



ENGINEERING GRADUATE SALARY DATASET PREDICTION

GROUP -11 / BHAVANS VIVEKANANDA COLLEGE
Aishwarya| Rishil Ajay kumar | Jai kumar

Abstract

This project develops a predictive model to estimate engineering graduates' salaries based on factors like academic performance, technical skills, work experience, and demographics. It offers insights into salary determinants, aiding educational institutions, recruiters, and students in making informed decisions, promoting transparency in hiring, and helping graduates understand their earning potential in the job market.

Objective

The goal is to create a predictive machine learning model that forecasts engineering graduates' salaries using academic, technical, and demographic data to promote equitable hiring and informed career planning.



CONTENT

• Introduction	4
• Literature Review	5
• Data Preprocessing	8
• Exploratory Data Analysis	12
• Data Modelling & Evaluation	20
• Summary	30
• Appendix	34



Introduction

Predicting salaries for engineering graduates is vital for understanding career opportunities. Accurate predictions help students, institutions, and employers align educational outcomes with industry needs. Factors like academic performance and work experience affect earning potential, making it essential for transparency and informed choices.

The rise of machine learning and data analytics enhances salary prediction by analyzing key data points, providing actionable insights for fair salary decisions and better career guidance in a changing job market.

Literature Review



Literature Review - 1

Author(s): Lee & Kumar

Title: Predicting Engineering Graduate Salaries with Machine Learning

This paper investigates machine learning methods for predicting engineering graduates' starting salaries using academic, demographic, and industry factors. Algorithms such as multiple linear regression, random forests, and support vector machines (SVMs) were employed to analyze the impact of attributes like GPA, specialization, and internship experience on salaries.

The study highlights the effectiveness of ensemble methods like random forests for processing diverse, nonlinear datasets and emphasizes the importance of feature selection for better model interpretability. It concludes that machine learning can help universities and recruiters align graduate skills with market demands.

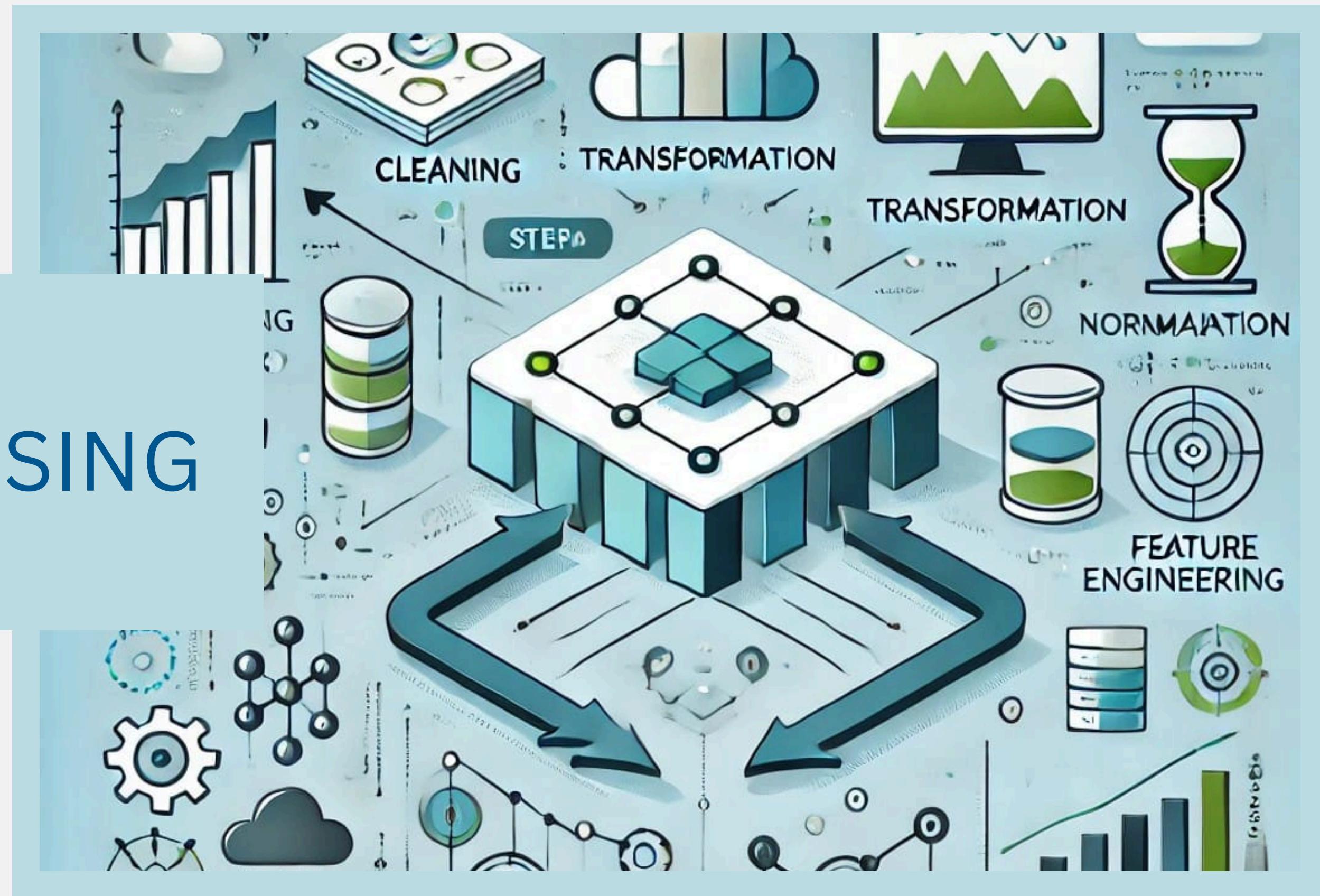
Literature Review - 2

Author(s): Smith & Zhang
Title: Machine Learning Models for Predicting Engineering Graduate Salaries
Published In: International Journal of Educational Data Science, 2021

This study examines the effectiveness of machine learning models in forecasting engineering graduates' salaries based on educational and experiential data. The authors implemented algorithms such as gradient boosting machines (GBM), neural networks, and k-nearest neighbors (KNN) to predict salaries using features like academic performance, certifications, job location, and industry type.

The research highlights the effectiveness of gradient boosting techniques in identifying complex patterns, surpassing traditional statistical models. It also examines the role of industry trends and market dynamics in enhancing prediction accuracy, recommending the integration of external economic indicators for better salary prediction frameworks.

DATA PREPROCESSING



DATA

Dataset: Our Dataset consists of 34 variables and 2998 records

Source: [https://drive.google.com/drive/folders/1vGSRChqSxEH53BgLqhh32F1qNKqfOim?](https://drive.google.com/drive/folders/1vGSRChqSxEH53BgLqhh32F1qNKqfOim?usp=drive_link)
usp=drive_link

Variables:

ID	Gender	DOB	10percen	10board	12gradu	12percen	12board	College1	College1	Degree	Speciali	collegeC	CollegeC	Graduat	English	Logical	Quant	Domain	Comput	Electron	Comput	Mechan	Electrica	Telecom	CivilEng	consoiel	agreeab	extraver	nuerotic	openess	Salary		
604339	f	#####	87.8	cbse	2009	84	cbse	6920	1	B.Tech	instrumentation	73.82	6920	1	Delhi	2013	650	665	810	0.6945	485	366	-1	-1	-1	-1	-1	-0.159	0.3789	1.2396	0.1459	0.2889	445000
988334	m	#####	57	cbse	2010	64.5	cbse	6624	2	B.Tech	computer	65	6624	0	UttarPradesh	2014	440	435	210	0.3423	365	-1	-1	-1	-1	-1	11336	0.0459	1.2396	0.5262	-0.286	110000	
301647	m	#####	77.33	maharas	2007	85.17	amravati	9084	2	B.Tech	electron	61.94	9084	0	Maharashtra	2011	485	475	505	0.8247	-1	400	-1	-1	-1	260	-1	0.51	-0.123	15428	-0.29	-0.288	255000
582313	m	#####	84.3	cbse	2009	86	cbse	8195	1	B.Tech	compute	80.4	8195	1	Delhi	2013	675	620	635	0.99	655	-1	-1	-1	-1	-1	-1	-0.446	0.2124	0.3174	0.2727	0.4805	420000
339001	f	#####	82	cbse	2008	75	cbse	4889	2	B.Tech	biotech	64.3	4889	1	TamilNa	2012	575	495	365	0.2785	315	-1	-1	-1	-1	-1	-1	-1.499	-0.747	-1.07	0.0622	0.1864	200000
609356	f	#####	83.16	icse	2007	77	cbse	10950	1	M.Tech.	instrume	99.93	10950	0	Punjab	2013	535	595	620	0.3761	455	300	-1	-1	-1	313	-1	0.8463	-0.62	-0.759	-0.995	-0.286	440000

Variables

Categorical variables		Continuous variables	
Gender		ID	Logical
DOB		10percentage	Quant
10board		12graduation	Domain
12board		12percentage	ElectricsAndSemicon
Degree		CollegeID	ComputerScience
Specialization		Collegetier	MechanicalEngg
College state		CollegeGPA	ElectricalEngg
		CollegeCityID	TelecomEngg
		CollegeCityTier	CivilEngg
		Graduation Year	conscientiousness
		English	agreeableness
		Computer Programming	extraversion
		nueroticism	openness_to_experience
		salary	

Data Cleaning

Data Cleaning is the process of identifying and correcting errors or inconsistencies in data to ensure it is accurate, complete, and usable for analysis. It involves tasks such as:

- ***Handling missing values: Filling in or removing missing data points.***
- ***Correcting errors: Identifying and fixing typos, incorrect formatting, or outliers.***
- ***Standardizing data: Ensuring consistency in units, date formats, and categories.***
- ***Removing duplicates: Identifying and eliminating redundant records.***
- ***Data transformation: Converting data into a suitable format for analysis.***

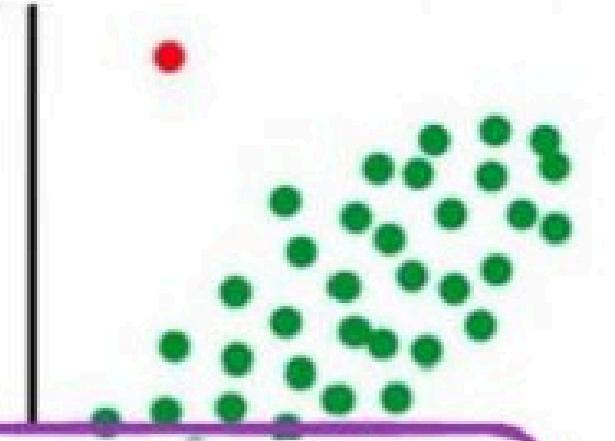


Effective data cleaning improves the quality of the dataset, leading to more reliable insights and better decision-making.

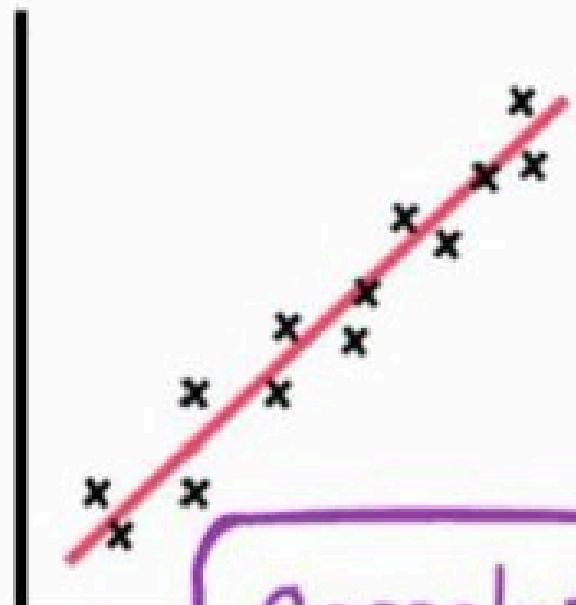
Exploratory Data Analysis



Data distribution

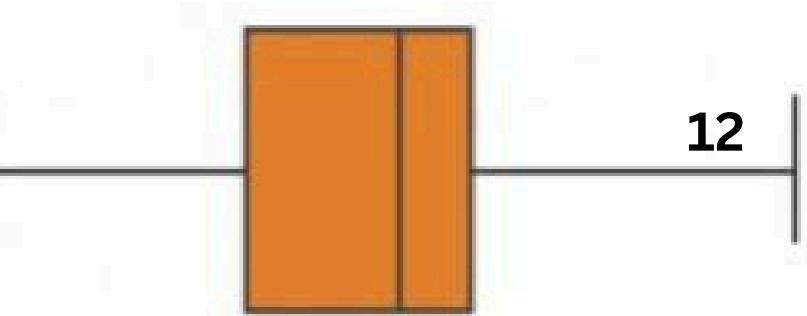


Outliers



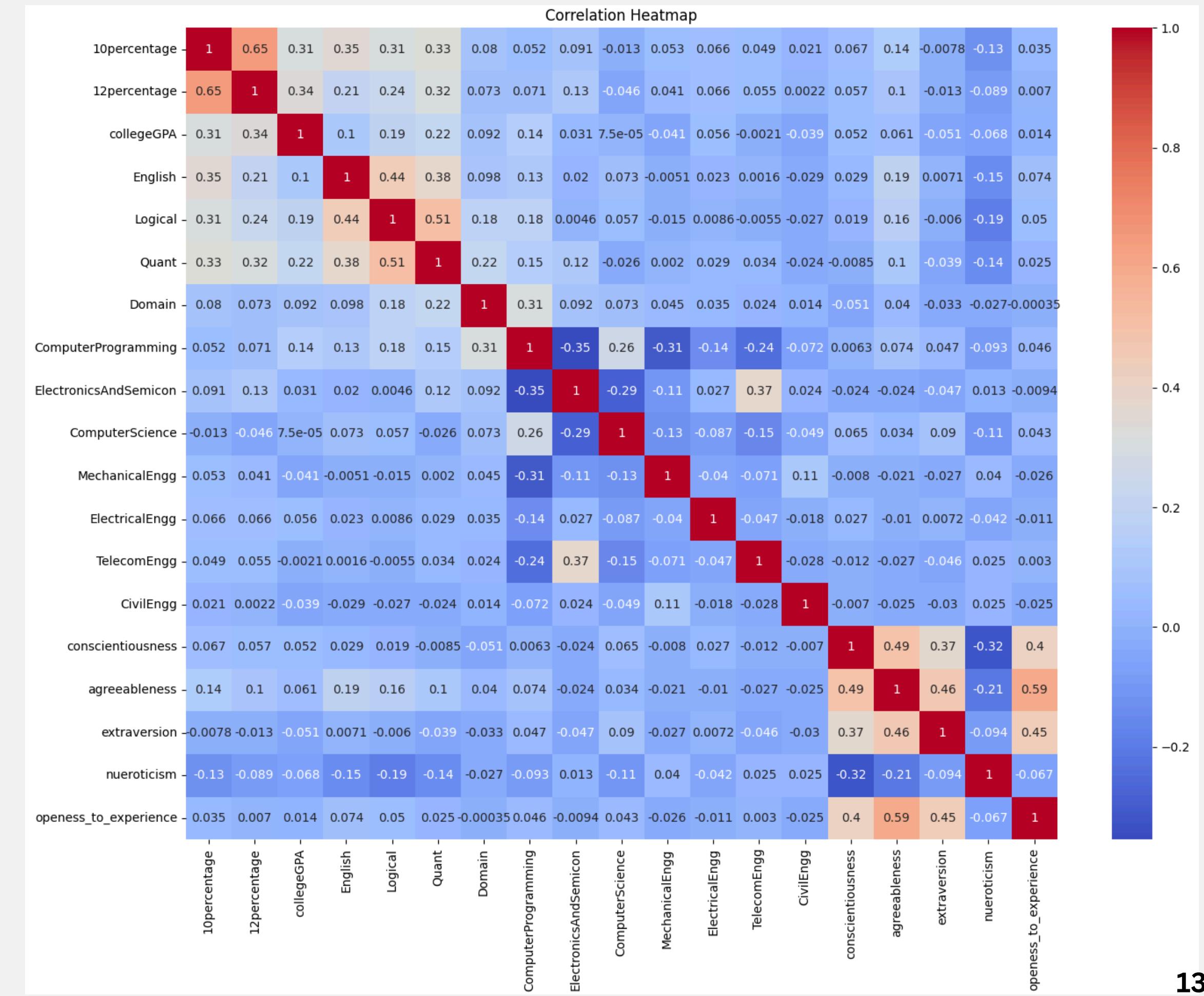
Correlation

int64
object
int64
float64
datetime64[ns]

