```
# necessary imports

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')

plt.style.use('ggplot')
```

```
df = pd.read_csv('/content/insurance_claims.csv')
```

```
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umb
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	

5 rows × 40 columns

```
# we can see some missing values denoted by '?' so lets replace missing values with np.nan

df.replace('?', np.nan, inplace = True)
```

```
df.describe()
```

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	1000.000000	100
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150	1.101000e+06	501214.488000	2512
std	115.113174	9.140287	257063.005276	611.864673	244.167395	2.297407e+06	71701.610941	2787
min	0.000000	19.000000	100804.000000	500.000000	433.330000	-1.000000e+06	430104.000000	
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500	0.000000e+00	448404.500000	
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000	0.000000e+00	466445.500000	
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000	0.000000e+00	603251.000000	5102
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000	1.000000e+07	620962.000000	1005

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_number                1000 non-null   int64
 3   policy_bind_date             1000 non-null   object
 4   policy_state                 1000 non-null   object
 5   policy_csl                   1000 non-null   object
 6   policy_deductable            1000 non-null   int64
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64
 9   insured_zip                  1000 non-null   int64
 10  insured_sex                  1000 non-null   object
 11  insured_education_level      1000 non-null   object
 12  insured_occupation           1000 non-null   object
 13  insured_hobbies              1000 non-null   object
 14  insured_relationship         1000 non-null   object
 15  capital-gains                1000 non-null   int64
 16  capital-loss                 1000 non-null   int64
 17  incident_date                1000 non-null   object
```

```
 18  incident_type               1000 non-null   object
 19  collision_type              822 non-null    object
 20  incident_severity           1000 non-null   object
 21  authorities_contacted       909 non-null    object
 22  incident_state              1000 non-null   object
 23  incident_city               1000 non-null   object
 24  incident_location           1000 non-null   object
 25  incident_hour_of_the_day    1000 non-null   int64
 26  number_of_vehicles_involved 1000 non-null   int64
 27  property_damage             640 non-null    object
 28  bodily_injuries             1000 non-null   int64
 29  witnesses                   1000 non-null   int64
 30  police_report_available     657 non-null    object
 31  total_claim_amount          1000 non-null   int64
 32  injury_claim                1000 non-null   int64
 33  property_claim              1000 non-null   int64
 34  vehicle_claim               1000 non-null   int64
 35  auto_make                   1000 non-null   object
 36  auto_model                  1000 non-null   object
 37  auto_year                   1000 non-null   int64
 38  fraud_reported              1000 non-null   object
 39  _c39                        0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

Data Pre-Processing

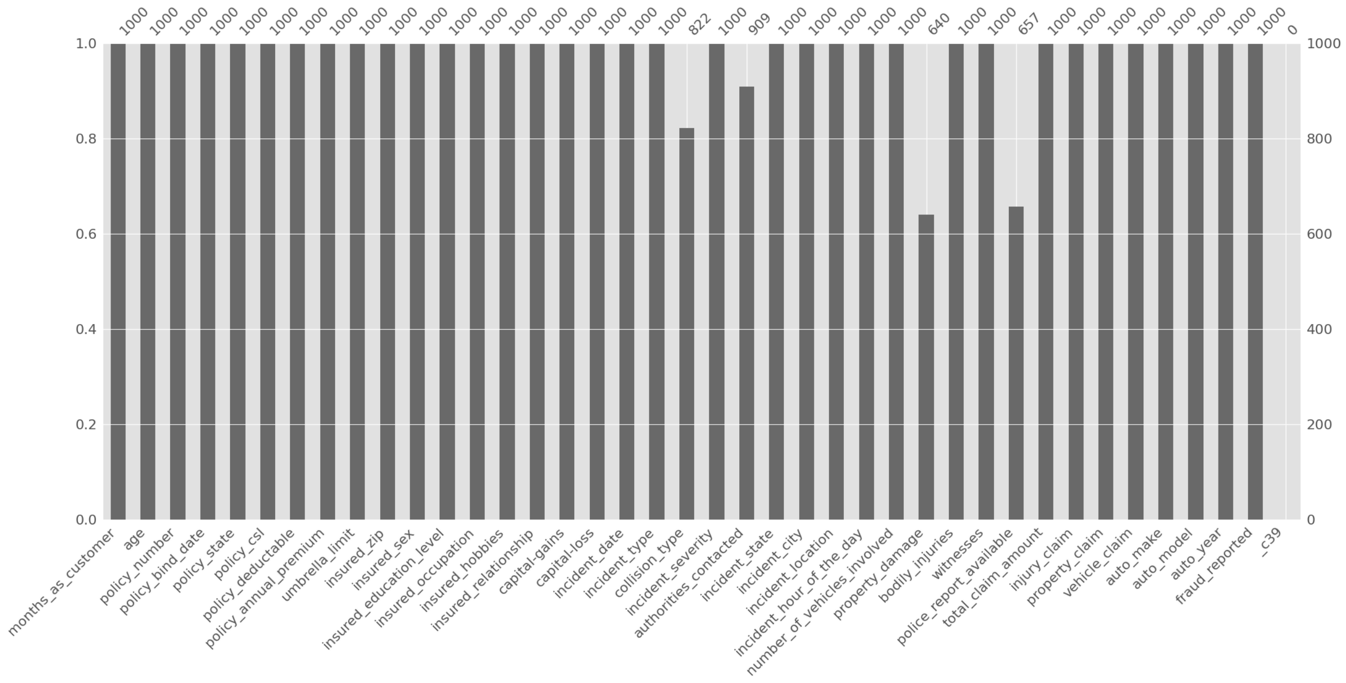
```
# missing values
df.isna().sum()
```

	0
months_as_customer	0
age	0
policy_number	0
policy_bind_date	0
policy_state	0
policy_csl	0
policy_deductable	0
policy_annual_premium	0
umbrella_limit	0
insured_zip	0
insured_sex	0
insured_education_level	0
insured_occupation	0
insured_hobbies	0
insured_relationship	0
capital-gains	0
capital-loss	0
incident_date	0
incident_type	0
collision_type	178
incident_severity	0
authorities_contacted	91
incident_state	0
incident_city	0
incident_location	0
incident_hour_of_the_day	0
number_of_vehicles_involved	0
property_damage	360
bodily_injuries	0
witnesses	0
police_report_available	343
total_claim_amount	0
injury_claim	0
property_claim	0
vehicle_claim	0
auto_make	0
auto_model	0
auto_year	0
fraud_reported	0
_c39	1000

dtype: int64

Visualizing Missing Values

```
import missingno as msno

msno.bar(df)
plt.show()
```



Handling Missing Values

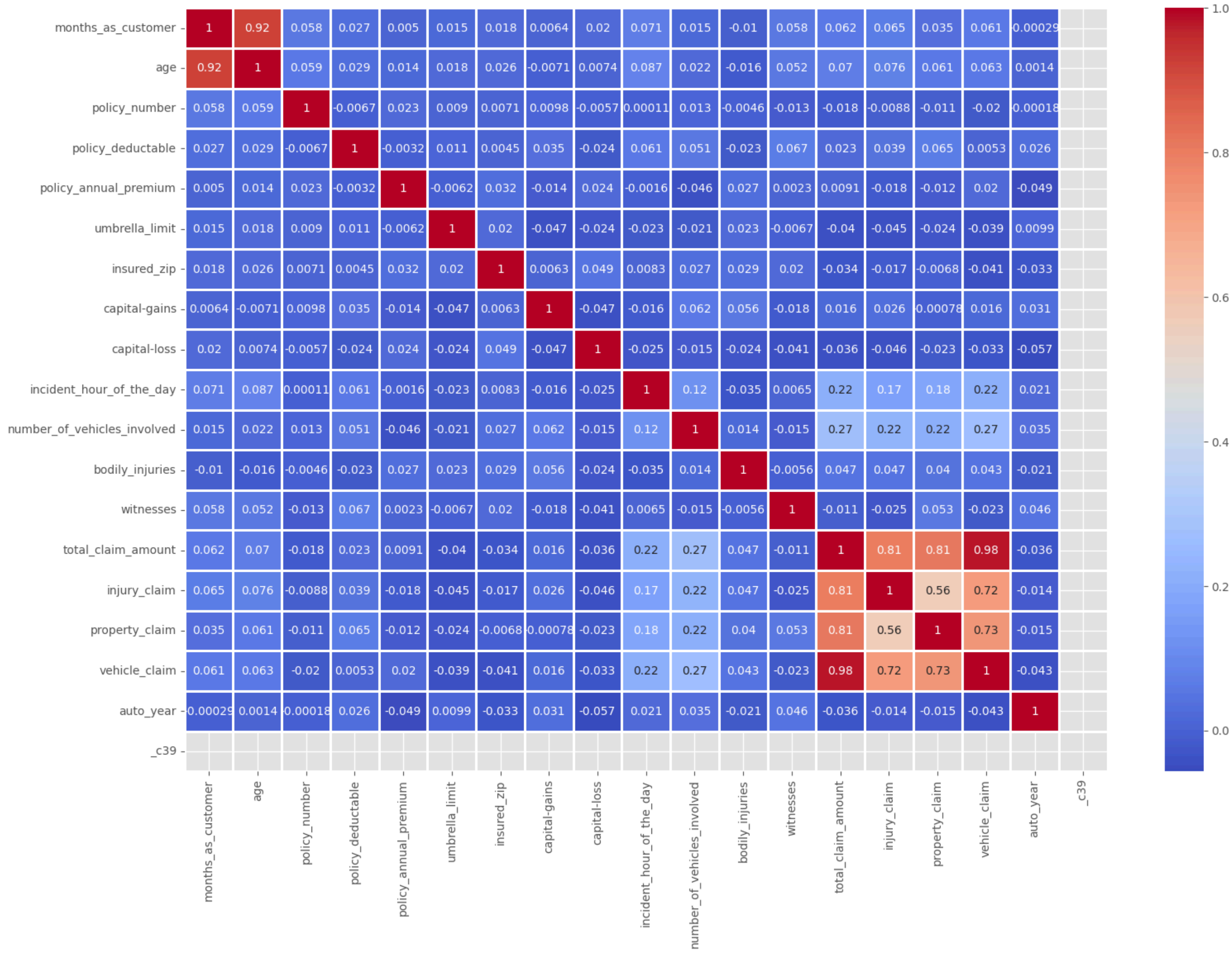
```
df['collision_type'] = df['collision_type'].fillna(df['collision_type'].mode()[0])

df['property_damage'] = df['property_damage'].fillna(df['property_damage'].mode()[0])

df['police_report_available'] = df['police_report_available'].fillna(df['police_report_available'].mode()[0])

df.isna().sum()
```

	0
months_as_customer	0
age	0
policy_number	0
policy_bind_date	0
policy_state	0
policy_csl	0
policy_deductable	0
policy_annual_premium	0
umbrella_limit	0
insured_zip	0
insured_sex	0
insured_education_level	0
insured_occupation	0
insured_hobbies	0
insured_relationship	0
capital-gains	0
capital-loss	0
incident_date	0
incident_type	0
collision_type	0
incident_severity	0
authorities_contacted	91
incident_state	0
incident_city	0
incident_location	0
incident_hour_of_the_day	0
number_of_vehicles_involved	0
property_damage	0
bodily_injuries	0
witnesses	0
police_report_available	0
total_claim_amount	0
injury_claim	0
property_claim	0
vehicle_claim	0
auto_make	0
auto_model	0
auto_year	0
fraud_reported	0
_c39	1000

dtype: int64

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns
df_numeric = df.select_dtypes(include=['number'])

# Compute correlation
corr = df_numeric.corr()

# Plot heatmap
```

```
plt.figure(figsize=(18, 12))
sns.heatmap(data=corr, annot=True, fmt='.2g', linewidths=1, cmap="coolwarm")
plt.show()
```




```
df.nunique()
```

