1) Insurance Risk & Claim Dataset

TC. Antony

Project Observation

This project focuses on building a fraud risk prediction app for the insurance industry, using historical policyholder details, claims history, and fraud indicators. By analyzing key variables like customer demographics, policy type, claim history, and risk scores,

to predict the likelihood of fraudulent claims. This helps insurers identify suspicious activity, minimize losses,

Index:

4. Anomalies Detected

i) Isolation Forest

ii) Auto Encoder

5. Predictions

1] Risk Classification - Prediction

I] Feature Selection

II] Filter Method / Wrapper Method

III] Model Training - [Random Forest Classifier]

IV] Balance or Unbalanced check

V] Cross Validation [Stratified K-fold]

VI] Hyper Parameter Tuning
* GridSearchCV

VII] Deep Learning - Neural Network
* Tensorflow keras

VIII] Model Saved (.h5)

IX] Future Prediction

2] Claim Prediction

I] Feature Selection

II] Model Training

III] Balance Check

IV] AUC - ROC Curve

V] Cross Validation - Stratified K-Folds

3] Fraud Detection

I] Feature Selection

II] Model Training

III] Future Prediction

1 - EDA - [11 - 36 Slide]
2 - SQL - [38 - 50 Slide]
3 - Preprocessing - [52 - 69 Slide]
4 - Anomalies Detected - [71 - 88 Slide]
5 - 1st - Risk Classification - [89 - 113 Slide]
6 - 2nd - Claim Prediction - [114 - 131 Slide]
7 - 3rd - Fraud Detection - [132 - 147 Slide]
8 - 4th - Claim Amount Model Training [149 - 174 Slide]
9 - 4th- Claim Amount Prediction [175 - 180 Slide]

8 - Streamlit Application - [181 - 195 Slide]
9 - Conclusion - [196 Slide]

Connect Dataframe

```
[2]  1 import pandas as pd
```

```
[3]  1 df1=pd.read_csv('/content/df1-insurance_risk_claim_dataset.csv')
     2
```

```
     1 df1.head()
```

e	Gender	Policy_Type	Annual_Income	Vehicle_Age_Property_Age	Claim_History	Fraudulent_Claim	Premium_Amount	Claim_Amount	Risk_Score
8	Male	Health	153479.09	7	1	0	3814.12	33834.97	High
3	Other	Property	25720.88	2	1	0	2774.10	1326.80	Low

```
]   1 df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Policy_ID                  1000 non-null   object
 1   Customer_Age               1000 non-null   int64
 2   Gender                     1000 non-null   object
 3   Policy_Type                1000 non-null   object
 4   Annual_Income              1000 non-null   float64
 5   Vehicle_Age_Property_Age   1000 non-null   int64
 6   Claim_History              1000 non-null   int64
 7   Fraudulent_Claim           1000 non-null   int64
 8   Premium_Amount             1000 non-null   float64
 9   Claim_Amount               1000 non-null   float64
 10  Risk_Score                 1000 non-null   object
dtypes: float64(3), int64(4), object(4)
memory usage: 86.1+ KB
```

```
1 df1.isnull().sum()
```

	0
Policy_ID	0
Customer_Age	0
Gender	0
Policy_Type	0
Annual_Income	0
Vehicle_Age_Property_Age	0
Claim_History	0
Fraudulent_Claim	0
Premium_Amount	0
Claim_Amount	0
Risk_Score	0

Missing Values ->

Show Categorical and Numeric Columns

```python
# Identify categorical and numerical columns
categorical_columns = df1.select_dtypes(include=['object']).columns.tolist()
numerical_columns = df1.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical Columns:", categorical_columns)
print("Numerical Columns:", numerical_columns)
```

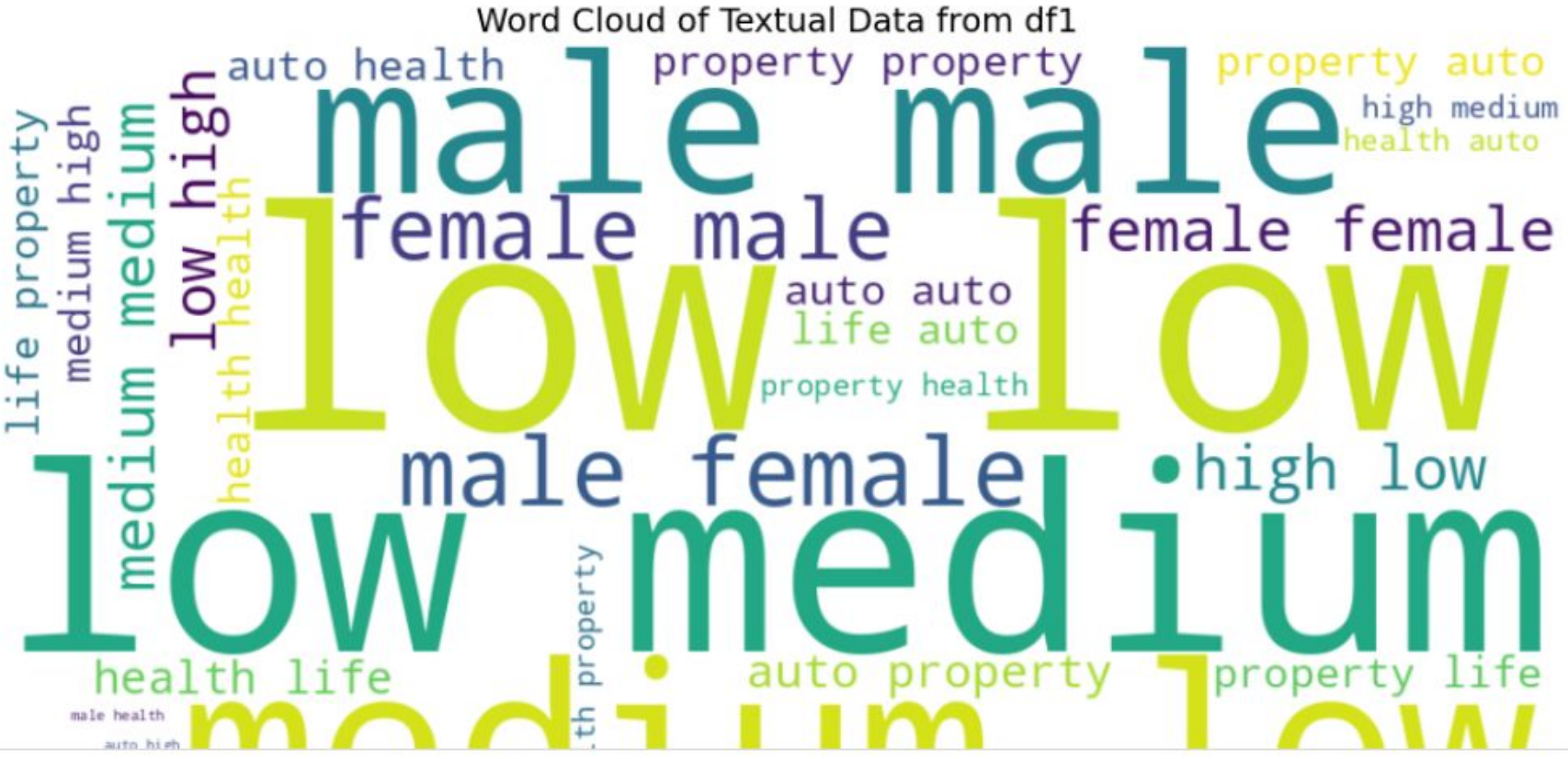
```
Categorical Columns: ['Policy_ID', 'Gender', 'Policy_Type', 'Risk_Score']
Numerical Columns: ['Customer_Age', 'Annual_Income', 'Vehicle_Age_Property_Age', 'Claim_History', 'Fraudulent_Claim', 'Premium_Amount', 'Cla
```

```python
df1.nunique()
```

	0
Policy_ID	1000
Customer_Age	63
Gender	3
Policy_Type	4
Annual_Income	1000
Vehicle_Age_Property_Age	31
Claim_History	6
Fraudulent_Claim	2
Premium_Amount	1000
Claim_Amount	1000
Risk_Score	3

<- Find Unique Values

Using Word Cloud Just Understand the dataset

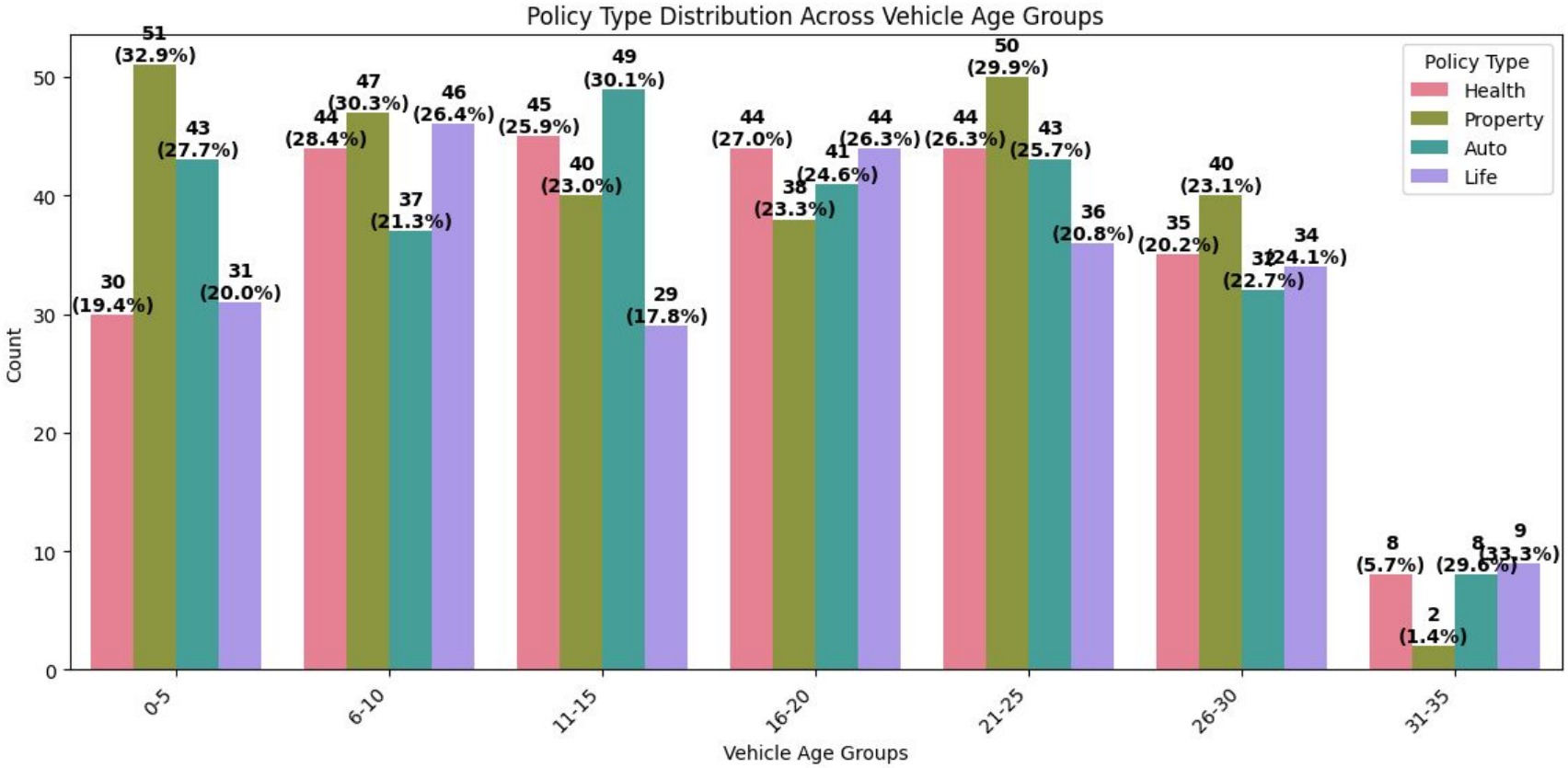


This dataset contains historical policyholder data, claim details, and fraud labels.

Data Analysis Overview

	Primary Column	Cross-Check with	Why?	Best Charts
0	Policy_Type	Vehicle_Age, Property_Age	Older vehicles may have different policy types	Countplot (Bar Chart)
1	Customer_Age	Claim_History, Fraudulent_Claim	Older customers may have different claim patterns	Boxplot, Violin Plot
2	Annual_Income	Premium_Amount, Claim_Amount	Higher-income groups may have different premiu...	Scatterplot, Boxplot
3	Risk_Score	Claim_History, Fraudulent_Claim	High-risk scores may correlate with fraudulent...	Heatmap, Boxplot
4	Vehicle_Age, Property_Age	Claim_History, Fraudulent_Claim	Older vehicles may have higher fraud risks	Bar Chart, Heatmap
5	Customer_Age_Bin	Premium_Amount, Claim_Amount	Different age groups might have varying premiu...	Boxplot, Line Chart
6	Annual_Income_Bin	Risk_Score	Lower or higher income may affect risk levels	Violin Plot, Bar Chart

1) Plicy_Type vs Vehicle_Age_Property_Age



Claim Frequency by Policy Type:

Health and Auto: The high prevalence of these policies suggests potentially higher claim frequencies in these categories across all age groups.

Property: Significant representation indicates a substantial number of property-related claims, particularly for vehicles aged 6-20 years.

Life: The low count of life policies (especially in younger age groups) suggests fewer claims in this category. Claim Trends by Vehicle Age:

Older Vehicles (26-35 years):

A dramatic drop in all policy types suggests fewer vehicles in this age group are being insured or that claims are less frequent.

The relatively higher percentage of life insurance claims in this group may indicate a correlation between vehicle age and life insurance needs.

Mid-Range Vehicles (11-25 years):

The balanced distribution of policy types implies a consistent level of claims across different categories.

Newer Vehicles (0-10 years):

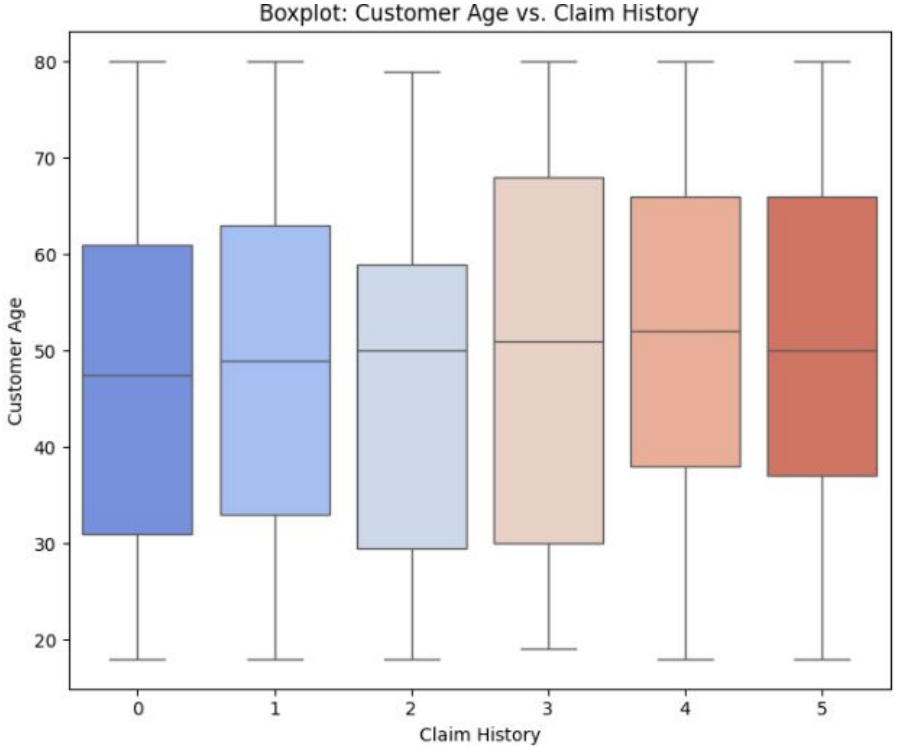
The dominance of Health and Auto policies suggests that claims related to these categories are more common in newer vehicles. Claim Patterns and Risk:

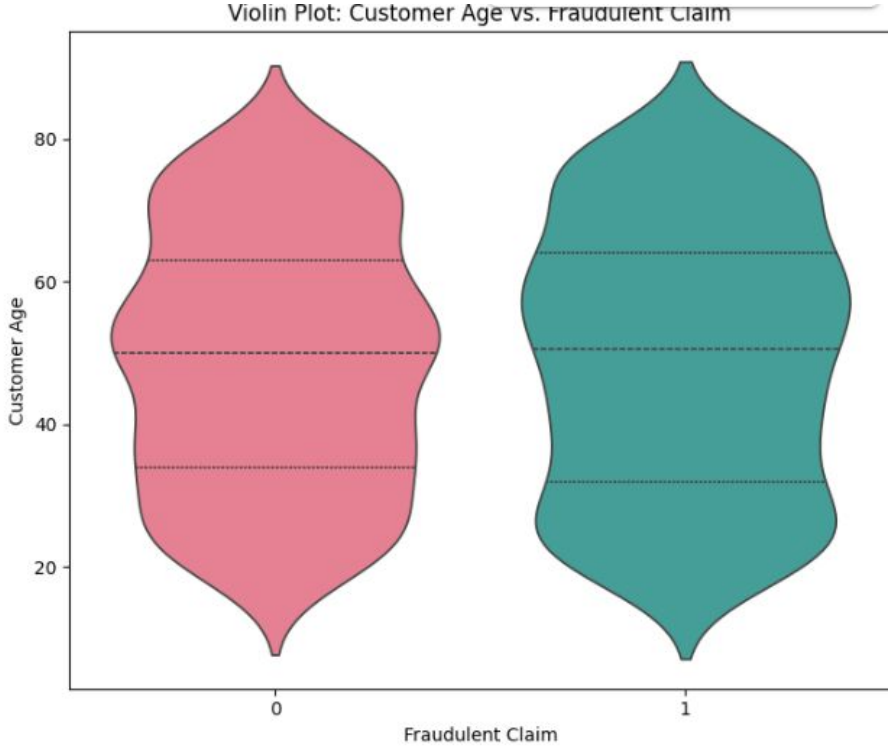
The data suggests potential risk factors associated with different vehicle ages and policy types.

High claim frequencies in Health and Auto policies for newer vehicles may indicate higher risk of accidents or health-related issues.

The decrease in claims for older vehicles could be due to lower usage, better maintenance, or different risk profiles.

2) Customer_Age vs Claim_History, Fraudulent_Claim





1 = Fraud, 0 = Genuine (Target Variable)

Insights:

Boxplot: Customer Age vs. Claim History (Relationship to Fraud/Genuine)

Claim History and Age (Potential Link to Fraud):

The boxplot shows that as the claim history increases **(more claims), the median age** of customers tends to be higher.

This is significant in the context of fraud because it suggests **older customers with multiple claims** might be a **higher-risk group for fraudulent activity.**

While not a direct indicator of fraud, a **higher claim history could be a red flag** for further investigation, especially in older age groups. **Age Distribution for No Claims (Genuine)**:

Customers with no claims (0) have a younger age distribution. This aligns with the expectation **that younger individuals might have fewer opportunities or reasons to file claims**.

This could be a characteristic of **genuine claims, as younger customers might be less likely** to have a history of prior incidents.

Violin Plot: Customer Age vs. Fraudulent Claim (Target Variable Direct)

Age Distribution for Fraudulent (1) vs. Genuine (0):

The violin plot directly shows the age distribution for fraudulent (1) and genuine (0) claims.

The median age for fraudulent claims is slightly higher than for genuine claims.

The fraudulent claim distribution is somewhat narrower, indicating a more concentrated age range (potentially older) for fraudulent activity. Age as a Predictor of Fraud:

The slight difference in age distribution suggests that age could be a contributing factor in predicting fraudulent claims.

Older customers are slightly more represented in the fraudulent claim category, but the difference is not dramatic. This highlights that age alone is not a definitive predictor, but it's a factor to consider. Recommendations (Target Variable Focused):

Enhanced **Fraud Detection for Older Customers with Multiple Claims:**

Implement a multi-layered fraud detection strategy, specifically targeting older customers with a history of multiple claims. Use predictive models that incorporate age, claim history, and other relevant features to identify potentially fraudulent claims. Anomaly Detection for Claim Patterns:

Develop algorithms to identify unusual claim patterns for older customers. Investigate claims where **older customers have a sudden increase in claim** frequency or severity.

Targeted Investigations:

Prioritize investigations for claims filed by older customers with a history of multiple claims, especially when anomalies are detected. Use data analytics to identify specific patterns or red flags that warrant closer scrutiny.

Customer Education and Communication (Fraud Prevention):

Implement **targeted educational campaigns to raise awareness about insurance fraud, especially among older customers**. Emphasize the consequences of fraudulent activity and the importance of accurate claim reporting.

Policy Adjustments Based on Risk:

Consider implementing risk-based pricing strategies that reflect the increased risk associated with older customers and those with a history of multiple claims.

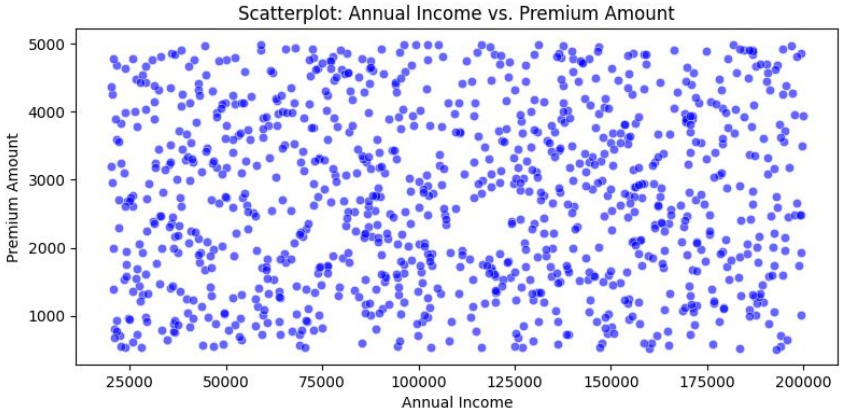
Evaluate the need for **additional policy restrictions or requirements for high-risk customers**.

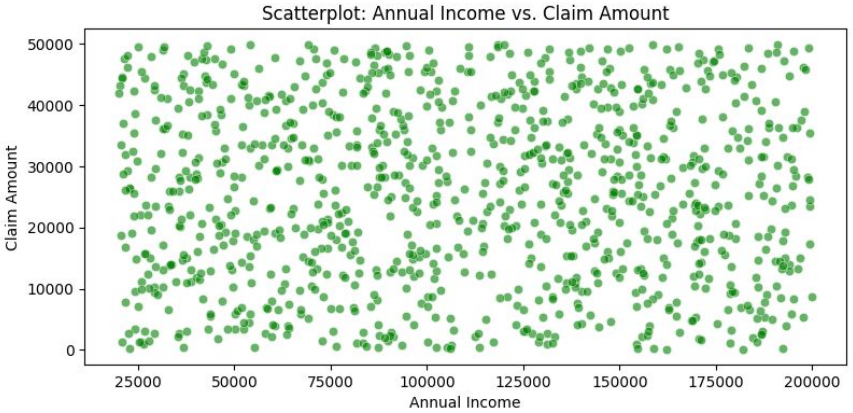
Data Enrichment and Feature Engineering:

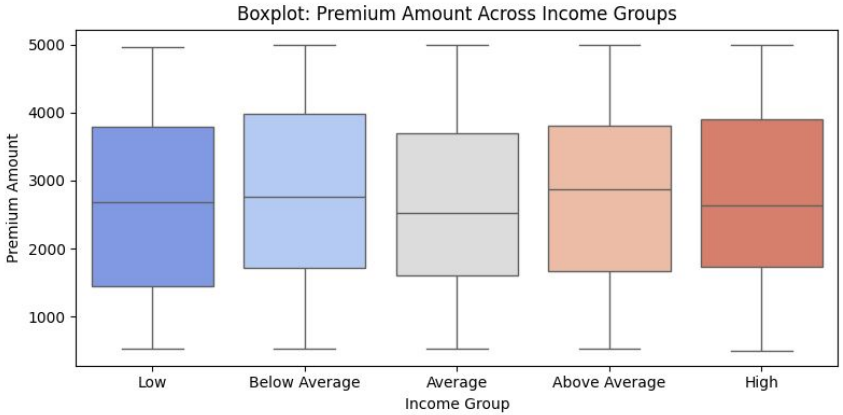
Enrich the data with additional demographic, behavioral, and transactional information to improve fraud prediction accuracy. Develop new features that capture the nuances of fraudulent behavior, such as claim patterns, policy changes, and customer interactions. Machine Learning for Fraud Prediction:

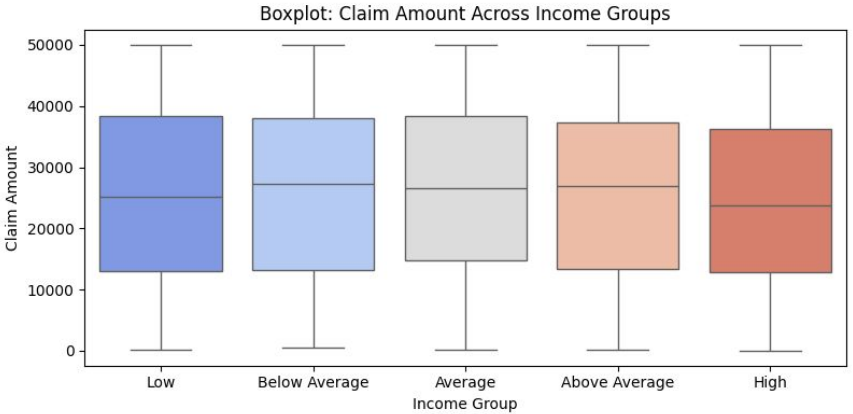
Train machine learning models to predict fraudulent claims based on various features, including age, claim history, and other relevant factors. Continuously monitor and update the models to adapt to evolving fraud patterns.

