AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

FSI307

Vanguard: Personalization at scale with Amazon SageMaker

Ayoub Hazrati (he/him)
Senior Data Scientist
Vanguard Group

John Formento, Jr.
Senior Solutions Architect
AWS

aws

Agenda

Introduction

Dive into the personalized Glide Path

Building the solution

- Data generation
- Model training
- CFE surface building and analysis

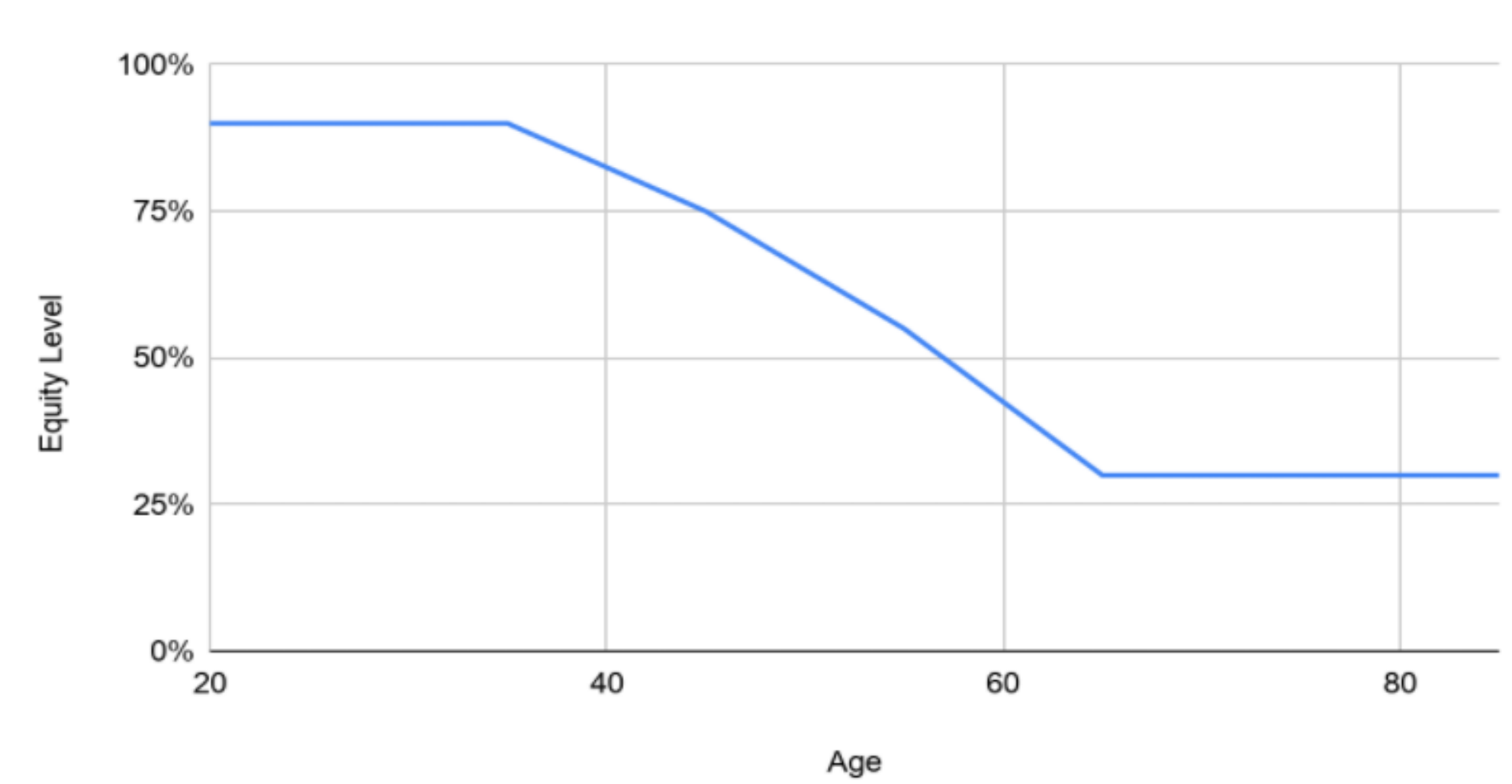
Conclusion

Q&A

Introduction

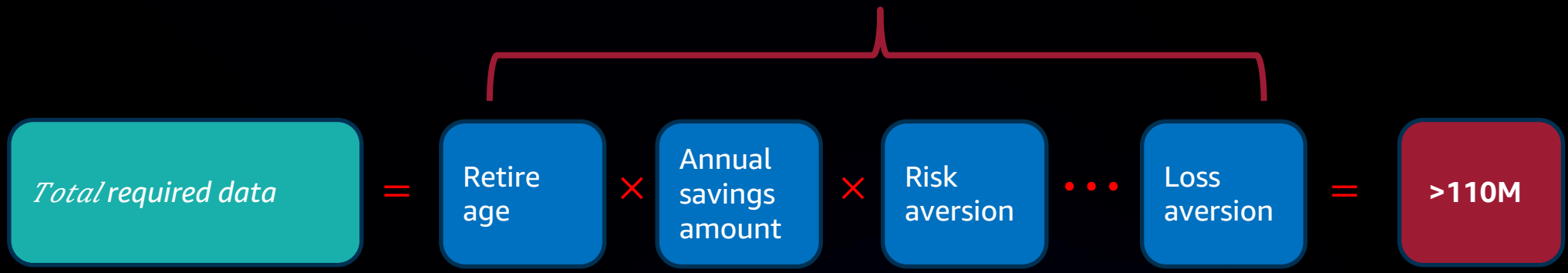
- Vanguard Group is one if the world’s largest investment management companies
- Digital financial advice and planning is one big initiative at Vanguard Group
- Goal of financial advice at Vanguard Group – provide accessible, high quality financial advice for millions of investors
- Vanguard established Enterprise Advice as a pillar toward this goal – more than 400 crew members are working in Enterprise Advice
- Enterprise Advice is the leading division in Vanguard in terms of AWS resource usage

What is a Glide Path?



Challenges

To build each surface with high granularity, we need to generate more than 10K–15K CFE points for each surface with the AI/ML Vanguard Life-Cycle Investing Model (VLCM)



Vanguard Life-Cycle Investing Model (VLCM)

Vanguard

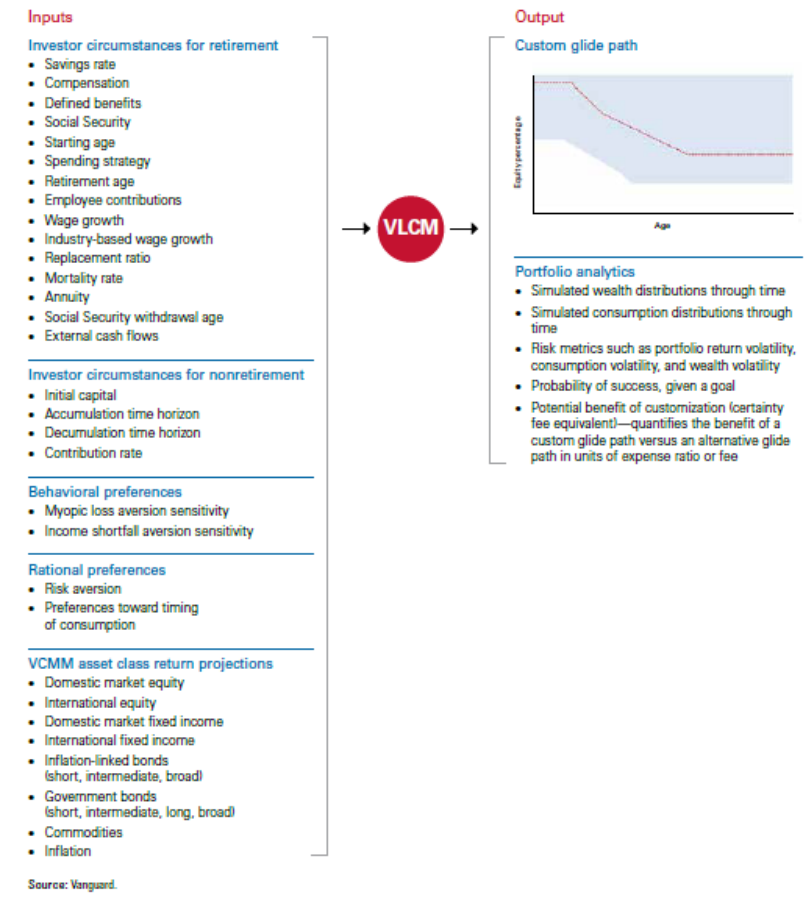
Vanguard's Life-Cycle Investing Model (VLCM): A general portfolio framework for goals-based investing