	0
months_as_customer	391
age	46
policy_number	1000
policy_bind_date	951
policy_state	3
policy_csl	3
policy_deductable	3
policy_annual_premium	991
umbrella_limit	11
insured_zip	995
insured_sex	2
insured_education_level	7
insured_occupation	14
insured_hobbies	20
insured_relationship	6
capital-gains	338
capital-loss	354
incident_date	60
incident_type	4
collision_type	3
incident_severity	4
authorities_contacted	4
incident_state	7
incident_city	7
incident_location	1000
incident_hour_of_the_day	24
number_of_vehicles_involved	4
property_damage	2
bodily_injuries	3
witnesses	4
police_report_available	2
total_claim_amount	763
injury_claim	638
property_claim	626
vehicle_claim	726
auto_make	14
auto_model	39
auto_year	21
fraud_reported	2
_c39	0

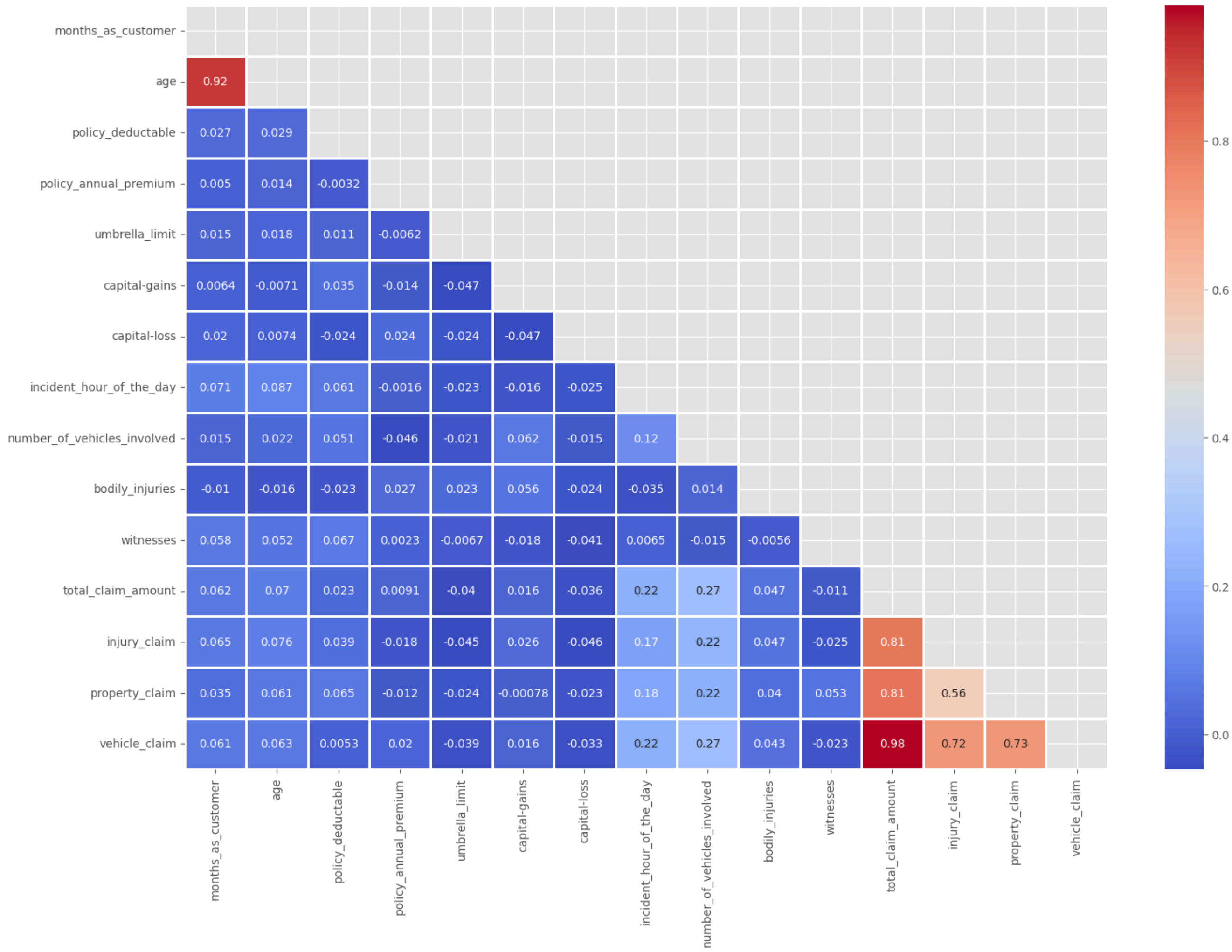
dtype: int64

```
# dropping columns which are not necessary for prediction

to_drop = ['policy_number','policy_bind_date','policy_state','insured_zip','incident_location','incident_date',
           'incident_state','incident_city','insured_hobbies','auto_make','auto_model','auto_year', '_c39']

df.drop(to_drop, inplace = True, axis = 1)


df.head()
```

	months_as_customer	age	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_lev
0	328	48	250/500	1000	1406.91	0	MALE	M
1	228	42	250/500	2000	1197.22	5000000	MALE	M
2	134	29	100/300	2000	1413.14	5000000	FEMALE	P
3	256	41	250/500	2000	1415.74	6000000	FEMALE	P
4	228	44	500/1000	1000	1583.91	6000000	MALE	Associ

5 rows × 27 columns

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert numeric columns, coercing errors to NaN
df_cleaned = df.apply(pd.to_numeric, errors='coerce')

# Drop rows/columns with NaN values if necessary
df_cleaned = df_cleaned.dropna(axis=1)  # Drops non-numeric columns

# Compute correlation
corr = df_cleaned.corr()

# Create upper triangular mask
mask = np.triu(np.ones_like(corr, dtype=bool))

# Plot heatmap
plt.figure(figsize=(18, 12))
sns.heatmap(data=corr, mask=mask, annot=True, fmt='.2g', linewidths=1, cmap="coolwarm")
plt.show()
```



From the above plot, we can see that there is high correlation between age and months_as_customer.We will drop the "Age" column. Also there is high correlation between total_clam_amount, injury_claim, property_claim, vehicle_claim as total claim is the sum of all others. So we will drop the total claim column.

```
df.drop(columns = ['age', 'total_claim_amount'], inplace = True, axis = 1)
```

```
df.head()
```

	months_as_customer	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	i
0	328	250/500	1000	1406.91	0	MALE	MD	
1	228	250/500	2000	1197.22	5000000	MALE	MD	
2	134	100/300	2000	1413.14	5000000	FEMALE	PhD	
3	256	250/500	2000	1415.74	6000000	FEMALE	PhD	
4	228	500/1000	1000	1583.91	6000000	MALE	Associate	

5 rows × 25 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 25 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           1000 non-null   int64  
 1   policy_csl                   1000 non-null   object 
 2   policy_deductable            1000 non-null   int64  
 3   policy_annual_premium        1000 non-null   float64
 4   umbrella_limit               1000 non-null   int64  
 5   insured_sex                  1000 non-null   object 
 6   insured_education_level      1000 non-null   object 
 7   insured_occupation           1000 non-null   object 
 8   insured_relationship         1000 non-null   object 
 9   capital-gains                1000 non-null   int64  
 10  capital-loss                 1000 non-null   int64  
 11  incident_type                1000 non-null   object 
 12  collision_type               1000 non-null   object 
 13  incident_severity            1000 non-null   object 
 14  authorities_contacted        909 non-null    object 
 15  incident_hour_of_the_day     1000 non-null   int64  
 16  number_of_vehicles_involved  1000 non-null   int64  
 17  property_damage              1000 non-null   object 
 18  bodily_injuries              1000 non-null   int64  
 19  witnesses                    1000 non-null   int64  
 20  police_report_available      1000 non-null   object 
 21  injury_claim                 1000 non-null   int64  
 22  property_claim               1000 non-null   int64  
 23  vehicle_claim                1000 non-null   int64  
 24  fraud_reported               1000 non-null   object 
dtypes: float64(1), int64(12), object(12)
memory usage: 195.4+ KB
```

```
# separating the feature and target columns

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

Encoding Categorical Columns

```
# extracting categorical columns
cat_df = X.select_dtypes(include = ['object'])
```

```
cat_df.head()
```

	policy_csl	insured_sex	insured_education_level	insured_occupation	insured_relationship	incident_type	collision_type	incide
0	250/500	MALE	MD	craft-repair	husband	Single Vehicle Collision	Side Collision	M
1	250/500	MALE	MD	machine-op-inspct	other-relative	Vehicle Theft	Rear Collision	M
2	100/300	FEMALE	PhD	sales	own-child	Multi-vehicle Collision	Rear Collision	M
3	250/500	FEMALE	PhD	armed-forces	unmarried	Single Vehicle Collision	Front Collision	M
4	500/1000	MALE	Associate	sales	unmarried	Vehicle Theft	Rear Collision	M

```
# printing unique values of each column
for col in cat_df.columns:
    print(f"{col}: \n{cat_df[col].unique()}\n")
```

```
policy_csl:
['250/500' '100/300' '500/1000']

insured_sex:
['MALE' 'FEMALE']

insured_education_level:
['MD' 'PhD' 'Associate' 'Masters' 'High School' 'College' 'JD']

insured_occupation:
['craft-repair' 'machine-op-inspct' 'sales' 'armed-forces' 'tech-support'
 'prof-specialty' 'other-service' 'priv-house-serv' 'exec-managerial'
 'protective-serv' 'transport-moving' 'handlers-cleaners' 'adm-clerical'
 'farming-fishing']

insured_relationship:
['husband' 'other-relative' 'own-child' 'unmarried' 'wife' 'not-in-family']
```

```
incident_type:
['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']

collision_type:
['Side Collision' 'Rear Collision' 'Front Collision']

incident_severity:
['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']

authorities_contacted:
['Police' nan 'Fire' 'Other' 'Ambulance']

property_damage:
['YES' 'NO']

police_report_available:
['YES' 'NO']
```

```
cat_df = pd.get_dummies(cat_df, drop_first = True)
```

```
cat_df.head()
```

	policy_csl_250/500	policy_csl_500/1000	insured_sex_MALE	insured_education_level_College	insured_education_level_High School	insure
0	True	False	True	False	False	
1	True	False	True	False	False	
2	False	False	False	False	False	
3	True	False	False	False	False	
4	False	True	True	False	False	

5 rows × 40 columns

```
# extracting the numerical columns

num_df = X.select_dtypes(include = ['int64'])
```

```
num_df.head()
```

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved
0	328	1000	0	53300	0	5	1
1	228	2000	5000000	0	0	8	1
2	134	2000	5000000	35100	0	7	3
3	256	2000	6000000	48900	-62400	5	1
4	228	1000	6000000	66000	-46000	20	1

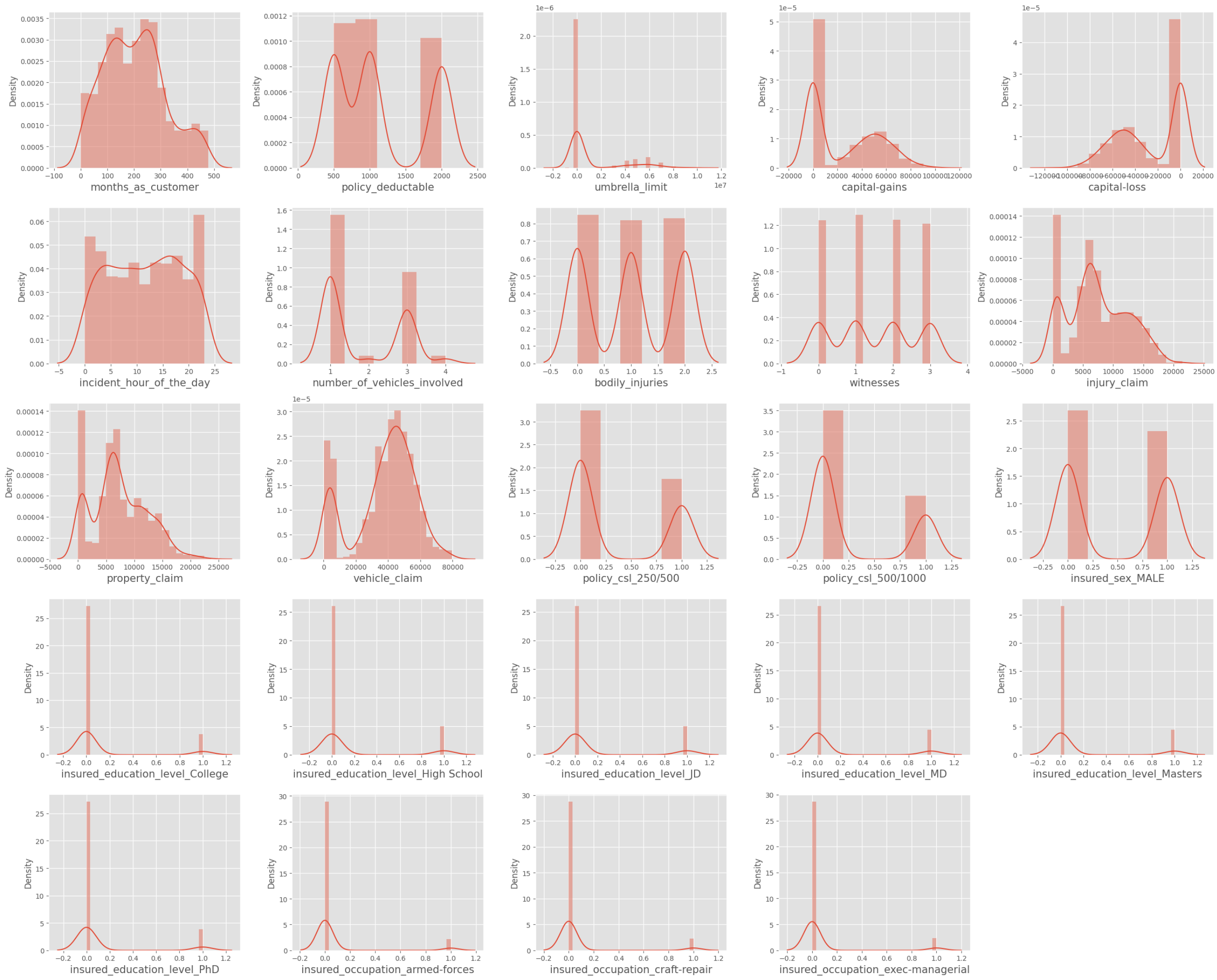
```
# combining the Numerical and Categorical dataframes to get the final dataset

X = pd.concat([num_df, cat_df], axis = 1)
```

```
X.head()
```

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved
0	328	1000	0	53300	0	5	1
1	228	2000	5000000	0	0	8	1
2	134	2000	5000000	35100	0	7	3
3	256	2000	6000000	48900	-62400	5	1
4	228	1000	6000000	66000	-46000	20	1

5 rows × 52 columns

```
plt.figure(figsize = (25, 20))
plotnumber = 1

for col in X.columns:
    if plotnumber <= 24:
        ax = plt.subplot(5, 5, plotnumber)
        sns.distplot(X[col])
        plt.xlabel(col, fontsize = 15)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

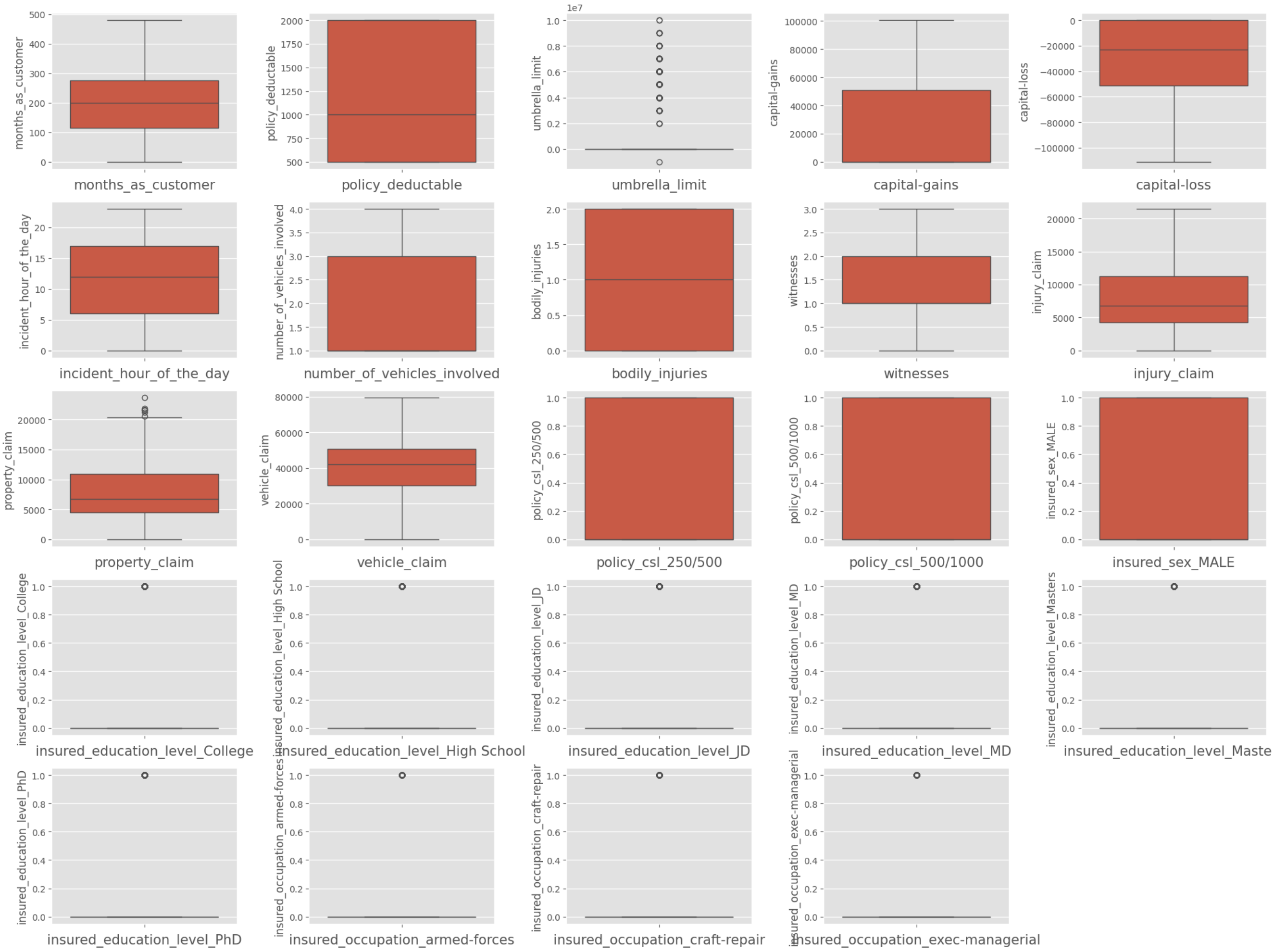


Data looks good, let's check for outliers.

Outliers Detection

```
plt.figure(figsize = (20, 15))
plotnumber = 1
```

```
for col in X.columns:
    if plotnumber <= 24:
        ax = plt.subplot(5, 5, plotnumber)
        sns.boxplot(X[col])
        plt.xlabel(col, fontsize = 15)

    plotnumber += 1
plt.tight_layout()
plt.show()
```



Outliers are present in some numerical columns we will scale numerical columns later

```
# splitting data into training set and test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

```
X_train.head()
```

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involve
642	143	1000	0	79900	0	1	
834	132	2000	0	43100	-31900	21	
331	283	2000	0	53500	-73600	3	
956	95	2000	0	67800	-48600	21	
592	253	1000	0	36600	0	17	

5 rows × 52 columns

```
num_df = X_train[['months_as_customer', 'policy_deductable', 'umbrella_limit',
       'capital-gains', 'capital-loss', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'injury_claim', 'property_claim',
       'vehicle_claim']]
```

```
# Scaling the numeric values in the dataset

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(num_df)
```

```
scaled_num_df = pd.DataFrame(data = scaled_data, columns = num_df.columns, index = X_train.index)
scaled_num_df.head()
```

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involve
642	-0.541112	-0.207625	-0.47214	1.937428	0.972951	-1.546128	-0.8428
834	-0.637833	1.422936	-0.47214	0.621772	-0.157986	1.304403	-0.8428
331	0.689887	1.422936	-0.47214	0.993588	-1.636359	-1.261075	1.1081
956	-0.963169	1.422936	-0.47214	1.504835	-0.750044	1.304403	1.1081
592	0.426102	-0.207625	-0.47214	0.389387	0.972951	0.734297	-0.8428

```
X_train.drop(columns = scaled_num_df.columns, inplace = True)
```

```
X_train = pd.concat([scaled_num_df, X_train], axis = 1)
```

```
X_train.head()
```

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involve
642	-0.541112	-0.207625	-0.47214	1.937428	0.972951	-1.546128	-0.8428
834	-0.637833	1.422936	-0.47214	0.621772	-0.157986	1.304403	-0.8428
331	0.689887	1.422936	-0.47214	0.993588	-1.636359	-1.261075	1.1081
956	-0.963169	1.422936	-0.47214	1.504835	-0.750044	1.304403	1.1081
592	0.426102	-0.207625	-0.47214	0.389387	0.972951	0.734297	-0.8428

5 rows × 52 columns

Models

Support Vector Classifier

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)
```

```
# accuracy_score, confusion_matrix and classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

svc_train_acc = accuracy_score(y_train, svc.predict(X_train))
svc_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of Support Vector Classifier is : {svc_train_acc}")
print(f"Test accuracy of Support Vector Classifier is : {svc_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of Support Vector Classifier is : 0.844
Test accuracy of Support Vector Classifier is : 0.724
[[181   0]
 [ 69   0]]
              precision    recall  f1-score   support

           N       0.72      1.00      0.84       181
           Y       0.00      0.00      0.00        69

    accuracy                           0.72       250
   macro avg       0.36      0.50      0.42       250
weighted avg       0.52      0.72      0.61       250
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 30)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

```
# accuracy_score, confusion_matrix and classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn_train_acc = accuracy_score(y_train, knn.predict(X_train))
knn_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of KNN is : {knn_train_acc}")
print(f"Test accuracy of KNN is : {knn_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of KNN is : 0.7626666666666667
Test accuracy of KNN is : 0.724
[[181   0]
 [ 69   0]]
              precision    recall  f1-score   support

           N       0.72      1.00      0.84       181
           Y       0.00      0.00      0.00        69

    accuracy                           0.72       250
   macro avg       0.36      0.50      0.42       250
weighted avg       0.52      0.72      0.61       250
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)
```

```
# accuracy_score, confusion_matrix and classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

dtc_train_acc = accuracy_score(y_train, dtc.predict(X_train))
dtc_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of Decision Tree is : {dtc_train_acc}")
```

```
print(f"Test accuracy of Decision Tree is : {dtc_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of Decision Tree is : 1.0
Test accuracy of Decision Tree is : 0.58
[[120  61]
 [ 44  25]]
              precision    recall  f1-score   support

