INSURANCE FRAUD DETECTION

#Data Collection

- we have insurance claims data (insurance_claim.csv). However, this data is not perfect.
- It may contain outliers, missing values and irrelevant (not related to insurance claims) information

In [1]:
```
import warnings
warnings.filterwarnings(action='ignore', category=FutureWarning)
```

In [2]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# To get all the rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

In [3]:
```
# Reading the Dataset File

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

In [4]:
```
df.head()
```

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annu
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	

In [5]:
```
# Check the columns
df.columns

# _c39 is extra columns let's remove it
df.drop('_c39', axis = 1, inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annu
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	

In [7]:
```
# Data description:
df.describe()
```

Out[7]:

	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 [8]:
```
# Number of rows and columns of data
print("Number of rows:", len(df))
print("Number of columns:", len(df.columns))
```

```
Number of rows: 1000
Number of columns: 39
```

Data Preparation

- We prepare the data column wise, as every column would require different preprocessing

In [9]:
```
def display_basic_information(df, column_name):
    """
    Function is responsible for printing out basic information
    about the dataset such as Null values, Data types, etc.
    """
    data_type = df[column_name].dtype
    null_values = df[column_name].isna().sum()
    print("####################")
    print("Basic Information:")
    print("####################\n")
    print("column name:", column_name)
    print("data type:", data_type)
    print("total number of null values:", null_values)
    print()

column_headers = list(df.columns)
for i in column_headers:
    display_basic_information(df,i)
```

```
####################
Basic Information:
####################

column name: months_as_customer
data type: int64
total number of null values: 0

####################
Basic Information:
####################

column name: age
data type: int64
total number of null values: 0
```

In [7]:

```python
# Data description
df.describe()
```

Out[7]:

	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 [9]:

```python
def display_basic_information(df, column_name):
    """
        Function is responsible for printing out basic information
        about the dataset such as Null values, Data types, etc.
    """
    data_type = df[column_name].dtype
    null_values = df[column_name].isna().sum()
    print("###################")
    print("Basic Information:")
    print("###################\n")
    print("column name:", column_name)
    print("data type:", data_type)
    print("total number of null values:", null_values)
    print()

column_headers = list(df.columns)
for i in column_headers:
    display_basic_information(df,i)
```

```
###################
Basic Information:
###################

column name: months_as_customer
data type: int64
total number of null values: 0

###################
Basic Information:
###################

column name: age
data type: int64
total number of null values: 0

###################
Basic Information:
###################
```

In [8]:

```python
# Number of rows and columns of data
print("Number of rows:", len(df))
print("Number of columns:", len(df.columns))
```

```
Number of rows: 1000
Number of columns: 39
```

Data Preparation

- We prepare the data column wise, as every column would require different preprocessing

In [10]:

```python
# To check the number of unique values inside the dataframe
for col in df.columns:
    print(col, df[col].dtype , df[col].nunique())
```

```
months_as_customer int64 391
age int64 46
policy_number int64 1000
policy_bind_date object 951
policy_state object 3
policy_csl object 3
policy_deductable int64 3
policy_annual_premium float64 991
umbrella_limit int64 11
insured_zip int64 995
insured_sex object 2
insured_education_level object 7
insured_occupation object 14
insured_hobbies object 20
insured_relationship object 6
capital-gains int64 338
capital-loss int64 354
incident_date object 60
incident_type object 4
collision_type object 4
incident_severity object 4
authorities_contacted object 5
incident_state object 7
incident_city object 7
incident_location object 1000
incident_hour_of_the_day int64 24
number_of_vehicles_involved int64 4
property_damage object 3
bodily_injuries int64 3
witnesses int64 4
police_report_available object 3
total_claim_amount int64 763
injury_claim int64 638
property_claim int64 626
vehicle_claim int64 726
auto_make object 14
auto_model object 39
auto_year int64 21
fraud_reported object 2
```

Checking the quality of data

In [13]:

```python
# Checking the null values
df.isna().sum()
```

Out[13]:

```
months_as_customer             0
age                            0
policy_state                   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_relationship           0
capital-gains                  0
capital-loss                   0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          0
incident_state                 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
```

In [11]:

```python
# From the above we can see that policy_number, policy_csl, policy_bind_date,
# incident_location, insured_hobbies, incident_date, incident_city, auto_model
# won't be helpful in knowledge extraction hence removing those.

df.drop(['policy_number', 'policy_csl', 'policy_bind_date', 'incident_location',\
         'insured_hobbies', 'incident_date', 'incident_city', 'auto_model'],\
        axis=1, inplace=True)
```

In [12]:

```python
# To check if the drop method is succesful
df.columns
```

Out[12]:

```
Index(['months_as_customer', 'age', 'policy_state', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_relationship',
       'capital-gains', 'capital-loss', 'incident_type', 'collision_type',
       'incident_severity', 'authorities_contacted', 'incident_state',
       'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'property_damage', 'bodily_injuries', 'witnesses',
       'police_report_available', 'total_claim_amount', 'injury_claim',
       'property_claim', 'vehicle_claim', 'auto_make', 'auto_year',
       'fraud_reported'],
      dtype='object')
```

```
bodily_injuries int64 3
witnesses int64 4
police_report_available object 3
total_claim_amount int64 763
injury_claim int64 638
property_claim int64 626
vehicle_claim int64 726
auto_make object 14
auto_model object 39
auto_year int64 21
fraud_reported object 2
```

Checking the quality of data

In [13]:

```python
# Checking the null values
df.isna().sum()
```

Out[13]:

```
months_as_customer             0
age                            0
policy_state                   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_relationship           0
capital-gains                  0
capital-loss                   0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          0
incident_state                 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_year                      0
fraud_reported                 0
dtype: int64
```

In [11]:

```python
# From the above we can see that policy_number, policy_csl, policy_bind_date,
# incident_location, insured_hobbies, incident_date, incident_city, auto_model
# won't be helpful in knowledge extraction hence removing those.

df.drop(['policy_number', 'policy_csl', 'policy_bind_date', 'incident_location',\
         'insured_hobbies', 'incident_date', 'incident_city', 'auto_model'],\
        axis=1, inplace=True)
```

In [12]:

```python
# To check if the drop method is succesful
df.columns
```

Out[12]:

```
Index(['months_as_customer', 'age', 'policy_state', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_relationship',
       'capital-gains', 'capital-loss', 'incident_type', 'collision_type',
       'incident_severity', 'authorities_contacted', 'incident_state',
       'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'property_damage', 'bodily_injuries', 'witnesses',
       'police_report_available', 'total_claim_amount', 'injury_claim',
       'property_claim', 'vehicle_claim', 'auto_make', 'auto_year',
       'fraud_reported'],
      dtype='object')
```

In [14]:

```python
# Check the values present inside the object type columns

for col in df.columns:
    if df[col].dtype == 'object':
        print(col, df[col].unique())
```

```
policy_state ['OH' 'IN' 'IL']
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-famil
y']
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']
incident_state ['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']
property_damage ['YES' '?' 'NO']
police_report_available ['YES' '?' 'NO']
auto_make ['Saab' 'Mercedes' 'Dodge' 'Chevrolet' 'Accura' 'Nissan' 'Audi' 'Toyota'
 'Ford' 'Suburu' 'BMW' 'Jeep' 'Honda' 'Volkswagen']
fraud_reported ['Y' 'N']
```

In [15]:

```python
# collision_type, property_damage, police_report_available has '?' symbol.
# We can treat the '?' as new category with 'Unknown' value.

df['collision_type'].replace('?','Not Known', inplace=True)
df['property_damage'].replace('?','Not Known', inplace=True)
df['police_report_available'].replace('?','Not Known', inplace=True)
```

In [16]:

```python
# # Installing the pandas profilling for the profile reports
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

```
Collecting https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (http
s://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
  Using cached https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (htt
ps://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
Requirement already satisfied: scipy<1.12,>=1.4.1 in c:\users\shekhar\anaconda3\lib\site-pa
ckages (from ydata-profiling==0.0.dev0) (1.8.1)
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in c:\users\shekhar\anaconda3\lib\si
te-packages (from ydata-profiling==0.0.dev0) (1.4.2)
Requirement already satisfied: matplotlib<=3.7.3,>=3.2 in c:\users\shekhar\appdata\roaming
\python\python39\site-packages (from ydata-profiling==0.0.dev0) (3.5.2)
Collecting pydantic>=2
  Using cached pydantic-2.4.2-py3-none-any.whl (395 kB)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\shekhar\anaconda3\lib\site-pa
ckages (from ydata-profiling==0.0.dev0) (5.4.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\shekhar\anaconda3\lib\site-p
ackages (from ydata-profiling==0.0.dev0) (2.11.3)
Requirement already satisfied: visions[type_image_path]==0.7.5 in c:\users\shekhar\anaconda
3\lib\site-packages (from ydata-profiling==0.0.dev0) (0.7.5)
Requirement already satisfied: numpy<1.26,>=1.16.0 in c:\users\shekhar\appdata\roaming\pyth
```


```python
#Check the values present inside the object type columns

for col in df.columns:
    if df[col].dtype == 'object':
        print(col, df[col].unique())
```

```
policy_state ['OH' 'IN' 'IL']
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-famil
y']
incident_type ['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']
collision_type ['Side Collision' 'Not Known' 'Rear Collision' 'Front Collision']
incident_severity ['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']
authorities_contacted ['Police' 'None' 'Fire' 'Other' 'Ambulance']
incident_state ['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']
property_damage ['YES' 'Not Known' 'NO']
police_report_available ['YES' 'Not Known' 'NO']
auto_make ['Saab' 'Mercedes' 'Dodge' 'Chevrolet' 'Accura' 'Nissan' 'Audi' 'Toyota'
 'Ford' 'Suburu' 'BMW' 'Jeep' 'Honda' 'Volkswagen']
fraud_reported ['Y' 'N']
```


Pandas Profiling

In [17]:

```python
from pandas_profiling import ProfileReport

# Generating the report for the dataset
profile = ProfileReport(df, title= 'Insurance Fraud Claimns Report', html={'style':{'full_width
profile.to_notebook_iframe()

# Saving the output in HTML format to view it
profile.to_file(output_file = 'Insurance_Prediction_Report.html')
```

Exploratory Data Analysis

In [16]:
```
df['police_report_available'].replace('?','Not Known', inplace=True)
# # Installing the pandas profilling for the profile reports
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

```
Collecting https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (https://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
  Using cached https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (https://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
Requirement already satisfied: scipy<1.12,>=1.4.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (1.8.1)
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (1.4.2)
Requirement already satisfied: matplotlib<=3.7.3,>=3.2 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from ydata-profiling==0.0.dev0) (3.5.2)
Collecting pydantic>=2
  Using cached pydantic-2.4.2-py3-none-any.whl (395 kB)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (5.4.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (2.11.3)
Requirement already satisfied: visions[type_image_path]==0.7.5 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (0.7.5)
Requirement already satisfied: numpy<1.26,>=1.16.0 in c:\users\shekhar\appdata\roaming\pyth
```

```
# Check the values present inside the object type columns
for col in df.columns:
    if df[col].dtype == 'object':
        print(col, df[col].unique())
```

```
policy_state ['OH' 'IN' 'IL']
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' 'Not Known' 'Rear Collision' 'Front Collision']
incident_severity ['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']
authorities_contacted ['Police' 'None' 'Fire' 'Other' 'Ambulance']
incident_state ['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']
property_damage ['YES' 'Not Known' 'NO']
police_report_available ['YES' 'Not Known' 'NO']
auto_make [... 'Mercedes' 'Dodge' 'Chevrolet' 'Accura' 'Nissan' 'Audi' 'Toyota'
 'Ford' 'Suburu' 'BMW' 'Jeep' 'Honda' 'Volkswagen']
fraud_reported ['Y' 'N']
```

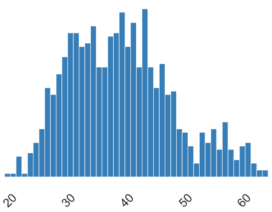
Pandas Profiling

In [17]:
```
from pandas_profiling import ProfileReport

# Generating reports for the dataset
profile = ProfileReport(df, title= 'Insurance Fraud Claimns Report', html={'style':{'full_width
profile.to_notebook_iframe()

# Saving the output in HTML format to view it
profile.to_file(output_file = 'Insurance_Prediction_Report.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	38.948

Minimum	19
Maximum	64
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	7.9 KiB



Quantile statistics

Minimum	19
5-th percentile	26
Q1	32

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

#Exploratory Data Analysis

In [20]:
```
eda_df = df
eda_df.head(10)
```

Out[20]:

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insu
0	328	48	OH	1000	1406.91	0	466132	
1	228	42	IN	2000	1197.22	5000000	468176	
2	134	29	OH	2000	1413.14	5000000	430632	
3	256	41	IL	2000	1415.74	6000000	608117	
4	228	44	IL	1000	1583.91	6000000	610706	
5	256	39	OH	1000	1351.10	0	478456	
6	137	34	IN	1000	1333.35	0	441716	
7	165	37	IL	1000	1137.03	0	603195	
8	27	33	IL	500	1442.99	0	601734	
9	212	42	IL	500	1315.68	0	600983	

In [18]:
```
!pip install pyyaml==5.4.1
```

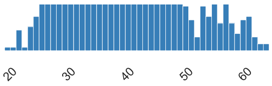
```
Requirement already satisfied: pyyaml==5.4.1 in c:\users\shekhar\anaconda3\lib\site-packages (5.4.1)
```

In [19]:
```
import plotly.graph_objects as go
import plotly.express as px
```

Data Distribution using Histogram

In [21]:
```
def count_plot(df):
    """
```

In [20]:
```python
eda_df = df
eda_df.head(10)
```



Out[20]:

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insu
0	328	48	OH	1000	1406.91	0	466132	
1	228	42	IN	2000	1197.22	5000000	468176	
2	134	29	OH	2000	1413.14	5000000	430632	
3	256	41	IL	2000	1415.74	6000000	608117	
4	228	44	IL	1000	1583.91	6000000	610706	
5	256	39	OH	1000	1351.10	0	478456	
6	137	34	IN	1000	1333.35	0	441716	
7	165	37	IL	1000	1137.03	0	603195	
8	27	33	IL	500	1442.99	0	601734	
9	212	42	IL	500	1315.68	0	600983	

Quantile statistics

Minimum	19
5-th percentile	26
Q1	32

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [18]:
```python
!pip install pyyaml==5.4.1
```

```
Requirement already satisfied: pyyaml==5.4.1 in c:\users\shekhar\anaconda3\lib\site-packages
(5.4.1)
```

In [19]:
```python
import plotly.graph_objects as go
import plotly.express as px
```

Data Distribution using Histogram

In [21]:
```python
def count_plot(df):
    """
        Function is resposible for printing out the count plot on sys.out
    """
    for col in df.columns:
        fig = px.histogram(df, col, nbins=20, width=500, height=400)
        fig.show()
count_plot(eda_df)
```

###Correlation

In [22]:
```python
# Correlation heatmap for all numeric values
plt.figure(figsize = (18,10))
sns.heatmap(eda_df.corr(), annot=True)
```

Out[22]:
```
<AxesSubplot:>
```

Following observations were made

1. months_as_customer is highly correlated with age
2. total_claim_amount is highly correlated with injury_claim and vehicle_claim and property_claim

In [25]:
```python
#Is the age of person have any prominent relationship with  insuarnce fruad  ?
from matplotlib import pyplot as plt
import seaborn as sns
plt.figure(figsize=(15,8))
sns.countplot(data=eda_df, x='age', hue='fraud_reported')
```

In [23]:
```python
# Let's remove months_as_customer and total_claim_amount column as they provides high correlatio
eda_df.drop(['months_as_customer', 'total_claim_amount'], axis=1, inplace=True)
```

Out[25]:
```
<AxesSubplot:xlabel='age', ylabel='count'>
```

In [24]:
```python
# Correlation heatmap for all numeric values
plt.figure(figsize=(18,10))
sns.heatmap(eda_df.corr(), annot=True)
```

Out[24]:
```
<AxesSubplot:>
```

In [26]:
```python
#Did highly educated policy holder are involved in fruad on high scale ?
plt.figure(figsize=(15,8))
sns.countplot(data=eda_df, x='insured_education_level', hue='fraud_reported')
```

Out[26]:
```
<AxesSubplot:xlabel='insured_education_level', ylabel='count'>
```

###Hypothesis Based on EDA 1.Is the age of person have any prominent relationship with insuarnce fruad ?

2.Did highly educated policy holder are involved in fruad on high scale ?

3.which incident type have more probability of fraud?

4.which incident_severity have more impact on fraud?

5.which city incident are leads to fraud ?

6.does more old the auto/vehicle more the probability to get lead to fraud ?

In [27]:
```python
#which incident type have more probability of  fraud?
plt.figure(figsize=(15,8))
sns.countplot(data=eda_df, x='incident_type', hue='fraud_reported')
```

Out[27]:
```
<AxesSubplot:xlabel='incident_type', ylabel='count'>
```

In [28]:
```python
#which incident_severity have more impact on  fraud?
plt.figure(figsize=(15,8))
sns.countplot(data=eda_df, x='incident_severity', hue='fraud_reported')
```

Out[28]:
```
<AxesSubplot:xlabel='incident_severity', ylabel='count'>
```

In [29]:
```python
#which state incident are leads to fraud ?
plt.figure(figsize=(15,8))
```

```
In [25]: #Is the age of person have any prominent relationship with  insuarnce fraud  ?
from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(15,8))
sns.countplot(data=eda_df, x='age', hue='fraud_reported')
```

Out[25]: <AxesSubplot:xlabel='age', ylabel='count'>

Following observations were made

1. months_as_customer is highly correlated with age
2. total_claim_amount is highly correlated with injury_claim and vehicle_claim and property_claim

```
In [23]: # Let's remove months_as_customer and total_claim_amount column as they provides high correlatic
eda_df.drop(['months_as_customer', 'total_claim_amount'], axis=1, inplace=True)
```

```
In [24]: #Corrrelation heatmap for all numeric values
plt.figure(figsize=(18,10))
sns.heatmap(eda_df.corr(), annot=True)
```

Out[24]: <AxesSubplot:>

```
In [26]: #Did highly educated policy holder are involved in fruad on high scale ?
plt.figure(figsize=(15,8))

sns.countplot(data=eda_df, x='insured_education_level', hue='fraud_reported')
```

Out[26]: <AxesSubplot:xlabel='insured_education_level', ylabel='count'>

###Hypothesis Based on EDA 1.Is the age of person have any prominent relationship with insuarnce fraud ?

2.Did highly educated policy holder are involved in fruad on high scale ?

3.which incident type have more probability of fraud?

4.which incident severity have more impact on fraud?

5.which city incident are leads to fraud ?

6.does more old the auto/vehicle more the probability to get lead to fraud ?

```
In [27]: #which incident type have more probability of  fraud?
plt.figure(figsize=(15,8))

sns.countplot(data=eda_df, x='incident_type', hue='fraud_reported')
```

Out[27]: <AxesSubplot:xlabel='incident_type', ylabel='count'>

```
In [28]: #which incident_severity have more impact on  fraud?
plt.figure(figsize=(15,8))

sns.countplot(data=eda_df, x='incident_severity', hue='fraud_reported')
```

Out[28]: <AxesSubplot:xlabel='incident_severity', ylabel='count'>

```
In [29]: #which state incident are leads to fraud ?
plt.figure(figsize=(15,8))

sns.countplot(data=eda_df, x='incident_state', hue='fraud_reported')
```

Out[29]: <AxesSubplot:xlabel='incident_state', ylabel='count'>

```
In [30]: 
#does more old the auto/vehicle more the probability to get lead to fraud ?
plt.figure(figsize=(15,8))

sns.countplot(data=eda_df, x='auto_year', hue='fraud_reported')
```

Out[30]: <AxesSubplot:xlabel='auto_year', ylabel='count'>

#Data Preprocessing (One Hot)

Since there are multiple categorical columns, we need to processed it using multiple methods.

For the current problem statement we would be using:

1. One hot encoding (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjhybaXiYD8AhUqS2wGHXrXC48QFno one-hot-encode-data-in-machine-learning%2F&usg=AOvVaw0wM6DDmQLIcewKfleh-O-v)
2. Weight of evidence encoding (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwii9eekiYD8AhVWSWwGHRljCdoQFnoI of-evidence-woe-and-information-value-iv%2F&usg=AOvVaw0L9j12BFxR75K46dbAi_VR)

Handling columns

1.Creating copy of eda_df i.e eda data frame into new data frame for data processing

```
In [31]: processed_df=eda_df.copy()
processed_df.head(10)
```

Out[31]:

	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educ
0	48	OH	1000	1406.91	0	466132	MALE	
1	42	IN	2000	1197.22	5000000	468176	MALE	
2	29	OH	2000	1413.14	5000000	430632	FEMALE	
3	41	IL	2000	1415.74	6000000	608117	FEMALE	
4	44	IL	1000	1583.91	6000000	610706	MALE	
5	39	OH	1000	1351.10	0	478456	FEMALE	
6	34	IN	1000	1333.35	0	441716	MALE	
7	37	IL	1000	1137.03	0	603195	MALE	
8	33	IL	500	1442.99	0	601734	FEMALE	
9	42	IL	500	1315.68	0	600983	MALE	

Handling rows

```
In [32]: # Check the values present inside the object type columns

for col in df.columns:
    if df[col].dtype == 'object':
        print(col, df[col].unique())
```

```
policy_state ['OH' 'IN' 'IL']
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-famil
y']
incident_type ['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']
collision_type ['Side Collision' 'Not Known' 'Rear Collision' 'Front Collision']
incident_severity ['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']
authorities_contacted ['Police' 'None' 'Fire' 'Other' 'Ambulance']
incident_state ['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']
property_damage ['YES' 'Not Known' 'NO']
police_report_available ['YES' 'Not Known' 'NO']
auto_make ['Saab' 'Mercedes' 'Dodge' 'Chevrolet' 'Accura' 'Nissan' 'Audi' 'Toyota'
 'Ford' 'Suburu' 'BMW' 'Jeep' 'Honda' 'Volkswagen']
fraud_reported ['Y' 'N']
```

1. fraud_reported is our target column in string format such as "Yes","No" we will convert it to numeric indication "Yes" = 1 and "No"= 0
2. insured_sex also have gender column in format such as "male" and "female" we will convert that to numeric indicating "male"=1 "female"= 0

```
processed_df.head(10)
```

Out[31]:

	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educ
0	48	OH	1000	1406.91	0	466132	MALE	
1	42	IN	2000	1197.22	5000000	468176	MALE	
2	29	OH	2000	1413.14	5000000	430632	FEMALE	
3	41	IL	2000	1415.74	6000000	608117	FEMALE	
4	44	IL	1000	1583.91	6000000	610706	MALE	
5	39	OH	1000	1351.10	0	478456	FEMALE	
6	34	IN	1000	1333.35	0	441716	MALE	
7	37	IL	1000	1137.03	0	603195	MALE	
8	33	IL	500	1442.99	0	601734	FEMALE	
9	42	IL	500	1315.68	0	600983	MALE	

Handling rows

In [32]:

```
# Check the values present inside the object type columns
for col in df.columns:
    if df[col].dtype == 'object':
        print(col, df[col].unique())
```

```
policy_state ['OH' 'IN' 'IL']
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-famil
y']
incident_type ['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']
collision_type ['Side Collision' 'Not Known' 'Rear Collision' 'Front Collision']
incident_severity ['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']
authorities_contacted ['Police' 'None' 'Fire' 'Other' 'Ambulance']
incident_state ['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']
property_damage ['YES' 'Not Known' 'NO']
police_report_available ['YES' 'Not Known' 'NO']
auto_make ['Saab' 'Mercedes' 'Dodge' 'Chevrolet' 'Accura' 'Nissan' 'Audi' 'Toyota'
 'Ford' 'Suburu' 'BMW' 'Jeep' 'Honda' 'Volkswagen']
fraud_reported ['Y' 'N']
```

1. `fraud_reported` is our target column in string format such as "Yes","No" we will convert it to numeric indication "Yes" = 1 and "No"= 0
2. `insured_sex` also have gender column in format such as "male" and "female" we will convert that to numeric indicating "male"=1 "female"= 0
3. `education_level` and `incident_severity` is replaced with Ordinal Encoding

In [33]:

```
processed_df.fraud_reported.replace(('Y','N'),(1,0),inplace=True)
processed_df.insured_sex.replace(('MALE','FEMALE'),(1,0),inplace=True)
processed_df.insured_education_level.replace(('MD', 'PhD', 'Associate','Masters','High School',
processed_df.incident_severity.replace(('Trivial Damage','Minor Damage','Major Damage','Total Lo
processed_df.head(10)
```

Out[33]:

	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educ
0	48	OH	1000	1406.91	0	466132	1	
1	42	IN	2000	1197.22	5000000	468176	1	
2	29	OH	2000	1413.14	5000000	430632	0	
3	41	IL	2000	1415.74	6000000	608117	0	
4	44	IL	1000	1583.91	6000000	610706	1	
5	39	OH	1000	1351.10	0	478456	0	
6	34	IN	1000	1333.35	0	441716	1	
7	37	IL	1000	1137.03	0	603195	1	
8	33	IL	500	1442.99	0	601734	0	
9	42	IL	500	1315.68	0	600983	1	

In [35]:

```
from sklearn.preprocessing import StandardScaler
# Importing the standard scaler class
scaler = StandardScaler()
```

In [34]:

```
# Applying one hot encoding using get_dummies method
one_hot_df = pd.get_dummies(processed_df, drop_first=True)
one_hot_df.head(10)
```

Out[34]:

	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_education_level	c
0	48	1000	1406.91	0	466132	1	4	
1	42	2000	1197.22	5000000	468176	1	4	
2	29	2000	1413.14	5000000	430632	0	5	
3	41	2000	1415.74	6000000	608117	0	5	
4	44	1000	1583.91	6000000	610706	1	3	
5	39	1000	1351.10	0	478456	0	5	
6	34	1000	1333.35	0	441716	1	5	
7	37	1000	1137.03	0	603195	1	3	
8	33	500	1442.99	0	601734	0	5	
9	42	500	1315.68	0	600983	1	5	

In [36]:

```
# Dropping the target columns as scaling is not required on that
scaled_df = one_hot_df.drop('fraud_reported', axis=1)
```

In [37]:

```
# Scaling the dataframe
one_hot_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [38]:

```
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['fraud_reported']
x = one_hot_scaled_df
```

In [39]:

```
from sklearn.decomposition import PCA

pca = PCA()

# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```

In [40]:

```
# Variance explained by each principle component
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Feature Selection (One Hot)

Due to one hot, we got large set of columns. To select the best columns among them we need to perform Principle Component Analysis (PCA).

Before PCA we need to standardize and scale all the columns using Standard Scaler.

Out[40]:

	0
0	0.076126
1	0.102400
2	0.126466

In [34]:
```python
# Applying one hot encoding using get_dummies method
one_hot_df = pd.get_dummies(processed_df, drop_first=True)
one_hot_df.head(10)
```

Out[34]:

	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_education_level
0	48	1000	1406.91	0	466132	1	4
1	42	2000	1197.22	5000000	468176	1	4
2	29	2000	1413.14	5000000	430632	0	5
3	41	2000	1415.74	6000000	608117	0	5
4	44	1000	1583.91	6000000	610706	1	3
5	39	1000	1351.10	0	478456	0	5
6	34	1000	1333.35	0	441716	1	5
7	37	1000	1137.03	0	603195	1	3
8	33	500	1442.99	0	601734	0	5
9	42	500	1315.68	0	600983	1	5

Feature Selection (One Hot)

Due to one hot, we got large set of columns. To select the best columns among them we need to perform Principle Component Analysis (PCA).

Before PCA we need to standardize and scale all the columns using Standard Scaler.

In [35]:
```python
from sklearn.preprocessing import StandardScaler
```

In [36]:
```python
# Importing the standard scaler class
scaler = StandardScaler()
```

In [36]:
```python
# Dropping the target columns as scaling is not required on that
scaled_df = one_hot_df.drop(['fraud_reported'], axis=1)
```

In [37]:
```python
# Scaling the dataframe
one_hot_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [38]:
```python
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['fraud_reported']
x = one_hot_scaled_df
```

In [39]:
```python
from sklearn.decomposition import PCA

pca = PCA()

# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```

In [40]:
```python
# Variance explained by each principle component
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Out[40]:

	0
0	0.076126
1	0.102400
2	0.126466
3	0.149562
4	0.172372
5	0.194346
6	0.216174
7	0.237454
8	0.258261
9	0.278391
10	0.298387

In [41]:
```python
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```

```
C:\Users\SHEKHAR\AppData\Local\Temp\ipykernel_10448\2189574362.py:6: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI ba
ckend, so cannot show the figure.
```

We can see that around 99% of variance is being explained by 66 components. So instead of giving all columns as input in our algorithm let's use these principle components instead

In [42]:
```python
pca = PCA(n_components=66)
pca_onehot_df = pca.fit_transform(x)

column_name = [f'PC-{i}' for i in range(1, 67)]
print(column_name)

principal_onehot_df = pd.DataFrame(pca_onehot_df,columns=column_name)
principal_onehot_df.head()
```

```
['PC-1', 'PC-2', 'PC-3', 'PC-4', 'PC-5', 'PC-6', 'PC-7', 'PC-8', 'PC-9', 'PC-10', 'PC-11', 'PC
-12', 'PC-13', 'PC-14', 'PC-15', 'PC-16', 'PC-17', 'PC-18', 'PC-19', 'PC-20', 'PC-21', 'PC-2
2', 'PC-23', 'PC-24', 'PC-25', 'PC-26', 'PC-27', 'PC-28', 'PC-29', 'PC-30', 'PC-31', 'PC-32',
'PC-33', 'PC-34', 'PC-35', 'PC-36', 'PC-37', 'PC-38', 'PC-39', 'PC-40', 'PC-41', 'PC-42', 'PC-
43', 'PC-44', 'PC-45', 'PC-46', 'PC-47', 'PC-48', 'PC-49', 'PC-50', 'PC-51', 'PC-52', 'PC-53',
'PC-54', 'PC-55', 'PC-56', 'PC-57', 'PC-58', 'PC-59', 'PC-60', 'PC-61', 'PC-62', 'PC-63', 'PC-
64', 'PC-65', 'PC-66']
```

Out[42]:

	PC-1	PC-2	PC-3	PC-4	PC-5	PC-6	PC-7	PC-8	PC-9	PC-10	PC-1
0	-1.150205	-1.851882	2.085950	2.037077	0.554450	-0.431060	-0.017820	-1.173733	-0.877127	-0.353131	0.97192
1	4.250807	-0.634484	-1.126089	-2.105770	-0.103788	-1.943815	-0.810049	-1.493347	0.403880	-2.692573	-1.12425
2	0.462334	1.517464	0.466534	1.050595	-1.461622	1.140974	1.415316	-2.125614	-0.028492	-0.084200	-0.37689
3	-0.022272	-0.297652	-0.137941	-0.959080	-1.082542	2.342532	2.223181	0.200544	3.316820	0.382864	0.57734
4	4.768356	0.076853	-0.594282	0.754430	-0.623008	0.805024	-1.049508	-1.466508	1.559798	-0.091410	-0.00228

#Data Modeling (One Hot)

In [43]:
```python
import sklearn.metrics as sm
from sklearn.model_selection import KFold

#function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))

    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

```
principal_onehot_df = pd.DataFrame(pca_onehot_df,columns=column_name)
import sklearn.metrics as sm
from sklearn.model_selection import KFold

#function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))

    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

In [43]:
```
principal_onehot_df.head()
```

```
['PC-1', 'PC-2', 'PC-3', 'PC-4', 'PC-5', 'PC-6', 'PC-7', 'PC-8', 'PC-9', 'PC-10', 'PC-11', 'PC
12', 'PC-13', 'PC-14', 'PC-15', 'PC-16', 'PC-17', 'PC-18', 'PC-19', 'PC-20', 'PC-21', 'PC-2
2', 'PC-23', 'PC-24', 'PC-25', 'PC-26', 'PC-27', 'PC-28', 'PC-29', 'PC-30', 'PC-31', 'PC-32',
'PC-33', 'PC-34', 'PC-35', 'PC-36', 'PC-37', 'PC-38', 'PC-39', 'PC-40', 'PC-41', 'PC-42', 'PC-
43', 'PC-44', 'PC-45', 'PC-46', 'PC-47', 'PC-48', 'PC-49', 'PC-50', 'PC-51', 'PC-52', 'PC-53',
'PC-54', 'PC-55', 'PC-56', 'PC-57', 'PC-58', 'PC-59', 'PC-60', 'PC-61', 'PC-62', 'PC-63', 'PC-
64', 'PC-65', 'PC-66']
```

Out[42]:

	PC-1	PC-2	PC-3	PC-4	PC-5	PC-6	PC-7	PC-8	PC-9	PC-10	PC-1
0	-1.150205	-1.851882	2.085950	2.037077	0.554450	-0.431060	-0.017820	-1.173733	-0.877127	-0.353131	0.97192
1	4.250807	-0.634484	-1.126089	-2.105779	-0.103788	-1.943815	0.810049	-1.493347	0.403880	-2.692573	-1.12425
2	0.462334	1.517464	0.466534	1.050595	-1.461622	1.140974	1.415316	-2.125614	-0.028492	-0.084200	-0.37689
3	-0.022272	-0.297652	-0.137941	-0.959080	-1.082542	2.342532	2.223181	0.200544	3.316820	0.382864	0.57734
4	4.768356	0.076853	-0.594282	0.754430	-0.623008	0.805024	-1.049508	-1.466508	1.559798	-0.091410	-0.00228

#Data Modeling (One Hot)

##Random Forest

In [44]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import lightgbm as lgb
```

In [45]:
```
y=one_hot_df['fraud_reported']
X= principal_onehot_df
```

In [46]:
```
columns = ["Algorithm", "Encoding", "Accuracy"]
data = []
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
result= ["Random Forest", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

########
Results
########

Average accuracy score:  0.7489999999999999
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.
```

##Decision Tree

In [48]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.641
```

##XGBoost

In [49]:
```
from xgboost import XGBClassifier
result= ["XGBoost", "OneHot"]
# on WoE data
model=XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7270000000000001
```

##Adaboost

In [50]:
```
from sklearn.ensemble import AdaBoostClassifier
```

```
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.
```

##Decision Tree

In [48]:

```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########
Average accuracy score:  0.641
```

##XGBoost

In [49]:

```python
from xgboost import XGBClassifier
result= ["XGBoost", "OneHot"]
# on WoE data
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7489999999999999
########
Results
########

Average accuracy score:  0.7270000000000001
```

##Adaboost

In [50]:

```python
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.6990000000000001
```

Logistic Regression

In [51]:

```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result= ["LogisticRegression", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.732
```

SVM

In [52]:

```python
from sklearn.svm import SVC

model = SVC()
result= ["SVM", "OneHot"]

result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.
```

SVM

In [52]:
```python
from sklearn.svm import SVC

model = SVC()
result= ["SVM", "OneHot"]

result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.


########
Results
########

Average accuracy score:  0.753

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Undefine
dMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division`
parameter to control this behavior.
```

In [55]:
```python
! pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\shekhar\anaconda3\lib\site-packag
es (2.5.1.post0)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\shekhar\anaconda3\lib\site-packa
ges (from category_encoders) (0.13.2)
Requirement already satisfied: pandas>=1.0.5 in c:\users\shekhar\anaconda3\lib\site-packages
(from category_encoders) (1.4.2)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\shekhar\anaconda3\lib\site-pac
kages (from category_encoders) (1.0.2)
Requirement already satisfied: numpy>=1.14.0 in c:\users\shekhar\appdata\roaming\python\python
39\site-packages (from category_encoders) (1.23.1)
Requirement already satisfied: patsy>=0.5.1 in c:\users\shekhar\anaconda3\lib\site-packages (f
rom category_encoders) (0.5.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (f
rom category_encoders) (1.8.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\shekhar\appdata\roaming\pyth
on\python39\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\shekhar\anaconda3\lib\site-packages (f
rom pandas>=1.0.5->category_encoders) (2021.3)
Requirement already satisfied: six in c:\users\shekhar\appdata\roaming\python\python39\site-pa
ckages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shekhar\anaconda3\lib\site-pac
kages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\shekhar\anaconda3\lib\site-packages (f
rom scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: packaging>=21.3 in c:\users\shekhar\appdata\roaming\python\pyth
on39\site-packages (from statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\shekhar\appdata\roaming\py
thon\python39\site-packages (from packaging>=21.3->statsmodels>=0.9.0->category_encoders) (3.
0.9)
```

KNN

In [53]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.661
```

#Data Preprocessing (WOE Encoding)

Handling Columns and Rows

In [54]:
```python
processed_df = eda_df.copy()
processed_df.fraud_reported.replace(('Y','N'),(1,0),inplace=True)
```

```
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
```

In [55]:
```python
! pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\shekhar\anaconda3\lib\site-packages (2.5.1.post0)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (0.13.2)
Requirement already satisfied: pandas>=1.0.5 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.4.2)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.0.2)
Requirement already satisfied: numpy>=1.14.0 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from category_encoders) (1.23.1)
Requirement already satisfied: patsy>=0.5.1 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (0.5.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.8.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\shekhar\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2021.3)
Requirement already satisfied: six in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\shekhar\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: packaging>=21.3 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from packaging>=21.3->statsmodels>=0.9.0->category_encoders) (3.0.9)
```

KNN

In [53]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result = ["GaussianNB", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Result
########
Average accuracy score 0.691
```

#Data Preprocessing (WOE Encoding)

Handling Columns and Rows

In [54]:
```python
processed_df = eda_df.copy()
processed_df.fraud_reported.replace(('Y','N'),(1,0),inplace=True)
```

In [56]:
```python
for col in processed_df.columns:
    if processed_df[col].dtype == 'object':
        print(processed_df[col].value_counts())
```

```
OH    352
IL    338
IN    310
Name: policy_state, dtype: int64
FEMALE    537
MALE      463
Name: insured_sex, dtype: int64
JD             161
High School    160
Associate      145
MD             144
Masters        143
PhD            125
College        122
Name: insured_education_level, dtype: int64
machine-op-inspct    93
prof-specialty       85
tech-support         78
sales                76
exec-managerial      76
craft-repair         74
transport-moving     72
other-service        71
priv-house-serv      71
armed-forces         69
adm-clerical         65
protective-serv      63
handlers-cleaners    54
farming-fishing      53
Name: insured_occupation, dtype: int64
own-child         183
other-relative    177
```

In [56]:
```python
for col in processed_df.columns:
    if processed_df[col].dtype == 'object':
        print(processed_df[col].value_counts())
```

```
OH    352
IL    338
IN    310
Name: policy_state, dtype: int64
FEMALE    537
MALE      463
Name: insured_sex, dtype: int64
JD             161
High School    160
Associate      145
MD             144
Masters        143
PhD            125
College        122
Name: insured_education_level, dtype: int64
machine-op-inspct    93
prof-specialty       85
tech-support         78
sales                76
exec-managerial      76
craft-repair         74
transport-moving     72
other-service        71
priv-house-serv      71
armed-forces         69
adm-clerical         65
protective-serv      63
handlers-cleaners    54
farming-fishing      53
Name: insured_occupation, dtype: int64
own-child         183
other-relative    177
not-in-family     174
husband           170
wife              155
unmarried         141
Name: insured_relationship, dtype: int64
Multi-vehicle Collision     419
Single Vehicle Collision    403
Vehicle Theft                94
Parked Car                   84
Name: incident_type, dtype: int64
Rear Collision     292
Side Collision     276
Front Collision    254
Not Known          178
Name: collision_type, dtype: int64
Minor Damage      354
Total Loss        280
Major Damage      276
Trivial Damage     90
Name: incident_severity, dtype: int64
Police       292
Fire         223
Other        198
Ambulance    196
None          91
Name: authorities_contacted, dtype: int64
NY    262
SC    248
WV    217
VA    110
NC    110
PA     30
OH     23
Name: incident_state, dtype: int64
Not Known    360
NO           338
YES          302
Name: property_damage, dtype: int64
Not Known    343
NO           343
YES          314
Name: police_report_available, dtype: int64
Saab          80
Dodge         80
Suburu        80
Nissan        78
Chevrolet     76
Ford          72
BMW           72
Toyota        70
Audi          69
Accura        68
Volkswagen    68
Jeep          67
Mercedes      65
Honda         55
Name: auto_make, dtype: int64
```

In [58]:
```python
woe_df.head()
```

Out[58]:

	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educ
0	48	0.062709	1000	1406.91	0	466132	0.075555	
1	42	0.044554	2000	1197.22	5000000	468176	0.075555	
2	29	0.062709	2000	1413.14	5000000	430632	-0.067571	
3	41	-0.102371	2000	1415.74	6000000	608117	-0.067571	
4	44	-0.102371	1000	1583.91	6000000	610706	0.075555	

Feature Selection (WOE)

In [59]:
```python
from sklearn.preprocessing import StandardScaler
```

```python
# Using StandardScaler class
scaler = StandardScaler()
```

In [57]:
```python
# Applying WOE Encoding to the Dataframe
from category_encoders.woe import WOEEncoder
WOE_encoder = WOEEncoder()
categorical_cols = ['policy_state', 'insured_sex', 'insured_education_level', 'insured_occupati
for column in categorical_cols:
    processed_df[column] = WOE_encoder.fit_transform(processed_df[column], processed_df['fraud_r
```

In [60]:
```python
# Dropping the target column as scaling is not required on it.
scaled_df = woe_df.drop('fraud_reported', axis=1)
```

In [61]:
```python
woe_df_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

```python
woe_df = processed_df.copy()
```

In [62]:
```python
y = one_hot_df['fraud_reported']
x = woe_df_scaled_df
```

```
YES          302
Name: property_damage, dtype: int64
Not Known    343
NO           343
YES          314
Name: police_report_available, dtype: int64
Saab          80
Dodge         80
Suburu        80
Nissan        78
Chevrolet     76
Ford          72
BMW           72
Toyota        70
Audi          69
Acura         68
Volkswagen    68
Jeep          67
Mercedes      65
Honda         55
Name: auto_make, dtype: int64
```

In [58]:
```python
woe_df.head()
```

Out[58]:

	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educ
0	48	0.062709	1000	1406.91	0	466132	0.075555	
1	42	0.044654	2000	1197.22	5000000	468176	0.075555	
2	29	0.062709	2000	1413.14	5000000	430632	-0.067571	
3	41	-0.102371	2000	1415.74	6000000	608117	-0.067571	
4	44	-0.102371	1000	1583.91	6000000	610706	0.075555	

Feature Selection (WOE)

In [59]:
```python
from sklearn.preprocessing import StandardScaler
```

In [57]:
```python
# Applying WOE Encoding to the Dataframe
from category_encoders.woe import WOEEncoder
WOE_encoder = WOEEncoder()
categorical_cols = ['policy_state', 'insured_sex', 'insured_education_level', 'insured_occupat
for column in categorical_cols:
    processed_df[column] = WOE_encoder.fit_transform(processed_df[column], processed_df['fraud_r
woe_df = processed_df.copy()
```

```python
# Using StandardScaler class
scaler = StandardScaler()
```

In [60]:
```python
# Dropping the target column as scaling is not required on it.
scaled_df = woe_df.drop('fraud_reported', axis=1)
```

In [61]:
```python
woe_df_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [62]:
```python
y = one_hot_df['fraud_reported']
x = woe_df_scaled_df
```

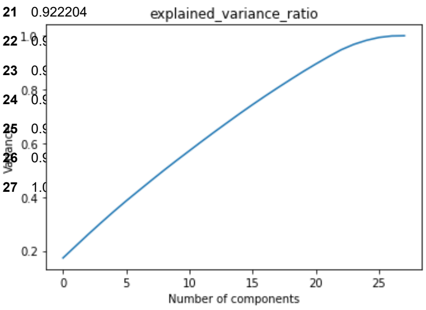
In [63]:
```python
from sklearn.decomposition import PCA

pca = PCA()

# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```

In [64]:
```python
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Out[64]:

	0
0	0.173348
1	0.217504
2	0.261429
3	0.304032
4	0.345631
5	0.385753
6	0.424229
7	0.462451
8	0.500173
9	0.536739
10	0.572417
11	0.608033
12	0.643018
13	0.677278
14	0.711192
15	0.744087
16	0.776089
17	0.807061
18	0.837335
19	[illegible]
20	0.895163
21	0.922204
22	0.9[illegible]
23	0.9[illegible]
24	0.9[illegible]
25	0.9[illegible]
26	0.9[illegible]
27	1.0[illegible]

In [65]:
```python
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```



We can see that around 99% of variance is being explained by 25 components. So instead of giving all columns as input in our algorithm let's use these principle components instead

In [66]:
```python
pca = PCA(n_components=25)
pca_woe_df = pca.fit_transform(x)

column_name = [f'PC-{i}' for i in range(1, 26)]

principal_woe_df = pd.DataFrame(pca_woe_df, columns=column_name)
principal_woe_df.head()
```

16 0.776089

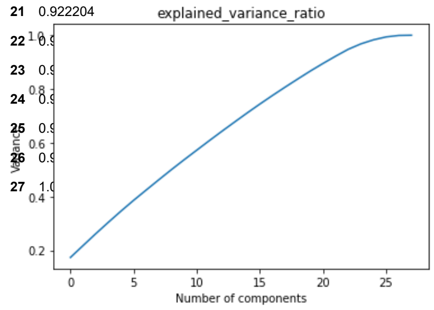
17 0.807060

18 0.837335

20 0.895163

21 0.922204

In [65]:
```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```



We can see that around 99% of variance is being explained by 25 components. So instead of giving all columns as input in our algorithm let's use these principle components instead

In [66]:
```
pca = PCA(n_components=25)
pca_woe_df = pca.fit_transform(x)

column_name = [f'PC-{i}' for i in range(1, 26)]

principal_woe_df = pd.DataFrame(pca_woe_df, columns=column_name)
principal_woe_df.head()
```

Out[66]:

	PC-1	PC-2	PC-3	PC-4	PC-5	PC-6	PC-7	PC-8	PC-9	PC-10	PC-1
0	-0.997570	0.780926	-0.533583	0.184909	-0.679628	-1.819256	-1.283710	-0.288213	0.726652	-1.200167	0.26542
1	4.026343	0.842083	1.913565	0.716060	-0.067698	1.320849	0.535631	1.558847	0.755646	0.518444	1.28170
2	0.253120	-0.283566	-1.111903	-0.477507	1.440599	0.566111	-1.542508	-1.292452	-0.435717	0.977837	0.05361
3	-0.529702	2.001171	0.996700	-1.318997	1.149643	1.095464	-1.880315	-1.191650	0.908376	0.964282	1.68231
4	4.738184	0.456561	-0.062727	1.323874	2.119771	0.866995	-0.387641	0.888265	0.054812	-0.839057	-0.12678

Data Modelling (WoE)

In [67]:
```
import sklearn.metrics as sm
from sklearn.model_selection import KFold

# Function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores =[]

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)
        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))

    #display average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))
    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

In [69]:
```
y=woe_df['fraud_reported']
X= principal_woe_df
```

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
result= ["Random Forest", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########
Average accuracy score:  0.7730000000000001
```

Decision Tree

In [71]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########
Average accuracy score:  0.706
```

Random Forest

In [68]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import lightgbm as lgb
```

XGBoost

```
        #Evaluating model
        y_pred = model.predict(x_test)
        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))
    #display average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))
    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

In [69]:
```
y=woe_df['fraud_reported']
X= principal_woe_df
```

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
result= ["Random Forest", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7730000000000001
```

Random Forest

In [68]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import lightgbm as lgb
```

Decision Tree

In [71]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.706
```

XGBoost

In [72]:
```
from xgboost import XGBClassifier
result= ["XGBoost", "WoE"]
# on WoE data
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.787
```

Adaboost

In [73]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.781
```

Logistic Regression

In [74]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result= ["LogisticRegression", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.804
```

KNN

In [76]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.806
```

SVM

In [75]:
```
from sklearn.svm import SVC
model = SVC()
result= ["SVM", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8019999999999999
```

Saving Results for Insurance Fraud Detection

In [77]:
```
results_df = pd.DataFrame(data, columns=columns)
results_df = results_df[['Algorithm', 'Encoding', 'Accuracy']]
results_df.sort_values(by=['Accuracy', 'Encoding'], ascending=False)
```

Out[77]:

	Algorithm	Encoding	Accuracy
13	GaussianNB	WoE	0.806
11	LogisticRegression	WoE	0.804
12	SVM	WoE	0.802
9	XGBoost	WoE	0.787
10	Adaboost	WoE	0.781
7	Random Forest	WoE	0.773

```
result.extend(evaluate_model(model, X, y))
data.append(result)
```

KNN

```
########
Results
########
Average accuracy score:  0.804
```

In [76]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "WoE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

SVM

```
########
Results
########
Average accuracy score:  0.806
```

In [75]:
```
from sklearn.svm import SVC

model = SVC()
result= ["SVM", "WoE"]

result.extend(evaluate_model(model, X, y))
data.append(result)
```

Saving Results for Insurance Fraud Detection

In [77]:
```
results_df = pd.DataFrame(data, columns=columns)
results_df = results_df[['Algorithm', 'Encoding', 'Accuracy']]
results_df.sort_values(by=['Accuracy', 'Encoding'], ascending=False)
```

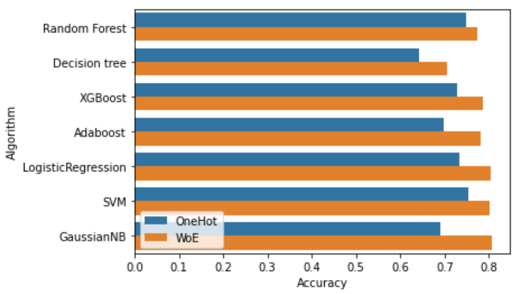
```
########
Results
########
Average accuracy score:  0.8019999999999999
```

Out[77]:

	Algorithm	Encoding	Accuracy
13	GaussianNB	WoE	0.806
11	LogisticRegression	WoE	0.804
12	SVM	WoE	0.802
9	XGBoost	WoE	0.787
10	Adaboost	WoE	0.781
7	Random Forest	WoE	0.773
5	SVM	OneHot	0.753
0	Random Forest	OneHot	0.749
4	LogisticRegression	OneHot	0.732
2	XGBoost	OneHot	0.727
8	Decision tree	WoE	0.706
3	Adaboost	OneHot	0.699
6	GaussianNB	OneHot	0.691
1	Decision tree	OneHot	0.641

In [78]:
```
sns.barplot(data=results_df, x='Accuracy', y='Algorithm', hue='Encoding')
plt.legend(loc='lower left')
```

Out[78]: <matplotlib.legend.Legend at 0x15e50c899d0>



In [81]:
```
results_df.to_csv("Insurance_Claims.csv")
```

In [85]:
```
# Reading the Dataset
df=pd.read_csv('insurance_data_final.csv' )

# df = df.drop(df.columns[0], axis = 1) # To drop the first index column
```

INCOME PREDICTION

In [86]:
```
df.head()
```

#Data Collection

- we have data consisting of income and insurance (merged_data.csv). However, this data is not perfect.
- It may contain outliers, missing values and irrelevant (not related to insurance claims) information

Out[86]:

	Work	Marital Status	Hours_Week	Country	Above_Below 50K	Education	Gender	Occupation	CustomerID	Age	months_
0	Self-emp-not-inc	Married-civ-spouse	13	United-States	<=50K	Bachelors	Male	Exec-managerial	902	50	
1	Private	Divorced	40	United-States	<=50K	HS-grad	Male	Handlers-cleaners	5506	38	
2	Private	Married-civ-spouse	40	United-States	<=50K	11th	Male	Handlers-cleaners	3869	53	
3	Private	Married-civ-spouse	40	Cuba	<=50K	Bachelors	Female	Prof-specialty	7106	28	
4	Private	Married-civ-spouse	40	United-States	<=50K	Masters	Female	Exec-managerial	1760	37	

In [83]:
```
# Mounting Google Drive
#drive.mount('/content/drive')
```

In [84]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# To get all the rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

Describing some common statistics on numeric columns

Reading the Data

In [87]:
```
# Data description:
df.describe()
```

Out[87]:

In [85]:
```
# Reading the Dataset
df=pd.read_csv('insurance_data_final.csv' )

# df = df.drop(df.columns[0], axis = 1) # To drop the first index column
```

INCOME PREDICTION

In [86]:
```
df.head()
```

Out[86]:

	Work	Marital Status	Hours_Week	Country	Above_Below 50K	Education	Gender	Occupation	CustomerID	Age	months_
0	Self-emp-not-inc	Married-civ-spouse	13	United-States	<=50K	Bachelors	Male	Exec-managerial	902	50	
1	Private	Divorced	40	United-States	<=50K	HS-grad	Male	Handlers-cleaners	5506	38	
2	Private	Married-civ-spouse	40	United-States	<=50K	11th	Male	Handlers-cleaners	3869	53	
3	Private	Married-civ-spouse	40	Cuba	<=50K	Bachelors	Female	Prof-specialty	7106	28	
4	Private	Married-civ-spouse	40	United-States	<=50K	Masters	Female	Exec-managerial	1760	37	

#Data Collection

- we have data consisting of income and insurance (merged_data.csv). However, this data is not perfect.
- It may contain outliers, missing values and irrelevant (not related to insurance claims) information

In [83]:
```
# Mounting Google Drive
#drive.mount('/content/drive')
```

In [84]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# To get all the rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

Describing some common statistics on numeric columns

Reading the Data

In [87]:
```
# Data description:
df.describe()
```

Out[87]:

	Hours_Week	CustomerID	Age	months_as_customer	policy_deductable	policy_annual_premium	ins
count	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.00000	900
mean	40.549667	5035.180111	38.443778	203.954000	1136.000000	1256.40615	50121
std	12.320334	2892.177024	13.579920	115.061996	611.592643	244.05884	7166
min	1.000000	1.000000	17.000000	0.000000	500.000000	433.33000	43010
25%	40.000000	2530.000000	28.000000	115.750000	500.000000	1089.60750	44840
50%	40.000000	5037.500000	37.000000	199.500000	1000.000000	1257.20000	46644
75%	45.000000	7562.000000	47.000000	276.250000	2000.000000	1415.69500	60325
max	99.000000	9998.000000	90.000000	479.000000	2000.000000	2047.59000	62096

Checking the null values present in the dataset

In [88]:
```
df.isna().sum()
```

Out[88]:
```
Work                           0
Marital Status                 0
Hours_Week                     0
Country                        0
Above_Below 50K                0
Education                      0
Gender                         0
Occupation                     0
CustomerID                     0
Age                            0
months_as_customer             0
policy_deductable              0
policy_annual_premium          0
insured_zip                    0
insured_hobbies                0
insured_relationship           0
capital-gains                  0
capital-loss                   0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          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_year                      0
fraud_reported                 0
dtype: int64
```

- We prepare the data column wise, as every column would require different preprocessing

In [90]:
```
def display_basic_information(df, column_name):
    data_type = df[column_name].dtype
    null_values = df[column_name].isna().sum()
    print("###################")
    print("Basic Information:")
    print("###################\n")
    print("column name:", column_name)
    print("data type:", data_type)
    print("total number of null values:", null_values)
    print()

column_headers = list(df.columns)
for i in column_headers:
    display_basic_information(df,i)
```

```
###################
Basic Information:
###################

column name: Work
data type: object
total number of null values: 0

###################
Basic Information:
###################

column name: Marital Status
data type: object
total number of null values: 0

###################
Basic Information:
###################
```

In [89]:
```
# Number of rows and columns of data
print("Number of rows:", len(df))
print("Number of columns:", len(df.columns))
```

```
Number of rows: 9000
Number of columns: 37
```

Data Preparation

```
insured_relationship           0
capital-gains                  0
capital-loss                   0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          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_year                      0
fraud_reported                 0
dtype: int64
```

In [89]:
```python
# Number of rows and columns of data
print("Number of rows:", len(df))
print("Number of columns:", len(df.columns))
```

```
Number of rows: 9000
Number of columns: 37
```

Data Preparation

We prepare the data column wise, as every column would require different preprocessing

In [90]:
```python
def display_basic_information(df, column_name):
    data_type = df[column_name].dtype
    null_values = df[column_name].isna().sum()
    print("###################")
    print("Basic Information:")
    print("###################\n")
    print("column name:", column_name)
    print("data type:", data_type)
    print("total number of null values:", null_values)
    print()

column_headers = list(df.columns)
for i in column_headers:
    display_basic_information(df,i)
```

```
###################
Basic Information:
###################

column name: Work
data type: object
total number of null values: 0
###################
Basic Information:
###################

column name: Marital Status
data type: object
total number of null values: 0
###################
Basic Information:
###################
```

#Exploratory Data Analysis

In [91]:
```python
# As only first 10 columns are required
income_df = df.iloc[:, : 10]

# CustomerID is not required for Knowledge extraction
income_df.drop('CustomerID', axis=1, inplace=True)

income_df.head()
```

Out[91]:

	Work	Marital Status	Hours_Week	Country	Above_Below 50K	Education	Gender	Occupation	Age
0	Self-emp-not-inc	Married-civ-spouse	13	United-States	<=50K	Bachelors	Male	Exec-managerial	50
1	Private	Divorced	40	United-States	<=50K	HS-grad	Male	Handlers-cleaners	38
2	Private	Married-civ-spouse	40	United-States	<=50K	11th	Male	Handlers-cleaners	53
3	Private	Married-civ-spouse	40	Cuba	<=50K	Bachelors	Female	Prof-specialty	28
4	Private	Married-civ-spouse	40	United-States	<=50K	Masters	Female	Exec-managerial	37

In [92]:
```python
# # Installing the pandas profilling for the profile reports
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

```
Collecting https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (https://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
  Using cached https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (https://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
Requirement already satisfied: scipy<1.12,>=1.4.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (1.8.1)
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (1.4.2)
Requirement already satisfied: matplotlib<=3.7.3,>=3.2 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from ydata-profiling==0.0.dev0) (3.5.2)
Collecting pydantic>=2
  Using cached pydantic-2.4.2-py3-none-any.whl (395 kB)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (5.4.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (2.11.3)
Requirement already satisfied: visions[type_image_path]==0.7.5 in c:\users\shekhar\anaconda3\lib\site-packages (from ydata-profiling==0.0.dev0) (0.7.5)
Requirement already satisfied: numpy<1.26,>=1.16.0 in c:\users\shekhar\appdata\roaming\pyth
```

Pandas Profiling

In [94]:
```python
from pandas_profiling import ProfileReport

# Generating reports for the dataset
profile = ProfileReport(income_df, title='Income Detection Report', html={'style':{'full_width
profile.to_notebook_iframe()

# Saving the output in HTML format to view it
profile.to_file(output_file='Income_Detection_Report.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	9

```
s://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
  Using cached https://github.com/pandas-profiling/pandas-profiling/archive/master.zip (htt
ps://github.com/pandas-profiling/pandas-profiling/archive/master.zip)
Requirement already satisfied: scipy<1.12,>=1.4.1 in c:\users\shekhar\anaconda3\lib\site-pa
ckages (from ydata-profiling==0.0.dev0) (1.8.1)
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in c:\users\shekhar\anaconda3\lib\si
te-packages (from ydata-profiling==0.0.dev0) (1.4.2)
Requirement already satisfied: matplotlib<=3.7.3,>=3.2 in c:\users\shekhar\appdata\roaming
\python\python39\site-packages (from ydata-profiling==0.0.dev0) (3.5.2)
Collecting pydantic>=2
  Using cached pydantic-2.4.2-py3-none-any.whl (395 kB)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\shekhar\anaconda3\lib\site-pa
ckages (from ydata-profiling==0.0.dev0) (5.4.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\shekhar\anaconda3\lib\site-p
ackages (from ydata-profiling==0.0.dev0) (2.11.3)
Requirement already satisfied: visions[type_image_path]==0.7.5 in c:\users\shekhar\anaconda
3\lib\site-packages (from ydata-profiling==0.0.dev0) (0.7.5)
Requirement already satisfied: numpy<1.26,>=1.16.0 in c:\users\shekhar\appdata\roaming\pyth
```

Pandas Profiling

In [94]:
```python
from pandas_profiling import ProfileReport

# Generating reports for the dataset
profile = ProfileReport(income_df, title='Income Detection Report', html={'style':{'full_width
profile.to_notebook_iframe()

# Saving the output in HTML format to view it
profile.to_file(output_file='Income_Detection_Report.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	9
Number of observations	9000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	540
Duplicate rows (%)	6.0%
Total size in memory	632.9 KiB
Average record size in memory	72.0 B

Variable types

Categorical	7
Numeric	2

Alerts

Dataset has 540 (6.0%) duplicate rows	Duplicates
`Work` is highly overall correlated with `Occupation`	High correlation
`Marital Status` is highly overall correlated with `Age`	High correlation

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [95]:
```python
!pip install pyyaml==5.4.1
```

```
Requirement already satisfied: pyyaml==5.4.1 in c:\users\shekhar\anaconda3\lib\site-packages
(5.4.1)
```

In [96]:
```python
import plotly.graph_objects as go
import plotly.express as px
```

Preprocessing the Dataframe for Exploratory Data Analysis

In [97]:
```python
eda_df = income_df

# Changing the column name of Above_Below 50k and it's values to 1 and 0
eda_df.rename(columns = {'Above_Below 50K': 'Above 50K'}, inplace=True)

# Replacing the less than equal to values with 0 and greater than equal
#to values with 1

eda_df['Above 50K'].replace(['<=50K', '<=50K.'], 0, inplace=True)
eda_df['Above 50K'].replace(['>50K', '>50K.'], 1, inplace=True)

eda_df.head(10)
```

Out[97]:

	Work	Marital Status	Hours_Week	Country	Above 50K	Education	Gender	Occupation	Age
0	Self-emp-not-inc	Married-civ-spouse	13	United-States	0	Bachelors	Male	Exec-managerial	50
1	Private	Divorced	40	United-States	0	HS-grad	Male	Handlers-cleaners	38
2	Private	Married-civ-spouse	40	United-States	0	11th	Male	Handlers-cleaners	53
3	Private	Married-civ-spouse	40	Cuba	0	Bachelors	Female	Prof-specialty	28
4	Private	Married-civ-spouse	40	United-States	0	Masters	Female	Exec-managerial	37
5	Private	Married-spouse-absent	16	Jamaica	0	9th	Female	Other-service	49
6	Self-emp-not-inc	Married-civ-spouse	45	United-States	1	HS-grad	Male	Exec-managerial	52

In [96]:
```
import plotly.graph_objects as go
import plotly.express as px
```

Preprocessing the Dataframe for Exploratory Data Analysis

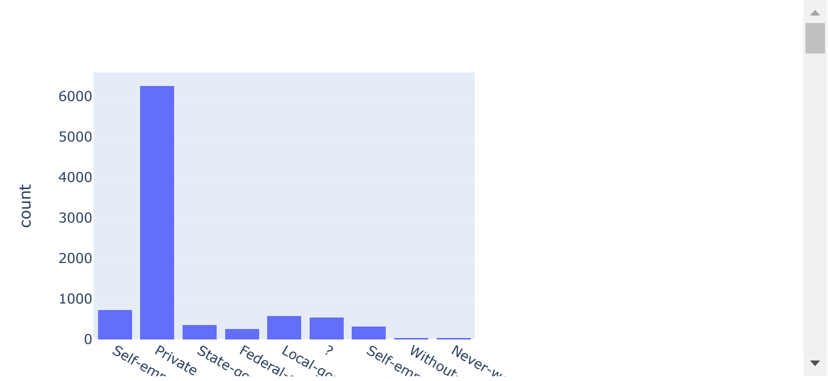
In [97]:
```
eda_df = income_df

# Changing the column name of Above_Below 50k and it's values to 1 and 0
eda_df.rename(columns = {'Above_Below 50K': 'Above 50K'}, inplace=True)

# Replacing the less than equal to values with 0 and greater than equal
#to values with 1

eda_df['Above 50K'].replace(['<=50K', '<=50K.'], 0, inplace=True)
eda_df['Above 50K'].replace(['>50K', '>50K.'], 1, inplace=True)

eda_df.head(10)
```

Out[97]:

	Work	Marital Status	Hours_Week	Country	Above 50K	Education	Gender	Occupation	Age
0	Self-emp-not-inc	Married-civ-spouse	13	United-States	0	Bachelors	Male	Exec-managerial	50
1	Private	Divorced	40	United-States	0	HS-grad	Male	Handlers-cleaners	38
2	Private	Married-civ-spouse	40	United-States	0	11th	Male	Handlers-cleaners	53
3	Private	Married-civ-spouse	40	Cuba	0	Bachelors	Female	Prof-specialty	28
4	Private	Married-civ-spouse	40	United-States	0	Masters	Female	Exec-managerial	37
5	Private	Married-spouse-absent	16	Jamaica	0	9th	Female	Other-service	49
6	Self-emp-not-inc	Married-civ-spouse	45	United-States	1	HS-grad	Male	Exec-managerial	52
7	Private	Never-married	50	United-States	1	Masters	Female	Prof-specialty	31
8	Private	Married-civ-spouse	40	United-States	1	Bachelors	Male	Exec-managerial	42
9	Private	Married-civ-spouse	80	United-States	1	Some-college	Male	Exec-managerial	37

Data Distribution using Histogram

In [98]:
```
def count_plot(df):
  for col in df.columns:
    fig = px.histogram(df, col, nbins=20, width=500, height=400)
    fig.show()
count_plot(eda_df)
```



###Correlation

Box plot for outliers

In [99]:
```
# Correlation heatmap for all numeric values
plt.figure(figsize=(16,10))
sns.heatmap(eda_df.corr(), annot=True)
```

As we can see from the histogram that there are not much significant outliers present in any of the columns and hence outliers removal can be skipped.

Out[99]: <AxesSubplot:>

Hypothesis Based on EDA

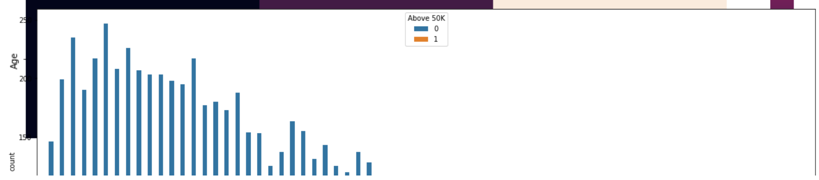
1.Is [illegible]

2.D [illegible]

3.D [illegible]

In [100]:
```
#Is the age of person have any prominent relationship with  insuarnce fruad  ?
from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,8))
sns.countplot(data=eda_df, x='Age', hue='Above 50K')
```

Out[100]: <AxesSubplot:xlabel='Age', ylabel='count'>



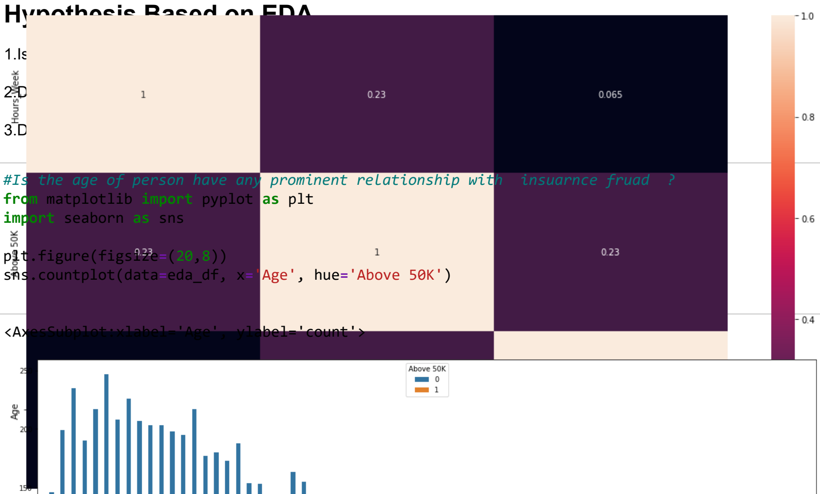
###Correlation

Box plot for outliers

In [99]:
```
# Correlation heatmap for all numeric values
plt.figure(figsize=(10,10))
sns.heatmap(eda_df.corr(), annot=True)
```

As we can see from the histogram that there are not much significant outliers present in any of the columns and hence outliers removal can be skipped.

Out[99]: <AxesSubplot:>



Hypothesis Based on EDA

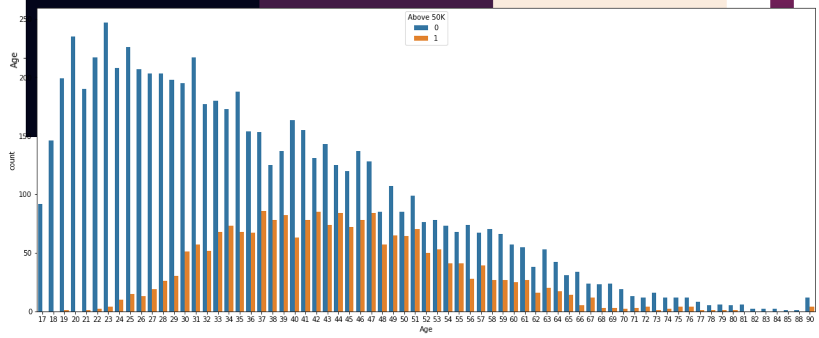
1.Is

2.D

3.D

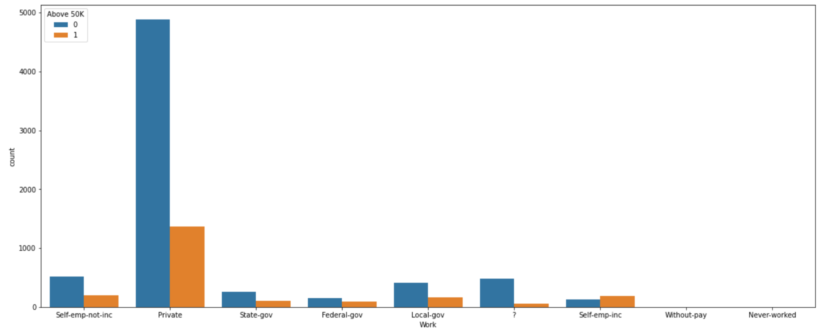
In [100]:
```
#Is the age of person have any prominent relationship with  insuarnce fruad  ?
from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,8))
sns.countplot(data=eda_df, x='Age', hue='Above 50K')
```

Out[100]: <AxesSubplot:xlabel='Age', ylabel='count'>



In [101]:
```
# Does WorkClass affects the income range

from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,8))
sns.countplot(data=eda_df, x='Work', hue='Above 50K')
```

Out[101]: <AxesSubplot:xlabel='Work', ylabel='count'>



Handling columns

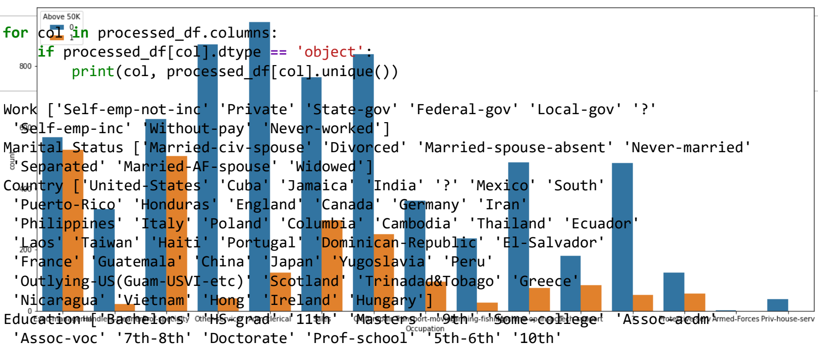
1.Creating copy of the eda data frame into new data frame for data processing

In [102]:
```
# Does Occupation affects the income range

from matplotlib import pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,8))
sns.countplot(data=eda_df, x='Occupation', hue='Above 50K')
```

In [103]:
```
processed_df=eda_df.copy()
```

Out[102]: <AxesSubplot:xlabel='Occupation', ylabel='count'>

Handling rows



In [104]:
```
for col in processed_df.columns:
    if processed_df[col].dtype == 'object':
        print(col, processed_df[col].unique())
```

```
Work ['Self-emp-not-inc' 'Private' 'State-gov' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
Marital Status ['Married-civ-spouse' 'Divorced' 'Married-spouse-absent' 'Never-married'
 'Separated' 'Married-AF-spouse' 'Widowed']
Country ['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary']
Education ['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
Sex [' Male' ' Female']
Occupation ['Exec-managerial' 'Handlers-cleaners' 'Prof-specialty' 'Other-service'
 'Adm-clerical' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
```

#Data Preprocessing(One hot Encoding)

Handling columns

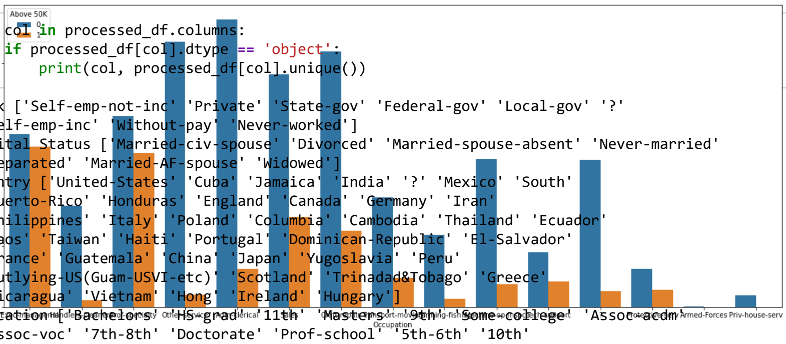
1. Creating copy of eda_df i.e eda data frame into new data frame for data processing

In [102]:
```
# Does Occupation affects the income range
from matplotlib import pyplot as plt
import seaborn as sns
```

In [103]:
```
processed_df=eda_df.copy()
```

```
plt.figure(figsize=(20,8))
sns.countplot(data=eda_df, x='Occupation', hue='Above 50K')
```

Out[102]: <AxesSubplot:xlabel='Occupation', ylabel='count'>



Handling rows

In [104]:
```
for col in processed_df.columns:
    if processed_df[col].dtype == 'object':
        print(col, processed_df[col].unique())
```

```
Work ['Self-emp-not-inc' 'Private' 'State-gov' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
Marital Status ['Married-civ-spouse' 'Divorced' 'Married-spouse-absent' 'Never-married'
 'Separated' 'Married-AF-spouse' 'Widowed']
Country ['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary']
Education ['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
Sex ['Male' 'Female']
Occupation ['Exec-managerial' 'Handlers-cleaners' 'Prof-specialty' 'Other-service'
 'Adm-clerical' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
```

Data Preprocessing (One hot Encoding)

1. Dropping the `Education` column as `EducationNum` column is provided.
2. Adding new category as 'Not Known' for the '?' values in `Work`, `Country`, and `Occupation`

In [105]:
```
# Drop the Education columns

processed_df.drop('Education', axis=1, inplace=True)

# Work, Country, and Occupation column have '?' values. We will introduce the new category for t
df['Work'].replace('?','Not Known', inplace=True)
df['Country'].replace('?','Not Known', inplace=True)
df['Occupation'].replace('?','Not Known', inplace=True)


# Applying One Hot Encoding to the Dataframe
one_hot_df = pd.get_dummies(processed_df, drop_first=True)

one_hot_df.head(5)
```

Out[105]:

	Hours_Week	Above 50K	Age	Work_Federal-gov	Work_Local-gov	Work_Never-worked	Work_Private	Work_Self-emp-inc	Work_Self-emp-not-inc	Work_S
0	13	0	50	0	0	0	0	0	1	
1	40	0	38	0	0	0	1	0	0	
2	40	0	53	0	0	0	1	0	0	
3	40	0	28	0	0	0	1	0	0	
4	40	0	37	0	0	0	1	0	0	

In [106]:
```
# processed_df.to_csv('drive/MyDrive/Inventory Project/processed_df.csv')
```

In [112]:
```
# Variance values for all the principle components
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Feature Selection (One Hot)

Due to one hot, we got large set of columns. Before PCA we need to standardize and scale all the columns using Standard Scaler.

Out[112]:

	0
0	0.037740
1	0.065273
2	0.090879
3	0.112668
4	0.132264
5	0.151083
6	0.168569
7	0.185500
8	0.202042
9	0.218328
10	0.234502

In [107]:
```
from sklearn.preprocessing import StandardScaler

# Using Standard Scaler Class

scaler = StandardScaler()
```

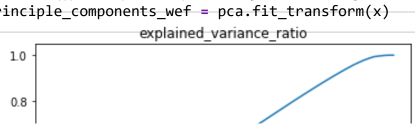
In [108]:
```
# Dropping the target columns as scaling don't require it
scaled_df = one_hot_df.drop('Above 50K', axis=1)
```

In [109]:
```
one_hot_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [110]:
```
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['Above 50K']
x = one_hot_scaled_df
```

In [113]:
```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```

In [111]:
```
from sklearn.decomposition import PCA
pca = PCA()
# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```



Feature Selection (One Hot)

Due to one hot, we got large set of columns. Before PCA we need to standardize and scale all the columns using Standard Scaler.

In [107]:
```python
from sklearn.preprocessing import StandardScaler

# Using Standard Scaler Class

scaler = StandardScaler()
```

In [108]:
```python
# Dropping the target columns as scaling don't require it
scaled_df = one_hot_df.drop('Above 50K', axis=1)
```

In [109]:
```python
one_hot_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [110]:
```python
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['Above 50K']
x = one_hot_scaled_df
```

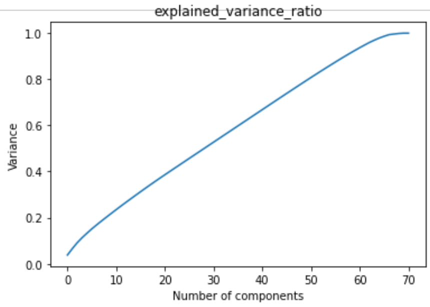
In [111]:
```python
from sklearn.decomposition import PCA

pca = PCA()
# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```

In [112]:
```python
# Variance values for all the principle components
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Out[112]:

	0
0	0.037740
1	0.065273
2	0.090879
3	0.112668
4	0.132264
5	0.151083
6	0.168569
7	0.185500
8	0.202042
9	0.218328
10	0.234502

In [113]:
```python
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```



We can see that around 99% of variance is being explained by 66 components. So instead of giving all columns as input in our algorithm let's use these principle components instead

In [114]:
```python
pca = PCA(n_components=66)
pca_onehot_df = pca.fit_transform(x)

column_name = [f'PC-{i}' for i in range(1, 67)]

principal_onehot_df = pd.DataFrame(pca_onehot_df,columns=column_name)
principal_onehot_df.head()
```

Out[114]:

	PC-1	PC-2	PC-3	PC-4	PC-5	PC-6	PC-7	PC-8	PC-9	PC-10	PC-1
0	2.305275	-0.862720	0.473330	-0.061877	1.888662	-0.025479	0.124982	0.001428	-1.134315	-0.176133	1.24761
1	-0.599388	0.670837	-1.293432	-0.333856	-0.519609	-0.589518	-0.193658	-0.892236	0.153488	0.218329	0.31647
2	0.703507	0.908241	-1.343380	0.301603	-0.773007	-0.768850	-0.164568	-0.743841	0.033581	0.216421	0.21656
3	-0.112300	2.812372	3.201290	1.246660	-0.947925	0.631300	-0.809534	-0.447217	-0.972917	2.486118	-1.99486
4	0.473289	-0.030681	-0.613164	1.221729	-0.146936	1.434805	-0.103488	-0.387173	-0.433728	-0.303914	1.07282

Data Modeling (One Hot)

In [115]:
```python
import sklearn.metrics as sm
from sklearn.model_selection import KFold

# Function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores =[]

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
```

Data Modeling (One Hot)

In [115]:
```
import sklearn.metrics as sm
from sklearn.model_selection import KFold

# Function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores =[]

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))
    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

###Random Forest

In [116]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import lightgbm as lgb
```

In [117]:
```
y=one_hot_df['Above 50K']
X= principal_onehot_df
```

In [118]:
```
columns = ["Algorithm", "Encoding", "Accuracy"]
data = []
```

In [119]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
result= ["Random Forest", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7946666666666666
```

###Decision Tree

In [120]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7671111111111112
```

###XGBoost

In [122]:
```
from xgboost import XGBClassifier
result= ["XGBoost", "OneHot"]
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8029999999999999
```

###Adaboost

In [123]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8112222222222222
```

SVM

In [121]:
```
from sklearn.svm import SVC
model = SVC()
result= ["SVM", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8187777777777777
```

Logistic Regression

In [124]:
```
from sklearn.linear_model import LogisticRegression
```

###XGBoost

In [122]:
```python
from xgboost import XGBClassifier
result= ["XGBoost", "OneHot"]
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8029999999999999
```

###Decision Tree

In [120]:
```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7671111111111112
```

###Adaboost

In [123]:
```python
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8112222222222222
```

SVM

In [121]:
```python
from sklearn.svm import SVC
model = SVC()
result= ["SVC", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8187777777777777
```

Logistic Regression

In [124]:
```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result= ["LogisticRegression", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

########
Results
########

Average accuracy score:  0.8195555555555556
```

KNN

In [125]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "OneHot"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########
Average accuracy score:  0.3347777777777778
```

#Data Preprocessing (WOE Encoding)

Handling columns

1.Creating copy of eda_df i.e eda data frame into new data frame for data processing

In [126]:
```python
preprocessed_df=eda_df.copy()
```

Handling rows

Convert Categorical columns into numeric using Weight Of Evidence encoding

In [127]:
```
! pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\shekhar\anaconda3\lib\site-packag
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
C:\Users\SHEKHAR\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
########
Results
########
Averaged accuracy score: 0.8195555555555556
```

KNN

In [125]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "OneHot"]
result.extend(evaluate_model(model, x, y))
data.append(result)
```

```
########
Results
########
Average accuracy score: 0.3347777777777778
```

#Data Preprocessing (WOE Encoding)

Handling columns

1.Creating copy of eda_df i.e eda data frame into new data frame for data processing

In [126]:
```python
processed_df = eda_df.copy()
```

Handling rows

Convert Categorical columns into numeric using Weight Of Evidence encoding

In [127]:
```python
! pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\shekhar\anaconda3\lib\site-packages (2.5.1.post0)
Requirement already satisfied: numpy>=1.14.0 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from category_encoders) (1.23.1)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.0.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.8.1)
Requirement already satisfied: pandas>=1.0.5 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (1.4.2)
Requirement already satisfied: patsy>=0.5.1 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (0.5.2)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\shekhar\anaconda3\lib\site-packages (from category_encoders) (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\shekhar\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2021.3)
Requirement already satisfied: six in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\shekhar\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shekhar\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\shekhar\appdata\roaming\python\python39\site-packages (from packaging>=21.3->statsmodels>=0.9.0->category_encoders) (3.0.9)
```

In [128]:
```python
# Drop the Education columns

processed_df.drop('Education', axis=1, inplace=True)

# Work, Country, and Occupation column have '?' values. We will introduce the new category for t
df['Work'].replace('?','Not Known', inplace=True)
df['Country'].replace('?','Not Known', inplace=True)
df['Occupation'].replace('?','Not Known', inplace=True)
```

In [132]:
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

In [129]:
```python
# Applying one hot encoding to the Dataframe
from category_encoders.woe import WOEEncoder

WOE_encoder = WOEEncoder()

categorical_cols = ['Work', 'Marital Status', 'Occupation',  'Gender', 'Country']
for column in categorical_cols:
    processed_df[column] = WOE_encoder.fit_transform(processed_df[column], processed_df['Above 5
```

In [133]:
```python
scaled_df = wef_df.drop('Above 50K', axis=1)
```

In [134]:
```python
woe_df_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [135]:
```python
wef_df = processed_df.copy()
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['Above 50K']
x = woe_df_scaled_df
```

In [130]:
```python
wef_df.head()
```

Out[130]:

	Work	Marital Status	Hours_Week	Country	Above 50K	Gender	Occupation	Age
0	0.209936	0.948163	13	0.024828	0	0.313434	1.079632	50
1	-0.120254	-1.037370	40	0.024828	0	0.313434	-1.519335	38
2	-0.120254	0.948163	40	0.024828	0	0.313434	-1.519335	53
3	-0.120254	0.948163	40	-0.635448	0	-0.888444	0.941902	28
4	-0.120254	0.948163	40	0.024828	0	-0.888444	1.079632	37

In [136]:
```python
from sklearn.decomposition import PCA

pca = PCA()

# Get the principle components for WeF dataframe
principle_components_wef = pca.fit_transform(x)
```

In [137]:
```python
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

Out[137]:

	0
0	0.290995
1	0.447477
2	0.596197
3	0.724505
4	0.829041
5	0.930157

In [131]:
```python
#processed_df.to_csv('drive/MyDrive/Inventory Project/woe_processed_df.csv')
```

Feature Selection (WOE)

```
df['Occupation'].replace('?','Not Known', inplace=True)
```

In [132]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

In [129]:
```
# Applying one hot encoding to the Dataframe
from category_encoders.woe import WOEEncoder
WOE_encoder = WOEEncoder()

categorical_cols = ['Work', 'Marital Status', 'Occupation',  'Gender', 'Country']
for column in categorical_cols:
    processed_df[column] = WOE_encoder.fit_transform(processed_df[column], processed_df['Above 5
```

In [133]:
```
scaled_df = wef_df.drop('Above 50K', axis=1)
```

In [134]:
```
woe_df_scaled_df = pd.DataFrame(scaler.fit_transform(scaled_df), columns=scaled_df.columns)
```

In [135]:
```
# Before applying feature selection let's just save the Dependent column
y = one_hot_df['Above 50K']
x = woe_df_scaled_df
```

In [135]:
```
wef_df = processed_df.copy()
```

In [130]:
```
wef_df.head()
```

Out[130]:

	Work	Marital Status	Hours_Week	Country	Above 50K	Gender	Occupation	Age
0	0.209936	0.948163	13	0.024828	0	0.313434	1.079632	50
1	-0.120254	-1.037370	40	0.024828	0	0.313434	-1.519335	38
2	-0.120254	0.948163	40	0.024828	0	0.313434	-1.519335	53
3	-0.120254	0.948163	40	-0.635448	0	-0.888444	0.941902	28
4	-0.120254	0.948163	40	0.024828	0	-0.888444	1.079632	37

In [136]:
```
from sklearn.decomposition import PCA

pca = PCA()

# Get the principle components for wef dataframe
principle_components_wef = pca.fit_transform(x)
```

In [137]:
```
pd.DataFrame(np.cumsum(pca.explained_variance_ratio_))
```

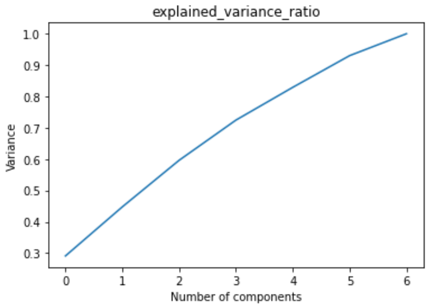
Out[137]:

	0
0	0.290995
1	0.447477
2	0.596197
3	0.724505
4	0.829041
5	0.930157
6	1.000000

In [131]:
```
#processed_df.to_csv('drive/MyDrive/Inventory Project/woe_processed_df.csv')
```

Feature Selection (WOE)

In [138]:
```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Variance')
plt.title('explained_variance_ratio')
plt.show()
```



We can see that around 95% of variance is being explained by 5 components. So instead of giving all columns as input in our algorithm let's use these principle components instead

In [139]:
```
pca = PCA(n_components=5)
pca_woe_df = pca.fit_transform(x)

column_name = [f'PC-{i}' for i in range(1, 6)]

principal_woe_df = pd.DataFrame(pca_woe_df, columns=column_name)
principal_woe_df.head()
```

Out[139]:

	PC-1	PC-2	PC-3	PC-4	PC-5
0	1.022950	0.502717	-1.334604	0.483262	-2.233079
1	-0.614968	0.609169	0.188460	-0.623830	-0.104710
2	0.554503	1.586018	-0.702159	-0.355450	0.091552
3	-0.112656	0.537810	1.162723	1.649826	0.374699
4	0.423505	-0.596555	-0.651692	0.577477	0.448472

##Data Modeling

In [140]:
```
import sklearn.metrics as sm
from sklearn.model_selection import KFold

# Function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores =[]

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))

    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

```
principal_woe_df = pd.DataFrame(pca_woe_df, columns=column_name)
principal_woe_df.head()
```

In [140]:
```
import sklearn.metrics as sm
from sklearn.model_selection import KFold
```

Out[139]:

	PC-1	PC-2	PC-3	PC-4	PC-5
0	1.022950	0.502717	-1.334604	0.483262	-2.233079
1	-0.614968	0.609169	0.188460	-0.623830	-0.104710
2	0.554503	1.586018	-0.702159	-0.355450	0.091552
3	-0.112656	0.537810	1.162723	1.649826	0.374699
4	0.423505	-0.596555	-0.651692	0.577477	0.448472

##Data Modeling

```
# Function to evaluate model
def evaluate_model(model, x, y):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores =[]

    kf = KFold(n_splits=10, random_state=0, shuffle=True)
    for train_index, test_index in kf.split(x):
        #setting up the data
        x_train, x_test = x.values[train_index], x.values[test_index]
        y_train, y_test = y.values[train_index], y.values[test_index]

        #Training model
        model.fit(x_train,y_train)

        #Evaluating model
        y_pred = model.predict(x_test)

        accuracy_scores.append(sm.accuracy_score(y_test,y_pred))
        precision_scores.append(sm.precision_score(y_test,y_pred))
        recall_scores.append(sm.recall_score(y_test,y_pred))
        f1_scores.append(sm.f1_score(y_test, y_pred))


    #displaying average results
    print("########")
    print("Results")
    print("########\n")
    print("Average accuracy score: ", (sum(accuracy_scores)/len(accuracy_scores)))

    results = [(sum(accuracy_scores)/len(accuracy_scores))]
    return results
```

###Random Forest

In [141]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import lightgbm as lgb
```

In [142]:
```
y=wef_df['Above 50K']
X= principal_woe_df
```

In [143]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
result= ["Random Forest", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7963333333333333
```

###Decision Tree

In [144]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
result= ["Decision tree", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

In [146]:
```
from xgboost import XGBClassifier
result= ["XGBoost", "WOE"]
# result.extend(evaluate_model(model, X, y))
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7628888888888888
```

```
########
Results
########

Average accuracy score:  0.7971111111111111
```

SVM

###Adaboost

In [145]:
```
from sklearn.svm import SVC
model = SVC()
result= ["SVM", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

In [147]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7991111111111111
```

```
########
Results
########

Average accuracy score:  0.8152222222222221
```

###XGBoost

Logistic Regression

In [148]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result= ["LogisticRegression", "WOE"]
result.extend(evaluate_model(model, X, y))
```

```
result= ["Decision tree", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7628888888888888
```

###XGBoost

In [146]:
```
from xgboost import XGBClassifier
# result.extend(evaluate_model(model, X, y))
result= ["XGBoost", "WOE"]
model = XGBClassifier()
result.extend(evaluate_model(model,X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7971111111111111
```

SVM

In [145]:
```
from sklearn.svm import SVC
model = SVC()
result= ["SVM", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7991111111111111
```

###Adaboost

In [147]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=0)
result= ["Adaboost", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.8152222222222221
```

Logistic Regression

In [148]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result= ["LogisticRegression", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7936666666666667
```

KNN

In [149]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
result= ["GaussianNB", "WOE"]
result.extend(evaluate_model(model, X, y))
data.append(result)
```

```
########
Results
########

Average accuracy score:  0.7947777777777778
```

Saving Results for Income Prediction

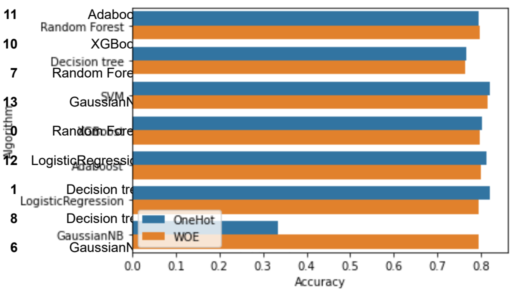
In [150]:
```
results_df = pd.DataFrame(data,columns=columns)
results_df = results_df[['Algorithm', 'Encoding', 'Accuracy']]
results_df.sort_values(by=['Accuracy', 'Encoding'], ascending=False)
```

Out[150]:

	Algorithm	Encoding	Accuracy
5	LogisticRegression	OneHot	0.819556
2	SVM	OneHot	0.818778
9	SVM	WOE	0.815222
4	Adaboost	OneHot	0.811222
3	XGBoost	OneHot	0.803000
11	Adabo		
10	XGBo		
7	Random Fore		
13	GaussianN		
0	Random Fore		
12	LogisticRegressi		
1	Decision tr		
8	Decision tr		
6	GaussianN		

In [151]:
```
sns.barplot(data=results_df, x='Accuracy', y='Algorithm', hue='Encoding')
plt.legend(loc='lower left')
```

Out[151]: <matplotlib.legend.Legend at 0x15e5742b0a0>

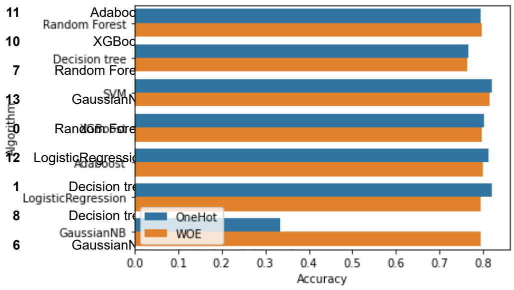


In [152]:
```
results_df.to_csv("income_prediction_results.csv")
```

2	SVM	OneHot	0.818778
9	SVM	WOE	0.815222
4	Adaboost	OneHot	0.811222
3	XGBoost	OneHot	0.803000
11	Adaboo		
10	XGBoo		
7	Random Fore		
13	GaussianN		
0	Random Fore		
12	LogisticRegressio		
1	Decision tr		
8	Decision tr		
6	GaussianN		

In [151]:
```
sns.barplot(data=results_df, x='Accuracy', y='Algorithm', hue='Encoding')
plt.legend(loc='lower left')
```

Out[151]: <matplotlib.legend.Legend at 0x15e5742b0a0>



In [152]:
```
results_df.to_csv("income_prediction_results.csv")
```