Vanguard Research March 2021

Roger Aliaga-Díaz, PhD; Harshdeep Ahluwalia, M.Sc; Victor Zhu, CFA, CAIA; Scott Donaldson, CFA, CFP; Ankul Daga, CFA; and David Pakula, CFA

- Investors have multiple goals throughout their lifetime, each requiring them to make complex, interconnected decisions about saving, spending, and asset allocation. We present a framework for making asset allocation decisions based on an investor's goals, preferences, and personal circumstances and factoring in the uncertainty of asset returns.
- The Vanguard Life-Cycle Investing Model (VLCM) is a proprietary model for glide-path construction that can assist in the creation of custom investment portfolios for retirement as well as nonretirement goals, such as saving for college.
- The VLCM embodies key principles of life-cycle investing theory, including a utility-based framework encompassing risk aversion and time preference. It also incorporates important behavioral finance considerations such as loss aversion and income shortfall aversion. The use of the VLCM enables cost-benefit analysis of glide-path customization, evaluation of risk-return trade-offs of various asset and sub-asset allocation choices, and multiple portfolio analytics of the probability of success and odds of income sufficiency.
- Based on VLCM's analytical framework, we find that risk-aversion levels are the dominant factor behind the broad stock-bond split in the glide path, affecting both glide-path slope and ending allocation.

Figure 1. The VLCM process



Source: Vanguard.

Appendix 5. Investor characteristics

Details of the investor characteristics needed to model wealth outcomes in VLCM are as follows.

Custom characteristics

Characteristic	Description	25th	50th	75th
Risk aversion	Risk aversion can be defined as aversion to uncertainty of outcomes. In other words, a risk-averse investor dislikes uncertain outcomes and prefers a degree of certainty. A low-level risk-averse investor would tolerate uncertainty for a better outcome, and an extremely risk-neutral investor would only care about the best outcomes and be indifferent toward uncertainty.	Low	Medium	High
DB plan income	DB plan income is computed by: • Defined benefit replacement ratio: the typical proportion of a final-year salary that a defined benefit will cover, or • Defined benefit formula: the calculation used by the plan sponsor to contribute to a DB plan.	NA	No DB	Yes (20% RR)
Savings rate	Savings rates typically differ by age; investors save relatively less when they are young and more as they approach retirement. Based on information from Alling et al. (2020), the average investor starts out at age 25 with an 8.8% savings rate and ends at age 65 with a 12.0% savings rate, including an employer match.	5.0%–8.0%	8.8%–12.0%	12.7%–16.8%
Starting salary	The investor's starting salary in dollars.	$26,600	$45,000	$65,000
Retirement age	The average age at which the investor leaves the workforce.	NA	65	70
Starting age	Starting age of investor.	25	25	25
Behavioral preferences	• Myopic loss aversion: how the investor perceives short-term market losses. • Income shortfall aversion: how the investor perceives a fall in income below a physiological threshold. • Time preference: the preference to consume now versus later.	Low	Medium	High
Spending	Spending (or consumption) is a direct function of wealth. The model will spend down wealth by following a range of different spending rules. These include: • Fixed-dollar spending (hybrid): The investor will spend a fixed dollar amount each year in retirement until wealth hits below a threshold, in which case spending is converted to percentage-of-portfolio. • RMD spending: The investor will draw down the portfolio in accordance with the Required Minimum Distribution starting at age 72. Post-retirement and before age 72, the investor will spend the first RMD percentage for age 72. • Percentage-of-portfolio spending: The investor will spend a fixed percentage every year in retirement. • Dynamic spending with a ceiling and a floor: The same as percentage-of-portfolio spending except that the investor cannot spend more than a specified ceiling or less than a specified floor of last year's spending.	Fixed-dollar spending (hybrid) with a replacement ratio of 0.81	Fixed-dollar spending (hybrid) with a replacement ratio of 0.86	Fixed-dollar spending (hybrid) with a replacement ratio of 0.91

Source: Vanguard.

https://institutional.vanguard.com/VGApp/iip/site/institutional/researchcommentary/article/InvResVanguardLifeCycleInvestingModel

aws

AI/ML VLCM

A Python version of VLCM has been developed for this research

Inputs	AI/ML VLCM	Outputs (Glide Paths CE)

• 27
• 8 combinatorial
  • Risk aversion
  • Loss aversion
  • Saving rate
  • Spending ratio (SR)
  • Portfolio value
  • Current salary
  • Retirement age
  • Age
	• VLCM-driven Certainty Equivalence (CE) calculator developed
	  • Utility function is a function of consumer and stability
	• Vanguard Capital Markets Model (VCMM) scenarios
	  • 10K
	  • US equity/bond
	  • International equity/bond
	  • Cash
	  • Inflation
	• Wage curve considered
		A list of 1,198 Glide Paths with their associated CEs

aws

Personalized Glide Path

Personalized Glide Path (PGP) leverages the VLCM to create personalized Glide Paths for individual investors based on each client's unique characteristics

- Risk aversion
- Loss aversion
- Saving rate
- Spending ratio
- Age
- Retirement age
- …

aws

Business goal

	Comparison Glide Path	gp 353	Current Age	55	Retirement Age	65			
			Current Salary	$50,000.00	Portfolio Spend	$20,000.00			
	Spending Ratio and Risk Aversion								
	6	6	6	6	6	6	6	6	6
Savings	10	15	17.5	20	22	24	26	28	30
2.00%	20.8	5	0.8	0.3	0	0.3	1.6	1.2	1.6
6.00%	19.4	4.1	0.8	0	0.5	0.9	1.6	1.8	1.8
10.00%	15.8	2.3	0	0	0.9	0.8	1.8	1.8	1.2
14.00%	12.2	0.7	0	0.2	0.3	1.6	1.6	1.8	1.6
18.00%	9.5	0.3	0.5	0.5	1.1	1.6	2.1	1.6	2.1
22.00%	6.8	0.8	0.2	0.9	1.6	2	1.2	1.6	34.4

			Spending Ratio		
			Low	Medium	High
			15	20	26
Savings	Low	6%	gp 75	gp 353	gp 356
Rate	Medium	10%	gp 75	gp 353	gp 542
	High	18%	gp 351	gp 135	gp 181

- Each point in the table is a certainty fee equivalence (CFE) value
- CFE = max Glide Path CE – baseline Glide Path CE
- Each profile has a baseline Glide Path
- Differentiation in CFE happens before reaching to corner points
- We need to build these surfaces for more than 9,000 personas

aws

Accuracy requirement

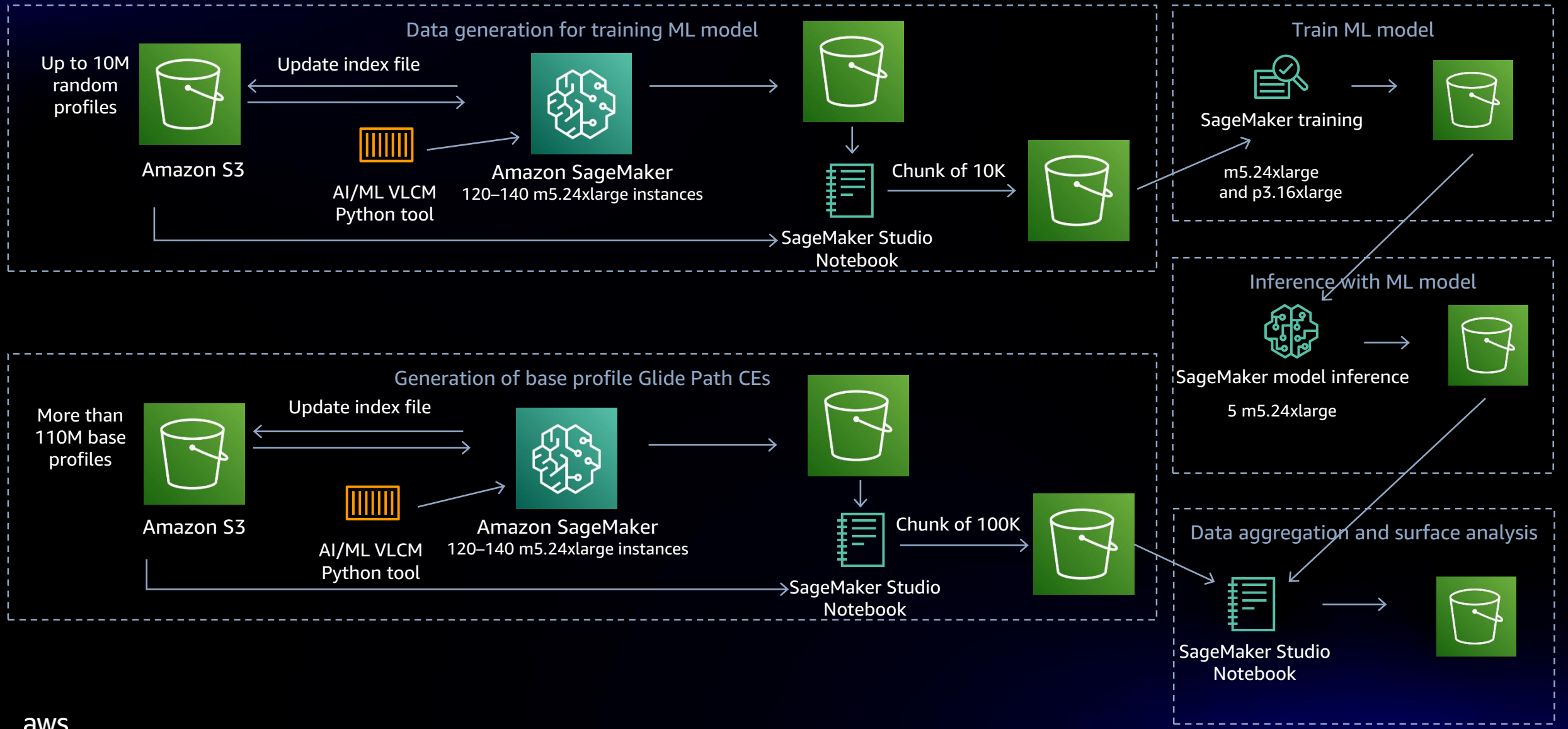
- Sample AI/ML VLCM output

PID	gp	sAge	retAge	varEndAge	pVal	cSal	rInc	rIncStart	rDB	...	gp 1189	gp 1190	gp 1191	gp 1192	gp 1193	gp 1194	gp 1195	gp 1196	gp 1197	gp 1198
[redacted]	VLCM	44	70	100	240025.116935	104877.386567	0.4	70	0	...	0.02950	0.02816	0.02691	0.02584	0.02542	0.02529	0.02355	0.02134	0.02022	0.01966
[redacted]	VLCM	65	70	100	595099.866988	112793.757958	0.4	70	0	...	0.02896	0.02658	0.02389	0.02162	0.02084	0.02072	0.02005	0.01699	0.01520	0.01483
[redacted]	VLCM	46	69	100	957844.592582	108541.164047	0.4	70	0	...	0.00904	0.00901	0.00890	0.00872	0.00858	0.00856	0.00822	0.00748	0.00691	0.00670
[redacted]	VLCM	37	62	100	418239.254509	90171.175442	0.4	70	0	...	0.02408	0.02125	0.01980	0.01956	0.01936	0.01933	0.01423	0.01245	0.01209	0.01090
[redacted]	VLCM	53	69	100	957844.592582	112793.757958	0.4	70	0	...	0.01467	0.01284	0.01246	0.01212	0.01204	0.01200	0.01161	0.01057	0.00992	0.00962

- 1 basis point is 0.0001 difference in Glide Path CE
- Model accuracy of 3 basis points is the equivalent of around 98.5% accuracy
- Model accuracy of 1.5 basis points is the equivalent of around 99.25% accuracy

aws

Data generation architecture

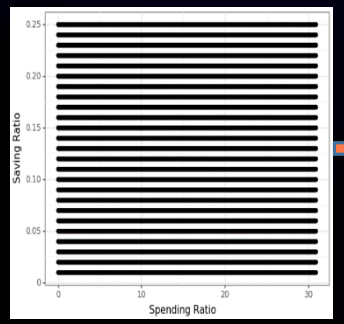


Data generation with Amazon SageMaker Processing

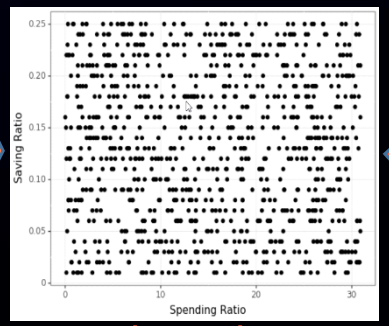


Profile generation

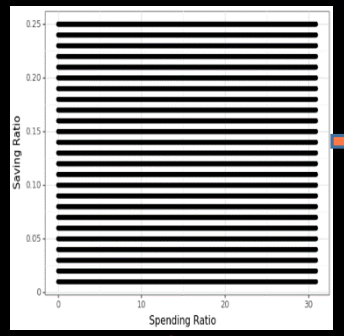
TAKE RANDOM SAMPLE FROM EACH PROFILE (AROUND 4%–6%) AND AGGREGATE THEM



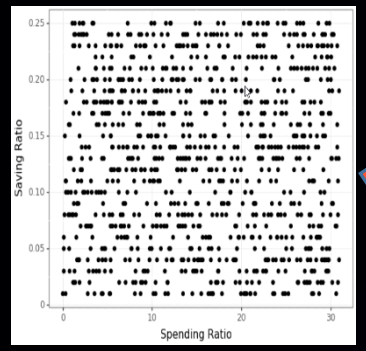
Profile i



Random selection



Profile j



Random selection

Aggregate randomly selected inputs

```
HM = []
Inputlist = []
for i in range(len(allprofiles)):
    with open(allprofiles[i],'rb') as f:
        tmp = pickle.load(f)
    hmany = int(0.005 * len(tmp))
    # print(hmany)
    HM.append(hmany)
    randomlist = random.sample(tmp, hmany)
    Inputlist.extend(randomlist)
```

Input list contains at most 10M records with a size around 1 GB

aws

Amazon SageMaker Processing

- Goal – generate 10 million data points with large instances of Amazon SageMaker
- We used ml.m5.24xlarge instance
- Reason behind selecting ml.m5.24xlarge instance is that we need an instance type with the highest number of CPU and big-memory size and reasonable price
- We used 120 instances of ml.m5.24xlarge for main profile data generation
- Total compute power for this part – 11,520 CPUs and 46,320 GB of memory!

aws

Amazon SageMaker Processing

HOW TO SYNCHRONIZE DATA GENERATION ACROSS INSTANCES

Divide list of available indexes into chunks of 25K personas and save them in Amazon S3 (25K is a Vanguard decision)

```
available_indexes = list(np.arange(0,len(inputlist),1))
random.shuffle(available_indexes) # Why? some age range need more computation, we want to distribute workload amongs instances uniformly
```

```
chunks = [available_indexes[x:x+25000] for x in range(0, len(available_indexes), 25000)]
```

```
key_index = os.path.join(folder,'Available_indexes_Uniform.sav')
pickle_byte_obj_taken = pickle.dumps(chunks)
s3_resource = boto3.resource('s3')
s3_resource.Object(bucket,key_index).put(Body=pickle_byte_obj_taken)
```



