```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno  # Missing Value Visualization
from scipy import stats
from sklearn.feature_selection import mutual_info_regression
```

```
# Ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv("/content/drive/MyDrive/GUVI Capstone projects/Project3/train.csv")
```

```
# Display first few rows
df.head()
```

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	Health Score	Location	...	Previous Claims	Vehicle Age	Credit Score	Insurance Duration
0	0	19.0	Female	10049.0	Married	1.0	Bachelor's	Self-Employed	22.598761	Urban	...	2.0	17.0	372.0	5.0
1	1	39.0	Female	31678.0	Divorced	3.0	Master's	NaN	15.569731	Rural	...	1.0	12.0	694.0	2.0
2	2	23.0	Male	25602.0	Divorced	3.0	High School	Self-Employed	47.177549	Suburban	...	1.0	14.0	NaN	3.0
3	3	21.0	Male	141855.0	Married	2.0	Bachelor's	NaN	10.938144	Rural	...	1.0	0.0	367.0	1.0
4	4	21.0	Male	39651.0	Single	1.0	Bachelor's	Self-Employed	20.376094	Rural	...	0.0	8.0	598.0	4.0

5 rows × 21 columns

```
# Check Dataset Overview
df.info()
```

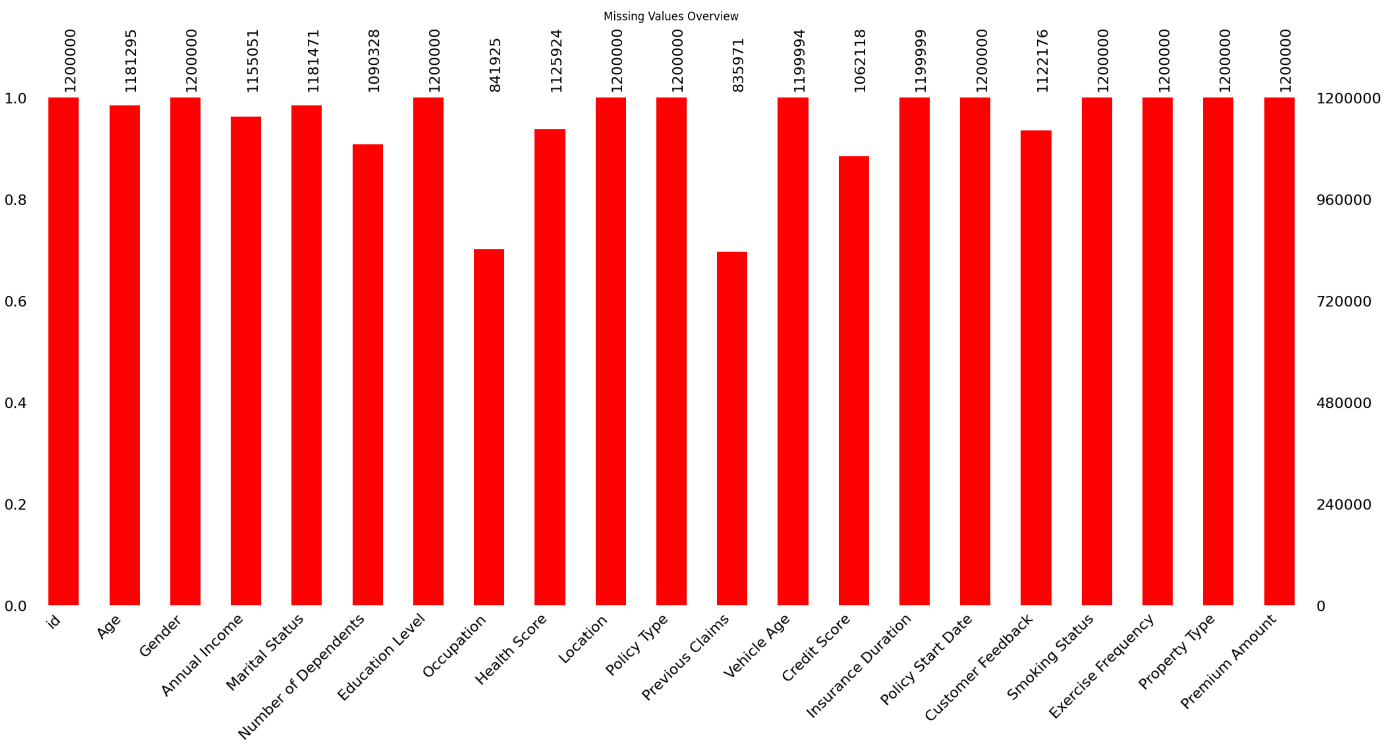
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200000 entries, 0 to 1199999
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   id                    1200000 non-null  int64
 1   Age                   1181295 non-null  float64
 2   Gender                1200000 non-null  object
 3   Annual Income         1155051 non-null  float64
 4   Marital Status        1181471 non-null  object
 5   Number of Dependents  1090328 non-null  float64
 6   Education Level       1200000 non-null  object
 7   Occupation            841925 non-null   object
 8   Health Score          1125924 non-null  float64
 9   Location              1200000 non-null  object
 10  Policy Type           1200000 non-null  object
 11  Previous Claims       835971 non-null   float64
 12  Vehicle Age           1199994 non-null  float64
 13  Credit Score          1062118 non-null  float64
 14  Insurance Duration    1199999 non-null  float64
 15  Policy Start Date     1200000 non-null  object
 16  Customer Feedback     1122176 non-null  object
 17  Smoking Status        1200000 non-null  object
 18  Exercise Frequency    1200000 non-null  object
 19  Property Type         1200000 non-null  object
 20  Premium Amount        1200000 non-null  float64
dtypes: float64(9), int64(1), object(11)
memory usage: 192.3+ MB
```

```
df.describe(include="all")
```

Show hidden output

```
# Check for Missing Values (Visualization)
plt.figure(figsize=(10,6))
```

```python
msno.bar(df, color='red')
plt.xticks(rotation=90)
plt.title("Missing Values Overview")
plt.show()
```




```python
# Show Percentage of Missing Values
missing_percent = (df.isnull().sum() / len(df)) * 100
print("Missing Values (%):\n", missing_percent[missing_percent > 0].sort_values(ascending=False))
```

```
Missing Values (%):
 Previous Claims         30.335750
Occupation              29.839583
Credit Score            11.490167
Number of Dependents     9.139333
Customer Feedback        6.485333
Health Score             6.173000
Annual Income            3.745750
Age                      1.558750
Marital Status           1.544083
Vehicle Age              0.000500
Insurance Duration       0.000083
dtype: float64
```

```python
# Convert 'Policy Start Date' to datetime
df['Policy Start Date'] = pd.to_datetime(df['Policy Start Date'])
df['Policy Start Year'] = df['Policy Start Date'].dt.year
```

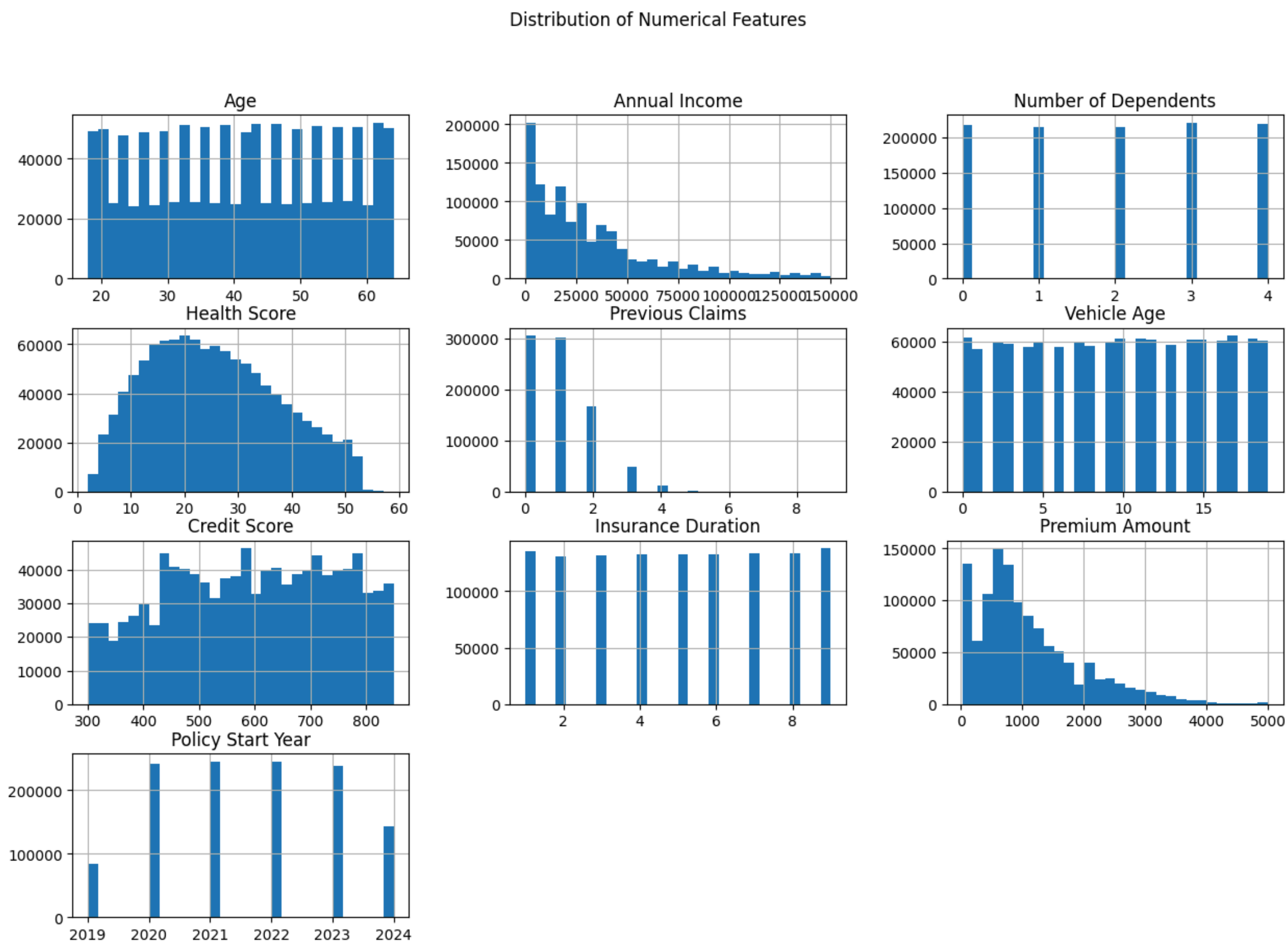
```python
# Drop Irrelevant Columns
df.drop(columns=['id', 'Policy Start Date'], inplace=True)
```

```python
# Identify Numerical & Categorical Features
num_cols = df.select_dtypes(include=['number']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

print("Numerical Columns:", num_cols)
print("Categorical Columns:", cat_cols)
```

```
Numerical Columns: ['Age', 'Annual Income', 'Number of Dependents', 'Health Score', 'Previous Claims', 'Vehicle Age', 'Credit Score', 'I
Categorical Columns: ['Gender', 'Marital Status', 'Education Level', 'Occupation', 'Location', 'Policy Type', 'Customer Feedback', 'Smok
```

```
for col in num_cols:
    df[col].fillna(df[col].mean(), inplace=True)
for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)


# Univariate Analysis - Histograms & Boxplots for Numerical Features
plt.figure(figsize=(15,10))
df[num_cols].hist(bins=30, figsize=(15,10))
plt.suptitle("Distribution of Numerical Features")
plt.show()
```

```
<Figure size 1500x1000 with 0 Axes>
```




```
for col in num_cols:
    plt.figure(figsize=(6, 3))
    sns.boxplot(y=df[col])
    plt.title(f"Boxplot of {col}")
    plt.show()
```

Show hidden output

categorical vs numerical

```
results = {}  # Store results in a dictionary

for cat_col in cat_cols:
    if df[cat_col].nunique() < 10:
        for num_col in num_cols:
            grouped = df.groupby(cat_col)[num_col].describe()
            results[f'{cat_col} vs {num_col}'] = grouped
# Display results
for key, value in results.items():
    print(f"\n{key}:\n")
    print(value)
```

```
Gender vs Age:

           count       mean        std   min   25%        50%   75%   max
Gender
Female  597429.0  41.142665  13.437654  18.0  30.0  41.145563  53.0  64.0
Male    602571.0  41.148436  13.430404  18.0  30.0  41.145563  53.0  64.0

Gender vs Annual Income:

           count          mean           std  min     25%      50%      75%  \
Gender
Female  597429.0  32774.818487  31580.698986  1.0  8607.0  24997.0  43945.0
Male    602571.0  32715.869662  31561.522478  2.0  8695.0  24986.0  43922.0

             max
Gender
Female  149996.0
Male    149997.0

Gender vs Number of Dependents:

           count      mean       std  min  25%  50%  75%  max
Gender
Female  597429.0  2.008803  1.351920  0.0  1.0  2.0  3.0  4.0
Male    602571.0  2.011055  1.350124  0.0  1.0  2.0  3.0  4.0

Gender vs Health Score:

           count       mean        std       min        25%        50%  \
Gender
Female  597429.0  25.573313  11.826661  2.012237  16.417788  25.613908
Male    602571.0  25.654156  11.814862  2.053458  16.634376  25.613908

              75%        max
Gender
Female  33.688314  58.975914
Male    33.838254  58.569689

Gender vs Previous Claims:

           count      mean       std  min  25%  50%       75%  max
Gender
Female  597429.0  1.002878  0.820700  0.0  0.0  1.0  1.002689  8.0
Male    602571.0  1.002502  0.819961  0.0  0.0  1.0  1.002689  9.0

Gender vs Vehicle Age:

           count      mean       std  min  25%   50%   75%   max
Gender
Female  597429.0  9.564384  5.771850  0.0  5.0  10.0  15.0  19.0
Male    602571.0  9.575347  5.780458  0.0  5.0  10.0  15.0  19.0

Gender vs Credit Score:

           count        mean         std    min    25%        50%    75%  \
Gender
Female  597429.0  592.817033  141.217972  300.0  483.0  592.92435  706.0
```

categorical vs categorical relationship

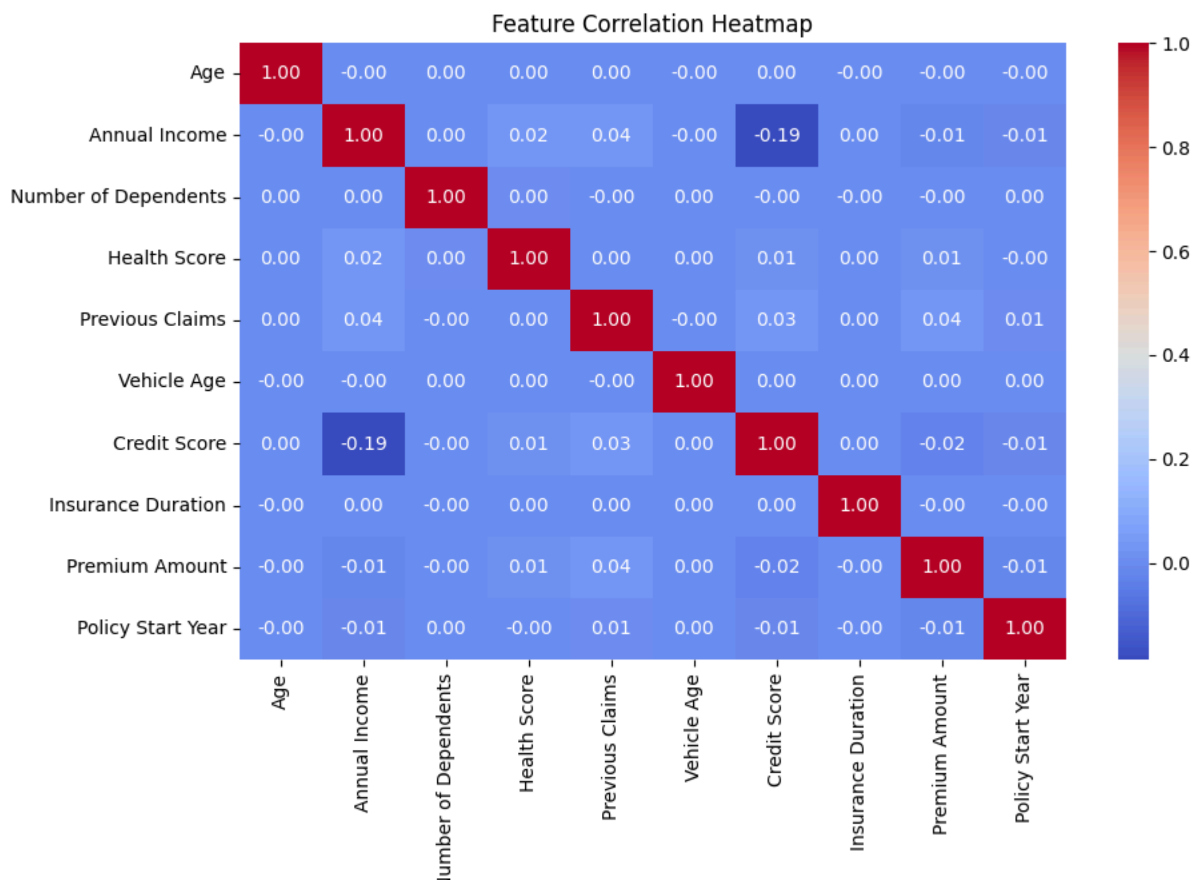
```
for i in range(len(cat_cols)):
    for j in range(i + 1, len(cat_cols)):
        if df[cat_cols[i]].nunique() < 10 and df[cat_cols[j]].nunique() < 10: #Avoid huge tables
            contingency_table = pd.crosstab(df[cat_cols[i]], df[cat_cols[j]])
            #print(f"Contingency Table: {categorical_features[i]} vs {categorical_features[j]}\n{contingency_table}\n")
            # Creating barplot
            barplot = contingency_table.plot.bar(rot=0)
```

Show hidden output

```
# Outlier Detection using IQR
def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers.shape[0]

outliers_summary = {col: detect_outliers(df, col) for col in num_cols}
print("\nOutlier Counts:", outliers_summary)
```

```
Outlier Counts: {'Age': 0, 'Annual Income': 70466, 'Number of Dependents': 0, 'Health Score': 0, 'Previous Claims': 62066, 'Vehicle Age'
```

```
# Correlation Heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df[num_cols].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



ANOVA test

The ANOVA test indicates that there is no statistically significant difference between the means of the groups being compared.

> 0.05 - no signi difference

```
ANOVO=pd.DataFrame(columns=['Feature', 'F-statistic', 'p-value'])
for feature in cat_cols:
  if feature in df.columns and 'Premium Amount' in df.columns:
      groups = [df['Premium Amount'][df[feature] == category] for category in df[feature].unique()]
      f_statistic, p_value = stats.f_oneway(*groups)
      ANOVO.loc[-1] = [feature,float("{:.5f}".format(f_statistic)),float("{:.5f}".format(p_value))]  # adding a row
      ANOVO.index = ANOVO.index + 1  # shifting index
      ANOVO=ANOVO.sort_index()
      #print(f"ANOVA {feature} vs Premium Amount: F-statistic={f_statistic}, p-value={p_value}")
ANOVO.sort_values(by='p-value', ascending=False,ignore_index=True)
```

	Feature	F-statistic	p-value
0	Gender	0.03110	0.86002
1	Smoking Status	0.03178	0.85850
2	Exercise Frequency	0.48306	0.69405
3	Policy Type	0.47091	0.62443
4	Location	0.67641	0.50844
5	Property Type	1.05118	0.34952
6	Education Level	1.14511	0.32925
7	Marital Status	8.21588	0.00027
8	Occupation	8.47766	0.00021
9	Customer Feedback	32.86461	0.00000

```
for feature in num_cols:
    if feature in df.columns:
        try:
            # Bin the numerical feature
            df[f'{feature}_binned'] = pd.cut(df[feature], bins=5, labels=False, include_lowest=True)
            groups = [df["Premium Amount"][df[f'{feature}_binned'] == i] for i in range(5)]
            f_statistic, p_value = stats.f_oneway(*groups)
            ANOVO.loc[len(ANOVO)] = [feature, float("{:.5f}".format(f_statistic)),float("{:.5f}".format(p_value))]
        except ValueError as e:
            print(f"Error binning {feature}: {e}")
        finally:
            if f'{feature}_binned' in df.columns:
                df.drop(columns=[f'{feature}_binned'], inplace=True)
ANOVO.sort_values(by='p-value', ascending=False, ignore_index=True)
```

	Feature	F-statistic	p-value
0	Gender	3.110000e-02	0.86002
1	Smoking Status	3.178000e-02	0.85850
2	Exercise Frequency	4.830600e-01	0.69405
3	Policy Type	4.709100e-01	0.62443
4	Location	6.764100e-01	0.50844
5	Property Type	1.051180e+00	0.34952
6	Education Level	1.145110e+00	0.32925
7	Insurance Duration	1.177480e+00	0.31838
8	Age	2.668210e+00	0.03050
9	Vehicle Age	3.660880e+00	0.00550
10	Marital Status	8.215880e+00	0.00027
11	Occupation	8.477660e+00	0.00021
12	Annual Income	3.058429e+02	0.00000
13	Number of Dependents	1.465342e+01	0.00000
14	Health Score	1.427919e+02	0.00000
15	Previous Claims	6.335913e+02	0.00000
16	Credit Score	3.601030e+02	0.00000
17	Customer Feedback	3.286461e+01	0.00000
18	Premium Amount	2.537946e+06	0.00000
19	Policy Start Year	5.927497e+01	0.00000

Chi- Square

The Chi-Square test indicates that there is a statistically significant association between the two categorical variables being compared.

> 0.05 - no signi assosciation

```
# Chi-Square (Example: Gender vs. Smoking Status)
chi_df=pd.DataFrame(columns=["Feature1","Feature2","chi2","p-value"])
for i in range(len(cat_cols)):
    for j in range(i + 1, len(cat_cols)):
        if df[cat_cols[i]].nunique() < 10 and df[cat_cols[j]].nunique() < 10:
          contingency = pd.crosstab(df[cat_cols[i]], df[cat_cols[j]])
          chi2, p, dof, expected = stats.chi2_contingency(contingency)
          chi_df.loc[len(chi_df)] = [cat_cols[i], cat_cols[j], float("{:.5f}".format(chi2)),float("{:.5f}".format(p))]
          #print(f"Chi-Square: chi2={chi2}, p-value={p}")
```

```
chi_df.shape
```

```
(45, 4)
```

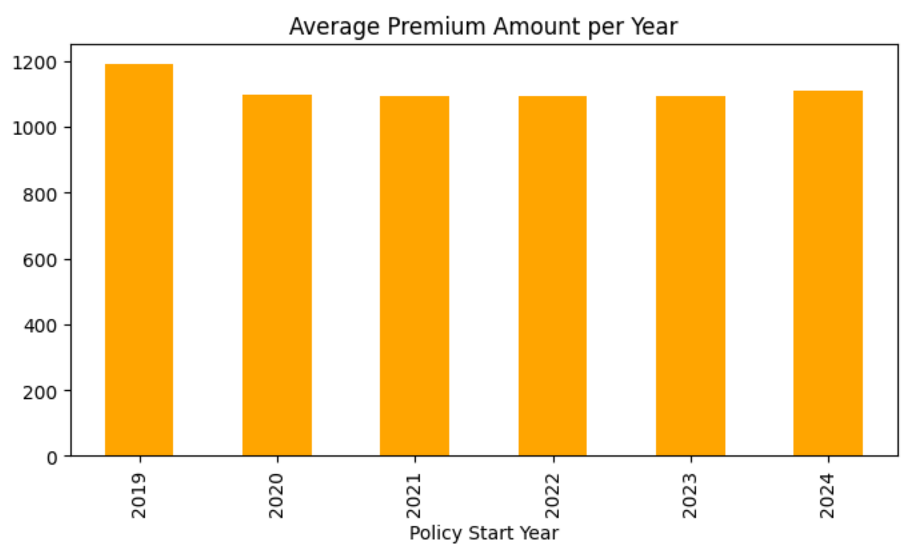
```
chi_df.sort_values(by='p-value', ascending=False, ignore_index=True).head(10)
```

	Feature1	Feature2	chi2	p-value
0	Marital Status	Exercise Frequency	1.80472	0.93675
1	Customer Feedback	Smoking Status	0.43219	0.80566
2	Location	Property Type	1.82081	0.76867
3	Marital Status	Policy Type	1.94153	0.74651
4	Location	Exercise Frequency	3.55689	0.73639
5	Gender	Education Level	1.44928	0.69402
6	Education Level	Customer Feedback	4.01865	0.67415
7	Occupation	Location	3.05227	0.54912
8	Education Level	Occupation	5.15434	0.52418
9	Gender	Exercise Frequency	2.33777	0.50532

```
# Feature vs Target (Premium Amount)
sample_df=df.sample(1000)
for col in num_cols:
    if col != 'Premium Amount':
        plt.figure(figsize=(6, 3))
        sns.scatterplot(x=sample_df[col], y=sample_df['Premium Amount'])
        plt.title(f"{col} vs Premium Amount")
        plt.show()
```

Show hidden output

```
# Time-Based Analysis
df.groupby('Policy Start Year')['Premium Amount'].mean().plot(kind='bar', figsize=(8, 4), color='orange')
plt.title("Average Premium Amount per Year")
plt.show()
```

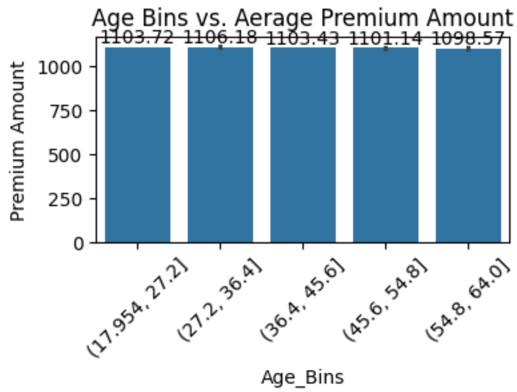


Age and Premium Amount

How does "Age" influence "Premium Amount"? older customers tend to pay higher or lower premiums

```
sample_data=df.copy()
```

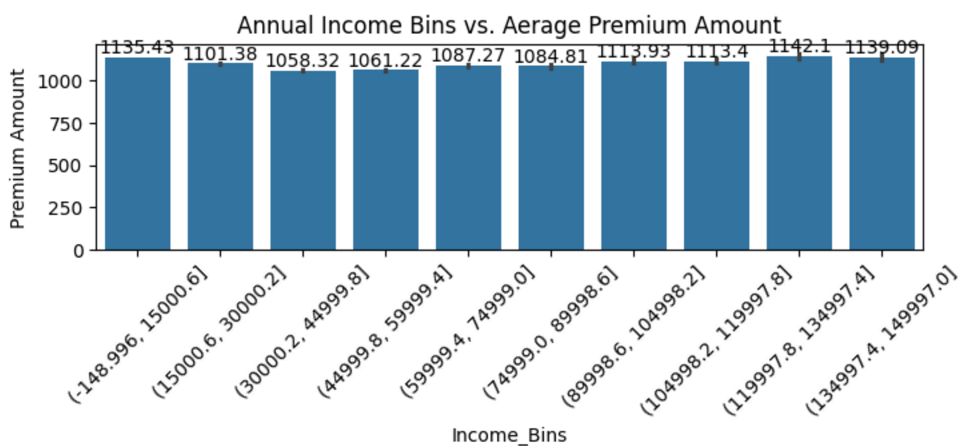
```
sample_data['Age_Bins'] = pd.cut(sample_data['Age'], bins=5)

plt.figure(figsize=(4,2))
ax=sns.barplot(x='Age_Bins', y='Premium Amount', data=sample_data)
ax.bar_label(ax.containers[0], fontsize=10)
plt.title('Age Bins vs. Aerage Premium Amount')
plt.xticks(rotation=45)
plt.show()
```



What is the relationship between "Annual Income" and "Premium Amount"?

This examines if income level correlates with premium costs.

```
sample_data['Income_Bins'] = pd.cut(sample_data['Annual Income'], bins=10)

plt.figure(figsize=(8, 2))
ax=sns.barplot(x='Income_Bins', y='Premium Amount', data=sample_data)
ax.bar_label(ax.containers[0], fontsize=10)
plt.title('Annual Income Bins vs. Aerage Premium Amount')
plt.xticks(rotation=45)
plt.show()
```

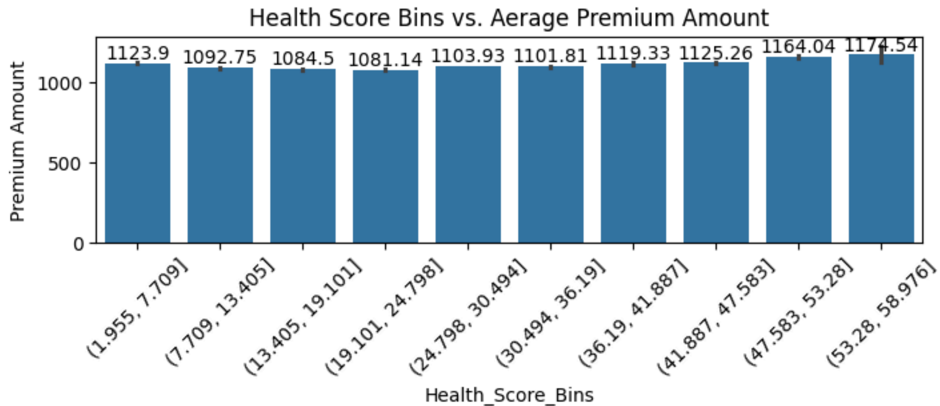



```
print(sample_data["Annual Income"].min(),sample_data["Annual Income"].max())
```

Does "Health Score" have a significant impact on "Premium Amount"?

This explores if healthier individuals get better premium rates.

```
sample_data['Health_Score_Bins'] = pd.cut(sample_data['Health Score'], bins=10)

plt.figure(figsize=(8, 2))
ax=sns.barplot(x='Health_Score_Bins', y='Premium Amount', data=sample_data)
ax.bar_label(ax.containers[0], fontsize=10)
plt.title('Health Score Bins vs. Aerage Premium Amount')
plt.xticks(rotation=45)
plt.show()
```



Health Score Distribution by Policy Type

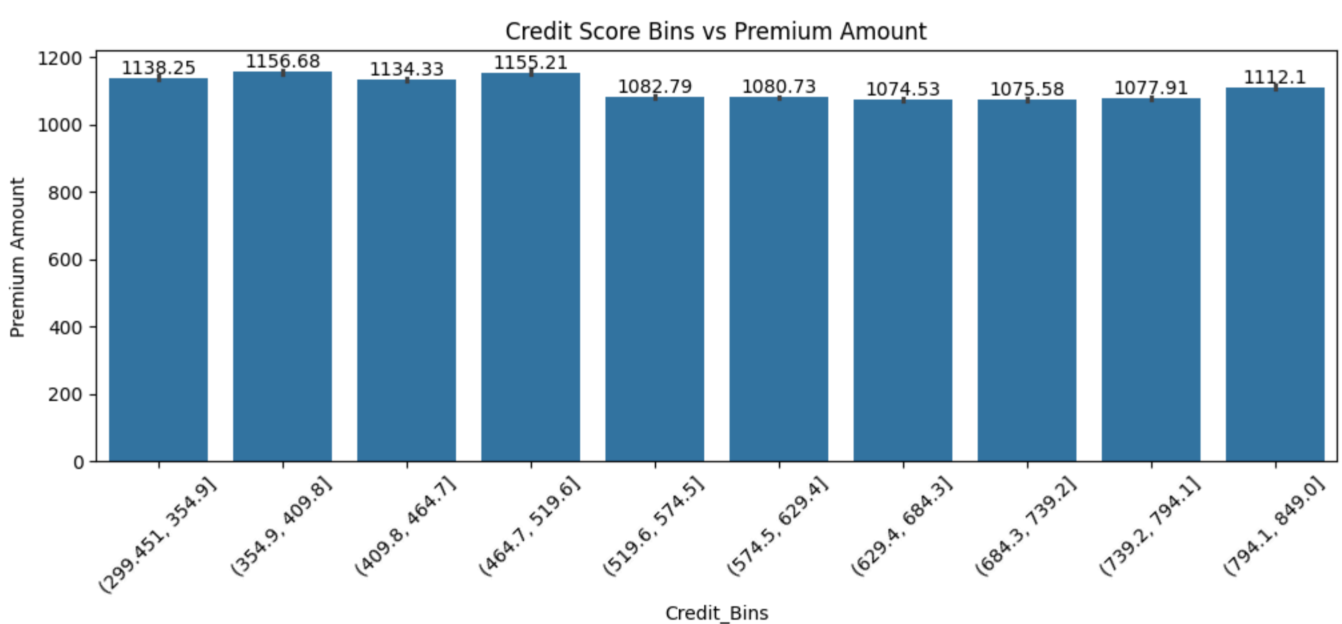
```
sample_data["Health Score"].groupby([sample_data["Policy Type"]]).describe()
```

Policy Type	count	mean	std	min	25%	50%	75%	max
Basic	398554.0	25.650416	11.817550	2.024415	16.569504	25.613908	33.837383	58.975914
Comprehensive	399600.0	25.609021	11.825234	2.012237	16.539119	25.613908	33.764905	58.569689
Premium	401846.0	25.582558	11.819551	2.056559	16.549517	25.613908	33.694463	58.886035

Is there a correlation between "Credit Score" and "Premium Amount"?

This investigates if creditworthiness affects premium calculations.

```
sample_data['Credit_Bins'] = pd.cut(sample_data['Credit Score'], bins=10)
plt.figure(figsize=(12,4))
ax=sns.barplot(x='Credit_Bins', y = 'Premium Amount', data = sample_data)
ax.bar_label(ax.containers[0], fontsize=10)
plt.title("Credit Score Bins vs Premium Amount")
plt.xticks(rotation=45)
plt.show()
```



Premium Amount by Gender and Smoking Status

```
plt.figure(figsize=(4, 2))
ax=sns.barplot(x='Gender', y='Premium Amount', hue='Smoking Status', data=df)
plt.title('Premium Amount by Gender and Smoking Status')
plt.show()
```