**Fraudulent Claim Detection**

**Problem Statement**

Global Insure, a leading insurance company, processes thousands of claims annually. However, a significant percentage of these claims turn out to be fraudulent, resulting in considerable financial losses. The company’s current process for identifying fraudulent claims involves manual inspections, which is time-consuming and inefficient. Fraudulent claims are often detected too late in the process, after the company has already paid out significant amounts. Global Insure wants to improve its fraud detection process using data-driven insights to classify claims as fraudulent or legitimate early in the approval process. This would minimise financial losses and optimise the overall claims handling process.

**Business Objective**

Global Insure wants to build a model to classify insurance claims as either fraudulent or legitimate based on historical claim details and customer profiles. By using features like claim amounts, customer profiles and claim types, the company aims to predict which claims are likely to be fraudulent before they are approved.

Based on this assignment, you have to answer the following questions:

● How can we analyse historical claim data to detect patterns that indicate fraudulent claims?  
● Which features are most predictive of fraudulent behaviour?  
● Can we predict the likelihood of fraud for an incoming claim, based on past data?  
● What insights can be drawn from the model that can help in improving the fraud detection process?

**Assignment Tasks**

You need to perform the following steps for successfully completing this assignment:

1. Data Preparation
2. Data Cleaning
3. Train Validation Split 70-30
4. EDA on Training Data
5. EDA on Validation Data (optional)
6. Feature Engineering
7. Model Building
8. Predicting and Model Evaluation

**Data Dictionary**

The insurance claims data has 40 Columns and 1000 Rows. Following data dictionary provides the description for each column present in dataset:

| **Column Name** | **Description** |
| --- | --- |
| months\_as\_customer | Represents the duration in months that a customer has been associated with the insurance company. |
| age | Represents the age of the insured person. |
| policy\_number | Represents a unique identifier for each insurance policy. |
| policy\_bind\_date | Represents the date when the insurance policy was initiated. |
| policy\_state | Represents the state where the insurance policy is applicable. |
| policy\_csl | Represents the combined single limit for the insurance policy. |
| policy\_deductable | Represents the amount that the insured person needs to pay before the insurance coverage kicks in. |
| policy\_annual\_premium | Represents the yearly cost of the insurance policy. |
| umbrella\_limit | Represents an additional layer of liability coverage provided beyond the limits of the primary insurance policy. |
| insured\_zip | Represents the zip code of the insured person. |
| insured\_sex | Represents the gender of the insured person. |
| insured\_education\_level | Represents the highest educational qualification of the insured person. |
| insured\_occupation | Represents the profession or job of the insured person. |
| insured\_hobbies | Represents the hobbies or leisure activities of the insured person. |
| insured\_relationship | Represents the relationship of the insured person to the policyholder. |
| capital-gains | Represents the profit earned from the sale of assets such as stocks, bonds, or real estate. |
| capital-loss | Represents the loss incurred from the sale of assets such as stocks, bonds, or real estate. |
| incident\_date | Represents the date when the incident or accident occurred. |
| incident\_type | Represents the category or type of incident that led to the claim. |
| collision\_type | Represents the type of collision that occurred in an accident. |
| incident\_severity | Represents the extent of damage or injury caused by the incident. |
| authorities\_contacted | Represents the authorities or agencies that were contacted after the incident. |
| incident\_state | Represents the state where the incident occurred. |
| incident\_city | Represents the city where the incident occurred. |
| incident\_location | Represents the specific location or address where the incident occurred. |
| incident\_hour\_of\_the\_day | Represents the hour of the day when the incident occurred. |
| number\_of\_vehicles\_involved | Represents the total number of vehicles involved in the incident. |
| property\_damage | Represents whether there was any damage to property in the incident. |
| bodily\_injuries | Represents the number of bodily injuries resulting from the incident. |
| witnesses | Represents the number of witnesses present at the scene of the incident. |
| police\_report\_available | Represents whether a police report is available for the incident. |
| total\_claim\_amount | Represents the total amount claimed by the insured person for the incident. |
| injury\_claim | Represents the amount claimed for injuries sustained in the incident. |
| property\_claim | Represents the amount claimed for property damage in the incident. |
| vehicle\_claim | Represents the amount claimed for vehicle damage in the incident. |
| auto\_make | Represents the manufacturer of the insured vehicle. |
| auto\_model | Represents the specific model of the insured vehicle. |
| auto\_year | Represents the year of manufacture of the insured vehicle. |
| fraud\_reported | Represents whether the claim was reported as fraudulent or not. |
| \_c39 | Represents an unknown or unspecified variable. |

**1. Data Preparation**

In this step, read the dataset provided in CSV format and look at basic statistics of the data, including preview of data, dimension of data, column descriptions and data types.

**1.0 Import Libraries**

In [242]:

*# Supress unnecessary warnings*

**import** warnings

warnings**.**filterwarnings("ignore")

In [243]:

*# Import necessary libraries*

**import** numpy **as** np

**import** pandas **as** pd

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**1.1 Load the Data**

In [244]:

*# Load the dataset*

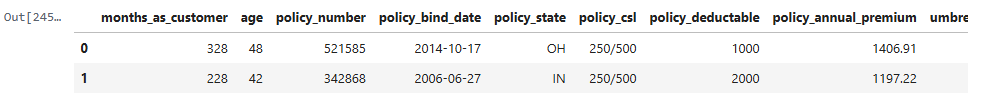
df**=**pd**.**read\_csv("insurance\_claims.csv")

In [245]:

*# Check at the first few entries*

df**.**head(2)

Out[245]:



2 rows × 40 columns

In [246]:

*# Inspect the shape of the dataset*

df**.**shape

Out[246]:

(1000, 40)

In [247]:

*# Inspect the features in the dataset*

df**.**columns

Out[247]:

Index(['months\_as\_customer', 'age', 'policy\_number', 'policy\_bind\_date',

'policy\_state', 'policy\_csl', 'policy\_deductable',

'policy\_annual\_premium', 'umbrella\_limit', 'insured\_zip', 'insured\_sex',

'insured\_education\_level', 'insured\_occupation', 'insured\_hobbies',

'insured\_relationship', 'capital-gains', 'capital-loss',

'incident\_date', 'incident\_type', 'collision\_type', 'incident\_severity',

'authorities\_contacted', 'incident\_state', 'incident\_city',

'incident\_location', 'incident\_hour\_of\_the\_day',

'number\_of\_vehicles\_involved', 'property\_damage', 'bodily\_injuries',

'witnesses', 'police\_report\_available', 'total\_claim\_amount',

'injury\_claim', 'property\_claim', 'vehicle\_claim', 'auto\_make',

'auto\_model', 'auto\_year', 'fraud\_reported', '\_c39'],

dtype='object')

**2. Data Cleaning [10 marks]**

**2.1 Handle null values [2 marks]**

**2.1.1 Examine the columns to determine if any value or column needs to be treated [1 Mark]**

In [248]:

*# Check the number of missing values in each column*

df**.**isnull()**.**mean()**\***100

Out[248]:

months\_as\_customer 0.0

age 0.0

policy\_number 0.0

policy\_bind\_date 0.0

policy\_state 0.0

policy\_csl 0.0

policy\_deductable 0.0

policy\_annual\_premium 0.0

umbrella\_limit 0.0

insured\_zip 0.0

insured\_sex 0.0

insured\_education\_level 0.0

insured\_occupation 0.0

insured\_hobbies 0.0

insured\_relationship 0.0

capital-gains 0.0

capital-loss 0.0

incident\_date 0.0

incident\_type 0.0

collision\_type 0.0

incident\_severity 0.0

authorities\_contacted 0.0

incident\_state 0.0

incident\_city 0.0

incident\_location 0.0

incident\_hour\_of\_the\_day 0.0

number\_of\_vehicles\_involved 0.0

property\_damage 0.0

bodily\_injuries 0.0

witnesses 0.0

police\_report\_available 0.0

total\_claim\_amount 0.0

injury\_claim 0.0

property\_claim 0.0

vehicle\_claim 0.0

auto\_make 0.0

auto\_model 0.0

auto\_year 0.0

fraud\_reported 0.0

\_c39 100.0

dtype: float64

**2.1.2 Handle rows containing null values [1 Mark]**

In [249]:

*# Handle the rows containing null values*

df**.**drop('\_c39',axis**=**1,inplace**=True**)

df**.**shape

Out[249]:

(1000, 39)

**2.2 Identify and handle redundant values and columns [5 marks]**

**2.2.1 Examine the columns to determine if any value or column needs to be treated [2 Mark]**

In [250]:

*# Write code to display all the columns with their unique values and counts and check for redundant values*

In [251]:

*# Loop through all columns and show unique values and their counts*

**for** col **in** df**.**columns:

print(f"\nColumn: {col}")

print("Unique Values and Counts:")

print(df[col]**.**value\_counts(dropna**=False**))

print(f"Total Unique: {df[col]**.**nunique(dropna**=False**)}")

Column: months\_as\_customer

Unique Values and Counts:

194 8

285 7

140 7

230 7

128 7

..

347 1

113 1

337 1

117 1

0 1

Name: months\_as\_customer, Length: 391, dtype: int64

Total Unique: 391

Column: age

Unique Values and Counts:

43 49

39 48

41 45

34 44

30 42

31 42

38 42

37 41

33 39

32 38

40 38

29 35

46 33

35 32

36 32

42 32

44 32

28 30

45 26

26 26

48 25

47 24

27 24

57 16

25 14

49 14

55 14

50 13

53 13

61 10

24 10

54 10

60 9

51 9

58 8

56 8

23 7

21 6

59 5

62 4

52 4

64 2

63 2

22 1

20 1

19 1

Name: age, dtype: int64

Total Unique: 46

Column: policy\_number

Unique Values and Counts:

116735 1

107181 1

430794 1

115399 1

328387 1

..

218456 1

179538 1

357713 1

247116 1

296960 1

Name: policy\_number, Length: 1000, dtype: int64

Total Unique: 1000

Column: policy\_bind\_date

Unique Values and Counts:

2006-01-01 3

1992-04-28 3

1992-08-05 3

1997-11-07 2

2010-03-11 2

..

1994-09-17 1

2011-08-25 1

1997-08-04 1

2003-05-29 1

1998-11-13 1

Name: policy\_bind\_date, Length: 951, dtype: int64

Total Unique: 951

Column: policy\_state

Unique Values and Counts:

OH 352

IL 338

IN 310

Name: policy\_state, dtype: int64

Total Unique: 3

Column: policy\_csl

Unique Values and Counts:

250/500 351

100/300 349

500/1000 300

Name: policy\_csl, dtype: int64

Total Unique: 3

Column: policy\_deductable

Unique Values and Counts:

1000 351

500 342

2000 307

Name: policy\_deductable, dtype: int64

Total Unique: 3

Column: policy\_annual\_premium

Unique Values and Counts:

1374.22 2

1558.29 2

1389.13 2

1073.83 2

1074.07 2

..

1268.79 1

1558.86 1

722.66 1

1302.40 1

1212.00 1

Name: policy\_annual\_premium, Length: 991, dtype: int64

Total Unique: 991

Column: umbrella\_limit

Unique Values and Counts:

0 798

6000000 57

5000000 46

4000000 39

7000000 29

3000000 12

8000000 8

9000000 5

2000000 3

10000000 2

-1000000 1

Name: umbrella\_limit, dtype: int64

Total Unique: 11

Column: insured\_zip

Unique Values and Counts:

446895 2

456602 2

477695 2

469429 2

431202 2

..

468313 1

474360 1

476502 1

460895 1

454656 1

Name: insured\_zip, Length: 995, dtype: int64

Total Unique: 995

Column: insured\_sex

Unique Values and Counts:

FEMALE 537

MALE 463

Name: insured\_sex, dtype: int64

Total Unique: 2

Column: insured\_education\_level

Unique Values and Counts:

JD 161

High School 160

Associate 145

MD 144

Masters 143

PhD 125

College 122

Name: insured\_education\_level, dtype: int64

Total Unique: 7

Column: insured\_occupation

Unique Values and Counts:

machine-op-inspct 93

prof-specialty 85

tech-support 78

sales 76

exec-managerial 76

craft-repair 74

transport-moving 72

priv-house-serv 71

other-service 71

armed-forces 69

adm-clerical 65

protective-serv 63

handlers-cleaners 54

farming-fishing 53

Name: insured\_occupation, dtype: int64

Total Unique: 14

Column: insured\_hobbies

Unique Values and Counts:

reading 64

exercise 57

paintball 57

bungie-jumping 56

camping 55

golf 55

movies 55

kayaking 54

yachting 53

hiking 52

video-games 50

base-jumping 49

skydiving 49

board-games 48

polo 47

chess 46

dancing 43

sleeping 41

cross-fit 35

basketball 34

Name: insured\_hobbies, dtype: int64

Total Unique: 20

Column: insured\_relationship

Unique Values and Counts:

own-child 183

other-relative 177

not-in-family 174

husband 170

wife 155

unmarried 141

Name: insured\_relationship, dtype: int64

Total Unique: 6

Column: capital-gains

Unique Values and Counts:

0 508

46300 5

68500 4

51500 4

48900 3

...

47700 1

94800 1

90700 1

100500 1

54800 1

Name: capital-gains, Length: 338, dtype: int64

Total Unique: 338

Column: capital-loss

Unique Values and Counts:

0 475

-53700 5

-50300 5

-31700 5

-49200 4

...

-32600 1

-46800 1

-50600 1

-40800 1

-43900 1

Name: capital-loss, Length: 354, dtype: int64

Total Unique: 354

Column: incident\_date

Unique Values and Counts:

2015-02-02 28

2015-02-17 26

2015-01-07 25

2015-01-24 24

2015-02-04 24

2015-01-10 24

2015-01-19 23

2015-01-08 22

2015-01-30 21

2015-01-13 21

2015-01-31 20

2015-02-12 20

2015-02-22 20

2015-02-06 20

2015-02-21 19

2015-01-01 19

2015-01-14 19

2015-02-23 19

2015-01-12 19

2015-01-21 19

2015-02-25 18

2015-01-18 18

2015-02-28 18

2015-02-01 18

2015-02-14 18

2015-01-03 18

2015-01-20 18

2015-01-06 17

2015-02-24 17

2015-02-26 17

2015-01-09 17

2015-02-08 17

2015-02-05 16

2015-02-15 16

2015-02-13 16

2015-01-16 16

2015-02-16 16

2015-01-15 15

2015-02-18 15

2015-01-17 15

2015-01-28 15

2015-02-27 14

2015-01-22 14

2015-02-20 14

2015-01-27 13

2015-02-03 13

2015-02-09 13

2015-01-23 13

2015-01-04 12

2015-03-01 12

2015-01-26 11

2015-01-29 11

2015-01-02 11

2015-02-07 10

2015-01-25 10

2015-02-19 10

2015-02-10 10

2015-02-11 10

2015-01-11 9

2015-01-05 7

Name: incident\_date, dtype: int64

Total Unique: 60

Column: incident\_type

Unique Values and Counts:

Multi-vehicle Collision 419

Single Vehicle Collision 403

Vehicle Theft 94

Parked Car 84

Name: incident\_type, dtype: int64

Total Unique: 4

Column: collision\_type

Unique Values and Counts:

Rear Collision 292

Side Collision 276

Front Collision 254

? 178

Name: collision\_type, dtype: int64

Total Unique: 4

Column: incident\_severity

Unique Values and Counts:

Minor Damage 354

Total Loss 280

Major Damage 276

Trivial Damage 90

Name: incident\_severity, dtype: int64

Total Unique: 4

Column: authorities\_contacted

Unique Values and Counts:

Police 292

Fire 223

Other 198

Ambulance 196

None 91

Name: authorities\_contacted, dtype: int64

Total Unique: 5

Column: incident\_state

Unique Values and Counts:

NY 262

SC 248

WV 217

VA 110

NC 110

PA 30

OH 23

Name: incident\_state, dtype: int64

Total Unique: 7

Column: incident\_city

Unique Values and Counts:

Springfield 157

Arlington 152

Columbus 149

Northbend 145

Hillsdale 141

Riverwood 134

Northbrook 122

Name: incident\_city, dtype: int64

Total Unique: 7

Column: incident\_location

Unique Values and Counts:

6971 Best Ridge 1

9910 Maple Ave 1

4625 MLK Drive 1

2886 Tree Ridge 1

3323 1st Lane 1

..

7135 Flute Lane 1

2644 MLK Drive 1

6717 Best Drive 1

6494 4th Ave 1

9169 Pine Ridge 1

Name: incident\_location, Length: 1000, dtype: int64

Total Unique: 1000

Column: incident\_hour\_of\_the\_day

Unique Values and Counts:

17 54

3 53

0 52

23 51

16 49

4 46

13 46

10 46

6 44

9 43

14 43

21 42

18 41

7 40

19 40

12 40

15 39

22 38

8 36

20 34

5 33

2 31

11 30

1 29

Name: incident\_hour\_of\_the\_day, dtype: int64

Total Unique: 24

Column: number\_of\_vehicles\_involved

Unique Values and Counts:

1 581

3 358

4 31

2 30

Name: number\_of\_vehicles\_involved, dtype: int64

Total Unique: 4

Column: property\_damage

Unique Values and Counts:

? 360

NO 338

YES 302

Name: property\_damage, dtype: int64

Total Unique: 3

Column: bodily\_injuries

Unique Values and Counts:

0 340

2 332

1 328

Name: bodily\_injuries, dtype: int64

Total Unique: 3

Column: witnesses

Unique Values and Counts:

1 258

2 250

0 249

3 243

Name: witnesses, dtype: int64

Total Unique: 4

Column: police\_report\_available

Unique Values and Counts:

? 343

NO 343

YES 314

Name: police\_report\_available, dtype: int64

Total Unique: 3

Column: total\_claim\_amount

Unique Values and Counts:

59400 5

75400 4

60600 4

2640 4

58500 4

..

57970 1

41580 1

45180 1

3690 1

71680 1

Name: total\_claim\_amount, Length: 763, dtype: int64

Total Unique: 763

Column: injury\_claim

Unique Values and Counts:

0 25

480 7

640 7

580 5

6340 5

..

16820 1

5960 1

10840 1

8000 1

5530 1

Name: injury\_claim, Length: 638, dtype: int64

Total Unique: 638

Column: property\_claim

Unique Values and Counts:

0 19

860 6

660 5

480 5

10000 5

..