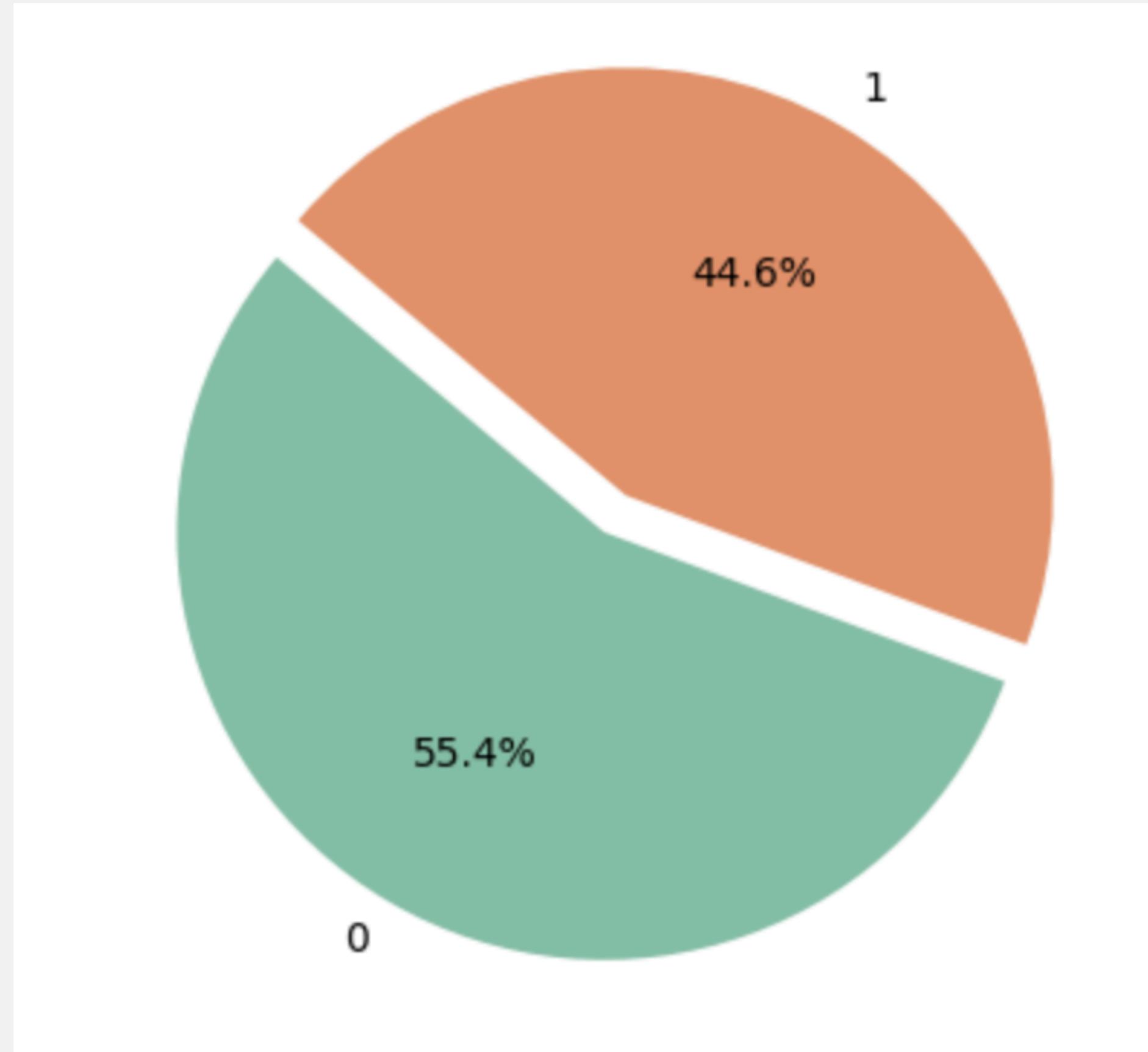


12

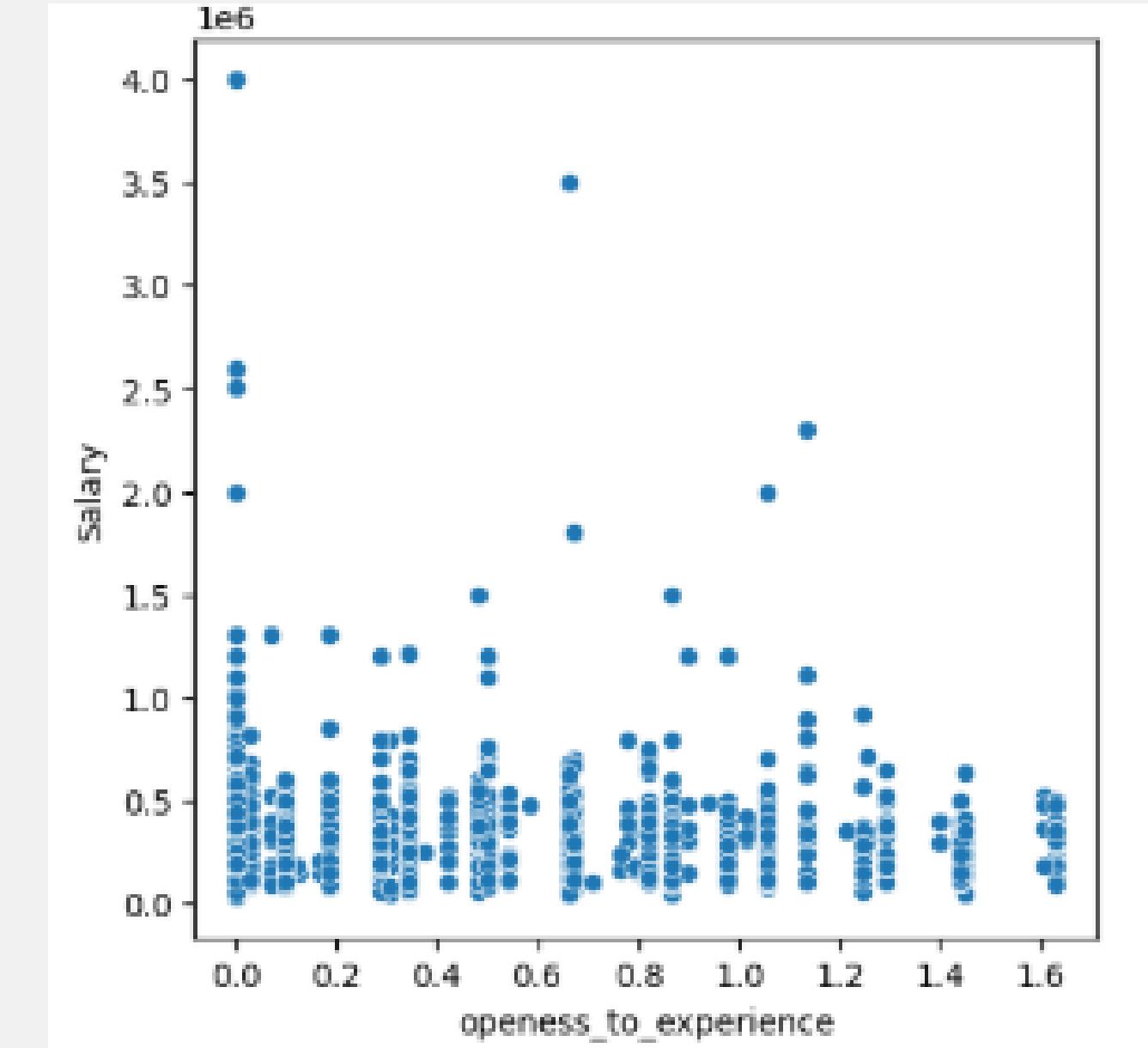
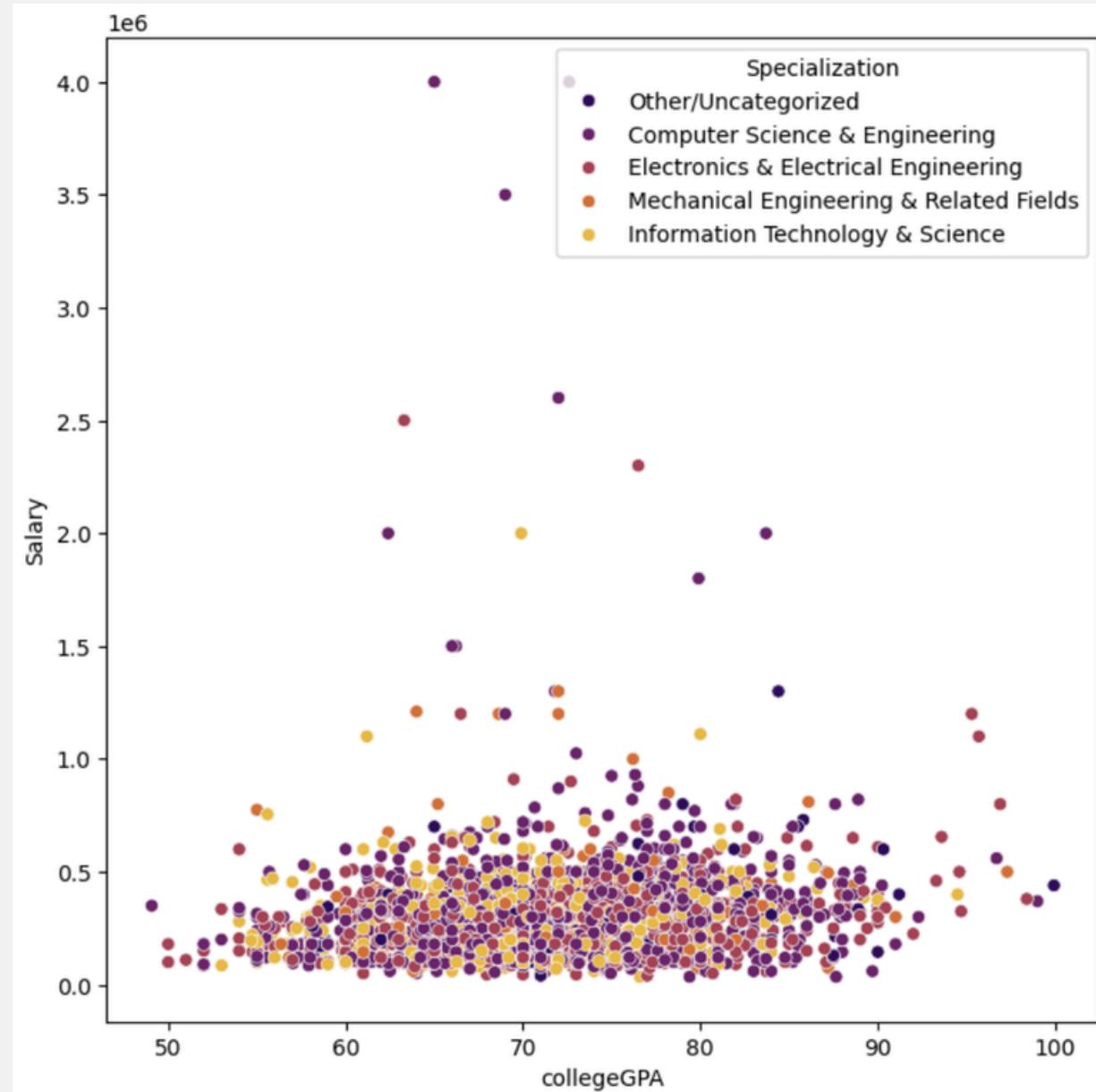
CORRELATION MATRIX



