Insurance Claims- Fraud Detection

Insurance companies make money by betting on risk - the risk that insured won't die before his/her time and make the insurer payout, or the risk insured house won't burn down or insured SUV won't be totalled in a crash. The first task of any insurer is to price risk and charge a premium for assuming it. When a customer files a claim, the company must process it, check it for accuracy, and submit payment. This adjusting process is necessary to filter out fraudulent claims and minimize the risk of loss to the company. Insurance fraud costs the auto insurers and consumers billions of dollars each year, which forces insurance premium prices to go up annually. Fraud may be committed at different points in the transaction by applicants, policy holders, third-party claimants, or professionals who provide services to claimants. Whether the matter involves a consumer misrepresenting a claim or a retailer or service provider misreporting products or labour, insurance fraud is considered a crime. Investigating fraudulent claims is costly and time consuming for insurers. It is physically impossible for insurance companies to do a thorough check of the thousands of claims that enter their systems daily.
Common frauds include ‘padding’, (which means to add damages, injuries and fictitious passengers to insurance claims) or inflating claims; misrepresenting facts on an insurance application; submitting claims for injuries or damage that never occurred; and staging accidents. Some lines of insurance are more vulnerable to fraud than others. Healthcare, workers compensation, and auto insurance are generally considered to be the sectors most affected. Any effort aiming to debug the fraudulent transactions in the above-mentioned businesses and probably in many other ones is named as a fraud detection process. In terms of insurance fraud detection, machine learning applies aspects of AI to give systems the ability to improve from experience with no extra programming by analyzing large, labelled data sets.
Machine learning can improve fraud detection techniques in the following ways: • Data is processed in a short period of time.
• The system highlights where connections can exist between various factors that human eyes cannot detect. • The application of variations in data analysis will allow the discovery of new fraud schemes.

Although it borrows underlying principals found in statistical models, the main focus of machine learning is producing predictions. These predictions are based on the analysis of known outcomes, known as “ground truth.”

Problem Statement:

I will be creating a predictive model that predicts if an insurance claim is fraudulent or not. The answere between YES/NO, is a Binary Classification task. A comparison study has been performed to understand which ML algorithm suits best to the dataset.

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import sklearn.metrics
from pylab import rcParams
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
```

In [3]:
```
insr_data = pd.read_csv("https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv")
```

In [4]:
```
insr_data.shape
```

Out[4]:
```
(1000, 40)
```

In [4]:
```
pd.set_option('display.max_columns', None)
```

In [ ]:
```
insr_data.head(10)
```

In [5]:
```
insr_data['_c39'].value_counts()
```

Out[5]:
```
Series([], Name: _c39, dtype: int64)
```

In [6]:
```
insr_data['_c39'].unique()
```

Out[6]:
```
array([nan])
```

In [7]:
```
# It is clear from above that '_c39','policy_number', 'incident_location' are not going to affect our study; so dropping it:

auto_data = insr_data.drop(['_c39','policy_number', 'incident_location'], axis=1)
```

In [8]:
```
auto_data.columns
```

Out[8]:
```
Index(['months_as_customer', 'age', 'policy_bind_date', 'policy_state',
       'policy_csl', 'policy_deductable', 'policy_annual_premium',
       'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'capital-gains', 'capital-loss',
       'incident_date', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_state', 'incident_city',
       '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_model',
       'auto_year', 'fraud_reported'],
      dtype='object')
```

In [9]:
```
auto_data['fraud_reported'].value_counts()
```

Out[9]:
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

Target class is very imbalanced. We should try to avoid as many false positives as we can.

In [10]:
```
# As columns in the dataset are randomly arranged , rearranging the columns of the dataset:

data_1 = auto_data[['policy_bind_date','policy_state','policy_csl','policy_deductable','policy_annual_premium','umbrella_limit','insured_zip','months_as_custo
```

In [ ]:
```
data_1.head(15)
```

In [11]:
```
data_1['policy_csl'].value_counts()
```

Out[11]:
```
250/500     351
100/300     349
500/1000    300
Name: policy_csl, dtype: int64
```

Combined single limit (CSL) CSL is a single number that describes the predetermined limit for the combined total of the Bodily Injury Liability coverage and Property Damage Liability coverage per occurrence or accident.

As policy_csl is given so its not necessary to keep the injury_claim, property_claim and vehicle_claim columns seperately. Total_claim is also given.

In [12]:
```
data_2 = data_1.drop(['injury_claim','property_claim', 'vehicle_claim'], axis=1)
```

In [13]:
```
data_3 = data_2.drop(['insured_zip'], axis=1)
```

In [14]:
```
data_3['policy_state'].unique()
```

Out[14]:
```
array(['OH', 'IN', 'IL'], dtype=object)
```

In [15]:
```
data_3['policy_deductable'].unique()
```

Out[15]:
```
array([1000, 2000,  500], dtype=int64)
```

In [ ]:
```
data_3['umbrella_limit'].unique()
```

In [16]:
```
#As umbrella limit cannot be negative so removing '-' :

data_3['umbrella_limit'].replace(to_replace = -1000000, value = 1000000, inplace=True)
```

In [ ]:
```
data_3['umbrella_limit'].unique()
```

In [17]:
```
data_3['collision_type'].unique()
```

Out[17]:
```
array(['Side Collision', '?', 'Rear Collision', 'Front Collision'],
      dtype=object)
```

In [18]:
```
data_3['collision_type'].replace(to_replace='?', value = 'NA', inplace=True)
```

In [ ]:
```
data_3['collision_type'].value_counts()
```

In [19]:
```
data_3['property_damage'].unique()
```

Out[19]:
```
array(['YES', '?', 'NO'], dtype=object)
```

In [20]:
```
data_3['property_damage'].replace(to_replace='?', value = 'NA', inplace=True)
```

In [ ]:
```
data_3['property_damage'].value_counts()
```

In [21]:
```
data_3['police_report_available'].unique()
```

Out[21]:
```
array(['YES', '?', 'NO'], dtype=object)
```

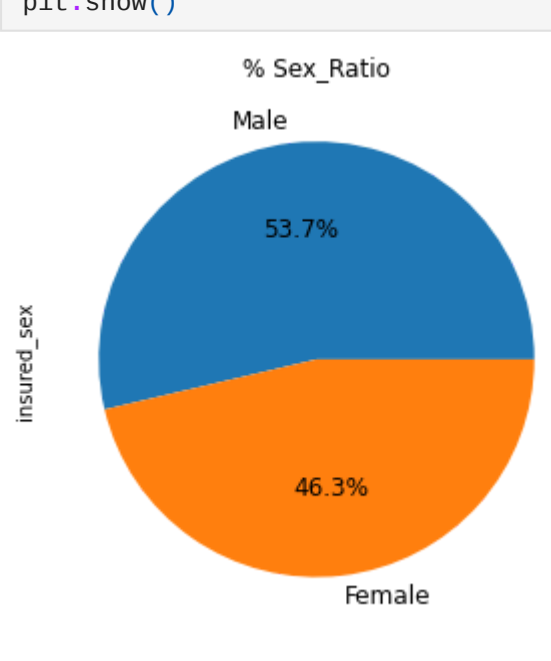
In [22]:
```
data_3['police_report_available'].replace(to_replace='?', value = 'NA', inplace=True)
```

In [ ]:
```
data_3['police_report_available'].value_counts()
```

Data Visualization

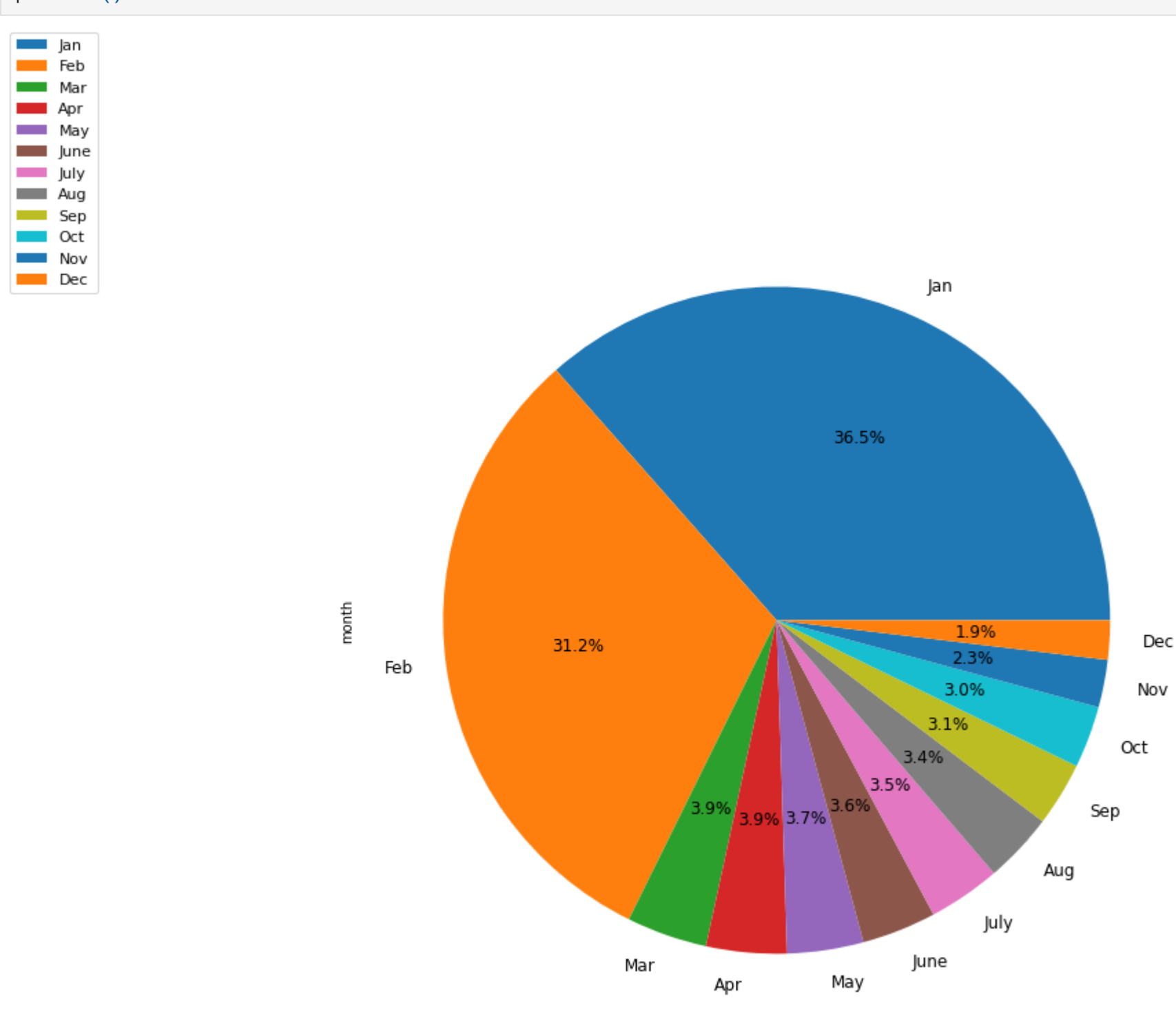
In [23]:
```
import matplotlib.pyplot as plt
```

In [24]:
```
fig = plt.figure(figsize=(7,5))
ax = (data_3['insured_sex'].value_counts()*100.0 /len(data_3))\
.plot.pie(autopct='%.1f%%', labels = ['Male', 'Female'], fontsize=12)
ax.set_title('% Sex_Ratio')
plt.show()
```



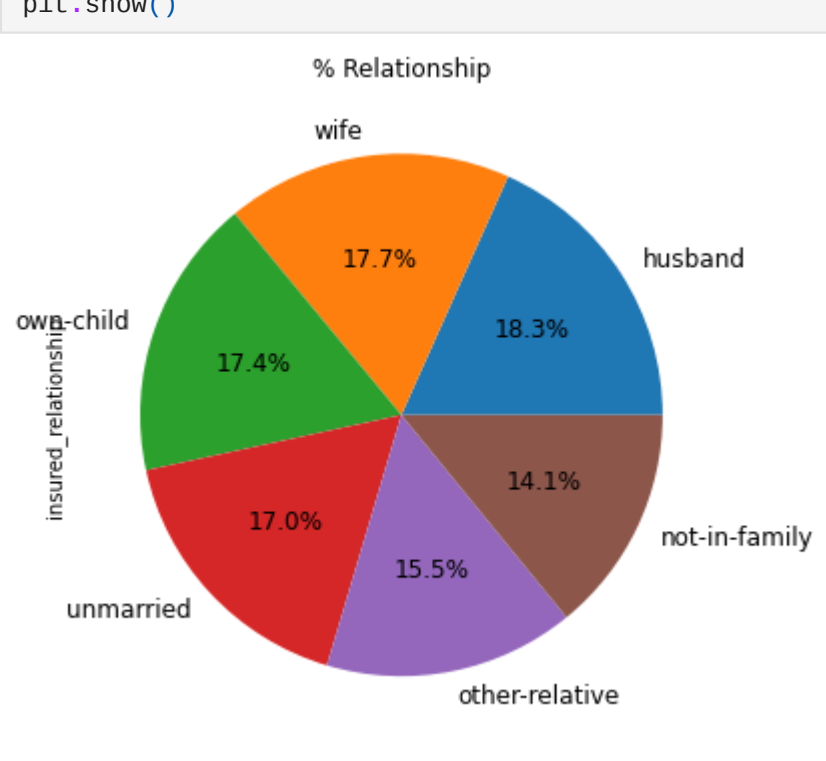
In [25]:
```
# trying to figure out that is there any affect of the month on incidents:
# extracting month from incident_date

data_3['month'] = pd.DatetimeIndex(data_3['incident_date']).month
```

In [26]:
```
sizes = [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]


fig1, ax1 = plt.subplots(figsize=(12, 12))
fig1.subplots_adjust(0.3, 0, 1, 1)

theme = plt.get_cmap('copper')
ax1.set_prop_cycle("color", [theme(1. * i / len(sizes))
                             for i in range(len(sizes))])

_, _ = ax.pie(sizes, startangle=90, radius=1800)
ax = (data_3['month'].value_counts()*100.0 /len(data_3))\
.plot.pie(autopct='%.1f%%', labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'July', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'], fontsize=12)


plt.legend(
    loc='upper left',
    prop={'size': 11},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure)

plt.show()
```

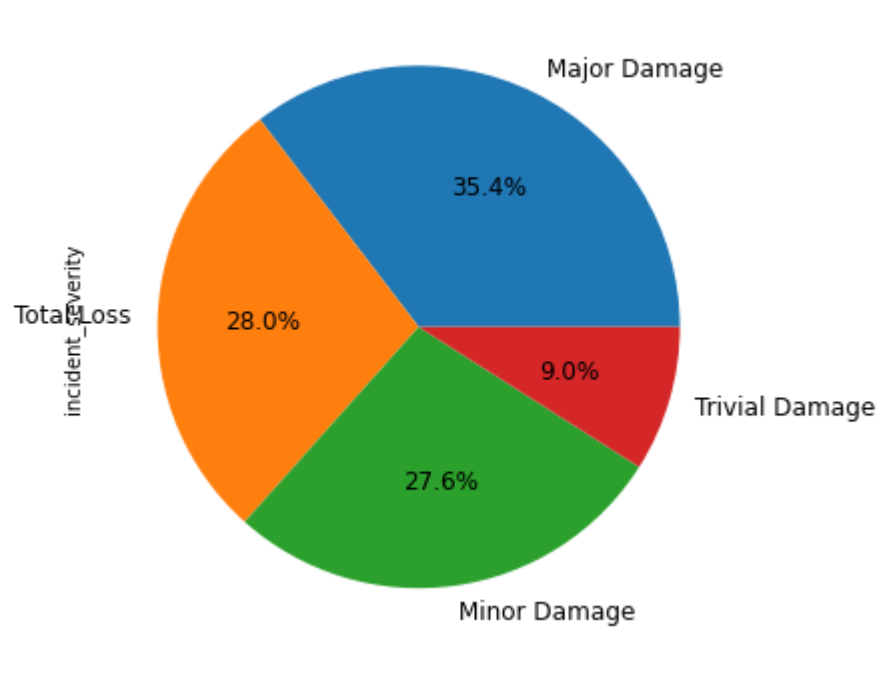


From this chart we can observe that the highest number of incidents are in the month of January followed by Feb.

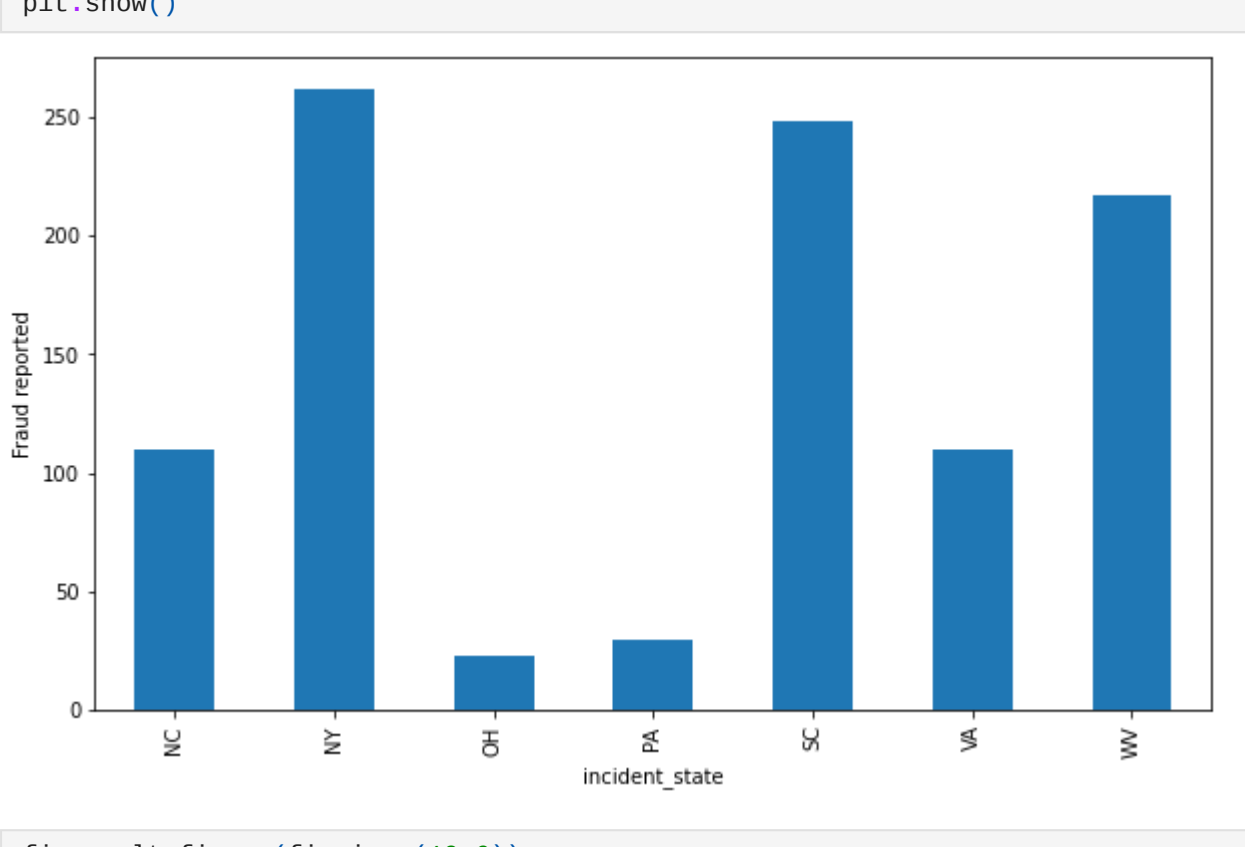
In [27]:
```
fig = plt.figure(figsize=(10,6))
ax = (data_3['insured_relationship'].value_counts()*100.0 /len(data_3))\
.plot.pie(autopct='%.1f%%', labels = ['husband', 'wife', 'own-child', 'unmarried', 'other-relative', 'not-in-family'],
         fontsize=12)
ax.set_title('% Relationship')
plt.show()
```



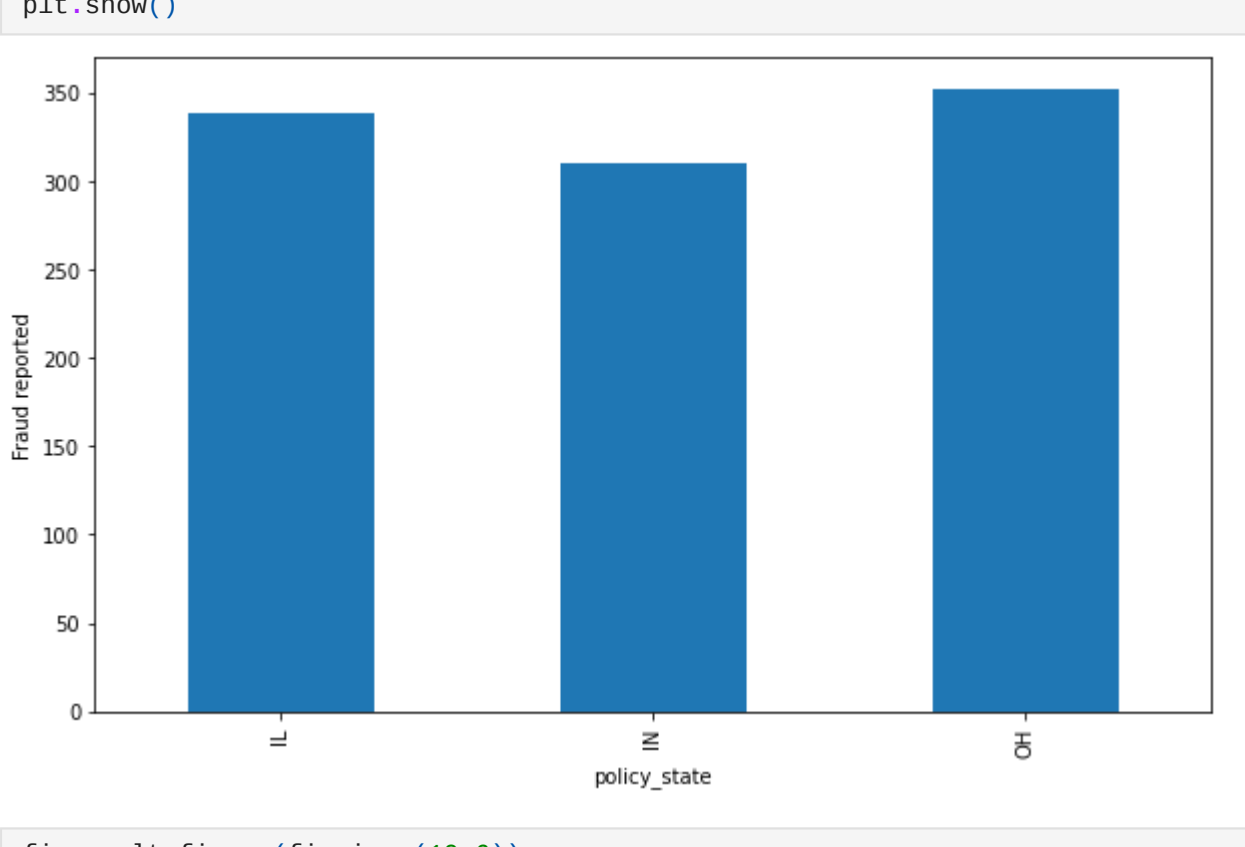
In [28]:
```
fig = plt.figure(figsize=(10,6))
ax = (data_3['incident_severity'].value_counts()*100.0 /len(data_3))\
.plot.pie(autopct='%.1f%%', labels = ['Major Damage', 'Total Loss', 'Minor Damage', 'Trivial Damage'],
         fontsize=12)
```



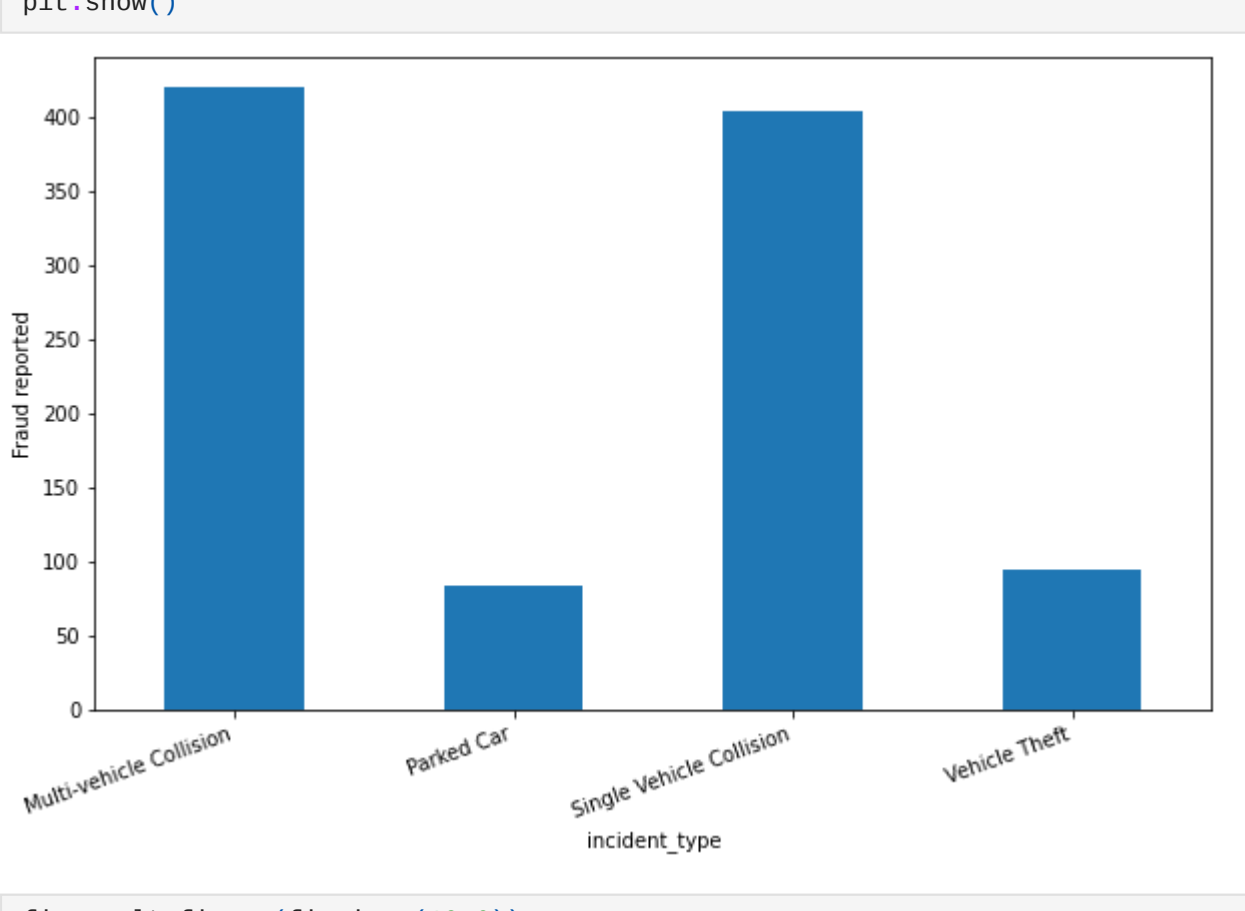
In [29]:
```
fig = plt.figure(figsize=(10,6))
ax = data_3.groupby('incident_state').fraud_reported.count().plot.bar(ylim=0)
ax.set_ylabel('Fraud reported')
plt.show()
```



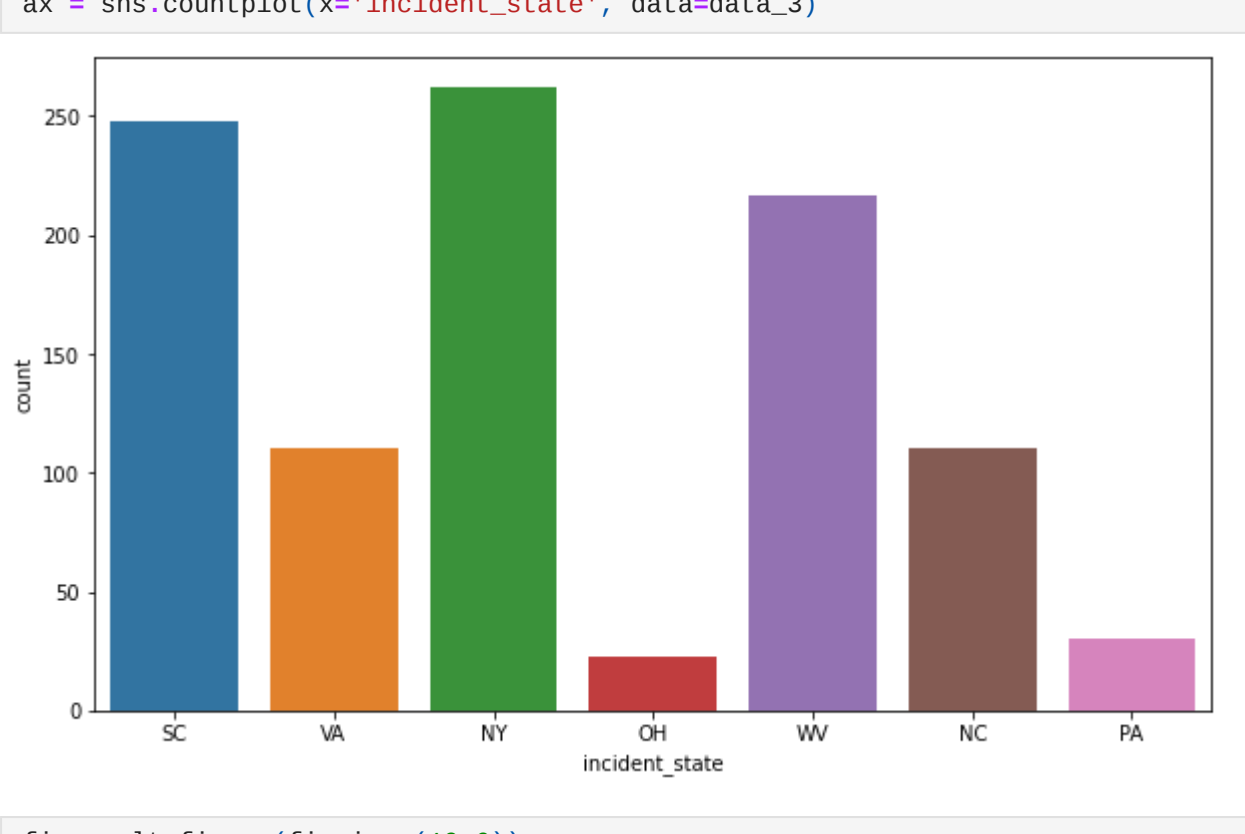
In [30]:
```
fig = plt.figure(figsize=(10,6))
ax = data_3.groupby('policy_state').fraud_reported.count().plot.bar(ylim=0)
ax.set_ylabel('Fraud reported')
plt.show()
```



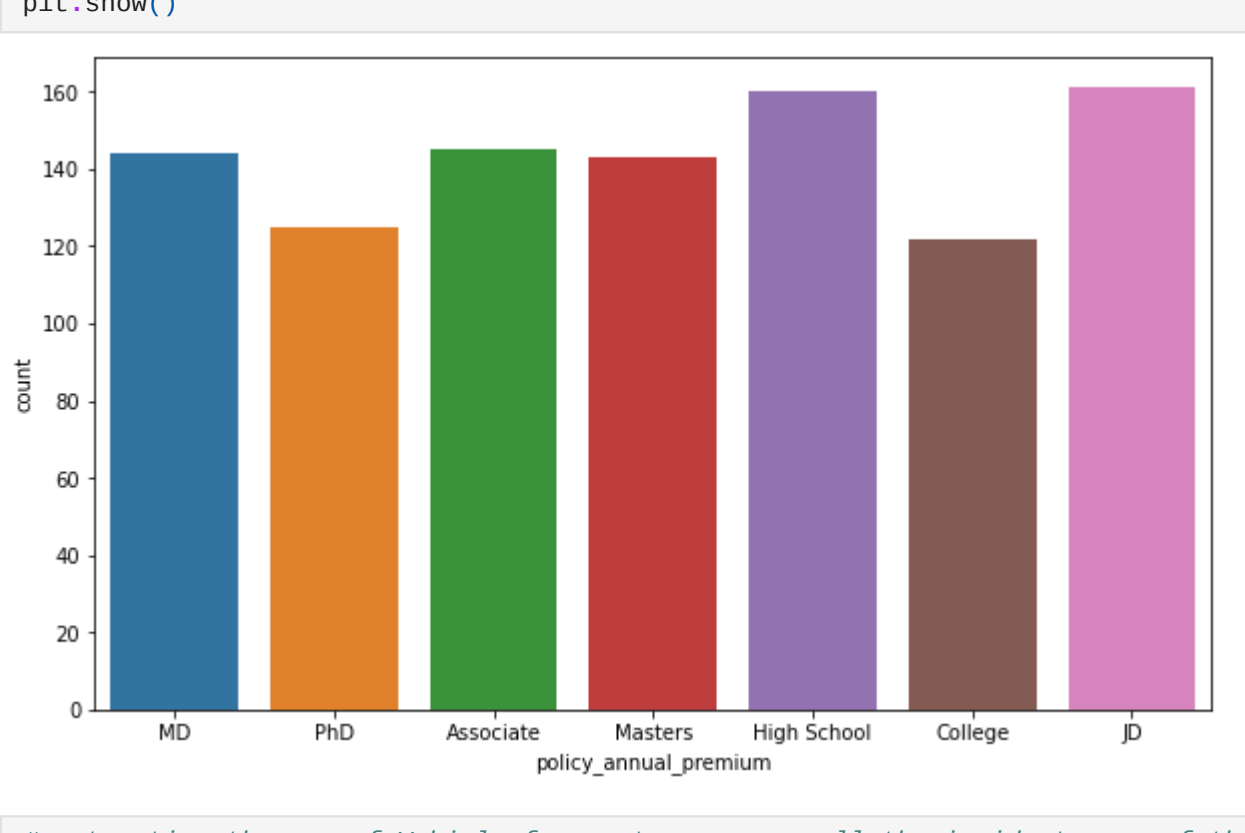
In [37]:
```
fig = plt.figure(figsize=(10,6))
ax = data_3.groupby('incident_type').fraud_reported.count().plot.bar(ylim=0)
ax.set_xticklabels(ax.get_xticklabels(), rotation=20, ha="right")
ax.set_ylabel('Fraud reported')
plt.show()
```



In [31]:
```
fig = plt.figure(figsize=(10,6))
ax = sns.countplot(x='incident_state', data=data_3)
```



In [32]:
```
fig = plt.figure(figsize=(10,6))
ax = sns.countplot(x = 'insured_education_level', data=data_3)
ax.set_xlabel('policy_annual_premium')
plt.show()
```



In [33]:
```
# extracting the age of Vehicle from auto_year, as all the incidents are of the year 2015 :

data_3['Vehicle_Age'] = 2015 - data_3['auto_year']
```

=> Policy bind date and incident date can give the difference of time between policy taken and occurence of incident. Right now I am leaving it due to shartage of time.

In [34]:
```
# Dealing with time it can tell us that at what time of the day major of the incidents occur:

bins = [-1, 3, 6, 9, 12, 17, 20, 24]  # Factorize according to the time period of the day.

names = ["midnight", "early_morning", "late_morning", "afternoon", "evening", "night", "late_night"]

data_3['incident_part_of_day'] = pd.cut(data_3.incident_hour_of_the_day, bins, labels=names).astype(object)

data_3[['incident_hour_of_the_day', 'incident_part_of_day']].head(10)
```

Out[34]:

	incident_hour_of_the_day	incident_part_of_day
0	5	early_morning
1	8	late_morning
2	7	late_morning
3	5	early_morning
4	20	night
5	19	night
6	0	midnight
7	23	late_night
8	21	late_night
9	14	evening

In [35]:
```
# As property damage, police report available are having values NA (not available) so they cannot be converted here:
# converting fraud_reported and insured_sex into binary:

data_3['fraud_reported'].replace(to_replace ='Y', value=1, inplace=True)
data_3['fraud_reported'].replace(to_replace ='N',  value=0, inplace=True)
data_3['insured_sex'].replace(to_replace ='FEMALE', value=1, inplace=True)
data_3['insured_sex'].replace(to_replace ='MALE',  value=0, inplace=True)
# importing one hot encoder from sklearn
# There are changes in OneHotEncoder class

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

In [36]:
```
data_4 = data_3.drop(columns = ['month','auto_year','incident_date','policy_bind_date'])
```

In [37]:
```
cat_features = data_4.select_dtypes(include='O').keys()
```

In [38]:
```
cat_features
```

Out[38]:
```
Index(['policy_state', 'policy_csl', 'insured_education_level',
       'insured_occupation', 'insured_hobbies', 'insured_relationship',
       'incident_state', 'incident_city', 'incident_type', 'incident_severity',
       'collision_type', 'authorities_contacted', 'property_damage',
       'police_report_available', 'auto_make', 'auto_model',
       'incident_part_of_day'],
      dtype='object')
```

In [39]:
```
cat_features = ['policy_state', 'policy_csl', 'insured_education_level',
       'insured_occupation', 'insured_hobbies', 'insured_relationship',
       'incident_state', 'incident_city', 'incident_type', 'incident_severity',
       'collision_type', 'authorities_contacted', 'property_damage',
       'police_report_available', 'auto_make', 'auto_model',
       'incident_part_of_day']

from sklearn.preprocessing import LabelEncoder

# instantiate labelencoder object
le = LabelEncoder()

# apply le on categorical feature columns
data_4[cat_features] = data_4[cat_features].apply(lambda col: le.fit_transform(col))
```

In [ ]:
```
data_4
```

In [40]:
```
data_f = pd.get_dummies(data_4, columns = cat_features)
```

In [ ]:
```
data_f.head()
```

In [41]:
```
data_f.shape
```

Out[41]:
```
(1000, 165)
```

In [42]:
```
# Finding correlation between different variables:

corr_matrix = data_4.corr()
```

In [ ]:
```
X = data_f.drop(columns = ['collision_type'])
X.head(2)
```

In [43]:
```
# seperating the target variable and predictor data:

X = data_f.drop(columns = ['fraud_reported'])
```

In [44]:
```
y = data_f['fraud_reported']
```

In [45]:
```
# splitting dataset into training and testing part:

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=1234)

print('length of X_train and X_test: ', len(X_train), len(X_test))
print('length of y_train and y_test: ', len(y_train), len(y_test))
```

```
length of X_train and X_test:  800 200
length of y_train and y_test:  800 200
```

In [46]:
```
# checking the quality of data:

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=5, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

```
0.841
```

It means our data is now ready for further analysis.

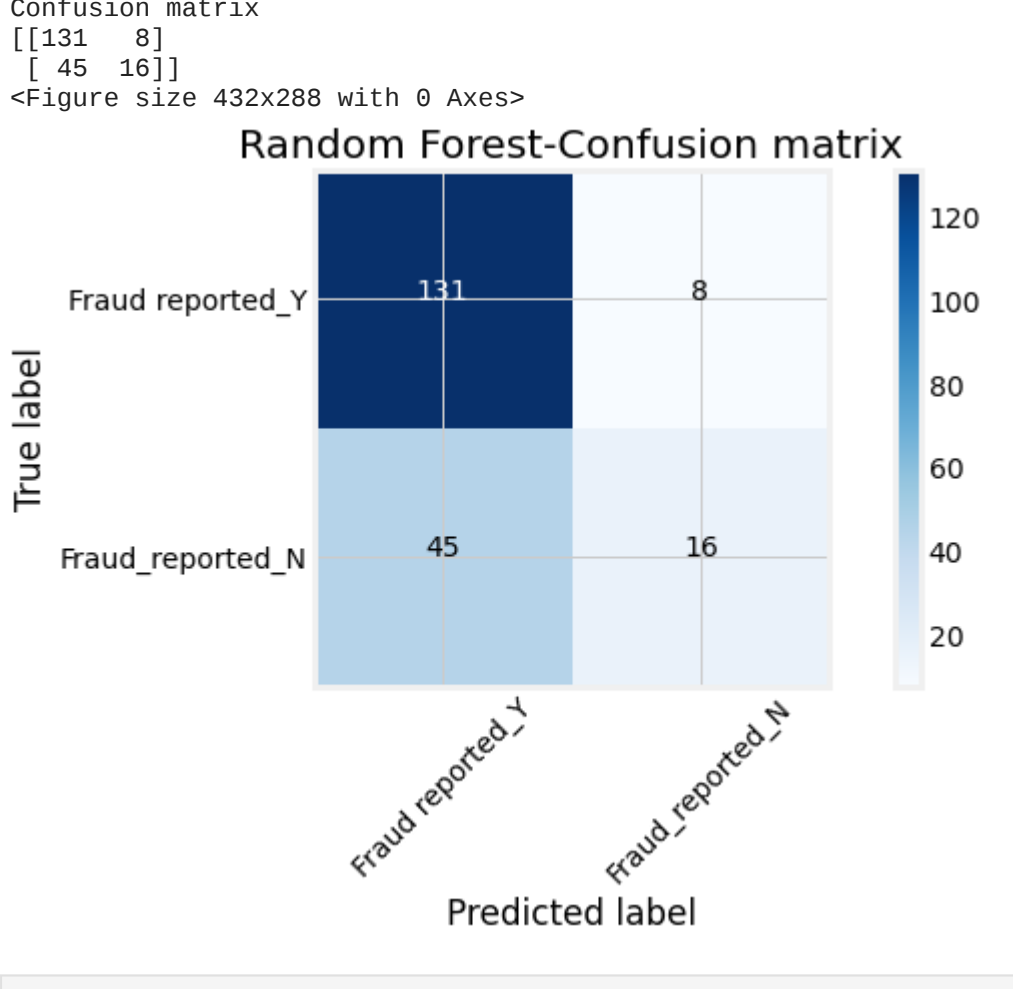
MODEL BUILDING

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, classification_report, cohen_kappa_score
from sklearn import metrics

# Baseline Random forest based Model
rfc = RandomForestClassifier(criterion = 'gini', n_estimators=1000, verbose=1, n_jobs = -1,
                             class_weight = 'balanced', max_features = 'auto')

# training the dataset:

model_rfc = rfc.fit(X_train,y_train)
predictions = model_rfc.predict(X_test)

print('Baseline: N_features: ', len(list(X.columns)))
print('Baseline: Accuracy: ', round(accuracy_score(y_test, predictions)*100, 2))
print( 'Cohen Kappa: '+ str(np.round(cohen_kappa_score(y_test, predictions),3)))
print('Baseline: Recall: ', round(recall_score(y_test, predictions)*100, 2))
print('\n Classification Report:\n', classification_report(y_test,predictions))
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    4.8s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.6s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.7s finished
Baseline: N_features:  164
Baseline: Accuracy:  73.5
Cohen Kappa: 0.247
Baseline: Recall:  26.23

 Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.94      0.83       139
           1       0.67      0.26      0.38        61

    accuracy                           0.73       200
   macro avg       0.71      0.60      0.60       200
weighted avg       0.72      0.73      0.69       200
```

In [48]:
```
from sklearn.metrics import confusion_matrix
import itertools

#Evaluation of RFModel - Confusion Matrix Plot

def plot_confusion_matrix(cm, classes, title ='Confusion matrix', normalize=False, cmap = plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    print('Confusion matrix')

    print(cm)

    fig = plt.figure(figsize=(10,6))
    plt.style.use('fivethirtyeight')
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix

cnf_matrix = confusion_matrix(y_test, predictions)
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Fraud reported_Y','Fraud_reported_N'],
                      title='Random Forest-Confusion matrix')
```

```
Confusion matrix
[[131   8]
 [ 45  16]]
<Figure size 432x288 with 0 Axes>
```

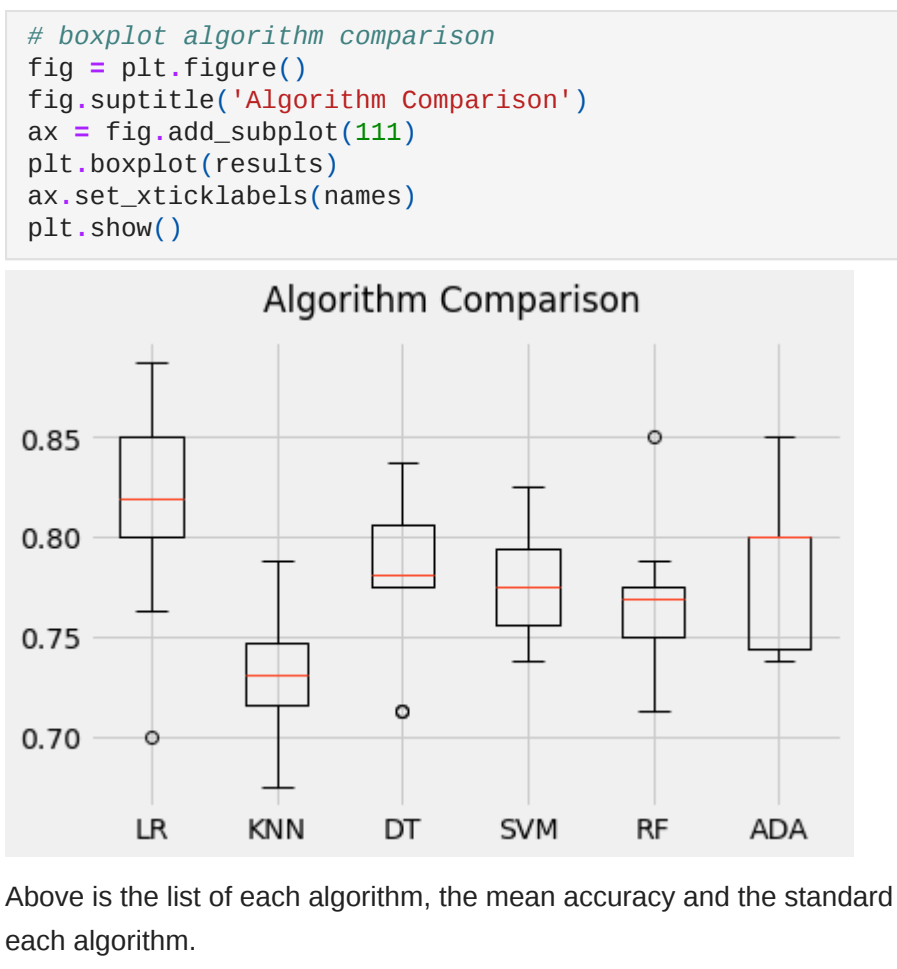


In [49]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [50]:
```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier


logreg2= LogisticRegressionCV(solver='lbfgs', cv=10)
knn = KNeighborsClassifier(5)
svcl = SVC()
adb = AdaBoostClassifier()
dtclf = DecisionTreeClassifier(max_depth=5)
rfclf = RandomForestClassifier()

# prepare configuration for cross validation test harness
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('SVM', SVC(gamma='auto')))
models.append(('RF', RandomForestClassifier(n_estimators=100)))
models.append(('ADA', AdaBoostClassifier(n_estimators=100)))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train_scaled, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.816250 (0.052752)
KNN: 0.732500 (0.029155)
DT: 0.778750 (0.038345)
SVM: 0.776250 (0.025280)
RF: 0.767500 (0.034551)
ADA: 0.786250 (0.038911)
```

In [51]:
```
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Above is the list of each algorithm, the mean accuracy and the standard deviation accuracy and a box & whisker plot showing the spread of the accuracy scores across each cross validation fold for each algorithm.

From above we can conclude that LogisticRegression and DecisionTreeClassifier are having the higest accuracy score.

Conclusion: We can use LogisticRegression Model to detect the fraud on insurance claims.