17880 1

1500 1

14080 1

7850 1

21630 1

Name: property\_claim, Length: 626, dtype: int64

Total Unique: 626

Column: vehicle\_claim

Unique Values and Counts:

5040 7

3360 6

3600 5

44800 5

33600 5

..

46680 1

3640 1

34320 1

40530 1

51200 1

Name: vehicle\_claim, Length: 726, dtype: int64

Total Unique: 726

Column: auto\_make

Unique Values and Counts:

Dodge 80

Suburu 80

Saab 80

Nissan 78

Chevrolet 76

BMW 72

Ford 72

Toyota 70

Audi 69

Accura 68

Volkswagen 68

Jeep 67

Mercedes 65

Honda 55

Name: auto\_make, dtype: int64

Total Unique: 14

Column: auto\_model

Unique Values and Counts:

RAM 43

Wrangler 42

A3 37

Neon 37

MDX 36

Jetta 35

Passat 33

A5 32

Legacy 32

Pathfinder 31

Malibu 30

92x 28

Forrestor 28

Camry 28

E400 27

95 27

F150 27

Grand Cherokee 25

93 25

Tahoe 24

Escape 24

Maxima 24

Ultima 23

X5 23

Highlander 22

Civic 22

Silverado 22

Fusion 21

TL 20

CRV 20

ML350 20

Corolla 20

Impreza 20

3 Series 18

C300 18

X6 16

M5 15

Accord 13

RSX 12

Name: auto\_model, dtype: int64

Total Unique: 39

Column: auto\_year

Unique Values and Counts:

1995 56

1999 55

2005 54

2011 53

2006 53

2007 52

2003 51

2010 50

2009 50

2013 49

2002 49

2015 47

1997 46

2012 46

2008 45

2014 44

2001 42

2000 42

1998 40

2004 39

1996 37

Name: auto\_year, dtype: int64

Total Unique: 21

Column: fraud\_reported

Unique Values and Counts:

N 753

Y 247

Name: fraud\_reported, dtype: int64

Total Unique: 2

**2.2.2 Identify and drop any columns that are completely empty [1 Mark]**

In [252]:

*# Identify and drop any columns that are completely empty*

*#there are no columns, \_c39 was one column which has been dropped.*

**2.2.3 Identify and drop rows where features have illogical or invalid values, such as negative values for features that should only have positive values [1 Mark]**

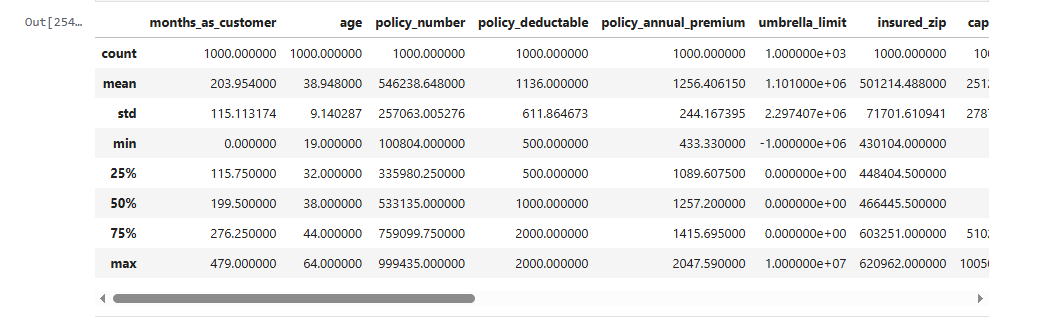
In [253]:

*# Identify and drop rows where features have illogical or invalid values, such as negative values for features that should only have positive values*

In [254]:

df**.**describe()

Out[254]:



In [255]:

*# some columns have invalid values '?', replacing it with null values*

cols**=**['collision\_type' ,'property\_damage','police\_report\_available']

**for** col **in** cols:

df[col]**=**df[col]**.**replace("?",np**.**nan)

In [256]:

df['capital-loss']**.**describe()

Out[256]:

count 1000.000000

mean -26793.700000

std 28104.096686

min -111100.000000

25% -51500.000000

50% -23250.000000

75% 0.000000

max 0.000000

Name: capital-loss, dtype: float64

In [257]:

df**.**drop('capital-loss',axis**=**1, inplace**=True**)

In [258]:

df['umbrella\_limit']**.**value\_counts()

Out[258]:

0 798

6000000 57

5000000 46

4000000 39

7000000 29

3000000 12

8000000 8

9000000 5

2000000 3

10000000 2

-1000000 1

Name: umbrella\_limit, dtype: int64

In [259]:

df**=**df[df['umbrella\_limit']**!=-**1000000]

In [260]:

*#SOME ROWS HAVE NULL WHCIH NEEDS TO BE TREATED*

df**.**isnull()**.**mean()**\***100

Out[260]:

months\_as\_customer 0.000000

age 0.000000

policy\_number 0.000000

policy\_bind\_date 0.000000

policy\_state 0.000000

policy\_csl 0.000000

policy\_deductable 0.000000

policy\_annual\_premium 0.000000

umbrella\_limit 0.000000

insured\_zip 0.000000

insured\_sex 0.000000

insured\_education\_level 0.000000

insured\_occupation 0.000000

insured\_hobbies 0.000000

insured\_relationship 0.000000

capital-gains 0.000000

incident\_date 0.000000

incident\_type 0.000000

collision\_type 17.817818

incident\_severity 0.000000

authorities\_contacted 0.000000

incident\_state 0.000000

incident\_city 0.000000

incident\_location 0.000000

incident\_hour\_of\_the\_day 0.000000

number\_of\_vehicles\_involved 0.000000

property\_damage 36.036036

bodily\_injuries 0.000000

witnesses 0.000000

police\_report\_available 34.234234

total\_claim\_amount 0.000000

injury\_claim 0.000000

property\_claim 0.000000

vehicle\_claim 0.000000

auto\_make 0.000000

auto\_model 0.000000

auto\_year 0.000000

fraud\_reported 0.000000

dtype: float64

In [261]:

df[cols]**.**describe()

Out[261]:

|  | **collision\_type** | **property\_damage** | **police\_report\_available** |
| --- | --- | --- | --- |
| **count** | 821 | 639 | 657 |
| **unique** | 3 | 2 | 2 |
| **top** | Rear Collision | NO | NO |
| **freq** | 292 | 338 | 343 |

In [262]:

*# missing value treatment using fillna*

df['collision\_type']**.**fillna(df['collision\_type']**.**mode()[0], inplace **=** **True**)

*# It may be the case that there are no responses for property damage then we might take it as No property damage.*

df['property\_damage']**.**fillna(df['property\_damage']**.**mode()[0], inplace **=** **True**)

*# again, if there are no responses fpr police report available then we might take it as No report available*

df['police\_report\_available']**.**fillna(df['police\_report\_available']**.**mode()[0], inplace **=** **True**)

df**.**isnull()**.**any()**.**any()

Out[262]:

False

**2.2.4 Identify and remove columns where a large proportion of the values are unique or near-unique, as these columns are likely to be identifiers or have very limited predictive power [1 Mark]**

In [263]:

*# Identify and remove columns that are likely to be identifiers or have very limited predictive power*

df**.**drop(['incident\_location','insured\_zip','policy\_number'],axis**=**1, inplace**=True**)

In [264]:

*# Check the dataset*

df**.**shape

Out[264]:

(999, 35)

**2.3 Fix Data Types [3 marks]**

Carefully examine the dataset and identify columns that contain date or time information but are not stored as the appropriate data type. Convert these columns to the correct datetime data type to enable proper analysis and manipulation of temporal information.

In [265]:

*# Fix the data types of the columns with incorrect data types*

df**.**dtypes

Out[265]:

months\_as\_customer int64

age int64

policy\_bind\_date object

policy\_state object

policy\_csl object

policy\_deductable int64

policy\_annual\_premium float64

umbrella\_limit int64

insured\_sex object

insured\_education\_level object

insured\_occupation object

insured\_hobbies object

insured\_relationship object

capital-gains int64

incident\_date object

incident\_type object

collision\_type object

incident\_severity object

authorities\_contacted object

incident\_state object

incident\_city object

incident\_hour\_of\_the\_day int64

number\_of\_vehicles\_involved int64

property\_damage object

bodily\_injuries int64

witnesses int64

police\_report\_available object

total\_claim\_amount int64

injury\_claim int64

property\_claim int64

vehicle\_claim int64

auto\_make object

auto\_model object

auto\_year int64

fraud\_reported object

dtype: object

In [266]:

date\_cols**=**['policy\_bind\_date','incident\_date']

**for** col **in** date\_cols:

df[col] **=** pd**.**to\_datetime(df[col], errors**=**'coerce')

cols**=**['bodily\_injuries','number\_of\_vehicles\_involved','witnesses','auto\_year','policy\_deductable','umbrella\_limit']

df[cols]**=**df[cols]**.**astype('object')

In [267]:

*# df.describe(include=object)*

In [268]:

*# Check the features of the data again*

df**.**dtypes

Out[268]:

months\_as\_customer int64

age int64

policy\_bind\_date datetime64[ns]

policy\_state object

policy\_csl object

policy\_deductable object

policy\_annual\_premium float64

umbrella\_limit object

insured\_sex object

insured\_education\_level object

insured\_occupation object

insured\_hobbies object

insured\_relationship object

capital-gains int64

incident\_date datetime64[ns]

incident\_type object

collision\_type object

incident\_severity object

authorities\_contacted object

incident\_state object

incident\_city object

incident\_hour\_of\_the\_day int64

number\_of\_vehicles\_involved object

property\_damage object

bodily\_injuries object

witnesses object

police\_report\_available object

total\_claim\_amount int64

injury\_claim int64

property\_claim int64

vehicle\_claim int64

auto\_make object

auto\_model object

auto\_year object

fraud\_reported object

dtype: object

**3. Train-Validation Split [5 marks]**

**3.1 Import required libraries**

In [269]:

*# Import train-test-split*

**from** sklearn.model\_selection **import** train\_test\_split

In [270]:

df**.**shape

Out[270]:

(999, 35)

**3.2 Define feature and target variables [2 Marks]**

In [271]:

*# Put all the feature variables in X*

*# Put the target variable in y*

y**=**df['fraud\_reported']

X**=**df**.**drop('fraud\_reported',axis**=**1)

**3.3 Split the data [3 Marks]**

In [272]:

*# Split the dataset into 70% train and 30% validation and use stratification on the target variable*

X\_train, X\_val, y\_train, y\_val **=** train\_test\_split(

X, y,

test\_size**=**0.3,

stratify**=**y,

random\_state**=**42

)

*# Reset index for all train and test sets*

*# Reset index for features*

X\_train **=** X\_train**.**reset\_index(drop**=True**)

X\_val **=** X\_val**.**reset\_index(drop**=True**)

*# Reset index for targets*

y\_train **=** y\_train**.**reset\_index(drop**=True**)

y\_val **=** y\_val**.**reset\_index(drop**=True**)

**4. EDA on training data [20 marks]**

**4.1 Perform univariate analysis [5 marks]**

**4.1.1 Identify and select numerical columns from training data for univariate analysis [1 Mark]**

In [273]:

*# df.describe()*

In [274]:

*# df.dtypes*

In [275]:

*# Select numerical columns*

numeric\_df **=** df**.**\_get\_numeric\_data()

numeric\_columns**=**numeric\_df**.**columns

**4.1.2 Visualise the distribution of selected numerical features using appropriate plots to understand their characteristics [4 Marks]**

In [276]:

numeric\_columns

Out[276]:

Index(['months\_as\_customer', 'age', 'policy\_annual\_premium', 'capital-gains',

'incident\_hour\_of\_the\_day', 'total\_claim\_amount', 'injury\_claim',

'property\_claim', 'vehicle\_claim'],

dtype='object')

In [277]:

*# Plot all the numerical columns to understand their distribution*

plt**.**figure(figsize**=**(15, len(numeric\_columns) **\*** 3))

**for** i, col **in** enumerate(numeric\_columns, 1):

plt**.**subplot(len(numeric\_columns), 1, i) *# Create a subplot for each column*

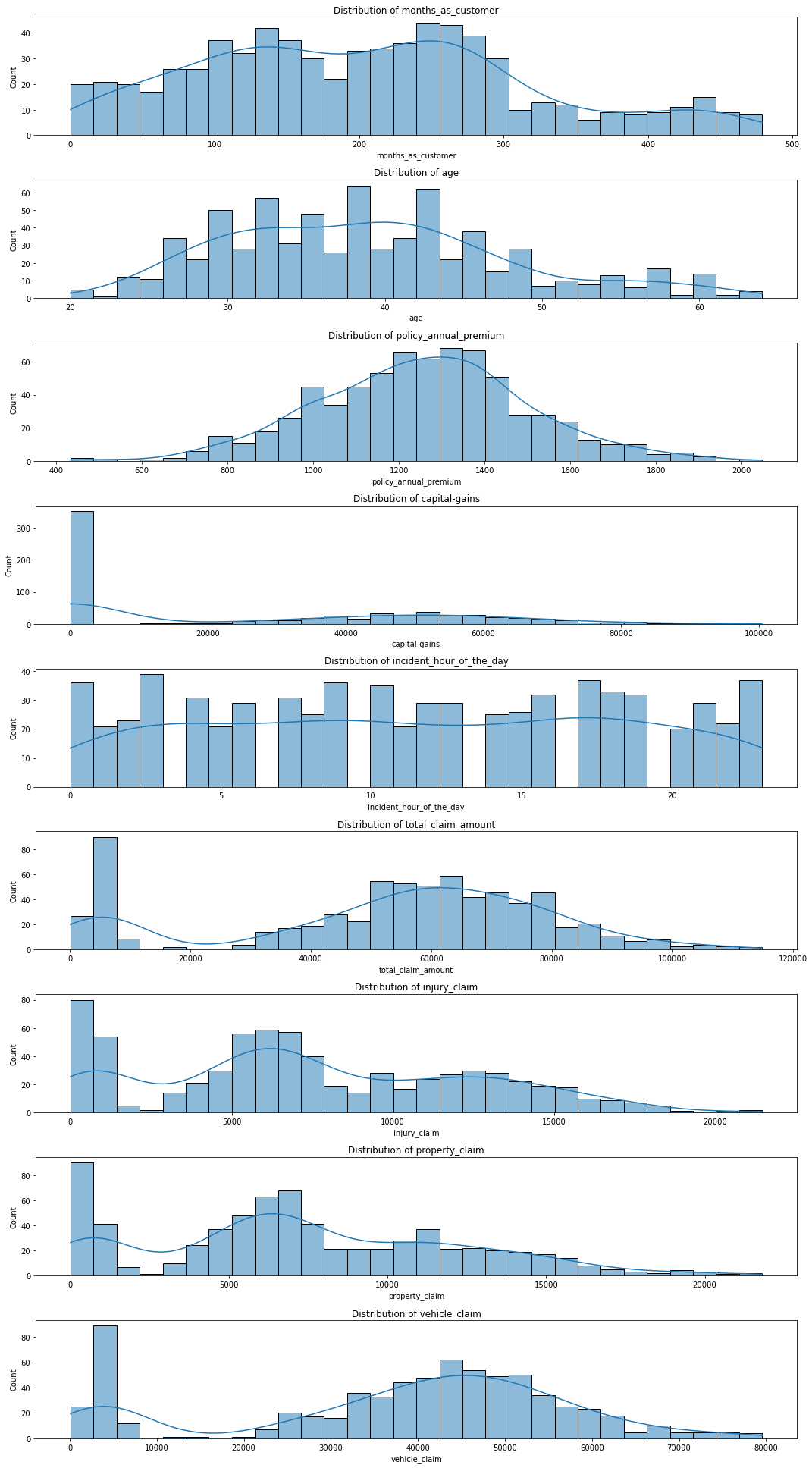
sns**.**histplot(X\_train[col], kde**=True**, bins**=**30)

plt**.**title(f"Distribution of {col}")

*# Adjust layout*

plt**.**tight\_layout()

plt**.**show()



**4.2 Perform correlation analysis [3 Marks]**

Investigate the relationships between numerical features to identify potential multicollinearity or dependencies. Visualise the correlation structure using an appropriate method to gain insights into feature relationships.

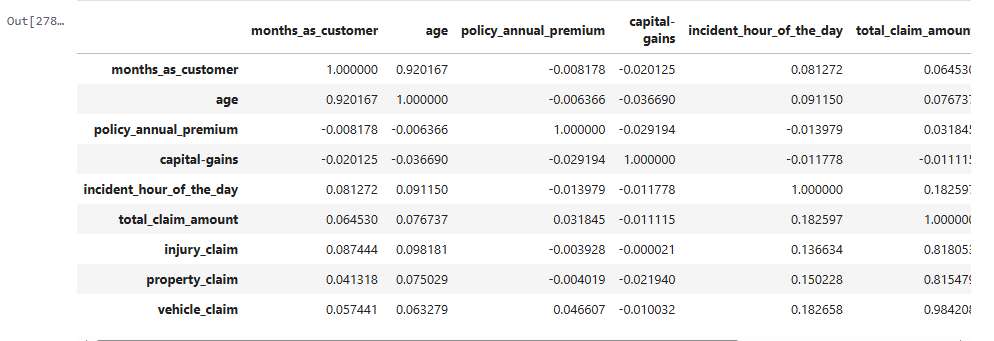
In [278]:

*# Create correlation matrix for numerical columns*

corr**=**X\_train**.**corr()

corr

Out[278]:

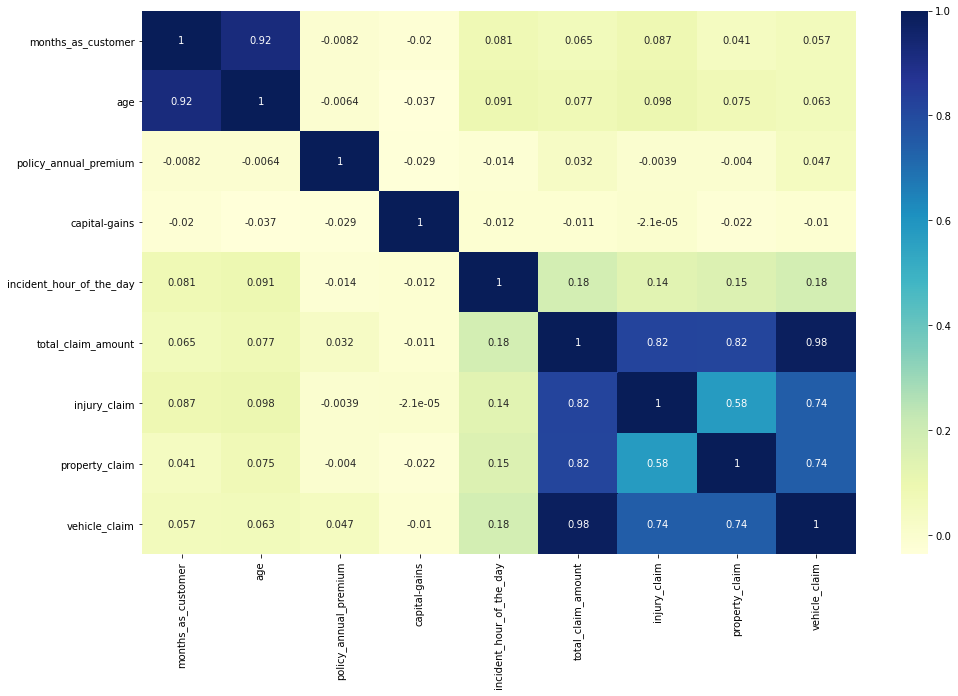


In [279]:

plt**.**figure(figsize **=** (16, 10))

sns**.**heatmap(corr, annot **=** **True**, cmap**=**"YlGnBu")

plt**.**show()



**🔍 Interpretation of Correlation Heatmap**

The above heatmap visualises the **correlation matrix of numerical features** in the dataset. Here's a breakdown of key insights:

* **Strong Positive Correlations**:
  + total\_claim\_amount shows high correlation with:
    - injury\_claim (0.82)
    - property\_claim (0.82)
    - vehicle\_claim (0.98)
  + This indicates that total\_claim\_amount is largely composed of these three component claims, as expected.
* **Multicollinearity Alert**:
  + months\_as\_customer and age are **very highly correlated** (0.92). This suggests that older customers tend to have been with the company longer. Consider dropping one of them to avoid multicollinearity in models.

**4.3 Check class balance [2 Marks]**

Examine the distribution of the target variable to identify potential class imbalances using visualisation for better understanding.

In [280]:

y\_train**.**value\_counts()

Out[280]:

N 526

Y 173

Name: fraud\_reported, dtype: int64

In [281]:

*# Plot a bar chart to check class balance*

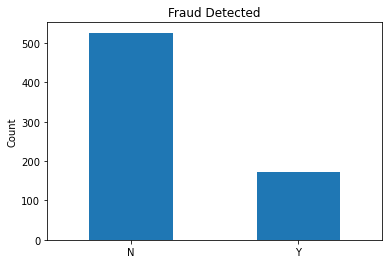
y\_train**.**value\_counts()**.**plot(kind**=**'bar')

plt**.**title('Fraud Detected')

plt**.**ylabel('Count')

plt**.**xticks(rotation**=**0)

plt**.**show()



**4.4 Perform bivariate analysis [10 Marks]**

**4.4.1 Target likelihood analysis for categorical variables. [5 Marks]**

Investigate the relationships between categorical features and the target variable by analysing the target event likelihood (for the 'Y' event) for each level of every relevant categorical feature. Through this analysis, identify categorical features that do not contribute much in explaining the variation in the target variable.

In [282]:

*# Write a function to calculate and analyse the target variable likelihood for categorical features*

In [283]:

*# Get all categorical columns*

cat\_columns **=** X\_train**.**select\_dtypes(include**=**['object', 'category'])**.**columns**.**tolist()

In [284]:

*# sns.countplot(x=X\_train['policy\_csl'], hue=y\_train)*

In [285]:

**def** target\_likelihood\_analysis(X, y, positive\_class):

*# Combine X and y for analysis*

df **=** X**.**copy()

df['target'] **=** y**.**values *# ensure alignment*

results **=** {}

**for** col **in** cat\_columns:

grouped **=** df**.**groupby(col)['target']**.**agg(

total**=**'count',

positive**=lambda** x: (x **==** positive\_class)**.**sum()

)

grouped['likelihood/fraud\_reported'] **=** grouped['positive'] **/** grouped['total']

results[col] **=** grouped**.**sort\_values('likelihood/fraud\_reported', ascending**=False**)

**return** results

**def** plot\_target\_likelihoods(likelihoods\_dict):

**for** col, data **in** likelihoods\_dict**.**items():

plt**.**figure(figsize**=**(8, 4))

sns**.**barplot(x**=**data**.**index, y**=**data['likelihood/fraud\_reported'])

plt**.**title(f'Target Likelihood per Category for "{col}"')

plt**.**ylabel('Likelihood of Positive Class')

plt**.**xlabel(col)

plt**.**xticks(rotation**=**45)

plt**.**tight\_layout()

plt**.**show()

In [286]:

*# First calculate likelihoods*

likelihoods **=** target\_likelihood\_analysis(X\_train, y\_train, positive\_class**=**'Y')

**for** col,data **in** likelihoods**.**items():

print(col)

print(data)

policy\_state

total positive likelihood/fraud\_reported

policy\_state

OH 241 61 0.253112

IN 220 54 0.245455

IL 238 58 0.243697

policy\_csl

total positive likelihood/fraud\_reported

policy\_csl

100/300 250 67 0.268000

250/500 235 59 0.251064

500/1000 214 47 0.219626

policy\_deductable

total positive likelihood/fraud\_reported

policy\_deductable

2000 219 59 0.269406

500 228 60 0.263158

1000 252 54 0.214286

umbrella\_limit

total positive likelihood/fraud\_reported

umbrella\_limit

2000000 3 2 0.666667

9000000 4 2 0.500000

8000000 5 2 0.400000

5000000 34 12 0.352941

4000000 26 9 0.346154

6000000 43 11 0.255814

7000000 16 4 0.250000

0 559 129 0.230769

3000000 9 2 0.222222

insured\_sex

total positive likelihood/fraud\_reported

insured\_sex

FEMALE 373 93 0.249330

MALE 326 80 0.245399

insured\_education\_level

total positive likelihood/fraud\_reported

insured\_education\_level

JD 113 32 0.283186

MD 95 25 0.263158

Masters 99 25 0.252525

College 81 20 0.246914

PhD 90 22 0.244444

Associate 111 26 0.234234

High School 110 23 0.209091

insured\_occupation

total positive likelihood/fraud\_reported

insured\_occupation

transport-moving 58 20 0.344828

exec-managerial 56 19 0.339286

craft-repair 53 16 0.301887

farming-fishing 32 9 0.281250

sales 55 15 0.272727

armed-forces 46 12 0.260870

tech-support 54 14 0.259259

machine-op-inspct 65 16 0.246154

protective-serv 37 9 0.243243

prof-specialty 63 12 0.190476

other-service 48 9 0.187500

priv-house-serv 50 9 0.180000

adm-clerical 47 8 0.170213

handlers-cleaners 35 5 0.142857

insured\_hobbies

total positive likelihood/fraud\_reported

insured\_hobbies

chess 32 29 0.906250

cross-fit 25 19 0.760000

polo 32 10 0.312500

yachting 37 11 0.297297

paintball 47 12 0.255319

hiking 36 9 0.250000

base-jumping 38 9 0.236842

video-games 34 8 0.235294

reading 49 11 0.224490

skydiving 36 8 0.222222

board-games 32 7 0.218750

sleeping 24 5 0.208333

bungie-jumping 40 7 0.175000

exercise 39 6 0.153846

movies 33 5 0.151515

dancing 34 4 0.117647

basketball 18 2 0.111111

kayaking 36 4 0.111111

camping 37 4 0.108108

golf 40 3 0.075000

insured\_relationship

total positive likelihood/fraud\_reported

insured\_relationship

other-relative 127 38 0.299213

unmarried 102 30 0.294118

wife 114 31 0.271930

not-in-family 121 29 0.239669

husband 114 24 0.210526

own-child 121 21 0.173554

incident\_type

total positive likelihood/fraud\_reported

incident\_type

Single Vehicle Collision 282 81 0.287234

Multi-vehicle Collision 291 81 0.278351

Parked Car 65 6 0.092308

Vehicle Theft 61 5 0.081967

collision\_type

total positive likelihood/fraud\_reported

collision\_type

Front Collision 173 50 0.289017

Side Collision 188 49 0.260638

Rear Collision 338 74 0.218935

incident\_severity

total positive likelihood/fraud\_reported

incident\_severity

Major Damage 192 117 0.609375

Total Loss 193 29 0.150259

Minor Damage 253 23 0.090909

Trivial Damage 61 4 0.065574

authorities\_contacted

total positive likelihood/fraud\_reported

authorities\_contacted

Ambulance 136 43 0.316176

Other 140 40 0.285714

Fire 157 44 0.280255

Police 198 40 0.202020

None 68 6 0.088235

incident\_state

total positive likelihood/fraud\_reported

incident\_state

OH 19 8 0.421053

PA 19 6 0.315789

NC 83 26 0.313253

SC 177 53 0.299435

VA 75 20 0.266667

NY 174 37 0.212644

WV 152 23 0.151316

incident\_city

total positive likelihood/fraud\_reported

incident\_city

Arlington 104 31 0.298077

Springfield 105 29 0.276190

Columbus 110 30 0.272727

Hillsdale 92 23 0.250000

Northbrook 89 19 0.213483

Riverwood 99 21 0.212121

Northbend 100 20 0.200000

number\_of\_vehicles\_involved

total positive likelihood/fraud\_reported

number\_of\_vehicles\_involved

2 19 6 0.315789

4 17 5 0.294118

3 255 70 0.274510

1 408 92 0.225490

property\_damage

total positive likelihood/fraud\_reported

property\_damage

YES 207 58 0.280193

NO 492 115 0.233740

bodily\_injuries

total positive likelihood/fraud\_reported

bodily\_injuries

2 225 63 0.280000

0 249 59 0.236948

1 225 51 0.226667

witnesses

total positive likelihood/fraud\_reported

witnesses

2 171 48 0.280702

3 164 43 0.262195

1 188 47 0.250000

0 176 35 0.198864

police\_report\_available

total positive likelihood/fraud\_reported

police\_report\_available

NO 490 125 0.255102

YES 209 48 0.229665

auto\_make

total positive likelihood/fraud\_reported

auto\_make

Audi 52 17 0.326923

Ford 54 16 0.296296

Mercedes 51 15 0.294118

Honda 31 9 0.290323

BMW 50 14 0.280000

Volkswagen 40 11 0.275000

Chevrolet 55 15 0.272727

Jeep 44 11 0.250000

Suburu 63 15 0.238095

Saab 51 12 0.235294

Toyota 47 11 0.234043

Dodge 53 10 0.188679

Nissan 58 10 0.172414

Accura 50 7 0.140000

auto\_model

total positive likelihood/fraud\_reported

auto\_model

X6 10 5 0.500000

Civic 12 6 0.500000

Silverado 16 7 0.437500

Tahoe 20 8 0.400000

Maxima 16 6 0.375000

Grand Cherokee 16 6 0.375000

A5 24 9 0.375000

F150 22 8 0.363636

C300 12 4 0.333333

Forrestor 24 8 0.333333

93 13 4 0.307692

M5 10 3 0.300000

X5 17 5 0.294118

E400 24 7 0.291667

A3 28 8 0.285714

Jetta 21 6 0.285714

Highlander 18 5 0.277778

ML350 15 4 0.266667

Fusion 15 4 0.266667

92x 19 5 0.263158

Passat 19 5 0.263158

Escape 17 4 0.235294

Camry 17 4 0.235294

Impreza 14 3 0.214286

Accord 10 2 0.200000

Neon 26 5 0.192308

RAM 27 5 0.185185

Wrangler 28 5 0.178571

MDX 29 5 0.172414

Corolla 12 2 0.166667

Legacy 25 4 0.160000

95 19 3 0.157895

Ultima 20 3 0.150000

CRV 9 1 0.111111

TL 10 1 0.100000

RSX 11 1 0.090909

3 Series 13 1 0.076923

Pathfinder 22 1 0.045455

Malibu 19 0 0.000000

auto\_year

total positive likelihood/fraud\_reported

auto\_year

2004 32 14 0.437500

1996 27 11 0.407407

2006 37 12 0.324324

2001 28 9 0.321429

2014 26 8 0.307692

2007 40 12 0.300000

2000 24 7 0.291667

1995 39 11 0.282051

2005 36 9 0.250000

2003 38 9 0.236842

2011 34 8 0.235294

2002 41 9 0.219512

2013 29 6 0.206897

2008 35 7 0.200000

1997 31 6 0.193548

2015 37 7 0.189189

2010 32 6 0.187500

1999 40 7 0.175000

1998 30 5 0.166667

2012 31 5 0.161290

2009 32 5 0.156250

In [287]:

X\_train['umbrella\_limit']**.**describe()

Out[287]:

count 699

unique 9

top 0

freq 559

Name: umbrella\_limit, dtype: int64

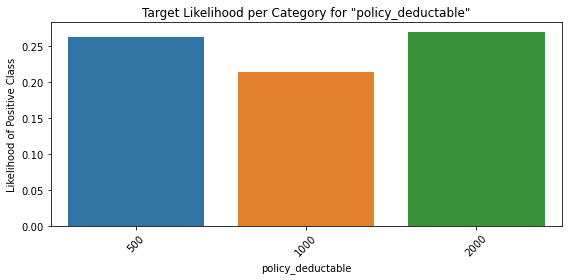
In [288]:

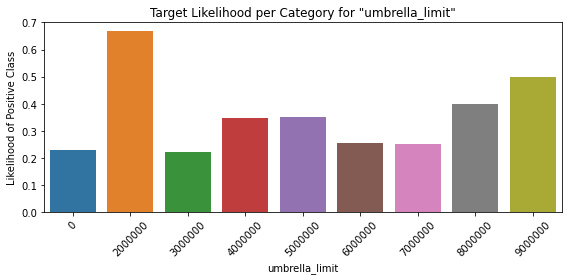
*# Then plot them*

plot\_target\_likelihoods(likelihoods)

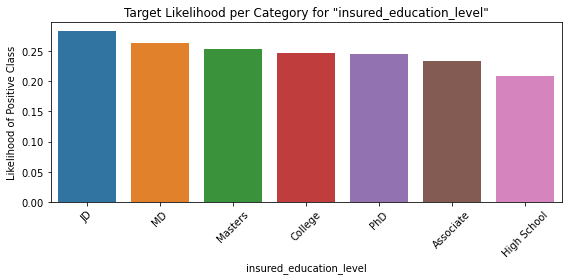


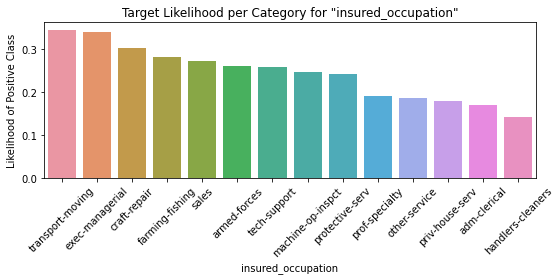


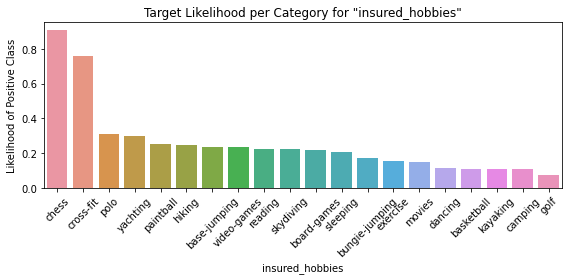


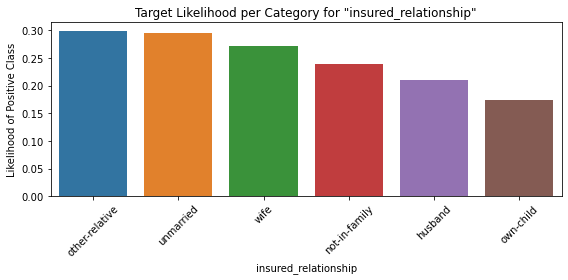






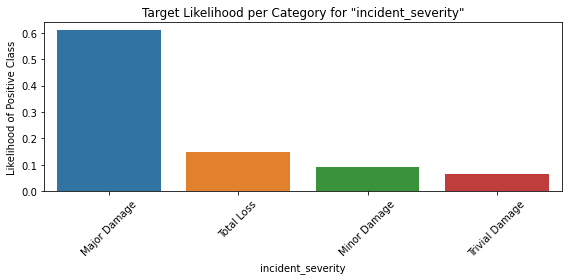




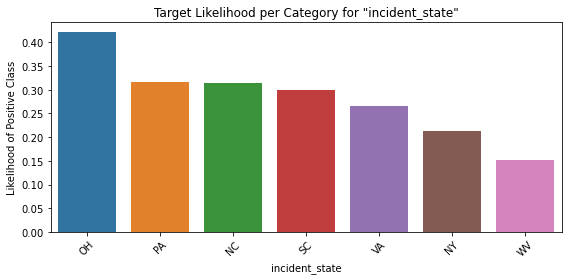


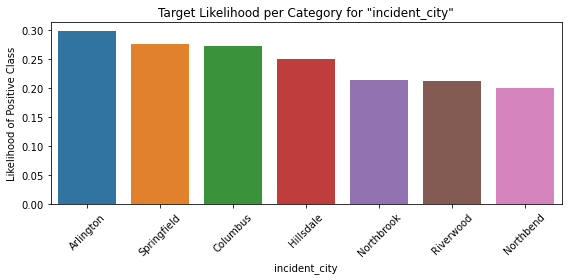


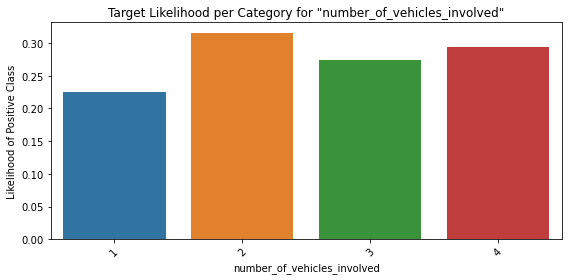




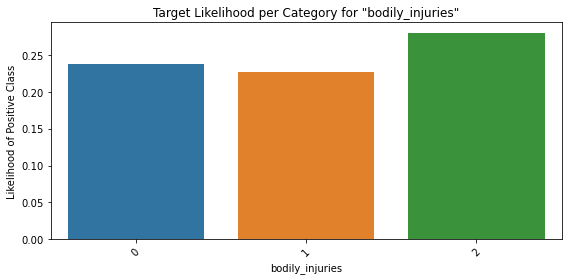


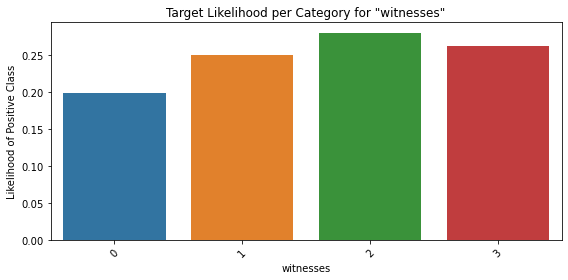




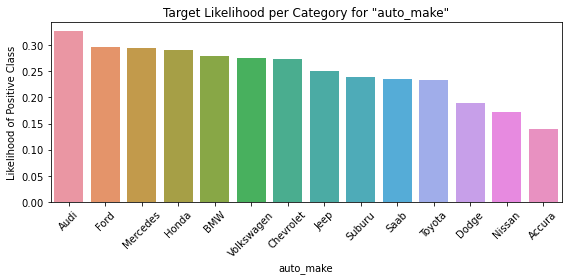


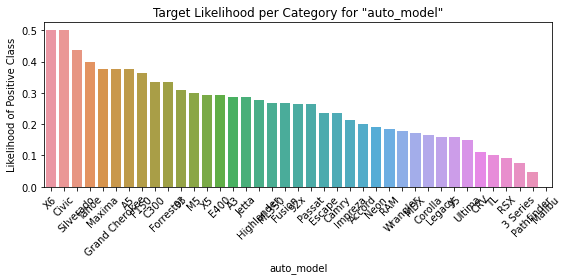


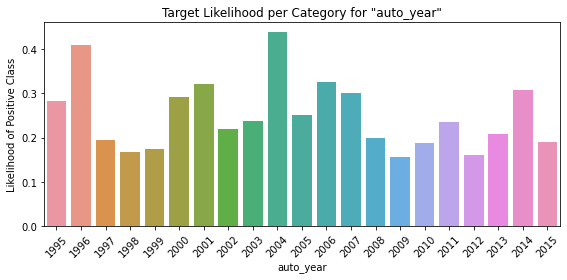












**Columns-**

**'policy\_state','insured\_sex', shows very very little variation and contribute less in explaining the target variable**