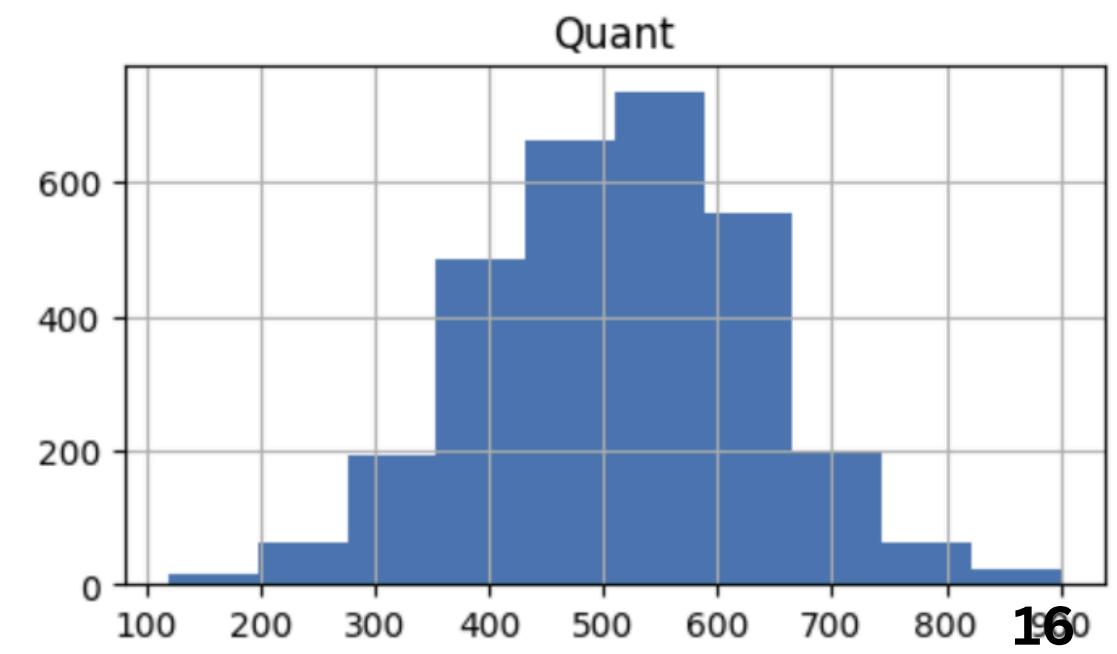
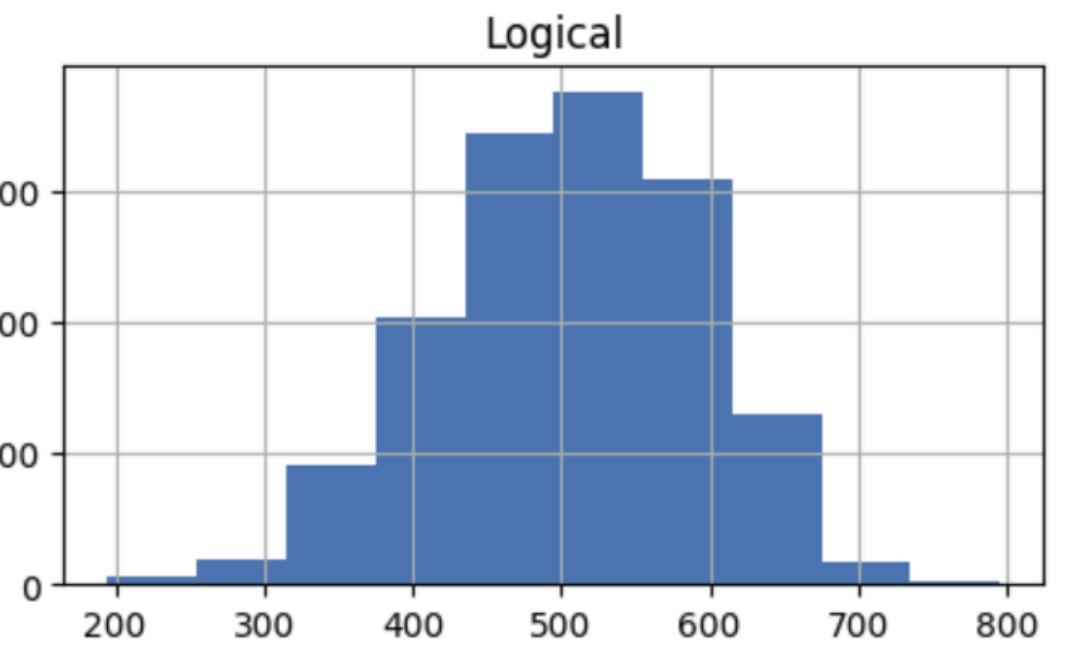
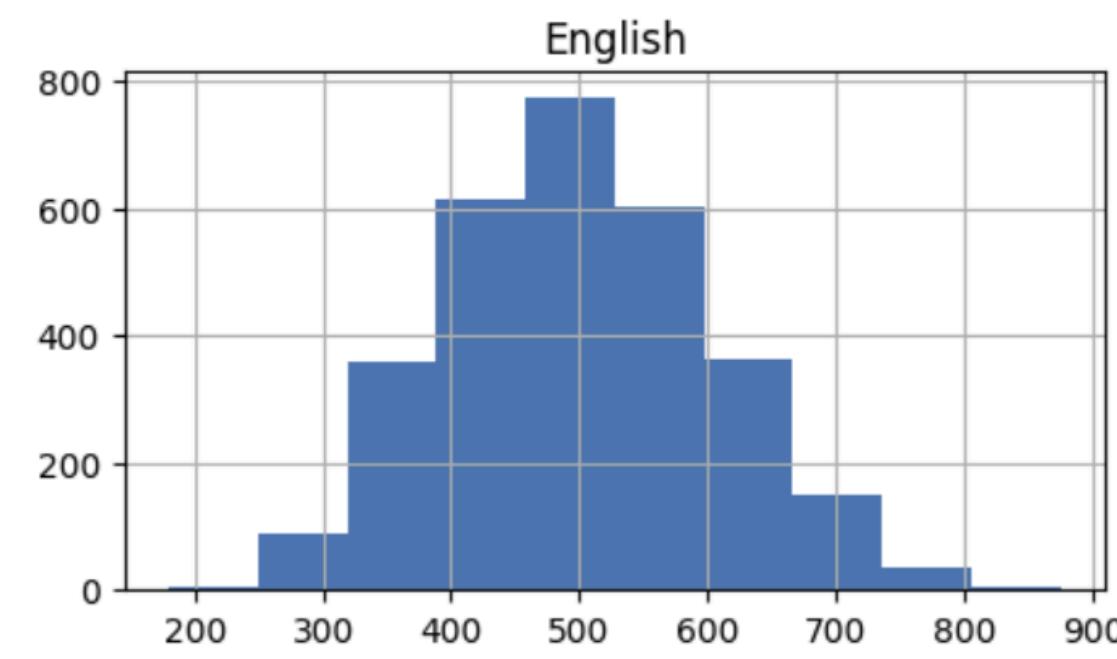
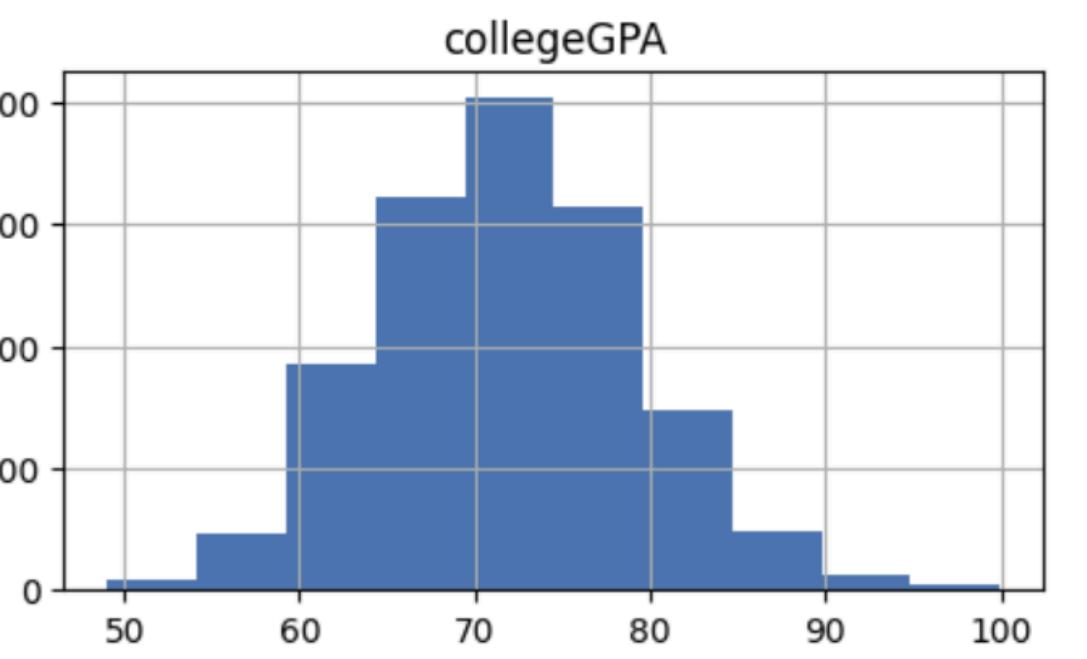
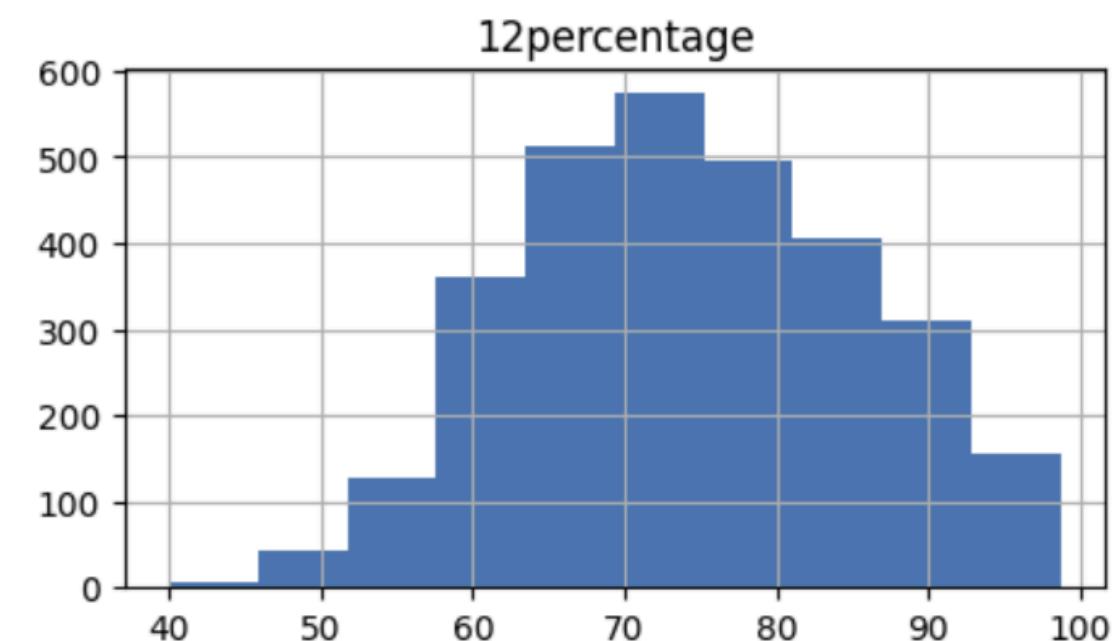
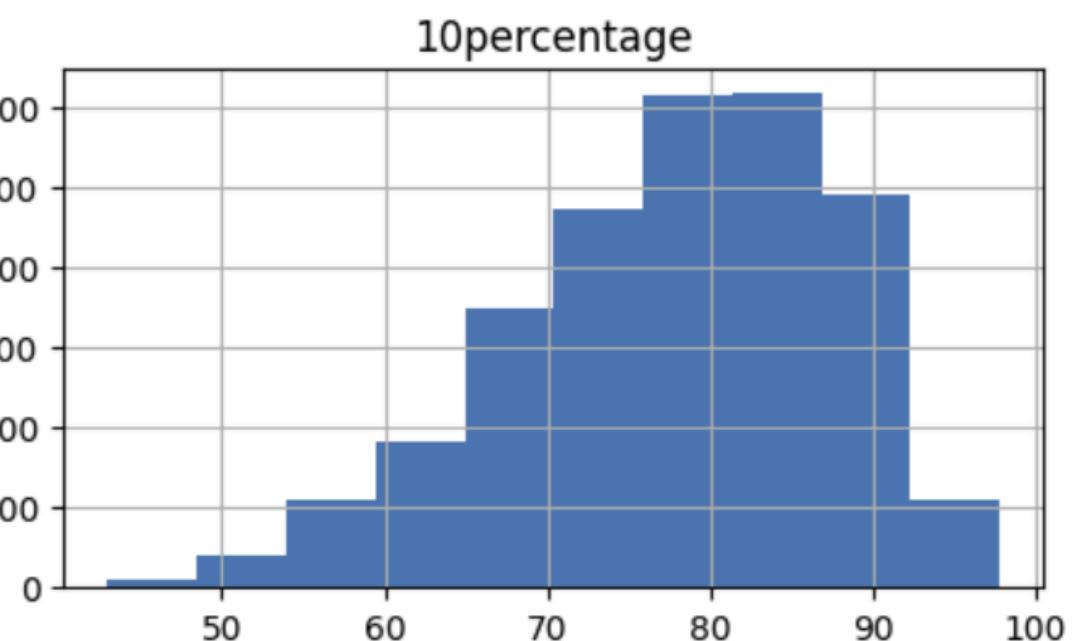
PIE CHART