Create a chunk tracker and save it in Amazon S3

```
number_of_chunks = list(np.arange(0,len(chunks),1))
```

```
key_index = os.path.join(folder,'Available_chunks_Uniform.sav')
pickle_byte_obj_taken = pickle.dumps(number_of_chunks)
s3_resource = boto3.resource('s3')
s3_resource.Object(bucket,key_index).put(Body=pickle_byte_obj_taken)
```

aws

Amazon SageMaker Processing

Whenever each instance starts processing, it first looks at the available chunks and selects a chunk that is available for processing, then removes that from Available_chunks_Uniform.sav

```
if __name__ == "__main__":

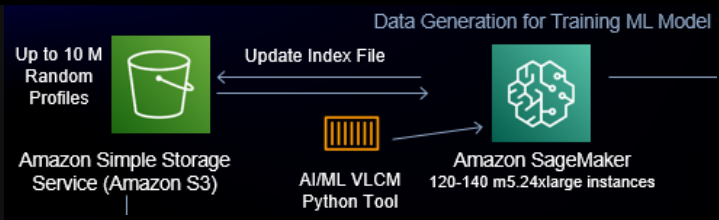
    sleep(randint(1,7200))

    tmp = pd.read_pickle(os.path.join('s3://',bucket,folder,'Available_chunks_Uniform.sav')) # 'Available_chunks_Uniform.sav'
    bach_n = tmp.pop(random.randrange(len(tmp)))

    key_index = os.path.join(folder,'Available_chunks_Uniform.sav')
    pickle_byte_obj_taken = pickle.dumps(tmp)
    s3_resource = boto3.resource('s3')
    s3_resource.Object(bucket,key_index).put(Body=pickle_byte_obj_taken)

    chunks = pd.read_pickle(os.path.join('s3://',bucket,folder,'Available_indexes_Uniform.sav'))  # 'Available_indexes_Uniform.sav'

    current_indexes = chunks[bach_n]
    print("---I got data and will start data generation---")
    run(current_indexes)
```



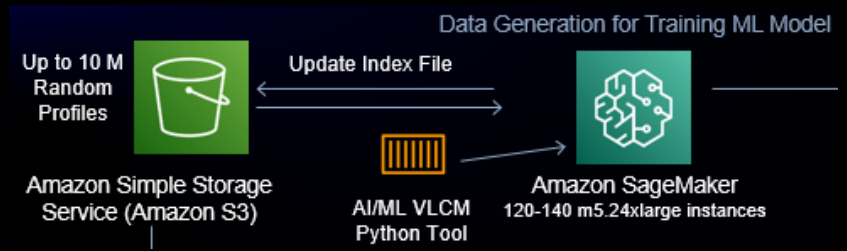
Amazon SageMaker Processing

PARALLEL PROCESSING IN EACH INSTANCE AND SAVING .SAV FILE FOR EACH INDEX FILE INTO AMAZON S3

Divide 25K input size for each instance into 96 parts and send each part to a CPU

```
def run(indexes_orig, N =  95):
    randomState =  np.random.RandomState()
    randomState.seed(vcmm_seed)
    time_dict = {}
    for n in range(N):
        indexes = indexes_orig[n::N]
        rangelist = [2]
#         print("test indexes:", indexes)
        p = Process(target=work, args=(dfSett, indexes, rangelist, n,))
        p.start()

    p.join()
```



Run the job for each index and save output as gp_ce_key_index.sav in S3

```
def work(dfSett, indexes, rangelist, proc):

    procTimeDict = {}
    indset = set()
    for i, j in product(indexes, rangelist):
        subfolder = 'CE_%s_%s'%(str(RES_FOLD_NUM*(i//RES_FOLD_NUM)).zfill(7), str(RES_FOLD_NUM*(1 +(i//RES_FOLD_NUM))).zfill(6))
        #bucket_name = 's3://vgi-ead-eng-us-east-1-adviceai-sandbox'
        #folder_name = 'Ayoub/PGP/Phase2Data/result'
        filename = join(subfolder, 'gp_ce_key%s.sav'%(str(i).zfill(6)))

        tmp1 = dt.datetime.now()
        dfSett = dfSettMaker(inputlist[i],dfSett)

        RunInput(dfSett,dfFixMort,dfRMD,dfWgCurve,dfUSEqPr,dfUSEqIc,dfUSBnPr,dfUSBnIc,dfInEqPr,dfInEqIc,
         dfInBnPr, dfInBnIc, dfInflation,dfCash,dictGPNames,dictGPs,dictGPNamesInv, filename = filename, endRange = 10001,
         index = i)
```

```
key = os.path.join(folder,filename)

pickle_byte_obj = pickle.dumps(ceNet)

s3_resource = boto3.resource('s3')
s3_resource.Object(bucket,key).put(Body=pickle_byte_obj)
```

aws

Amazon SageMaker Processing

CREATE DOCKER IMAGE AND PUSH THAT TO AMAZON ECR
(WE SELECT INSTANCE TYPE, NUMBER OF INSTANCES, AND DURATION TO RUN SAGEMAKER PROCESSING IN THIS STEP)

```
region = boto3.session.Session().region_name

test_instance_type = 'ml.m5.24xlarge' ### <--- Enter your desired instance type
test_instance_count = 120  ### <--- Enter your desired instance number
test_job_name = 'adviceai-PGP-15-instance-Uniform-V5-1203'
test_image = '            .dkr.ecr.us-east-1.amazonaws.com/adviceai-sm-processing-container-pgp:Second_Gen_Phase2_Uniform_V5'

# network_config = NetworkConfig(security_group_ids=security_group_ids,
#                                subnets=subnets,
#                                encrypt_inter_container_traffic=True
#                               )

processor = Processor(
                      image_uri=test_image,
                      role = 'arn:aws:iam::            :role/SMTrainingJobEx-ADVICEAINOTEBOOK-us-east-1', # Subject to change
                      instance_type= test_instance_type,
                      instance_count= test_instance_count,
                      max_runtime_in_seconds = 432000, # The maximum is here, 5 days to stop
                      base_job_name = test_job_name,
#                       network_config = network_config,
                      #volume_kms_key='dd449092-68f3-49e2-9ab2-a8a2730e21e3',
                      output_kms_key="alias/VGI-KMS-S3",
                      entrypoint= ['python3', './MultiProc_Mainv2.py'] # Enter your entrypoint here
                     )
```

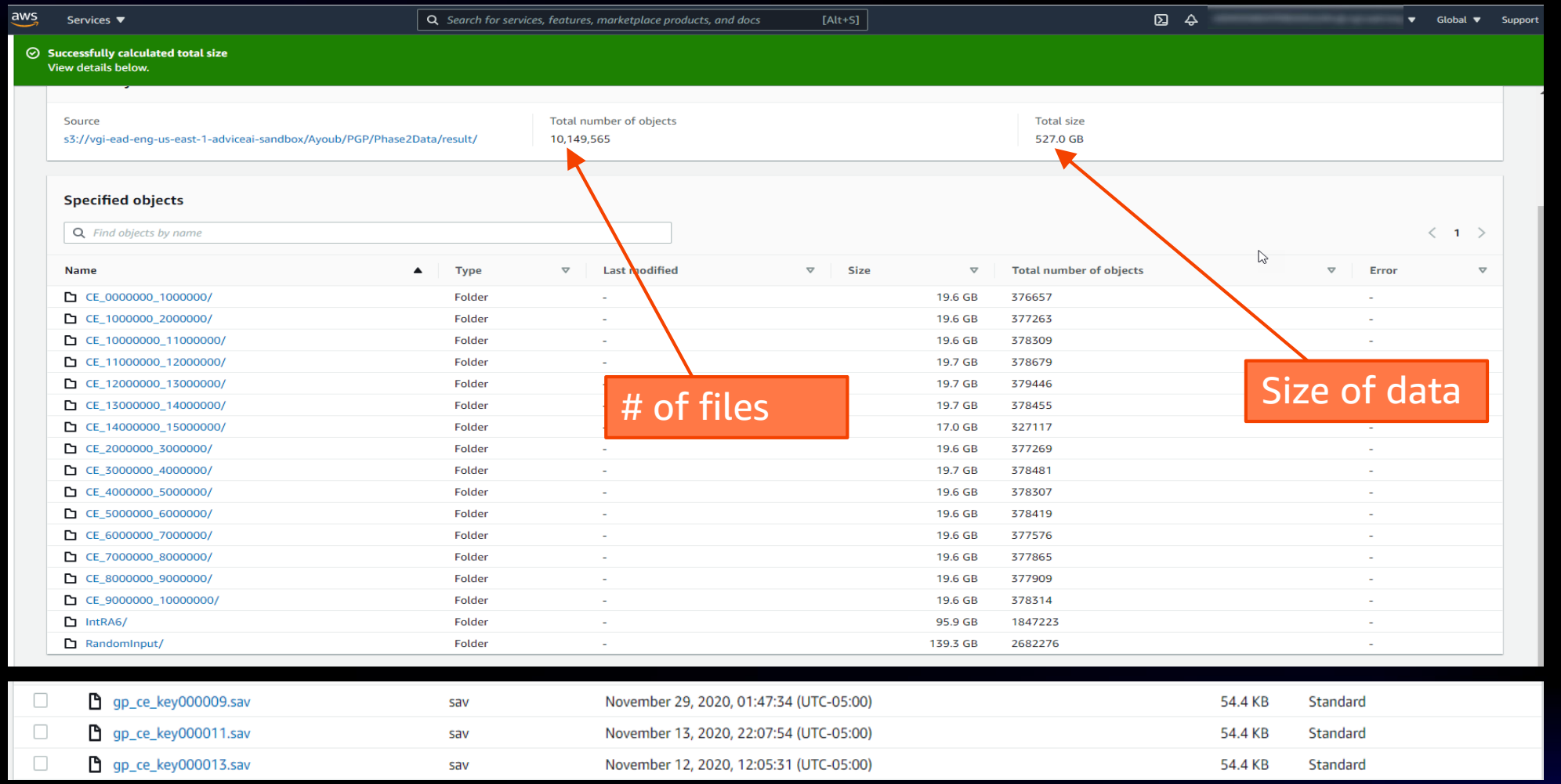
aws

Data aggregation with Amazon SageMaker Studio



Data aggregation

AGGREGATE DATA FOR TRAINING



aws

Data aggregation – Processing 525 GB of data

Read/write to Amazon S3 in parallel with Amazon SageMaker Studio (instance ml.m5.24xlarge)

Read .sav files and aggregate them

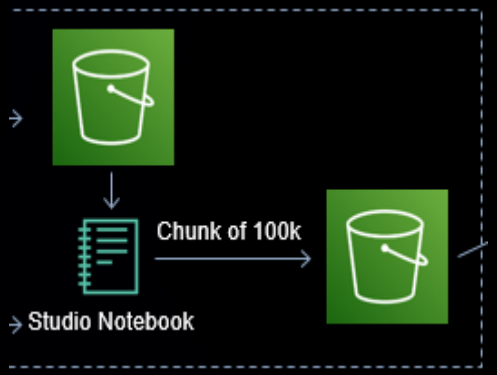
Read in files with pool function and utilize 96 CPUs of 24xlarge

```
[ ]: def FileReader(tmp):
         CEs= pickle.load(s3_file.open(tmp))
         index = int(re.match('.*key(\d+)\.sav', tmp).group(1))
         return index,CEs
```

```
[ ]: def ReadMerge(cbatch):

         start_time = time.time()
         filenames = cbatch  # CE_Random[1:]
         pool = Pool()
         result={}
         for CEs in enumerate(pool.imap(FileReader, filenames)):
             result[CEs[1][0]] = CEs[1][1]
         pool.close()
         pool.join()

         finish_time = time.time()
         print(finish_time - start_time)
         return(result)
```



aws

Data aggregation – Processing 525 GB of data

Merge aggregated .sav files with profile features

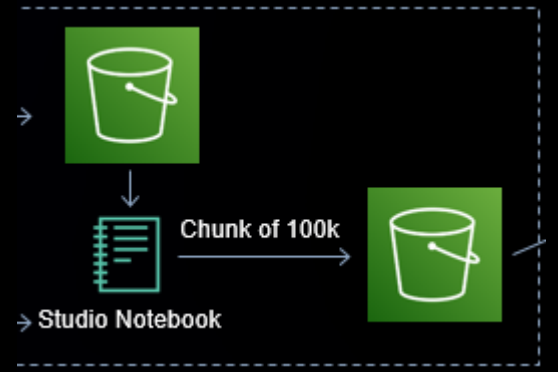
```
def dfMaker(inputlistname,newdatafile):
    # chunks = [newdatafile[x:x+10000] for x in range(0, len(newdatafile), 10000)]
    # today = date.today()
    inputcolumns = ['gp','sAge', 'retAge', 'varEndAge','pVal','cSal','rInc','rIncStart','rDB','rDBStart','strType','tCons','fltDiscBase',\
             'fltSavingsBase','strSav','fltSavInc','intRA','fltMLA','fltTLA','yrCons','arrFixMort','blnDetail','retGoal','appMLA',\
             'appSolver','chgVar','uWt','SpendingMultiple']
    df2 = pd.DataFrame(inputlistname, columns=inputcolumns)
    df2 = df2.reset_index()
    df2 = df2.rename(columns={"index":"PID"})

    with open('/root/PGP_Phase2/pgp/Data/wgCurve.pkl','rb') as f:
        dfWgCurve = pickle.load(f)

    dfWgCurveDict = dfWgCurve['Wage Scale'].to_dict()
    # pVal = [s*(tCons[0]-rInc[0])*(dfWgCurveDict[70]/dfWgCurveDict[25])*50000*(1.011**45) for s in spendingRatio]
    df2["spendingRatio"]= (df2['pVal']*dfWgCurveDict[25])/(0.4*dfWgCurveDict[70] * 50000*(1.011**45))

    rest =  ReadMerge(newdatafile)
    df1 = pd.DataFrame(rest)
    df1 = df1.T
    df1 = df1.reset_index()
    df1 = df1.rename(columns={"index":"PID"})
    df = pd.merge(df2,df1,how='inner',on='PID')

    return(df)
```



aws

Data aggregation – Processing 525 GB of data

Write data into Amazon S3; write data in CSV format for size 10K; total of 1,015 CSVs

```
def FileWriter(tmp):
    today = date.today()
    now = datetime.datetime.now()
    # filename = 'Local_Search_'+str(now.year)+"-"+str(now.month)+"-"+str(now.day)+"-"+str(now.hour)+":"+str(now.minute)+":"+str(now.second)+
    tt = tmp[1]
    filename = "df"+"_"+str(now.year)+"-"+str(now.month)+"-"+str(now.day)+"-"+str(now.hour)+":"+str(now.minute)+":"+str(now.second)+"_"+str(t
    pfile = '/root/PGP_Phase2/AggregatedData_Chunk/'+filename+'.csv'
    tt.to_csv(pfile)
    return(tmp[0])
```

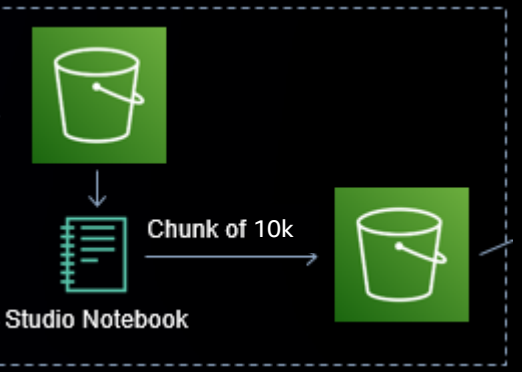
```
chunks = [N_P_Int[x:x+100000] for x in range(0, len(N_P_Int), 100000)]
```

```
for i in range(len(chunks)):
    df      = dfMaker(inputlist_IntRA,chunks[i])

    n = 10000  #chunk row size
    list_df = [df[i:i+n] for i in range(0,df.shape[0],n)]
    list_df = [(i,list_df[i]) for i in range(len(list_df))]

    start_time = time.time()
    filenames = list_df # CE_Random[1:]
    pool = Pool()
    result=[]
    for file in enumerate(pool.imap(FileWriter, filenames)):
        result.append(file)
    pool.close()
    pool.join()

    finish_time = time.time()
    print(i)
    print(finish_time - start_time)
```



aws

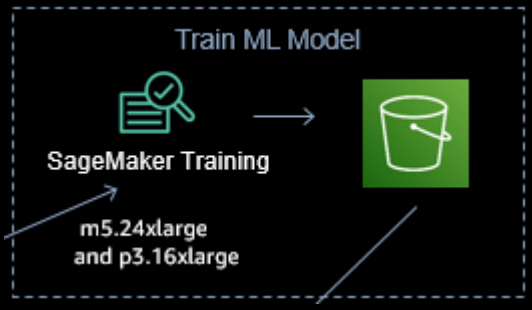
Model training with Amazon SageMaker



Model training

MODEL EXPERIMENTATIONS

- Single output regression – put Glide Path as feature and regress CE
  - Apply GP dimensionality reduction and more complicated feature engineering
  - Experimented different methods including boosting, random forest, simple regression model and neural network on this data set
- Multiple output regression models
  - Numerous deep neural network architectures and k-nearest neighbors (k-NN)
  - Chained multiple output regression
- Ranking models
  - Several ranking models



aws

Model training

Model	Model output	Model summary	Accuracy (basis points)	Accuracy
Deep neural network	CE of 1,198 Glide Paths	Regular multi-output neural net model with millions of parameters	9	95.3%
RNN	CE	RNN model with GP sequence as feature	18	90.36%
XGBoost	CE	Regression model with PCA1, PCA2, and PCA3 of Glide Path as feature	10	93.46%
Weighted k-NN	CE of 1,198 Glide Path	Weighted k-NN with 1,198 Glide Path CE as output	3.0–3.2	98.5%
k-NN	CE of 1,198 Glide Path	Regular k-NN with 1,198 CE as output; this is our baseline model	4.82	97.5%

- For DNN, RNN, and XGBoost, we used a training job (DNN and RNN ml.p3.16xlarge instance)
- For weighted k-NN, we set up a processing job with 5 instances of ml.m5.24xlarge

Data size impact on model accuracy

We captured the accuracy of a model that generated 1,198 Glide Path CE with various dataset sizes

Data size (profiles)	Model accuracy (basis point)	Data generation time (days)
160K	8.5–9	<1
500K	5.8–6.1	1
1M	4.1–4.3	2–3
2M	3–3.15	4–5
6M	2.1–2.3	12–13
10.1M	1.5–1.65	21

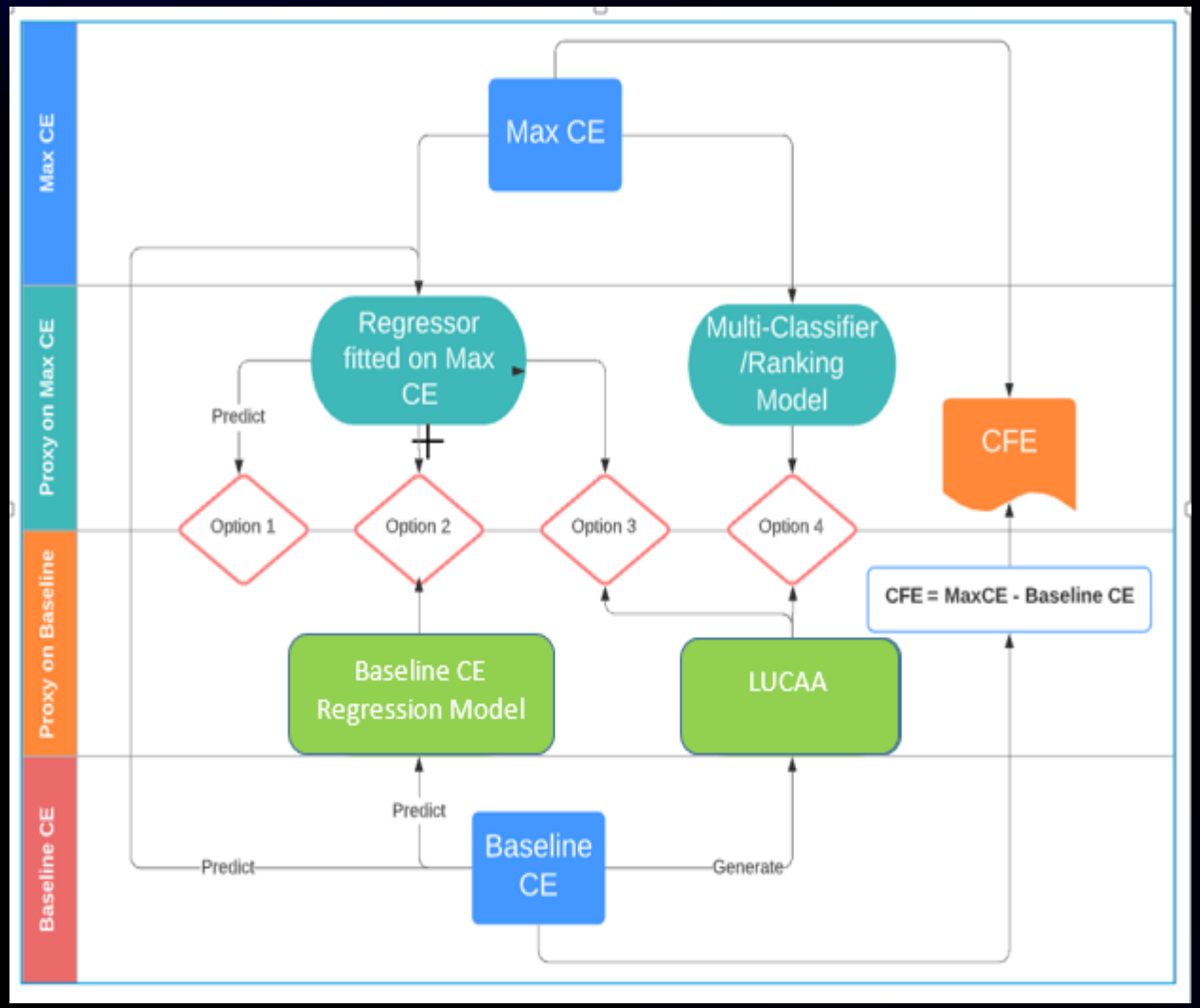
The models need huge amount of data to improve accuracy after reaching 3 basis points in accuracy

aws

Narrow down the available options



Modeling framework



OPTION 1
Regressor proxy on all 1,198 CEs and use the fitted model to predict both additional 1,198 CEs and baseline CE

REQUIRED ACCURACY
Total error should be 1.5 basis points or less

OPTION 2
Regressor to predict 1,198 CEs and a separate regressor on base CE prediction.

REQUIRED ACCURACY
Total error should be 3 basis points or less

OPTION 3
Regressor on all 1,198 CEs and generate ground truth baseline CE with the AI/ML VLCM

REQUIRED ACCURACY
Model error should be 3 basis points or less

OPTION 4
Multi-class/ranking model to predict top CE Glide Paths and generate ground truth baseline CE with the AI/ML VLCM

REQUIRED ACCURACY
Model error should be 3 basis points or less

aws

Cost/time comparisons of different options

Since we can tackle the problem in 4 different ways to get the desired accuracy, the distinguishing factors among them is cost and time of data generation and model training

Option	Description	Max CE required data	Base CE required data	Data gen cost	Training cost	Data gen time (days)
Option 1	One model	10M	10M	~350K	10K	20–21
Option 2	Two models	3M	35M	~150	~15	13–14
Option 3	Max CE model, base CE AI/ML VLCM	2M	110M	~160K	5K	10–11
Option 4	Ranking model	Stopped	Stopped	Stopped	Stopped	Stopped

- Ranking model didn’t reach a model that is comparable in terms of accuracy with other regression models
- Option 2 is also an interesting choice; however, it needs the training and maintenance of two models

Trade-off between data generation and model exploration

- We can either generate less data and experiment as much we can or generate more data and reach the desired accuracy with less model experimentation
- Cost/time analysis of different data sizes for the model accuracy of 1.5 basis points

Approach index	Approach name	Amount of data for training	Labor costs	Technology cost (data generation and training)	Accuracy is (will be) satisfactory?	Decision
A1	Small data	< 100K	2M (2 years of 5 scientists + PM + DE)	DG: 20K TC: ~100K	Not guaranteed	Risky decision
A2	Moderate data	500K–1M	1M (1 year of 5 scientists + PM + DE)	DG: 100K TC: ~50K	Might	Worth it to investigate
A3	Large data	10M	400–500K (9 months of 2 scientists + PM)	DG: 300–400K TC: ~20K	Guaranteed	Technology cost looks scary

DG: data generation

TC: training cost

DE: data engineer

PM: project manager

aws

Model inference with Amazon SageMaker Processing

aws

Inference and data generation

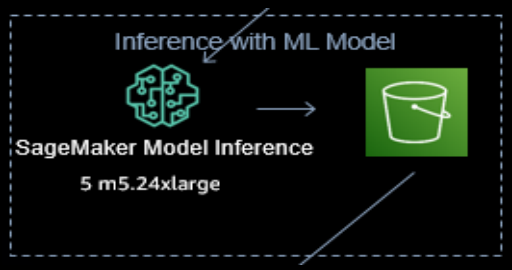
- Weighted k-NN was the final model
- We partitioned data based on the age feature with 5 years of overlap

Age range for training	Age range for inference
25–35	25–30
25–40	31–35
30–45	36–40
35–50	41–45
40–55	46–50
45–60	51–55
50–65	56–60
60–70	66–70

aws

Inference and data generation with model

- Amazon SageMaker Processing with 9 instances for inference, but smaller instances
- Each instance read the specific profile and model (results stored in Amazon S3)




```
def InputProfiles():
    fs = s3fs.S3FileSystem()
    s3_file = S3FileSystem()
    path = 'vgi-ead-eng-us-east-1-adviceai-sandbox/Ayoub/PGP/Phase2Data/Prediction/Profiles'
    allprofiles= fs.ls(path)
    allprofiles = allprofiles[1:]

    profile_25to30 = ["s3://"+x for x in allprofiles if ('Profiles/25_' in x) or ('Profiles/26_' in x) or ('Profiles/27_' in x)
or ('Profiles/28_' in x) or ('Profiles/29_' in x) or ('Profiles/30_' in x)]
    profile_31to35 = ["s3://"+x for x in allprofiles if ('Profiles/31_' in x) or ('Profiles/32_' in x) or ('Profiles/33_' in x)
or ('Profiles/34_' in x) or ('Profiles/35_' in x)]
    profile_36to40 = ["s3://"+x for x in allprofiles if ('Profiles/36_' in x) or ('Profiles/37_' in x) or ('Profiles/38_' in x)
or ('Profiles/39_' in x) or ('Profiles/40_' in x)]
    profile_41to45 = ["s3://"+x for x in allprofiles if ('Profiles/41_' in x) or ('Profiles/42_' in x) or ('Profiles/43_' in x)
or ('Profiles/44_' in x) or ('Profiles/45_' in x)]
    profile_46to50 = ["s3://"+x for x in allprofiles if ('Profiles/46_' in x) or ('Profiles/47_' in x) or ('Profiles/48_' in x)
or ('Profiles/49_' in x) or ('Profiles/50_' in x)]
    profile_51to55 = ["s3://"+x for x in allprofiles if ('Profiles/51_' in x) or ('Profiles/52_' in x) or ('Profiles/53_' in x)
or ('Profiles/54_' in x) or ('Profiles/55_' in x)]
    profile_56to60 = ["s3://"+x for x in allprofiles if ('Profiles/56_' in x) or ('Profiles/57_' in x) or ('Profiles/58_' in x)
or ('Profiles/59_' in x) or ('Profiles/60_' in x)]
    profile_61to65 = ["s3://"+x for x in allprofiles if ('Profiles/61_' in x) or ('Profiles/62_' in x) or ('Profiles/63_' in x)
or ('Profiles/64_' in x) or ('Profiles/65_' in x)]
    profile_66to70 = ["s3://"+x for x in allprofiles if ('Profiles/66_' in x) or ('Profiles/67_' in x) or ('Profiles/68_' in x)
or ('Profiles/69_' in x) or ('Profiles/70_' in x)]


    return
profile_25to30,profile_31to35,profile_36to40,profile_41to45,profile_46to50,profile_51to55,profile_56to60,profile_61to65,profile
_66to70
```

```
    if model_number == 1:
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_25_30.pickle")
        profiles = profile_25to30
    elif model_number ==2:
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_31_35.pickle")
        profiles = profile_31to35
    elif model_number == 3 :
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_36_40.pickle")
        profiles = profile_36to40
    elif model_number == 4 :
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_41_45.pickle")
        profiles = profile_41to45
    elif model_number == 5:
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_46_50.pickle")
        profiles = profile_46to50
    elif model_number == 6:
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_51_55.pickle")
        profiles = profile_51to55
    elif model_number == 7 :
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_56_60.pickle")
        profiles = profile_56to60
    elif model_number ==  8:
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_61_65.pickle")
        profiles = profile_61to65
    else :
        model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-
sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_66_70.pickle")
        profiles = profile_66to70
```

aws

Inference and data generation with model

```
# Main Function:
if __name__ == "__main__":

    sleep(randint(1,200))

    bucket = 'vgi-ead-eng-us-east-1-adviceai-sandbox'
    folder = 'Ayoub/PGP/Phase2Data/Prediction'


    available_model = pd.read_pickle(os.path.join('s3://',bucket,folder,'available_model.sav'))
    model_number = available_model.pop(0)

    key_index = os.path.join(folder,'available_model.sav')
    pickle_byte_obj_taken = pickle.dumps(available_model)
    s3_resource = boto3.resource('s3')
    s3_resource.Object(bucket,key_index).put(Body=pickle_byte_obj_taken)

    profile_25to30,profile_31to35,profile_36to40,profile_41to45,profile_46to50,profile_51to55,profile_56to60,profile_61to65,profile_66to70 =
InputProfiles()
```

```
if model_number == 1:
    model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_25_30.pickle")
    profiles = profile_25to30
elif model_number ==2:
    model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_31_35.pickle")
    profiles = profile_31to35
elif model_number == 3 :
    model = pd.read_pickle("s3://vgi-ead-eng-us-east-1-adviceai-sandbox/Ayoub/PGP/Phase2Data/Prediction/Models/Model_36_40.pickle")
    profiles = profile_36to40
elif model number == 4 :
```

```
print("-- I got the model and data for prediction---")
for i in profiles:
    filename =i.split("/")[8]
    X_train,current_df = fileMaker(i)
    predicted = model.predict(X_train)
    maxs = predicted.max(axis=1)
    current_df['maxCE'] = list(maxs)
    # store current_df
    tuples = [tuple(x) for x in current_df.to_numpy()]


    key = os.path.join(folder,filename)
    pickle_byte_obj = pickle.dumps(tuples)
    s3_resource = boto3.resource('s3')
    s3_resource.Object(bucket,key).put(Body=pickle_byte_obj)
```

- Total inference time less than 4 hours!
- Inference cost less than $200!

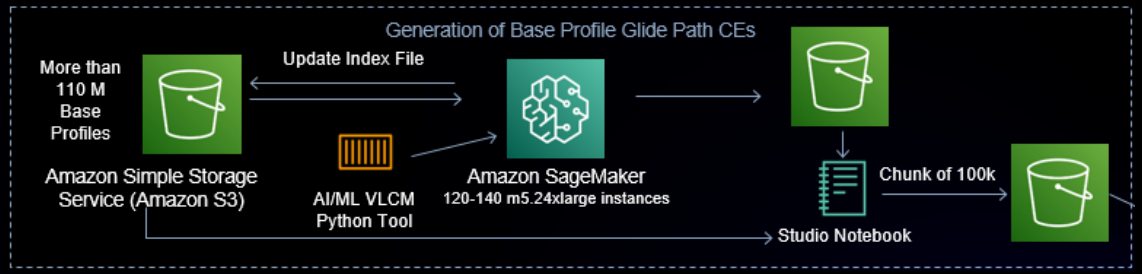
aws

Base profile data generation with Amazon SageMaker Processing



Base profile data generation

- Similar to all Glide Path data generation with Amazon SageMaker Processing except we generate CE for one Glide Path
- Another SageMaker Processing with 120 instances of ml.m5.24xlarge generated all data in less than 24 hours
- Sample of generated data after data aggregation




```
[3]: df.head(5)
```

[3]:

	PID	sAge	retAge	pVal	cSal	fltSavingsBase	intRA	fltMLA	spendingRatio	BaseGP	BaseCE
0	[redacted]	44	70	240025.116935	104877.386567	0.103	9	0.5	5.32	gp 139	0.03215
1	[redacted]	65	70	595099.866988	112793.757958	0.151	4	0.5	13.19	gp 134	0.03715
2	[redacted]	46	69	957844.592582	108541.164047	0.195	13	0.6	21.23	gp 143	0.00880
3	[redacted]	37	62	418239.254509	90171.175442	0.051	13	0.5	9.27	gp 143	0.03333
4	[redacted]	53	69	957844.592582	112793.757958	0.234	4	0.6	21.23	gp 139	0.02911

aws

Profiles CFE builder and surface analysis with Amazon SageMaker Studio

aws

Build profiles CFE

- Simple merge of two datasets
- Leveraged Amazon SageMaker Studio

```
def CFEMaker(i,allprofiles,df):
    inputcolumns = ['gp','sAge', 'retAge', 'varEndAge','pVal','cSal','rInc','rIncStart','rDB','rDBStart','strType','tCons','fltDiscBase',\
                    'fltSavingsBase','strSav','fltSavInc','intRA','fltMLA','fltTLA','yrCons','arrFixMort','blnDetail','retGoal','appMLA',\
                    'appSolver','chgVar','uWt','SpendingMultiple','spendingRatio','maxCE']
    dfm = pd.read_pickle(allprofiles[i])
    dfm = pd.DataFrame(dfm,columns=inputcolumns)

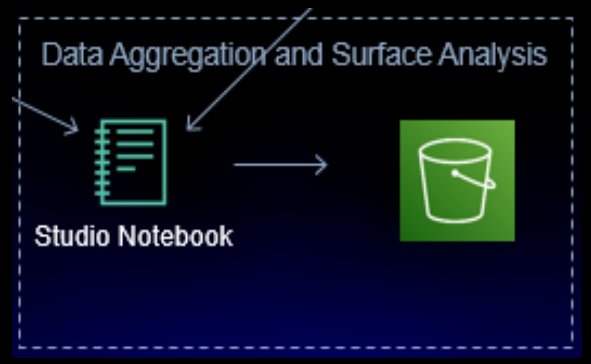
    profile = allprofiles[i].split("/")[8]
    profile = profile.split(".sav")[0]
    profile = profile.split("_")
    profile = [float(i) for i in profile]

    sAge   = int(profile[0])
    retAge = int(profile[1])
    intRA  = int(profile[2])
    fltMLA = profile[3]

    dfm = dfm[['spendingRatio','fltSavingsBase','maxCE']]
    dfb = df[(df['sAge']==sAge) &(df['retAge']==retAge) & (df['intRA']==intRA) &(df['fltMLA']==fltMLA)]

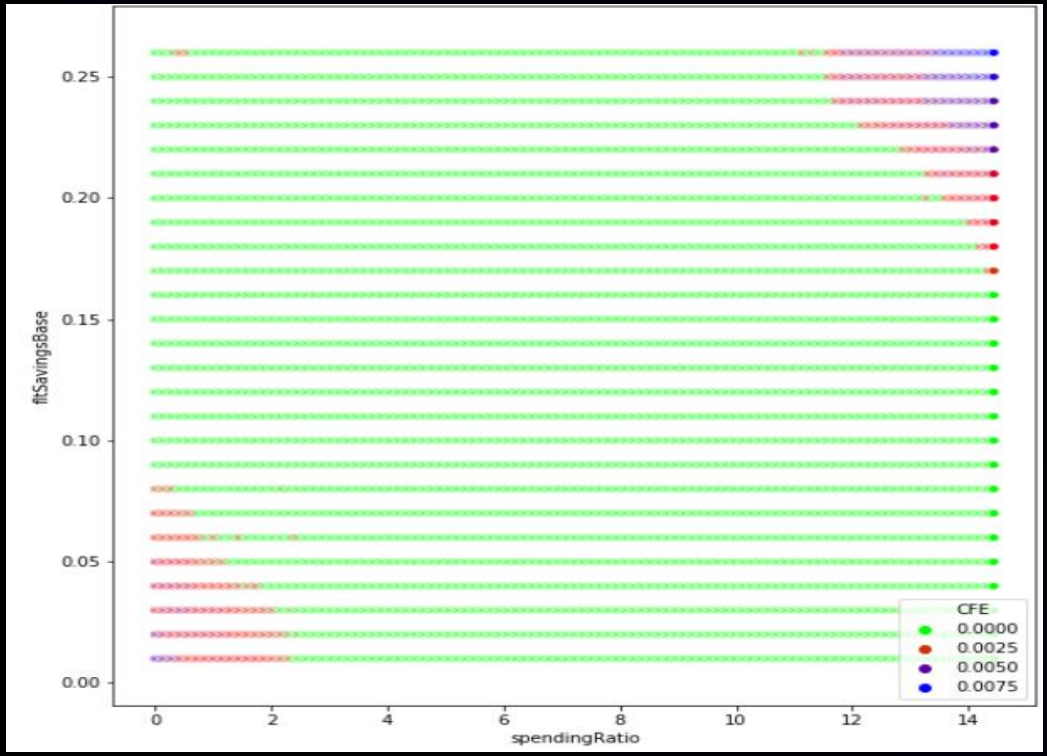
    dfb = dfb[['spendingRatio','fltSavingsBase','BaseCE']]
    dfb = dfb.reset_index(drop=True)
    dff = pd.merge(dfm,dfb, how='inner')
    dff['CFE'] = dff['maxCE']- dff['BaseCE']
    dff[dff < 0] = 0

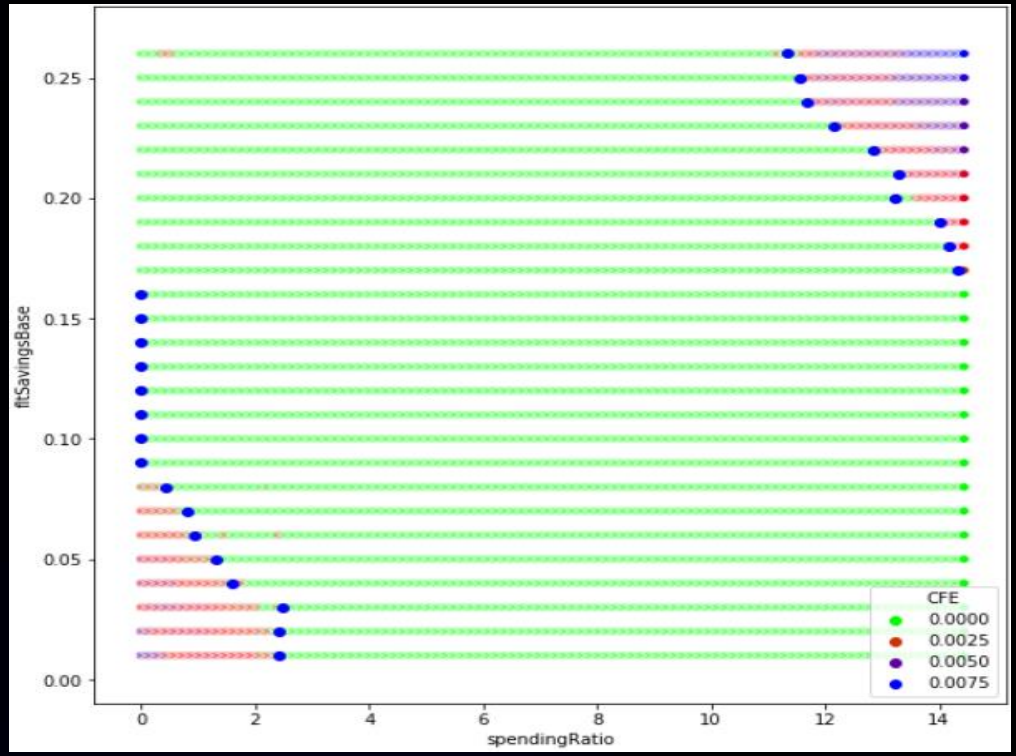
    return dff
```



aws

Surface visualization and breaking point detection





Apply surface analytics techniques

aws

Conclusion

- Vanguard's goal is to **provide a personalized Glide Path** for individual advice clients
- A **personalized Glide Path** relies on heavy computations that consider various market conditions, investor risk/loss aversion, and the financial and demographic situation of an investor
- **AWS computing resources** (large computing power) enabled Vanguard to move toward a personalized Glide Path and **made the unsolvable solvable**
- Vanguard reached their goal at a fraction of the original estimated cost **(4%–6% of estimated cost)** and time **(less than 1 month vs. 1 year for total data generation)** with the availability of ML algorithms and model training on AWS
- Vanguard reduced the computation cost from **$5M to less than $200K**

Questions?

Thank you!





Please complete the session survey

aws