In [10]:
```python
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')
```

In [11]:
```python
df = pd.read_csv('C:/Users/ROSHAN D K/Desktop/Python Projects/insurance_claims.csv')
df.head()
```

Out[11]:

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

5 rows × 40 columns

In [12]:
```python
df.replace('?', np.nan, inplace = True)
```

In [13]:
```python
df.describe()
```

Out[13]:

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

In [14]:
```python
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        1000 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
```

In [17]:

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

Out[17]:

```
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            0
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
```

In [18]:

```
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_ava

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

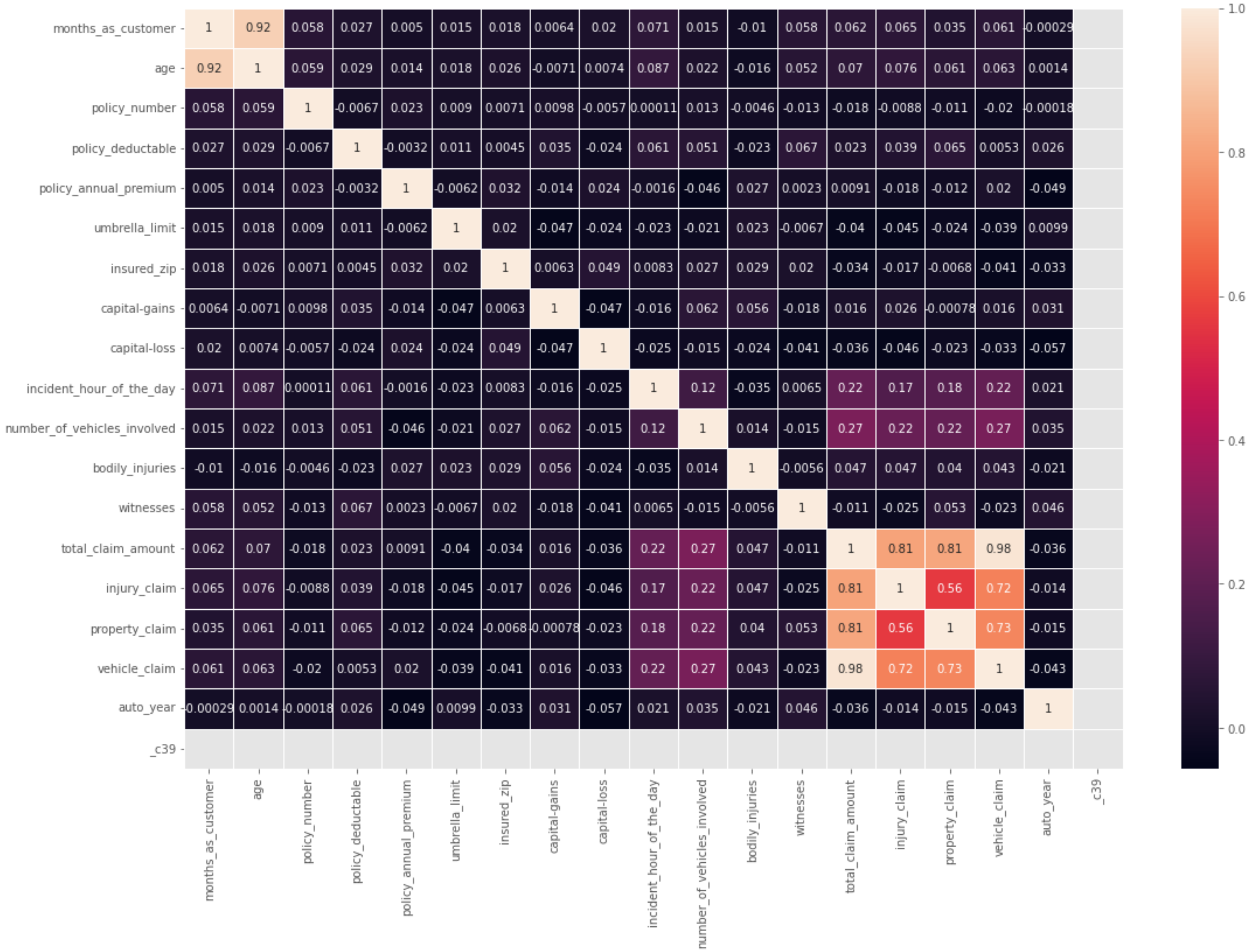
Out[18]:

```
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             0
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
```

In [19]:

```python
# heatmap
plt.figure(figsize = (18, 12))

corr = df.corr()

sns.heatmap(data = corr, annot = True, fmt = '.2g', linewidth = 1)
plt.show()

df.nunique()
```



Out[19]:

```
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           5
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
```

In [20]:

```
# dropping columns which are not necessary for prediction
to_drop = ['policy_number','policy_bind_date','policy_state','insured_zip','incident_locat
           'incident_state','incident_city','insured_hobbies','auto_make','auto_model','au

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

df.head()
```

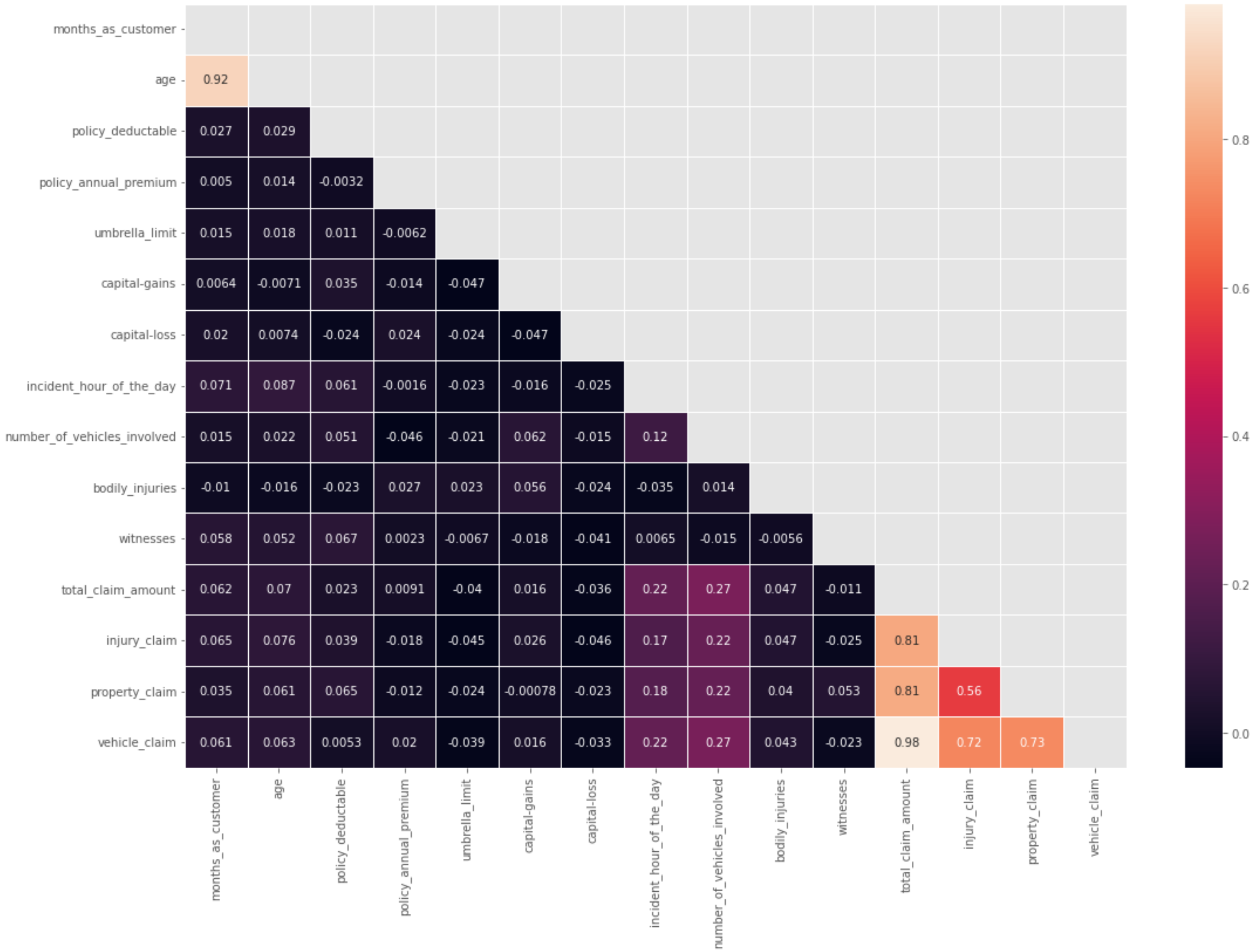
Out[20]:

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

5 rows × 27 columns

In [21]:

```
# checking for multicollinearity
plt.figure(figsize = (18, 12))

corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype = bool))

sns.heatmap(data = corr, mask = mask, annot = True, fmt = '.2g', linewidth = 1)
plt.show()
```



In [22]:

```
df.drop(columns = ['age', 'total_claim_amount'], inplace = True, axis = 1)
df.head()
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        1000 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
```

In [23]:

```
# separating the feature and target columns
X = df.drop('fraud_reported', axis = 1)
y = df['fraud_reported']
```

In [24]:

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

Out[24]:

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

In [25]:

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

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

cat_df.head()
```

```
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' 'None' 'Fire' 'Other' 'Ambulance']

property_damage:
['YES' 'NO']

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

Out[25]:

	policy_csl_250/500	policy_csl_500/1000	insured_sex_MALE	insured_education_level_College	insured_education_level_S
0	1	0	1	0	
1	1	0	1	0	
2	0	0	0	0	
3	1	0	0	0	
4	0	1	1	0	

5 rows × 41 columns

In [26]:

```python
# extracting the numerical columns

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

num_df.head()
```

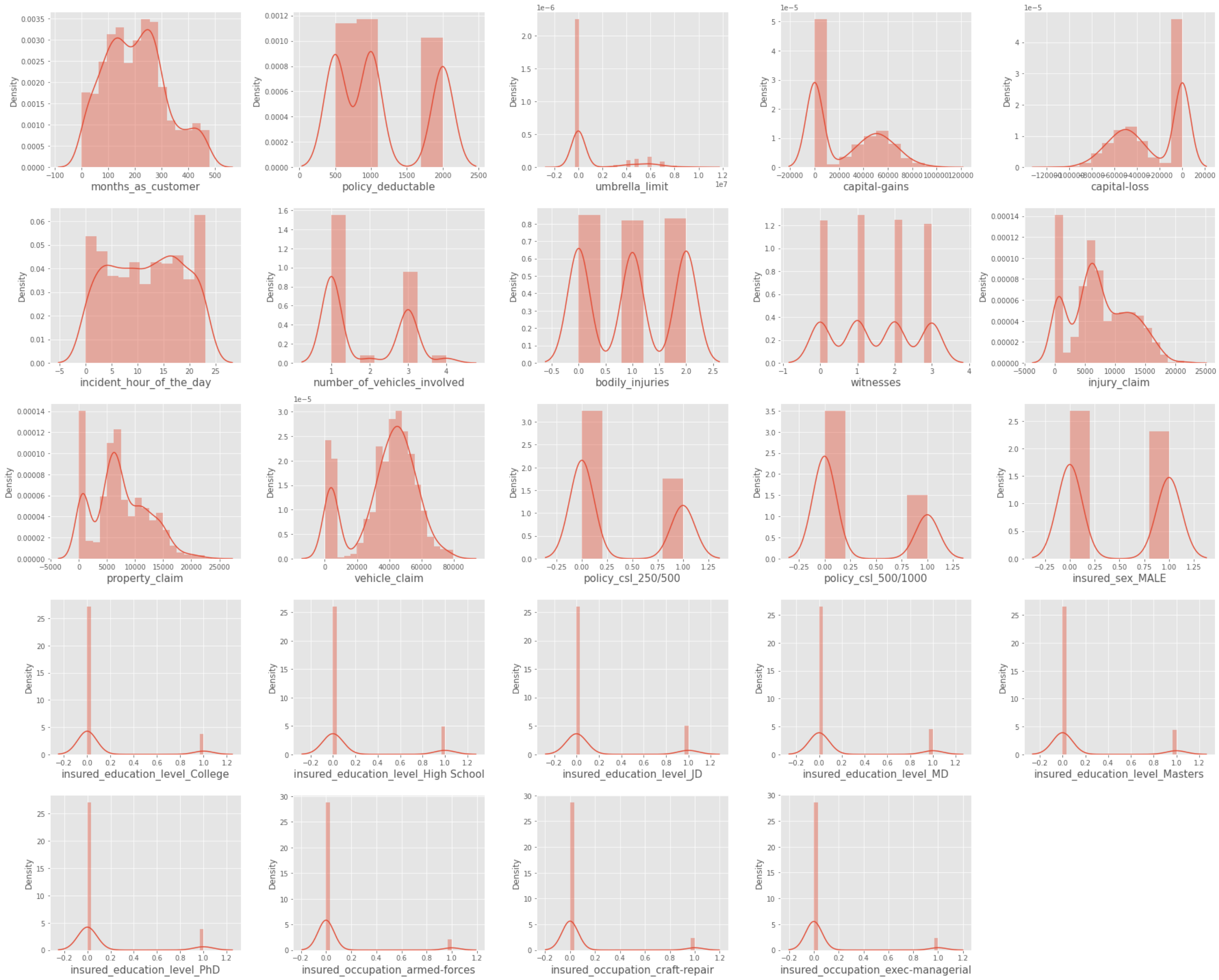
Out[26]:

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

In [27]:

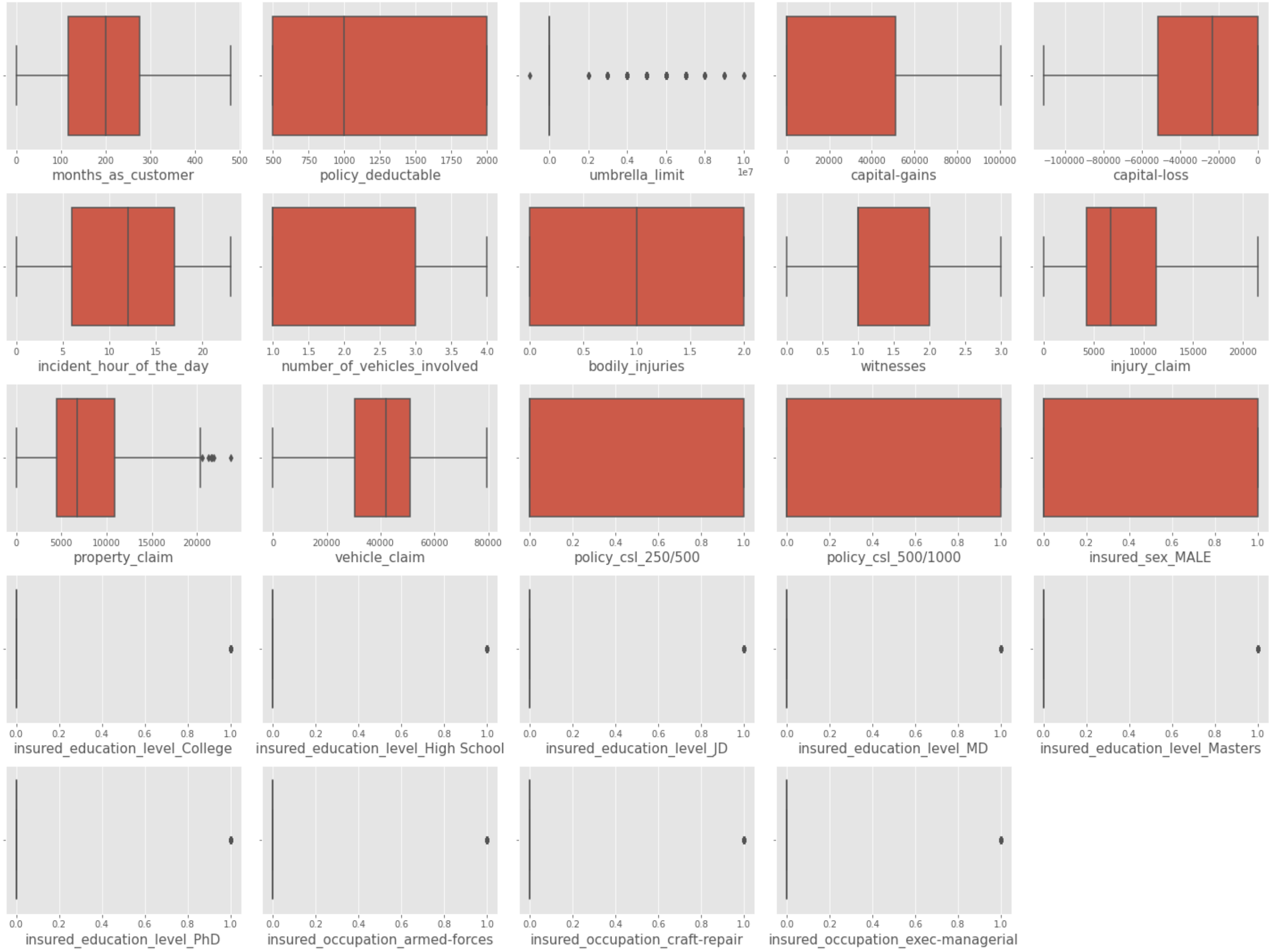
```python
# combining the Numerical and Categorical dataframes to get the final dataset