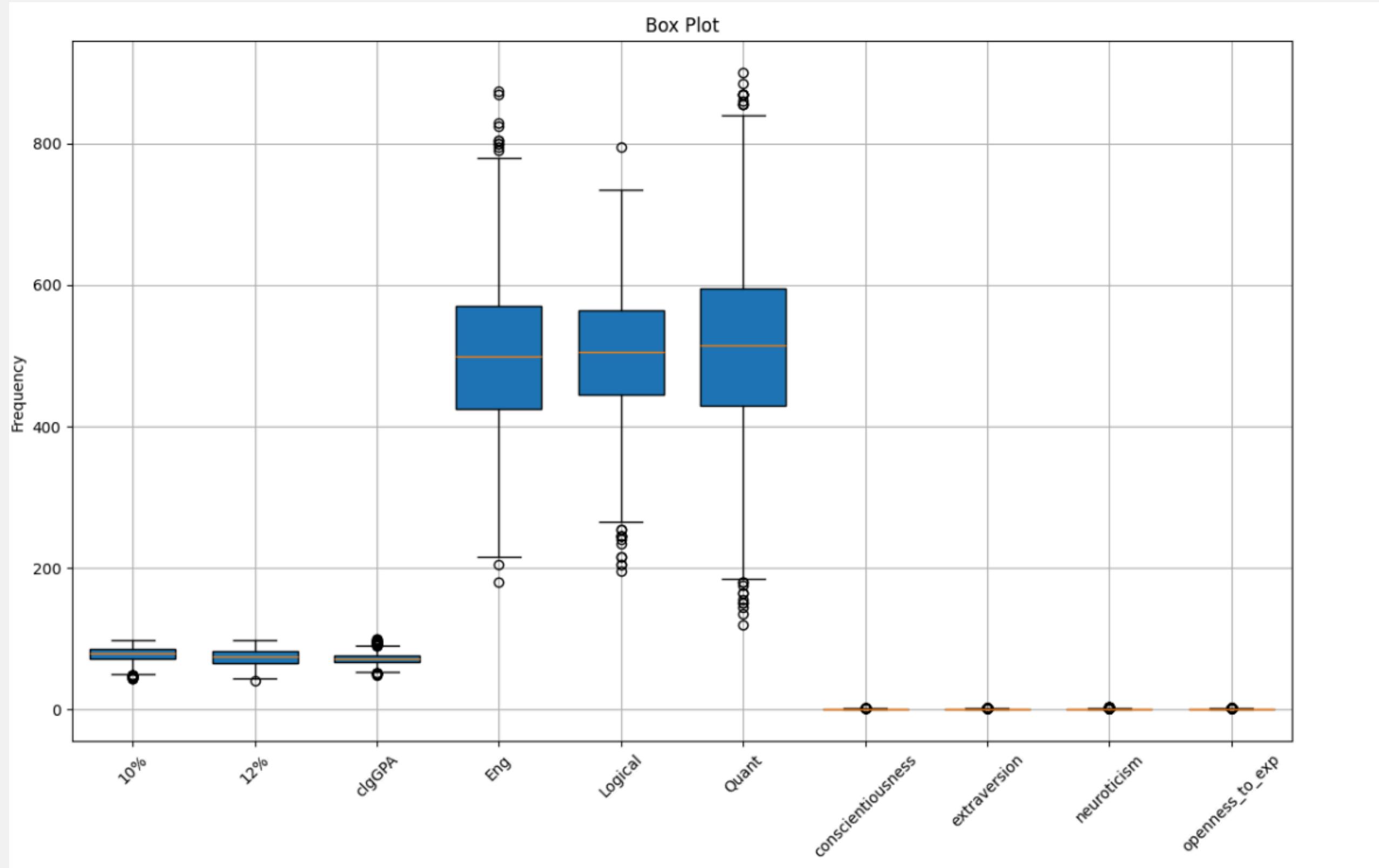
Scatter Plot



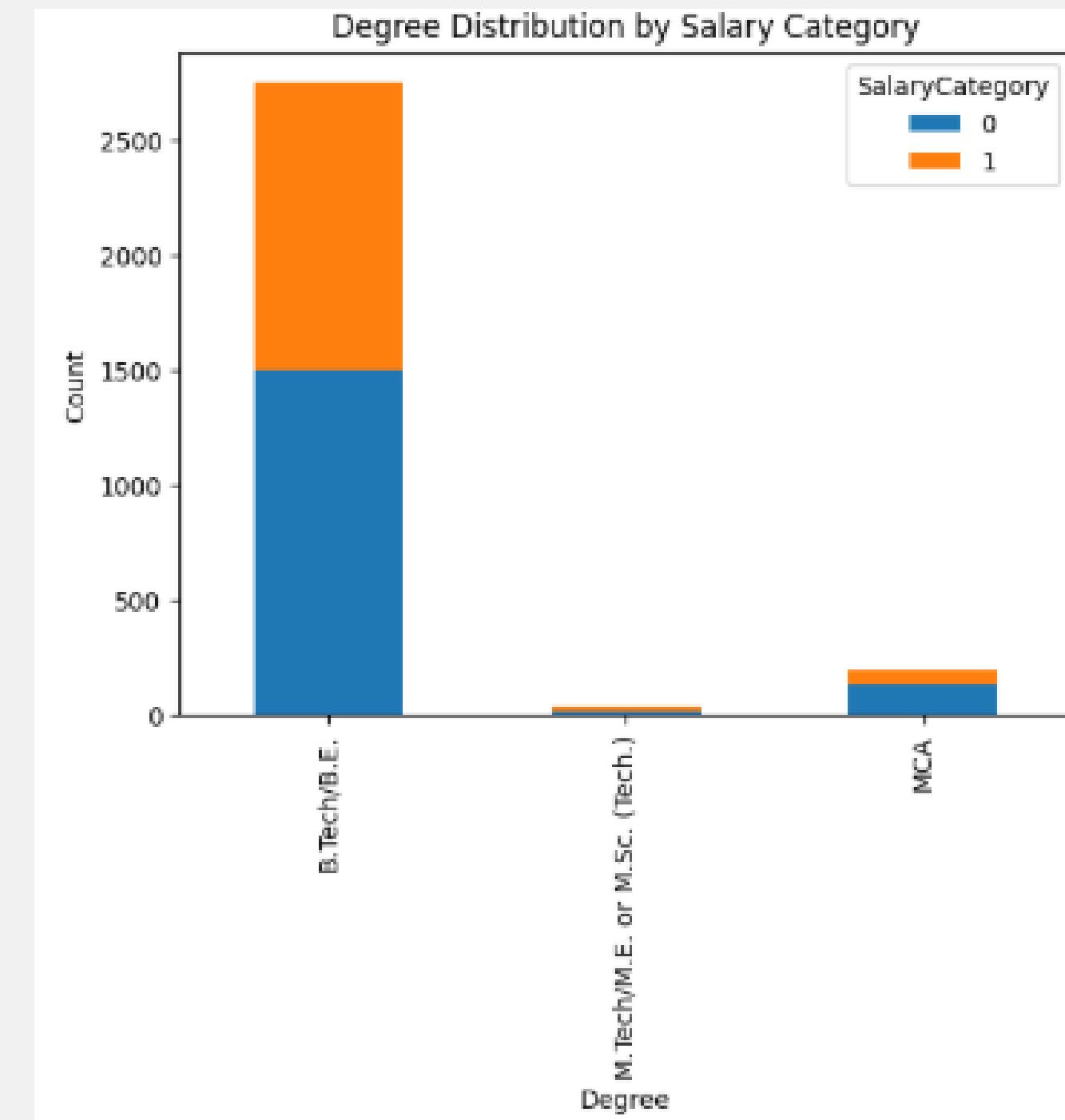
Histogram



Boxplot



Bar plot



Multicollinearity Check

variables with the greatest variance inflation factor (VIF>3) were removed

	variables	VIF
0	10percentage	117.7
1	12percentage	87.4
2	CollegeTier	37.5
3	collegeGPA	101.9
4	English	32.8
5	Logical	52.1
6	Quant	31.1
7	Domain	7.5
8	ComputerProgramming	7.7
9	ElectronicsAndSemicon	4.6
10	ComputerScience	1.6
11	MechanicalEngg	5.5
12	ElectricalEngg	1.4
13	TelecomEngg	1.4
14	CivilEngg	1.2
15	conscientiousness	2.0
16	agreeableness	2.6
17	extraversion	2.0
18	nuerotism	1.4
19	openness_to_experience	1.8
20	Gender_m	4.4
21	Degree_M.Tech/M.E. or M.Sc. (Tech.)	1.1
22	Degree_MCA	1.3
23	Specialization_Electronics & Electrical Engine...	7.3
24	Specialization_Information Technology & Science	1.5
25	Specialization_Mechanical Engineering & Relate...	5.9
26	Specialization_Other/Uncategorized	1.4



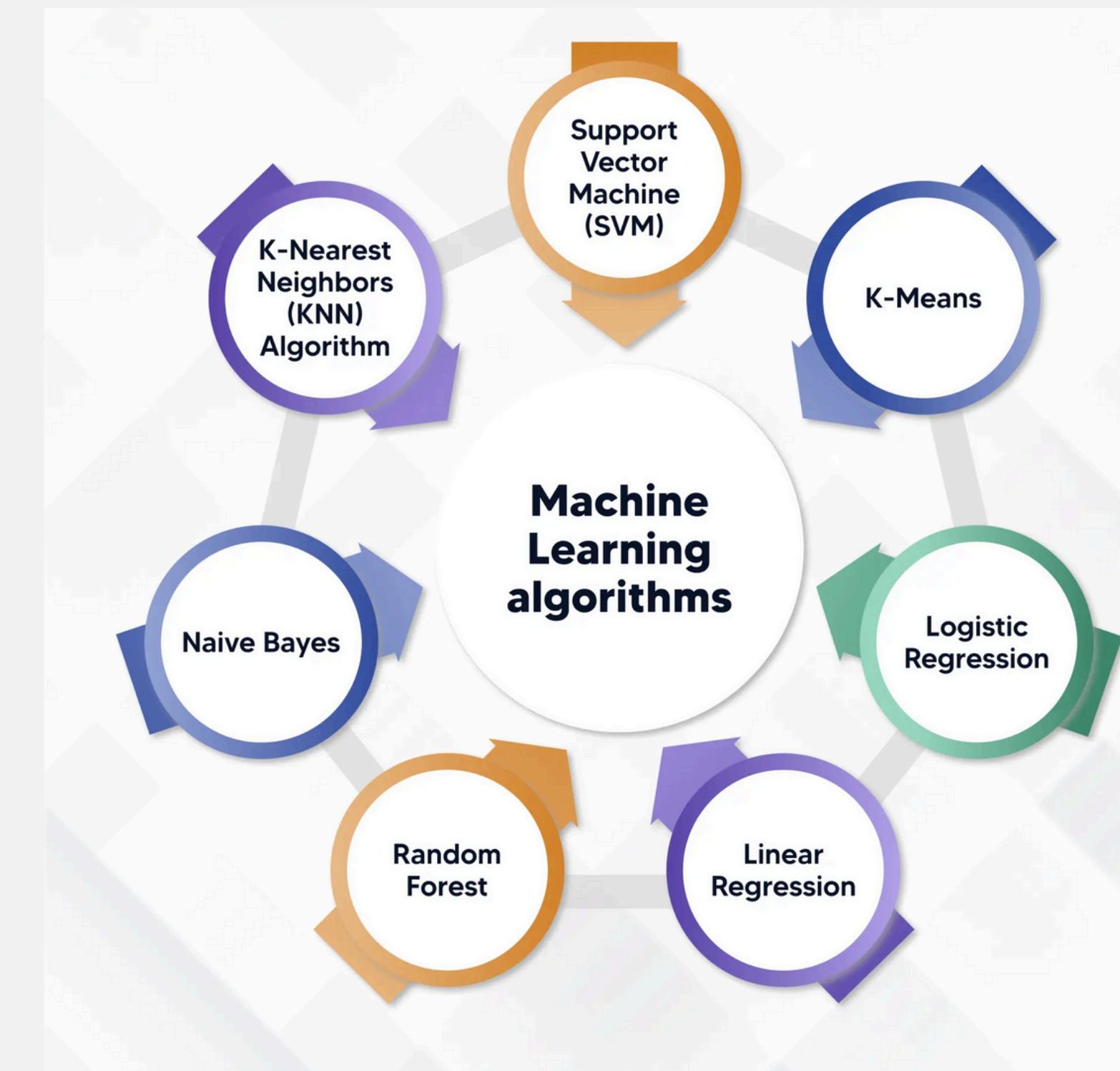
	variables	VIF
0	ComputerProgramming	3.5
1	ElectronicsAndSemicon	3.8
2	ComputerScience	1.5
3	MechanicalEngg	5.4
4	ElectricalEngg	1.3
5	TelecomEngg	1.4
6	CivilEngg	1.2
7	conscientiousness	2.0
8	agreeableness	2.5
9	extraversion	2.0
10	nuerotism	1.4
11	openness_to_experience	1.8
12	Gender_m	3.6
13	Degree_M.Tech/M.E. or M.Sc. (Tech.)	1.0
14	Degree_MCA	1.2
15	Specialization_Electronics & Electrical Engine...	5.2
16	Specialization_Information Technology & Science	1.4
17	Specialization_Mechanical Engineering & Relate...	5.5
18	Specialization_Other/Uncategorized	1.2



	variables	VIF
0	ComputerScience	1.4
1	ElectricalEngg	1.2
2	TelecomEngg	1.4
3	CivilEngg	1.1
4	conscientiousness	2.0
5	agreeableness	2.4
6	extraversion	2.0
7	nuerotism	1.3
8	openness_to_experience	1.8
9	Gender_m	2.8
10	Degree_M.Tech/M.E. or M.Sc. (Tech.)	1.0
11	Degree_MCA	1.1
12	Specialization_Electronics & Electrical Engine...	2.3
13	Specialization_Information Technology & Science	1.3
14	Specialization_Mechanical Engineering & Relate...	1.2
15	Specialization_Other/Uncategorized	1.2

ML Algorithms

- *Logistic Regression*
- *K-Nearest Neighbors(KNN)*
- *Decision Tree*
- *Random Forest*
- *XG Boosting*
- *ADA Boosting*
- *Support Vector Machine(SVM)*



80-20

Algorithms	Model -1 Before VIF Accuracy	Model -2 After VIF Accuracy
Logistic Regression	0.53	0.53
KNN	0.67	0.55
SVM	0.39	0.51
Decision Tree	0.63	0.55
Random Forest	0.69	0.55
XG Boost	0.70	0.56
ADA Boost	0.70	0.57

75-25

Algorithms	Model -1 Before VIF Accuracy	Model-2 After VIF Accuracy
Logistic Regression	0.64	0.55
KNN	0.69	0.55
SVM	0.38	0.53
Decision Tree	0.63	0.52
Random Forest	0.70	0.55
XG Boost	0.68	0.55
ADA Boost	0.70	0.57

70-30

Algorithms	Model -1 Before VIF Accuracy	Model -2 After VIF Accuracy
Logistic Regression	0.67	0.55
KNN	0.68	0.56
SVM	0.38	0.52
Decision Tree	0.63	0.53
Random Forest	0.70	0.56
XG Boost	0.70	0.58
ADA Boost	0.69	0.57

60-40

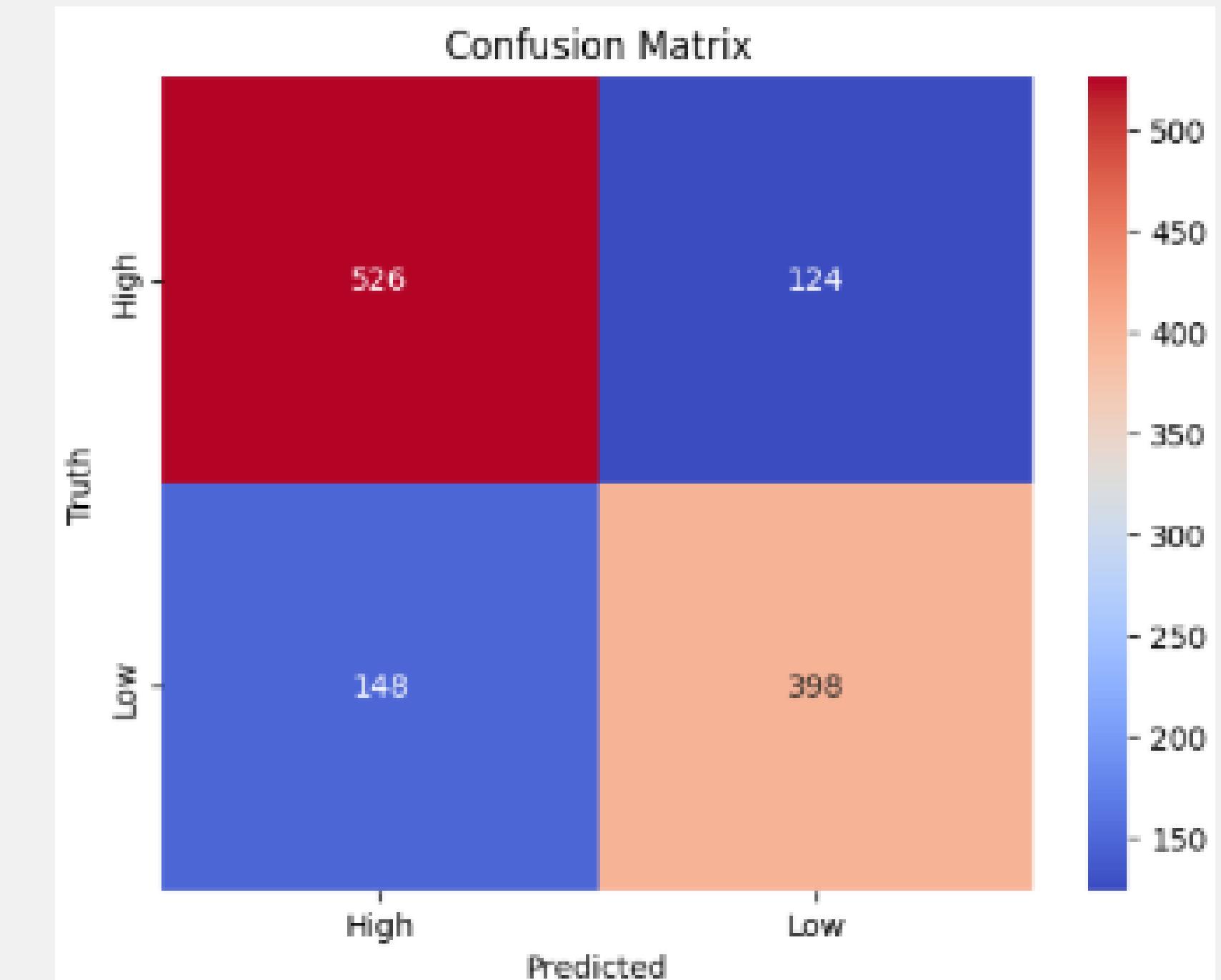
Algorithms	Model -1 Before VIF Accuracy	Model -2 After VIF Accuracy
Logistic Regression	0.64	0.58
KNN	0.68	0.58
SVM	0.38	0.54
Decision Tree	0.59	0.52
Random Forest	0.70	0.58
XG Boost	0.77	0.56
ADA Boost	0.67	0.59

Algorithms Comparision for MODEL-1

60-40 split before applying VIF

Algorithms	Model -1 Before VIF Accuracy
Logistic Regression	0.64
KNN	0.68
SVM	0.38
Decision Tree	0.59
Random Forest	0.70
XG Boost	0.77
ADA Boost	0.67

XG Boost(60-40)

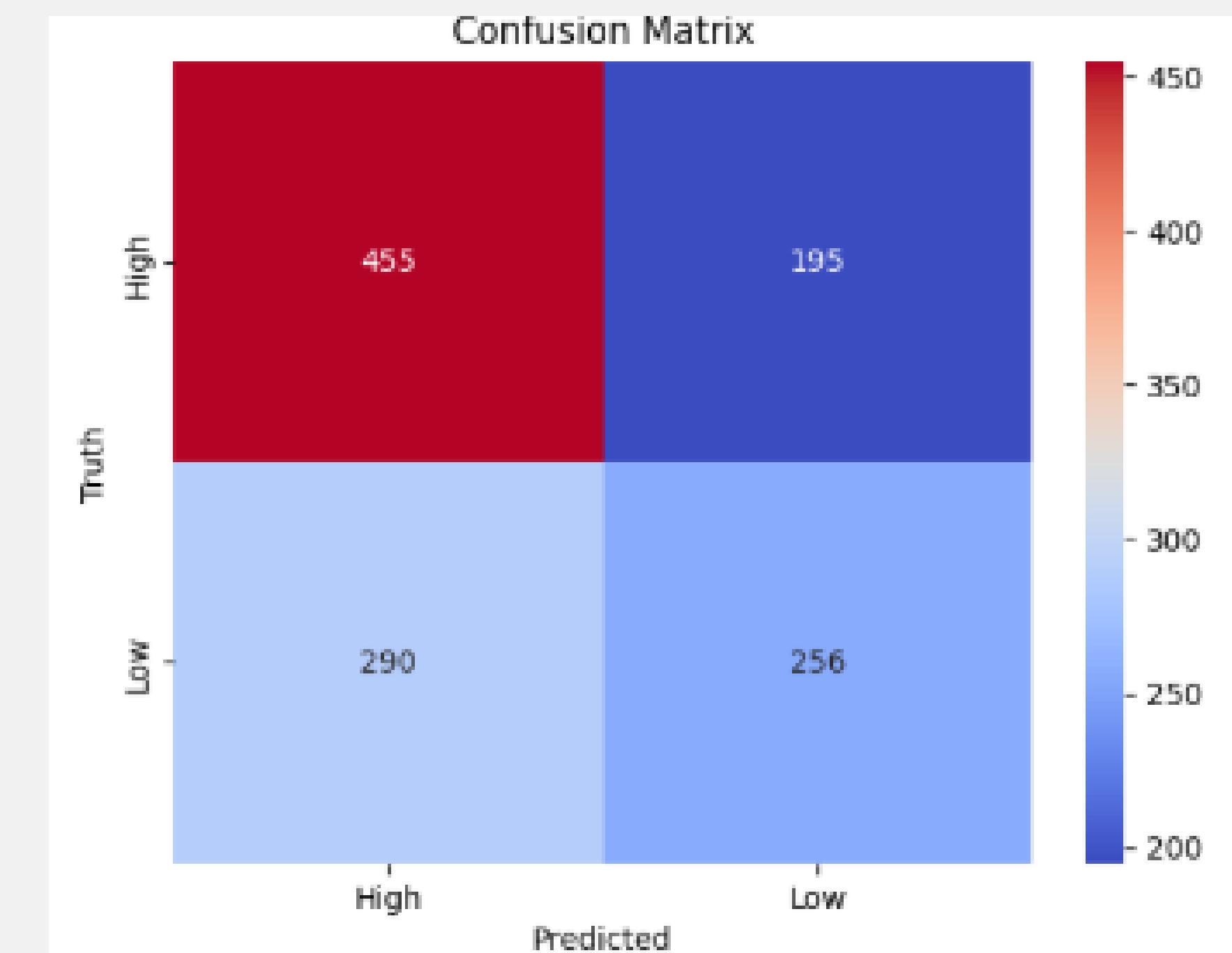


Algorithms Comparision for MODEL-2

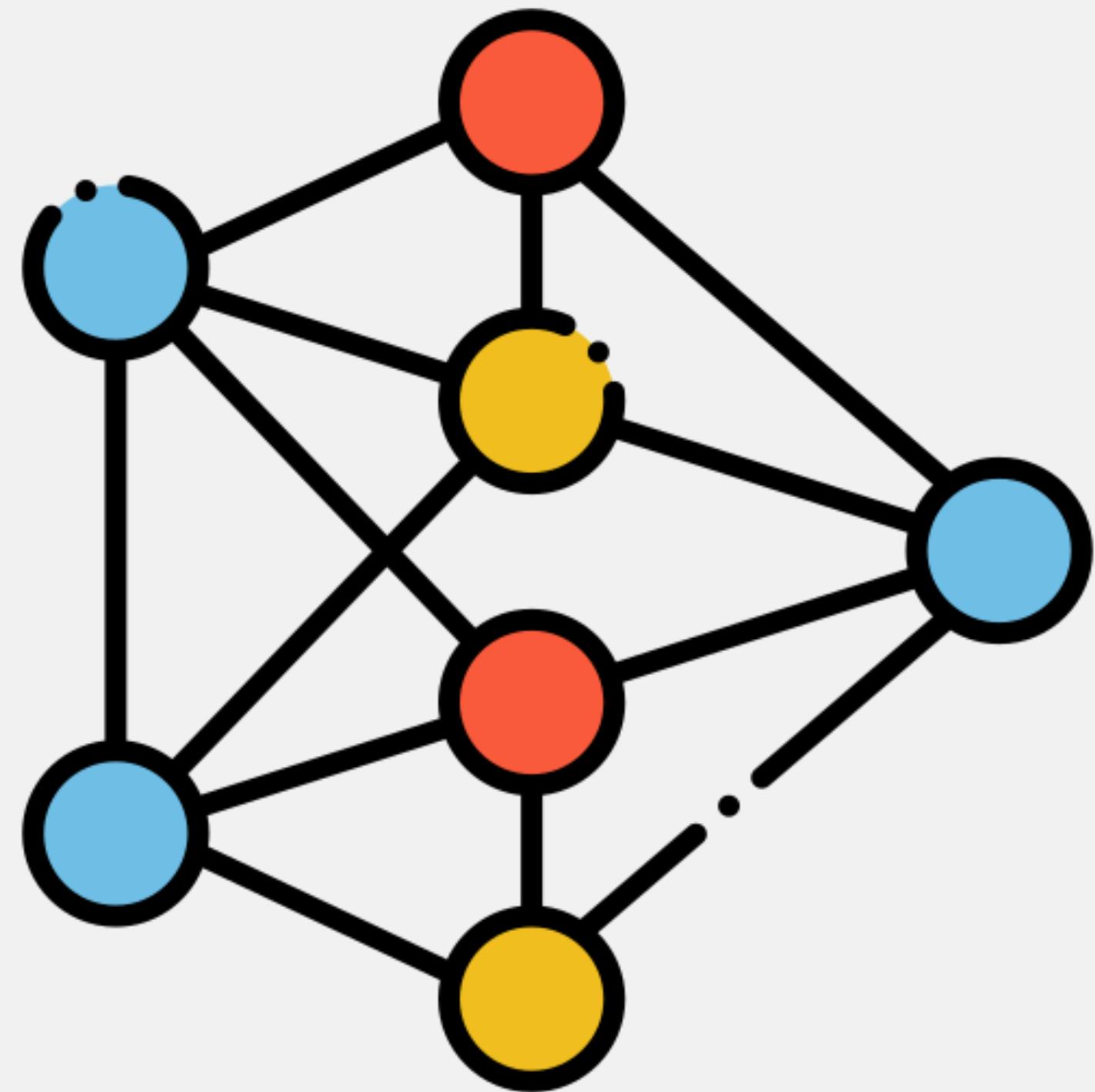
60-40 split before applying VIF

Algorithms	Model -2 After VIF Accuracy
Logistic Regression	0.58
KNN	0.58
SVM	0.54
Decision Tree	0.52
Random Forest	0.58
XG Boost	0.56
ADA Boost	0.59

ADA Boost (60-40)



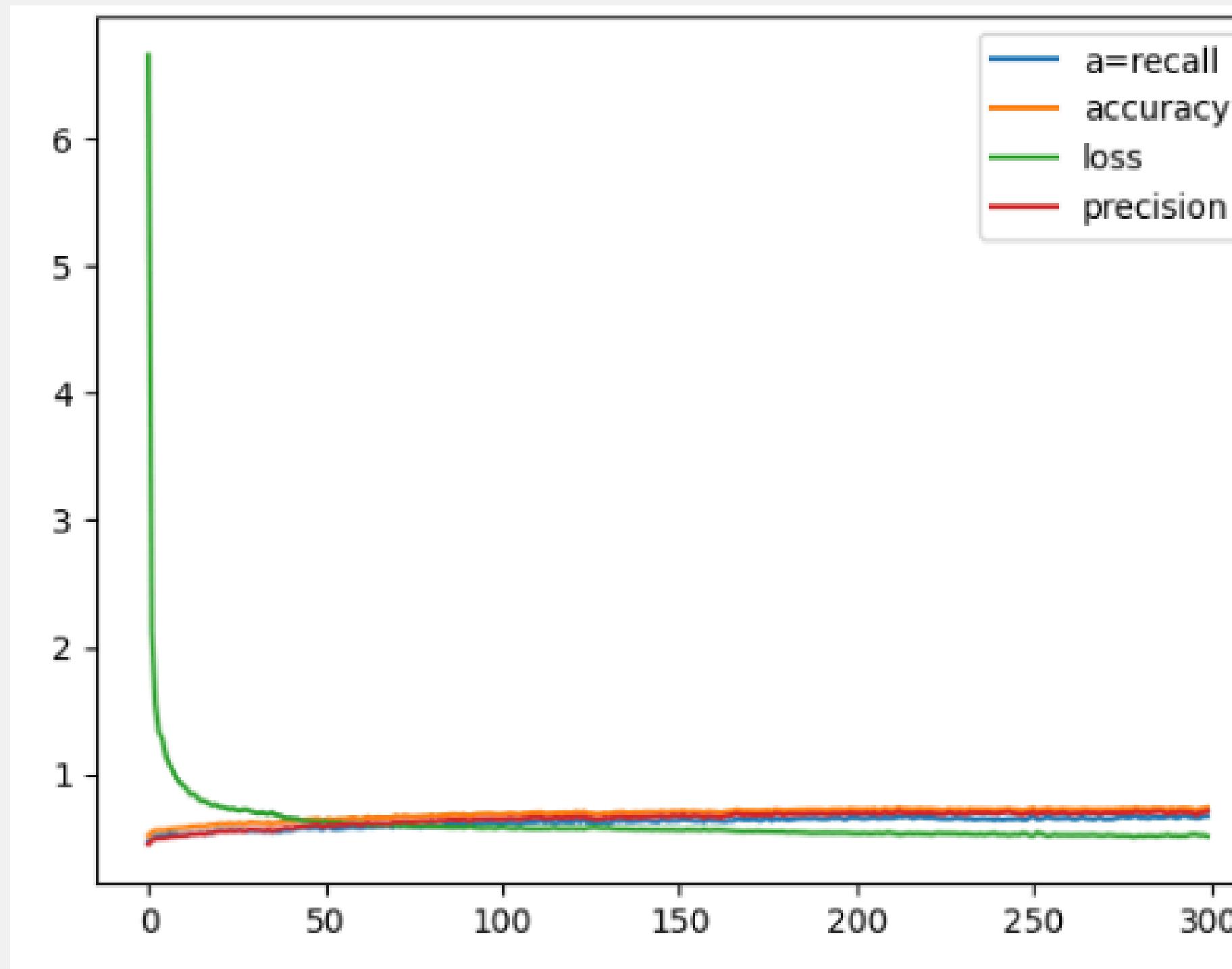
Neural Network



Neural Network

Train-Test	Architecture	Optimizer	Epochs	Precision	Recall	Accuracy	Loss
60-40	30-20-10-8-6-4-1	Adam	250	0.66	0.56	0.65	0.88
60-40	30-20-10-7-5-1	Adam	200	0.63	0.60	0.64	0.80
70-30	30-16-12-8-6-4-1	Adam	250	0.63	0.69	0.66	0.76
70-30	30-20-10-7-5-1	Adam	350	0.68	0.50	0.65	0.74
75-25	30-16-12-8-6-4-1	Adam	300	0.68	0.52	0.65	1.01
75-25	30-20-10-7-5-1	Adam	300	0.70	0.51	0.65	0.79
80-20	30-16-12-8-6-4-1	Adam	300	0.66	0.63	0.65	0.69
80-20	30-20-10-7-5-1	Adam	300	0.71	0.57	0.68	0.71

Neural Network plot for Observation



Train test split	80-20
Architecture	30-20-10-7-5-1
Optimizer	Adam
Epochs	300

SUMMARY

The purpose of this study is to develop a predictive model for estimating the salaries of engineering graduates.

In Model 1, the best fit model is XG Boost with accuracy of 0.77, whereas in Model 2, the best fit model is ADA Boost with accuracy of 0.59

Thus, XG Boost is the best model for forecasting engineering graduates salaries utilizing an 60:40 train split ratio.

FUTURE SCOPE

- Include additional features like social media presence, online portfolios, and extracurricular activities in the dataset.
- Partner with recruitment platforms to predict salaries and educational platforms for tailored course recommendations.
- Implement advanced models like deep learning, ensemble methods, and explainable AI for better predictions.
- Develop gamified simulations to help students understand the impact of skill-building on salaries.

WORK DISTRIBUTION

Team member 1
Rishi Bharadwaj

Collect information about
Engineering Graduate salary&
Literature Review

Team member 2
Ajay kumar

Data Preprocessing

Team member 3
Jai kumar

Exploratory Data Analysis

Team member 4
N.Aishwarya

Implement Machine Learning
Algorithms



Colab Notebook

Thank You

**N.AISHWARYA
RISHI BHARADWAJ
AJAY KUMAR
JAI KUMAR**

Appendix

Loading the Dataset

```
df= pd.read_csv('Engineering_graduate_salary.csv')
df.head()
```

	ID	Gender	DOB	10percentage	10board	12graduation	12percentage	12board	CollegeID	CollegeTier	...	MechanicalEngg	ElectricalEngg	TelecomEngg	CivilEngg	conscientiousness	agreeableness	extraversion	nueroticism	openess_to_experience	Salary
0	604399	f	1990-10-22	87.80	cbse	2009	84.00	cbse	6920	1	...	-1	-1	-1	-1	-0.1590	0.3789	1.2396	0.14590	0.2889	445000
1	988334	m	1990-05-15	57.00	cbse	2010	64.50	cbse	6624	2	...	-1	-1	-1	-1	1.1336	0.0459	1.2396	0.52620	-0.2859	110000
2	301647	m	1989-08-21	77.33	maharashtra state board,pune	2007	85.17	amravati divisional board	9084	2	...	-1	-1	260	-1	0.5100	-0.1232	1.5428	-0.29020	-0.2875	255000
3	582313	m	1991-05-04	84.30	cbse	2009	86.00	cbse	8195	1	...	-1	-1	-1	-1	-0.4463	0.2124	0.3174	0.27270	0.4805	420000
4	339001	f	1990-10-30	82.00	cbse	2008	75.00	cbse	4889	2	...	-1	-1	-1	-1	-1.4992	-0.7473	-1.0697	0.06223	0.1864	200000

5 rows × 34 columns

Null Values

df.isna().sum()	
10percentage	0
10board	0
12graduation	0
12percentage	0
12board	0
CollegeID	0
CollegeTier	0
Degree	0
Specialization	0
collegeGPA	0
CollegeCityID	0
CollegeCityTier	0
CollegeState	0
GraduationYear	0
English	0
Logical	0
Quant	0
Domain	0
ComputerProgramming	0
ElectronicsAndSemicon	0
ComputerScience	0
MechanicalEngg	0
ElectricalEngg	0
TelecomEngg	0
CivilEngg	0
conscientiousness	0
agreeableness	0
extraversion	0
nueroticism	0
openess_to_experience	0
Salary	0

Checking for the data type

df.info()			
<class 'pandas.core.frame.DataFrame'>			
RangeIndex: 2998 entries, 0 to 2997			
#	Column	Non-Null Count	Dtype
0	ID	2998 non-null	int64
1	Gender	2998 non-null	object
2	DOB	2998 non-null	object
3	10percentage	2998 non-null	float64
4	10board	2998 non-null	object
5	12graduation	2998 non-null	int64
6	12percentage	2998 non-null	float64
7	12board	2998 non-null	object
8	CollegeID	2998 non-null	int64
9	CollegeTier	2998 non-null	int64
10	Degree	2998 non-null	object
11	Specialization	2998 non-null	object
12	collegeGPA	2998 non-null	float64
13	CollegeCityID	2998 non-null	int64
14	CollegeCityTier	2998 non-null	int64
15	CollegeState	2998 non-null	object
16	GraduationYear	2998 non-null	int64
17	English	2998 non-null	int64
18	Logical	2998 non-null	int64
19	Quant	2998 non-null	int64
20	Domain	2998 non-null	float64
21	ComputerProgramming	2998 non-null	int64
22	ElectronicsAndSemicon	2998 non-null	int64
23	ComputerScience	2998 non-null	int64
24	MechanicalEngg	2998 non-null	int64
25	ElectricalEngg	2998 non-null	int64
26	TelecomEngg	2998 non-null	int64
27	CivilEngg	2998 non-null	int64
28	conscientiousness	2998 non-null	float64
29	agreeableness	2998 non-null	float64
30	extraversion	2998 non-null	float64
31	nueroticism	2998 non-null	float64
32	openess_to_experience	2998 non-null	float64
33	Salary	2998 non-null	int64
dtypes: float64(9), int64(18), object(7)			
memory usage: 796.5+ KB			

checking unique Values

```

for i in range(df.shape[1]):
    print(df.iloc[:,i].unique())
    print(df.iloc[:,i].value_counts())

-2.0105 -1.0524 -1.1169 -1.4356 1.4502 -1.2354 -0.4776 -0.169 -1.6273
-0.8608 -0.6035 0.0973 0.4234 0.8973 -1.244 0.9763 1.0554 0.0284
1.247 -1.8278 0.5024 1.6302 -0.9194 -1.8189 -0.7615 1.1343 -0.4392
-2.0253 1.6882 -2.6572 -5.0763 0.5419 0.7788 -2.3412 0.1275 -2.2821
-1.0774 0.1187 -0.643 -5.6512 -1.7093 0.7631 -0.0506 -0.9984 0.8183
0.167 -2.1833 -0.5245 -2.3937 -0.8799 -1.8673 -6.8009 -3.763 -2.9686
-2.3017 -5.2679 -0.4601 1.0158 -1.0872 -6.6092 0.3049 -1.5513 -0.406
-3.9266 0.7986 -6.9925 -2.7769 -1.359 -3.735 -5.686 0.0679 -1.9234
-2.7595 -2.9731 -3.1311 1.2528 -1.9463 -1.3539 1.3976 1.0395 -2.8152
-3.1602 -1.4724 -3.6051 -2.0648 -1.1291 0.7941 -3.4471 -0.4229 0.0916
-1.8386 -1.0458 -0.2511 1.4003 -0.0167 -3.5434 0.7104 -0.4139 0.585
-3.3518 -0.8845 0.7657 -0.1521 0.9404 -3.9605 1.2121 -0.0844 -5.4595
-1.6662 -0.8782 0.376 -7.3757 -4.3099]
openness_to_experience
0.0973 142
0.6721 148
0.4805 136
-0.0943 134
0.2889 133
...
1.0395 1
-2.8152 1
-1.1291 1
-0.4229 1
-4.3099 1
Name: count, Length: 131, dtype: int64
[ 445000 110000 255000 420000 200000 440000 150000 105000 195000
  335000 300000 480000 550000 325000 405000 500000 210000 400000
  360000 240000 120000 430000 600000 80000 330000 450000 215000
  315000 220000 180000 60000 165000 370000 145000 100000 510000
  310000 380000 170000 95000 75000 140000 320000 1210000 700000
  305000 410000 375000 295000 260000 160000 475000 250000 465000
  720000 505000 290000 350000 230000 615000 1110000 340000 205000
  280000 190000 520000 235000 715000 390000 455000 385000 355000
  490000 200000 150000 275000 185000 225000 525000 515000 265000
  395000 90000 730000 485000 760000 820000 135000 460000 560000
  415000 435000 155000 425000 115000 345000 245000 495000 45000
  565000 660000 70000 470000 35000 270000 2600000 1025000 365000
  175000 570000 680000 785000 285000 870000 130000 1200000 650000
  144000 810000 880000 1100000 675000 65000 770000 2500000 125000
1300000 610000 725000 775000 50000 555000 540000 705000 620000
  925000 800000 85000 545000 585000 590000 3500000 530000 535000
  630000 625000 605000 850000 40000 900000 640000 580000 690000
  930000 645000 655000 910000 750000 2300000 1800000 4000000 755000
  55000 1000000 575000]
Salary
300000 207
180000 189
200000 154
325000 134
120000 121
...
775000 1
705000 1
925000 1
585000 1
575000 1
Name: count, Length: 165, dtype: int64

```

Dropping few columns

```

df.drop(['ID','DOB','CollegeID','12graduation','10board','CollegeState','CollegeCityID', 'CollegeCityTier','GraduationYear'], axis=1, inplace=True)
print(df)

   2994     f      84.00      77.00      2  B.Tech/B.E.
   2995     m      91.40      65.56      2  B.Tech/B.E.
   2996     m      88.64      65.16      2  B.Tech/B.E.
   2997     m      77.00      75.50      2  B.Tech/B.E.

          Specialization  collegeGPA  English  Logical \
0      instrumentation and control engineering      73.82     658     665
1      computer science & engineering      65.00     448     435
2      electronics & telecommunications      61.94     485     475
3      computer science & engineering      80.40     675     620
4      biotechnology                  64.30     575     495
...
2993  electronics and communication engineering      70.00     505     485
2994           information technology      75.20     345     585
2995           information technology      73.19     385     425
2996           computer engineering      74.81     465     645
2997           information technology      69.30     370     390

          Quant ... MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg \
0       810 ...          -1          -1          -1          -1
1       210 ...          -1          -1          -1          -1
2       505 ...          -1          -1          260          -1
3       635 ...          -1          -1          -1          -1
4       365 ...          -1          -1          -1          -1
...
2993    445 ...          -1          -1          -1          -1
2994    395 ...          -1          -1          -1          -1
2995    485 ...          -1          -1          -1          -1
2996    505 ...          -1          -1          -1          -1
2997    285 ...          -1          -1          -1          -1

          conscientiousness  agreeableness  extraversion  nueroticism \
0            -0.1590        0.3789       1.2396      0.14590
1            1.1336        0.0459       1.2396      0.52620
2            0.5100        -0.1232       1.5428     -0.29020
3            -0.4463        0.2124       0.3174      0.27270
4            -1.4992        -0.7473      -1.0697      0.06223
...
2993    -1.1901        0.9688      -1.0697      1.35490
2994    -0.1082        0.0328      -0.4891     -0.29020
2995    -0.8810        0.1888      -0.3448      0.06230
2996    1.4374        1.2808      -0.4891     -1.46537
2997    -0.5899        -1.9521      0.3174      1.16010

          openness_to_experience  Salary
0            0.2889      445000
1            -0.2859      110000
2            -0.2875      255000
3            0.4805      420000
4            0.1864      200000
...
2993    0.0284      120000
2994    0.5024      120000
2995    0.6603      385000
2996    0.5419      530000
2997   -2.3937      200000

```

Renaming and merging few columns

```
df['Degree'] = df['Degree'].replace({
    'M.Tech./M.E.': 'M.Tech/M.E. or M.Sc. (Tech.)',
    'M.Sc. (Tech.)': 'M.Tech/M.E. or M.Sc. (Tech.)'
})
print(df)
```

	Gender	10percentage	12percentage	CollegeTier	Degree	\	
0	f	87.80	84.00	1	B.Tech/B.E.		
1	m	57.00	64.50	2	B.Tech/B.E.		
2	m	77.33	85.17	2	B.Tech/B.E.		
3	m	84.30	86.00	1	B.Tech/B.E.		
4	f	82.00	75.00	2	B.Tech/B.E.		
...		
2993	f	75.00	73.00	2	B.Tech/B.E.		
2994	f	84.00	77.00	2	B.Tech/B.E.		
2995	m	91.40	65.56	2	B.Tech/B.E.		
2996	m	88.64	65.16	2	B.Tech/B.E.		
2997	m	77.00	75.50	2	B.Tech/B.E.		
	Specialization	collegeGPA	English	Logical	\		
0	instrumentation and control engineering	73.82	650	665			
1	computer science & engineering	65.00	440	435			
2	electronics & telecommunications	61.94	485	475			
3	computer science & engineering	80.40	675	620			
4	biotechnology	64.30	575	495			
...			
2993	electronics and communication engineering	70.00	505	485			
2994	information technology	75.20	345	585			
2995	information technology	73.19	385	425			
2996	computer engineering	74.81	465	645			
2997	information technology	69.30	370	390			
	Quant	...	MechanicalEngg	ElectricalEngg	TelecomEngg	CivilEngg	\
0	810	...	-1	-1	-1	-1	
1	210	...	-1	-1	-1	-1	
2	505	...	-1	-1	260	-1	
3	635	...	-1	-1	-1	-1	
4	365	...	-1	-1	-1	-1	
...	
2993	445	...	-1	-1	-1	-1	
2994	395	...	-1	-1	-1	-1	
2995	485	...	-1	-1	-1	-1	
2996	505	...	-1	-1	-1	-1	
2997	285	...	-1	-1	-1	-1	
	conscientiousness	agreeableness	extraversion	nueroticism	\		
a	-0.1500	0.3700	0.2200	0.1400			

```
specialization_mapping = {
    #'instrumentation and control engineering': 'other',
    'computer science & engineering': 'Computer Science & Engineering',
    'electronics & telecommunications': 'Electronics & Electrical Engineering',
    #'biotechnology': 'other',
    'mechanical engineering': 'Mechanical Engineering & Related Fields',
    'information technology': 'Information Technology & Science',
    'electronics and communication engineering': 'Electronics & Electrical Engineering',
    'computer engineering': 'Computer Science & Engineering',
    'computer application': 'Computer Science & Engineering',
    'electrical engineering': 'Electronics & Electrical Engineering',
    'automobile/automotive engineering': 'Mechanical Engineering & Related Fields',
    'electronics and electrical engineering': 'Electronics & Electrical Engineering',
    'information science engineering': 'Information Technology & Science',
    #'chemical engineering': 'other',
    #'instrumentation engineering': 'other',
    'electronics & instrumentation eng': 'Electronics & Electrical Engineering',
    #'ceramic engineering': 'other',
    #'metallurgical engineering': 'other',
    #'aeronautical engineering': 'other',
    'electronics engineering': 'Electronics & Electrical Engineering',
    'electronics and instrumentation engineering': 'Electronics & Electrical Engineering',
    'applied electronics and instrumentation': 'Electronics & Electrical Engineering',
    #'civil engineering': 'other',
    'computer and communication engineering': 'Computer Science & Engineering',
    #'industrial & production engineering': 'other',
    'computer networking': 'Computer Science & Engineering',
    'electronics and computer engineering': 'Electronics & Electrical Engineering',
    #'control and instrumentation engineering': 'other',
    'mechanical & production engineering': 'Mechanical Engineering & Related Fields',
    'mechanical and automation': 'Mechanical Engineering & Related Fields',
    #'industrial & management engineering': 'other',
    #'biomedical engineering': 'other',
    'electrical and power engineering': 'Electronics & Electrical Engineering',
    'telecommunication engineering': 'Electronics & Electrical Engineering',
    #'industrial engineering': 'other',
    'mechatronics': 'Mechanical Engineering & Related Fields',
    'embedded systems technology': 'Computer Science & Engineering',
    'information & communication technology': 'Information Technology & Science',
    'information science': 'Information Technology & Science'
}

[16] df['Specialization'] = df['Specialization'].map(specialization_mapping)

# Fill NaNs if any specialization was not in the mapping
df['Specialization'].fillna('Other/Uncategorized', inplace=True)

# Check the transformed data
print(df['Specialization'].value_counts())

Specialization
Computer Science & Engineering      1180
Electronics & Electrical Engineering   1038
Information Technology & Science       526
Mechanical Engineering & Related Fields 168
Other/Uncategorized                      86
Name: count, dtype: int64
<ipython-input-16-c2a2de0c0c2c6>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

df['Specialization'].fillna('Other/Uncategorized', inplace=True)
```

Replacing columns with -1 to 0's

```
▶ df.replace(-1, 0, inplace=True)

# Verify results
print(df)

    Gender 10percentage 12percentage CollegeTier      Degree \
0       f        87.80        84.00          1  B.Tech/B.E.
1       m        57.00        64.50          2  B.Tech/B.E.
2       m        77.33        85.17          2  B.Tech/B.E.
3       m        84.30        86.00          1  B.Tech/B.E.
4       f        82.00        75.00          2  B.Tech/B.E.

...
2993    f        75.00        73.00          2  B.Tech/B.E.
2994    f        84.00        77.00          2  B.Tech/B.E.
2995    m        91.40        65.56          2  B.Tech/B.E.
2996    m        88.64        65.16          2  B.Tech/B.E.
2997    m        77.00        75.50          2  B.Tech/B.E.

           Specialization collegeGPA  English  Logical \
0  Other/Uncategorized        73.82      650     665
1  Computer Science & Engineering      65.00      440     435
2  Electronics & Electrical Engineering      61.94      485     475
3  Computer Science & Engineering        80.40      675     620
4  Other/Uncategorized        64.30      575     495

...
2993  Electronics & Electrical Engineering      78.00      585     485
2994  Information Technology & Science        75.20      345     585
2995  Information Technology & Science        73.19      385     425
2996  Computer Science & Engineering        74.81      465     645
2997  Information Technology & Science        69.30      370     390

    Quant ... MechanicalEngg ElectricalEngg TelecomEngg CivilEngg \
0     810 ...             0            0            0            0
1     210 ...             0            0            0            0
2     505 ...             0            0            0            0
3     635 ...             0            0            0            0
4     365 ...             0            0            0            0

...
2993   445 ...             0            0            0            0
2994   395 ...             0            0            0            0
2995   485 ...             0            0            0            0
2996   505 ...             0            0            0            0
2997   285 ...             0            0            0            0

  conscientiousness agreeableness extraversion nuerotism \
0        -0.1590       0.3789      1.2396      0.14590
1         1.1336       0.0459      1.2396      0.52620
2         0.5100      -0.1232      1.5428     -0.29020
3        -0.4463       0.2124      0.3174      0.27270
4        -1.4992      -0.7473     -1.0697      0.06223
```

Converting salary to salary category

```
[25] max_value = df['Salary'].max()
min_value = df['Salary'].min()
print("Maximum value:", max_value)
print("Minimum value:", min_value)
mean_salary = df['Salary'].mean()
print("Mean value:", mean_salary)

Maximum value: 4000000
Minimum value: 35000
Mean value: 305086.65105386416

salary_threshold = df['Salary'].mean() # You can also use mean() or a custom threshold

# Apply categorization
df['SalaryCategory'] = df['Salary'].apply(lambda x: 1 if x > salary_threshold else 0) #high:1,low:0

# Define the mapping dictionary
mapping = {"High": 1, "Low": 0}

# Verify the new columns
print(df[['Salary', 'SalaryCategory']])

# Check the distribution of categories
print(df['SalaryCategory'].value_counts())

    Salary  SalaryCategory
0     445000            1
1     110000            0
2     255000            0
3     420000            1
4     200000            0
...
2993   120000            0
2994   120000            0
2995   385000            1
2996   530000            1
2997   200000            0

[2989 rows x 2 columns]
SalaryCategory
0    1657
1    1332
Name: count, dtype: int64
```

Dividing the dataset

```
X=df.drop(['SalaryCategory'],axis=1)
print(X)
y = df['SalaryCategory']
print(y)

2993 Electronics & Electrical Engineering    70.00    505    485
2994 Information Technology & Science     75.20    345    585
2995 Information Technology & Science     73.19    385    425
2996 Computer Science & Engineering      74.81    465    645
2997 Information Technology & Science     69.30    370    390
   Quant ... ComputerScience MechanicalEngg ElectricalEngg \
0     810 ...          0         0         0
1     210 ...          0         0         0
2     505 ...          0         0         0
3     635 ...          0         0         0
4     365 ...          0         0         0
... ...
2993 445 ...          0         0         0
2994 395 ...          0         0         0
2995 485 ...          0         0         0
2996 505 ...          0         0         0
2997 285 ...          376        0         0
   TelecomEngg CivilEngg conscientiousness agreeableness extraversion \
0           0          0       0.0000      0.3789      1.2396
1           0          0       1.1336      0.0459      1.2396
2           0          0       0.5100      0.0000      1.5428
3           0          0       0.0000      0.2124      0.3174
4           0          0       0.0000      0.0000      0.0000
... ...
2993     0          0       0.0000      0.9688      0.0000
2994     0          0       0.0000      0.0328      0.0000
2995     0          0       0.0000      0.1888      0.0000
2996     0          0       1.4374      1.2888      0.0000
2997     0          0       0.0000      0.0000      0.3174
   neuroticism openness_to_experience
0     0.14590      0.2889
1     0.52620      0.0000
2     0.00000      0.0000
3     0.27270      0.4805
4     0.06223      0.1864
... ...
2993  1.35490      0.0284
2994  0.00000      0.5024
2995  0.06230      0.6603
2996  0.00000      0.5419
2997  1.16010      0.0000
[2989 rows x 23 columns]
0    1
1    0
2    0
3    1
4    0
...
2993  0
2994  0
2995  1
2996  1
2997  0
Name: SalaryCategory, Length: 2989, dtype: int64
```

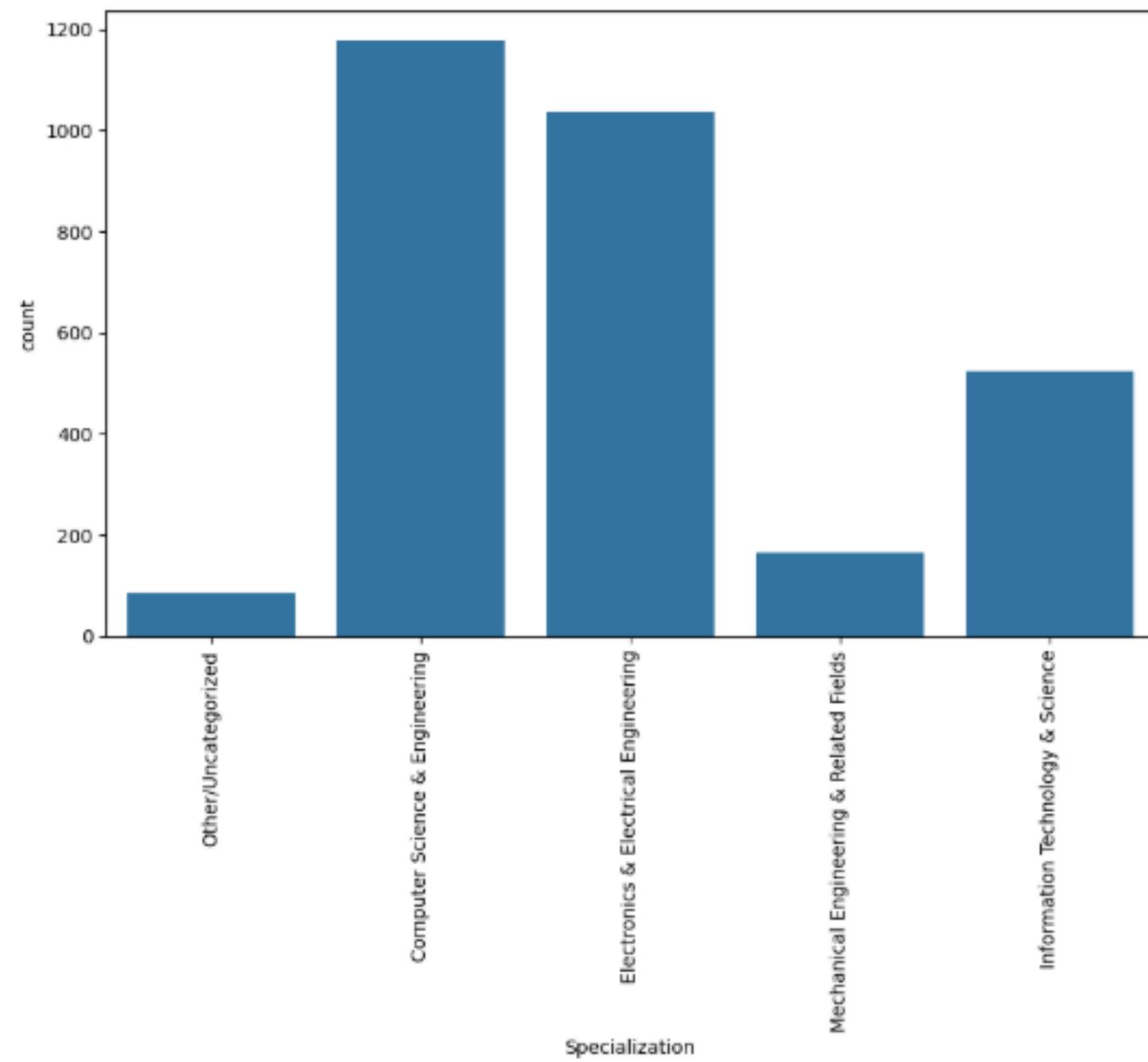
```
X= pd.get_dummies(X,dtype='int',drop_first=True)
print(X)

2994          0  0
2995          0  0
2996          0  0
2997          0  0
   Specialization_Electronics & Electrical Engineering \
0           0
1           0
2           0
3           0
4           0
... ...
2993          1
2994          0
2995          0
2996          0
2997          0
   Specialization_Information Technology & Science \
0           0
1           0
2           0
3           0
4           0
... ...
2993          0
2994          1
2995          1
2996          0
2997          1
   Specialization_Mechanical Engineering & Related Fields \
0           0
1           0
2           0
3           0
4           0
... ...
2993          0
2994          0
2995          0
2996          0
2997          0
   Specialization_Other/Uncategorized
0           1
1           0
2           0
3           0
4           1
... ...
2993          0
2994          0
2995          0
2996          0
2997          0
[2989 rows x 27 columns]
```

EDA

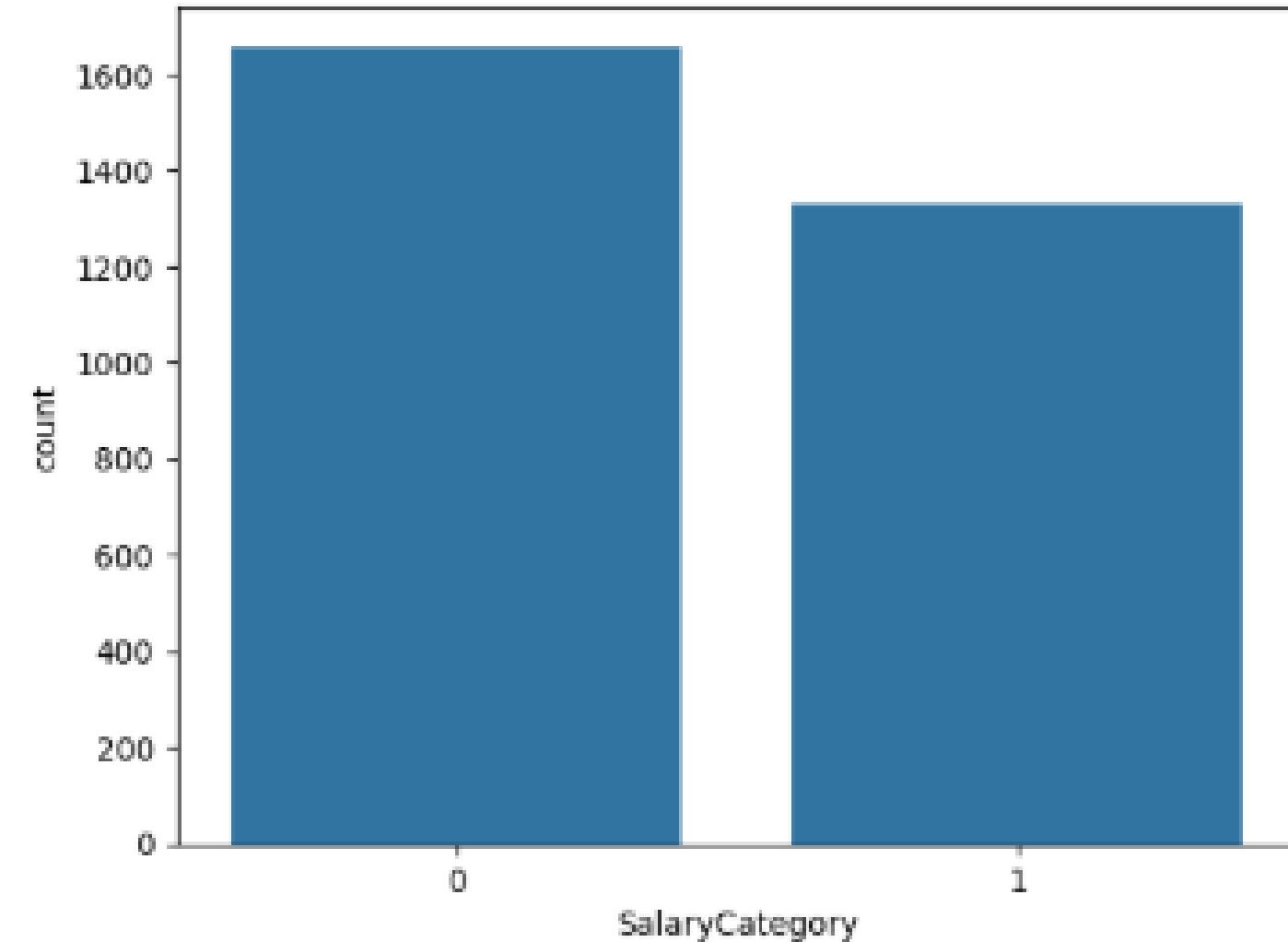
```
❶ plt.figure(figsize=(10, 6))
sns.countplot(x=df['Specialization'])
plt.xticks(rotation=90)

❷ ([0, 1, 2, 3, 4],
 [Text(0, 0, 'Other/Uncategorized'),
  Text(1, 0, 'Computer Science & Engineering'),
  Text(2, 0, 'Electronics & Electrical Engineering'),
  Text(3, 0, 'Mechanical Engineering & Related Fields'),
  Text(4, 0, 'Information Technology & Science')])
```



```
❶ sns.countplot(x='SalaryCategory', data=df)
plt.title('Distribution of Salary Categories')
plt.show()
```

❷ Distribution of Salary Categories



Logistic Regression

```
[ ] from sklearn.model_selection import train_test_split
X_nomulti = X.drop(['10percentage','12percentage','CollegeTier','collegeGPA','English','Logical','Quant','Domain','ComputerProgramming','ElectronicsAndSemicon','Mechanica

X_train1_nomulti, X_test1_nomulti, y_train1_nomulti, y_test1_nomulti = train_test_split(X_nomulti, y, test_size=0.20, train_size=0.80, random_state=1)
X_train2_nomulti, X_test2_nomulti, y_train2_nomulti, y_test2_nomulti = train_test_split(X_nomulti, y, test_size=0.25, train_size=0.75, random_state=1)
X_train3_nomulti, X_test3_nomulti, y_train3_nomulti, y_test3_nomulti = train_test_split(X_nomulti, y, test_size=0.30, train_size=0.70, random_state=1)
X_train4_nomulti, X_test4_nomulti, y_train4_nomulti, y_test4_nomulti = train_test_split(X_nomulti, y, test_size=0.40, train_size=0.60, random_state=1)
X_train5_nomulti, X_test5_nomulti, y_train5_nomulti, y_test5_nomulti = train_test_split(X_nomulti, y, test_size=0.50, train_size=0.50, random_state=1)
```

Logistic Regression after VIF

```
0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 0 1 0 0 0 1  
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1  
0 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0  
0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0  
1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0  
1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0  
1 0 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0  
0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0  
0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0  
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1  
0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0  
0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 1  
0 0 0 0 1 1 0 0 |
```

```
▶ from sklearn.linear_model import LogisticRegression  
logreg = LogisticRegression(C=1e9)
```

```
logreg.fit(X_train4_nomulti, y_train4_nomulti)
predictions4 = logreg.predict(X_test4_nomulti)
print(predictions4)
```

$$\vec{v} = [0 \ 1 \ 1 \ \dots \ 0 \ 0 \ 1]$$

```
[ ] from sklearn.metrics import confusion_matrix
```

```
[ ] z=confusion_matrix(y_test1_nomulti, predictions1)
[ ] z
```

```
array([[228,  80],  
       [203,  87]])
```

```
[ ] z=confusion_matrix(y_test2_nomulti, predictions2)
    z
```

```
array([[288, 108],  
       [231, 121]]))
```

```
[ ] z=confusion_matrix(y_test3_nomulti, predictions3)
z
```

```
array([[349, 118],  
       [285, 145]])
```

```
[ ] z=confusion_matrix(y_test4_nomulti, predictions4)
z
```

```
array([[503, 147],  
       [348, 198]])
```

```
[ ] from sklearn.metrics import accuracy_score  
accuracy_score(y_test1_nomulti,predictions1)
```

[$\frac{1}{\pi}$] = 0.5267558528428093

KNN

```
[ ] from sklearn.neighbors import KNeighborsClassifier  
X_train1_nomulti, X_test1_nomulti, y_train1_nomulti, y_test1_nomulti = train_test_split(X_nomulti, y, test_size=0.20, train_size=0.80, random_state=1)  
X_train2_nomulti, X_test2_nomulti, y_train2_nomulti, y_test2_nomulti = train_test_split(X_nomulti, y, test_size=0.25, train_size=0.75, random_state=1)  
X_train3_nomulti, X_test3_nomulti, y_train3_nomulti, y_test3_nomulti = train_test_split(X_nomulti, y, test_size=0.30, train_size=0.70, random_state=1)  
X_train4_nomulti, X_test4_nomulti, y_train4_nomulti, y_test4_nomulti = train_test_split(X_nomulti, y, test_size=0.40, train_size=0.60, random_state=1)  
X_train5_nomulti, X_test5_nomulti, y_train5_nomulti, y_test5_nomulti = train_test_split(X_nomulti, y, test_size=0.50, train_size=0.50, random_state=1)
```

80-20

```
[ ] model=KNeighborsClassifier(n_neighbors=25)  
model.fit(X_train1_nomulti,y_train1_nomulti)
```

KNeighborsClassifier(n_neighbors=25)

```
y_pred1_nomulti = model.predict(X_test1_nomulti)
print(y_pred1_nomulti)
```

```
[0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1  
0 1 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0  
0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0  
1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0  
1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 1 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0  
1 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0  
0 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1  
0 1 1 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0  
0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0  
0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1  
0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1  
0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 1  
0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 1  
0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1  
0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1  
0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1  
0 0 1 1 1 0 ]
```

```
[ ] df=pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})  
print(df)
```

	Predicted	Actual
1651	0	0
2828	1	1
383	0	1
1601	0	0
2376	0	1
...
303	0	1
1548	1	1
2218	1	1
2662	1	1
134	0	0

[598 rows x 2 columns]

```
[0 0 1 1 1 0]

[ ] df=pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})
print(df)

[+] Predicted Actual
1651      0      0
2828      1      1
383       0      1
1601      0      0
2376      0      1
...
303       0      1
1540      1      1
2210      1      1
2662      1      1
134       0      0
[598 rows x 2 columns]

[ ] from sklearn.metrics import accuracy_score
accuracy_score(y_test1_nomulti,y_pred1_nomulti)

[+] 0.5518394648829431

[ ] from sklearn.metrics import confusion_matrix
cm1_nomulti = confusion_matrix(y_test1_nomulti,y_pred1_nomulti)
cm1_nomulti

[+] array([[206, 102],
       [166, 124]])

[ ] sns.heatmap(cm1_nomulti, annot=True, fmt="d", cmap="coolwarm", xticklabels=["High", "Low"], yticklabels=["High", "Low"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

[+] Confusion Matrix



|        |      | Predicted |     |
|--------|------|-----------|-----|
|        |      | High      | Low |
| Actual | High | 206       | 102 |
|        | Low  | 166       | 124 |


```

SVM

```
[ ] from sklearn.svm import SVC  
X_train1_nomulti, X_test1_nomulti, y_train1_nomulti, y_test1_nomulti = train_test_split(X_nomulti, y, test_size=0.20, train_size=0.80, random_state=1)  
X_train2_nomulti, X_test2_nomulti, y_train2_nomulti, y_test2_nomulti = train_test_split(X_nomulti, y, test_size=0.25, train_size=0.75, random_state=1)  
X_train3_nomulti, X_test3_nomulti, y_train3_nomulti, y_test3_nomulti = train_test_split(X_nomulti, y, test_size=0.30, train_size=0.70, random_state=1)  
X_train4_nomulti, X_test4_nomulti, y_train4_nomulti, y_test4_nomulti = train_test_split(X_nomulti, y, test_size=0.40, train_size=0.60, random_state=1)  
X_train5_nomulti, X_test5_nomulti, y_train5_nomulti, y_test5_nomulti = train_test_split(X_nomulti, y, test_size=0.50, train_size=0.50, random_state=1)  
  
[ ] model = SVC(kernel='sigmoid')  
model.fit(X_train1_nomulti, y_train1_nomulti)
```

```
SVC(kernel='sigmoid')
```

```
y_pred1_nomulti = model.predict(X_test1_nomulti)
print(y_pred1_nomulti)
```

```
[ ] df = pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})
print(df)
```

	Predicted	Actual
1651	0	0
2828	0	1
383	0	1
1601	0	0
2376	0	1
...
303	0	1
1540	0	1
2210	0	1
2662	0	1
134	0	0

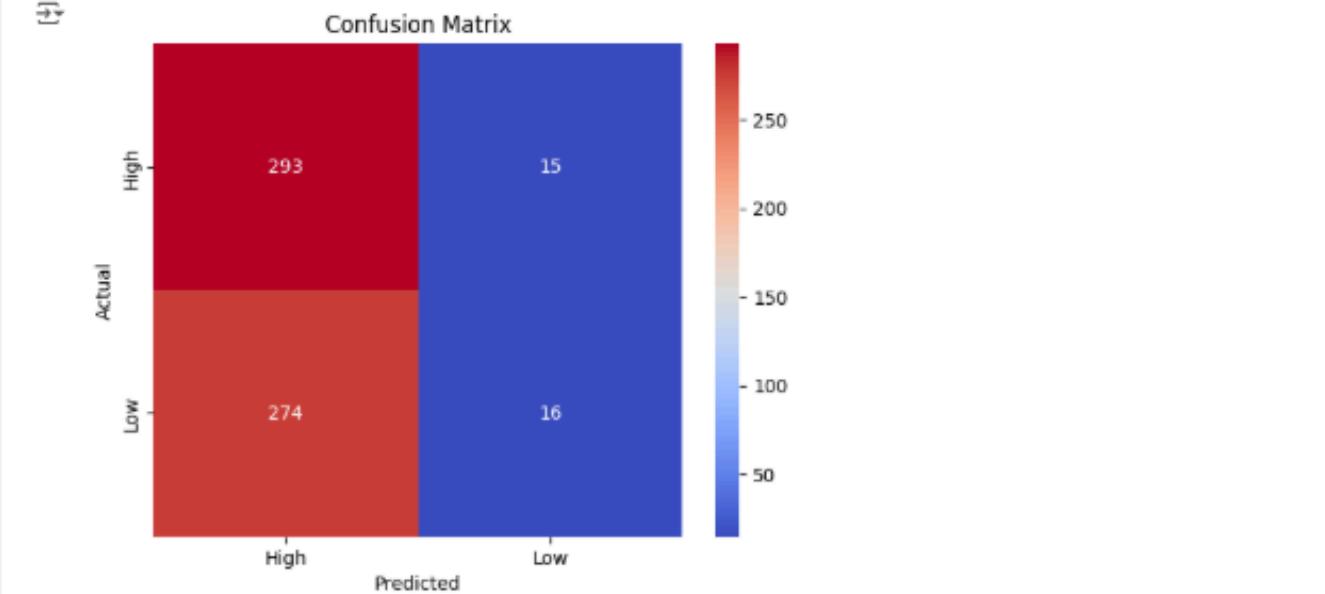
```
[ ] df = pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})  
print(df)
```

	Predicted	Actual
1651	0	0
2828	0	1
383	0	1
1681	0	0
2376	0	1
...
303	0	1
1540	0	1
2210	0	1
2662	0	1
134	0	0

```
[598 rows x 2 columns]
```

```
from sklearn.metrics import confusion_matrix  
cm1_nomulti = confusion_matrix(y_test1_nomulti,y_pred1_nomulti)  
cm1_nomulti
```

```
[ ] sns.heatmap(cm1_nomulti, annot=True, fmt="d", cmap="coolwarm", xticklabels=["High", "Low"], yticklabels=["High", "Low"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Decision tree classifier

```
[ ] import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.20, train_size=0.80,random_state=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75,random_state=1)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70,random_state=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60,random_state=1)
```

80-20

```
[ ] clf = DecisionTreeClassifier()
clf1 = clf.fit(X_train1,y_train1)
print(clf1)
```

```
→ DecisionTreeClassifier()
```

```
● y_pred1 = clf.predict(X_test1)
print(y_pred1)
```

```
→ [1 1 1 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0
0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1 1 1
1 0 0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 1 1 1
0 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 1 0
0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0
0 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1
0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1
0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1
1 1 0 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0
0 1 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 0 1 0
1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1
0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1
1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1
0 1 0 1 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1
0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 0
0 1 1 1 1 1]
```

```
[ ] from sklearn.metrics import classification_report
classification_metrics1 = classification_report(y_test1, y_pred1)
print(classification_metrics1)
```

```
→
precision    recall   f1-score   support
      0       0.62      0.67      0.64      388
      1       0.62      0.57      0.59      290

accuracy                           0.62      598
macro avg       0.62      0.62      0.62      598
weighted avg    0.62      0.62      0.62      598
```

Random Forest Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')

# Display the first few rows of the dataset
df.head()

# Check for missing values
df.isnull().sum()

# Separate the features and target variable
X = df.drop(['species'], axis=1)
y = df['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rfc.fit(X_train, y_train)

# Predict the species for the test set
y_pred = rfc.predict(X_test)

# Print the classification report and confusion matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

XG Boost classifier

```
XG BOOST

[ ] import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.20, train_size=0.80,random_state=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75,random_state=1)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70,random_state=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60,random_state=1)

[ ] model = xgb.XGBClassifier()
model = model.fit(X_train1, y_train1)

❶ y_pred1 = model.predict(X_test1)
print(y_pred1)

[ ] [0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0
1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1
1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 1
0 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0
1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0
1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0
0 1 1 0 0 1 1 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 1 0 0 1 1 1 1
0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1
0 1 1 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 1 1 1 0 0 1
0 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 0 0
0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 0 0 0
0 0 1 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 1
0 0 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1
0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1
0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1
0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0
1 1 0 1 1 0]]

[ ] print(classification_report(y_test1, y_pred1))

[ ] precision    recall    f1-score   support
      0       0.69      0.75      0.72      308
      1       0.71      0.65      0.68      290

  accuracy         0.70      0.70      0.70      598
 macro avg       0.70      0.70      0.70      598
weighted avg     0.70      0.70      0.70      598

[ ] print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))

[ ] Accuracy: 0.7006688963210702
```



ADA Boost Classifier

```
[1]: import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.20, train_size=0.80,random_state=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75,random_state=1)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70,random_state=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60,random_state=1)

0-20

[2]: model = AdaBoostClassifier(n_estimators=50, random_state=42)
model.fit(X_train1, y_train1)

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated
  warnings.warn(
+     AdaBoostClassifier    ⓘ ⓘ
AdaBoostClassifier(random_state=42)

[3]: y_pred1 = model.predict(X_test1)
print(y_pred1)

[4]: [0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0
 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1
 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0
 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1
 0 0 1 1 0 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0 1 1 0 0 0 1
 1 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0
 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 1
 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 0
 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1
 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1
 1 1 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0
 0 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 1
 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 0
 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 1
 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1
 1 1 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0
 0 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1
 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1
 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 1
 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 1 1
 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1
 1 1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0]
```

Neural Networks :

```
[ ] import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt

[ ] from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.20, train_size=0.80,random_state=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75,random_state=1)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70,random_state=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60,random_state=1)

80-20

[ ] tf.random.set_seed(42)

# STEP1: Creating the model

model_nom1= tf.keras.Sequential([
    tf.keras.layers.Dense(30 ,activation='relu'),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(7, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model_nom1.compile(loss= tf.keras.losses.binary_crossentropy,
                    optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
                    metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                            tf.keras.metrics.Precision(name='precision'),
                            tf.keras.metrics.Recall(name='a=recall')
                           ]
                  )

# STEP1: Fit the model

history= model_nom1.fit(X_train1, y_train1, epochs= 300)
```

Epoch 1/300
75/75 6s 8ms/step - a=recall: 0.4825 - accuracy: 0.5300 - loss: 10.1588 - precision: 0.4577
Epoch 2/300
75/75 0s 6ms/step - a=recall: 0.4778 - accuracy: 0.5541 - loss: 2.4216 - precision: 0.4832
Epoch 3/300
75/75 1s 5ms/step - a=recall: 0.5086 - accuracy: 0.5671 - loss: 1.6361 - precision: 0.4980
Epoch 4/300
75/75 0s 4ms/step - a=recall: 0.5172 - accuracy: 0.5729 - loss: 1.3974 - precision: 0.5058
Epoch 5/300
75/75 1s 3ms/step - a=recall: 0.5306 - accuracy: 0.5701 - loss: 1.3800 - precision: 0.5022
Epoch 6/300
75/75 0s 2ms/step - a=recall: 0.5215 - accuracy: 0.5710 - loss: 1.1996 - precision: 0.5046
Epoch 7/300
75/75 0s 2ms/step - a=recall: 0.5351 - accuracy: 0.5703 - loss: 1.1305 - precision: 0.5022
Epoch 8/300
75/75 0s 2ms/step - a=recall: 0.5342 - accuracy: 0.5740 - loss: 1.0641 - precision: 0.5065