3) Annual_Income vs Premium_Amount, and Claim_Amount









Insights:

Chart Analysis: Annual Income vs. Premium Amount & Claim Amount

Scatterplots: Annual Income vs. Premium Amount & Claim Amount

Premium Amount and Income:

The scatterplot of Annual Income vs. Premium Amount shows a **weak positive correlation or a lack of a strong linear relationship**. While there's a general spread of data points, it's **not clear that higher incomes directly translate to significantly higher premiums**. There's considerable variation in **premium amounts across all income levels**.

Claim Amount and Income:

The scatterplot of **Annual Income vs. Claim Amount also shows a weak or negligible correlation**. **Similar to premiums, claim amounts are spread across all income levels**, suggesting that **income alone doesn't strongly predict claim amounts**. There are **high claim amounts even for lower-income individuals and vice versa**.

2) **Boxplots:** Premium Amount & Claim Amount Across Income Groups

Premium Amount Distribution Across Income Groups:

The boxplot of **Premium Amount across income groups (Low, Below Average, Average, Above Average, High)** reveals subtle differences in median premium amounts.

The median premium tends to increase slightly with higher income groups, but the differences are not drastic. The interquartile ranges (boxes) overlap significantly, indicating substantial variability within each income group.

Claim Amount Distribution Across Income Groups:

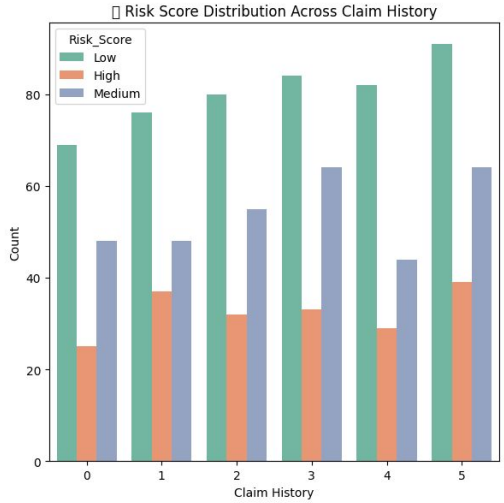
The boxplot of Claim Amount across income groups shows a similar pattern to the premium plot. There's no significant difference in the median claim amounts across income groups. The overlap of interquartile ranges is substantial, reinforcing the idea that **income isn't a strong predictor of claim amounts.**

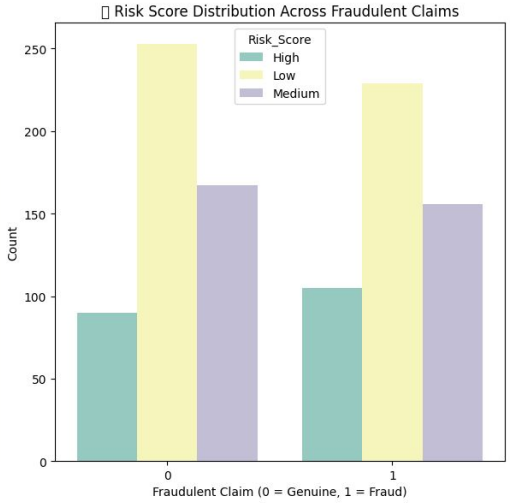
4) Risk_Score vs Claim_History, Fraudulent_Claim

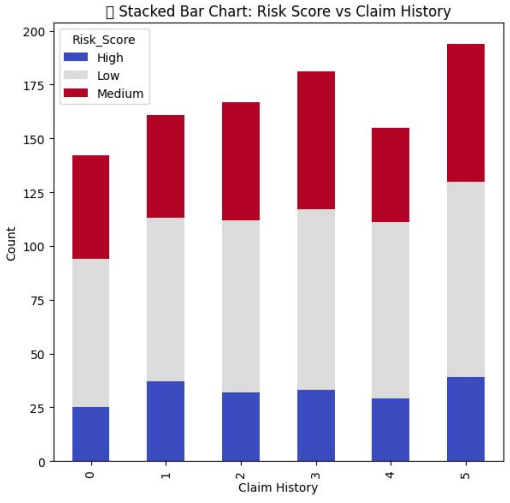
```
df1['Risk_Score'].value_counts()
```

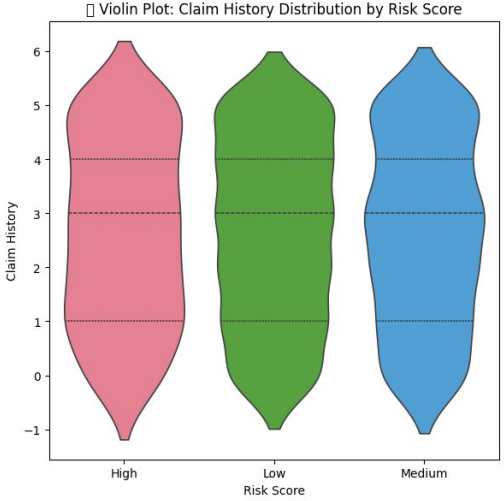
Risk_Score	count
Low	482
Medium	323
High	195

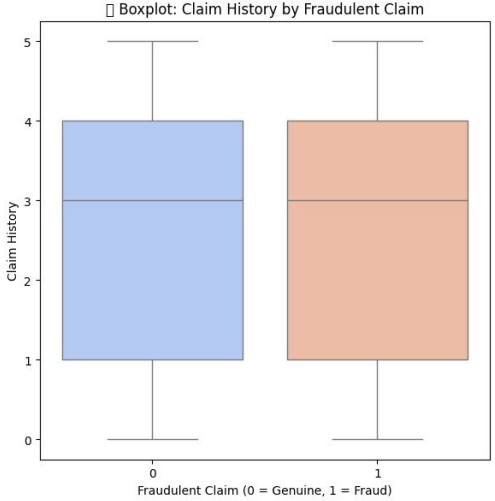
dtype: int64


Risk Score Distribution Across Claim History
Risk_Score
Low
High
Medium
Count
Claim History


Risk Score Distribution Across Fraudulent Claims
Risk_Score
High
Low
Medium
Count
Fraudulent Claim (0 = Genuine, 1 = Fraud)


Stacked Bar Chart: Risk Score vs Claim History
Risk_Score
High
Low
Medium
Count
Claim History


Violin Plot: Claim History Distribution by Risk Score
Claim History
High
Low
Medium
Risk Score


Boxplot: Claim History by Fraudulent Claim
Claim History
Fraudulent Claim (0 = Genuine, 1 = Fraud)

1. Risk Score Distribution Across Claim History (Top Left)

Observation:

The distribution of risk scores (Low, Medium, High) varies across different claim histories (0-5).

As **claim history increases, the proportion of "High" risk scores tends to increase.**

Conversely, the proportion of "Low" risk scores decreases with increasing claim history.

Insight:

Customers with **more claims are more likely to have higher risk scores.** This suggests that prior claims are a significant factor in determining risk.

2. Risk Score Distribution Across Fraudulent Claims (Top Middle)

Observation:

There's a noticeable difference in risk score distribution between **genuine (0) and fraudulent (1) claims.**

Fraudulent claims tend to have a higher proportion of "High" risk scores compared to genuine claims.

Genuine claims have a higher proportion of "Low" and "Medium" risk scores.

Insight:

High-risk customers are more likely to file fraudulent claims. **Risk score is a valuable predictor of potential fraud**.

3. Stacked Bar Chart: Risk Score vs. Claim History (Top Right)

Observation:

This chart visually reinforces the trend seen in the first chart. As claim history increases, the stacked bars show a growing dominance of "High" risk scores.

Insight:

The stacked format provides a clear view of how risk score composition changes with claim history.

4. Violin Plot: Claim History Distribution by Risk Score (Bottom Left)

Observation:

The distribution of claim history varies across different risk scores. "High" risk scores have a wider distribution of claim history, including more instances of higher claim counts. "Low" risk scores have a narrower distribution, skewed towards lower claim counts.

Insight:

High-risk customers exhibit a wider range of claim behaviors. This might indicate that **high risk customers are more likely to have a higher variability in their claims.**

Overall Insights and Recommendations for Insurance Claims:

Risk Score as a Predictor:

Risk score is a strong indicator of both claim frequency and fraud risk. Utilize risk scores in predictive models for claims processing and fraud detection.

Claim History as a Risk Factor:

Customers with a higher claim history are more likely to be high-risk and potentially fraudulent. Implement stricter scrutiny for customers with multiple claims. Fraud Detection Strategies:

Focus fraud detection efforts on high-risk customers, especially those with a history of claims. Use machine learning models that incorporate risk score and claim history to identify suspicious claims. Policy Adjustments:

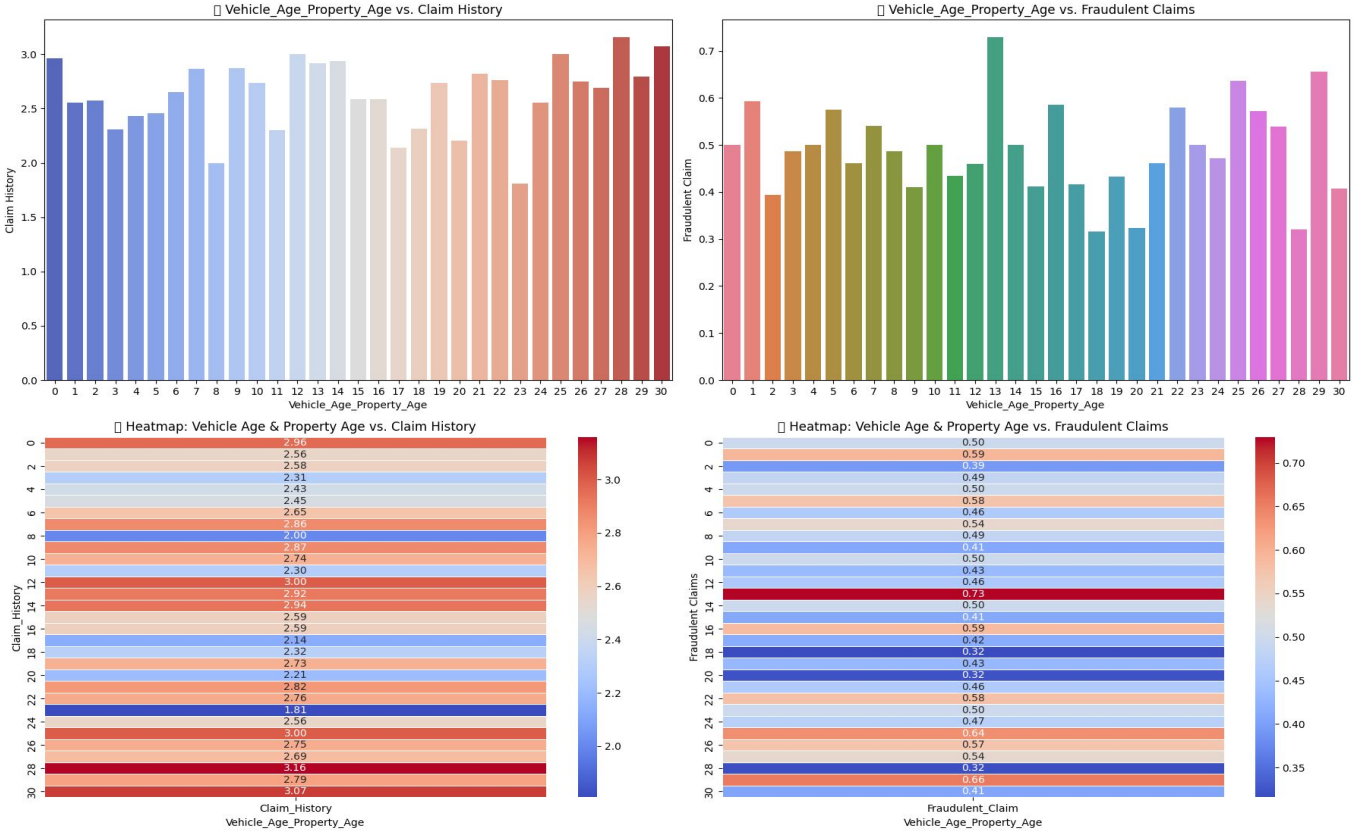
Consider adjusting premiums based on risk scores and claim history. Implement stricter policy terms for high-risk customers. Data-Driven Decisions:

Continuously analyze claim data to refine risk assessment and fraud detection strategies. Use visualizations to monitor trends and identify potential issues. Customer Behavior Analysis:

Investigate the factors that contribute to high-risk behavior and fraudulent activity. Consider implementing customer education programs to reduce fraudulent claims. Further Investigation:

It would be very useful to know what factors are used to create the risk score. Investigate the correlation of other data points with the risk score.

5) Vehicle_Age_Property_Age vs Claim_History, Fraudulent_Claim



Color Interpretation (Coolwarm Palette)

🔵 Blue Shades → Lower Fraudulent Claims

🔴 Red Shades → Higher Fraudulent Claims

What it Means:

If a section is deep red, it suggests more fraudulent claims in those age groups. Blue sections indicate less fraud activity in those vehicle/property ages.

(I) #**Vehicle Age & Property Age vs. Claim History (Left Side):**

Bar Chart:

There's a noticeable trend of **increasing claim history as the combined "Vehicle Age Property Age" increases**, especially in the **range of 20-30**. This suggests that older vehicles in older properties tend to have more claims. The claim history distribution appears relatively uniform in the **lower ranges (0-20)**, with a more pronounced increase towards the higher values.

Heatmap:

The heatmap reinforces the bar chart's findings. **The darker red shades, indicating higher claim history**, are concentrated in the higher "Vehicle Age Property Age" values. The heatmap provides a granular view of the relationship, allowing you to see specific claim history values for each combined age.

(II) Vehicle Age & Property Age vs. Fraudulent Claims (Right Side):

Bar Chart:

The distribution of fraudulent claims is less consistent than the overall claim history. While there are fluctuations, there isn't a clear, linear trend. **Some specific "Vehicle Age Property Age" values exhibit higher instances of fraudulent claims, but it's not a uniform increase with age.**

Heatmap:

The heatmap shows a more nuanced picture of fraudulent claims. There are pockets of higher fraudulent claim rates scattered across the "Vehicle Age Property Age" spectrum. The heatmap highlights that fraudulent claims aren't solely tied to older vehicles/properties; they can occur across various age combinations

Recommendations:

Targeted Risk Assessment:

Insurance providers should implement more granular risk assessment models that consider the combined "Vehicle Age Property Age." The visualizations clearly show that this combination is a significant predictor of claim history. For **older vehicle/property combinations (20-30 range), consider higher premiums or more stringent underwriting guidelines due to the increased likelihood of claims**.

Fraud Detection Enhancement:

Use the insights to tailor marketing and customer engagement strategies. For **customers with older vehicle/property combinations, provide proactive information about maintenance and risk mitigation.** For areas with higher fraudulent claim rates, consider targeted awareness campaigns to educate customers about fraud prevention.

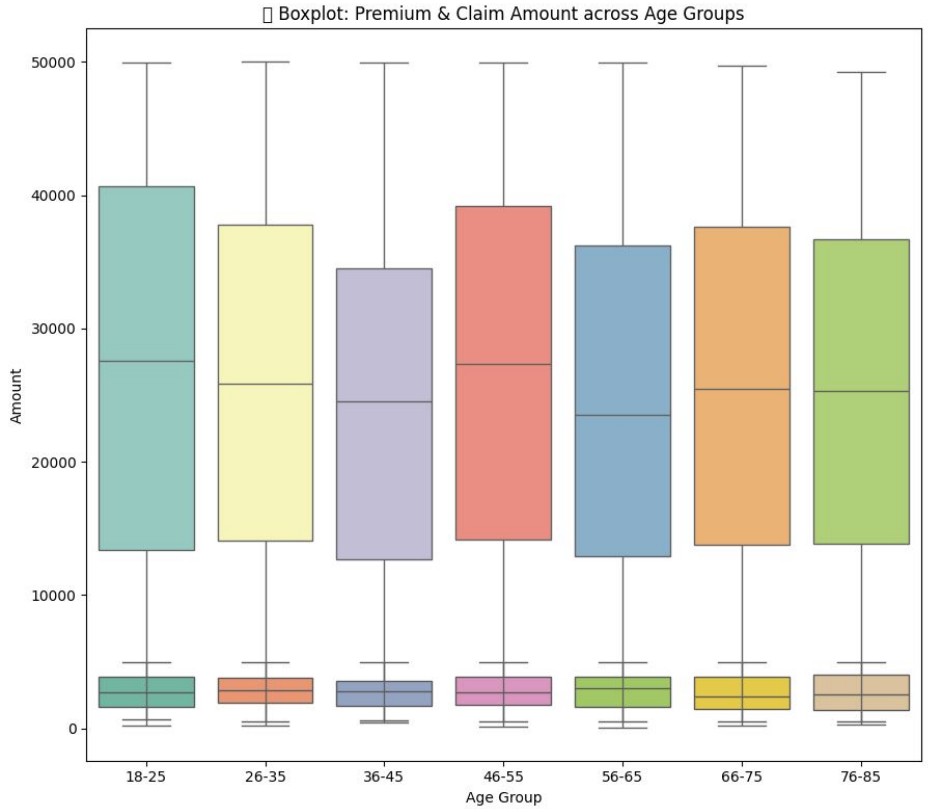
Further Investigation:

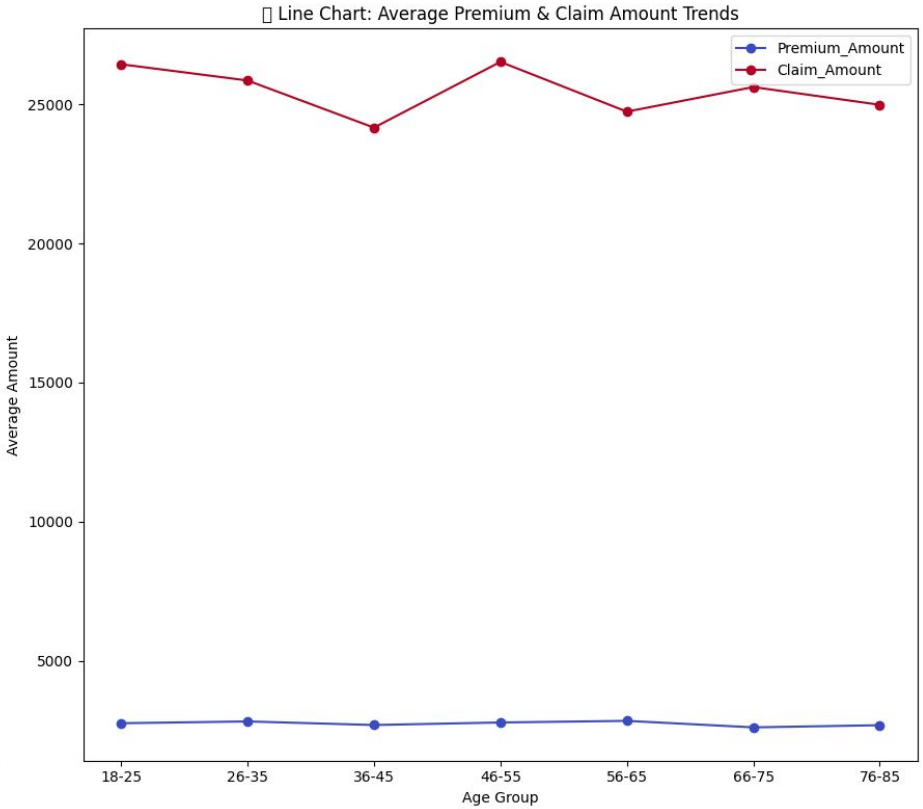
Investigate the factors contributing to the spikes in fraudulent claims at specific "Vehicle Age Property Age" values. Are there specific types of vehicles or properties involved? Are there regional or demographic factors at play?

Explore other variables that might influence claim history and fraudulent claims, such as driving history, location, and policy type. Refine "Vehicle Age Property Age" Metric:

Consider whether the simple addition of vehicle age and property age is the most effective metric. Explore other ways to combine these variables or incorporate additional factors.

6) Customer_Age_Bin vs Premium_Amount, Claim_Amount





Boxplot: Premium & Claim Amount Across Age Groups (Left Chart)

Premium Amount (Lower Boxplots):

Premiums are relatively low and consistent across all age groups. There's slight variation, but no dramatic differences. **The spread (interquartile range) of premiums is narrow, indicating less variability in the premium amounts within each age group**.

Claim Amount (Upper Boxplots):

Claim amounts show a much wider distribution and greater variability compared to premiums.

The median claim amount appears to be **highest in the 46-55 and 56-65 age groups.**

The **18-25 and 26-35 age groups have relatively lower median claim amounts.** There are significant outliers (very high claim amounts) across all age groups, suggesting some high-value claims occur regardless of age.

Line Chart: Average Premium & Claim Amount Trends (Right Chart)

Average Premium Amount (Blue Line):

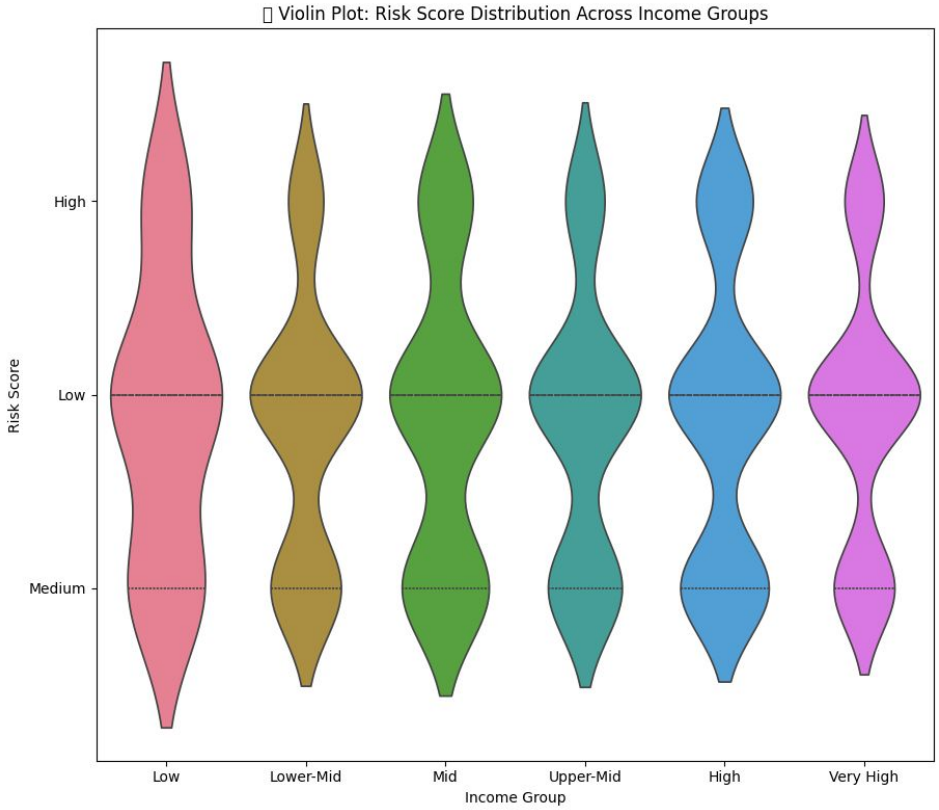
The average premium amount remains relatively flat across all age groups, confirming the consistency seen in the boxplots.

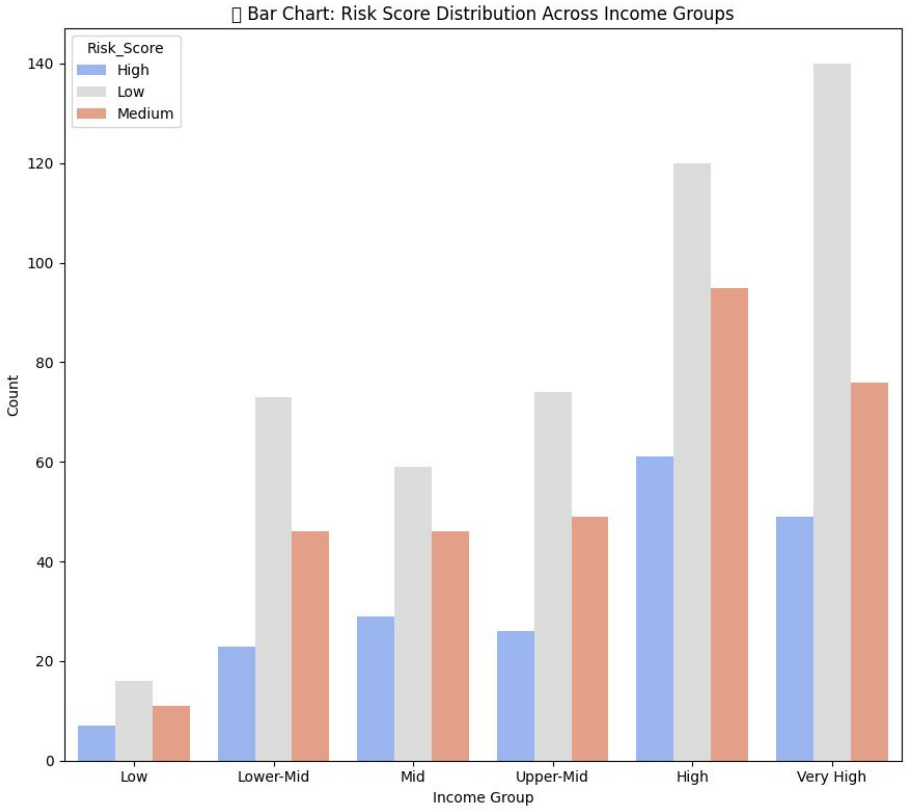
Average Claim Amount (Red Line):

The **average claim amount fluctuates more than the premium**. There are **peaks in the 46-55 and 56-65 age groups**, aligning with the boxplot observations.

It **dips in the 26-35 age range**. The average claim amount is higher than the average premium amount in all age ranges.

7) Annual_Income_Bin vs Risk_Score





```
bins = [0, 25000, 50000, 75000, 100000, 150000, 200000] # Define income bins

labels = ["Low", "Lower-Mid", "Mid", "Upper-Mid", "High", "Very High"]
```

Insights:

Violin Plot (Left Chart):

Risk Score Distribution:

The violin plots illustrate the distribution of risk scores within each income group.

The width of the violins indicates the density of data points at different risk score levels.

We can see that the distribution of risk scores varies significantly across income groups.

Median Risk Score:

The dashed lines within the violins represent the median risk score for each income group.

Visually, the **median risk score appears to increase as income increases**.

Shape and Spread:

The shapes of the violins differ, suggesting variations in the spread and skewness of risk scores across income groups.

The lower income groups show a wider spread toward the lower risk scores./ The higher income groups show a wider spread toward the higher risk scores.

Risk Score Categories:

The horizontal lines designating “High” “Low” and “Medium” risk scores allow for quick visual analysis of where each income group is centered.

2)Bar Chart (Right Chart):

As **income increases, the count of individuals with "High" risk scores generally increases**.

Conversely, the count of individuals with "Low" risk scores generally decreases. The “Medium” risk scores fluctuate through the income brackets.

Recomendation:-

Targeted Risk Management Strategies:

The clear correlation between income and risk score distribution suggests that risk management strategies should be tailored to different income groups.

For higher-income groups, focus on mitigating high-risk factors and providing appropriate risk management tools.

For lower-income groups, emphasize strategies to improve risk scores and provide access to resources that can help them mitigate risks.

Policy and Program Development:

Use these insights to inform the development of policies and programs that address the specific needs and risks of different income groups.

Consider implementing programs that provide incentives for individuals to improve their risk scores.

SQL

[2]

```python
import sqlite3
import pandas as pd

# File name of the uploaded CSV
csv_filename = "/content/df1-insurance_risk_claim_dataset.csv"  # Replace this with the uploaded file name

# Load CSV into a pandas DataFrame
df = pd.read_csv(csv_filename)

# Connect to SQLite database (or create a new one)
conn = sqlite3.connect("example.db")
cursor = conn.cursor()

# Write DataFrame to SQLite table
table_name = "df1"  # Specify your table name
df.to_sql(table_name, conn, if_exists="replace", index=False)

print(f"Table '{table_name}' created in SQLite database.")

```

```
Table 'df1' created in SQLite database.
```

1) Retrieve all records

Write a query to select all records from the dataset.

```
SELECT * from df1;
```

Code cell output actions

Policy_ID	Customer_Age	Gender	Policy_Type	Annual_Income	Vehicle_Age_Property_Age	Claim_History	Fraudulent_Claim	Premium_Amount	Claim_Amount	Risk_Score
bdd640fb-0667-4ad1-9c80-7fa3b1799d	58	Male	Health	153479.09	7	1	0	3814.12	33834.97	High
23b8c1e9-3924-46de-beb1-)046685257	23	Other	Property	25720.88	2	1	0	2774.10	1326.80	Low
bd9c66b3-ad3c-4d6d-9a3d-a7bc8960a9	59	Other	Property	59679.31	18	2	0	3914.63	7982.97	Low

2) Filter by Age

Write a query to find all policies where the Customer_Age is greater than 60.

```
SELECT
Policy_ID,Customer_Age,Gender,Policy_Type,Annual_Income,Vehicle_Age_Property_Age
,Claim_History,Fraudulent_Claim,Premium_Amount,Claim_Amount,Risk_Score from df1
where Customer_Age>=60 order by Customer_Age ASC;
```

.icy_ID	Customer_Age	Gender	Policy_Type	Annual_Income	Vehicle_Age_Property_Age	Claim_History	Fraudulent_Claim	Premium_Amount	Claim_Amount	Risk_Score
fed2c43-56-46dc-8f54-7b5b2bc	60	Female	Auto	127107.42	30	0	1	3015.15	32174.66	Low
lc7472a-4e-4a13-829c-a02eaec	60	Female	Auto	113194.84	16	1	1	1829.94	2728.46	High
a432f0a-laa-49f0-80b6-e53790a	60	Other	Auto	41499.74	22	1	0	1237.25	47650.81	Medium
ec259dc-95-497c-a76a-50755d9	60	Female	Auto	149830.98	20	3	1	1667.42	20928.28	Low
327f1bc-84-478f-	60	Other	Property	46757.80	26	4	0	552.40	46718.57	Medium

3) Count Policies by Gender

Write a query to count the number of policies for each Gender

```
SELECT Gender,count(Policy_ID) from df1 GROUP BY Gender;
```

	Gender	count(Policy_ID)
0	Female	332
1	Male	328
2	Other	340

4) Average Premium Amount

Write a query to calculate the average Premium_Amount.

```
SELECT Gender,AVG(Premium_Amount) from df1 GROUP BY Gender;
```

	Gender	AVG(Premium_Amount)
0	Female	2709.441355
1	Male	2675.588323
2	Other	2838.893618

```
SELECT Customer_Age,Gender,AVG(Premium_Amount) as Avg_Premium from df1 GROUP BY Customer_Age;
```

	Customer_Age	Gender	Avg_Premium
0	18	Other	3087.086429
1	19	Other	2732.448000
2	20	Female	3091.127647
3	21	Male	3226.958750

5) Identify High-Risk Policies

Write a query to retrieve policies where Risk_Score is greater than 4

```
SELECT Policy_Type,Risk_Score as Max_Risk from df1 WHERE Risk_Score>='High'
ORDER by Max_Risk LIMIT 10;
```

	Policy_Type	Max_Risk
0	Health	High
1	Auto	High
2	Health	High
3	Health	High
4	Health	High
5	Life	High
6	Health	High
7	Auto	High
8	Life	High
9	Auto	High

Identify Medium Risk Policies

```
SELECT Policy_Type,Risk_Score as Max_Risk from df1
WHERE Risk_Score>='Medium'  ORDER by Max_Risk LIMIT
10;
```

	Policy_Type	Max_Risk
0	Property	Medium
1	Life	Medium
2	Health	Medium
3	Health	Medium
4	Auto	Medium
5	Auto	Medium
6	Life	Medium
7	Property	Medium
8	Auto	Medium
9	Life	Medium

6) Find Total Claim Amount

Write a query to sum up the total Claim_Amount for all policies

```
SELECT Policy_Type,sum(Claim_Amount) as Total_Claims FROM df1 Group by Policy_Type Order by Total_Claims DESC
```

	Policy_Type	Total_Claims
0	Health	6772183.55
1	Auto	6605255.14
2	Property	6407299.53
3	Life	5745462.02

7) Detect Fraudulent Claims

Write a query to list policies where Fraudulent_Claim is greater than 1

1 = Fraud, 0 = Genuine

```
SELECT Policy_Type,COUNT(Fraudulent_Claim) as Fraud_Claims FROM df1 Where
Fraudulent_Claim=1 Group by Policy_Type Order by Fraud_Claims DESC
```

	Policy_Type	Fraud_Claims
0	Property	127
1	Auto	127
2	Health	119
3	Life	117

```
SELECT Policy_Type,COUNT(Fraudulent_Claim) as Genune_Claims
FROM df1 Where Fraudulent_Claim=0 Group by Policy_Type ORDER by
Genune_Claims DESC
```

	Policy_Type	Genune_Claims
0	Property	141
1	Health	131
2	Auto	126
3	Life	112

Both Fraud & Genuine Claims Almost Equal

8) Group Policies by Policy Type

```
SELECT Policy_Type,count(Policy_ID) as Total_Polices FROM
df1 Group by Policy_Type Order by Total_Polices DESC
```

	Policy_Type	Total_Polices
0	Property	268
1	Auto	253
2	Health	250
3	Life	229

9) Compare Premium and Claim Amounts

Write a query to find policies where the Claim_Amount is greater than the Premium_Amount.

1 = Fraud, 0 = Genuine

```
SELECT
Policy_ID,Policy_Type,Customer_Age,Claim_Amount,Premium_Amount,Fraudulent_Cl
aim FROM df1 where Claim_Amount>Premium_Amount Order by Claim_Amount DESC
```

	Policy_ID	Policy_Type	Customer_Age	Claim_Amount	Premium_Amount	Fraudulent_Claim
0	9d6b9b62-6b53-4a19-a902-b73fcf39f648	Property	32	49997.71	2227.82	0
1	4223623b-cc3e-4dde-9ad5-cf06364d7c87	Property	52	49965.63	1839.69	0
2	e916da57-f248-43ab-977a-8a5f6ffe33b3	Property	18	49947.58	1575.78	1
3	5d678bb1-945e-42e4-888a-93ec70d9c9f8	Property	58	49936.41	2945.61	1
4	6fdc0bad-5e36-4127-8ca1-b45c1fdd980a	Property	44	49936.17	1306.84	1
...	...	...	...	...	...	...

10) Find Customers with No Claim History

Write a query to retrieve all policies where Claim_History is equal to 0

```
SELECT Policy_ID,Policy_Type,Gender FROM df1 Where Claim_History=0
```

	Policy_ID	Policy_Type	Gender
0	972a8469-1641-4f82-8b9d-2434e465e150	Auto	Male
1	47378190-96da-4dac-b2ff-5d2a386ecbe0	Auto	Other
2	1a2a73ed-562b-4f79-8374-59eef50bea63	Life	Other
3	759cde66-bacf-43d0-8b1f-9163ce9ff57f	Health	Other
4	ec1b8ca1-f91e-4d4c-9ff4-9b7889463e85	Health	Male
...	...	...	...
137	fc737d92-11ab-4d11-a4eb-8000ef40d162	Health	Male
138	083f64c3-1ce3-42f8-83f7-25feb99b2b5a	Property	Other
139	9d6b9b62-6b53-4a19-a902-b73fcf39f648	Property	Male
140	df495037-b40a-4181-ac09-bdb799086e47	Life	Other
141	d18183d1-ac2b-4cfc-85c5-1060991121e7	Health	Male

142 rows × 3 columns

Upto 142 Datasets are no Claim History out of 1000 Datasets

Preprocessing

1) Missing Values

```
1 df1.isnull().sum()
```

	0
Policy_ID	0
Customer_Age	0
Gender	0
Policy_Type	0
Annual_Income	0
Vehicle_Age_Property_Age	0
Claim_History	0
Premium_Amount	0
Claim_Amount	0
Risk_Score	0
Fraudulent_Claim	0

dtype: int64

No Missing Values in this Dataset

If Missing Values have this Datasets.

Step 1:

check % of Missing Values if more than 70% Missing Values -> Delete that Columns

Step 2:

Check this Columns have Normal Distribution and No Outliers - Use Mean Imputation

Else use Median Imputation

Step 3:

Check that Columns Have Categorical Columns Use Mode

Step 4:

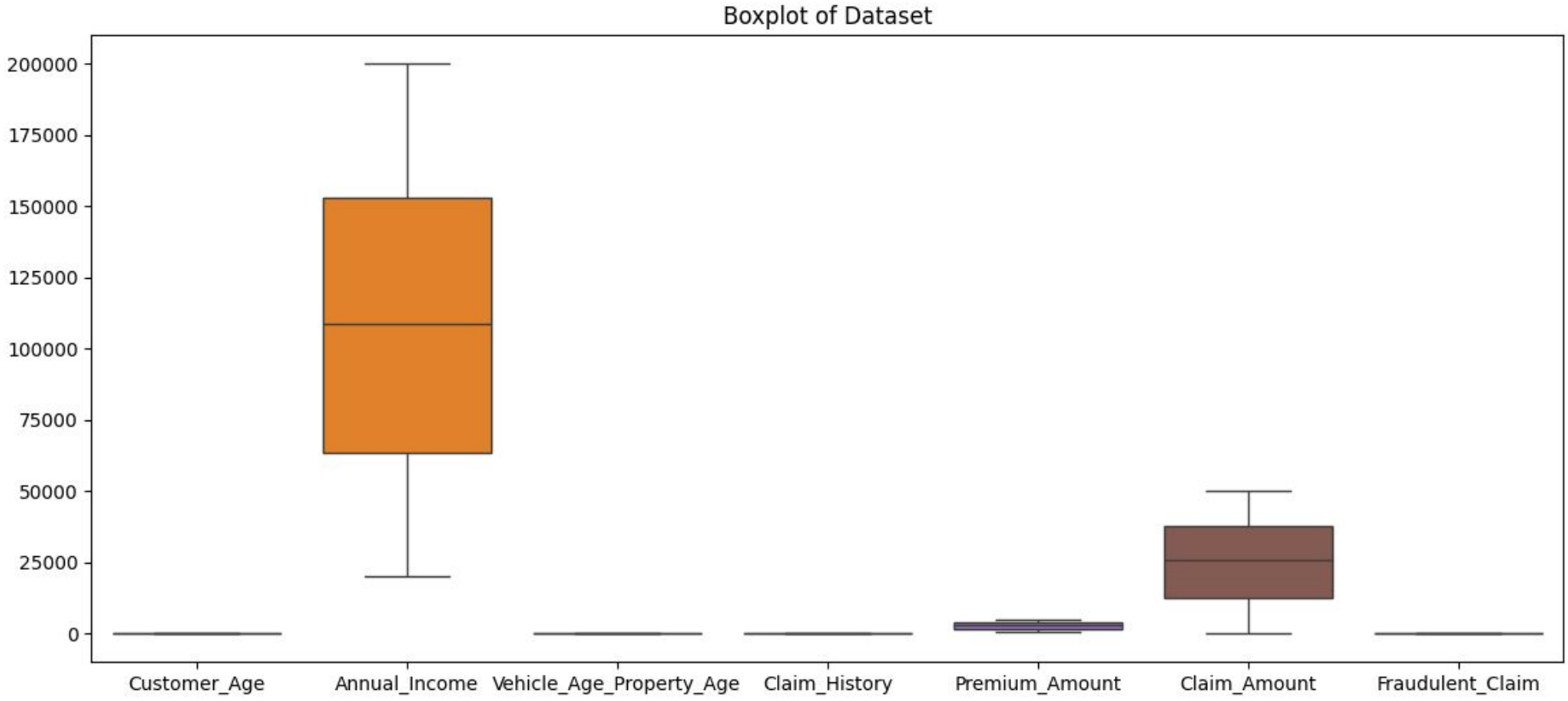
If That Dataset have Times series Use Forward fill or Backward Fill

Step 5:

If that Missing Values was Voluntarily Happen Based On Confidential else any other Situation

Client or SME or BA Give us Custom values fill us Cutom method to fill Missing Values

2) Outliers


Boxplot of Dataset
200000
175000
150000
125000
100000
75000
50000
25000
0
Customer_Age
Annual_Income
Vehicle_Age_Property_Age
Claim_History
Premium_Amount
Claim_Amount
Fraudulent_Claim

```
outliers = {}

for col in df1.select_dtypes(include=['number']).columns:
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers[col] = df1[(df1[col] < lower_bound) | (df1[col] > upper_bound)][col].tolist()

# Print outliers for each column
for col, out in outliers.items():
    print(f"Outliers in {col}: {out}")

```

```
Outliers in Customer_Age: []
Outliers in Annual_Income: []
Outliers in Vehicle_Age_Property_Age: []
Outliers in Claim_History: []
Outliers in Premium_Amount: []
Outliers in Claim_Amount: []
Outliers in Fraudulent_Claim: []
```

No Outlier Found this Dataset

If Outlier Found in this Datasets:-

Step 1: Understand the Outliers

Check Data Entry Errors: Typos or incorrect values? Fix them if possible.

Assess Context: Are these extreme values realistic or expected in your data domain?

Step 2: Choose an Outlier Treatment Approach:

1) Removal Methods:

Delete Outliers: If they're definitely errors or irrelevant.

df_clean = df1[~((df1['A'] < lower_bound) | (df1['A'] > upper_bound))]

2) Imputation Methods:

Replace with Mean/Median: Works well when outliers aren't extreme.

median = df1['A'].median() df1['A'] = np.where((df1['A'] < lower_bound) | (df1['A'] > upper_bound), median, df1['A'])

Use Mode: For categorical data with outliers

3) Transformation Methods:

Log Transformation: Reduces effect of right-skewed outliers.

df1['A_log'] = np.log1p(df1['A'])

Square Root or Box-Cox: For data with different types of skew.

4) Capping or Clipping

Winsorization: Limit extreme values to percentiles.

from scipy.stats.mstats import winsorize df1['A_winsorized'] = winsorize(df1['A'], limits=[0.05, 0.05]) # Caps at 5th & 95th percentile

Cap at Boundaries: Replace outliers with IQR bounds.

df1['A'] = np.clip(df1['A'], lower_bound, upper_bound)

3) Encoding

1) One-Hot Encoding for [Policy_Type, Gender].

```
[ ]  1 from sklearn.preprocessing import OneHotEncoder
     2
```

```
[ ]  1
     2 # OneHotEncoder with updated argument
     3 onehot_encoder = OneHotEncoder(sparse_output=False)
     4 encoded = onehot_encoder.fit_transform(df1[['Policy_Type', 'Gender']])
     5
     6 # Convert back to DataFrame for readability and cast to int
     7 encoded_df = pd.DataFrame(encoded, columns=onehot_encoder.get_feature_names_out(['Policy_Type', 'Gender'])).astype(int)
     8
     9 print(encoded_df)
    10
```

Policy_Type_Auto	Policy_Type_Health	Policy_Type_Life	Policy_Type_Property	Gender_Female	Gender_Male	Gender_Other
0	1	0	0	0	1	0

2) Risk_Score -> Ordinal Encoding

```
from sklearn.preprocessing import OrdinalEncoder
```

```


# Define the order of categories from low to high risk
risk_categories = ['Low', 'Medium', 'High']

# Create the OrdinalEncoder with category order
lbl_encoder = OrdinalEncoder(categories=[risk_categories])

# Fit and transform the Risk_Score column
df1['Risk_Score_Encoded'] = lbl_encoder.fit_transform(df1[['Risk_Score']]).astype(int)

print(df1)

```

Risk_Score_Encoded
0
0
1
1
2

```
df1_encoded=pd.concat([df1,encoded_df],axis=1)
```

```
df1_encoded.nunique()
```

	0
Policy_ID	1000
Customer_Age	63
Annual_Income	1000
Vehicle_Age_Property_Age	31
Claim_History	6
Premium_Amount	1000
Claim_Amount	999
Fraudulent_Claim	2
Risk_Score_Encoded	3
Policy_Type_Auto	2
Policy_Type_Health	2
Policy_Type_Life	2
Policy_Type_Property	2
Gender_Female	2
Gender_Male	2
Gender_Other	2

Save Encoded Files

1) One Hot Encoder:

```
[ ]  1 import pickle
```

```
[ ]  1 with open('onehot_encoder.pkl', 'wb') as file:
     2     pickle.dump(onehot_encoder, file)
```

2) Label Encoder

```
[ ]  1 with open('lbl_encoder.pkl', 'wb') as file:
     2     pickle.dump(lbl_encoder, file)
```

in Nominal or One Hot Encoding Means -> 0 is No and 1 is Yes

4) Feature Scaling

Normalize

Annual_Income, Claim_Amount, Premium_Amount

Normalize done by reason of minimizing the value for better model training and reduce time - values ranges (0 to 1)

```
[ ]  1 from sklearn.preprocessing import MinMaxScaler
```

```
[ ]  1
     2 # Create MinMaxScaler
     3 scaler = MinMaxScaler()
     4
     5 # Fit and transform the columns
     6 df1_encoded[['Annual_Income', 'Premium_Amount', 'Claim_Amount']] = scaler.fit_transform(
     7     df1_encoded[['Annual_Income', 'Premium_Amount', 'Claim_Amount']]
     8 )
     9
```

Annual_Income	Premium_Amount	Claim_Amount
0.741442	0.139545	0.102080
0.590267	0.093670	0.232579
0.715891	0.419729	0.449681
0.697997	0.155499	0.958985
0.379538	0.604068	0.808517

5) Feature Selection or Feature Engineering

i) Filter Methods: (Fast, Statistical)

What it does: Selects features based on their statistical relationship with the target variable — independently of any model.

Examples:

1) Correlation Coefficient:

(e.g., Pearson, Spearman) Measures linear or rank-based correlation.

2) Chi-Square Test:

For categorical features, checks if feature and target are independent.

3) Mutual Information:

Measures how much knowing one variable reduces uncertainty about the other.

4) ANOVA (Analysis of Variance):

Tests whether feature means differ across target groups.

Pros:

Fast and computationally efficient.

Works well for very high-dimensional datasets.

Model-agnostic — can be used with any ML algorithm.

Cons:

Ignores feature interactions.

Can miss features that only work well in combination with others.

When to use:

When you have a lot of features and want a quick way to narrow them down.

When you're doing preliminary analysis and need insights on feature importance.

ii) Wrapper Methods: (Slow, Model-Based)

What it does:

Uses the performance of a specific model to evaluate feature subsets — essentially trying different combinations and picking the best.

Examples:

Forward Selection:

Starts with no features, adds them one by one based on model performance.

Backward Elimination:

Starts with all features, removes them one by one.

Recursive Feature Elimination (RFE):

Trains a model and removes the least important feature iteratively.

Pros:

Takes feature interaction into account. Often results in higher model performance since it's directly tied to the target metric.

Cons:

Computationally expensive — especially with many features.

Results are model-dependent — changing the model might change the selected features.

When to use:

When you have a moderate number of features and computational resources.

When model performance is your top priority.

So which one's the best?

If you want speed and scalability → **Filter methods**

If you want accuracy and performance → **Wrapper methods**

If you want a balance → **Embedded methods** (like feature importance from tree-based models or Lasso regression)

Anomalies Detected

Isolation Forest & Autoencoder

- **Isolation Forest** → Detects a *broader set* of potential fraud, including more low and medium-risk frauds.
- **Autoencoder** → Tends to focus on the *most unusual and extreme cases*, often the very high-risk frauds.

- Some Insurance Company Filters all Frauds (Isolation Forest)
- But Some Insurance Company focus to increase a Customer so they filter only very High Risk Frauds Only (Autoencoder - Neural Network)

1. Isolation Forest:

a) An unsupervised machine learning algorithm designed for anomaly detection.

b) Works by isolating observations by randomly selecting features and splitting them.

c) Anomalies get isolated quicker because they differ from normal points.

Find **Contamination rate: 0.3 - 30%** Based Fraudulent claim Dataset and 30% Fraud in Dataset So its take Contamination rate as 30% - 0.3

1) (known fraud cases):

Calculate the actual proportion of anomalies:

```
[ ]  1 known_anomalies = df1[df1['Fraudulent_Claim'] == 1]  # Or however fraud is labeled
     2 contamination_rate = len(known_anomalies) / len(df1)
     3 print(f"Contamination rate: {contamination_rate}")
```

```
Contamination rate: 0.3
```

Contamination rate: 0.3 - 30%

So i select Contaminated Parameter Value based on Fraudulent_Claim so get Accurate Value

"Contamination=0.3" means that, in a given context, there is a **30% level of unwanted or foreign material present**, indicating a relatively low level of contamination; essentially, **only 30 out of every 100 samples or units** are considered contaminated

Percentage interpretation: "0.3" is the same as 30% when expressed as a percentage.

Contamination is mean as Affected

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest



# Features for anomaly detection
features = [
    'Customer_Age', 'Annual_Income', 'Vehicle_Age_Property_Age', 'Claim_History',
    'Fraudulent_Claim', 'Premium_Amount', 'Claim_Amount', 'Risk_Score_Encoded',
    'Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life', 'Policy_Type_Property',
    'Gender_Female', 'Gender_Male', 'Gender_Other'
]

# Apply Isolation Forest
iso_forest = IsolationForest(contamination=0.3, random_state=42) #Find Contamination using by Fraudulent_Claim
df1['Anomaly_IsoForest'] = iso_forest.fit_predict(df1[features])

```

Sklearn.ensemble method used for IsolationForest

Ensemble Method -

1)Machine Learning Models that Combines Multiple Model to improve prediction Accuracy

2) Reduce Bias & Variance

2. AutoEncoder

```python
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# Features for anomaly detection
features = [
    'Customer_Age', 'Annual_Income', 'Vehicle_Age_Property_Age', 'Claim_History',
    'Fraudulent_Claim', 'Premium_Amount', 'Claim_Amount', 'Risk_Score_Encoded',
    'Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life', 'Policy_Type_Property',
    'Gender_Female', 'Gender_Male', 'Gender_Other'
]
```

```python
# Build Autoencoder
autoencoder = Sequential([
    Dense(8, activation='relu', input_shape=(len(features),)), #hidden Layer 1,2,3 used -> relu activation functio
    Dense(4, activation='relu'),                               #if losses happen Activation activate backward prop
    Dense(8, activation='relu'),                               #loss means mse
    Dense(len(features), activation='linear')
])

autoencoder.compile(optimizer='adam', loss='mse')

# Train Autoencoder
autoencoder.fit(df1[features], df1[features], epochs=20, batch_size=32, validation_split=0.2, verbose=1)

# Get reconstruction errors
reconstructions = autoencoder.predict(df1[features])
mse = np.mean(np.square(reconstructions - df1[features]), axis=1)

# Set anomaly threshold
threshold = np.percentile(mse, 95)  #mean square error above (95 percentile) Threshold consider as a Anomaly

# Tag anomalies from Autoencoder
df1['Anomaly_Autoencoder'] = (mse > threshold).astype(int)

# Output results
print(df1[['Policy_ID', 'Anomaly_IsoForest', 'Anomaly_Autoencoder']])
```

1. **What does `Dense` mean?**

   - `Dense` is a fully connected layer in a neural network — meaning each neuron in the layer is connected to **every neuron** in the previous and next layers. It's a standard building block in Keras for creating neural networks.

Keras For Creating a Neural Network

```
Dense(8, activation='relu', input_shape=(len(features),))
Dense(4, activation='relu')
Dense(8, activation='relu')
Dense(len(features), activation='linear')
```

First hidden layer: 8 neurons

Second hidden layer: 4 neurons

Third hidden layer: 8 neurons

ReLU activation function work?

`ReLU` stands for **Rectified Linear Unit**.

- If the input is positive → it passes through unchanged.
- If the input is negative → it becomes zero.

It's fast and effective, especially for deep networks.

- **Backpropagation**:
  - During training, the network makes predictions and calculates **loss** (like `mse` — mean squared error).
  - Backpropagation is the process of sending the error back through the network so the model can adjust its weights and improve.
  - `ReLU` helps keep backpropagation efficient and prevents problems like the **vanishing gradient** (where small updates stop happening).

Adam optimizer

`adam` stands for **Adaptive Moment Estimation** — it's one of the most popular optimizers for training deep learning models.

- It combines two techniques:
  - **Momentum**: Helps the model move faster in the right direction by using past gradients.
  - **Adaptive Learning Rates**: It adjusts the learning rate for each parameter so the model learns efficiently.
- Adam often converges faster and more smoothly compared to older optimizers like **SGD** (Stochastic Gradient Descent).

1) Isolation Forest

```
1 df1['Anomaly_IsoForest'].value_counts()
```

Anomaly_IsoForest	count
1	700
-1	300

dtype: int64

Detect All Fraud - Include High, Low and Medium Fraud Also

2) Auto Encoder

```
1 df1['Anomaly_Autoencoder'].value_counts()
```

Anomaly_Autoencoder	count
0	950
1	50

dtype: int64

Detect : High Risk Fraud Only

Isolation Forest

contamination=0.49: This sets the proportion of data we expect to be anomalies.

random_state=42: Ensures reproducibility of results.

fit_predict():

Fits the Isolation Forest model on the features. Predicts if each row is an anomaly:

-1 = Anomaly (suspicious data point). 1 = Normal data point.

Why use Isolation Forest?

It works well on high-dimensional data.

It's unsupervised — we don't need labeled data.

It's fast and effective for finding outliers.

Autoencoder

Sequential() — a simple, stack-based neural network.

Dense() — fully connected layers:

8 neurons → 4 neurons → 8 neurons — compressing data into a smaller representation and then reconstructing it.

relu — activation function that helps the network learn non-linear patterns.

Final layer: Same size as the input (features) because the Autoencoder tries to **reconstruct the original data**.

adam — an efficient and widely used optimization algorithm.

mse (Mean Squared Error) — measures the reconstruction error. Anomalies will likely have higher errors because they don't fit the learned pattern

Training the Autoencoder to learn normal behavior by trying to reconstruct the input.

validation_split=0.1: Reserves 10% of data for validation.

i am selecting validation_split=0.1 take 20% of data for testing

reconstructions: The output from the Autoencoder — it tries to replicate the input.

mse: Measures how far off the reconstruction is from the actual data row by row.

Higher MSE → More likely to be an anomaly because the model couldn't "understand" that pattern well

threshold = np.percentile(mse, 95)

95th percentile of the MSE values — the top 5% most "unusual" reconstructions.

Rows with MSE above this threshold will be labeled as anomalies.

df1['Anomaly_Autoencoder'] = (mse > threshold).astype(int)

1 = Anomaly (reconstruction error above the threshold).

0 = Normal data point.

Policy_ID — for reference.

Anomaly_IsoForest — anomalies detected by Isolation Forest.

Anomaly_Autoencoder — anomalies detected by the Autoencoder

Why use both methods?

Isolation Forest is fast and rule-based — great for broad anomaly detection.

Autoencoder captures complex, non-linear patterns in the data.

Together, they give you a more complete view of potential anomalies.

How to tune contamination and threshold?

*If you know the actual percentage of fraud, set contamination to that.

*For Autoencoder, experiment with different percentiles (90%, 95%, 99%) to see what works best.

```
df1[['Policy_ID', 'Anomaly_IsoForest', 'Anomaly_Autoencoder']]
```

	Policy_ID	Anomaly_IsoForest	Anomaly_Autoencoder
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	1	0
1	23b8c1e9-3924-46de-beb1-3b9046685257	-1	0
2	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	1	0
3	972a8469-1641-4f82-8b9d-2434e465e150	-1	0
4	17fc695a-07a0-4a6e-8822-e8f36c031199	1	0
...	...	...	...
995	fe54ee4b-80ff-4173-a3df-ecd70e731dd7	1	0

Anomaly_IsoForest

-1 = Anomaly (suspicious data point).

1 = Normal data point.

Anomaly_Autoencoder

1 = Anomaly (reconstruction error above the threshold).

0 = Normal data point.

```
[ ]  1 df1['Anomaly_IsoForest'].value_counts()
```

Anomaly_IsoForest	count
1	700
-1	300

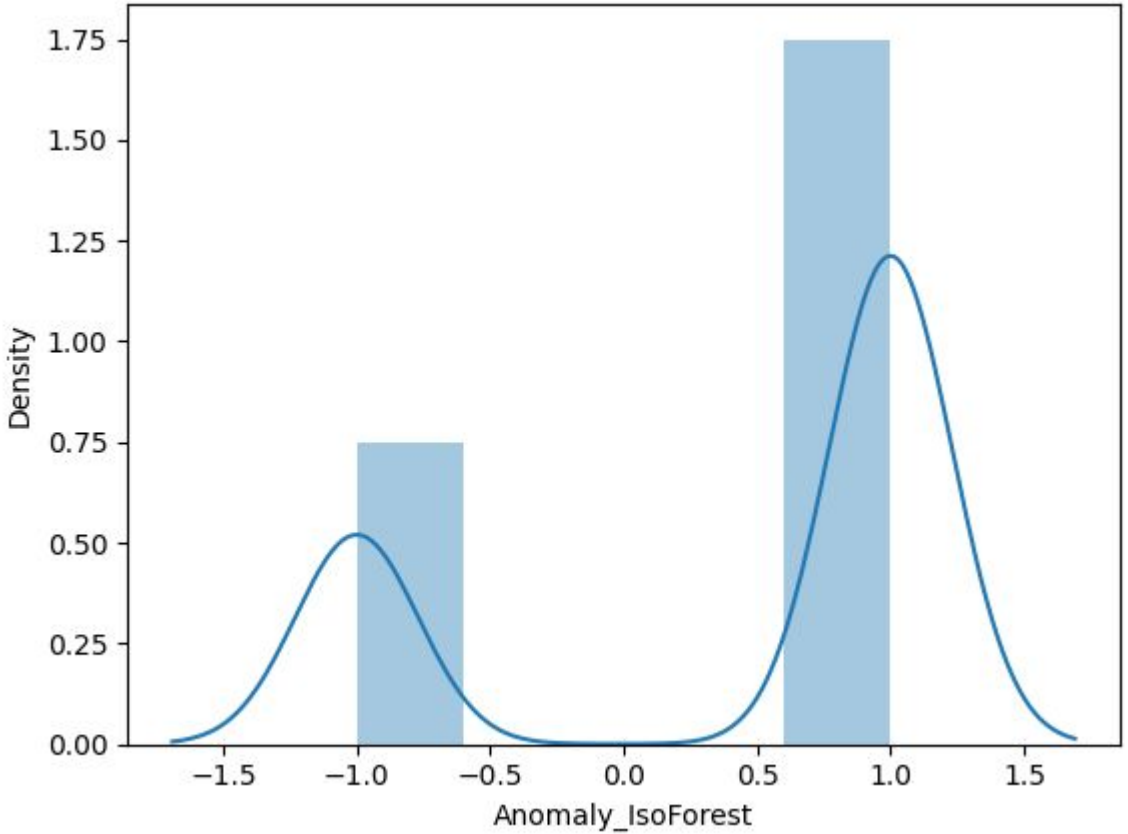
dtype: int64

```
[ ]  1 df1['Anomaly_Autoencoder'].value_counts()
```

Anomaly_Autoencoder	count
0	950
1	50

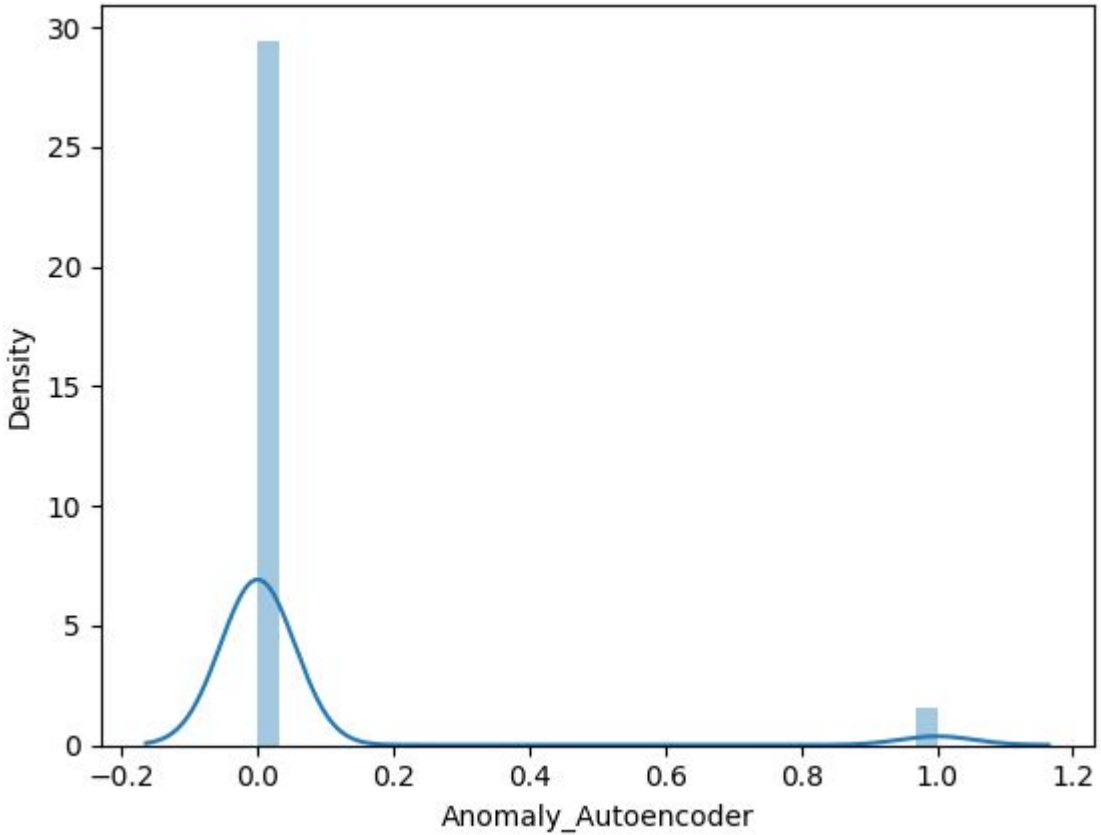
- 300 Overall Frauds included (High, Medium & Low)
- 50 Very High Risk Frauds

Anomaly_IsoForest



Skew : -0.8741833827654 95 -> Slightly Negative Skewed - Left Tail

Anomaly_Autoencoder



Skew : 4.135689330282186 Highly Positively Skewed

1. Risk Classification

Feature Selection / Feature Engineering

It is used to train models for risk classification, claim prediction, and fraud detection.

```
]    1 df1_scaled.columns.values
```

```
array(['Unnamed: 0', 'Policy_ID', 'Customer_Age', 'Annual_Income',
       'Vehicle_Age_Property_Age', 'Claim_History', 'Premium_Amount',
       'Claim_Amount', 'Fraudulent_Claim', 'Risk_Score_Encoded',
       'Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life',
       'Policy_Type_Property', 'Gender_Female', 'Gender_Male',
       'Gender_Other'], dtype=object)
```

1st Prediction about Risk classification, / Risk_Score_Encoded Column

Risk_Score_Encoded as Target Columns

```
[ ]    1 feature = df1_scaled.copy()
       2
       3 feature = feature.drop(columns=['Unnamed: 0','Policy_ID','Risk_Score_Encoded',])
```

```
[ ]    1 target = df1_scaled['Risk_Score_Encoded']
```

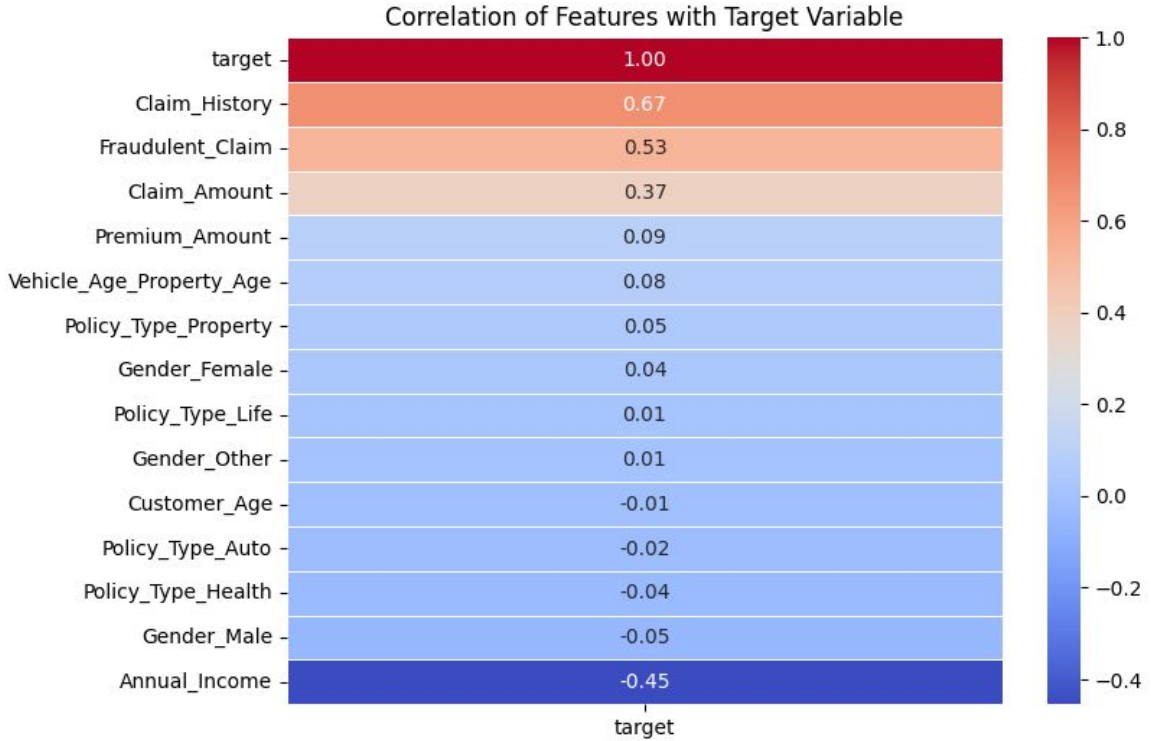
```
[ ]    1 feature.head()
```

	Customer_Age	Annual_Income	Vehicle_Age_Property_Age	Claim_History	Premium_Amount	Claim_Amount	Fraudulent_Claim	Policy_Type_Auto	Pol
0	58	0.741442	7	1	0.139545	0.102080	0	0	
1	65	0.590267	1	0	0.093670	0.232579	0	0	
2	56	0.715891	22	4	0.419729	0.449681	0	1	

```
1 target.head()
```

	Risk_Score_Encoded
0	0
1	0
2	1
3	1
4	2

dtype: int64



Filter Method:

Correlation Check

target (Risk_Score_Encoded) - Highly Contributed Feature Columns near to 0 is No Contributed

Model Training / Machine Learning

For 1) **Risk Classification**

1) Model Training

```
[ ]  1 x = feature
     2 y = target
```

```
[ ]  1 from sklearn.model_selection import train_test_split
```

```
[ ]  1 x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
```

```
1 target.value_counts()
```

Risk_Score_Encoded	count
2	340
0	330
1	330

dtype: int64

target is multiclass

Target is Multiclass so i use Random Forest Classifier

```
]    1 from sklearn.ensemble import RandomForestClassifier
```

```
]    1 # 3. Initialize the model (Random Forest Classifier)
     2 rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

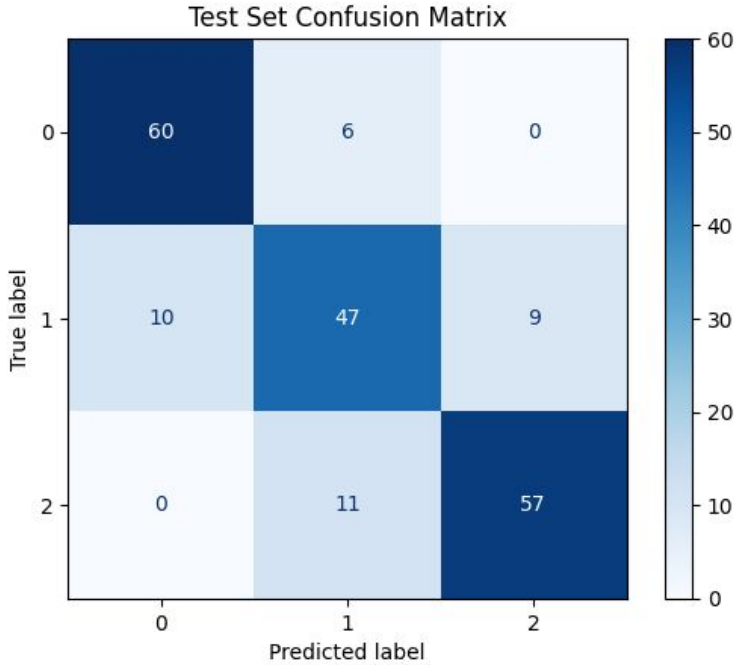
```
]    1 rf_model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Test Set Accuracy Score: 0.82

Test Set Classification Report:

	precision	recall	f1-score	support
0	0.86	0.91	0.88	66
1	0.73	0.71	0.72	66
2	0.86	0.84	0.85	68
accuracy			0.82	200
macro avg	0.82	0.82	0.82	200
weighted avg	0.82	0.82	0.82	200

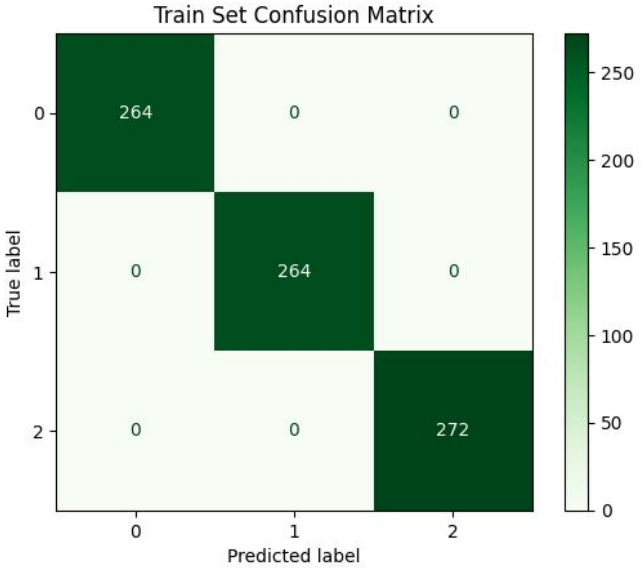


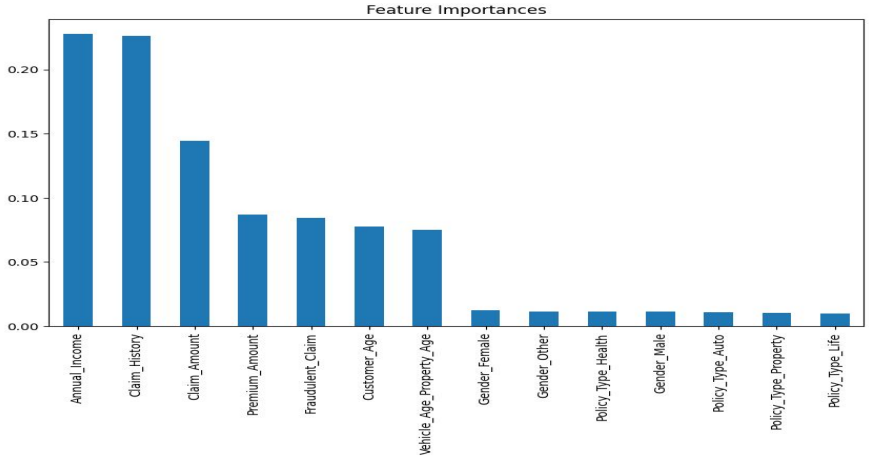

```
Class: 0
True Positive: 60
False Positive: 10
False Negative: 6
True Negative: 124

Class: 1
True Positive: 47
False Positive: 17
False Negative: 19
True Negative: 117

Class: 2
True Positive: 57
False Positive: 9
False Negative: 11
True Negative: 123
Class 0: High false positive rate – consider tuning the model to improve precision.
Class 1: High false negative rate – consider tuning the model to improve recall.
Class 2: High false negative rate – consider tuning the model to improve recall.
```

Train Set Accuracy Score: 1.0

Train Set Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	264
1	1.00	1.00	1.00	264
2	1.00	1.00	1.00	272
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800





High Contribute Features

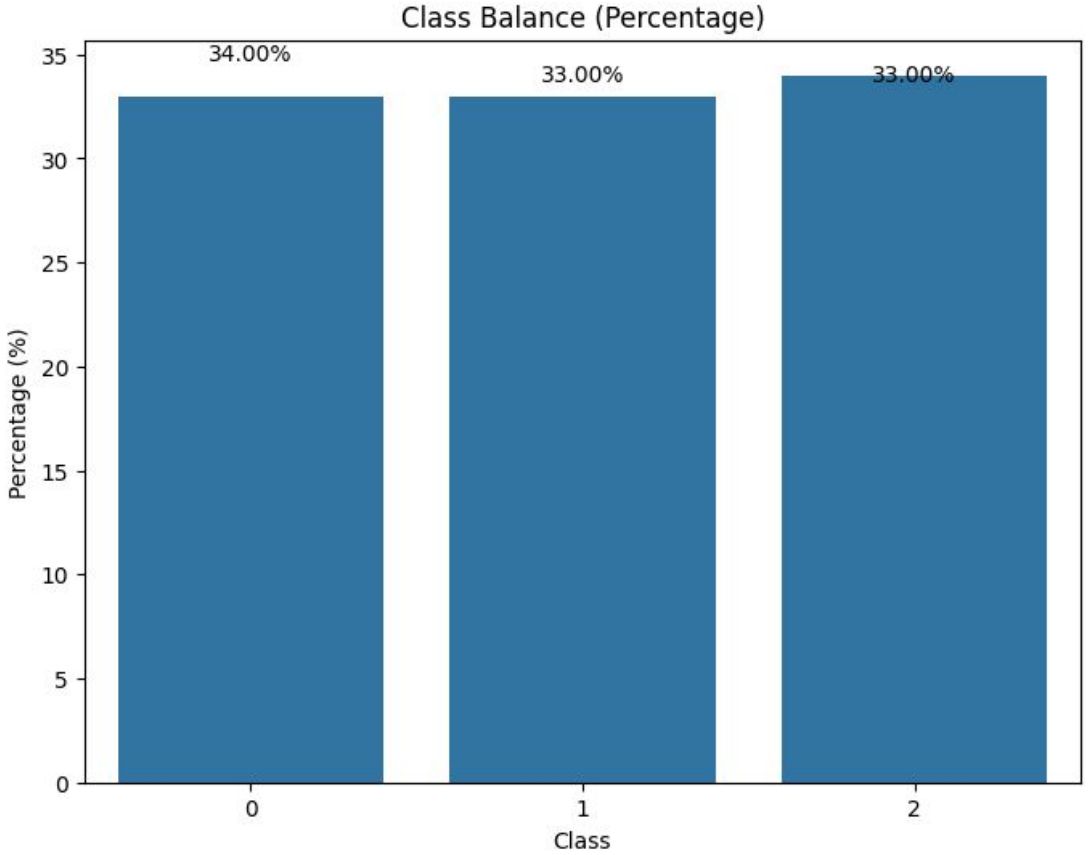
Check Accuracy Status Overfit or Underfit

```python
# Determine if the model is overfitting, underfitting, or generalizing well
test_accuracy = accuracy_score(y_test, y_pred_test)
train_accuracy = accuracy_score(y_train, y_pred_train)

print("Test Set Accuracy Score:", test_accuracy)
print("Train Set Accuracy Score:", train_accuracy)

if train_accuracy > test_accuracy + 0.1:
    print("Model is overfitting - it performs well on training data but poorly on test data.")
elif test_accuracy > train_accuracy + 0.1:
    print("Model is underfitting - it struggles to perform well even on training data.")
else:
    print("Model is generalizing well - performance is balanced between training and test sets.")

```

```
Test Set Accuracy Score: 0.82
Train Set Accuracy Score: 1.0
Model is overfitting - it performs well on training data but poorly on test data.
```

Train Accuracy 18% Higher than Test Accuracy that Means Train Reads More and Memorized - So Model is Overfit

Check Dataset Balanced or Unbalanced




```
Class distribution (absolute):
 Risk_Score_Encoded
2    340
0    330
1    330
Name: count, dtype: int64
Class distribution (percentage):
 Risk_Score_Encoded
2    34.0
0    33.0
1    33.0
Name: count, dtype: float64
The dataset is fairly balanced.
```

The dataset is balanced

3) now use cross validation

Cross Validation using Balanced Dataset

x_resampled, y_resampled

Using Folds 2

```
# Determine if the model is overfitting, underfitting, or generalizing well
test_accuracy = accuracy_score(y_test, y_pred_test)
train_accuracy = accuracy_score(y_train, y_pred_train)

print("Test Set Accuracy Score:", test_accuracy)
print("Train Set Accuracy Score:", train_accuracy)

if train_accuracy > test_accuracy + 0.1:
    print("Model is overfitting - it performs well on training data but poorly on test data.")
elif test_accuracy > train_accuracy + 0.1:
    print("Model is underfitting - it struggles to perform well even on training data.")
else:
    print("Model is generalizing well - performance is balanced between training and test sets.")

```

```
Test Set Accuracy Score: 0.82
Train Set Accuracy Score: 1.0
Model is overfitting - it performs well on training data but poorly on test data.
```

Accuracy is Overfitted so now try Hyper Parameter Tuning

4) Hyper Parameter tuning

Hyper Parameter Tunning user For Grid search CV using Random Forest Algorithm

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid for Random Forest
grid_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Grid Search CV without Stratified K-Fold
grid_search = GridSearchCV(estimator=rf_model, param_grid=grid_params, scoring='accuracy', n_jobs=-1)

# Fit the Grid Search to the resampled data (assuming x_resampled and y_resampled are your balanced datase
grid_search.fit(x, y)
```

```
21
22 # Best parameters and best score
23 print(f"Best parameters: {grid_search.best_params_}")
24 print(f"Best accuracy: {grid_search.best_score_:.4f}")
25
26 # Train the best model on the full training set
27 best_rf_model = grid_search.best_estimator_
28
29 # Predictions on the test set
30 y_pred_test = best_rf_model.predict(x_test)
31
32 # Test set evaluation
33 print("Test Set Accuracy Score:", accuracy_score(y_test, y_pred_test))
34 print("\nTest Set Classification Report:\n", classification_report(y_test, y_pred_test))
35
```

```
Test Set Accuracy Score: 0.83
Train Set Accuracy Score: 0.995
Model is overfitting – it performs well on training data but poorly on test data.
```

After Hyperparamter Tuning showing Test Accuracy is 83%

Now Using Deep Learning to Prevent Overfit

5) Deep Learning Neural Network

```
]    1 !pip install tensorflow
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

1. `Sequential`**:**

This is a class from Keras that lets you build a *linear stack* of layers — one after another. It's the simplest way to create a neural network where each layer has exactly one input and one output.

```
model = Sequential()
```

2. `Dense`**:**

This is a *fully connected (dense)* layer — one where every neuron in this layer is connected to every neuron in the previous and next layer. It's the most common type of layer in feedforward neural networks.

Parameters you'll often use:

- `units`: Number of neurons in the layer.
- `activation`: Activation function (like ReLU, sigmoid, softmax) that decides how to transform the layer's input.

3. `Dropout`**:**

This is a *regularization technique* to prevent overfitting. It randomly drops (deactivates) a fraction of neurons during training, forcing the model to learn more robust patterns.

The number you pass is the *dropout rate* — the fraction of neurons to drop.

Example:

python Copy Edit

```python
from tensorflow.keras.layers import Dropout
model.add(Dropout(rate=0.5))  # Drops 50% of neurons during training
```

4. `to_categorical`:

This is a utility function that *one-hot encodes* labels — turning numerical class labels (like `0`, `1`, `2`) into vectors where only one element is `1`, and the rest are `0`. It's often used for classification tasks.

Example:

python Copy Edit

```python
from tensorflow.keras.utils import to_categorical
labels = [0, 1, 2]
one_hot_labels = to_categorical(labels, num_classes=3)
print(one_hot_labels)
# Output: [[1. 0. 0.]
#          [0. 1. 0.]
#          [0. 0. 1.]]
```

```
# Example: Assuming X (features) and y (target) are already defined
X = feature
y = target

# One-hot encode target for multiclass classification
y = to_categorical(y)

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

# Define the neural network model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(y.shape[1], activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Evaluate the model on train data
train_loss, train_acc = model.evaluate(X_train, y_train, verbose=0)
print(f"Train Accuracy: {train_acc:.4f}")

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_acc:.4f}")

# Predict and get classification report
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred))
```

```
Train Accuracy: 0.8087
Test Accuracy: 0.8150
Test Set Accuracy Score: 0.8149999976158142
Train Set Accuracy Score: 0.8087499737739563
Model is generalizing well – performance is balanced between training and test sets.
```

Model is generalizing well - After deep learning

1) Increase or decrease the number of neurons or layers.

2) Tune the dropout rate to prevent overfitting.

3) Experiment with different activation functions like LeakyReLU or ELU.

4) Adjust learning rate in the Adam optimizer.

5) Try adding BatchNormalization layers.

Now Save the Deep Learning Model in .h5 #hdf5 Format

```
[ ]  1 model.save('my_model.h5')  # Or 'my_model' for SavedModel format
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.s
```

Future Prediction for Risk Classification

1) Deep Learning (Keras/TensorFlow) → .h5 (HDF5) on SavedModel

```
[ ]  1 from tensorflow.keras.models import load_model
```

```
[ ]  1 loaded_model = load_model('my_model.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics`
```

```
[ ]  1 loaded_model
```

```
<Sequential name=sequential_2, built=True>
```

```python
from tensorflow.keras.models import load_model
import numpy as np

# Load the saved model
loaded_model = load_model('my_model.h5')

# Example: New data for prediction (make sure it's preprocessed the same way)
new_data = np.array([[58, 0.741442111931488, 7, 1, 0.139544916784529, 0.102079830222524, 0, 0, 1, 0, 0, 0, 1, 0]])  # Sha

# Predict class probabilities
predictions = loaded_model.predict(new_data)

# If it's a classification model and you want the predicted class
predicted_class = np.argmax(predictions, axis=1)

print(f"Predicted class: {predicted_class[0]}")
print(f"Class probabilities: {predictions}")

```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
Predicted class: 0
Class probabilities: [[8.7902659e-01 1.2057626e-01 3.9713000e-04]]
```

Risk_Score prediction -> 0 : Low, 1 : Medium, 2 : High

```
[ ]
```

```python
from tensorflow.keras.models import load_model
import numpy as np

# Load the saved model
loaded_model1 = load_model('my_model.h5')

# Example: New data for prediction (make sure it's preprocessed the same way)
new_data1 = np.array([[23,0.379737271766022,14,5,0.834618703244114,0.162392620839527,0,0,1,0,0,0,1,0]])  # Shape: (1, numb

# Predict class probabilities
predictions1 = loaded_model1.predict(new_data1)

# If it's a classification model and you want the predicted class
predicted_class1 = np.argmax(predictions1, axis=1)

print(f"Predicted class: {predicted_class1[0]}")
print(f"Class probabilities: {predictions1}")

```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
Predicted class: 2
Class probabilities: [[0.00399181 0.19506039 0.8009478 ]]
```

3) Encoded to Decoded and Scaled to Descaled run Model for User Convinence

```
[ ]
import pickle
```

```
[ ]
with open('lbl_encoder.pkl', 'rb') as file:
    lbl_encoder = pickle.load(file)

with open('onehot_encoder.pkl', 'rb') as file:
    onehot_encoder = pickle.load(file)

```

2) Claim Prediction

General Condition

Claim There are upto 5 Claims Happens so i fix it when claim history is More than 2 means claim will be failed

Overall Working Procedure

 # filed claim write based on claim history more than 2 is filed and write policy type health more than 3 is filed

 # and policy type more than 1 is filed and gender female means policy type health more than 4 is filed

 # and Vehicle_Age_Property_Age is more than 15 is filed using

```python
import pandas as pd


# Define 'Filed_Claim' based on multiple conditions
df1['Filed_Claim'] = df1.apply(lambda row: 1 if (
    row['Claim_History'] > 2 or
    row['Policy_Type_Health'] > 3 or
    row['Policy_Type_Auto'] > 1 or
    row['Policy_Type_Life'] > 1 or
    row['Policy_Type_Property'] > 1 or
    (row['Gender_Female'] == 1 and row['Policy_Type_Health'] > 4) or
    row['Vehicle_Age_Property_Age'] > 15
) else 0, axis=1)

```

Filed_Claim
0
0
1
0

146 Customers did't claim single time

486 Customers are claim less than or equal 2 Claims

so focus on claim below 2 was eligible for claims otherwise failed

Customer_Age → Older or younger customers may have different claim behaviors

Gender → Sometimes used for risk profiling (though it depends on local regulations)

Policy_Type → Different policies have different likelihoods of claims (e.g., Auto vs. Life)

Annual_Income → Income levels can affect the likelihood of claims

Vehicle_Age / Property_Age → Older assets might have more claims due to wear and tear

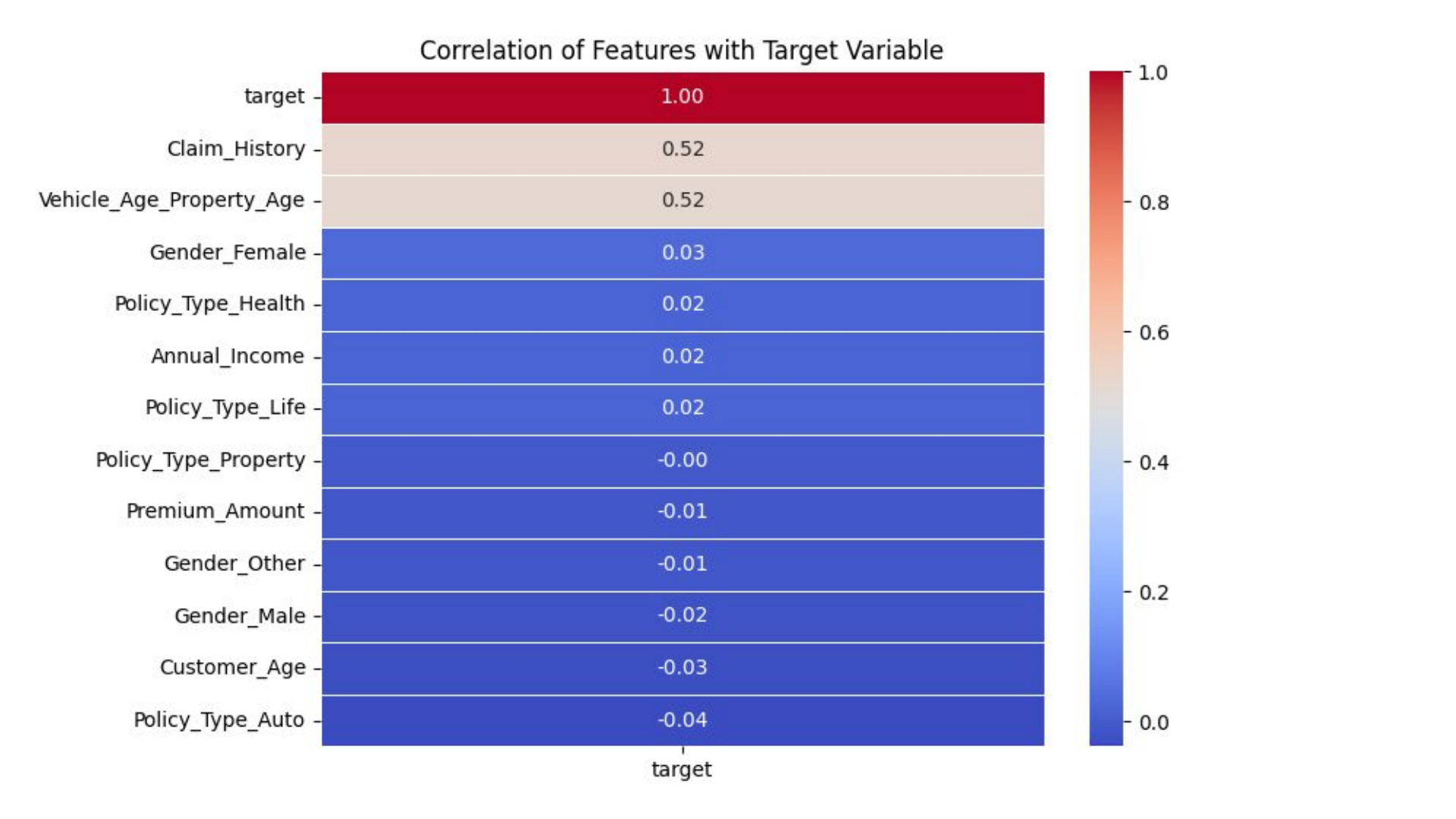
Claim_History → A customer with frequent claims is likely to claim again

Premium_Amount → Higher premiums could indicate higher-value policies, which may correlate with higher or more frequent claims

Risk_Score → Pre-assessed risk category might indicate likelihood of claims

Input features: Customer_Age, Gender, Policy_Type, Annual_Income, Vehicle_Age_Property_Age, Premium_Amount, Claim_History

Taget Column: Filed_Claim


Correlation of Features with Target Variable
target
Claim_History
Vehicle_Age_Property_Age
Gender_Female
Policy_Type_Health
Annual_Income
Policy_Type_Life
Policy_Type_Property
Premium_Amount
Gender_Other
Gender_Male
Customer_Age
Policy_Type_Auto
1.00
0.52
0.52
0.03
0.02
0.02
0.02
-0.00
-0.01
-0.01
-0.02
-0.03
-0.04
target
1.0
0.8
0.6
0.4
0.2
0.0

Model Training - Logistic Regression

Target is Binary Classification Type

```
[ ]  1 from sklearn.linear_model import LogisticRegression
```

```
[ ]  1 logis_model = LogisticRegression()
```

```
     1 logis_model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

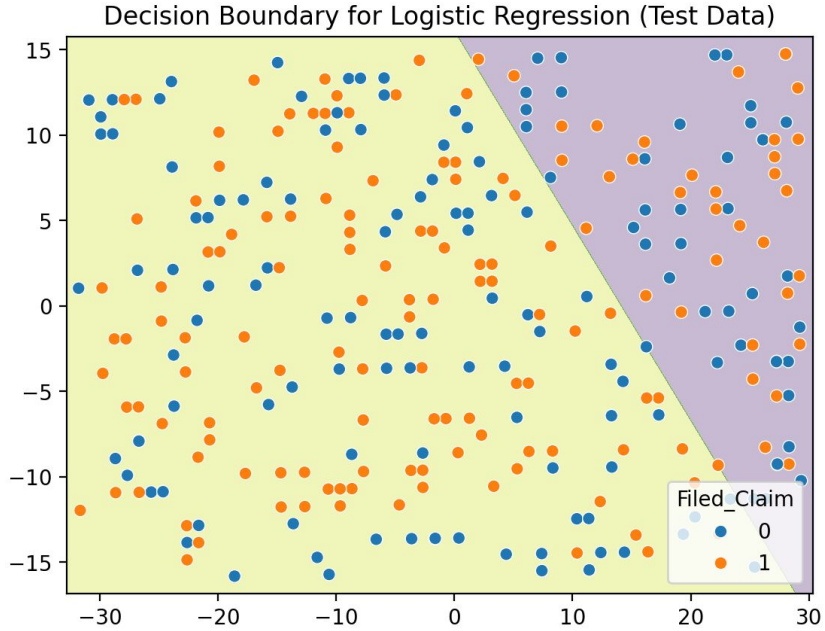
LogisticRegression()

```
[ ]  1 y_pred_test = logis_model.predict(x_test)
```

Test Data

```
[ ]  1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
     2
     3 # Calculate metrics
     4 accuracy = accuracy_score(y_test, y_pred_test)
     5 precision = precision_score(y_test, y_pred_test)
     6 recall = recall_score(y_test, y_pred_test)
     7 f1 = f1_score(y_test, y_pred_test)
     8 conf_matrix = confusion_matrix(y_test, y_pred_test)
     9 report = classification_report(y_test, y_pred_test)
    10
    11 # Print metrics
    12 print(f"Accuracy: {accuracy}")
    13 print(f"Precision: {precision}")
    14 print(f"Recall: {recall}")
    15 print(f"F1 Score: {f1}")
    16 print(f"Confusion Matrix:\n{conf_matrix}")
    17 print(f"Classification Report:\n{report}")
    18
```

```
Accuracy: 0.915
Precision: 0.9371069182389937
Recall: 0.9551282051282052
F1 Score: 0.946031746031746
Confusion Matrix:
[[ 34  10]
 [  7 149]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.77      0.80        44
           1       0.94      0.96      0.95       156

    accuracy                           0.92       200
   macro avg       0.88      0.86      0.87       200
weighted avg       0.91      0.92      0.91       200
```

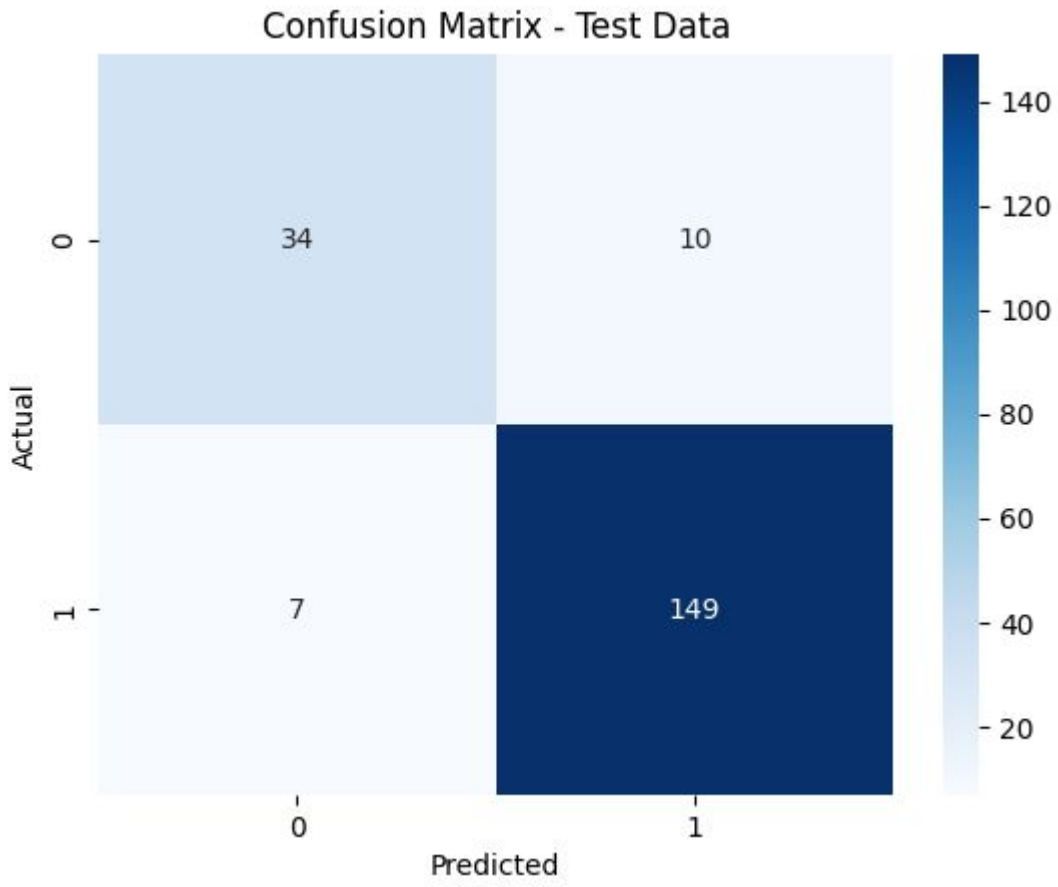


1) 0 (blue points): No filed claim - as 2 or Below Claims are alow to next Claim

2) 1 (orange points): Filed claim

3) Each shade shows the area where the model predicts either class "0" or "1".For example, the lighter region might represent the model's prediction for "No filed claim" (class 0) - Still 0 Claims Happens,

4) and the darker region for "Filed claim" (class 1).


Confusion Matrix - Test Data
34
10
7
149
0
1
Actual
0
1
Predicted
140
120
100
80
60
40
20

```
Test Data Results:
Accuracy: 0.915
Precision: 0.9371069182389937
Recall: 0.9551282051282052
F1 Score: 0.946031746031746
True Positives (TP): 149
True Negatives (TN): 34
False Positives (FP): 10
False Negatives (FN): 7

Interpretation:
- True Positives: 149 cases where the model correctly predicted a filed claim.
- True Negatives: 34 cases where the model correctly predicted no filed claim.
- False Positives: 10 cases where the model wrongly predicted a filed claim when there was none.
- False Negatives: 7 cases where the model missed an actual filed claim.

Train Data Results:
Accuracy: 0.91625
Precision: 0.9389830508474576
Recall: 0.947008547008547
F1 Score: 0.9429787234042554

Model seems well-balanced between train and test data.
```

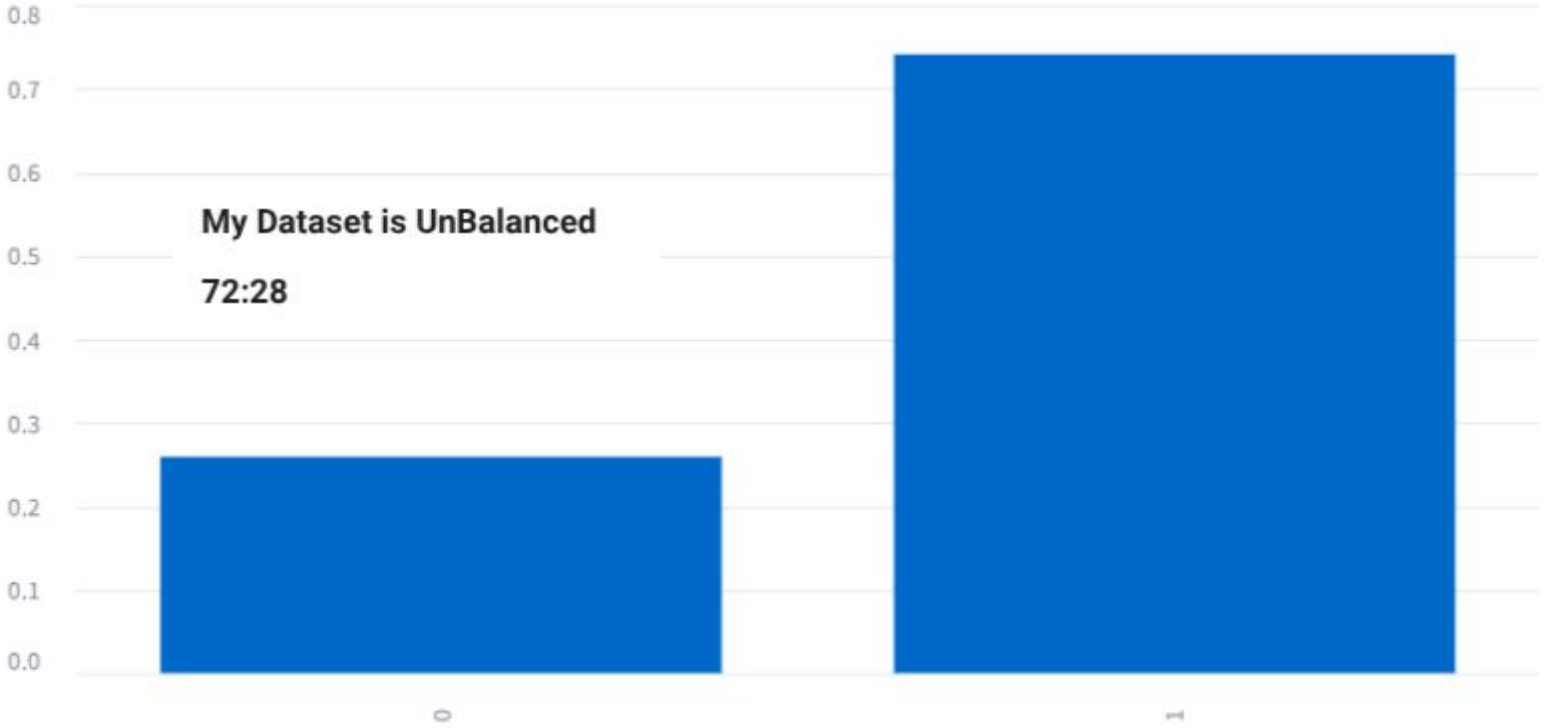
Model is Well Balanced

Interpretation:

- True Positives: 149 cases where the model correctly predicted a filed claim.
- True Negatives: 34 cases where the model correctly predicted no filed claim.
- False Positives: 10 cases where the model wrongly predicted a filed claim when there was none.
- False Negatives: 7 cases where the model missed an actual filed claim.

Dataset Balance Status: Unbalanced

Class Distribution:

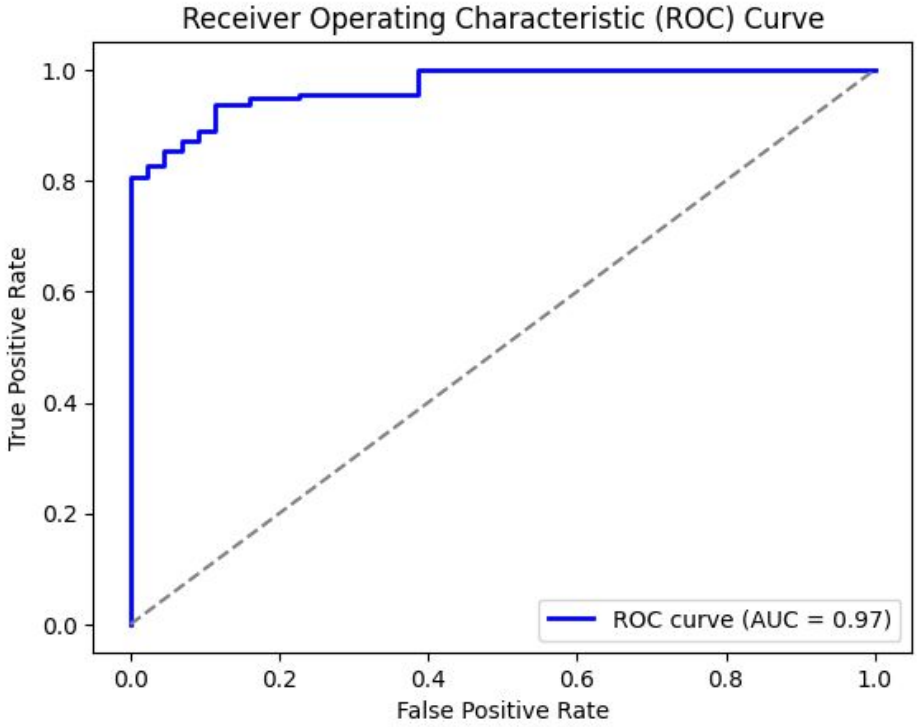


ROC and AUC Curve

```
from sklearn.metrics import  roc_curve, auc
```

```
# ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_probs_test)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

print(f"\nAUC (Area Under the Curve): {roc_auc}")

# Interpretation of ROC and AUC
print("\nWhy ROC and AUC are important:")
print("- The ROC curve shows the trade-off between sensitivity (True Positive Rate) and specificity (1 - False Positive Rate).
print("- AUC represents the probability that the model ranks a random positive example more highly than a random negative exam
print("- AUC closer to 1 means a better-performing model, while 0.5 suggests no discrimination between classes.")
```

```python
# Choosing an appropriate threshold
best_threshold_index = (tpr - fpr).argmax()
best_threshold = thresholds[best_threshold_index]
print(f"\nSuggested optimal threshold: {best_threshold:.2f}")
print("- This threshold balances sensitivity and specificity to achieve the best classification performance.")
```



```
AUC (Area Under the Curve): 0.9694055944055944

Why ROC and AUC are important:
- The ROC curve shows the trade-off between sensitivity (True Positive Rate) and specificity (1 - False Positive Rate).
- AUC represents the probability that the model ranks a random positive example more highly than a random negative example.
- AUC closer to 1 means a better-performing model, while 0.5 suggests no discrimination between classes.

Suggested optimal threshold: 0.70
- This threshold balances sensitivity and specificity to achieve the best classification performance.

Train Data Results:
Accuracy: 0.91625
Precision: 0.9389830508474576
Recall: 0.947008547008547
F1 Score: 0.9429787234042554

Model seems well-balanced between train and test data.
```

Suggested Optimal Threshold is 0.70

Differnce of True Posative Rate (TPR) and False Posative Rate (FPR) : TPR-FPR to find Best Optimal Threshold for this model

AUC (Area Under the Curve): 0.9755244755244754 is Almost near to 1: so its very Good

Cross Validation: Stratified K-folds

Focusing Class 0 as -> 25% and Class 1 as -> 75%

to extra focusing on Minority Class also

```
from sklearn.model_selection import StratifiedKFold
```

```
# Class distribution focus: 25% class 0, 75% class 1
class_distribution = {0: 0.25, 1: 0.75}
```

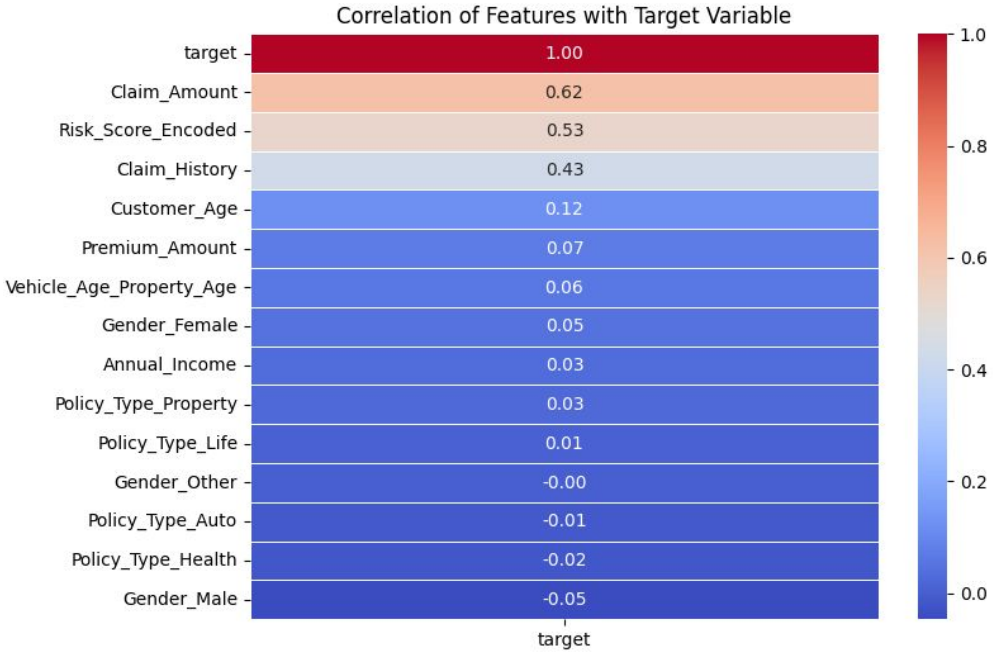
```
Cross-Validation Results:
Average Train Accuracy: 0.92
Average Test Accuracy: 0.91
Average Precision: 0.93
Average Recall: 0.94
Average F1 Score: 0.94
Average AUC: 0.97
```

Model Saved

```
1 import joblib
2 # Fit model
3 logis_model.fit(x_train, y_train)
4
5     # Save model
6 joblib.dump(logis_model, 'logistic_regression_model.pkl')
```

3) Fraud Detection

Feature Selection - Filter Method (HeatMap)



Feature Selection - Wrapper Method (RFE)

Select Highly Contributed Columns for feature using RFE -Recursive Feature Elimination (RFE) (Wrapper Method)

can apply RFE with Random Forest Regressor:

```
Selected Features: Index(['Customer_Age', 'Annual_Income', 'Vehicle_Age_Property_Age',
       'Claim_History', 'Premium_Amount', 'Claim_Amount', 'Risk_Score_Encoded',
       'Gender_Female', 'Gender_Male', 'Gender_Other'],
      dtype='object')
                     Feature  Ranking
0               Customer_Age        1
1              Annual_Income        1
2   Vehicle_Age_Property_Age        1
3              Claim_History        1
4             Premium_Amount        1
5               Claim_Amount        1
6         Risk_Score_Encoded        1
11             Gender_Female        1
12               Gender_Male        1
13              Gender_Other        1
8         Policy_Type_Health        2
9           Policy_Type_Life        3
10      Policy_Type_Property        4
7           Policy_Type_Auto        5
```

Model Training

```
[ ]  1 x = feature
     2 y = target
```

```
[ ]  1 from sklearn.model_selection import train_test_split
```

```
[ ]  1 x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
```

```
[ ]  1 target.value_counts()
```

Fraudulent_Claim	count
0	700
1	300

Target is Binary Classification Type

```
[ ]  1 from sklearn.linear_model import LogisticRegression
```

```
[ ]  1 logi_model = LogisticRegression()
```

```
[ ]  1 logi_model.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression()
```

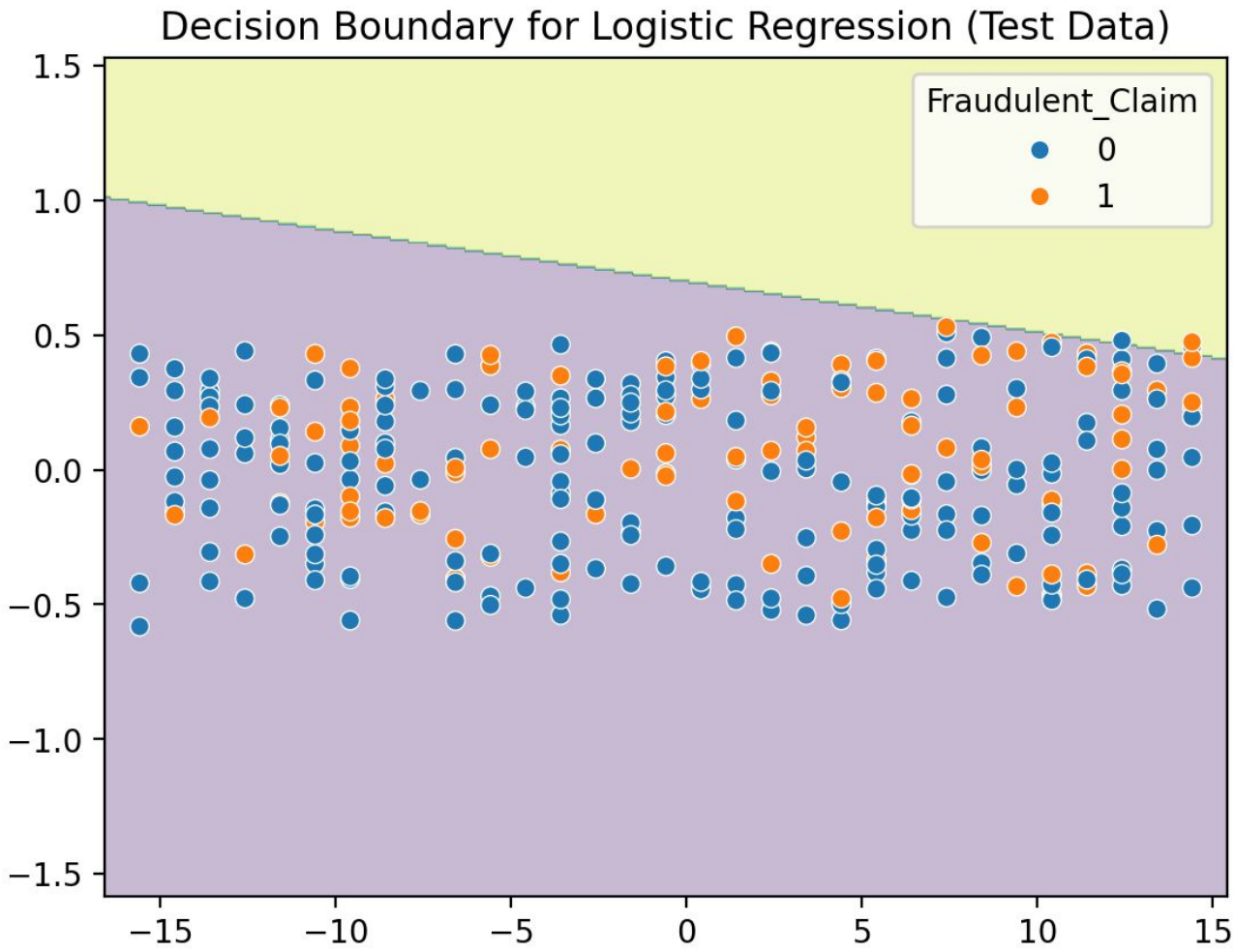
For Test prediction

```
[ ]  1 y_pred_test = logi_model.predict(x_test)
```

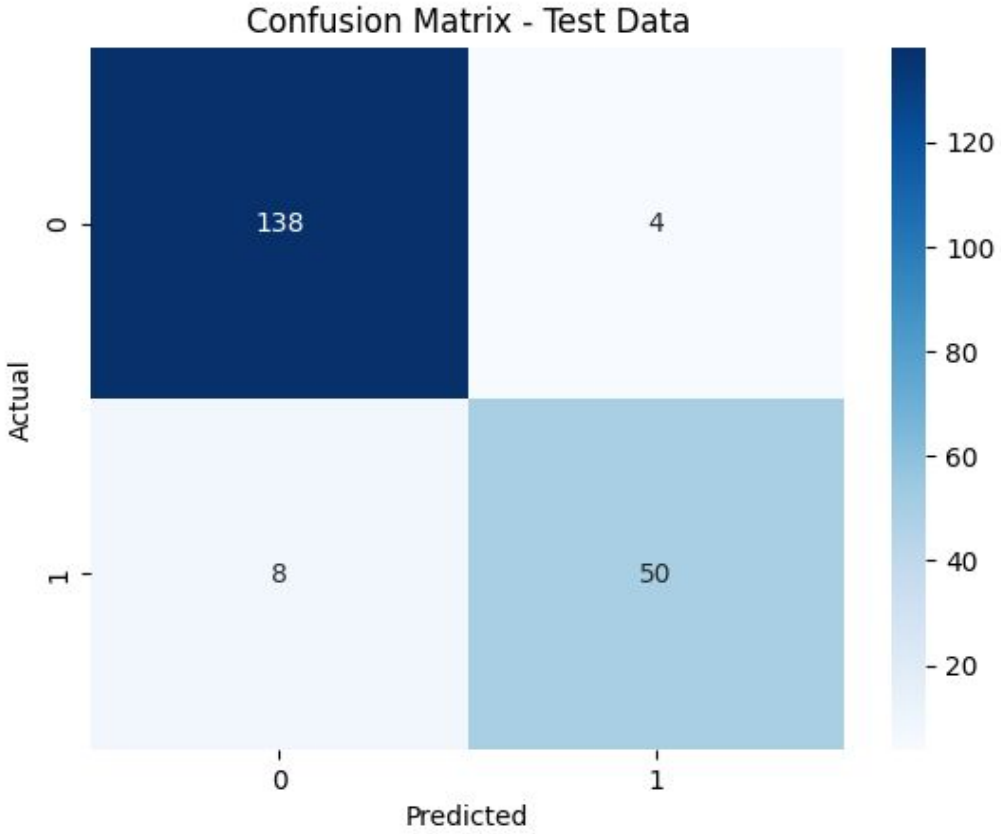
Test Data

```
[ ]  1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
     2
     3 # Calculate metrics
     4 accuracy = accuracy_score(y_test, y_pred_test)
     5 precision = precision_score(y_test, y_pred_test)
     6 recall = recall_score(y_test, y_pred_test)
     7 f1 = f1_score(y_test, y_pred_test)
     8 conf_matrix = confusion_matrix(y_test, y_pred_test)
     9 report = classification_report(y_test, y_pred_test)
    10
    11 # Print metrics
    12 print(f"Accuracy: {accuracy}")
    13 print(f"Precision: {precision}")
    14 print(f"Recall: {recall}")
    15 print(f"F1 Score: {f1}")
    16 print(f"Confusion Matrix:\n{conf_matrix}")
    17 print(f"Classification Report:\n{report}")
    18
```

```
Accuracy: 0.94
Precision: 0.9259259259259259
Recall: 0.8620689655172413
F1 Score: 0.8928571428571429
```


Decision Boundary for Logistic Regression (Test Data)
Fraudulent_Claim
0
1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−15
−10
−5
0
5
10
15

Train and Test Result Compare to Overfit or Underfit



Interpretation:
- True Positives: 50 cases where the model correctly predicted fraud.
- True Negatives: 138 cases where the model correctly predicted no fraud.
- False Positives: 4 cases where the model wrongly predicted fraud when there was none.
- False Negatives: 8 cases where the model missed actual fraud.

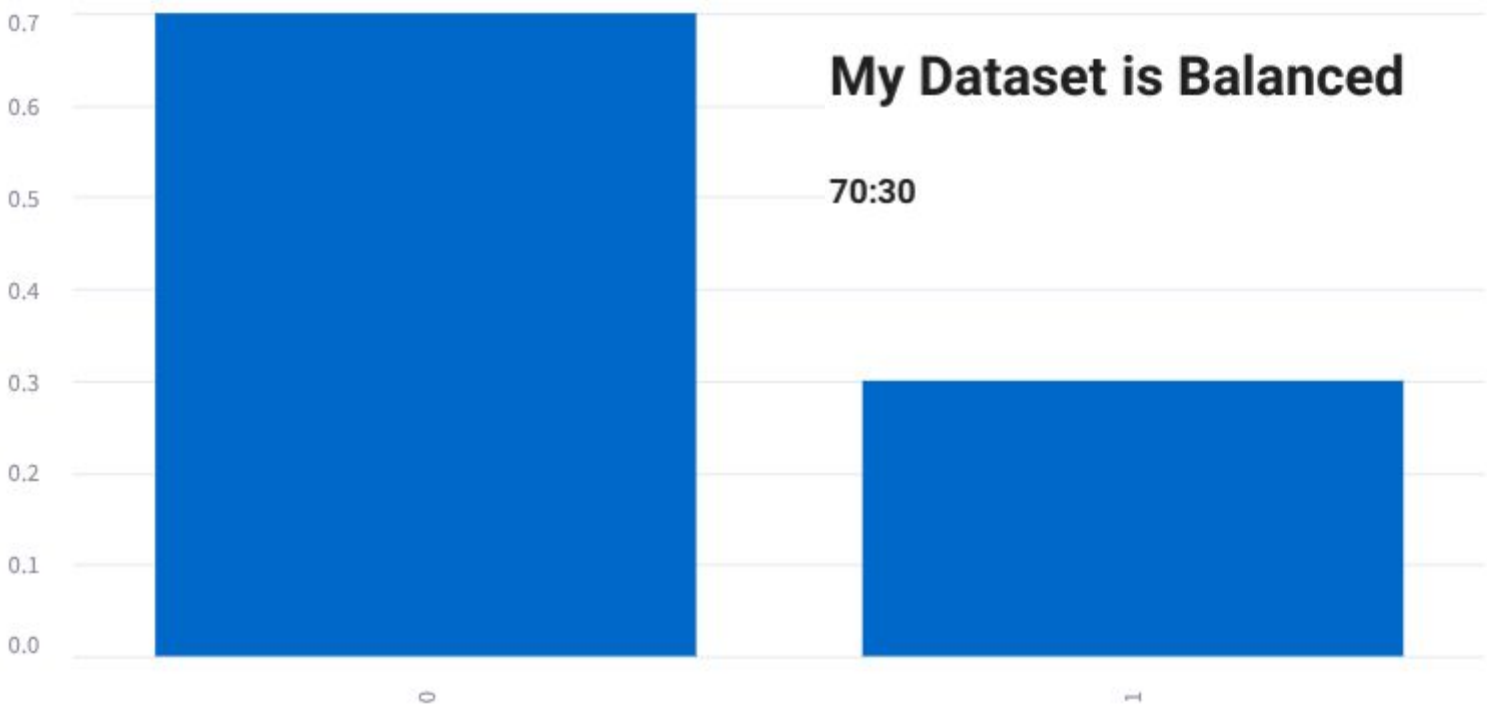
Test Data Results:
Accuracy: 0.94
Precision: 0.9259259259259259
Recall: 0.8620689655172413
F1 Score: 0.8928571428571429

Train Data Results:
Accuracy: 0.93625
Precision: 0.920704845814978
Recall: 0.8636363636363636
F1 Score: 0.8912579957356077

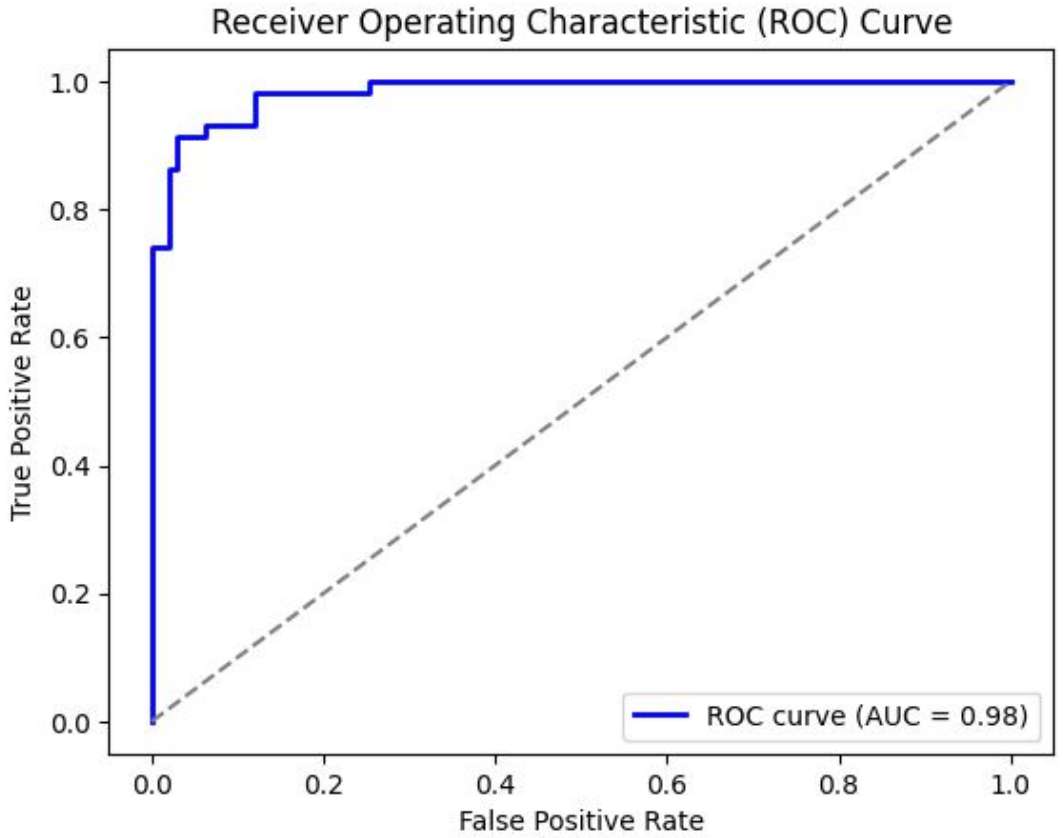
Model seems well-balanced between train and test data

Dataset Balance Status: Balanced

Class Distribution:



ROC and AUC Curve



```
AUC (Area Under the Curve): 0.9843370568237009

Why ROC and AUC are important:
- The ROC curve shows the trade-off between sensitivity (True Positive Rate) and specificity (1 - False Positive Rate).
- AUC represents the probability that the model ranks a random positive example more highly than a random negative example.
- AUC closer to 1 means a better-performing model, while 0.5 suggests no discrimination between classes.

Suggested optimal threshold: 0.49
- This threshold balances sensitivity and specificity to achieve the best classification performance.
```

Differnce of True Posative Rate (TPR) and False Posative Rate (FPR) : TPR-FPR to find Best Optimal Threshold for this model

AUC (Area Under the Curve): 0.9843370568237009 is Almost near to 1: so its very Good

Statement: My Point of View

is AUC and ROC Curve focus maximum increasing a true posative value from false posative

eg: to get avoid little bit fraud person to get more customer bank.

The goal is usually to avoid too many false positives while still catching enough fraud — like you said, so the bank avoids falsely rejecting too many good customers while still identifying fraud

Save this Model:

```
[ ]
1 import joblib
2
3 # Save the model to a file
4 joblib.dump(logi_model, 'logi_model.pkl')
5
6 print("Model saved successfully!")
7
```

```
Model saved successfully!
```

Prediction:

Binary Classification Model Saved

```
[ ]
1 import joblib
2
3 # Load the saved model
4 logi_model = joblib.load('logi_model.pkl')
5
6 print("Model loaded successfully!")
7
```

```
Model loaded successfully!
```

```python
import joblib
import numpy as np
import pandas as pd

# Load the saved logistic regression model
logi_model = joblib.load('logi_model.pkl')

# Example: New data for prediction (ensure it's preprocessed the same way)
new_data = pd.DataFrame({
    'Vehicle_Age_Property_Age': [7],
    'Premium_Amount': [0.139544916784529],
    'Claim_Amount': [0.102079830222524],
    'Claim_History': [1]
})

# Predict class probabilities
predictions = logi_model.predict_proba(new_data)

# Predicted class (0 or 1)
predicted_class = logi_model.predict(new_data)

print(f"Predicted class: {predicted_class[0]}")
print(f"Class probabilities: {predictions}")

```

```
Predicted class: 0
Class probabilities: [[9.99741618e-01 2.58382396e-04]]
```

```python
import joblib
import numpy as np
import pandas as pd

# Load the saved logistic regression model
logi_model = joblib.load('logi_model.pkl')

# Example: New data for prediction (ensure it's preprocessed the same way)
new_data = pd.DataFrame({
    'Vehicle_Age_Property_Age': [21],
    'Premium_Amount': [0.0713602682820519],
    'Claim_Amount': [0.635925845869675],
    'Claim_History': [5]
})

# Predict class probabilities
predictions = logi_model.predict_proba(new_data)

# Predicted class (0 or 1)
predicted_class = logi_model.predict(new_data)

print(f"Predicted class: {predicted_class[0]}")
print(f"Class probabilities: {predictions}")

```

```
Predicted class: 1
Class probabilities: [[0.17726304 0.82273696]]
```

1 Mean by Fraud and 0 Means by Geniune

Prediction Work Perfectly

4. Claim Amount Model Training

Claim Amount Prediction

```
[ ]  1 import pandas as pd
```

```
[ ]  1 df = pd.read_csv('df1_scaled.csv')
```

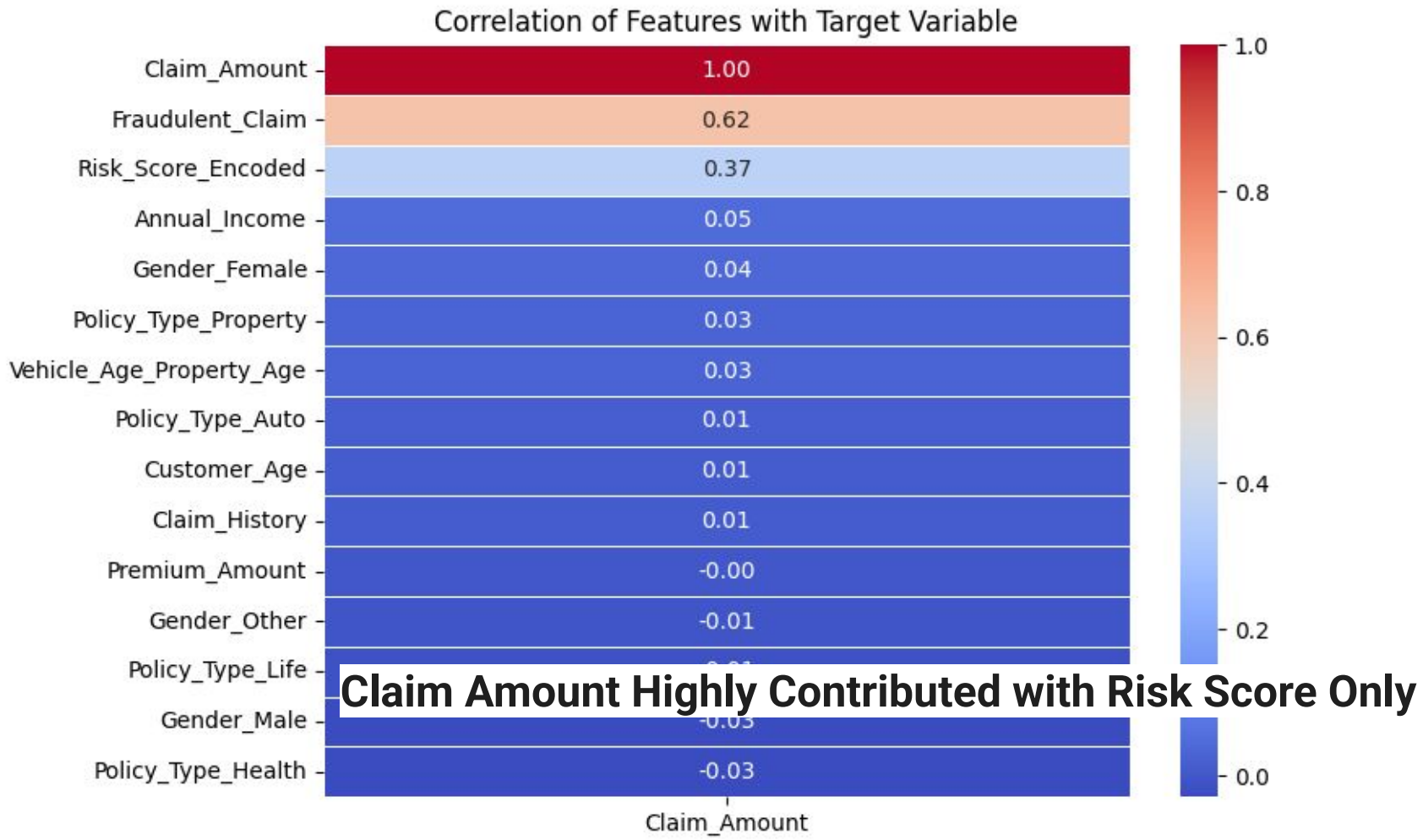
```
[ ]  1 df.head()
```

```
[ ]  1 df.columns
```

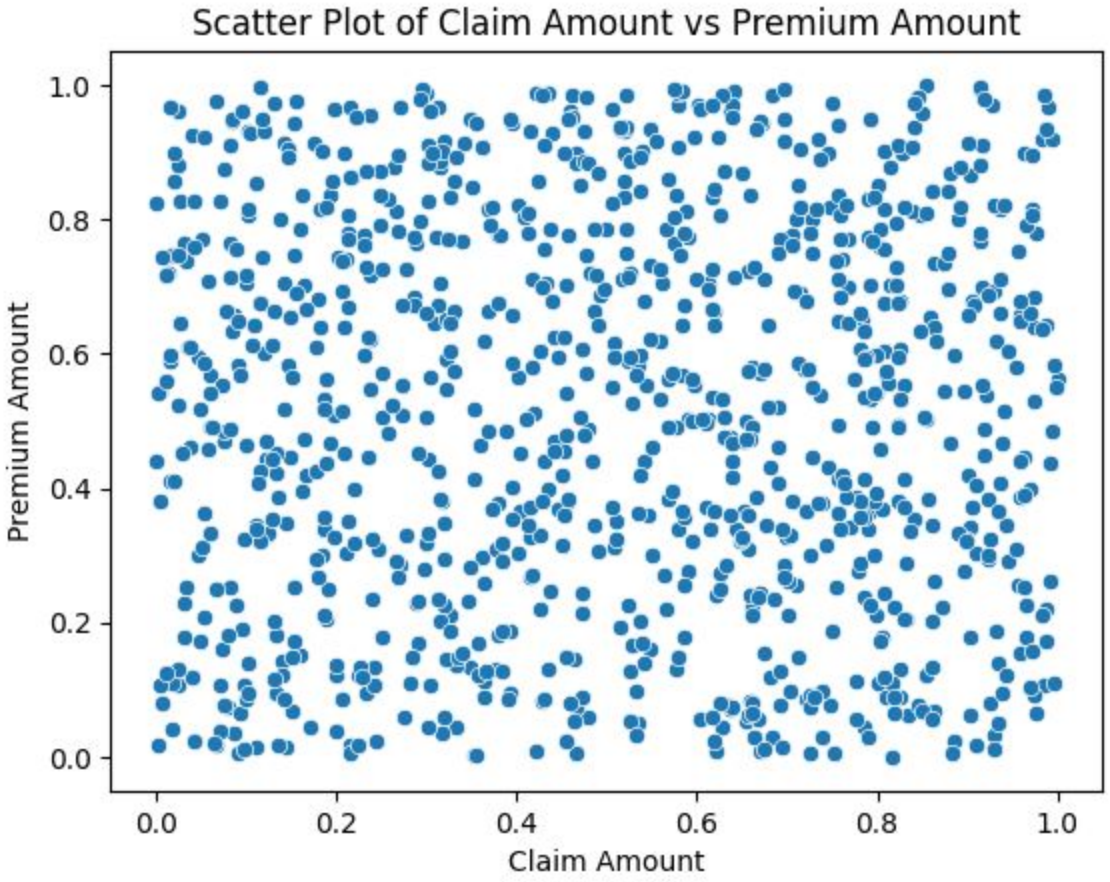
```
Index(['Unnamed: 0', 'Policy_ID', 'Customer_Age', 'Annual_Income',
       'Vehicle_Age_Property_Age', 'Claim_History', 'Premium_Amount',
       'Claim_Amount', 'Fraudulent_Claim', 'Risk_Score_Encoded',
       'Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life',
       'Policy_Type_Property', 'Gender_Female', 'Gender_Male', 'Gender_Other'],
      dtype='object')
```

```
[ ]  1 feature_correlated = [['Customer_Age', 'Annual_Income',
     2        'Vehicle_Age_Property_Age', 'Claim_History', 'Premium_Amount',
     3        'Fraudulent_Claim', 'Risk_Score_Encoded',
     4        'Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life',
     5        'Policy_Type_Property', 'Gender_Female', 'Gender_Male', 'Gender_Other']]
     6 target_correlated = ['Claim_Amount']
```

```
[ ]  1 df['Claim_Amount'].head()
```



Claim Amount Highly Contributed with Risk Score Only



No Correlation

So am taking Linear Based Synthetic Dataset

```
# Save to CSV
df.to_csv("Insurance_RISK_and_CLAIM.csv", index=False)
```

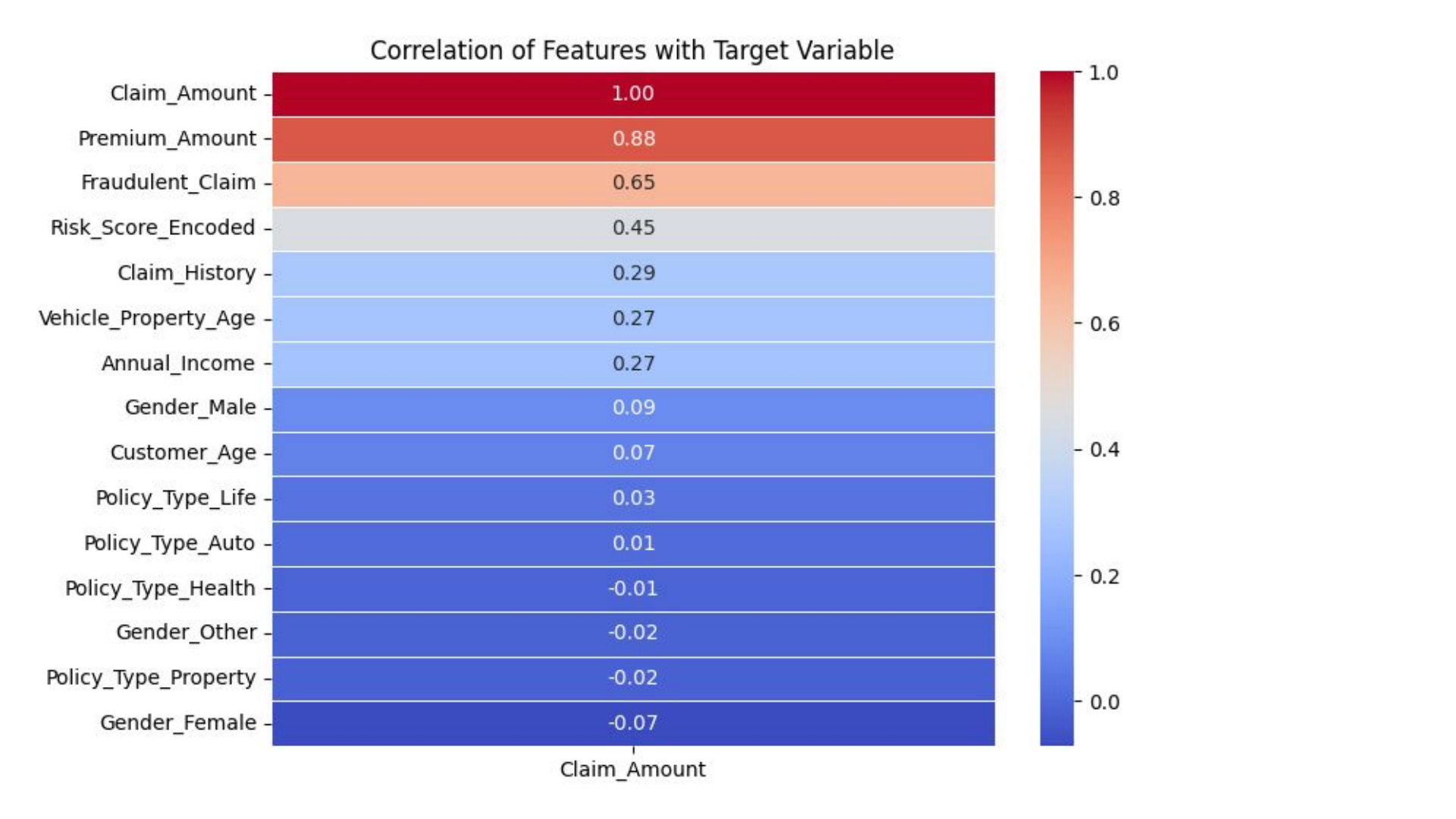
```
df1_new = pd.read_csv('Insurance_RISK_and_CLAIM.csv')
df1_new.head()
```

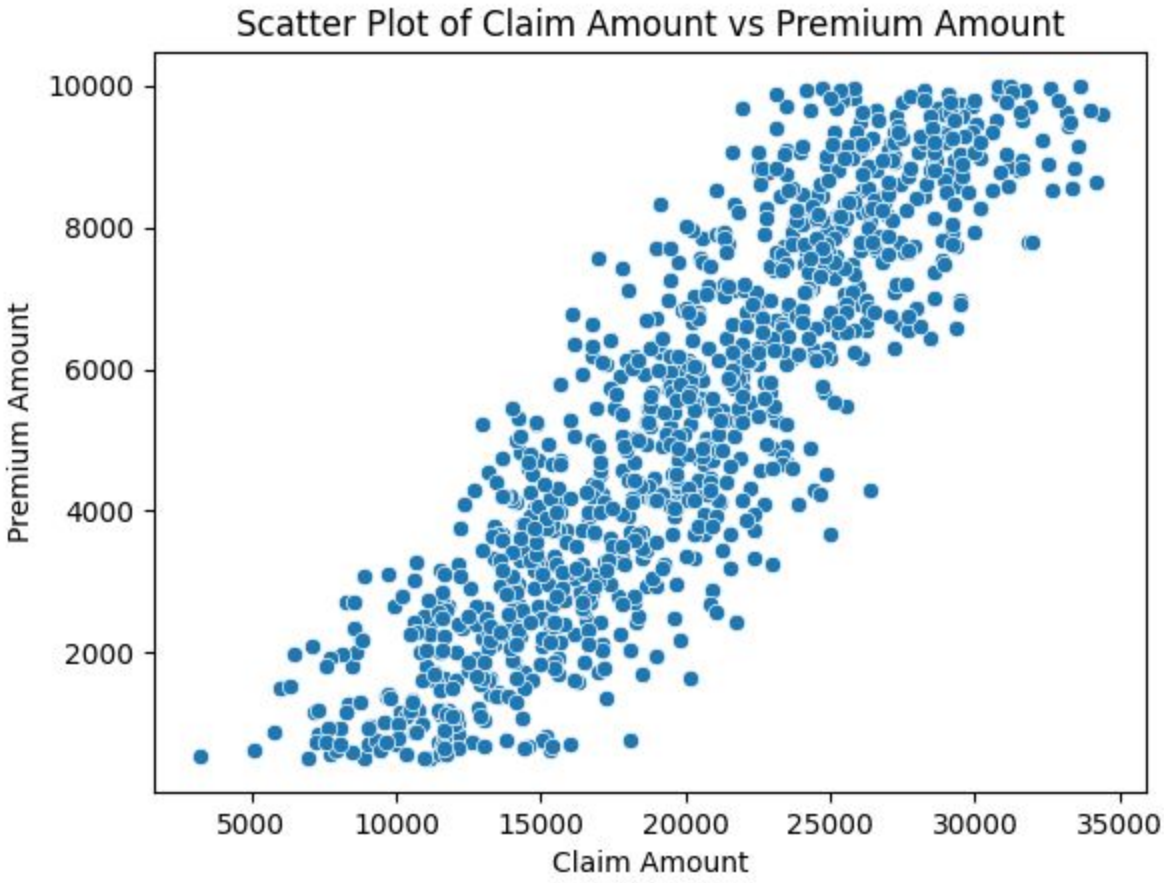
	Policy_ID	Customer_Age	Gender	Policy_Type	Annual_Income	Vehicle_Property_Age	Claim_History	Premium_Amount	Claim_Amount	Risk_Score
0	09c8c556-904d-4c3b-8967-ab2b12ae887e	56	Other	Property	92292	25	1	2400	15595.321018	Lov
1	7f86c80e-76e3-484c-8ecc-090441c6f512	69	Male	Auto	53833	16	2	7480	23020.823333	Higl
	064fab0f-fc58-									

Save encoded Files

```
import pickle
with open('onehot_encoder_claim.pkl', 'wb') as file:
    pickle.dump(onehot_encoder, file)
```

```
with open('lbl_encoder_claim.pkl', 'wb') as file:
    pickle.dump(lbl_encoder, file)
```


Correlation of Features with Target Variable
Claim_Amount
Premium_Amount
Fraudulent_Claim
Risk_Score_Encoded
Claim_History
Vehicle_Property_Age
Annual_Income
Gender_Male
Customer_Age
Policy_Type_Life
Policy_Type_Auto
Policy_Type_Health
Gender_Other
Policy_Type_Property
Gender_Female
1.00
0.88
0.65
0.45
0.29
0.27
0.27
0.09
0.07
0.03
0.01
-0.01
-0.02
-0.02
-0.07
Claim_Amount
1.0
0.8
0.6
0.4
0.2
0.0


Scatter Plot of Claim Amount vs Premium Amount
Premium Amount
Claim Amount
10000
8000
6000
4000
2000
5000
10000
15000
20000
25000
30000
35000

its a Linear Regressor type Highly positive Correlation

this dataset suite for predict Claim Amount

Based on Heat correlation map Recommentded Features

1) Premium Amount
2) Fraudulent Claim
3) Risk Score
4) Claim History
5) Vehicle Property Age
6) Annual Income

Based on Subject Matter Expert Recommendation

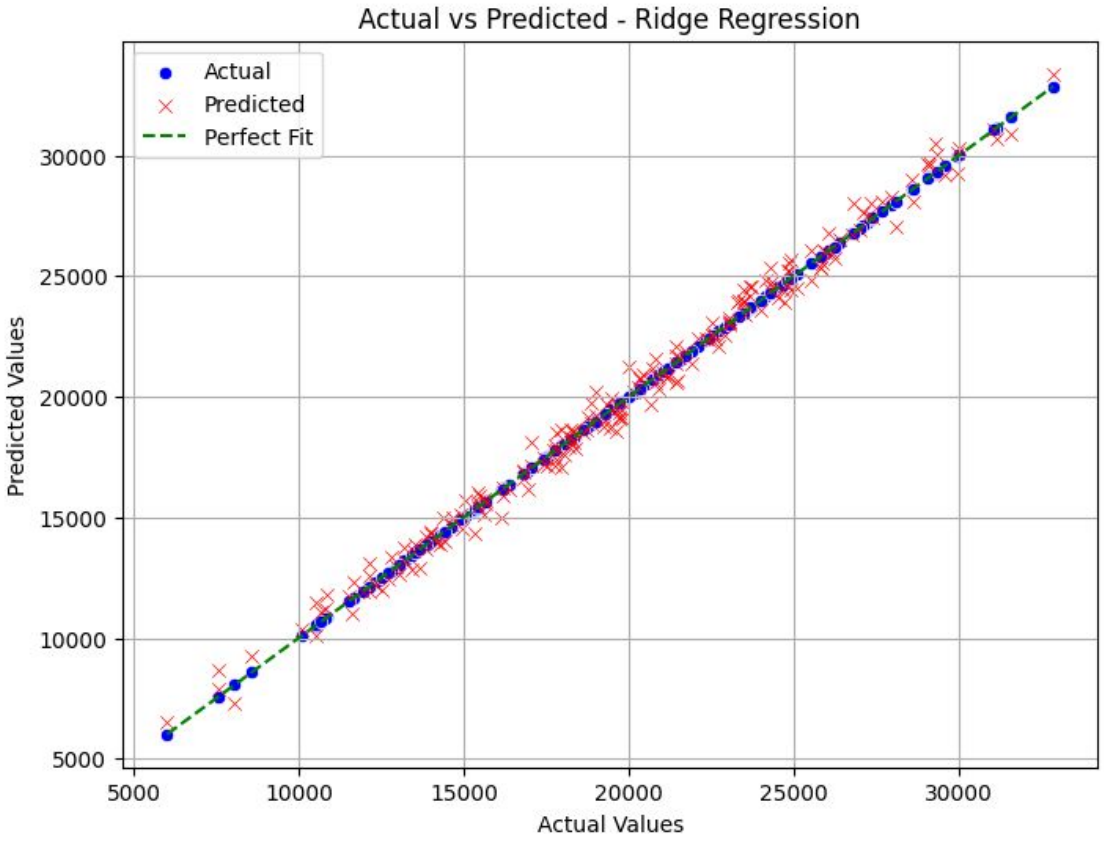
Recommended Features:

1. **Customer_Age** – Older customers might have different claim behaviors.
2. **Annual_Income** – Higher income policyholders may have different claim patterns.
3. **Vehicle_Age_Property_Age** – Older vehicles/properties might have higher or lower claim amounts depending on depreciation and maintenance.
4. **Claim_History** – A strong predictor; past claims can indicate future claim amounts.
5. **Premium_Amount** – Higher premiums might correlate with higher claim payouts.
6. **Risk_Score_Encoded** – A higher risk score might indicate a higher claim amount.
7. **Policy_Type_Auto, Policy_Type_Health, Policy_Type_Life, Policy_Type_Property** – The type of policy directly influences the claim amount.

Select Feature based on SME Recommendation

```
feature = df1_new[['Customer_Age', 'Annual_Income', 'Vehicle_Property_Age',
       'Claim_History', 'Premium_Amount','Policy_Type_Auto', 'Policy_Type_Health', 'Policy_Type_Life',
       'Policy_Type_Property']]
target = df1_new['Claim_Amount']
```

1) Linear Regression



Actual Values

Test Data Results:
Mean Absolute Error (MAE): 420.26977164793294
Mean Squared Error (MSE): 266113.85489287315
Root Mean Squared Error (RMSE): 515.8622441048319
R^2 Score: 0.9914071633673864
Adjusted R^2 Score: 0.9910001342637363

Train Data Results:
Mean Absolute Error (MAE): 394.4371148309234
Mean Squared Error (MSE): 240285.03275418357
Root Mean Squared Error (RMSE): 490.1887725705104
R^2 Score: 0.9938218396955943
Adjusted R^2 Score: 0.9937514555908606

Model seems well-balanced between train and test data.

Model Accuracy is good but Mean Squared Error (MSE): 266057 is High

so use Gradient descent to add weight to reduce error

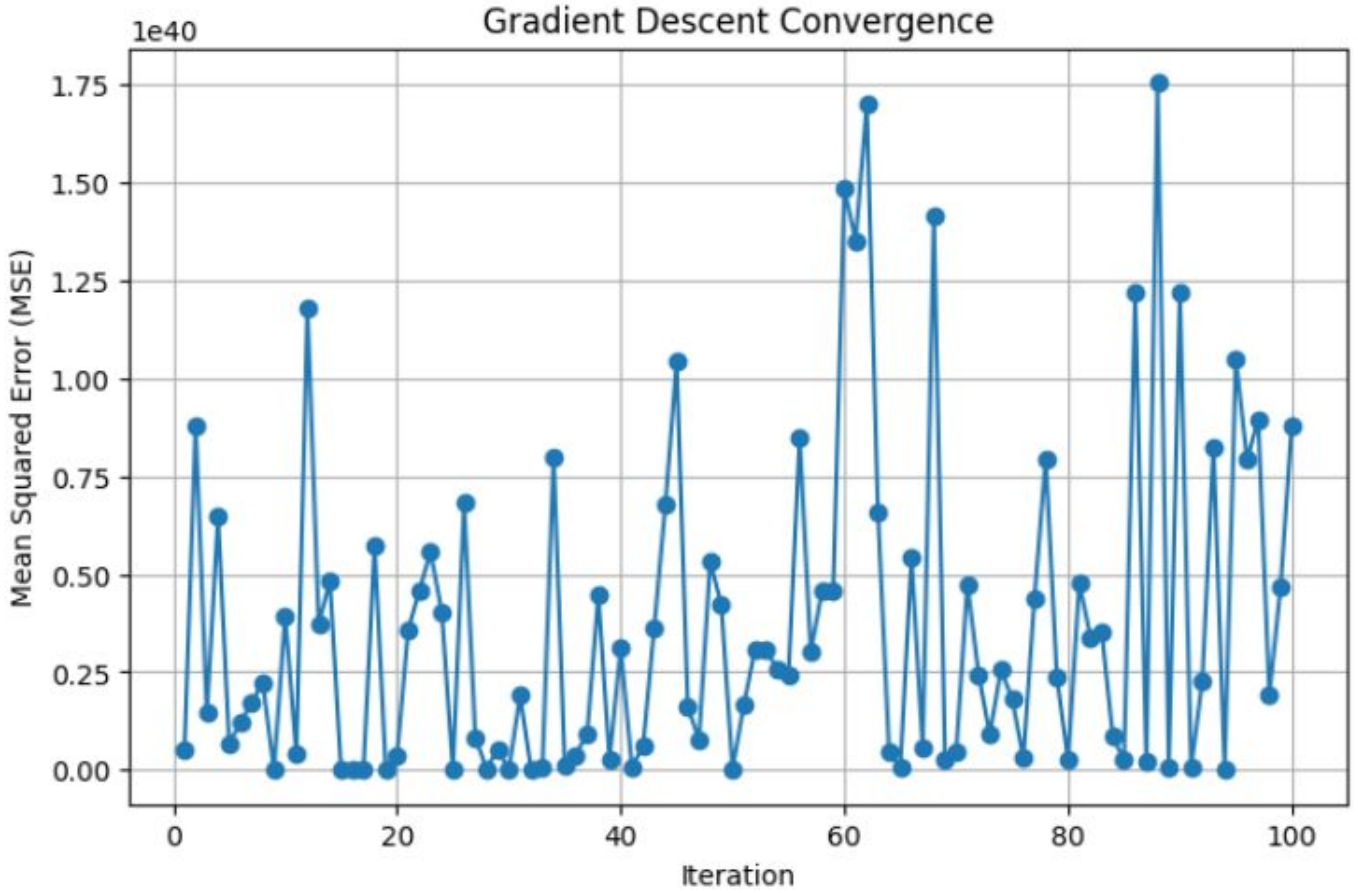
1️⃣ Using Gradient Descent (Manual Optimization)

Instead of using `LinearRegression()`, we'll implement **Gradient Descent** from scratch using `SGDRegressor` (Stochastic Gradient Descent).

2️⃣ Using Regularization (Lasso & Ridge)

- **Ridge Regression (L2 Regularization):** Penalizes large coefficients to prevent overfitting.
- **Lasso Regression (L1 Regularization):** Can shrink some coefficients to **zero**, effectively selecting only the most important features.

🚀 Gradient Descent Model:
Test MSE: 8.800572522906216e+39
Test R² Score: -2.8417115671501897e+32



Using Gradient Descent R2 Score was Droped

U-Shape Not Attained

```
🔥 Ridge Regression (L2):
Test MSE: 266125.34961845167
Test R² Score: 0.9914067922018225
```

So Now try Ridge

All predicted Point was Fitted - Actual Points slightly not fitted

Alpha value now put as 0.5 no changes

now Alpha value now put as 1

```
ridge_model = Ridge(alpha=1)  # α controls regularization strength
ridge_model.fit(x_train, y_train)
```

🔥 Ridge Regression (L2):
Test MSE: 266137.2607424332
Test R² Score: 0.9914064075907224

MSE Value Increased Compared to Previos One

So Now Trying Lasso L1 Regularization

```
⚡ Lasso Regression (L1):
Test MSE: 266103.5938296286
Test R² Score: 0.9914074946978998
```

using Alpha Value 0.1 used - mse value reduced slighlty compare to previos one

so now am trying alpha value 1

```
⚡ Lasso Regression (L1):
Test MSE: 266027.5354558695
Test R² Score: 0.9914099506285785
```

Using Alpha Value 1 is MSE Value is very Slighly Reduced

so now trying Alpha value is 2

```
⚡ Lasso Regression (L1):
Test MSE: 265977.419343764
Test R² Score: 0.9914115688816535
```

Using Alpha value 2 - mse reduced but alpha value increased its stronger penalty

Readucing MSE Errors in Linear Regression try - Decision Tree Regressor

```
Test Data Results:
Mean Absolute Error (MAE): 1537.8707152297713
Mean Squared Error (MSE): 3852827.1528118565
Root Mean Squared Error (RMSE): 1962.8619800719196
R² Score: 0.8755919179362638
Adjusted R² Score: 0.8696989035227184

Train Data Results:
Mean Absolute Error (MAE): 0.0
Mean Squared Error (MSE): 0.0
Root Mean Squared Error (RMSE): 0.0
R² Score: 1.0
Adjusted R² Score: 1.0

Possible Overfitting: Model performs much better on training data than test data.
```

Data Leakage Happen Model Over Trained on Training Data

Avoid Data leakage use validation method for Regression is K-Fold Cross Validation Method

Now Using Hyperparameter Regularization type to reduce overfit

```
dec_reg_model = DecisionTreeRegressor(max_depth=8, min_samples_split=35,
min_samples_leaf=16, random_state=42)
```

```
Test Data Results:
Mean Absolute Error (MAE): 1878.2428075796163
Mean Squared Error (MSE): 5262375.979976702
Root Mean Squared Error (RMSE): 2293.9869180047
R² Score: 0.8300774790041183
Adjusted R² Score: 0.8220285174832608

Train Data Results:
Mean Absolute Error (MAE): 1329.125331271733
Mean Squared Error (MSE): 2852310.941391702
Root Mean Squared Error (RMSE): 1688.8786046935707
R² Score: 0.9266619562943992
Adjusted R² Score: 0.9258264595939556

Possible Overfitting: Model performs much better on training data than test data.
```

Using Ensemble Method for Robust Model

Random Forest Regressor

```
🌲 Random Forest Regression:
Test MSE: 2568197.5556530063
Test R² Score: 0.9170727054219429
Test Adjusted R² Score: 0.9131445704156138

Train Data Results:
Train MSE: 1694319.2290051067
Train R² Score: 0.9564360056735651
Train Adjusted R² Score: 0.9559397070040234
```

Model is Genralized Well but MSE value is High

Reason for MSE High Because of Synthetic dataset

Distance between Actual Point and Predicted Point distance was high

Model Save Both Linear and Random Forest - because linear have Low MSE compare to Random Forest

Linear Have may be Data Leakage Happen so Using Random Forest

```
[ ]
```

```python
import joblib

# 3️⃣ ✅ Save the trained model
joblib.dump(lin_reg_model, "claim_linear_model.pkl")
print("✅ Model saved as 'claim_linear_model.pkl'")


# 3️⃣ ✅ Save the trained model
joblib.dump(rf_model, "claim_rf_model.pkl")
print("✅ Model saved as 'claim_rf_model.pkl'")
```

```
✅ Model saved as 'claim_linear_model.pkl'
✅ Model saved as 'claim_rf_model.pkl'
```

XG Boosting Method

```
⚡ XGBoost Regression:
Test MSE: 591619.2007502405
Test R² Score: 0.9808965709703064
Test Adjusted R² Score: 0.9799916717004787

Train Data Results:
Train MSE: 80955.09585495791
Train R² Score: 0.9979184989014188
Train Adjusted R² Score: 0.9978947855977641
```

```
[ ]  1 joblib.dump(xgb_model, "claim_xg_model.pkl")
     2 print("✅ Model saved as 'claim_xg_model.pkl'")
```

```
✅ Model saved as 'claim_xg_model.pkl'
```

4. Claim Amount Prediction

df1 - Claim Regressor Future Prediction

```
[ ]  1 ######################################################################
```

```
[ ]  1 import joblib
     2
     3 len_model_loaded = joblib.load("claim_linear_model.pkl")
     4 rf_model_loaded = joblib.load("claim_rf_model.pkl")
```

```
     1 len_model_loaded
```

```
LinearRegression
LinearRegression()
```

```
[ ]  1 len_model_loaded.feature_names_in_
```

```
array(['Customer_Age', 'Annual_Income', 'Vehicle_Property_Age',
       'Claim_History', 'Premium_Amount', 'Policy_Type_Auto',
       'Policy_Type_Health', 'Policy_Type_Life', 'Policy_Type_Property'],
      dtype=object)
```

```
[ ]
```

```python
# Load the saved linear regression model
len_model_loaded = joblib.load("claim_linear_model.pkl")
print("Model loaded successfully!")

# Example: New data for prediction (ensure it's preprocessed the same way)
new_data = pd.DataFrame({
    'Customer_Age': [56],
    'Annual_Income': [92292],
    'Vehicle_Property_Age': [25],
    'Claim_History': [1],
    'Premium_Amount': [2400],
    'Policy_Type_Auto': [0],
    'Policy_Type_Health': [0],
    'Policy_Type_Life': [0],
    'Policy_Type_Property': [1]
})

# Predict the output
predicted_value = len_model_loaded.predict(new_data)

print(f"Predicted Claim Amount Value: {predicted_value[0]}")

```

```
Model loaded successfully!
Predicted Claim Amount Value: 15455.868316705366
```

Predicted Claim Amount Value Almost Same show nearly but not exactly almost accurate

so already saved Random Forest i will try that type

Model loaded successfully!
Predicted Claim Amount Value: 14644.636848449407

Linear Regression Show Better Prediction Compare to Random Forest

Now Try Decision Tree Regressor

```
Model loaded successfully!
Predicted Claim Amount Value: 14484.065635258832
```

XG Model

```
Model loaded successfully!
Predicted Claim Amount Value: 14940.552734375
```

so i finalise Linear Regressor Model

Linear Regressor Model Compare Best to Other Models

Because of Linear Regressor have Low MSE Value

Streamlit Application

1) **Risk Classification**
2) **Claim Prediction**
3) **Fraud Detection**
4) **Claim Amount Prediction**

Required Libraries

```
import streamlit as st
import joblib
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model
import pickle
```

Connect Loaded Models

```
# Load models and encoders
logi_model = joblib.load('/content/logi_model.pkl')
logis_model = joblib.load('/content/logistic_regression_model.pkl')
loaded_model = load_model('/content/my_model.h5')
```

Encoder Loaded File : For Matching Decoded Words

```
with open('/content/onehot_encoder.pkl', 'rb') as file:
    onehot_encoder = pickle.load(file)
```

Client Entered Unscaled Value: But Dataset has Scaled
So Convert Unscaled value to scaled then match the model

```
# Read dataset for scaling
df1 = pd.read_csv('/content/df1-synthetic_insurance_dataset_fixed.csv')

# Get min and max values for scaling
min_values = df1[['Annual_Income', 'Premium_Amount', 'Claim_Amount']].min()
max_values = df1[['Annual_Income', 'Premium_Amount', 'Claim_Amount']].max()

# Fit the scaler
min_max_values = np.array([min_values, max_values])
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(min_max_values)
```

Insurance Fraud Risk Prediction App

Claim Prediction | Fraud Detection | Advanced Fraud Risk Model

Claim Prediction

Customer Age

30 − +

Annual Income (Raw Value)

50000.00 − +

Raw Values to Scaled Values

Annual_Income (Raw Value)

50000.00 − +

Premium_Amount (Raw Value)

2300.00 − +

Claim_Amount (Raw Value)

30000.00 − +

Raw and Scaled Values

Raw Annual Income: 50000.000000, Scaled: 0.166112

Raw Premium Amount: 2300.000000, Scaled: 0.400195

Raw Claim Amount: 30000.000000, Scaled: 0.600858

Decoded to Encoded Value

Policy_Type

Auto

Gender

Female

Encoded Values

Encoded Policy Type and Gender: [[1. 0. 0. 0. 1. 0. 0.]]

1) Claim Prediction

Enter Unscaled Raw Value

Annual Income (Raw Value)

50000.00 − +

Premium Amount (Raw Value)

1000.00 − +

Enter Categorical Values it automatically Encoded verify with Encoded Loaded Model

Policy Type

Auto

Gender

Female

Predicted class: 0

Risk Score: Filed Claim 2 or Below / did not file a claim Single time (0)

Class probabilities: [[0.98551587 0.01448413]]

- **Prediction Work Based on Probabilities of Values -**
- **0.9 is Class No Claim and 0.01 is Class Already Filed Claim**

2) Fraud Detection

Fraud Detection

Vehicle/Property Age

7 – +

Premium Amount (Raw Value)

2500.00 – +

Claim Amount (Raw Value)

35000.00 – +

Claim History

3 – +

Predict Simple Fraud

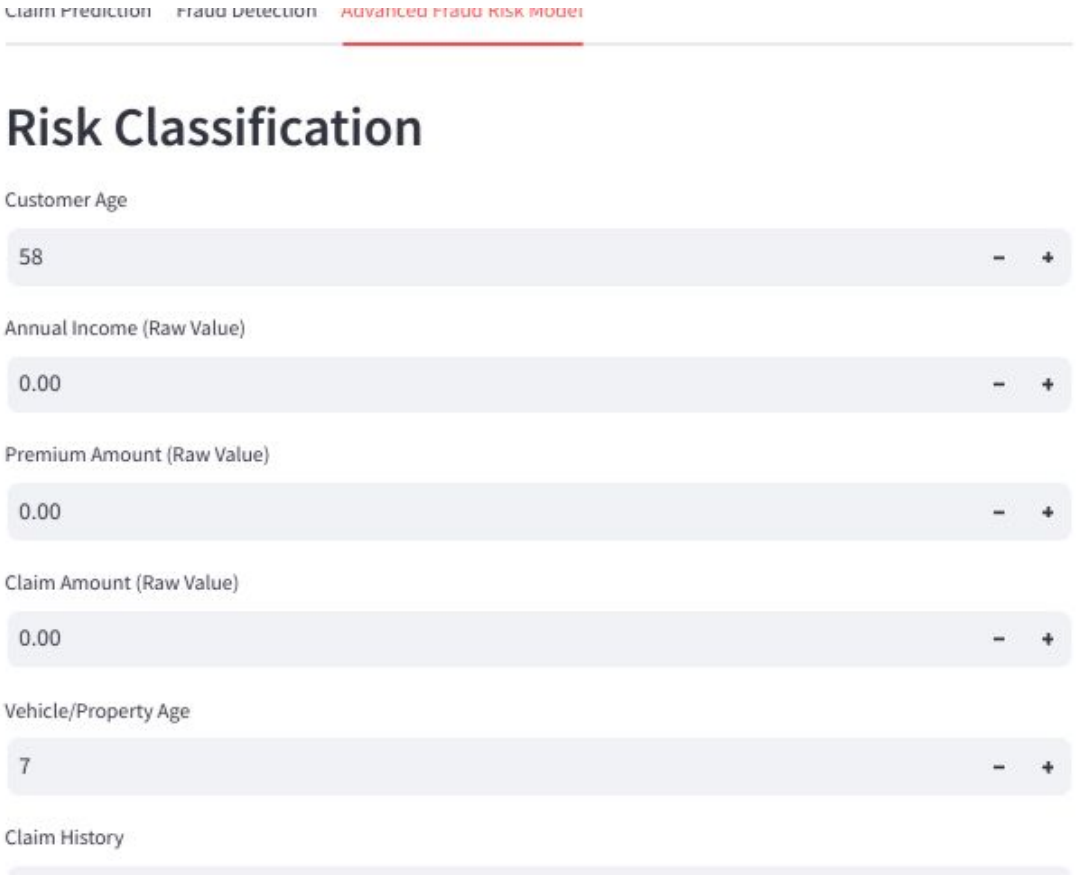
Predicted class: 1

Risk Score: Fraud (1)

Class probabilities: [[0.4861395 0.5138605]]

Based on Claim Amount & Claim History - Compare to Property/Vehicle Age

3) Risk Classification



Vehicle/Property Age

7 − +

Claim History

1 − +

Fraudulent Claim

0 − +

Policy Type

Auto

Gender

Female

Predict Advanced Fraud Risk

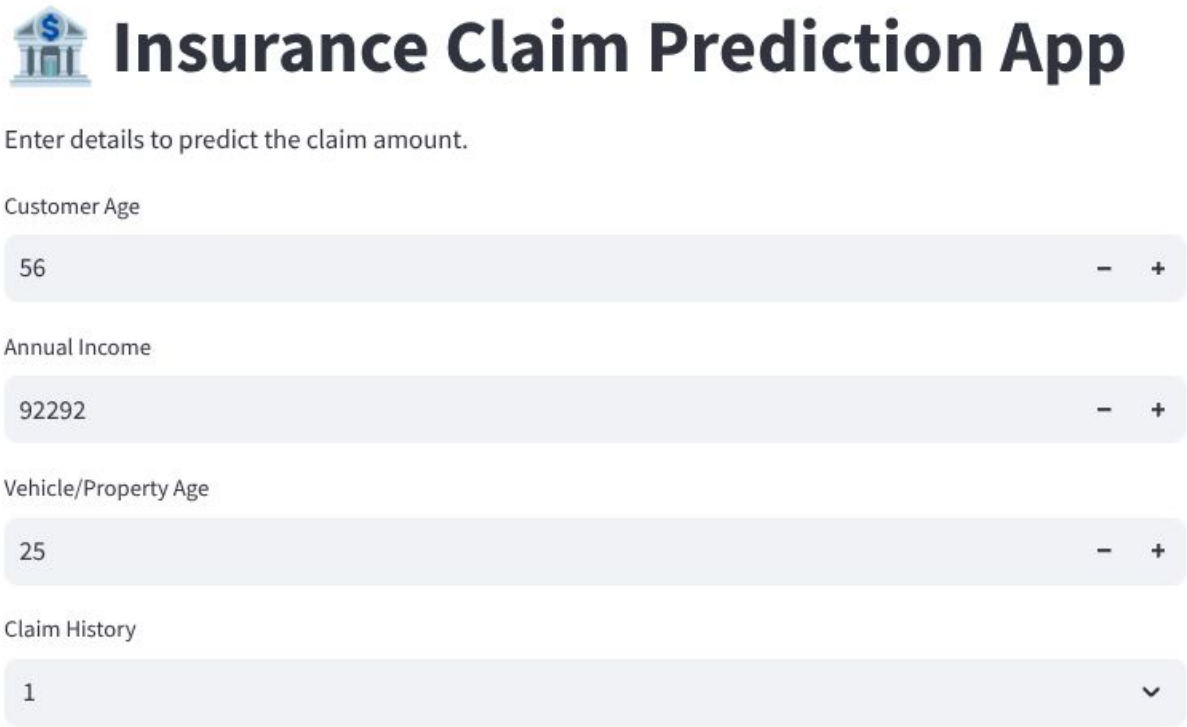
Predict Advanced Fraud Risk

Predicted class: 2

Risk Score: High (2)

Class probabilities: [[3.461611e-06 2.545815e-02 9.745384e-01]]

4) Claim Amount Prediction



Claim History

1

Premium Amount

2400

- [ ] Auto Policy
- [ ] Health Policy
- [ ] Life Policy
- [x] Property Policy

Predict Claim Amount

Predicted Claim Amount: $15,455.87

Conclusion:

This project provided valuable hands-on experience in building a complete end-to-end insurance fraud risk prediction app using Streamlit and a TensorFlow Keras model. Key skills gained include data preprocessing, feature scaling, one-hot encoding, and working with multiple machine learning models like logistic regression and deep learning. It also strengthened understanding of model deployment, user interface design, and handling real-world data for risk assessment. Overall, this project enhanced both technical and practical knowledge in fraud detection and predictive analytics

Thank You