**'policy\_csl','police\_report\_available','insured\_education\_level' also shows little variation and comparitively contribute less in explaining the target variable**

**4.4.2 Explore the relationships between numerical features and the target variable to understand their impact on the target outcome using appropriate visualisation techniques to identify trends and potential interactions. [5 Marks]**

In [289]:

*# Visualise the relationship between numerical features and the target variable to understand their impact on the target outcome*

**for** col **in** numeric\_columns:

sns**.**histplot(data**=**X\_train, x**=**col, hue**=**y\_train, kde**=True**, bins**=**30)

plt**.**show()

sns**.**boxplot(x**=**y\_train, y**=**X\_train[col])

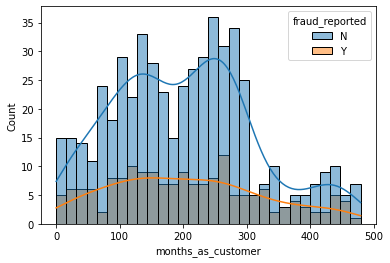
plt**.**title(f'{col} vs Fraud Detection')

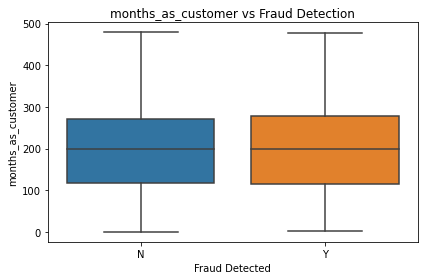
plt**.**xlabel('Fraud Detected')

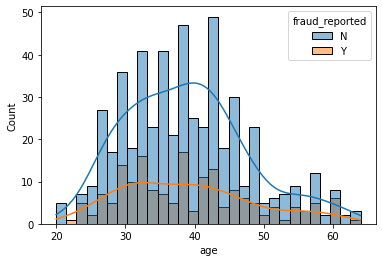
plt**.**ylabel(col)

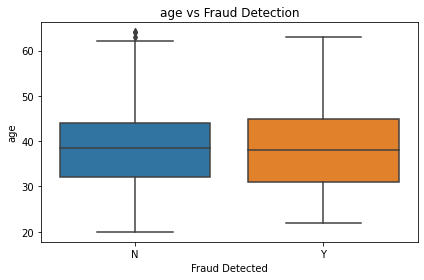
plt**.**tight\_layout()

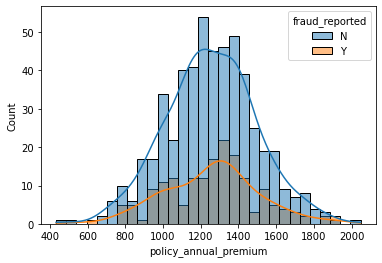
plt**.**show()

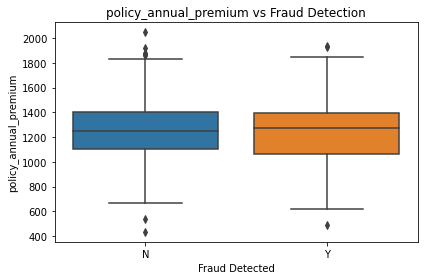


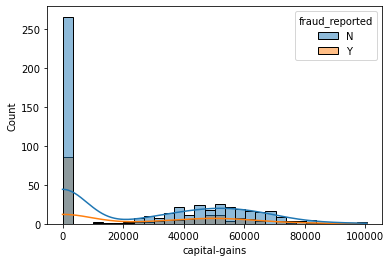


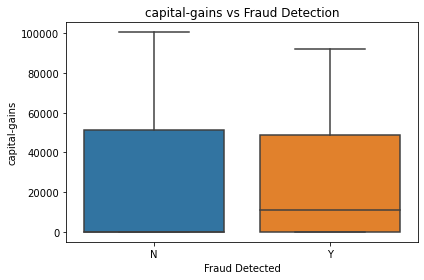


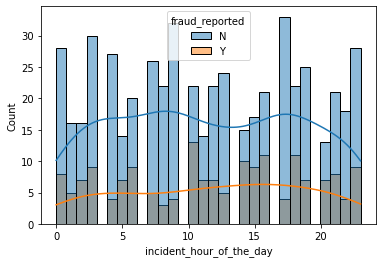


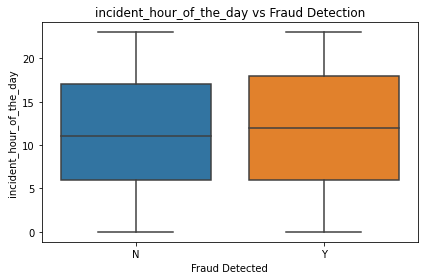


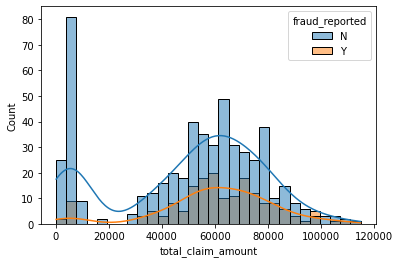


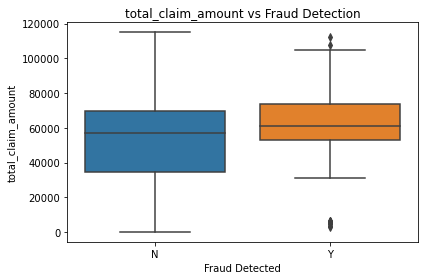


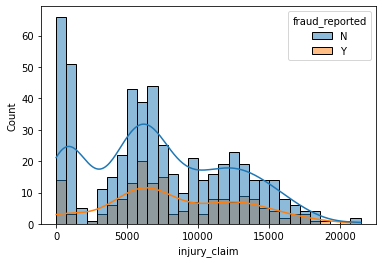


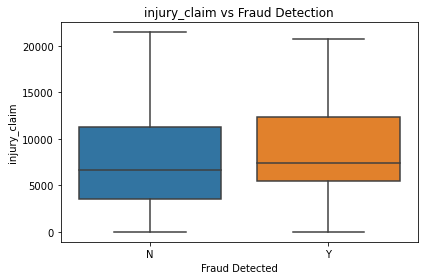


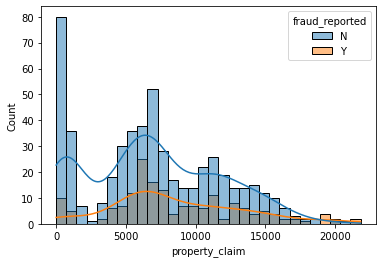


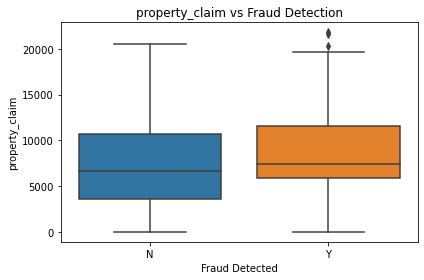


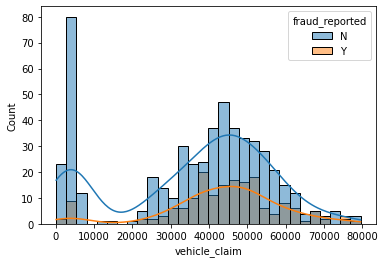


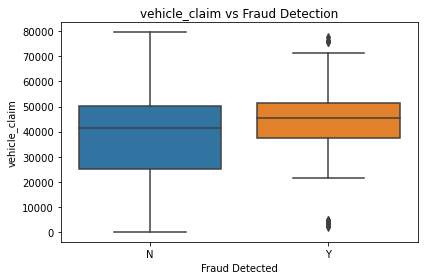












In [290]:

*#*

In [ ]:

**5. EDA on validation data [OPTIONAL]**

**5.1 Perform univariate analysis**

**5.1.1 Identify and select numerical columns from training data for univariate analysis.**

In [291]:

*# Select numerical columns*

**5.1.2 Visualise the distribution of selected numerical features using appropriate plots to understand their characteristics.**

In [292]:

*# Plot all the numerical columns to understand their distribution*

**5.2 Perform correlation analysis**

Investigate the relationships between numerical features to identify potential multicollinearity or dependencies. Visualise the correlation structure using an appropriate method to gain insights into feature relationships.

In [293]:

*# Create correlation matrix for numerical columns*

*# Plot Heatmap of the correlation matrix*

**5.3 Check class balance**

Examine the distribution of the target variable to identify potential class imbalances. Visualise the distribution for better understanding.

In [294]:

*# Plot a bar chart to check class balance*

**5.4 Perform bivariate analysis**

**5.4.1 Target likelihood analysis for categorical variables.**

Investigate the relationships between categorical features and the target variable by analysing the target event likelihood (for the 'Y' event) for each level of every relevant categorical feature. Through this analysis, identify categorical features that do not contribute much in explaining the variation in the target variable.

In [295]:

*# Write a function to calculate and analyse the target variable likelihood for categorical features*

**5.4.2 Explore the relationships between numerical features and the target variable to understand their impact on the target outcome. Utilise appropriate visualisation techniques to identify trends and potential interactions.**

In [296]:

*# Visualise the relationship between numerical features and the target variable to understand their impact on the target outcome*

**6. Feature Engineering [25 marks]**

**6.1 Perform resampling [3 Marks]**

Handle class imbalance in the training data by applying resampling technique.

Use the **RandomOverSampler** technique to balance the data and handle class imbalance. This method increases the number of samples in the minority class by randomly duplicating them, creating synthetic data points with similar characteristics. This helps prevent the model from being biased toward the majority class and improves its ability to predict the minority class more accurately.

**Note:** You can try other resampling techniques to handle class imbalance

In [297]:

y\_train**.**value\_counts()

Out[297]:

N 526

Y 173

Name: fraud\_reported, dtype: int64

In [298]:

*# Import RandomOverSampler from imblearn library*

**from** imblearn.over\_sampling **import** RandomOverSampler

*# Perform resampling on training data*

ros **=** RandomOverSampler(random\_state**=**42)

*# Perform oversampling*

X\_resampled, y\_resampled **=** ros**.**fit\_resample(X\_train, y\_train)

*# Check the new class distribution*

print(y\_resampled**.**value\_counts())

Y 526

N 526

Name: fraud\_reported, dtype: int64

In [299]:

X\_resampled**.**shape

Out[299]:

(1052, 34)

**6.2 Feature Creation [4 marks]**

Create new features from existing ones to enhance the model's ability to capture patterns in the data. This may involve deriving features from date/time columns, combining features, or creating interaction terms.

In [300]:

*# Create new features based on your understanding for both training and validation data*

X\_resampled**.**columns

Out[300]:

Index(['months\_as\_customer', 'age', 'policy\_bind\_date', 'policy\_state',

'policy\_csl', 'policy\_deductable', 'policy\_annual\_premium',

'umbrella\_limit', 'insured\_sex', 'insured\_education\_level',

'insured\_occupation', 'insured\_hobbies', 'insured\_relationship',

'capital-gains', 'incident\_date', 'incident\_type', 'collision\_type',

'incident\_severity', 'authorities\_contacted', 'incident\_state',

'incident\_city', 'incident\_hour\_of\_the\_day',

'number\_of\_vehicles\_involved', 'property\_damage', 'bodily\_injuries',

'witnesses', 'police\_report\_available', 'total\_claim\_amount',

'injury\_claim', 'property\_claim', 'vehicle\_claim', 'auto\_make',

'auto\_model', 'auto\_year'],

dtype='object')

In [301]:

*# X\_resampled['months\_as\_customer'].describe()*

*# X\_resampled['umbrella\_limit'].describe()*

In [302]:

X\_resampled['incident\_days\_since\_policy']**=**(X\_resampled['policy\_bind\_date']**-**X\_resampled['incident\_date'])**.**dt**.**days**.**abs()

X\_resampled['incident\_days\_since\_policy']**.**describe()

Out[302]:

count 1052.000000

mean 4654.530418

std 2679.381095

min 6.000000

25% 2434.000000

50% 4556.000000

75% 7016.000000

max 9172.000000

Name: incident\_days\_since\_policy, dtype: float64

In [303]:

X\_val['incident\_days\_since\_policy']**=**(X\_val['policy\_bind\_date']**-**X\_resampled['incident\_date'])**.**dt**.**days**.**abs()

X\_val['incident\_days\_since\_policy']**.**describe()

Out[303]:

count 300.000000

mean 4770.093333

std 2668.381291

min 72.000000

25% 2564.250000

50% 4666.500000

75% 7020.750000

max 9153.000000

Name: incident\_days\_since\_policy, dtype: float64

In [304]:

X\_resampled['incident\_hour\_of\_the\_day']**.**describe()

Out[304]:

count 1052.000000

mean 11.849810

std 6.979259

min 0.000000

25% 6.000000

50% 12.000000

75% 18.000000

max 23.000000

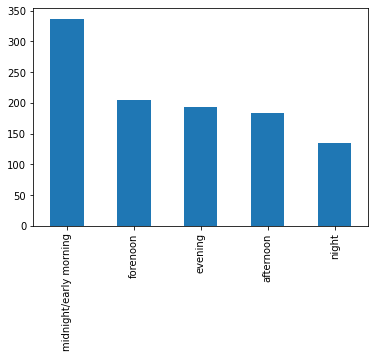
Name: incident\_hour\_of\_the\_day, dtype: float64

In [305]:

X\_resampled['incident\_event\_time']**=**pd**.**cut(X\_resampled['incident\_hour\_of\_the\_day'], bins**=**[0, 7,12, 16,20, 24], labels**=**['midnight/early morning', 'forenoon','afternoon','evening' ,'night'], right**=True**, include\_lowest**=True**)

X\_resampled['incident\_event\_time']**.**value\_counts()**.**plot**.**bar()

plt**.**show()

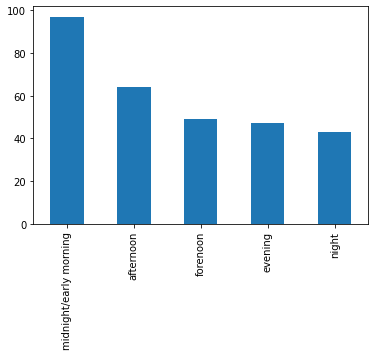


In [306]:

X\_val['incident\_event\_time']**=**pd**.**cut(X\_val['incident\_hour\_of\_the\_day'],bins**=**[0, 7,12, 16,20, 24], labels**=**['midnight/early morning', 'forenoon','afternoon','evening' ,'night'], right**=True**, include\_lowest**=True**)

X\_val['incident\_event\_time']**.**value\_counts()**.**plot**.**bar()

plt**.**show()

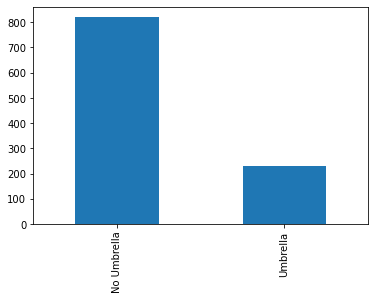


In [307]:

X\_resampled['has\_Umbrella']**=**X\_resampled['umbrella\_limit']**.**apply(**lambda** x: 'Umbrella' **if** x**!=**0 **else** "No Umbrella")**.**astype('str')

X\_resampled['has\_Umbrella']**.**value\_counts()**.**plot**.**bar()

plt**.**show()

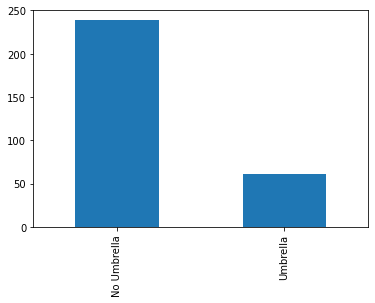


In [308]:

X\_val['has\_Umbrella']**=**X\_val['umbrella\_limit']**.**apply(**lambda** x: 'Umbrella' **if** x**!=**0 **else** "No Umbrella")

X\_val['has\_Umbrella']**.**value\_counts()**.**plot**.**bar()

plt**.**show()

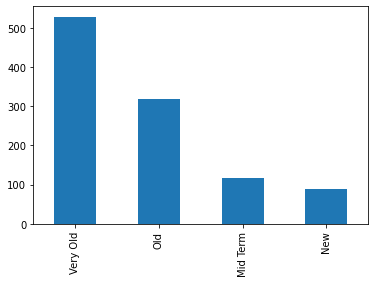


In [309]:

X\_resampled['customer\_type'] **=** pd**.**cut(X\_resampled['months\_as\_customer'], bins**=**[0, 50, 100, 200, 500], labels**=**['New', 'Mid Term', 'Old', 'Very Old'], right**=True**, include\_lowest**=True**)

X\_resampled['customer\_type']**.**value\_counts()**.**plot**.**bar()

plt**.**show()

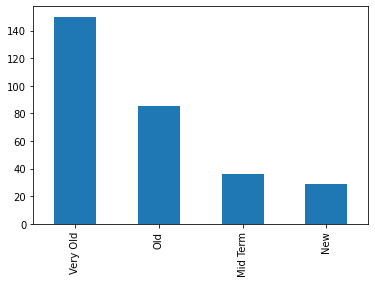


In [310]:

X\_val['customer\_type'] **=** pd**.**cut(X\_val['months\_as\_customer'], bins**=**[0, 50, 100, 200, 500], labels**=**['New', 'Mid Term', 'Old', 'Very Old'], right**=True**, include\_lowest**=True**)**.**astype(str)

X\_val['customer\_type']**.**value\_counts()**.**plot**.**bar()

plt**.**show()



**6.3 Handle redundant columns [3 marks]**

Analyse the data to identify features that may be redundant or contribute minimal information toward predicting the target variable and drop them.

* You can consider features that exhibit high correlation with other variables, which you may have observed during the EDA phase.
* Features that don't strongly influence the prediction, which you may have observed during the EDA phase.
* Categorical columns with low value counts for some levels can be remapped to reduce number of unique levels, and features with very high counts for just one level may be removed, as they resemble unique identifier columns and do not provide substantial predictive value.
* Additionally, eliminate any columns from which the necessary features have already been extracted in the preceding step.

In [311]:

**from** statsmodels.stats.outliers\_influence **import** variance\_inflation\_factor

vif **=** pd**.**DataFrame()

vif['Features'] **=** numeric\_columns

vif['VIF'] **=** [variance\_inflation\_factor(X\_resampled[numeric\_columns]**.**values, i) **for** i **in** range(X\_resampled[numeric\_columns]**.**shape[1])]

vif['VIF'] **=** round(vif['VIF'], 2)

vif **=** vif**.**sort\_values(by**=**'VIF',ascending **=** **False**)

vif

Out[311]:

|  | **Features** | **VIF** |
| --- | --- | --- |
| **5** | total\_claim\_amount | inf |
| **6** | injury\_claim | inf |
| **7** | property\_claim | inf |
| **8** | vehicle\_claim | inf |
| **1** | age | 66.27 |
| **2** | policy\_annual\_premium | 20.42 |
| **0** | months\_as\_customer | 18.97 |
| **4** | incident\_hour\_of\_the\_day | 4.04 |
| **3** | capital-gains | 1.87 |

In [312]:

X\_resampled**.**columns

Out[312]:

Index(['months\_as\_customer', 'age', 'policy\_bind\_date', 'policy\_state',

'policy\_csl', 'policy\_deductable', 'policy\_annual\_premium',

'umbrella\_limit', 'insured\_sex', 'insured\_education\_level',

'insured\_occupation', 'insured\_hobbies', 'insured\_relationship',

'capital-gains', 'incident\_date', 'incident\_type', 'collision\_type',

'incident\_severity', 'authorities\_contacted', 'incident\_state',

'incident\_city', 'incident\_hour\_of\_the\_day',

'number\_of\_vehicles\_involved', 'property\_damage', 'bodily\_injuries',

'witnesses', 'police\_report\_available', 'total\_claim\_amount',

'injury\_claim', 'property\_claim', 'vehicle\_claim', 'auto\_make',

'auto\_model', 'auto\_year', 'incident\_days\_since\_policy',

'incident\_event\_time', 'has\_Umbrella', 'customer\_type'],

dtype='object')

**Dropping some variables**

**'policy\_bind\_date','incident\_date','months\_as\_customer','umbrella\_limit', 'incident\_hour\_of\_the\_day' as they have been used to derive a new variable,**

**'age' as it is highly corelated with 'months\_as\_customer'**

**'injury\_claim', 'property\_claim', 'vehicle\_claim', as they are highly corelated with total\_claim\_amount**

**'policy\_state', 'insured\_sex','policy\_csl','police\_report\_available','insured\_education\_level' as they contribute less in prediction**

**'auto\_model','auto\_year' ,'insured\_hobbies','insured\_occupation' are also irrelevant**

In [313]:

cols\_to\_drop**=**['policy\_bind\_date','incident\_date','months\_as\_customer','incident\_hour\_of\_the\_day','umbrella\_limit','age','injury\_claim', 'property\_claim', 'vehicle\_claim','policy\_state', 'insured\_sex','policy\_csl','police\_report\_available','insured\_education\_level','auto\_model','auto\_year','insured\_hobbies','insured\_occupation']

In [314]:

X\_resampled**.**drop(cols\_to\_drop,axis**=**1,inplace**=True**)

X\_val**.**drop(cols\_to\_drop,axis**=**1,inplace**=True**)

In [315]:

*# Check the data*

X\_resampled**.**shape

Out[315]:

(1052, 20)

In [316]:

X\_val**.**shape

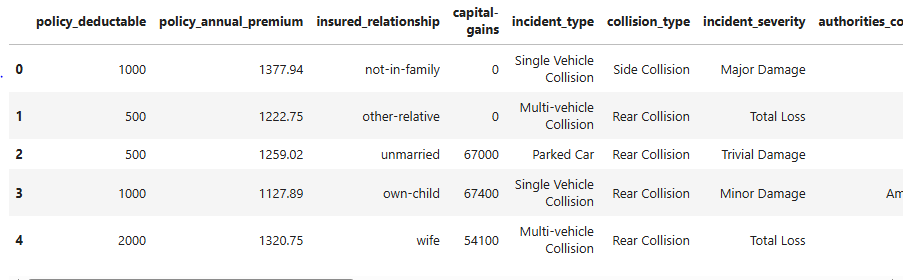
Out[316]:

(300, 20)

In [317]:

X\_resampled**.**head()

Out[317]:



In [318]:

numeric\_df **=** X\_resampled**.**\_get\_numeric\_data()

numeric\_columns**=**numeric\_df**.**columns

cat\_columns **=** X\_resampled**.**select\_dtypes(include**=**['object', 'category'])**.**columns**.**tolist()

In [319]:

numeric\_columns

Out[319]:

Index(['policy\_annual\_premium', 'capital-gains', 'total\_claim\_amount',

'incident\_days\_since\_policy'],

dtype='object')

In [320]:

cat\_columns

Out[320]:

['policy\_deductable',

'insured\_relationship',

'incident\_type',

'collision\_type',

'incident\_severity',

'authorities\_contacted',

'incident\_state',

'incident\_city',

'number\_of\_vehicles\_involved',

'property\_damage',

'bodily\_injuries',

'witnesses',

'auto\_make',

'incident\_event\_time',

'has\_Umbrella',

'customer\_type']

**6.4 Combine values in Categorical Columns [6 Marks]**

During the EDA process, categorical columns with multiple unique values may be identified. To enhance model performance, it is essential to refine these categorical features by grouping values that have low frequency or provide limited predictive information.

Combine categories that occur infrequently or exhibit similar behavior to reduce sparsity and improve model generalisation.

In [321]:

*# Combine categories that have low frequency or provide limited predictive information*

**for** cols **in** cat\_columns:

print(f"Categorical Feature Name:",cols)

print(100**\***X\_resampled[cols]**.**value\_counts()**/**len(X\_resampled[cols]))

print("\*"**\*** 50)

Categorical Feature Name: policy\_deductable

1000 34.790875

2000 33.174905

500 32.034221

Name: policy\_deductable, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: insured\_relationship

other-relative 18.250951

wife 17.490494

not-in-family 17.015209

unmarried 16.254753

husband 15.874525

own-child 15.114068

Name: insured\_relationship, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_type

Multi-vehicle Collision 44.866920

Single Vehicle Collision 41.159696

Parked Car 7.699620

Vehicle Theft 6.273764

Name: incident\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: collision\_type

Rear Collision 45.817490

Side Collision 27.281369

Front Collision 26.901141

Name: collision\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_severity

Major Damage 41.349810

Minor Damage 28.612167

Total Loss 23.859316

Trivial Damage 6.178707

Name: incident\_severity, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: authorities\_contacted

Police 27.281369

Fire 22.718631

Ambulance 21.577947

Other 20.627376

None 7.794677

Name: authorities\_contacted, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_state

SC 26.901141

NY 24.429658

WV 18.441065

NC 12.832700

VA 10.361217

OH 4.277567

PA 2.756654

Name: incident\_state, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_city

Arlington 16.920152

Columbus 15.684411

Springfield 14.733840

Hillsdale 14.068441

Northbend 14.068441

Riverwood 13.593156

Northbrook 10.931559

Name: incident\_city, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: number\_of\_vehicles\_involved

1 55.133080

3 38.403042

2 3.992395

4 2.471483

Name: number\_of\_vehicles\_involved, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: property\_damage

NO 68.726236

YES 31.273764

Name: property\_damage, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: bodily\_injuries

2 35.076046

0 34.125475

1 30.798479

Name: bodily\_injuries, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: witnesses

2 25.855513

1 25.095057

0 24.809886

3 24.239544

Name: witnesses, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: auto\_make

Ford 9.220532

Audi 8.840304

Chevrolet 8.745247

Saab 8.174905

Dodge 8.079848

Nissan 7.889734

Suburu 7.509506

BMW 7.319392

Mercedes 6.653992

Accura 6.273764

Toyota 5.798479

Jeep 5.703422

Volkswagen 5.418251

Honda 4.372624

Name: auto\_make, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_event\_time

midnight/early morning 32.034221

forenoon 19.391635

evening 18.441065

afternoon 17.395437

night 12.737643

Name: incident\_event\_time, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: has\_Umbrella

No Umbrella 78.041825

Umbrella 21.958175

Name: has\_Umbrella, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: customer\_type

Very Old 50.190114

Old 30.133080

Mid Term 11.121673

New 8.555133

Name: customer\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In [322]:

*# Combine categories that have low frequency or provide limited predictive information*

**for** cols **in** cat\_columns:

print(f"Categorical Feature Name:",cols)

print(100**\***X\_val[cols]**.**value\_counts()**/**len(X\_val[cols]))

print("\*"**\*** 50)

Categorical Feature Name: policy\_deductable

500 37.666667

1000 33.000000

2000 29.333333

Name: policy\_deductable, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: insured\_relationship

own-child 20.666667

husband 18.666667

not-in-family 17.666667

other-relative 16.666667

wife 13.333333

unmarried 13.000000

Name: insured\_relationship, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_type

Multi-vehicle Collision 42.666667

Single Vehicle Collision 40.000000

Vehicle Theft 11.000000

Parked Car 6.333333

Name: incident\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: collision\_type

Rear Collision 44.0

Side Collision 29.0

Front Collision 27.0

Name: collision\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_severity

Minor Damage 33.666667

Total Loss 29.000000

Major Damage 27.666667

Trivial Damage 9.666667

Name: incident\_severity, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: authorities\_contacted

Police 31.333333

Fire 22.000000

Ambulance 19.666667

Other 19.333333

None 7.666667

Name: authorities\_contacted, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_state

NY 29.333333

SC 23.666667

WV 21.666667

VA 11.666667

NC 8.666667

PA 3.666667

OH 1.333333

Name: incident\_state, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_city

Springfield 17.333333

Hillsdale 16.333333

Arlington 15.666667

Northbend 15.000000

Columbus 13.000000

Riverwood 11.666667

Northbrook 11.000000

Name: incident\_city, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: number\_of\_vehicles\_involved

1 57.333333

3 34.333333

4 4.666667

2 3.666667

Name: number\_of\_vehicles\_involved, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: property\_damage

NO 68.666667

YES 31.333333

Name: property\_damage, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: bodily\_injuries

2 35.666667

1 34.333333

0 30.000000

Name: bodily\_injuries, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: witnesses

3 26.333333

2 26.333333

0 24.333333

1 23.000000

Name: witnesses, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: auto\_make

Saab 9.666667

Volkswagen 9.333333

Dodge 9.000000

Honda 8.000000

Toyota 7.666667

Jeep 7.666667

BMW 7.333333

Chevrolet 6.666667

Nissan 6.666667

Accura 6.000000

Ford 6.000000

Suburu 5.666667

Audi 5.666667

Mercedes 4.666667

Name: auto\_make, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_event\_time

midnight/early morning 32.333333

afternoon 21.333333

forenoon 16.333333

evening 15.666667

night 14.333333

Name: incident\_event\_time, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: has\_Umbrella

No Umbrella 79.666667

Umbrella 20.333333

Name: has\_Umbrella, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: customer\_type

Very Old 50.000000

Old 28.333333

Mid Term 12.000000

New 9.666667

Name: customer\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In [323]:

*# Function to create less frequent categories ( threshold = <4%)*

*#By looking at both train and validation data, we found common these 2categories which can be combined*

cols\_combined**=**['incident\_state','number\_of\_vehicles\_involved']

**def** combine\_rare\_categories(df, categorical\_columns, threshold):

*#Combines rare categories in categorical columns into 'Others' if their percentage frequency is below the threshold.*

total\_rows **=** len(df)

**for** col **in** categorical\_columns:

freq\_percent **=** df[col]**.**value\_counts(normalize**=True**) **\*** 100

rare\_categories **=** freq\_percent[freq\_percent **<** threshold]**.**index

df[col] **=** df[col]**.**apply(**lambda** x: 'Others' **if** str(x) **in** rare\_categories **else** str(x))

**return** df

X\_resampled **=** combine\_rare\_categories(X\_resampled, cols\_combined, 5)

X\_val **=** combine\_rare\_categories(X\_val, cols\_combined, 5)

In [324]:

**for** cols **in** cat\_columns:

print(f"Categorical Feature Name:",cols)

print(X\_resampled[cols]**.**value\_counts(normalize**=True**) **\*** 100)

print("\*"**\*** 50)

Categorical Feature Name: policy\_deductable

1000 34.790875

2000 33.174905

500 32.034221

Name: policy\_deductable, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: insured\_relationship

other-relative 18.250951

wife 17.490494

not-in-family 17.015209

unmarried 16.254753

husband 15.874525

own-child 15.114068

Name: insured\_relationship, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_type

Multi-vehicle Collision 44.866920

Single Vehicle Collision 41.159696

Parked Car 7.699620

Vehicle Theft 6.273764

Name: incident\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: collision\_type

Rear Collision 45.817490

Side Collision 27.281369

Front Collision 26.901141

Name: collision\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_severity

Major Damage 41.349810

Minor Damage 28.612167

Total Loss 23.859316

Trivial Damage 6.178707

Name: incident\_severity, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: authorities\_contacted

Police 27.281369

Fire 22.718631

Ambulance 21.577947

Other 20.627376

None 7.794677

Name: authorities\_contacted, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_state

SC 26.901141

NY 24.429658

WV 18.441065

NC 12.832700

VA 10.361217

Others 7.034221

Name: incident\_state, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_city

Arlington 16.920152

Columbus 15.684411

Springfield 14.733840

Hillsdale 14.068441

Northbend 14.068441

Riverwood 13.593156

Northbrook 10.931559

Name: incident\_city, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: number\_of\_vehicles\_involved

1 55.133080

3 38.403042

2 3.992395

4 2.471483

Name: number\_of\_vehicles\_involved, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: property\_damage

NO 68.726236

YES 31.273764

Name: property\_damage, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: bodily\_injuries

2 35.076046

0 34.125475

1 30.798479

Name: bodily\_injuries, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: witnesses

2 25.855513

1 25.095057

0 24.809886

3 24.239544

Name: witnesses, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: auto\_make

Ford 9.220532

Audi 8.840304

Chevrolet 8.745247

Saab 8.174905

Dodge 8.079848

Nissan 7.889734

Suburu 7.509506

BMW 7.319392

Mercedes 6.653992

Accura 6.273764

Toyota 5.798479

Jeep 5.703422

Volkswagen 5.418251

Honda 4.372624

Name: auto\_make, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: incident\_event\_time

midnight/early morning 32.034221

forenoon 19.391635

evening 18.441065

afternoon 17.395437

night 12.737643

Name: incident\_event\_time, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: has\_Umbrella

No Umbrella 78.041825

Umbrella 21.958175

Name: has\_Umbrella, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Categorical Feature Name: customer\_type

Very Old 50.190114

Old 30.133080

Mid Term 11.121673

New 8.555133

Name: customer\_type, dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6.5 Dummy variable creation [6 Marks]**

Transform categorical variables into numerical representations using dummy variables. Ensure consistent encoding between training and validation data.

**6.5.1 Identify categorical columns for dummy variable creation [1 Mark]**

In [325]:

*# Identify the categorical columns for creating dummy variables*

cat\_columns **=** X\_resampled**.**select\_dtypes(include**=**['object', 'category'])**.**columns**.**tolist()

cat\_columns

Out[325]:

['policy\_deductable',

'insured\_relationship',

'incident\_type',

'collision\_type',

'incident\_severity',

'authorities\_contacted',

'incident\_state',

'incident\_city',

'number\_of\_vehicles\_involved',

'property\_damage',

'bodily\_injuries',

'witnesses',

'auto\_make',

'incident\_event\_time',

'has\_Umbrella',

'customer\_type']

**6.5.2 Create dummy variables for categorical columns in training data [2 Marks]**

In [326]:

*# Create dummy variables using the 'get\_dummies' for categorical columns in training data*

X\_train\_encoded **=** pd**.**get\_dummies(X\_resampled, columns**=**cat\_columns, drop\_first**=True**)

**6.5.3 Create dummy variables for categorical columns in validation data [2 Marks]**

In [327]:

*# Create dummy variables using the 'get\_dummies' for categorical columns in validation data*

X\_val\_encoded**=** pd**.**get\_dummies(X\_val, columns**=**cat\_columns, drop\_first**=True**)

In [328]:

*# This function in pandas is used to make two DataFrames have the same row and/or column structure,*

X\_train\_encoded, X\_val\_encoded **=** X\_train\_encoded**.**align(X\_val\_encoded, join**=**'left', axis**=**1, fill\_value**=**0)

**6.5.4 Create dummy variable for dependent feature in training and validation data [1 Mark]**

In [329]:

*# Create dummy variable for dependent feature in training data*

y\_train\_resampled**=**y\_resampled**.**map({'Y': 1, 'N': 0})

*# Create dummy variable for dependent feature in validation data*

y\_val\_resampled**=**y\_val**.**map({'Y': 1, 'N': 0})

**6.6 Feature scaling [3 marks]**

Scale numerical features to a common range to prevent features with larger values from dominating the model. Choose a scaling method appropriate for the data and the chosen model. Apply the same scaling to both training and validation data.

In [330]:

*# Import the necessary scaling tool from scikit-learn*

**from** sklearn.preprocessing **import** StandardScaler

*# Scale the numeric features present in the training data*

scalar **=** StandardScaler()

X\_train\_scaled **=** scalar**.**fit\_transform(X\_train\_encoded)

X\_train\_scaled **=**pd**.**DataFrame(X\_train\_scaled ,columns**=**X\_train\_encoded**.**columns)

*# Scale the numeric features present in the validation data*

X\_val\_scaled **=** scalar**.**transform(X\_val\_encoded)

X\_val\_scaled **=**pd**.**DataFrame(X\_val\_scaled ,columns**=**X\_val\_encoded**.**columns)

**7. Model Building [50 marks]**

In this task, you will be building two machine learning models: Logistic Regression and Random Forest. Each model will go through a structured process to ensure optimal performance. The key steps for each model are outlined below:

**Logistic Regression Model**

* Feature Selection using RFECV – Identify the most relevant features using Recursive Feature Elimination with Cross-Validation.
* Model Building and Multicollinearity Assessment – Build the logistic regression model and analyse statistical aspects such as p-values and VIFs to detect multicollinearity.
* Model Training and Evaluation on Training Data – Fit the model on the training data and assess initial performance.
* Finding the Optimal Cutoff – Determine the best probability threshold by analysing the sensitivity-specificity tradeoff and precision-recall tradeoff.
* FInal Prediction and Evaluation on Training Data using the Optimal Cutoff – Generate final predictions using the selected cutoff and evaluate model performance.

**Random Forest Model**

* Get Feature Importances - Obtain the importance scores for each feature and select the important features to train the model.
* Model Evaluation on Training Data – Assess performance metrics on the training data.
* Check Model Overfitting using Cross-Validation – Evaluate generalisation by performing cross-validation.
* Hyperparameter Tuning using Grid Search – Optimise model performance by fine-tuning hyperparameters.
* Final Model and Evaluation on Training Data – Train the final model using the best parameters and assess its performance.

**7.1 Feature selection [4 marks]**

Identify and select the most relevant features for building a logistic regression model using Recursive Feature Elimination with Cross-Validation (RFECV).

**7.1.1 Import necessary libraries [1 Mark]**

In [331]:

*# Import necessary libraries*

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.feature\_selection **import** RFECV

**from** sklearn.model\_selection **import** StratifiedKFold

In [332]:

estimator**=**LogisticRegression()

cv **=** StratifiedKFold(n\_splits**=**5)

**7.1.2 Perform feature selection [2 Mark]**

In [333]:

*# Apply RFECV to identify the most relevant features*

rfe**=**RFECV(estimator,scoring**=**'f1',step**=**1,cv**=**5)

rfe**.**fit(X\_train\_scaled,y\_train\_resampled)

Out[333]:

RFECV(cv=5, estimator=LogisticRegression(), scoring='f1')

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [334]:

*# Display the features ranking by RFECV in a DataFrame*

list(zip(X\_train\_scaled**.**columns,rfe**.**support\_,rfe**.**ranking\_))

Out[334]:

[('policy\_annual\_premium', False, 11),

('capital-gains', False, 25),

('total\_claim\_amount', False, 20),

('incident\_days\_since\_policy', False, 23),

('policy\_deductable\_1000', True, 1),

('policy\_deductable\_2000', False, 17),

('insured\_relationship\_not-in-family', True, 1),

('insured\_relationship\_other-relative', True, 1),

('insured\_relationship\_own-child', True, 1),

('insured\_relationship\_unmarried', True, 1),

('insured\_relationship\_wife', True, 1),

('incident\_type\_Parked Car', True, 1),

('incident\_type\_Single Vehicle Collision', False, 15),

('incident\_type\_Vehicle Theft', False, 18),

('collision\_type\_Rear Collision', False, 9),

('collision\_type\_Side Collision', True, 1),

('incident\_severity\_Minor Damage', True, 1),

('incident\_severity\_Total Loss', True, 1),

('incident\_severity\_Trivial Damage', True, 1),

('authorities\_contacted\_Fire', True, 1),

('authorities\_contacted\_None', True, 1),

('authorities\_contacted\_Other', True, 1),

('authorities\_contacted\_Police', True, 1),

('incident\_state\_NY', True, 1),

('incident\_state\_Others', True, 1),

('incident\_state\_SC', False, 4),

('incident\_state\_VA', False, 21),

('incident\_state\_WV', True, 1),

('incident\_city\_Columbus', False, 7),

('incident\_city\_Hillsdale', False, 10),

('incident\_city\_Northbend', True, 1),

('incident\_city\_Northbrook', True, 1),

('incident\_city\_Riverwood', False, 3),

('incident\_city\_Springfield', False, 6),

('number\_of\_vehicles\_involved\_2', True, 1),

('number\_of\_vehicles\_involved\_3', False, 14),

('number\_of\_vehicles\_involved\_4', False, 19),

('property\_damage\_YES', True, 1),

('bodily\_injuries\_1', True, 1),

('bodily\_injuries\_2', False, 24),

('witnesses\_1', False, 16),

('witnesses\_2', True, 1),

('witnesses\_3', True, 1),

('auto\_make\_Audi', True, 1),

('auto\_make\_BMW', True, 1),

('auto\_make\_Chevrolet', True, 1),

('auto\_make\_Dodge', True, 1),

('auto\_make\_Ford', True, 1),

('auto\_make\_Honda', True, 1),

('auto\_make\_Jeep', True, 1),

('auto\_make\_Mercedes', True, 1),

('auto\_make\_Nissan', False, 13),

('auto\_make\_Saab', True, 1),

('auto\_make\_Suburu', True, 1),

('auto\_make\_Toyota', False, 2),

('auto\_make\_Volkswagen', True, 1),

('incident\_event\_time\_forenoon', True, 1),

('incident\_event\_time\_afternoon', False, 22),

('incident\_event\_time\_evening', True, 1),

('incident\_event\_time\_night', False, 5),

('has\_Umbrella\_Umbrella', True, 1),

('customer\_type\_Mid Term', True, 1),

('customer\_type\_Old', False, 12),

('customer\_type\_Very Old', False, 8)]

**7.1.2 Retain the selected features [1 Mark]**

In [335]:

*# Put columns selected by RFECV into variable 'col'*

col**=**X\_train\_scaled**.**columns[rfe**.**support\_] *#FEATURES EELECTED BY THE MODEL*

col

Out[335]:

Index(['policy\_deductable\_1000', 'insured\_relationship\_not-in-family',

'insured\_relationship\_other-relative', 'insured\_relationship\_own-child',

'insured\_relationship\_unmarried', 'insured\_relationship\_wife',

'incident\_type\_Parked Car', 'collision\_type\_Side Collision',

'incident\_severity\_Minor Damage', 'incident\_severity\_Total Loss',

'incident\_severity\_Trivial Damage', 'authorities\_contacted\_Fire',

'authorities\_contacted\_None', 'authorities\_contacted\_Other',

'authorities\_contacted\_Police', 'incident\_state\_NY',

'incident\_state\_Others', 'incident\_state\_WV', 'incident\_city\_Northbend',

'incident\_city\_Northbrook', 'number\_of\_vehicles\_involved\_2',

'property\_damage\_YES', 'bodily\_injuries\_1', 'witnesses\_2',

'witnesses\_3', 'auto\_make\_Audi', 'auto\_make\_BMW', 'auto\_make\_Chevrolet',

'auto\_make\_Dodge', 'auto\_make\_Ford', 'auto\_make\_Honda',

'auto\_make\_Jeep', 'auto\_make\_Mercedes', 'auto\_make\_Saab',

'auto\_make\_Suburu', 'auto\_make\_Volkswagen',

'incident\_event\_time\_forenoon', 'incident\_event\_time\_evening',

'has\_Umbrella\_Umbrella', 'customer\_type\_Mid Term'],

dtype='object')

**7.2 Build Logistic Regression Model [12 marks]**

After selecting the optimal features using RFECV, utilise these features to build a logistic regression model with Statsmodels. This approach enables a detailed statistical analysis of the model, including the assessment of p-values and Variance Inflation Factors (VIFs). Evaluating these metrics is crucial for detecting multicollinearity and ensuring that the selected predictors are not highly correlated.

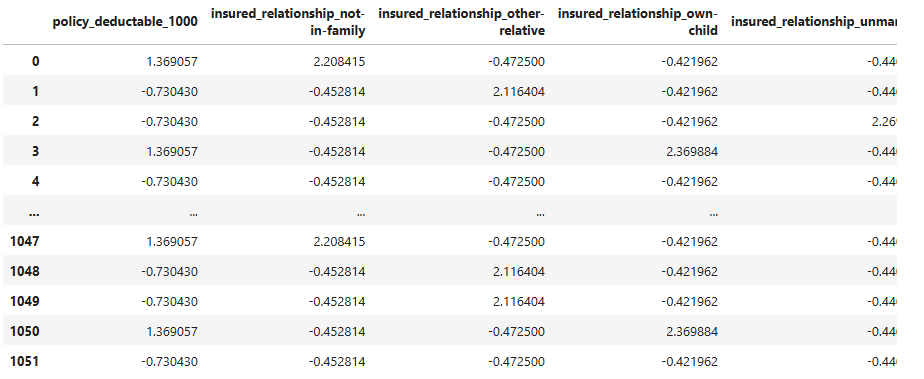
**7.2.1 Select relevant features and add constant in training data [1 Mark]**

In [336]:

*# Select only the columns selected by RFECV*

X\_selected\_RFECV**=**X\_train\_scaled[col]

X\_selected\_RFECV

Out[336]: 

1052 rows × 40 columns

In [337]:

*# Import statsmodels and add constant*

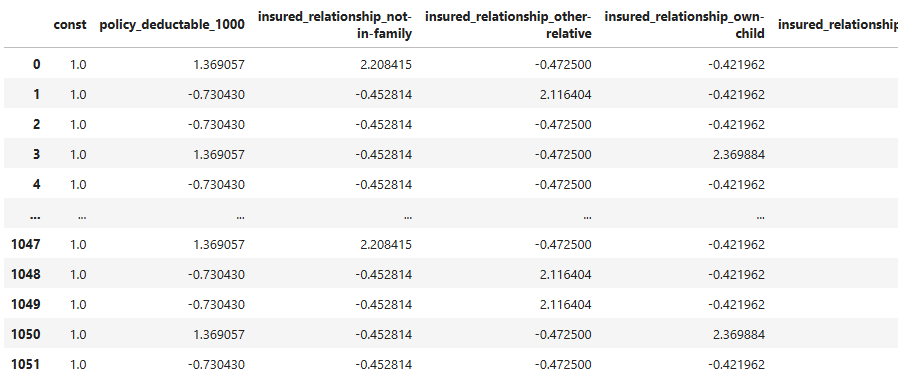
**import** statsmodels.api **as** sm

*# Check the data*

X\_selected\_RFECV**=**sm**.**add\_constant(X\_selected\_RFECV)

X\_selected\_RFECV

Out[337]:



1052 rows × 41 columns

**7.2.2 Fit logistic regression model [2 Marks]**

In [338]:

*# Fit a logistic Regression model on X\_train after adding a constant and output the summary*

logistic\_model**=**sm**.**Logit(y\_train\_resampled,X\_selected\_RFECV)**.**fit()

print(logistic\_model**.**summary())

Optimization terminated successfully.

Current function value: 0.457474

Iterations 6

Logit Regression Results

==============================================================================

Dep. Variable: fraud\_reported No. Observations: 1052

Model: Logit Df Residuals: 1011

Method: MLE Df Model: 40

Date: Tue, 20 May 2025 Pseudo R-squ.: 0.3400

Time: 16:16:08 Log-Likelihood: -481.26

converged: True LL-Null: -729.19

Covariance Type: nonrobust LLR p-value: 5.859e-80

=======================================================================================================

coef std err z P>|z| [0.025 0.975]

-------------------------------------------------------------------------------------------------------

const 0.0048 0.083 0.058 0.954 -0.158 0.167

policy\_deductable\_1000 -0.1858 0.087 -2.147 0.032 -0.355 -0.016

insured\_relationship\_not-in-family 0.2534 0.112 2.262 0.024 0.034 0.473

insured\_relationship\_other-relative 0.1577 0.113 1.396 0.163 -0.064 0.379

insured\_relationship\_own-child -0.2520 0.111 -2.261 0.024 -0.470 -0.034

insured\_relationship\_unmarried 0.2921 0.112 2.599 0.009 0.072 0.512

insured\_relationship\_wife 0.1001 0.112 0.894 0.371 -0.119 0.320

incident\_type\_Parked Car 0.2158 0.116 1.865 0.062 -0.011 0.442

collision\_type\_Side Collision -0.1507 0.087 -1.729 0.084 -0.322 0.020

incident\_severity\_Minor Damage -1.4838 0.107 -13.840 0.000 -1.694 -1.274

incident\_severity\_Total Loss -1.0622 0.091 -11.615 0.000 -1.241 -0.883

incident\_severity\_Trivial Damage -1.0428 0.121 -8.640 0.000 -1.279 -0.806

authorities\_contacted\_Fire -0.3365 0.106 -3.179 0.001 -0.544 -0.129

authorities\_contacted\_None -0.0985 0.123 -0.803 0.422 -0.339 0.142

authorities\_contacted\_Other -0.1168 0.104 -1.118 0.264 -0.321 0.088

authorities\_contacted\_Police -0.1927 0.110 -1.754 0.079 -0.408 0.023

incident\_state\_NY -0.2211 0.090 -2.454 0.014 -0.398 -0.045

incident\_state\_Others 0.1589 0.089 1.777 0.076 -0.016 0.334

incident\_state\_WV -0.4507 0.092 -4.894 0.000 -0.631 -0.270

incident\_city\_Northbend -0.1543 0.086 -1.790 0.073 -0.323 0.015

incident\_city\_Northbrook -0.1868 0.087 -2.138 0.033 -0.358 -0.016

number\_of\_vehicles\_involved\_2 0.1429 0.088 1.627 0.104 -0.029 0.315

property\_damage\_YES 0.1551 0.085 1.831 0.067 -0.011 0.321

bodily\_injuries\_1 -0.1269 0.085 -1.494 0.135 -0.293 0.040

witnesses\_2 0.2445 0.088 2.774 0.006 0.072 0.417

witnesses\_3 0.0969 0.091 1.066 0.286 -0.081 0.275

auto\_make\_Audi 0.3171 0.100 3.159 0.002 0.120 0.514

auto\_make\_BMW 0.3864 0.091 4.258 0.000 0.209 0.564

auto\_make\_Chevrolet 0.2835 0.098 2.894 0.004 0.091 0.476

auto\_make\_Dodge 0.1962 0.094 2.090 0.037 0.012 0.380

auto\_make\_Ford 0.2573 0.103 2.488 0.013 0.055 0.460

auto\_make\_Honda 0.2221 0.089 2.497 0.013 0.048 0.397

auto\_make\_Jeep 0.1191 0.095 1.255 0.210 -0.067 0.305

auto\_make\_Mercedes 0.0857 0.090 0.949 0.342 -0.091 0.263

auto\_make\_Saab 0.2375 0.096 2.480 0.013 0.050 0.425

auto\_make\_Suburu 0.0942 0.095 0.988 0.323 -0.093 0.281

auto\_make\_Volkswagen 0.1594 0.091 1.760 0.078 -0.018 0.337

incident\_event\_time\_forenoon -0.0859 0.085 -1.015 0.310 -0.252 0.080

incident\_event\_time\_evening -0.1009 0.088 -1.143 0.253 -0.274 0.072

has\_Umbrella\_Umbrella 0.3015 0.084 3.569 0.000 0.136 0.467

customer\_type\_Mid Term -0.1580 0.089 -1.770 0.077 -0.333 0.017

=======================================================================================================

**Model Interpretation**

The output summary table will provide the features used for building model along with coefficient of each of the feature and their p-value. The p-value in a logistic regression model is used to assess the statistical significance of each coefficient. Lesser the p-value, more significant the feature is in the model.

A positive coefficient will indicate that an increase in the value of feature would increase the odds of the event occurring. On the other hand, a negative coefficient means the opposite, i.e, an increase in the value of feature would decrease the odds of the event occurring.

Now check VIFs for presence of multicollinearity in the model.

**7.2.3 Evaluate VIF of features to assess multicollinearity [2 Marks]**

In [339]:

*# Import 'variance\_inflation\_factor'*

**from** statsmodels.stats.outliers\_influence **import** variance\_inflation\_factor

In [340]:

*# Make a VIF DataFrame for all the variables present*

**def** calculateVIF(df):

df **=** df**.**copy()

*# df = df.astype({col: 'int64' for col in df.select\_dtypes(include=['bool']).columns})*

vif **=** pd**.**DataFrame()

vif['Features'] **=** df**.**columns

vif['VIF'] **=** [variance\_inflation\_factor(df**.**values, i) **for** i **in** range(df**.**shape[1])]

vif['VIF'] **=** round(vif['VIF'], 2)

vif **=** vif**.**sort\_values(by **=** "VIF", ascending **=** **False**)

**return** vif

calculateVIF(X\_selected\_RFECV)

Out[340]:

|  | **Features** | **VIF** |
| --- | --- | --- |
| **13** | authorities\_contacted\_None | 2.35 |
| **7** | incident\_type\_Parked Car | 2.00 |
| **2** | insured\_relationship\_not-in-family | 1.90 |
| **3** | insured\_relationship\_other-relative | 1.90 |
| **6** | insured\_relationship\_wife | 1.88 |
| **5** | insured\_relationship\_unmarried | 1.85 |
| **15** | authorities\_contacted\_Police | 1.85 |
| **4** | insured\_relationship\_own-child | 1.79 |
| **14** | authorities\_contacted\_Other | 1.70 |
| **12** | authorities\_contacted\_Fire | 1.68 |
| **11** | incident\_severity\_Trivial Damage | 1.56 |
| **9** | incident\_severity\_Minor Damage | 1.46 |
| **30** | auto\_make\_Ford | 1.42 |
| **26** | auto\_make\_Audi | 1.42 |
| **34** | auto\_make\_Saab | 1.41 |
| **28** | auto\_make\_Chevrolet | 1.40 |
| **29** | auto\_make\_Dodge | 1.36 |
| **27** | auto\_make\_BMW | 1.35 |
| **35** | auto\_make\_Suburu | 1.33 |
| **33** | auto\_make\_Mercedes | 1.32 |
| **32** | auto\_make\_Jeep | 1.28 |
| **10** | incident\_severity\_Total Loss | 1.28 |
| **36** | auto\_make\_Volkswagen | 1.26 |
| **31** | auto\_make\_Honda | 1.23 |
| **18** | incident\_state\_WV | 1.23 |
| **25** | witnesses\_3 | 1.22 |
| **24** | witnesses\_2 | 1.22 |
| **16** | incident\_state\_NY | 1.22 |
| **17** | incident\_state\_Others | 1.18 |
| **38** | incident\_event\_time\_evening | 1.14 |
| **8** | collision\_type\_Side Collision | 1.14 |
| **21** | number\_of\_vehicles\_involved\_2 | 1.13 |
| **37** | incident\_event\_time\_forenoon | 1.13 |
| **1** | policy\_deductable\_1000 | 1.12 |
| **19** | incident\_city\_Northbend | 1.12 |
| **39** | has\_Umbrella\_Umbrella | 1.10 |
| **40** | customer\_type\_Mid Term | 1.10 |
| **20** | incident\_city\_Northbrook | 1.09 |
| **23** | bodily\_injuries\_1 | 1.08 |
| **22** | property\_damage\_YES | 1.08 |
| **0** | const | 1.00 |

In [341]:

X\_selected\_RFECV\_v2**=**X\_selected\_RFECV**.**drop('authorities\_contacted\_None',axis**=**1)

logistic\_model**=**sm**.**Logit(y\_train\_resampled,X\_selected\_RFECV\_v2)**.**fit()

X\_selected\_RFECV\_v2**=**X\_selected\_RFECV\_v2**.**drop('insured\_relationship\_wife',axis**=**1)

logistic\_model**=**sm**.**Logit(y\_train\_resampled,X\_selected\_RFECV\_v2)**.**fit()

X\_selected\_RFECV\_v2**=**X\_selected\_RFECV\_v2**.**drop('auto\_make\_Suburu',axis**=**1)

logistic\_model**=**sm**.**Logit(y\_train\_resampled,X\_selected\_RFECV\_v2)**.**fit()

X\_selected\_RFECV\_v2**=**X\_selected\_RFECV\_v2**.**drop('auto\_make\_Mercedes',axis**=**1)

logistic\_model**=**sm**.**Logit(y\_train\_resampled,X\_selected\_RFECV\_v2)**.**fit()

print(logistic\_model**.**summary())

Optimization terminated successfully.

Current function value: 0.457784

Iterations 6

Optimization terminated successfully.

Current function value: 0.458117

Iterations 6

Optimization terminated successfully.

Current function value: 0.458519

Iterations 6

Optimization terminated successfully.

Current function value: 0.458697

Iterations 6

Logit Regression Results

==============================================================================

Dep. Variable: fraud\_reported No. Observations: 1052

Model: Logit Df Residuals: 1015

Method: MLE Df Model: 36

Date: Tue, 20 May 2025 Pseudo R-squ.: 0.3382

Time: 16:16:09 Log-Likelihood: -482.55

converged: True LL-Null: -729.19

Covariance Type: nonrobust LLR p-value: 1.071e-81

=======================================================================================================

coef std err z P>|z| [0.025 0.975]

-------------------------------------------------------------------------------------------------------

const 0.0114 0.083 0.138 0.890 -0.150 0.173

policy\_deductable\_1000 -0.1818 0.086 -2.114 0.035 -0.350 -0.013

insured\_relationship\_not-in-family 0.1993 0.093 2.133 0.033 0.016 0.382

insured\_relationship\_other-relative 0.1161 0.094 1.242 0.214 -0.067 0.299

insured\_relationship\_own-child -0.2872 0.094 -3.069 0.002 -0.471 -0.104

insured\_relationship\_unmarried 0.2485 0.095 2.613 0.009 0.062 0.435

incident\_type\_Parked Car 0.1650 0.095 1.744 0.081 -0.020 0.350

collision\_type\_Side Collision -0.1462 0.086 -1.690 0.091 -0.316 0.023

incident\_severity\_Minor Damage -1.5010 0.106 -14.156 0.000 -1.709 -1.293

incident\_severity\_Total Loss -1.0594 0.091 -11.598 0.000 -1.238 -0.880

incident\_severity\_Trivial Damage -1.0587 0.117 -9.033 0.000 -1.288 -0.829

authorities\_contacted\_Fire -0.3076 0.102 -3.030 0.002 -0.507 -0.109

authorities\_contacted\_Other -0.0967 0.101 -0.960 0.337 -0.294 0.101

authorities\_contacted\_Police -0.1669 0.101 -1.653 0.098 -0.365 0.031

incident\_state\_NY -0.2257 0.089 -2.526 0.012 -0.401 -0.051

incident\_state\_Others 0.1635 0.088 1.850 0.064 -0.010 0.337

incident\_state\_WV -0.4413 0.092 -4.817 0.000 -0.621 -0.262

incident\_city\_Northbend -0.1547 0.086 -1.796 0.073 -0.324 0.014

incident\_city\_Northbrook -0.1829 0.087 -2.114 0.035 -0.352 -0.013

number\_of\_vehicles\_involved\_2 0.1409 0.088 1.603 0.109 -0.031 0.313

property\_damage\_YES 0.1481 0.084 1.763 0.078 -0.017 0.313

bodily\_injuries\_1 -0.1332 0.085 -1.576 0.115 -0.299 0.032

witnesses\_2 0.2352 0.088 2.676 0.007 0.063 0.408

witnesses\_3 0.0786 0.090 0.878 0.380 -0.097 0.254

auto\_make\_Audi 0.2734 0.092 2.975 0.003 0.093 0.453

auto\_make\_BMW 0.3464 0.084 4.145 0.000 0.183 0.510

auto\_make\_Chevrolet 0.2415 0.088 2.736 0.006 0.068 0.414

auto\_make\_Dodge 0.1592 0.086 1.846 0.065 -0.010 0.328

auto\_make\_Ford 0.2173 0.096 2.272 0.023 0.030 0.405

auto\_make\_Honda 0.1899 0.084 2.267 0.023 0.026 0.354

auto\_make\_Jeep 0.0902 0.089 1.018 0.309 -0.083 0.264

auto\_make\_Saab 0.1886 0.088 2.155 0.031 0.017 0.360

auto\_make\_Volkswagen 0.1243 0.084 1.482 0.138 -0.040 0.289

incident\_event\_time\_forenoon -0.0946 0.084 -1.122 0.262 -0.260 0.071

incident\_event\_time\_evening -0.0976 0.088 -1.108 0.268 -0.270 0.075

has\_Umbrella\_Umbrella 0.3139 0.084 3.748 0.000 0.150 0.478

customer\_type\_Mid Term -0.1729 0.089 -1.949 0.051 -0.347 0.001

=======================================================================================================

In [342]:

*# # Make a VIF DataFrame for all the variables present*

*# def calculateVIF(df):*

*# df = df.copy()*

*# # df = df.astype({col: 'int64' for col in df.select\_dtypes(include=['bool']).columns})*

*# vif = pd.DataFrame()*

*# vif['Features'] = df.columns*

*# vif['VIF'] = [variance\_inflation\_factor(df.values, i) for i in range(df.shape[1])]*

*# vif['VIF'] = round(vif['VIF'], 2)*

*# vif = vif.sort\_values(by = "VIF", ascending = False)*

*# return vif*

*# calculateVIF(X\_selected\_RFECV\_v2)*

In [416]:

selected\_cols\_logreg**=**X\_selected\_RFECV\_v2**.**columns**.**values[1:]

Proceed to the next step if p-values and VIFs are within acceptable ranges. If you observe high p-values or VIFs, drop the features and retrain the model. [THIS IS OPTIONAL]

**7.2.4 Make predictions on training data [1 Mark]**

In [343]:

*# Predict the probabilities on the training data*

X\_train\_pred**=**logistic\_model**.**predict(X\_selected\_RFECV\_v2)

X\_train\_pred

*# Reshape it into an array*

X\_train\_pred**.**values

Out[343]:

array([0.64383717, 0.348978 , 0.12412799, ..., 0.85089623, 0.1564893 ,

0.63357074])

In [344]:

y\_train\_resampled

Out[344]:

0 0

1 0

2 0

3 0

4 0

..

1047 1

1048 1

1049 1

1050 1

1051 1

Name: fraud\_reported, Length: 1052, dtype: int64

**7.2.5 Create a DataFrame that includes actual fraud reported flags, predicted probabilities, and a column indicating predicted classifications based on a cutoff value of 0.5 [1 Mark]**

In [345]:

*# Create a new DataFrame containing the actual fraud reported flag and the probabilities predicted by the model*

X\_train\_pred**=**pd**.**DataFrame({'Actual Fraud Reported':y\_train\_resampled,'Predicted Probability':X\_train\_pred})

*# # Create new column indicating predicted classifications based on a cutoff value of 0.5*

X\_train\_pred['Predicted Class'] **=** (X\_train\_pred['Predicted Probability'] **>** 0.5)**.**astype(int)

X\_train\_pred

Out[345]:

|  | **Actual Fraud Reported** | **Predicted Probability** | **Predicted Class** |
| --- | --- | --- | --- |
| **0** | 0 | 0.643837 | 1 |
| **1** | 0 | 0.348978 | 0 |
| **2** | 0 | 0.124128 | 0 |
| **3** | 0 | 0.061962 | 0 |
| **4** | 0 | 0.098087 | 0 |
| **...** | ... | ... | ... |
| **1047** | 1 | 0.829088 | 1 |
| **1048** | 1 | 0.249246 | 0 |
| **1049** | 1 | 0.850896 | 1 |
| **1050** | 1 | 0.156489 | 0 |
| **1051** | 1 | 0.633571 | 1 |

1052 rows × 3 columns

**Model performance evaluation**

Evaluate the performance of the model based on predictions made on the training data.

**7.2.6 Check the accuracy of the model [1 Mark]**

In [346]:

*# Import metrics from sklearn for evaluation*

**from** sklearn **import** metrics

*# Check the accuracy of the model*

metrics**.**accuracy\_score(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred['Predicted Class'])

Out[346]:

0.7899239543726235

**7.2.7 Create a confusion matrix based on the predictions made on the training data [1 Mark]**

In [347]:

*# Create confusion matrix*

confusion\_matrix**=**metrics**.**confusion\_matrix(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred['Predicted Class'])

**7.2.8 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [348]:

*# Create variables for true positive, true negative, false positive and false negative*

TN, FP, FN, TP **=** confusion\_matrix**.**ravel()

**7.2.9 Calculate sensitivity, specificity, precision, recall and F1-score [2 Marks]**

In [349]:

**from** sklearn.metrics **import** f1\_score

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.78

Specificity: 0.80

Precision: 0.80

Recall: 0.78

F1: 0.79

**7.3 Find the Optimal Cutoff [12 marks]**

Find the optimal cutoff to improve model performance by evaluating various cutoff values and their impact on relevant metrics.

**7.3.1 Plot ROC Curve to visualise the trade-off between true positive rate and false positive rate across different classification thresholds [2 Marks]**

In [350]:

*# Import libraries or function to plot the ROC curve*

**from** sklearn.metrics **import** roc\_curve,auc

*# Define ROC function*

fpr, tpr, thresholds**=**roc\_curve(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred['Predicted Probability'])

In [351]:

*# Call the ROC function*

auc\_score**=**auc(fpr,tpr)

print(auc\_score)

*# Plotting*

plt**.**figure(figsize**=**(6, 4))

plt**.**plot(fpr, tpr, label**=**f'AUC = {auc\_score:.2f}')

plt**.**plot([0, 1], [0, 1], 'k--', label**=**'No Skill')

plt**.**xlabel('False Positive Rate')

plt**.**ylabel('True Positive Rate')

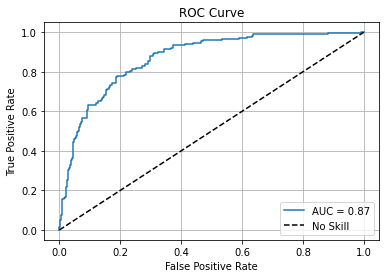
plt**.**title('ROC Curve')

plt**.**legend(loc**=**'lower right')

plt**.**grid(**True**)

plt**.**show()

0.8690236955861728



**Sensitivity and Specificity tradeoff**

After analysing the area under the curve of the ROC, check the sensitivity and specificity tradeoff to find the optimal cutoff point.

**7.3.2 Predict on training data at various probability cutoffs [1 Mark]**

In [352]:

*# Create columns with different probability cutoffs to explore the impact of cutoff on model performance*

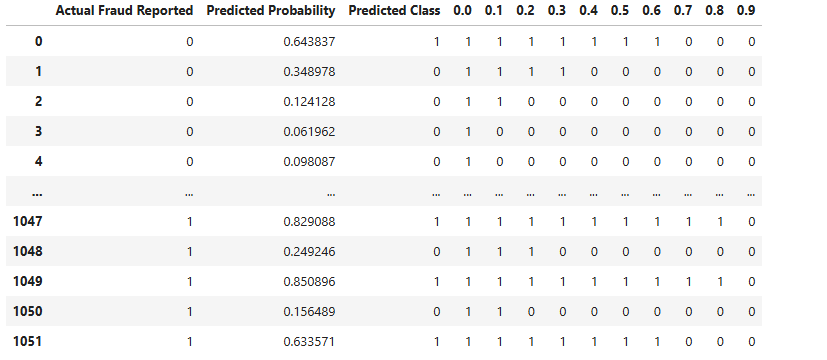
thresholds**=**[float(x)**/**10 **for** x **in** range(10)]

**for** threshold **in** thresholds:

X\_train\_pred[threshold] **=** (X\_train\_pred['Predicted Probability'] **>** threshold)**.**astype(int)

X\_train\_pred

Out[352]:



1052 rows × 13 columns

**7.3.3 Plot accuracy, sensitivity, specificity at different values of probability cutoffs [2 Marks]**

In [353]:

*# Create a DataFrame to see the values of accuracy, sensitivity, and specificity at different values of probability cutoffs*

**from** sklearn **import** metrics

train\_pred\_df**=**{}

train\_metrics**=**[]

**for** threshold **in** thresholds:

accuracy**=**metrics**.**accuracy\_score(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred[threshold])

confusion\_matrix**=**metrics**.**confusion\_matrix(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred[threshold])

TN, FP, FN, TP **=** confusion\_matrix**.**ravel()

sensitivity**=**TP**/**(TP**+**FN)

specificity**=**TN**/**(TN**+**FP)

train\_metrics**.**append([threshold, accuracy, sensitivity, specificity])

train\_pred\_df**=**pd**.**DataFrame(train\_metrics,columns**=**['threshold', 'accuracy', 'sensitivity', 'specificity'])

train\_pred\_df

Out[353]:

|  | **threshold** | **accuracy** | **sensitivity** | **specificity** |
| --- | --- | --- | --- | --- |
| **0** | 0.0 | 0.500000 | 1.000000 | 0.000000 |
| **1** | 0.1 | 0.611217 | 0.990494 | 0.231939 |
| **2** | 0.2 | 0.715779 | 0.960076 | 0.471483 |
| **3** | 0.3 | 0.779468 | 0.916350 | 0.642586 |
| **4** | 0.4 | 0.777567 | 0.821293 | 0.733840 |
| **5** | 0.5 | 0.789924 | 0.777567 | 0.802281 |
| **6** | 0.6 | 0.772814 | 0.699620 | 0.846008 |
| **7** | 0.7 | 0.769011 | 0.633080 | 0.904943 |
| **8** | 0.8 | 0.727186 | 0.522814 | 0.931559 |
| **9** | 0.9 | 0.601711 | 0.228137 | 0.975285 |

In [354]:

*# Plot accuracy, sensitivity, and specificity at different values of probability cutoffs*

plt**.**figure(figsize**=**(10, 6))

plt**.**plot(train\_pred\_df['threshold'], train\_pred\_df['accuracy'], label**=**'Accuracy')

plt**.**plot(train\_pred\_df['threshold'], train\_pred\_df['sensitivity'], label**=**'Sensitivity (Recall)', color**=**'green')

plt**.**plot(train\_pred\_df['threshold'], train\_pred\_df['specificity'], label**=**'Specificity', color**=**'red')

plt**.**xlabel('Probability Threshold')

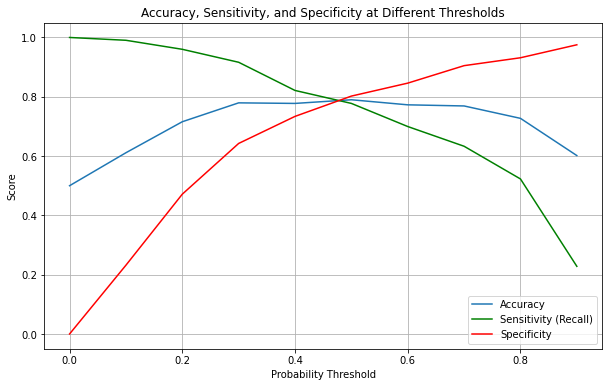
plt**.**ylabel('Score')

plt**.**title('Accuracy, Sensitivity, and Specificity at Different Thresholds')

plt**.**legend()

plt**.**grid(**True**)

plt**.**show()



Choosing a optimal cutoff- -the cutoff which maximizes accuracy, sensitivity and specificity.

-Here we are more concerned about FN(fraud cases considered non fraud), we would take a trade off where both recall and specificity would be balanced such that we can also prioritize recall,his minimizes False Negatives (missed frauds), even at the cost of more False Positives.

-so choosing a lower cutoff,0.4,(sweet spot) where sensitivity is still high, but specificity isn't too low.

**7.3.4 Create a column for final prediction based on optimal cutoff [1 Mark]**

In [355]:

*# Create a column for final prediction based on the optimal cutoff*

X\_train\_pred['Final Prediction']**=**X\_train\_pred['Predicted Probability']**.**map(**lambda** x: 1 **if** x**>**0.4 **else** 0)

**7.3.5 Calculate the accuracy [1 Mark]**

In [356]:

*# Check the accuracy now*

metrics**.**accuracy\_score(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred['Final Prediction'])

Out[356]:

0.7775665399239544

**7.3.6 Create confusion matrix [1 Mark]**

In [357]:

*# Create the confusion matrix once again*

final\_cm**=**metrics**.**confusion\_matrix(X\_train\_pred['Actual Fraud Reported'],X\_train\_pred['Final Prediction'])

**7.3.7 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [358]:

*# Create variables for true positive, true negative, false positive and false negative*

TN, FP, FN, TP **=** final\_cm**.**ravel()

**7.3.8 Calculate sensitivity, specificity, precision, recall and F1-score of the model [2 Mark]**

In [359]:

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.82

Specificity: 0.73

Precision: 0.76

Recall: 0.82

F1: 0.79

**Precision and Recall tradeoff**

Check optimal cutoff value by plotting precision-recall curve, and adjust the cutoff based on precision and recall tradeoff if required.

In [360]:

*# Import precision-recall curve function*

**from** sklearn.metrics **import** precision\_recall\_curve

**7.3.9 Plot precision-recall curve [1 Mark]**

In [361]:

*# Plot precision-recall curve*

p, r, thresholds **=** precision\_recall\_curve(X\_train\_pred['Actual Fraud Reported'], X\_train\_pred['Final Prediction'])

pr\_auc **=** auc(r, p)

plt**.**figure(figsize**=**(8, 6))

plt**.**plot(r, p, marker**=**'.', label**=**f'PR Curve (AUC = {pr\_auc:.2f})", color="blue')

plt**.**xlabel("Recall")

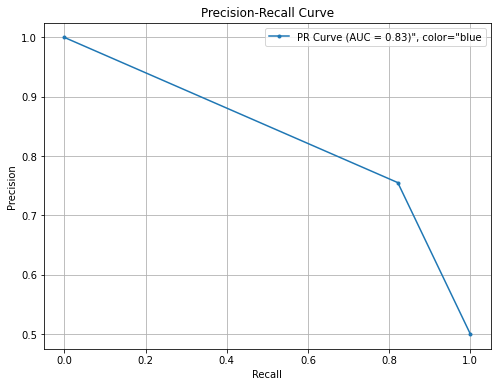
plt**.**ylabel("Precision")

plt**.**title("Precision-Recall Curve")

plt**.**legend()

plt**.**grid()

plt**.**show()



🔍 Interpretation of the above PR Curve:

PR AUC is especially valuable in imbalanced datasets.

The AUC (Area Under Curve) is 0.83, which is quite good & means the model maintains a good balance of precision and recall.

The curve starts high on the y-axis (high precision) but drops as recall increases, indicating the classic trade-off: As the model tries to catch more positives (higher recall), it may misclassify more negatives as positives, reducing precision.

In our case we are more concerned about reducing False Negatives, catching more frauds, thus increasing Recall.

**7.4 Build Random Forest Model [12 marks]**

Now that you have built a logistic regression model, let's move on to building a random forest model.

**7.4.1 Import necessary libraries**

In [362]:

*# Import necessary libraries*

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** classification\_report

**from** sklearn.model\_selection **import** cross\_val\_score, GridSearchCV

**7.4.2 Build the random forest model [1 Mark]**

In [363]:

*# Build a base random forest model*

rf **=** RandomForestClassifier(random\_state**=**42,n\_jobs**=-**1, n\_estimators**=**50, max\_depth**=**5,oob\_score**=True**)

rf**.**fit(X\_train\_scaled,y\_train\_resampled)

Out[363]:

RandomForestClassifier(max\_depth=5, n\_estimators=50, n\_jobs=-1, oob\_score=True,

random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

**7.4.3 Get feature importance scores and select important features [2 Marks]**

In [364]:

*# Get feature importance scores from the trained model*

rf**.**feature\_importances\_

*# Create a DataFrame to visualise the importance scores*

imp\_df **=** pd**.**DataFrame({

"Varname": X\_train\_scaled**.**columns,

"Imp": rf**.**feature\_importances\_

})

In [365]:

imp\_df**.**sort\_values(by**=**"Imp", ascending**=False**)**.**head(20)

Out[365]:

|  | **Varname** | **Imp** |
| --- | --- | --- |
| **16** | incident\_severity\_Minor Damage | 0.188639 |
| **2** | total\_claim\_amount | 0.112531 |
| **17** | incident\_severity\_Total Loss | 0.071269 |
| **18** | incident\_severity\_Trivial Damage | 0.054930 |
| **0** | policy\_annual\_premium | 0.051539 |
| **13** | incident\_type\_Vehicle Theft | 0.040071 |
| **3** | incident\_days\_since\_policy | 0.035713 |
| **1** | capital-gains | 0.034908 |
| **27** | incident\_state\_WV | 0.032692 |
| **24** | incident\_state\_Others | 0.016837 |
| **39** | bodily\_injuries\_2 | 0.016531 |
| **8** | insured\_relationship\_own-child | 0.016085 |
| **20** | authorities\_contacted\_None | 0.015836 |
| **11** | incident\_type\_Parked Car | 0.014589 |
| **14** | collision\_type\_Rear Collision | 0.013145 |
| **4** | policy\_deductable\_1000 | 0.012081 |
| **63** | customer\_type\_Very Old | 0.011286 |
| **25** | incident\_state\_SC | 0.010458 |
| **43** | auto\_make\_Audi | 0.009844 |
| **31** | incident\_city\_Northbrook | 0.009165 |

In [366]:

*# Select features with high importance scores*

selected\_cols**=**imp\_df[imp\_df['Imp']**>=**0.01]['Varname']**.**values

*# Create a new training data with only the selected features*

X\_train\_scaled\_selected**=**X\_train\_scaled[selected\_cols]

In [367]:

X\_train\_scaled\_selected**.**shape

Out[367]:

(1052, 18)

**7.4.4 Train the model with selected features [1 Mark]**

In [368]:

*# Fit the model on the training data with selected features*

rf**.**fit(X\_train\_scaled\_selected,y\_train\_resampled)

Out[368]:

RandomForestClassifier(max\_depth=5, n\_estimators=50, n\_jobs=-1, oob\_score=True,

random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

**7.4.5 Generate predictions on the training data [1 Mark]**

In [369]:

*# Generate predictions on training data*

y\_train\_pred**=**rf**.**predict(X\_train\_scaled\_selected)

**7.4.6 Check accuracy of the model [1 Mark]**

In [370]:

*# Check accuracy of the model*

metrics**.**accuracy\_score(y\_train\_resampled,y\_train\_pred)

Out[370]:

0.8526615969581749

**7.4.7 Create confusion matrix [1 Mark]**

In [371]:

*# Create the confusion matrix to visualise the performance*

rf\_cm**=**metrics**.**confusion\_matrix(y\_train\_pred,y\_train\_resampled)

**7.4.8 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [372]:

*# Create variables for true positive, true negative, false positive and false negative*

TN, FP, FN, TP **=** rf\_cm**.**ravel()

**7.4.9 Calculate sensitivity, specificity, precision, recall and F1-score of the model [2 Marks]**

In [373]:

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.85

Specificity: 0.85

Precision: 0.85

Recall: 0.85

F1: 0.85

**7.4.10 Check if the model is overfitting training data using cross validation [2 marks]**

In [374]:

*# Use cross validation to check if the model is overfitting*

cross\_val\_score(rf,X\_train\_scaled\_selected,y\_train\_resampled,cv**=**4,n\_jobs**=-**1)

Out[374]:

array([0.79467681, 0.84790875, 0.79467681, 0.82889734])

***Inference:***

1. The cross-validation scores are all reasonably high and consistent (ranging from ~0.79 to ~0.84).
2. This suggests the model is performing well on unseen subsets of the training data.
3. Since there's no large drop or fluctuation, and the values are not overly perfect (i.e., not all 1.0), there's no strong evidence of overfitting here.

**7.5 Hyperparameter Tuning [10 Marks]**

Enhance the performance of the random forest model by systematically exploring and selecting optimal hyperparameter values using grid search.

**7.5.1 Use grid search to find the best hyperparameter values [2 Marks]**

In [375]:

*# Use grid search to find the best hyperparamter values*

classifier\_rf **=** RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1)

*# Create the parameter grid based on the results of random search*

params **=** {

'max\_depth': [1, 2, 5, 8, 10],

'min\_samples\_leaf': [5, 10, 20, 50, 100],

'max\_features': [3,4,5,8,10,12,15,18],

'n\_estimators': [10, 30, 50, 100, 200]

}

*# Instantiate the grid search model*

grid\_search **=** GridSearchCV(estimator**=**classifier\_rf, param\_grid**=**params,

cv**=**4, n\_jobs**=-**1, verbose**=**1, scoring **=** "accuracy")

grid\_search**.**fit(X\_train\_scaled\_selected,y\_train\_resampled)

*# Best Hyperparameters*

rf\_best **=** grid\_search**.**best\_estimator\_

Fitting 4 folds for each of 1000 candidates, totalling 4000 fits

In [376]:

rf\_best

Out[376]:

RandomForestClassifier(max\_depth=10, max\_features=15, min\_samples\_leaf=5,

n\_estimators=200, n\_jobs=-1, random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

**7.5.2 Build a random forest model based on hyperparameter tuning results [1 Mark]**

In [377]:

*# Building random forest model based on results of hyperparameter tuning*

classifier\_rf\_best**=**rf\_best**.**fit(X\_train\_scaled\_selected,y\_train\_resampled)

classifier\_rf\_best

Out[377]:

RandomForestClassifier(max\_depth=10, max\_features=15, min\_samples\_leaf=5,

n\_estimators=200, n\_jobs=-1, random\_state=42)

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.  
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

**7.5.3 Make predictions on training data [1 Mark]**

In [378]:

*# Make predictions on training data*

y\_train\_pred\_final**=**classifier\_rf\_best**.**predict(X\_train\_scaled\_selected)

**7.5.4 Check accuracy of Random Forest Model [1 Mark]**

In [379]:

*# Check accuracy of the model*

metrics**.**accuracy\_score(y\_train\_pred\_final,y\_train\_resampled)

Out[379]:

0.9325095057034221

**7.5.5 Create confusion matrix [1 Mark]**

In [380]:

*# Create the confusion matrix*

rf\_cm\_final**=**metrics**.**confusion\_matrix(y\_train\_pred\_final,y\_train\_resampled)

rf\_cm\_final

Out[380]:

array([[479, 24],

[ 47, 502]], dtype=int64)

**7.5.6 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [381]:

*# Create variables for true positive, true negative, false positive and false negative*

TN, FP, FN, TP **=**rf\_cm\_final**.**ravel()

**7.5.7 Calculate sensitivity, specificity, precision, recall and F1-score of the model [3 Marks]**

In [382]:

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.91

Specificity: 0.95

Precision: 0.95

Recall: 0.91

F1: 0.93

**8. Prediction and Model Evaluation [20 marks]**

Use the model from the previous step to make predictions on the validation data with the optimal cutoff. Then evaluate the model's performance using metrics such as accuracy, sensitivity, specificity, precision, and recall.

**8.1 Make predictions over validation data using logistic regression model [10 marks]**

**8.1.1 Select relevant features for validation data and add constant [1 Mark]**

In [423]:

*# Select the relevant features for validation data*

X\_val\_logreg**=**X\_val\_scaled[selected\_cols\_logreg]

X\_val\_logreg**.**shape

Out[423]:

(300, 36)

In [424]:

X\_val\_logreg **=** sm**.**add\_constant(X\_val\_logreg, has\_constant**=**'add')

X\_val\_logreg**.**shape

Out[424]:

(300, 37)

**8.1.2 Make predictions over validation data [1 Mark]**

In [425]:

*# Make predictions on the validation data and store it in the variable 'y\_validation\_pred'*

y\_val\_pred\_logreg**=**logistic\_model**.**predict(X\_val\_logreg)

**8.1.3 Create DataFrame with actual values and predicted values for validation data [2 Marks]**

In [429]:

*# Create DataFrame with actual values and predicted values for validation data*

val\_pred\_df\_logreg**=**pd**.**DataFrame({'Actual Fraud':y\_val\_resampled,'Predicted Prob':y\_val\_pred\_logreg})

**8.1.4 Make final prediction based on cutoff value [1 Mark]**

In [430]:

*# Make final predictions on the validation data using the optimal cutoff*

val\_pred\_df\_logreg['Predcited Fraud']**=**val\_pred\_df\_logreg['Predicted Prob']**.**apply(**lambda** x: 1 **if** x**>**0.4 **else** 0)

In [431]:

val\_pred\_df\_logreg

Out[431]:

|  | **Actual Fraud** | **Predicted Prob** | **Predcited Fraud** |
| --- | --- | --- | --- |
| **0** | 0 | 0.385028 | 0 |
| **1** | 0 | 0.087611 | 0 |
| **2** | 0 | 0.205827 | 0 |
| **3** | 0 | 0.285100 | 0 |
| **4** | 0 | 0.430565 | 1 |
| **...** | ... | ... | ... |
| **295** | 0 | 0.669686 | 1 |
| **296** | 1 | 0.124980 | 0 |
| **297** | 0 | 0.265776 | 0 |
| **298** | 0 | 0.031182 | 0 |
| **299** | 0 | 0.060287 | 0 |

300 rows × 3 columns

**8.1.5 Check the accuracy of logistic regression model on validation data [1 Mark]**

In [432]:

*# Check the accuracy*

metrics**.**accuracy\_score(val\_pred\_df\_logreg['Actual Fraud'],val\_pred\_df\_logreg['Predcited Fraud'])

Out[432]:

0.7033333333333334

**8.1.6 Create confusion matrix [1 Mark]**

In [439]:

*# Create the confusion matrix*

logreg\_cm\_final**=**metrics**.**confusion\_matrix(val\_pred\_df\_logreg['Actual Fraud'],val\_pred\_df\_logreg['Predcited Fraud'])

logreg\_cm\_final

Out[439]:

array([[158, 68],

[ 21, 53]], dtype=int64)

**8.1.7 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [440]:

*#Create variables for true positive, true negative, false positive and false negative*

TN,FP,FN,TP**=**logreg\_cm\_final**.**ravel()

**8.1.8 Calculate sensitivity, specificity, precision, recall and f1 score of the model [2 Marks]**

In [441]:

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.72

Specificity: 0.70

Precision: 0.44

Recall: 0.72

F1: 0.54

**8.2 Make predictions over validation data using random forest model [10 marks]**

**8.2.1 Select the important features and make predictions over validation data [2 Marks]**

In [385]:

*# Select the relevant features for validation data*

X\_val\_scaled\_selected**=**X\_val\_scaled[selected\_cols]

*# Make predictions on the validation data*

y\_val\_pred**=**classifier\_rf\_best**.**predict(X\_val\_scaled\_selected)

**8.2.2 Check accuracy of random forest model [1 Mark]**

In [386]:

*# Check accuracy*

metrics**.**accuracy\_score(y\_val\_pred,y\_val\_resampled)

Out[386]:

0.78

**8.2.3 Create confusion matrix [1 Mark]**

In [387]:

*# Create the confusion matrix*

rf\_val\_cm**=**metrics**.**confusion\_matrix(y\_val\_pred,y\_val\_resampled)

rf\_val\_cm

Out[387]:

array([[187, 27],

[ 39, 47]], dtype=int64)

**8.2.4 Create variables for true positive, true negative, false positive and false negative [1 Mark]**

In [388]:

*# Create variables for true positive, true negative, false positive and false negative*

TN, FP, FN, TP **=**rf\_val\_cm**.**ravel()

**8.2.5 Calculate sensitivity, specificity, precision, recall and F1-score of the model [5 Marks]**

In [389]:

*# Calculate the sensitivity*

sensitivity**=**TP**/**(TP**+**FN)

*# Calculate the specificity*

specificity**=**TN**/**(TN**+**FP)

*# Calculate Precision*

precision**=**TP**/**(TP**+**FP)

*# Calculate Recall*

recall**=**TP**/**(TP**+**FN)

*# Calculate F1 Score*

f1 **=** 2 **\*** (precision **\*** recall) **/** (precision **+** recall)

print(f"Sensitivity: {sensitivity:.2f}")

print(f"Specificity: {specificity:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1: {f1:.2f}")

Sensitivity: 0.55

Specificity: 0.87

Precision: 0.64

Recall: 0.55

F1: 0.59

**Evaluation and Conclusion**

Write the conclusion.

**Model Comparison on Validation Set:**

| **Metric** | **Logistic Regression** | **Random Forest (Tuned)** |
| --- | --- | --- |
| Accuracy | 0.70 | 0.78 |
| Recall (Sensitivity) | 0.72 | 0.55 |
| Precision | 0.44 | 0.64 |
| Specificity | 0.70 | 0.87 |

| F1 Score | 0.54 | 0.59

1. Random Forest (tuned) clearly outperforms logistic regression across all metrics.
2. Recall and F1 Score are especially critical for fraud detection.
3. Recall: Ensures more fraudulent claims are caught early.
4. F1 Score: Balances the trade-off between catching fraud and minimizing false positives.

**Training vs Validation Performance:**

1. Logistic Regression shows overfitting, with a strong drop in validation performance.
2. Random Forest (without tuning) performs perfectly on training data - a sign of severe overfitting.
3. Tuned Random Forest generalizes well to unseen data, making it the most reliable model.

In [ ]: