Insurance Claims- Fraud Detection

Problem Statement:

Money has now become the most essential commodity along with food and shelter. It plays important role in every aspect may it be as industry, personal growth.

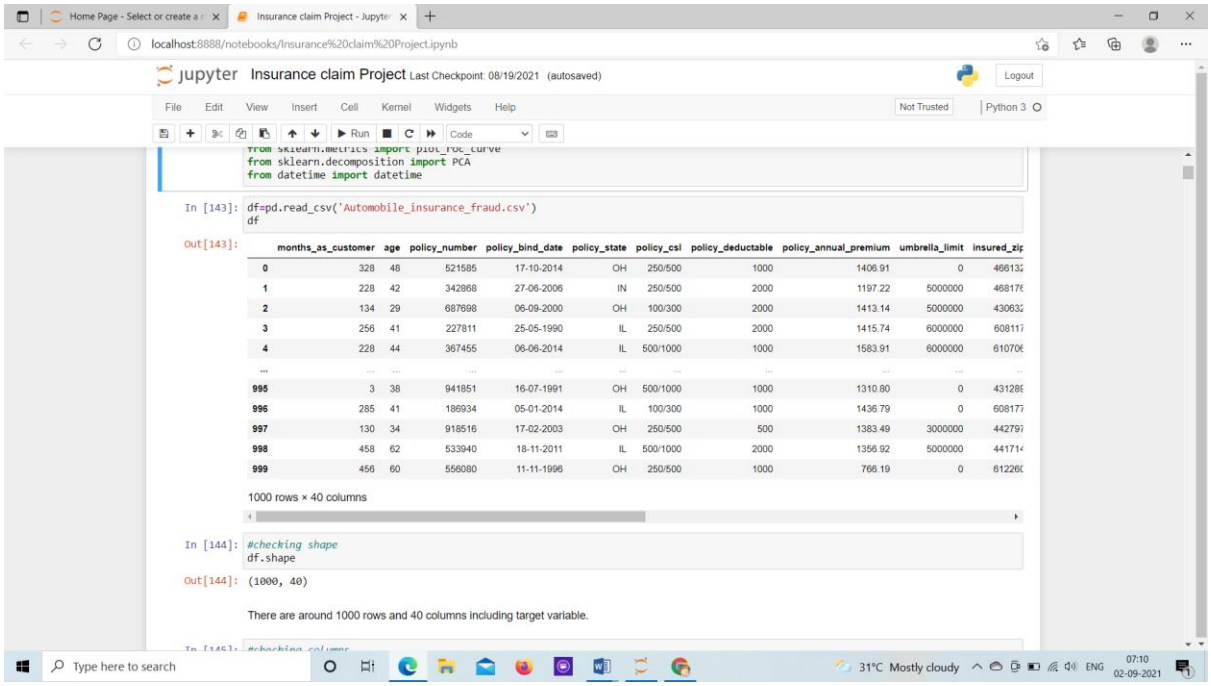
Insurance is also plays an important role in protection from financial loss. It is a form of risk management, primarily used to hedge against the risk of a contingent or uncertain loss.

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, you are provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

In this example, you will be working with some auto insurance data to demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

Data Analysis



EDA

Home Page - Select or create a n | Insurance claim Project - Jupyter

localhost:8888/notebooks/Insurance%20claim%20Project.ipynb

jupyter Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)

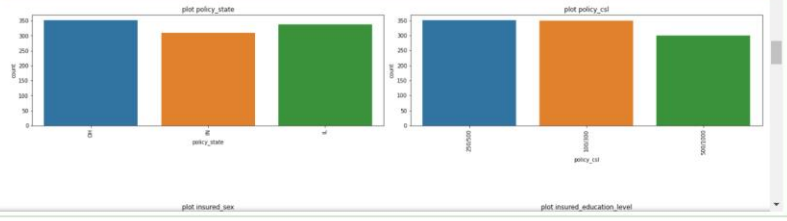
File Edit View Insert Cell Kernel Widgets Help

```
property_damage NO     698
YES    302
Name: property_damage, dtype: int64
```

Categorical columns Graphical Representation

In [169]:
```
ncol,nrow=10,2
ab=df_cat.columns.values
plt.figure(figsize=(20,50))
for index,i in enumerate(ab):
    ab=plt.subplot(ncol,nrow,index+1)
    sns.countplot(df_cat[i])
    plt.title(f"plot {i}")
    plt.xticks(rotation=90)
    plt.tight_layout()
plt.show()
```

```
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Type here to search | 31°C Mostly cloudy | ENG | 07:12 02-09-2021

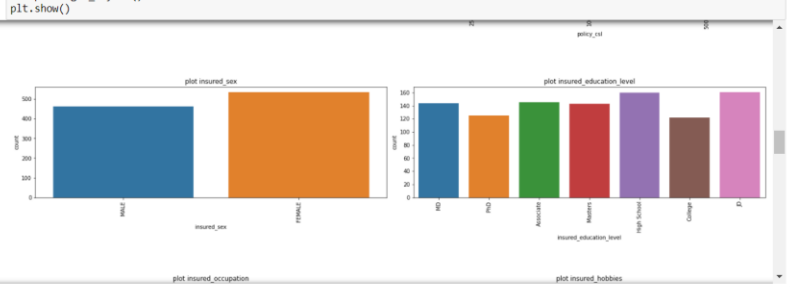
Home Page - Select or create a n | Insurance claim Project - Jupyter

localhost:8888/notebooks/Insurance%20claim%20Project.ipynb

jupyter Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
    sns.countplot(df_cat[i])
    plt.title(f"plot {i}")
    plt.xticks(rotation=90)
    plt.tight_layout()
plt.show()
```



Numerical Columns

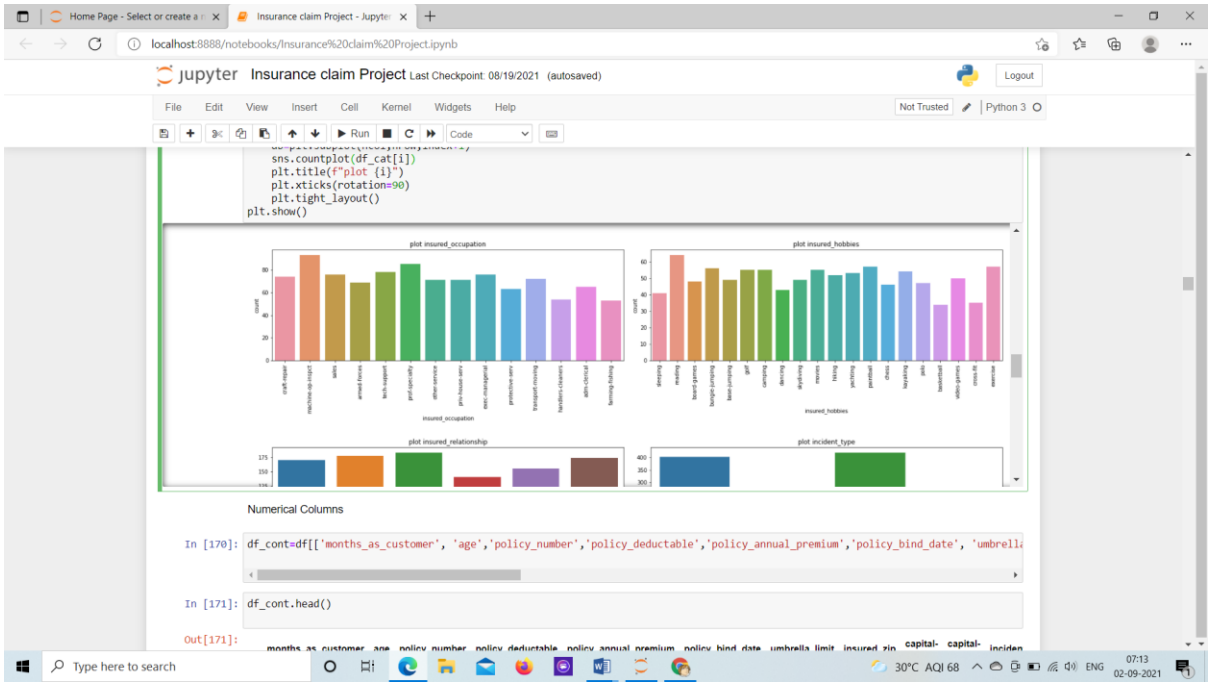
In [170]:
```
df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrella
```

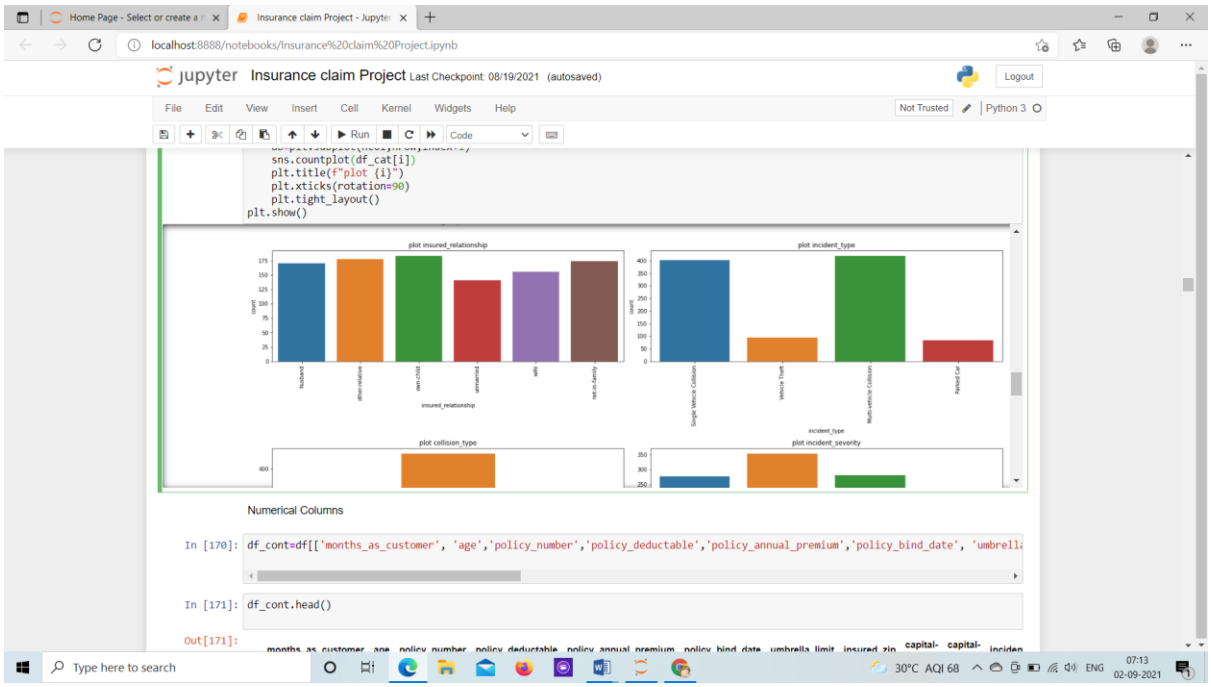
In [171]:
```
df_cont.head()
```

Out[171]:

months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	policy_bind_date	umbrella_limit	insured_zip	capital-	capital-	inciden

Type here to search | 30°C AQI 68 | ENG | 07:12 02-09-2021


Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)
sns.countplot(df_cat[i])
plt.title(f"plot {i}")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
plot insured_occupation
plot insured_hobbies
plot insured_relationship
plot incident_type
Numerical Columns
In [170]: df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrell
In [171]: df_cont.head()
Out[171]:


Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)
sns.countplot(df_cat[i])
plt.title(f"plot {i}")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
plot insured_relationship
plot incident_type
plot collision_type
plot incident_severity
Numerical Columns
In [170]: df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrell
In [171]: df_cont.head()
Out[171]:

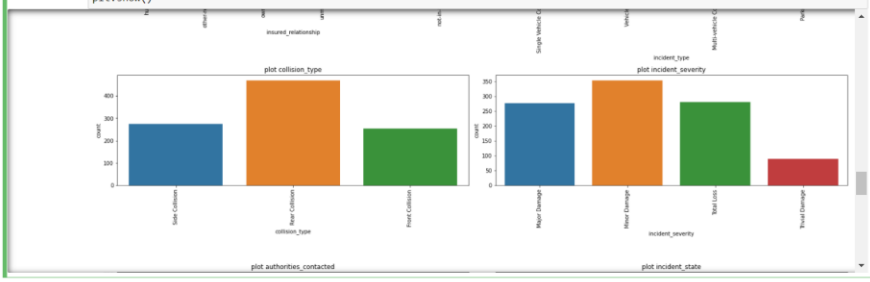
Home Page - Select or create a n | Insurance claim Project - Jupyter

localhost:8888/notebooks/Insurance%20claim%20Project.ipynb

jupyter Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
    sns.countplot(df_cat[i])
    plt.title(f"plot {i}")
    plt.xticks(rotation=90)
    plt.tight_layout()
plt.show()
```



Numerical Columns

In [170]:
```
df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrella
```

In [171]:
```
df_cont.head()
```

Out[171]:

Type here to search 30°C AQI 68 ENG 07:13 02-09-2021

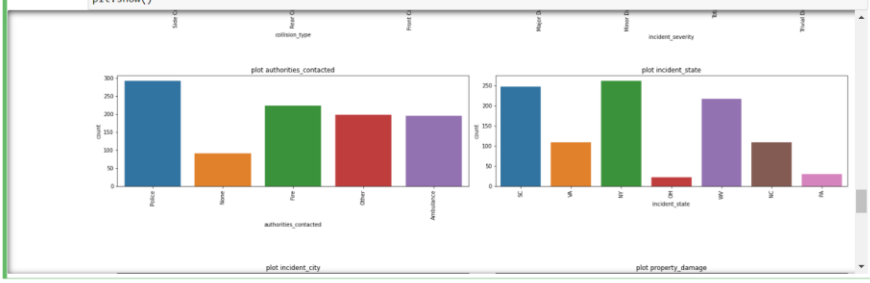
Home Page - Select or create a n | Insurance claim Project - Jupyter

localhost:8888/notebooks/Insurance%20claim%20Project.ipynb

jupyter Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
    sns.countplot(df_cat[i])
    plt.title(f"plot {i}")
    plt.xticks(rotation=90)
    plt.tight_layout()
plt.show()
```



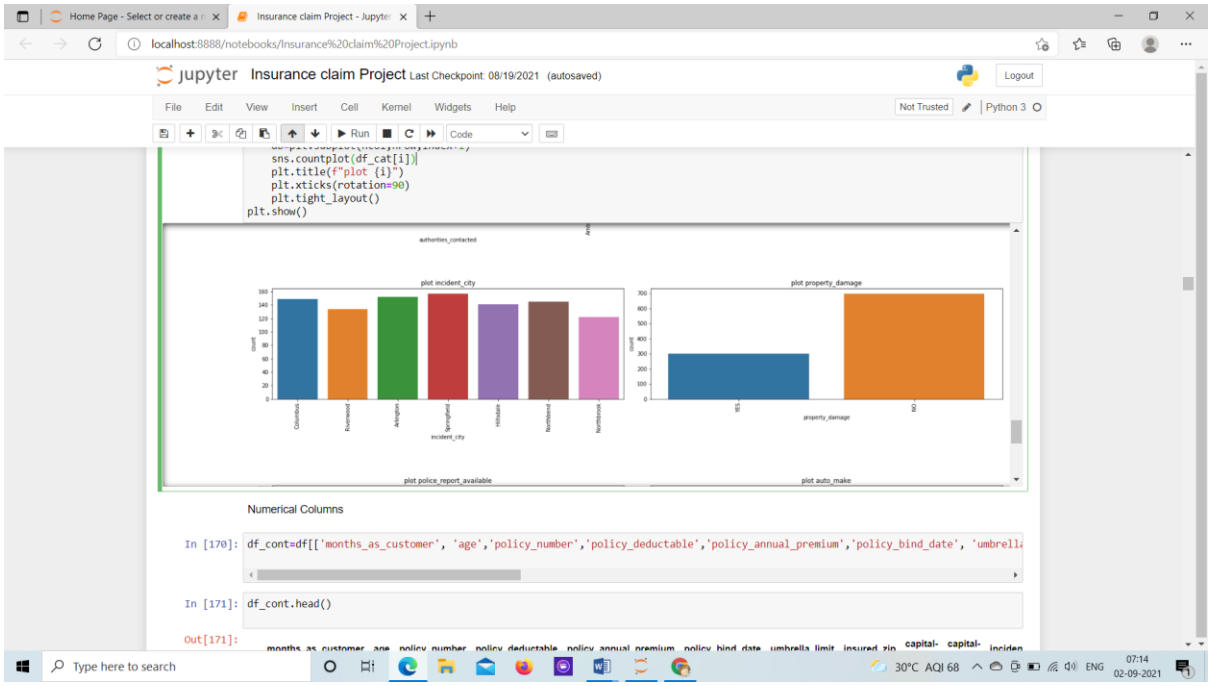
Numerical Columns

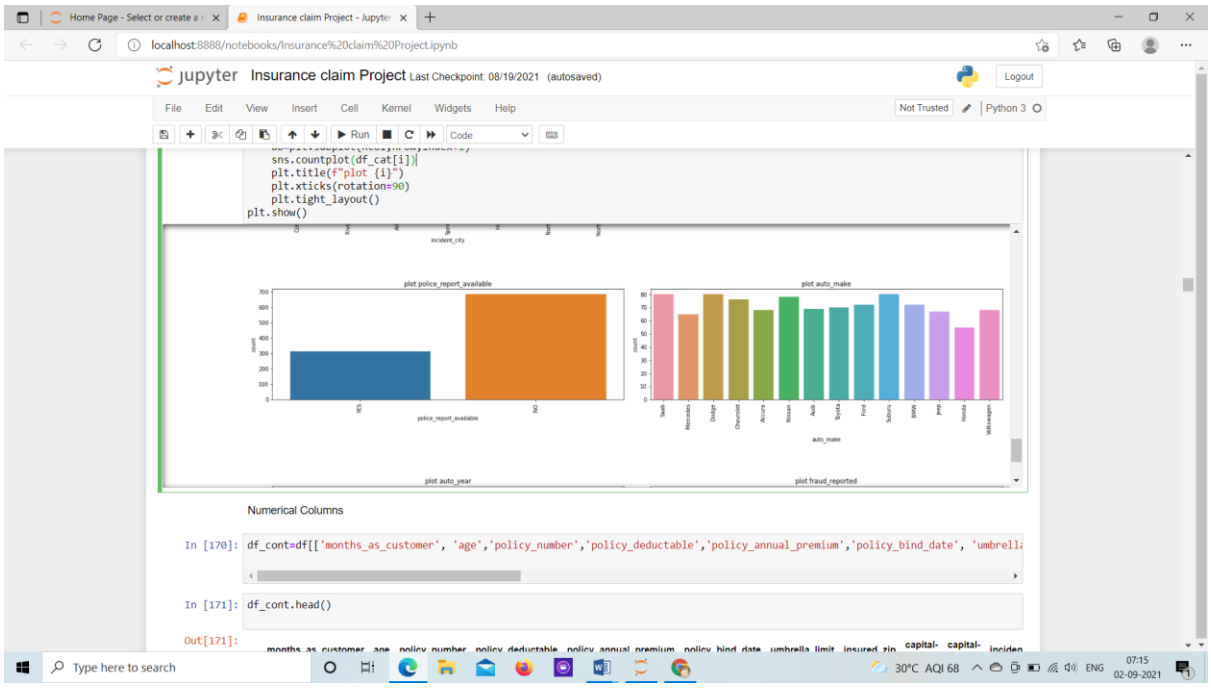
In [170]:
```
df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrella
```

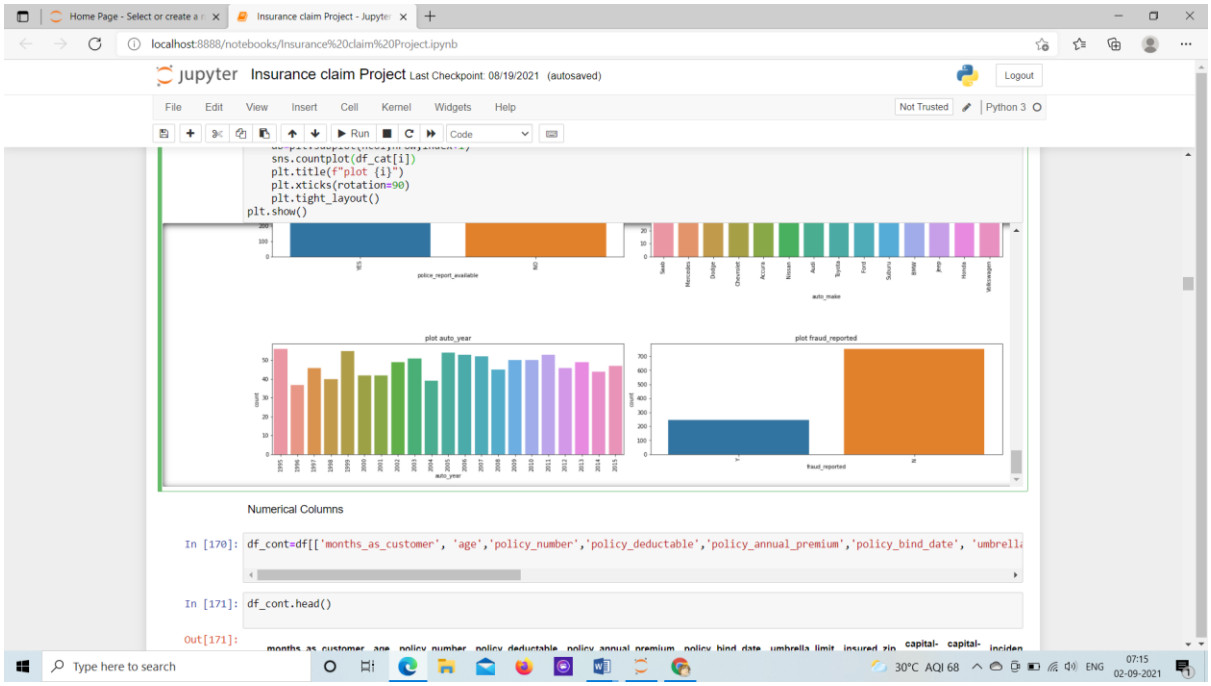
In [171]:
```
df_cont.head()
```

Out[171]:

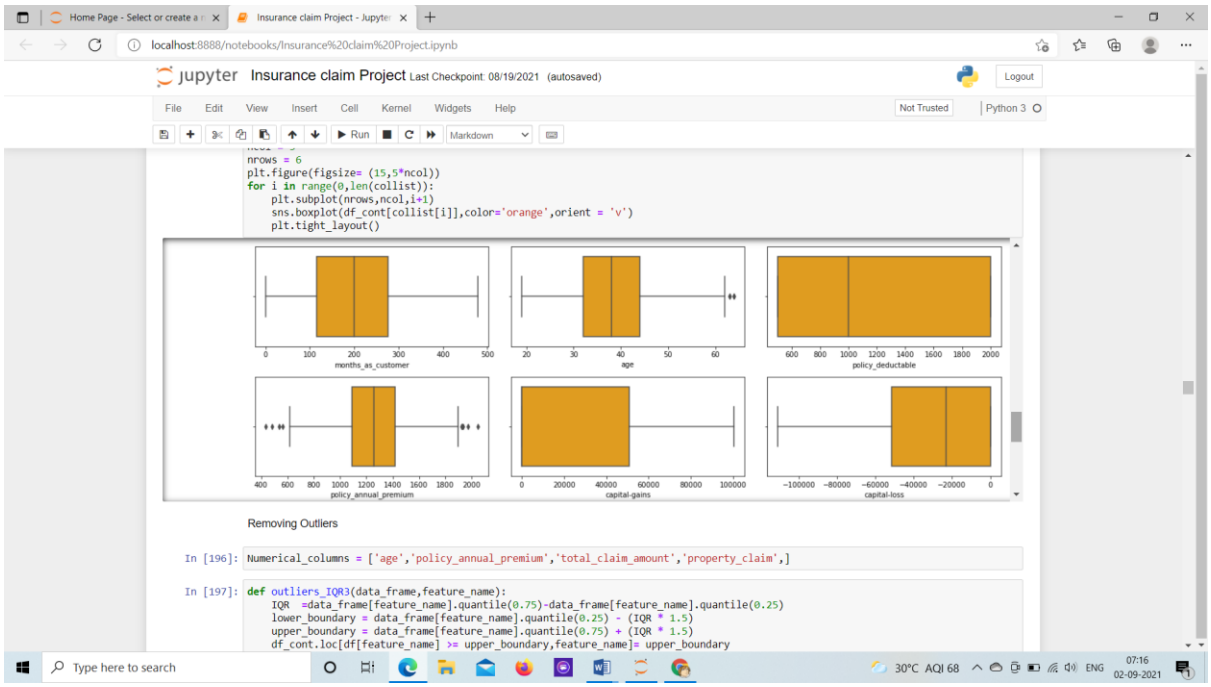
Type here to search 30°C AQI 68 ENG 07:14 02-09-2021

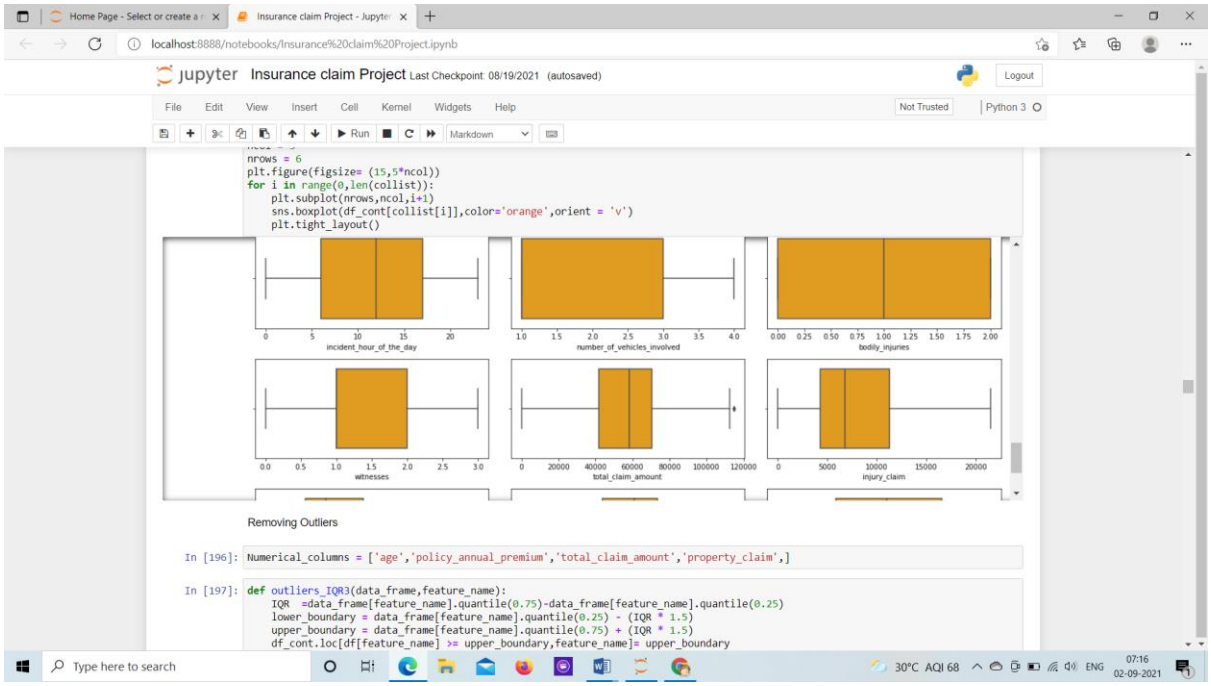

Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
sns.countplot(df_cat[i])
plt.title(f"plot {i}")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
plot incident_city
plot property_damage
plot police_report_available
plot auto_make
Numerical Columns
In [170]: df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrella
In [171]: df_cont.head()
Out[171]:


Insurance claim Project Last Checkpoint: 08/19/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
sns.countplot(df_cat[i])
plt.title(f"plot {i}")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
plot police_report_available
plot auto_make
plot auto_year
plot fraud_reported
Numerical Columns
In [170]: df_cont=df[['months_as_customer', 'age','policy_number','policy_deductable','policy_annual_premium','policy_bind_date', 'umbrella
In [171]: df_cont.head()
Out[171]:

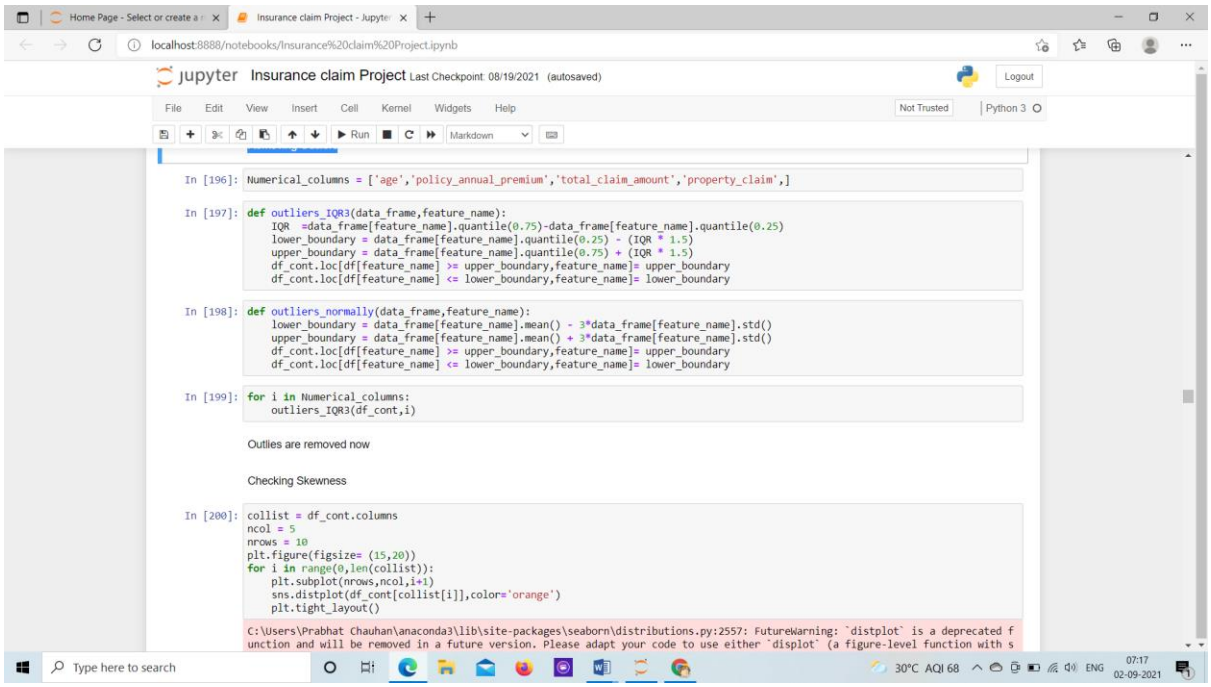


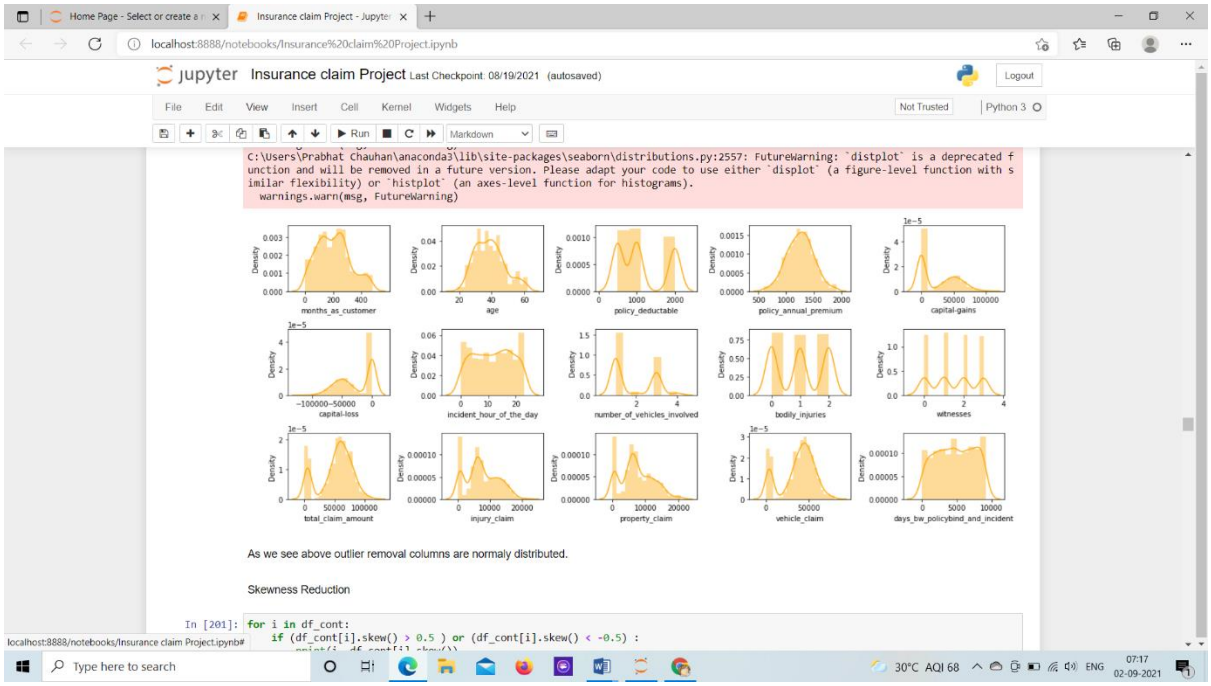
Checking Outliers using Boxplot



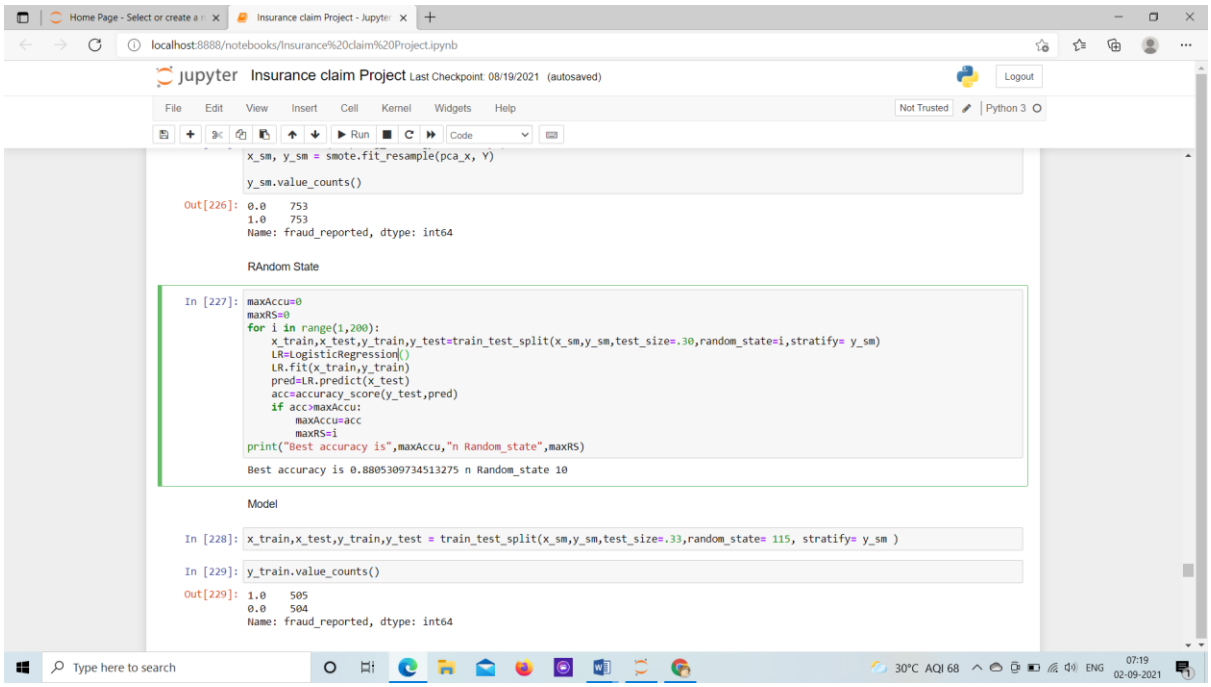


Removing Outliers

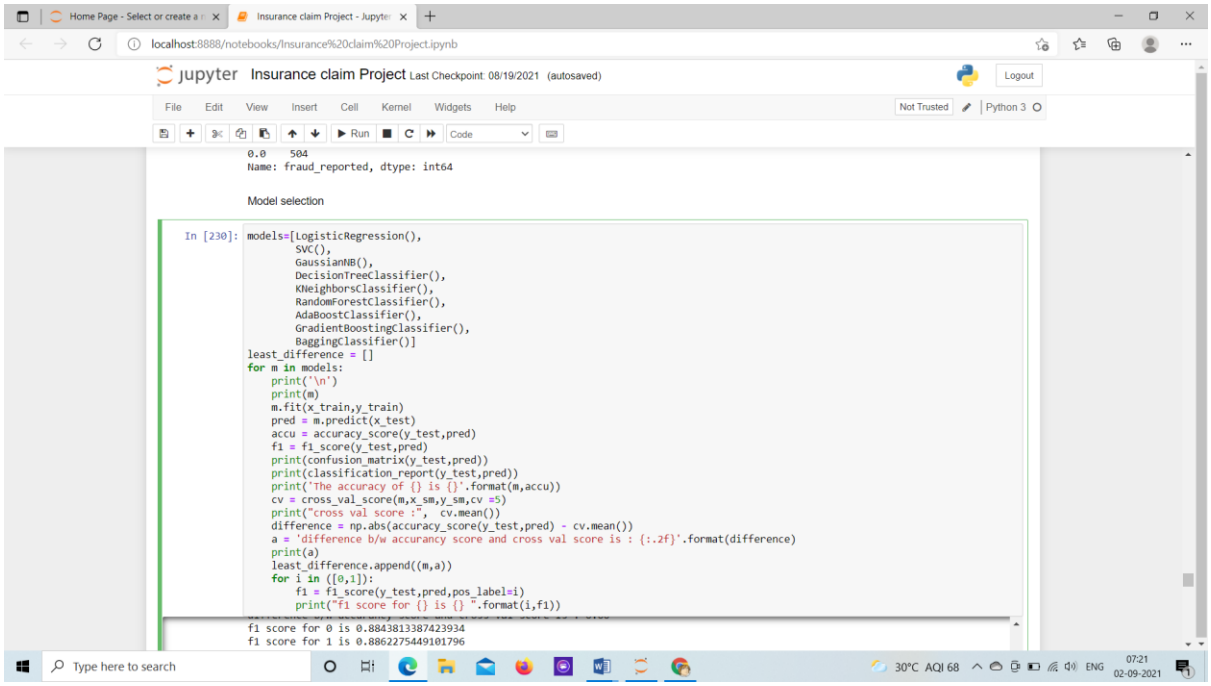




Random State



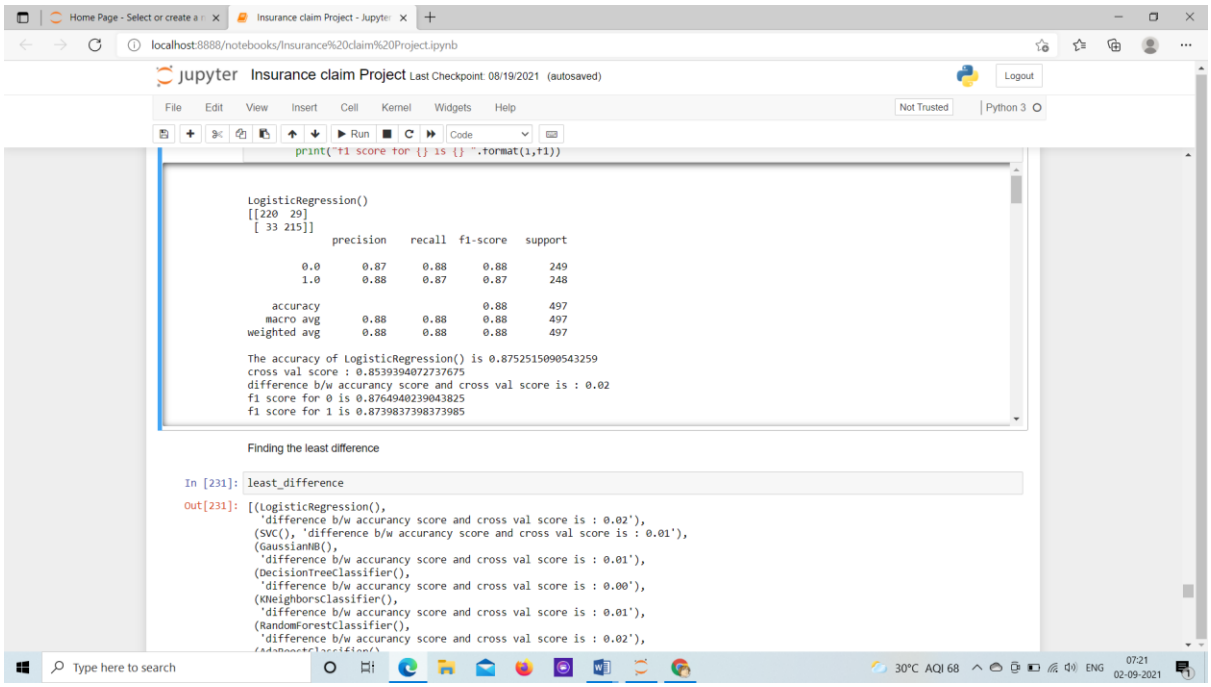
Model selection



```
0.0    504
Name: fraud_reported, dtype: int64
```

Model selection

```
In [230]: models=[LogisticRegression(),
                  SVC(),
                  GaussianNB(),
                  DecisionTreeClassifier(),
                  KNeighborsClassifier(),
                  RandomForestClassifier(),
                  AdaBoostClassifier(),
                  GradientBoostingClassifier(),
                  BaggingClassifier()]
          least_difference = []
          for m in models:
              print('\n')
              print(m)
              m.fit(x_train,y_train)
              pred = m.predict(x_test)
              accu = accuracy_score(y_test,pred)
              f1 = f1_score(y_test,pred)
              print(confusion_matrix(y_test,pred))
              print(classification_report(y_test,pred))
              print('The accuracy of {} is {}'.format(m,accu))
              cv = cross_val_score(m,x_sm,y_sm,cv =5)
              print("cross val score :",  cv.mean())
              difference = np.abs(accuracy_score(y_test,pred) - cv.mean())
              a = 'difference b/w accurancy score and cross val score is : {:.2f}'.format(difference)
              print(a)
              least_difference.append((m,a))
              for i in ([0,1]):
                  f1 = f1_score(y_test,pred,pos_label=i)
                  print("f1 score for {} is {} ".format(i,f1))

f1 score for 0 is 0.8843813387423934
f1 score for 1 is 0.8862275449101796
```

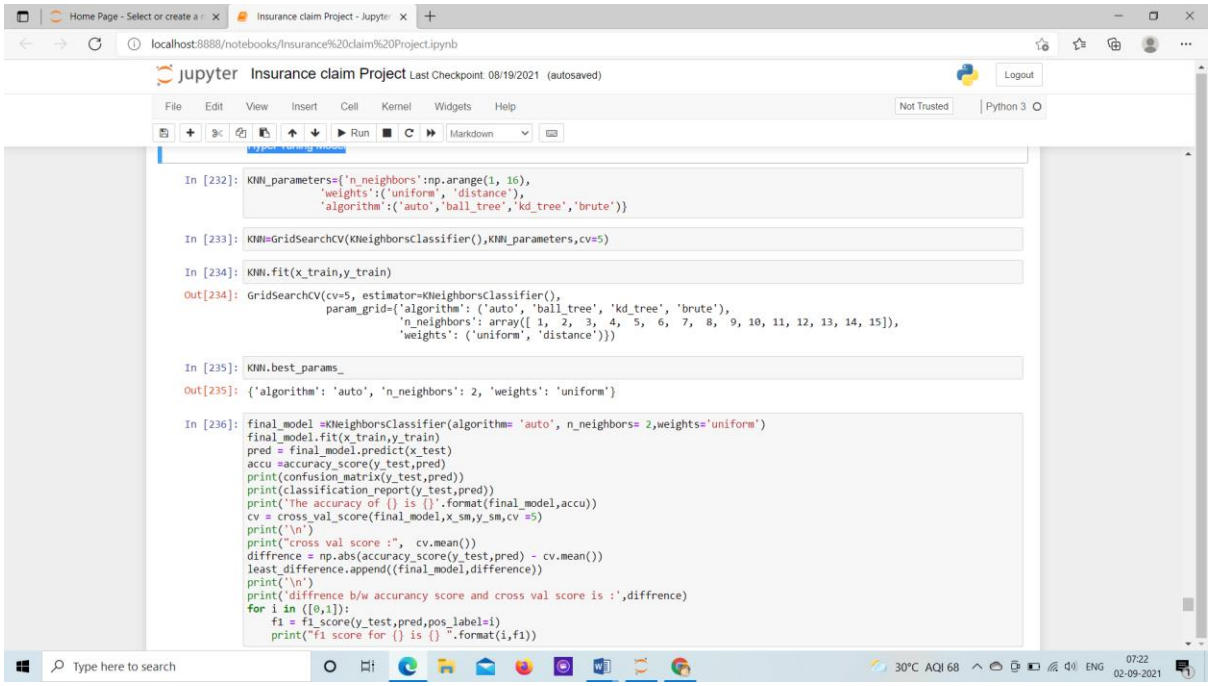


```
                  print("f1 score for {} is {} ".format(i,f1))


LogisticRegression()
[[220  29]
 [ 33 215]]
              precision    recall  f1-score   support

         0.0       0.87      0.88      0.88       249
         1.0       0.88      0.87      0.87       248

    accuracy                           0.88       497
   macro avg       0.88      0.88      0.88       497
weighted avg       0.88      0.88      0.88       497

The accuracy of LogisticRegression() is 0.8752515090543259
cross val score : 0.8539394072737675
difference b/w accurancy score and cross val score is : 0.02
f1 score for 0 is 0.8764940239043825
f1 score for 1 is 0.8739837398373985
```

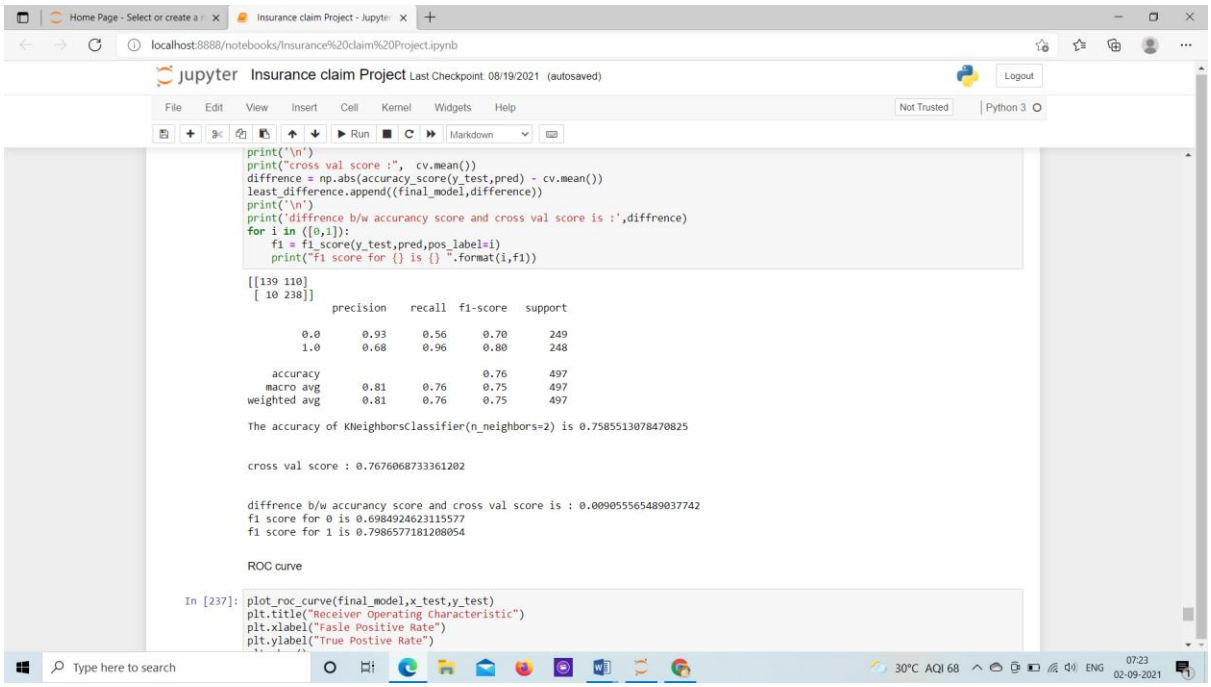
Finding the least difference

```
In [231]: least_difference

Out[231]: [(LogisticRegression(),
            'difference b/w accurancy score and cross val score is : 0.02'),
           (SVC(), 'difference b/w accurancy score and cross val score is : 0.01'),
           (GaussianNB(),
            'difference b/w accurancy score and cross val score is : 0.01'),
           (DecisionTreeClassifier(),
            'difference b/w accurancy score and cross val score is : 0.00'),
           (KNeighborsClassifier(),
            'difference b/w accurancy score and cross val score is : 0.01'),
           (RandomForestClassifier(),
            'difference b/w accurancy score and cross val score is : 0.02'),
```

Hyper Tuning Model

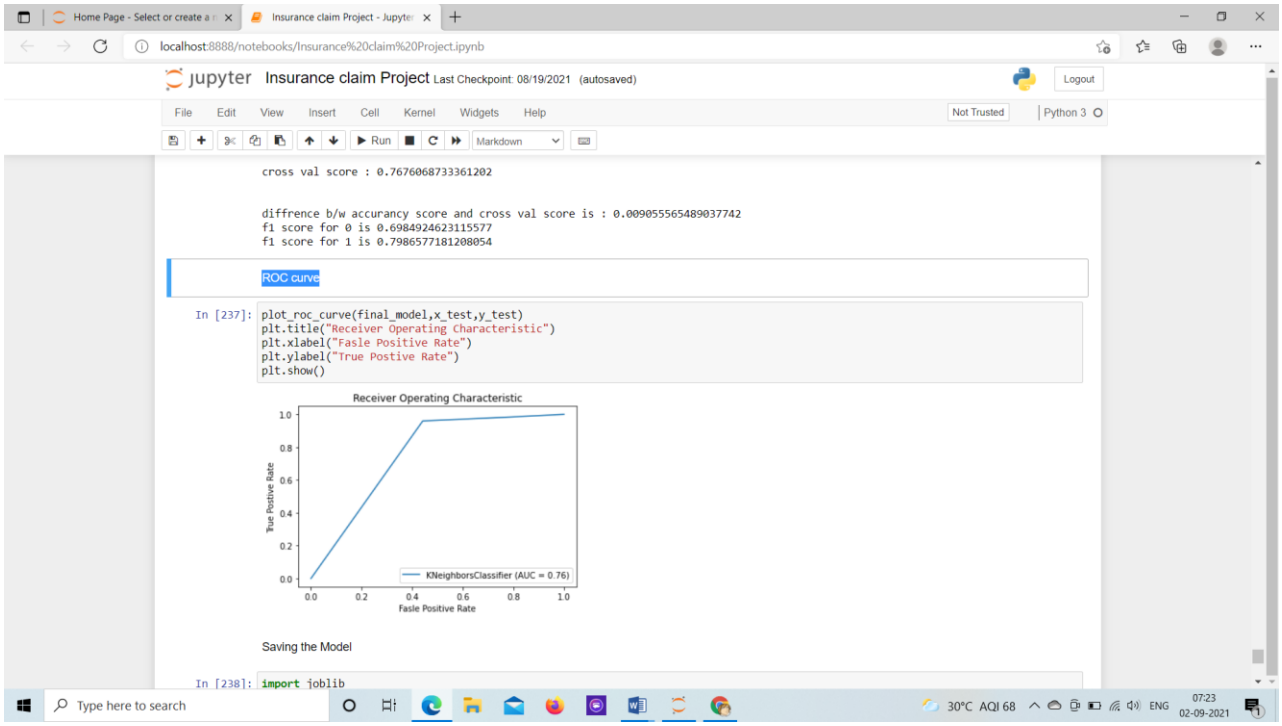


```
In [232]: KNN_parameters={'n_neighbors':np.arange(1, 16),
                          'weights':('uniform', 'distance'),
                          'algorithm':('auto','ball_tree','kd_tree','brute')}

In [233]: KNN=GridSearchCV(KNeighborsClassifier(),KNN_parameters,cv=5)

In [234]: KNN.fit(x_train,y_train)

Out[234]: GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
                       param_grid={'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'),
                                   'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
                                   'weights': ('uniform', 'distance')})

In [235]: KNN.best_params_

Out[235]: {'algorithm': 'auto', 'n_neighbors': 2, 'weights': 'uniform'}

In [236]: final_model =KNeighborsClassifier(algorithm= 'auto', n_neighbors= 2,weights='uniform')
          final_model.fit(x_train,y_train)
          pred = final_model.predict(x_test)
          accu =accuracy_score(y_test,pred)
          print(confusion_matrix(y_test,pred))
          print(classification_report(y_test,pred))
          print('The accuracy of {} is {}'.format(final_model,accu))
          cv = cross_val_score(final_model,x_sm,y_sm,cv =5)
          print('\n')
          print("cross val score :",  cv.mean())
          diffrence = np.abs(accuracy_score(y_test,pred) - cv.mean())
          least_difference.append((final_model,difference))
          print('\n')
          print('diffrence b/w accurancy score and cross val score is :',diffrence)
          for i in ([0,1]):
              f1 = f1_score(y_test,pred,pos_label=i)
              print("f1 score for {} is {} ".format(i,f1))
```



```
          print('\n')
          print("cross val score :",  cv.mean())
          diffrence = np.abs(accuracy_score(y_test,pred) - cv.mean())
          least_difference.append((final_model,difference))
          print('\n')
          print('diffrence b/w accurancy score and cross val score is :',diffrence)
          for i in ([0,1]):
              f1 = f1_score(y_test,pred,pos_label=i)
              print("f1 score for {} is {} ".format(i,f1))

[[139 110]
 [ 10 238]]
              precision    recall  f1-score   support

         0.0       0.93      0.56      0.70       249
         1.0       0.68      0.96      0.80       248

    accuracy                           0.76       497
   macro avg       0.81      0.76      0.75       497
weighted avg       0.81      0.76      0.75       497

The accuracy of KNeighborsClassifier(n_neighbors=2) is 0.7585513078470825


cross val score : 0.7676068733361202


diffrence b/w accurancy score and cross val score is : 0.009055565489037742
f1 score for 0 is 0.6984924623115577
f1 score for 1 is 0.7986577181208054

ROC curve

In [237]: plot_roc_curve(final_model,x_test,y_test)
          plt.title("Receiver Operating Characteristic")
          plt.xlabel("Fasle Positive Rate")
          plt.ylabel("True Postive Rate")
```

ROC curve



Result

Accuracy proportion we have is 75%