700000 < net_worth <= 800000:
        return 8
    elif 800000 < net_worth <= 900000:
        return 9
    elif net_worth > 900000:
        return 10
    return -1  # Just in case there are any out-of-bound values

# Apply the function to the 'net_worth' column to create 'Logic networth'
df['Logic networth'] = df['net_worth'].apply(categorize_net_worth)
```

[26]:
```
#@title 3.12: Function to categorize annual income
def categorize_annual_income(income):
    if income < 100000:
        return 0
    elif 100000 <= income < 300000:
        return 1
    elif 300000 <= income < 600000:
        return 2
    elif income >= 600000:
        return 3
    return -1  # In case there are any out-of-bound values

# Apply the function to the 'annual_income' column to create␣
 ↪'annual_income_score'
df['annual_income_score'] = df['annual_income'].apply(categorize_annual_income)
```

[29]:
```
#@title 4: Calculate the Risk Factor 'liquidity_needs_and_tolerance_for_risk'
# Sum the specified fields
df['total'] = df['age_category'] + df['employment_status'] +␣
 ↪df['investment_experience_and_objectives'] \
             + df['investment_time_horizon'] + df['trading_experience'] +␣
 ↪df['tax_status'] \
             + df['financial_knowledge'] + df['Logic networth'] +␣
 ↪df['annual_income_score']

# Function to categorize the total
def categorize_total(total):
    if 0 <= total <= 10: # minimum drawdown
        return 0
```

```
    elif 11 <= total <= 20: # Min drawdown + max return
        return 1
    elif 21 <= total <= 30: # max return
        return 2
    return -1  # In case there are any out-of-bound values

# Apply the function to categorize the total into the␣
 ↪'liquidity_needs_and_tolerance_for_risk'
df['liquidity_needs_and_tolerance_for_risk'] = df['total'].
 ↪apply(categorize_total)

# Drop the 'total' column if it's no longer needed
df.drop('total', axis=1, inplace=True)

# Write the DataFrame to a CSV file
df.to_csv('input_data_risk.csv', index=False) # Updated with the Risk
```