           N       0.73      0.66      0.70       181
           Y       0.29      0.36      0.32        69

    accuracy                           0.58       250
   macro avg       0.51      0.51      0.51       250
weighted avg       0.61      0.58      0.59       250
```

```
# hyper parameter tuning

from sklearn.model_selection import GridSearchCV

grid_params = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'min_samples_split' : range(2, 10, 1),
    'min_samples_leaf' : range(2, 10, 1)
}

grid_search = GridSearchCV(dtc, grid_params, cv = 5, n_jobs = -1, verbose = 1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
```

GridSearchCV

best_estimator_: DecisionTreeClassifier

DecisionTreeClassifier

```
# best parameters and best score

print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 7, 'min_samples_split': 2}
0.8119999999999999
```

```
# best estimator

dtc = grid_search.best_estimator_

y_pred = dtc.predict(X_test)
```

```
# accuracy_score, confusion_matrix and classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

dtc_train_acc = accuracy_score(y_train, dtc.predict(X_train))
dtc_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of Decision Tree is : {dtc_train_acc}")
print(f"Test accuracy of Decision Tree is : {dtc_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of Decision Tree is : 0.816
Test accuracy of Decision Tree is : 0.716
[[128  53]
 [ 18  51]]
              precision    recall  f1-score   support

           N       0.88      0.71      0.78       181
           Y       0.49      0.74      0.59        69

    accuracy                           0.72       250
   macro avg       0.68      0.72      0.69       250
weighted avg       0.77      0.72      0.73       250
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion= 'entropy', max_depth= 10, max_features= 'sqrt', min_samples_leaf= 1, min_samples_split= 3,
rand_clf.fit(X_train, y_train)

y_pred = rand_clf.predict(X_test)


# accuracy_score, confusion_matrix and classification_report

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

rand_clf_train_acc = accuracy_score(y_train, rand_clf.predict(X_train))
rand_clf_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of Random Forest is : {rand_clf_train_acc}")
print(f"Test accuracy of Random Forest is : {rand_clf_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of Random Forest is : 0.9666666666666667
Test accuracy of Random Forest is : 0.756
[[172   9]
 [ 52  17]]
              precision    recall  f1-score   support

           N       0.77      0.95      0.85       181
           Y       0.65      0.25      0.36        69

    accuracy                           0.76       250
   macro avg       0.71      0.60      0.60       250
weighted avg       0.74      0.76      0.71       250
```

Ada Boost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define base estimator (Decision Tree Classifier)
dtc = DecisionTreeClassifier()

# Corrected AdaBoostClassifier initialization
ada = AdaBoostClassifier(estimator=dtc)  # Use 'estimator' instead of 'base_estimator'

# Define hyperparameters for GridSearchCV
parameters = {
    'n_estimators': [50, 70, 90, 120, 180, 200],
    'learning_rate': [0.001, 0.01, 0.1, 1, 10],
    'algorithm': ['SAMME', 'SAMME.R']
}

# Perform Grid Search with Cross Validation
grid_search = GridSearchCV(ada, parameters, n_jobs=-1, cv=5, verbose=1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

GridSearchCV

best_estimator_: AdaBoostClassifier

estimator:
DecisionTreeClassifier

DecisionTreeClassifier

```
# best parameter and best score

print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'algorithm': 'SAMME', 'learning_rate': 10, 'n_estimators': 200}
0.7053333333333334
```

```
# best estimator

ada = grid_search.best_estimator_

y_pred = ada.predict(X_test)


# accuracy_score, confusion_matrix and classification_report

ada_train_acc = accuracy_score(y_train, ada.predict(X_train))
ada_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of Ada Boost is : {ada_train_acc}")
print(f"Test accuracy of Ada Boost is : {ada_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of Ada Boost is : 1.0
Test accuracy of Ada Boost is : 0.592
[[127  54]
 [ 48  21]]
              precision    recall  f1-score   support

           N       0.73      0.70      0.71       181
           Y       0.28      0.30      0.29        69

    accuracy                           0.59       250
   macro avg       0.50      0.50      0.50       250
weighted avg       0.60      0.59      0.60       250
```

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of gradient boosting classifier

gb_acc = accuracy_score(y_test, gb.predict(X_test))

print(f"Training Accuracy of Gradient Boosting Classifier is {accuracy_score(y_train, gb.predict(X_train))}")
print(f"Test Accuracy of Gradient Boosting Classifier is {gb_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, gb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, gb.predict(X_test))}")
```

```
Training Accuracy of Gradient Boosting Classifier is 0.9426666666666667
Test Accuracy of Gradient Boosting Classifier is 0.32

Confusion Matrix :-
[[ 15 166]
 [  4  65]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.79      0.08      0.15       181
           Y       0.28      0.94      0.43        69

    accuracy                           0.32       250
   macro avg       0.54      0.51      0.29       250
weighted avg       0.65      0.32      0.23       250
```

Stochastic Gradient Boosting (SGB)

```
sgb = GradientBoostingClassifier(subsample = 0.90, max_features = 0.70)
sgb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of stochastic gradient boosting classifier

sgb_acc = accuracy_score(y_test, sgb.predict(X_test))

print(f"Training Accuracy of Stochastic Gradient Boosting is {accuracy_score(y_train, sgb.predict(X_train))}")
print(f"Test Accuracy of Stochastic Gradient Boosting is {sgb_acc} \n")
```

```
print(f"Confusion Matrix :- \n{confusion_matrix(y_test, sgb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, sgb.predict(X_test))}")
```

```
Training Accuracy of Stochastic Gradient Boosting is 0.9426666666666667
Test Accuracy of Stochastic Gradient Boosting is 0.276

Confusion Matrix :-
[[  0 181]
 [  0  69]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.00      0.00      0.00       181
           Y       0.28      1.00      0.43        69

    accuracy                           0.28       250
   macro avg       0.14      0.50      0.22       250
weighted avg       0.08      0.28      0.12       250
```

XgBoost Classifier

```
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder

# Encode target variable (convert 'N' -> 0 and 'Y' -> 1)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)  # Transform test labels as well

# Train the XGBoost classifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train_encoded)

# Make predictions
y_pred = xgb.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder

# Encode y_test to match the format of y_pred
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)  # Encode y_test correctly

# Train the XGBoost classifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train_encoded)

# Make predictions
y_pred = xgb.predict(X_test)

# Calculate accuracy
xgb_train_acc = accuracy_score(y_train_encoded, xgb.predict(X_train))
xgb_test_acc = accuracy_score(y_test_encoded, y_pred)

print(f"Training accuracy of XGBoost: {xgb_train_acc}")
print(f"Test accuracy of XGBoost: {xgb_test_acc}")

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

```
Training accuracy of XGBoost: 1.0
Test accuracy of XGBoost: 0.696
[[158  23]
 [ 53  16]]
              precision    recall  f1-score   support

           0       0.75      0.87      0.81       181
           1       0.41      0.23      0.30        69

    accuracy                           0.70       250
   macro avg       0.58      0.55      0.55       250
weighted avg       0.66      0.70      0.67       250
```

```
param_grid = {"n_estimators": [10, 50, 100, 130], "criterion": ['gini', 'entropy'],
                              "max_depth": range(2, 10, 1)}
```

```
grid = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5,  verbose=3,n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
GridSearchCV
best_estimator_: AdaBoostClassifier
estimator:
DecisionTreeClassifier
DecisionTreeClassifier
```

```
# best estimator

xgb = grid_search.best_estimator_

y_pred = xgb.predict(X_test)


# accuracy_score, confusion_matrix and classification_report

xgb_train_acc = accuracy_score(y_train, xgb.predict(X_train))
xgb_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of XgBoost is : {xgb_train_acc}")
print(f"Test accuracy of XgBoost is : {xgb_test_acc}")

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of XgBoost is : 1.0
Test accuracy of XgBoost is : 0.572
[[103  78]
 [ 29  40]]
              precision    recall  f1-score   support

           N       0.78      0.57      0.66       181
           Y       0.34      0.58      0.43        69

    accuracy                           0.57       250
   macro avg       0.56      0.57      0.54       250
weighted avg       0.66      0.57      0.59       250
```

Cat Boost Classifier

```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.55.8)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 5.9 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

```
from catboost import CatBoostClassifier

cat = CatBoostClassifier(iterations=10)
cat.fit(X_train, y_train)
```

```
Learning rate set to 0.5
0:	learn: 0.5595318	total: 56.4ms	remaining: 508ms
1:	learn: 0.4452461	total: 63.8ms	remaining: 255ms
2:	learn: 0.4163483	total: 72ms	remaining: 168ms
3:	learn: 0.3930434	total: 81.8ms	remaining: 123ms
4:	learn: 0.3591954	total: 90.9ms	remaining: 90.9ms
5:	learn: 0.3322490	total: 95.7ms	remaining: 63.8ms
6:	learn: 0.3111951	total: 104ms	remaining: 44.5ms
7:	learn: 0.2934157	total: 109ms	remaining: 27.3ms
8:	learn: 0.2763937	total: 117ms	remaining: 13ms
9:	learn: 0.2643205	total: 125ms	remaining: 0us
<catboost.core.CatBoostClassifier at 0x7c3a29b08e90>
```

```
# accuracy score, confusion matrix and classification report of cat boost

cat_acc = accuracy_score(y_test, cat.predict(X_test))

print(f"Training Accuracy of Cat Boost Classifier is {accuracy_score(y_train, cat.predict(X_train))}")
print(f"Test Accuracy of Cat Boost Classifier is {cat_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, cat.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, cat.predict(X_test))}")
```

```
Training Accuracy of Cat Boost Classifier is 0.9186666666666666
Test Accuracy of Cat Boost Classifier is 0.692

Confusion Matrix :-
[[139  42]
 [ 35  34]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.80      0.77      0.78       181
           Y       0.45      0.49      0.47        69

    accuracy                           0.69       250
   macro avg       0.62      0.63      0.63       250
weighted avg       0.70      0.69      0.70       250
```

Extra Trees Classifier

```
from sklearn.ensemble import ExtraTreesClassifier

etc = ExtraTreesClassifier()
etc.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of extra trees classifier

etc_acc = accuracy_score(y_test, etc.predict(X_test))

print(f"Training Accuracy of Extra Trees Classifier is {accuracy_score(y_train, etc.predict(X_train))}")
print(f"Test Accuracy of Extra Trees Classifier is {etc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, etc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, etc.predict(X_test))}")
```

```
Training Accuracy of Extra Trees Classifier is 1.0
Test Accuracy of Extra Trees Classifier is 0.748

Confusion Matrix :-
[[159  22]
 [ 41  28]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.80      0.88      0.83       181
           Y       0.56      0.41      0.47        69

    accuracy                           0.75       250
   macro avg       0.68      0.64      0.65       250
weighted avg       0.73      0.75      0.73       250
```

LGBM Classifier

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(learning_rate = 1)
lgbm.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of lgbm classifier

lgbm_acc = accuracy_score(y_test, lgbm.predict(X_test))

print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train, lgbm.predict(X_train))}")
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

print(f"{confusion_matrix(y_test, lgbm.predict(X_test))}\n")
print(classification_report(y_test, lgbm.predict(X_test)))
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Training Accuracy of LGBM Classifier is 1.0
Test Accuracy of LGBM Classifier is 0.676

[[129  52]
 [ 29  40]]

              precision    recall  f1-score   support

           N       0.82      0.71      0.76       181
           Y       0.43      0.58      0.50        69
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier

classifiers = [('Support Vector Classifier', svc), ('KNN', knn),  ('Decision Tree', dtc), ('Random Forest', rand_clf),
               ('Ada Boost', ada), ('XGboost', xgb), ('Gradient Boosting Classifier', gb), ('SGB', sgb),
               ('Cat Boost', cat), ('Extra Trees Classifier', etc), ('LGBM', lgbm)]
```