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

X.head()

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()
```



In [28]:

```
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()
```



In [29]:

```
# 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()

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', 'pro
        'vehicle_claim']]
```

In [30]:

```
# 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_trair
scaled_num_df.head()

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

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

X_train.head()
```

Out[30]:

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_
267	-1.428192	1.400591	-0.484263	0.953726	-0.747459	1.466730	
455	0.506379	1.400591	-0.484263	-0.894855	0.628276	-0.388325	
4	0.202254	-0.242010	2.138100	1.446917	-0.715215	1.181337	
76	0.557066	-0.242010	-0.484263	-0.894855	0.227020	0.895944	
254	-1.554911	-0.242010	-0.484263	-0.894855	0.932801	1.324034	

5 rows × 53 columns

In [31]:

```
from sklearn.svm import SVC

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

y_pred = svc.predict(X_test)
```

In [32]:

```
# 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 : 0.8493333333333334
Test accuracy of Support Vector Classifier is : 0.728
[[182   0]
 [ 68   0]]
              precision    recall  f1-score   support

           N       0.73      1.00      0.84       182
           Y       0.00      0.00      0.00        68

    accuracy                           0.73       250
   macro avg       0.36      0.50      0.42       250
weighted avg       0.53      0.73      0.61       250
```

In [33]:

```
from sklearn.neighbors import KNeighborsClassifier

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

y_pred = knn.predict(X_test)
```

In [37]:

```
# 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 : 0.7613333333333333
Test accuracy of KNN is : 0.376
[[ 29 153]
 [  3  65]]
              precision    recall  f1-score   support

           N       0.91      0.16      0.27       182
           Y       0.30      0.96      0.45        68

    accuracy                           0.38       250
   macro avg       0.60      0.56      0.36       250
weighted avg       0.74      0.38      0.32       250
```

In [38]:

```
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 : 0.392
[[ 44 138]
 [ 14  54]]
              precision    recall  f1-score   support

           N       0.76      0.24      0.37       182
           Y       0.28      0.79      0.42        68

    accuracy                           0.39       250
   macro avg       0.52      0.52      0.39       250
weighted avg       0.63      0.39      0.38       250
```

In [39]:

```
# 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)
# best parameters and best score

print(grid_search.best_params_)
print(grid_search.best_score_)
# best estimator

dtc = grid_search.best_estimator_

y_pred = dtc.predict(X_test)
```

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split': 8}
0.8133333333333335
```

In [40]:

```
# 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 : 0.72
[[135  47]
 [ 23  45]]
              precision    recall  f1-score   support

           N       0.85      0.74      0.79       182
           Y       0.49      0.66      0.56        68

    accuracy                           0.72       250
   macro avg       0.67      0.70      0.68       250
weighted avg       0.76      0.72      0.73       250
```

In [44]:

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion= 'entropy', max_depth= 10, max_features= 'sqrt
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 : 0.728
[[147  35]
 [ 33  35]]
              precision    recall  f1-score   support

           N       0.82      0.81      0.81       182
           Y       0.50      0.51      0.51        68

    accuracy                           0.73       250
   macro avg       0.66      0.66      0.66       250
weighted avg       0.73      0.73      0.73       250
```

In [45]:

```
# from sklearn.ensemble import AdaBoostClassifier

# ada = AdaBoostClassifier(base_estimator = dtc)

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

# grid_search = GridSearchCV(ada, parameters, n_jobs = -1, cv = 5, verbose = 1)
# grid_search.fit(X_train, y_train)

# # best parameter and best score

# print(grid_search.best_params_)
# print(grid_search.best_score_)

# # 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))
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
{'algorithm': 'SAMME', 'learning_rate': 0.001, 'n_estimators': 180}
0.812
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
C:\Users\ROSHAN~1\AppData\Local\Temp/ipykernel_17276/1768374190.py in <module>
     19 # accuracy_score, confusion_matrix and classification_report
     20 
---> 21 ada_train_acc = accuracy_score(y_train, ada.predict(X_train))
     22 ada_test_acc = accuracy_score(y_test, y_pred)
     23 

~\anaconda3\lib\site-packages\sklearn\ensemble\_weight_boosting.py in predict(self, X)
    698             The predicted classes.
    699         """
--> 700         pred = self.decision_function(X)
    701 
    702         if self.n_classes_ == 2:
```

```
~\anaconda3\lib\site-packages\sklearn\ensemble\_weight_boosting.py in decision_function(se
lf, X)
    758             class in ``classes_``, respectively.
    759         """
--> 760         check_is_fitted(self)
    761         X = self._check_X(X)
    762

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_is_fitted(estimator, at
tributes, msg, all_or_any)
   1388
   1389     if not fitted:
-> 1390         raise NotFittedError(msg % {"name": type(estimator).__name__})
   1391
   1392

NotFittedError: This AdaBoostClassifier instance is not fitted yet. Call 'fit' with approp
riate arguments before using this estimator.
```

In [46]:

```
from sklearn.ensemble import GradientBoostingClassifier

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

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

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

print(f"Training Accuracy of Gradient Boosting Classifier is {accuracy_score(y_train, gb.p
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 0.9386666666666666
Test Accuracy of Gradient Boosting Classifier is 0.288

Confusion Matrix :-
[[  6 176]
 [  2  66]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.75      0.03      0.06       182
           Y       0.27      0.97      0.43        68

    accuracy                           0.29       250
   macro avg       0.51      0.50      0.24       250
weighted avg       0.62      0.29      0.16       250
```

In [47]:

```
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 boosti

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

print(f"Training Accuracy of Stochastic Gradient Boosting is {accuracy_score(y_train, sgb.
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 0.9386666666666666
Test Accuracy of Stochastic Gradient Boosting is 0.3

Confusion Matrix :-
[[  8 174]
 [  1  67]]

Classification Report :-
              precision    recall  f1-score   support

           N       0.89      0.04      0.08       182
           Y       0.28      0.99      0.43        68

    accuracy                           0.30       250
   macro avg       0.58      0.51      0.26       250
weighted avg       0.72      0.30      0.18       250
```

In [52]:

```
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
xgb = XGBClassifier()
xgb.fit(X_train, y_train)

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))

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)

# 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 : 0.728
[[150  32]
 [ 36  32]]
              precision    recall  f1-score   support

           0       0.81      0.82      0.82       182
           1       0.50      0.47      0.48        68

    accuracy                           0.73       250
   macro avg       0.65      0.65      0.65       250
weighted avg       0.72      0.73      0.73       250

Fitting 5 folds for each of 60 candidates, totalling 300 fits
Training accuracy of XgBoost is : 0.816
Test accuracy of XgBoost is : 0.812
[[159  23]
 [ 24  44]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       182
           1       0.66      0.65      0.65        68

    accuracy                           0.81       250
   macro avg       0.76      0.76      0.76       250
weighted avg       0.81      0.81      0.81       250
```

In [49]:

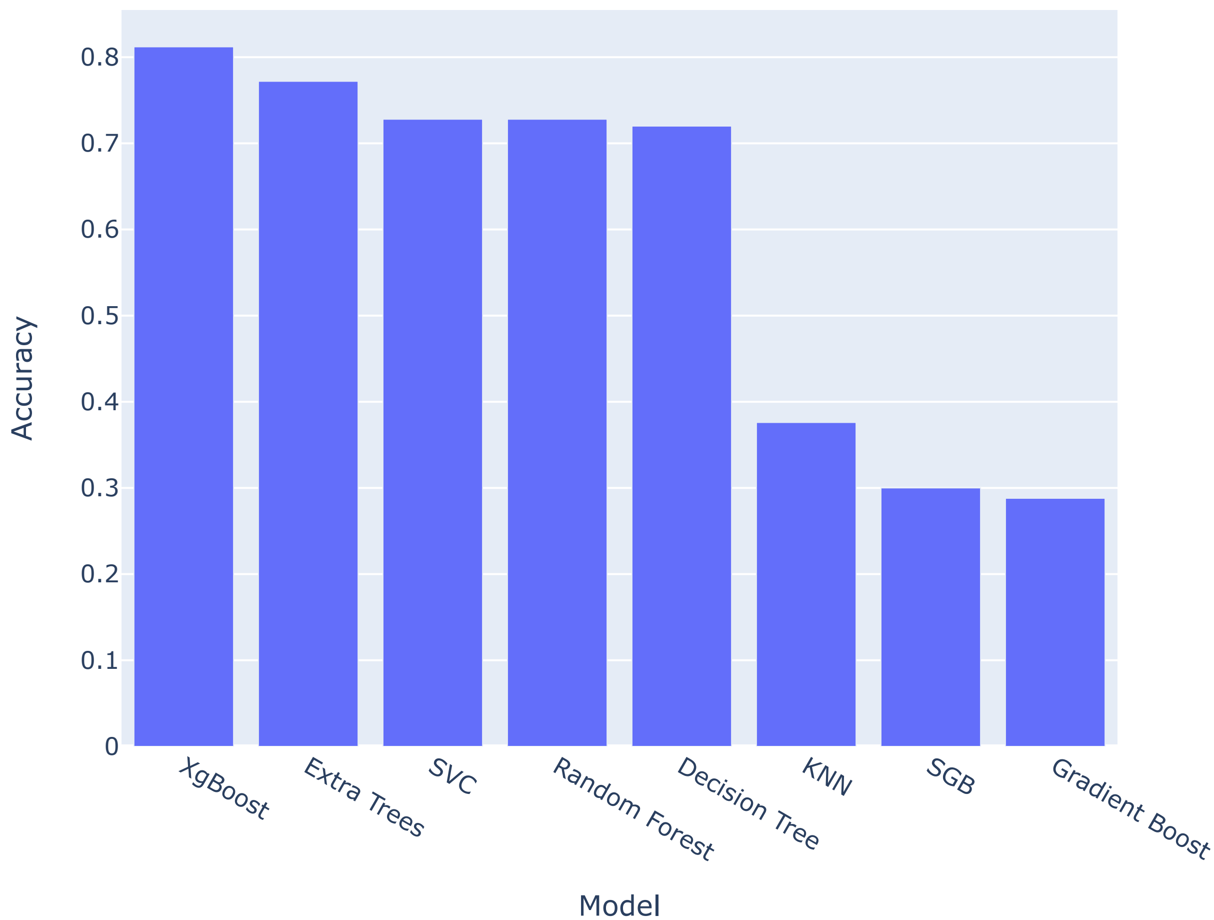
```python
from sklearn.ensemble import ExtraTreesClassifier

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.predic
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 0.772

Confusion Matrix :-
[[167  15]
 [ 42  26]]

Classification Report :-
               precision    recall  f1-score   support

           N       0.80      0.92      0.85       182
           Y       0.63      0.38      0.48        68

    accuracy                           0.77       250
   macro avg       0.72      0.65      0.67       250
weighted avg       0.75      0.77      0.75       250
```

In [54]:

```python
models = pd.DataFrame({
    'Model' : ['SVC', 'KNN', 'Decision Tree', 'Random Forest', 'Gradient Boost', 'SGB', 'E
    'Accuracy' : [svc_test_acc, knn_test_acc, dtc_test_acc, rand_clf_test_acc, gb_acc, sgb
```

```
models=models.sort_values(by = 'Accuracy', ascending = False)
models

px.bar(data_frame = models, x = 'Model', y = 'Accuracy')
```



In [ ]: