Configuration Manual

MSc Research Project
Data Analytics

Naumaan Mohammed Saeed Kazi
Student ID: x18130208

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Naumaan Mohammed Saeed Kazi
Student ID:	x18130208
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	15

Signature:	
Date:	12th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Naumaan Mohammed Saeed Kazi
x18130208
MSc Research Project in Data Analytics
11th December 2019

1 Introduction

This configuration manual would present the software requirement, hardware requirement and system setup. Along with this the codes that have been used for programming that is written for the implementation of the research study:

”Using Machine Learning Models to Study Human Error Related Factors in Aviation Accidents and Incidents”

2 System Configuration

2.1 Hardware

Processor: Intel(R) Core i7-7200U CPU@2.6GHz GPU: NVIDIA GeForce GTX 1060Ti RAM:8GB Storage: 1 TB HDD; Operating system: Windows 10, 64-bit.

2.2 Software

* Python using Jupyter notebook: Data analysis, data cleaning, pre-processing and manipulation. Feature selection and the implementation of machine learning algorithms and plots were done using libraries in Python.
* Microsoft Excel:Used for saving of data, data exploration, and plots for explorations.

3 Project Development

Steps for project development are as follows: data exploration, data pre-processing (exporting data, handling mull values, removing unused attributes and again calculating features) and using hyper-parameter for model tuning. Number of codes have implemented during the many steps for analysing such as feature selection using Pearson Correlation, Carmer’s V Rule and Random Forest for selection. K-fold validation using both approaches Stratified K-fold and 10-cross fold validation on all the models of machine learning used in this research. Generating confusion matrix, experimenting models and creating charts. In the later section codes for the study are shown with detailed information step by step.

3.1 Data Preparation & Pre-processing

The original data was downloaded from National Transportation Safety Board (NTSB) [1]. Which originally consist of a zip file, in which had the data file in MS Access database type. After which it was extracted to CS file file format to load in Python. After applying feature selection on the data, we had split the data into two categorizes test and train data for running the model. A systematic view of this data split is shown in the Figure 1

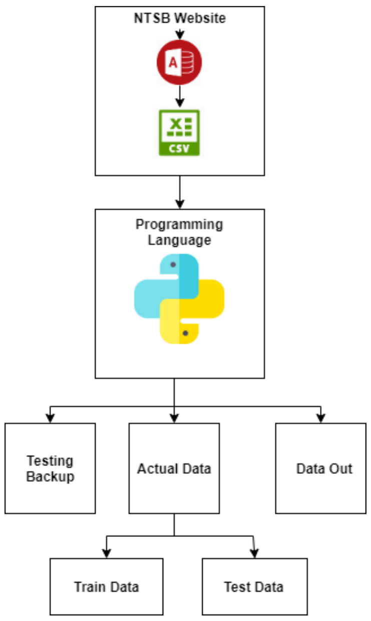


Figure 1: Data Collection and Splitting

The libraries used for data splitting is "sklearn.model_selection" from Python [2]. As 80% data is used for training purpose and the remaining 20% for testing. The data out is where the data that is no longer required for analysis i.e. discarded data. In data pre-processing contains label encoding as the values were "low" and "medium" which is shown in later. As the data required to be in 0's and 1' prior to model fitting. Data also needs to be normalized prior to fitting model, especially in Neural Network.

4 Code used for Machine Learning Models

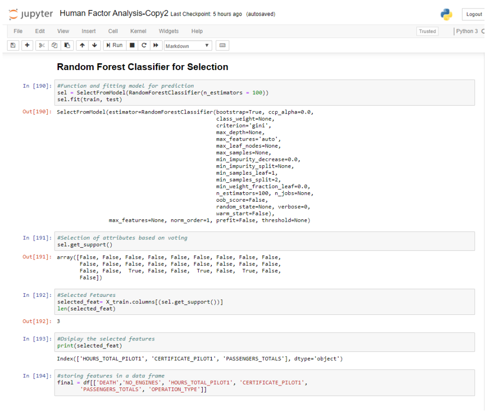
The coding for this study has been performed using Jupyter Notebook. As the complete code was executed from correlation, cleaning and model in Python language only. The layout of this codes for explanation are designed as follows: feature selection, cross validation, class imbalance, label encoder and experiments on models.

[1] `https://www.ntsb.gov/_layouts/ntsb.aviation/index.aspx`

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

4.1 Choosing Method for Feature Engineering

These three types are compared and contrasted against each other on depending which one best suit our classification techniques. As all these three methods are run on a single model which is XGBoost to see which gives the best accurate results. In Figure 2 we have used "SelectFromModel" function us used for random forest classifier to get the best feature from our data set[3]

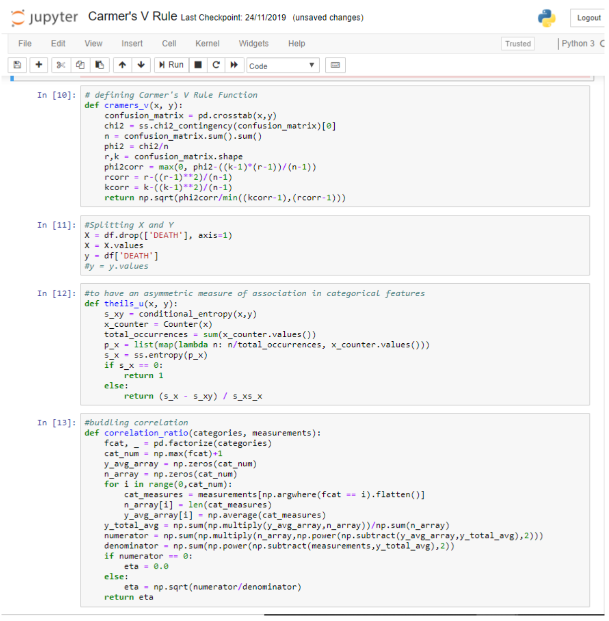

jupyter Human Factor Analysis-Copy2 Last Checkpoint: 5 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

Random Forest Classifier for Selection

```
In [190]: #Function and fitting model for prediction
          sel = SelectFromModel(RandomForestClassifier(n_estimators = 100))
          sel.fit(train, test)

Out[190]: SelectFromModel(estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                            class_weight=None,
                                                            criterion='gini',
                                                            max_depth=None,
                                                            max_features='auto',
                                                            max_leaf_nodes=None,
                                                            max_samples=None,
                                                            min_impurity_decrease=0.0,
                                                            min_impurity_split=None,
                                                            min_samples_leaf=1,
                                                            min_samples_split=2,
                                                            min_weight_fraction_leaf=0.0,
                                                            n_estimators=100, n_jobs=None,
                                                            oob_score=False,
                                                            random_state=None, verbose=0,
                                                            warm_start=False),
                          max_features=None, norm_order=1, prefit=False, threshold=None)

In [191]: #Selection of attributes based on voting
          sel.get_support()

Out[191]: array([False, False, False, False, False, False, False, False, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False,  True, False, False,  True, False,  True, False,
                 False])

In [192]: #Selected Fetaures
          selected_feat= X_train.columns[(sel.get_support())]
          len(selected_feat)

Out[192]: 3

In [193]: #Dsiplay the selected features
          print(selected_feat)

          Index(['HOURS_TOTAL_PILOT1', 'CERTIFICATE_PILOT1', 'PASSENGERS_TOTALS'], dtype='object')

In [194]: #storing features in a data frame
          final = df[['DEATH','NO_ENGINES', 'HOURS_TOTAL_PILOT1', 'CERTIFICATE_PILOT1',
                 'PASSENGERS_TOTALS', 'OPERATION_TYPE']]
```

Figure 2: Feature Selection Using Cramer's V Rule

Cramer's V Rule is most optimal for categorical data. In our dataset those variables which are categorical in nature are chosen and on them Carmer's V Rule is applied using the XGBoost model to check the accurate selection Figure 3. First, the cramer's function is defined using x and y. It used chi2 technique for feature selection[4].


jupyter Carmer's V Rule Last Checkpoint: 24/11/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

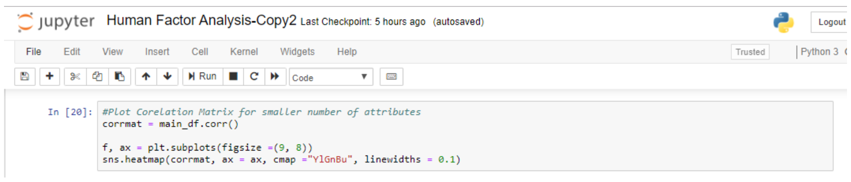
```
In [10]: # defining Carmer's V Rule Function
         def cramers_v(x, y):
             confusion_matrix = pd.crosstab(x,y)
             chi2 = ss.chi2_contingency(confusion_matrix)[0]
             n = confusion_matrix.sum().sum()
             phi2 = chi2/n
             r,k = confusion_matrix.shape
             phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
             rcorr = r-((r-1)**2)/(n-1)
             kcorr = k-((k-1)**2)/(n-1)
             return np.sqrt(phi2corr/min((kcorr-1),(rcorr-1)))

In [11]: #Splitting X and Y
         X = df.drop(['DEATH'], axis=1)
         X = X.values
         y = df['DEATH']
         #y = y.values

In [12]: #to have an asymmetric measure of association in categorical features
         def theils_u(x, y):
             s_xy = conditional_entropy(x,y)
             x_counter = Counter(x)
             total_occurrences = sum(x_counter.values())
             p_x = list(map(lambda n: n/total_occurrences, x_counter.values()))
             s_x = ss.entropy(p_x)
             if s_x == 0:
                 return 1
             else:
                 return (s_x - s_xy) / s_xs_x

In [13]: #buidling correlation
         def correlation_ratio(categories, measurements):
             fcat, _ = pd.factorize(categories)
             cat_num = np.max(fcat)+1
             y_avg_array = np.zeros(cat_num)
             n_array = np.zeros(cat_num)
             for i in range(0,cat_num):
                 cat_measures = measurements[np.argwhere(fcat == i).flatten()]
                 n_array[i] = len(cat_measures)
                 y_avg_array[i] = np.average(cat_measures)
             y_total_avg = np.sum(np.multiply(y_avg_array,n_array))/np.sum(n_array)
             numerator = np.sum(np.multiply(n_array,np.power(np.subtract(y_avg_array,y_total_avg),2)))
             denominator = np.sum(np.power(np.subtract(measurements,y_total_avg),2))
             if numerator == 0:
                 eta = 0.0
             else:
                 eta = np.sqrt(numerator/denominator)
             return eta
```

Figure 3: Feature Selection Using Random Forest Selection

[3]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html.Itisoneoftheverywidelyandreliableselectionmodels

[4]https://towardsdatascience.com/the-search-for-categorical-correlation-a1cf7f1888c9

Correlation was accomplished using Pearson's correlation matrix. As the value above 0.8 is extreme correlation and above 0.4 is medium correlations Figure 20.

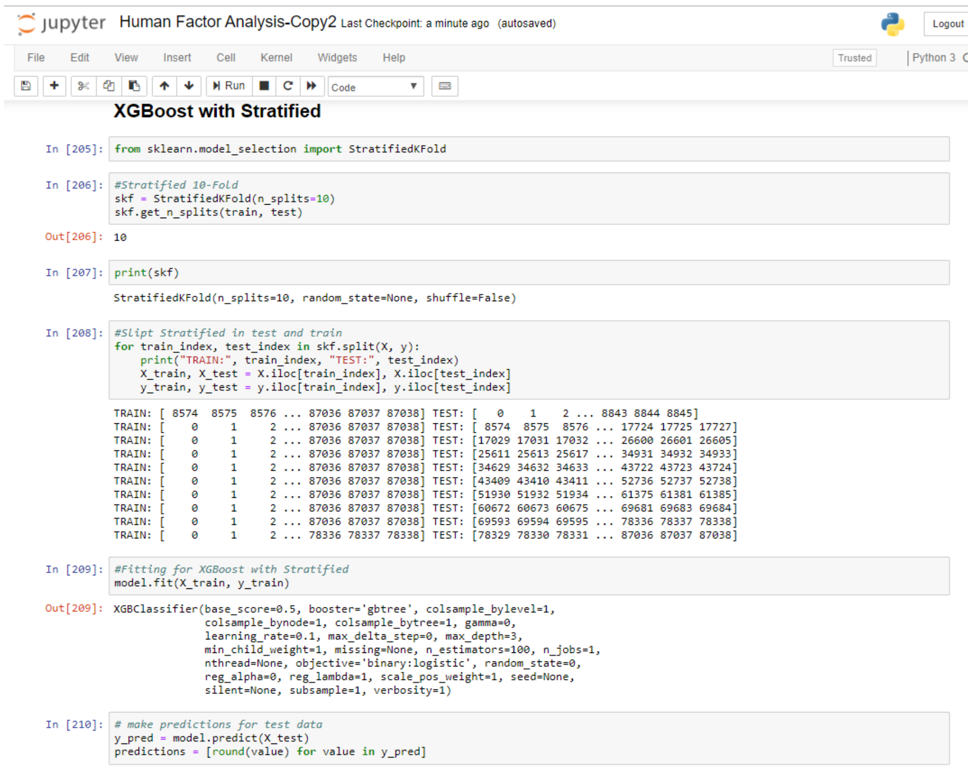

jupyter
Human Factor Analysis-Copy2 Last Checkpoint: 5 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [20]: #Plot Corelation Matrix for smaller number of attributes
         cormat = main_df.corr()

         f, ax = plt.subplots(figsize =(9, 8))
         sns.heatmap(cormat, ax = ax, cmap ="YlGnBu", linewidths = 0.1)
```

Figure 4: Feature Selection Using Correlation Matrices

4.2 Cross Validation

Cross validation (CV) is used in Zhang and Mahadevan (2019), the two approaches were opted first is the Stratified K-Fold and second is 10-fold CV. For all the models CV is used in the Figure 20 has shown for XGBoost. CV is helpful for problem like over-fitting & selection bias, as it gives meaningful insight about model. Using "Cross_val_score" function we implemented CV [5].


jupyter
Human Factor Analysis-Copy2 Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

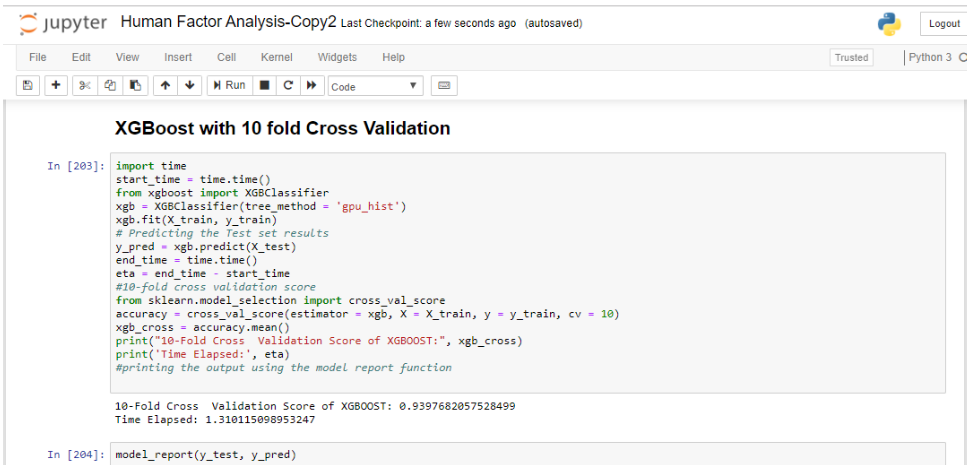
XGBoost with Stratified

```
In [205]: from sklearn.model_selection import StratifiedKFold

In [206]: #Stratified 10-Fold
          skf = StratifiedKFold(n_splits=10)
          skf.get_n_splits(train, test)

Out[206]: 10

In [207]: print(skf)

          StratifiedKFold(n_splits=10, random_state=None, shuffle=False)

In [208]: #Slipt Stratified in test and train
          for train_index, test_index in skf.split(X, y):
              print("TRAIN:", train_index, "TEST:", test_index)
              X_train, X_test = X.iloc[train_index], X.iloc[test_index]
              y_train, y_test = y.iloc[train_index], y.iloc[test_index]

          TRAIN: [ 8574  8575  8576 ... 87036 87037 87038] TEST: [    0     1     2 ... 8843 8844 8845]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [ 8574  8575  8576 ... 17724 17725 17727]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [17029 17031 17032 ... 26600 26601 26605]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [25611 25613 25617 ... 34931 34932 34933]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [34629 34632 34633 ... 43722 43723 43724]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [43409 43410 43411 ... 52736 52737 52738]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [51930 51932 51934 ... 61375 61381 61385]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [60672 60673 60675 ... 69681 69683 69684]
          TRAIN: [    0     1     2 ... 87036 87037 87038] TEST: [69593 69594 69595 ... 78336 78337 78338]
          TRAIN: [    0     1     2 ... 78336 78337 78338] TEST: [78329 78330 78331 ... 87036 87037 87038]

In [209]: #Fitting for XGBoost with Stratified
          model.fit(X_train, y_train)

Out[209]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                        colsample_bynode=1, colsample_bytree=1, gamma=0,
                        learning_rate=0.1, max_delta_step=0, max_depth=3,
                        min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                        nthread=None, objective='binary:logistic', random_state=0,
                        reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                        silent=None, subsample=1, verbosity=1)

In [210]: # make predictions for test data
          y_pred = model.predict(X_test)
          predictions = [round(value) for value in y_pred]
```

Figure 5: Stratified K-Fold for XGBoost

The CV which is very effective with model improvement is stratified k fold. The function "StratifiedKFold" function is used setting the n splits to 10. As the train and test would be split as per the CV Figure 20.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

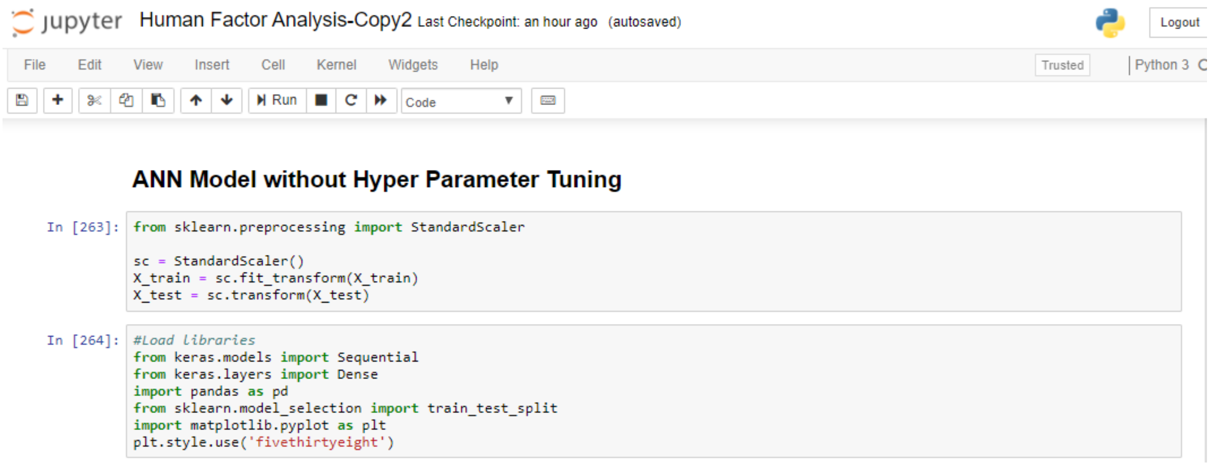

jupyter
Human Factor Analysis-Copy2 Last Checkpoint: a few seconds ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

XGBoost with 10 fold Cross Validation

```
In [203]: import time
          start_time = time.time()
          from xgboost import XGBClassifier
          xgb = XGBClassifier(tree_method = 'gpu_hist')
          xgb.fit(X_train, y_train)
          # Predicting the Test set results
          y_pred = xgb.predict(X_test)
          end_time = time.time()
          eta = end_time - start_time
          #10-fold cross validation score
          from sklearn.model_selection import cross_val_score
          accuracy = cross_val_score(estimator = xgb, X = X_train, y = y_train, cv = 10)
          xgb_cross = accuracy.mean()
          print("10-Fold Cross  Validation Score of XGBOOST:", xgb_cross)
          print('Time Elapsed:', eta)
          #printing the output using the model report function

          10-Fold Cross  Validation Score of XGBOOST: 0.9397682057528499
          Time Elapsed: 1.310115098953247

In [204]: model_report(y_test, y_pred)
```

Figure 6: 10-Fold Cross Validation for XGBoost

4.3 Experiment 1: ANN



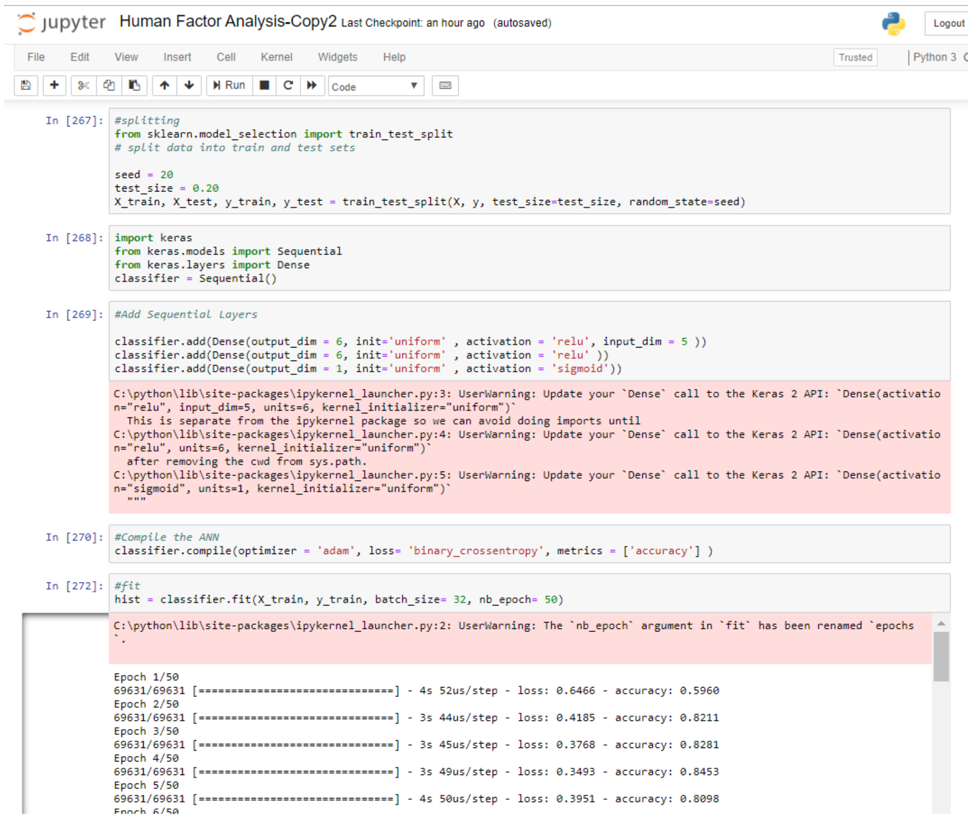
ANN Model without Hyper Parameter Tuning

```
In [263]: from sklearn.preprocessing import StandardScaler

          sc = StandardScaler()
          X_train = sc.fit_transform(X_train)
          X_test = sc.transform(X_test)

In [264]: #Load libraries
          from keras.models import Sequential
          from keras.layers import Dense
          import pandas as pd
          from sklearn.model_selection import train_test_split
          import matplotlib.pyplot as plt
          plt.style.use('fivethirtyeight')
```

Figure 7: Experiment on ANN

As observed from the Figure 7 before data is split and fitted into the model the data for ANN needs to be scaled. Using ”StandarScaler” function this scaling is achieved in ANN [6]. We have used Keras library for this implementation as the optimizer chosen is Adam and the activation function would be ’relu’. ANN was applied in Burnett and Si (2017)

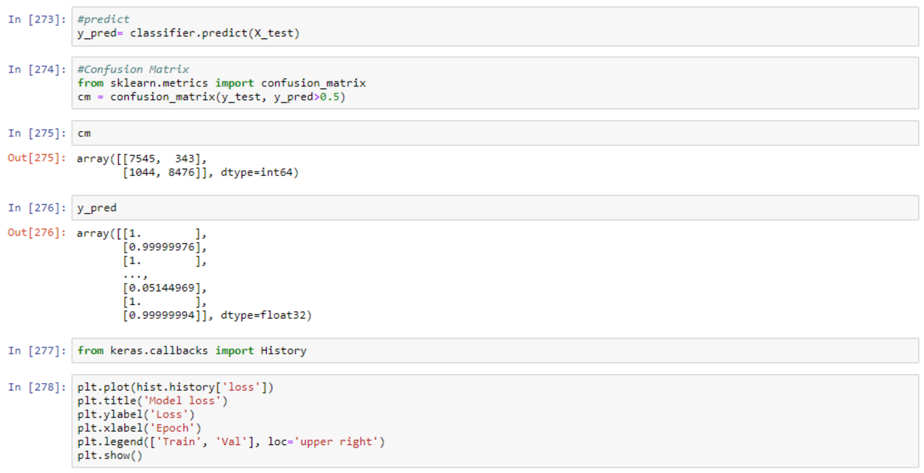
[6]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html



```
In [267]: #splitting
          from sklearn.model_selection import train_test_split
          # split data into train and test sets

          seed = 20
          test_size = 0.20
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)

In [268]: import keras
          from keras.models import Sequential
          from keras.layers import Dense
          classifier = Sequential()

In [269]: #Add Sequential Layers

          classifier.add(Dense(output_dim = 6, init='uniform' , activation = 'relu', input_dim = 5 ))
          classifier.add(Dense(output_dim = 6, init='uniform' , activation = 'relu' ))
          classifier.add(Dense(output_dim = 1, init='uniform' , activation = 'sigmoid'))

          C:\python\lib\site-packages\ipykernel_launcher.py:3: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activatio
          n="relu", input_dim=5, units=6, kernel_initializer="uniform")`
            This is separate from the ipykernel package so we can avoid doing imports until
          C:\python\lib\site-packages\ipykernel_launcher.py:4: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activatio
          n="relu", units=6, kernel_initializer="uniform")`
            after removing the cwd from sys.path.
          C:\python\lib\site-packages\ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activatio
          n="sigmoid", units=1, kernel_initializer="uniform")`
            """

In [270]: #Compile the ANN
          classifier.compile(optimizer = 'adam', loss= 'binary_crossentropy', metrics = ['accuracy'] )

In [272]: #fit
          hist = classifier.fit(X_train, y_train, batch_size= 32, nb_epoch= 50)

          C:\python\lib\site-packages\ipykernel_launcher.py:2: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs
          `.

          Epoch 1/50
          69631/69631 [==============================] - 4s 52us/step - loss: 0.6466 - accuracy: 0.5960
          Epoch 2/50
          69631/69631 [==============================] - 3s 44us/step - loss: 0.4185 - accuracy: 0.8211
          Epoch 3/50
          69631/69631 [==============================] - 3s 45us/step - loss: 0.3768 - accuracy: 0.8281
          Epoch 4/50
          69631/69631 [==============================] - 3s 49us/step - loss: 0.3493 - accuracy: 0.8453
          Epoch 5/50
          69631/69631 [==============================] - 4s 50us/step - loss: 0.3951 - accuracy: 0.8098
          Epoch 6/50
```

Figure 8: Experiment on ANN

The epoch is run for 10, 50, 100, 150 Burnett and Si (2017). on which the optimal performance was achieved at 50 for non-hyper parameter ANN Figure 8.



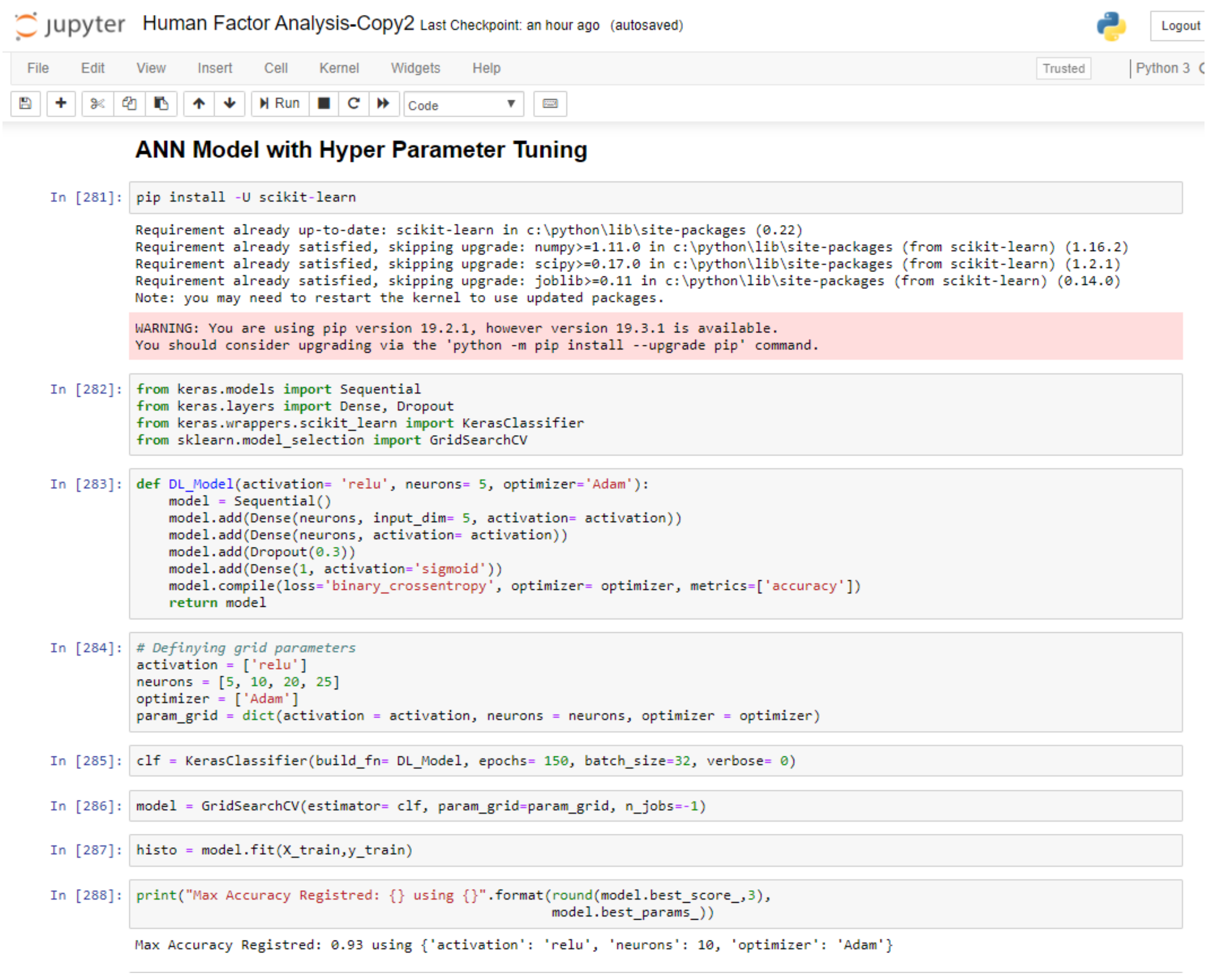
```
In [273]: #predict
          y_pred= classifier.predict(X_test)

In [274]: #Confusion Matrix
          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(y_test, y_pred>0.5)

In [275]: cm

Out[275]: array([[7545,  343],
                 [1044, 8476]], dtype=int64)

In [276]: y_pred

Out[276]: array([[1.        ],
                 [0.99999976],
                 [1.        ],
                 ...,
                 [0.05144969],
                 [1.        ],
                 [0.99999994]], dtype=float32)

In [277]: from keras.callbacks import History

In [278]: plt.plot(hist.history['loss'])
          plt.title('Model loss')
          plt.ylabel('Loss')
          plt.xlabel('Epoch')
          plt.legend(['Train', 'Val'], loc='upper right')
          plt.show()
```

Figure 9: Experiment on ANN

Classifier compile function is used for training the model of ANN as seen in Figure 9

4.4 Experiment 2: ANN Using Hyper-Parameter Tuning

Hyper-Parameter tuning is used to make the model better in terms of exploring more depths of the data and finding new hidden layers in the dataset. As observed from the Figure 10 we have defined only neurons for the Adam optimizer ANN function. It would run in the batches of 32 which the learning rate.



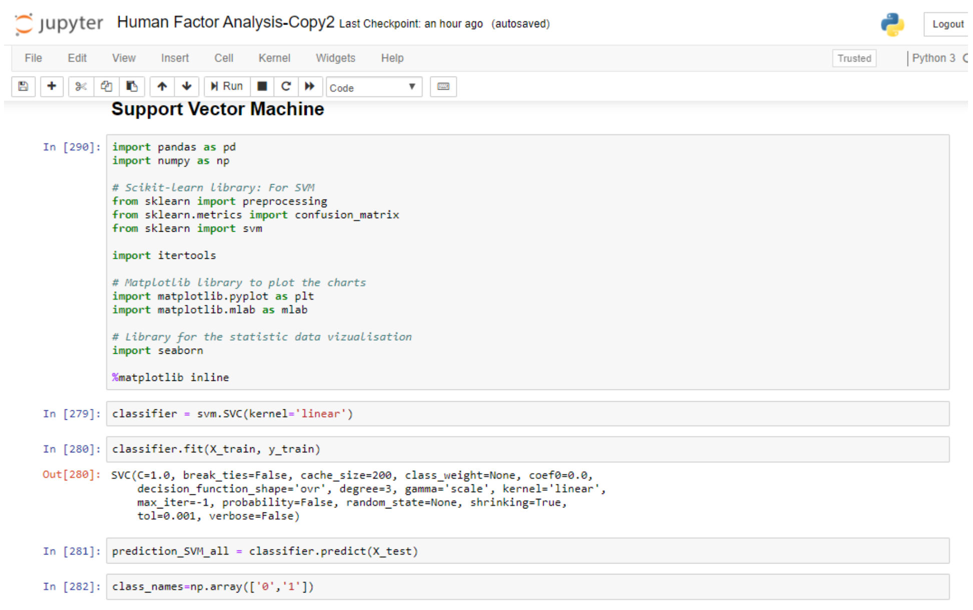
ANN Model with Hyper Parameter Tuning

```
In [281]: pip install -U scikit-learn

Requirement already up-to-date: scikit-learn in c:\python\lib\site-packages (0.22)
Requirement already satisfied, skipping upgrade: numpy>=1.11.0 in c:\python\lib\site-packages (from scikit-learn) (1.16.2)
Requirement already satisfied, skipping upgrade: scipy>=0.17.0 in c:\python\lib\site-packages (from scikit-learn) (1.2.1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in c:\python\lib\site-packages (from scikit-learn) (0.14.0)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 19.2.1, however version 19.3.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

In [282]: from keras.models import Sequential
          from keras.layers import Dense, Dropout
          from keras.wrappers.scikit_learn import KerasClassifier
          from sklearn.model_selection import GridSearchCV

In [283]: def DL_Model(activation= 'relu', neurons= 5, optimizer='Adam'):
              model = Sequential()
              model.add(Dense(neurons, input_dim= 5, activation= activation))
              model.add(Dense(neurons, activation= activation))
              model.add(Dropout(0.3))
              model.add(Dense(1, activation='sigmoid'))
              model.compile(loss='binary_crossentropy', optimizer= optimizer, metrics=['accuracy'])
              return model

In [284]: # Definying grid parameters
          activation = ['relu']
          neurons = [5, 10, 20, 25]
          optimizer = ['Adam']
          param_grid = dict(activation = activation, neurons = neurons, optimizer = optimizer)

In [285]: clf = KerasClassifier(build_fn= DL_Model, epochs= 150, batch_size=32, verbose= 0)

In [286]: model = GridSearchCV(estimator= clf, param_grid=param_grid, n_jobs=-1)

In [287]: histo = model.fit(X_train,y_train)

In [288]: print("Max Accuracy Registred: {} using {}".format(round(model.best_score_,3),
                                                             model.best_params_))

Max Accuracy Registred: 0.93 using {'activation': 'relu', 'neurons': 10, 'optimizer': 'Adam'}
```

Figure 10: Experiment on ANN using Hyper-Parameter Tuning

4.5 Experiment 3: SVM

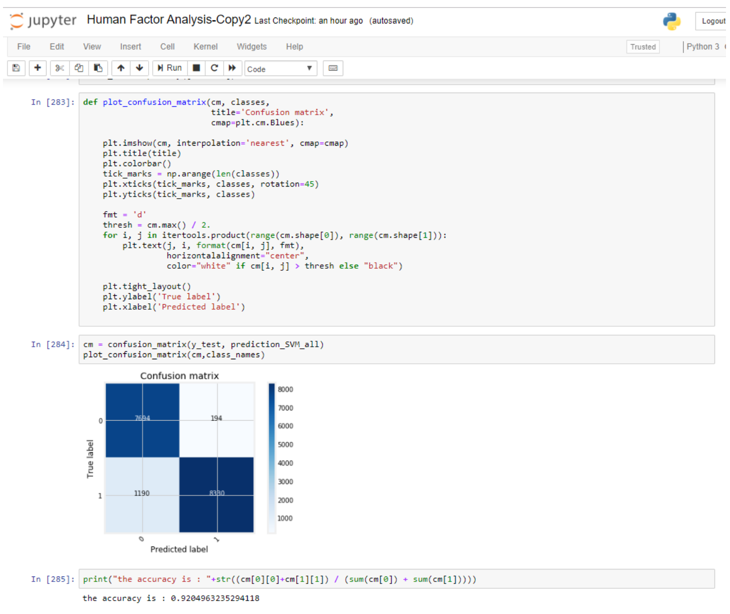
SVM was build using the ”sklearn” library in python as this was the slowest performing algorithm Kumar et al. (2016). the function used here is the SVC() function were the kernel defined is linear Burnett and Si (2017). Based on this the prediction is done. Figure 11.


Support Vector Machine

```
In [290]: import pandas as pd
          import numpy as np

          # Scikit-learn library: For SVM
          from sklearn import preprocessing
          from sklearn.metrics import confusion_matrix
          from sklearn import svm

          import itertools

          # Matplotlib library to plot the charts
          import matplotlib.pyplot as plt
          import matplotlib.mlab as mlab

          # Library for the statistic data vizualisation
          import seaborn

          %matplotlib inline

In [279]: classifier = svm.SVC(kernel='linear')

In [280]: classifier.fit(X_train, y_train)

Out[280]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
              decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
              max_iter=-1, probability=False, random_state=None, shrinking=True,
              tol=0.001, verbose=False)

In [281]: prediction_SVM_all = classifier.predict(X_test)

In [282]: class_names=np.array(['0','1'])
```

Figure 11: Experiment on SVM

4.6 Experiment 4: SVM using Cross Validation

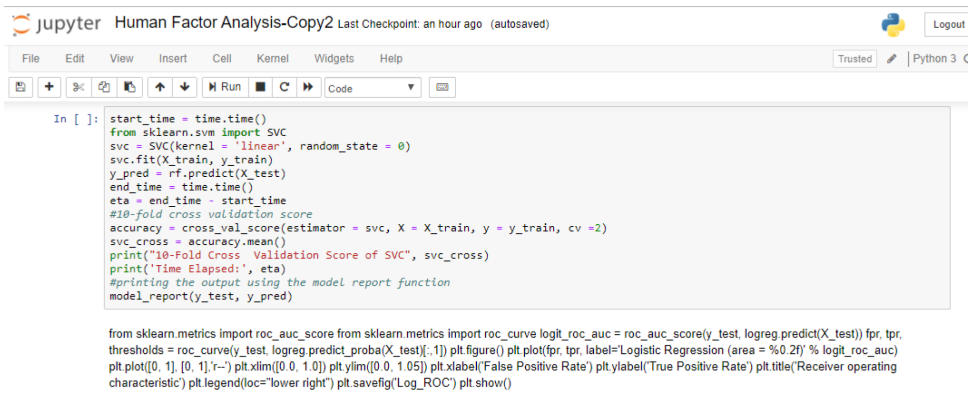


```
In [283]: def plot_confusion_matrix(cm, classes,
                                    title='Confusion matrix',
                                    cmap=plt.cm.Blues):

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

In [284]: cm = confusion_matrix(y_test, prediction_SVM_all)
          plot_confusion_matrix(cm,class_names)
```

```
In [285]: print("the accuracy is : "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))

          the accuracy is : 0.9204963235294118
```

Figure 12: Experiment on SVM Confusion Matrix

Using the "Confusion_Matrix" function the plot is sets and the prediction has been achieved. The model very precisely makes a very good results on the confusion matrix. Figure 12 Using estimator as svc we are able to get set that CM for the SVM. There has been 10 folds for the SVM Burnett and Si (2017), Management et al. (2014). Figure 13


jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

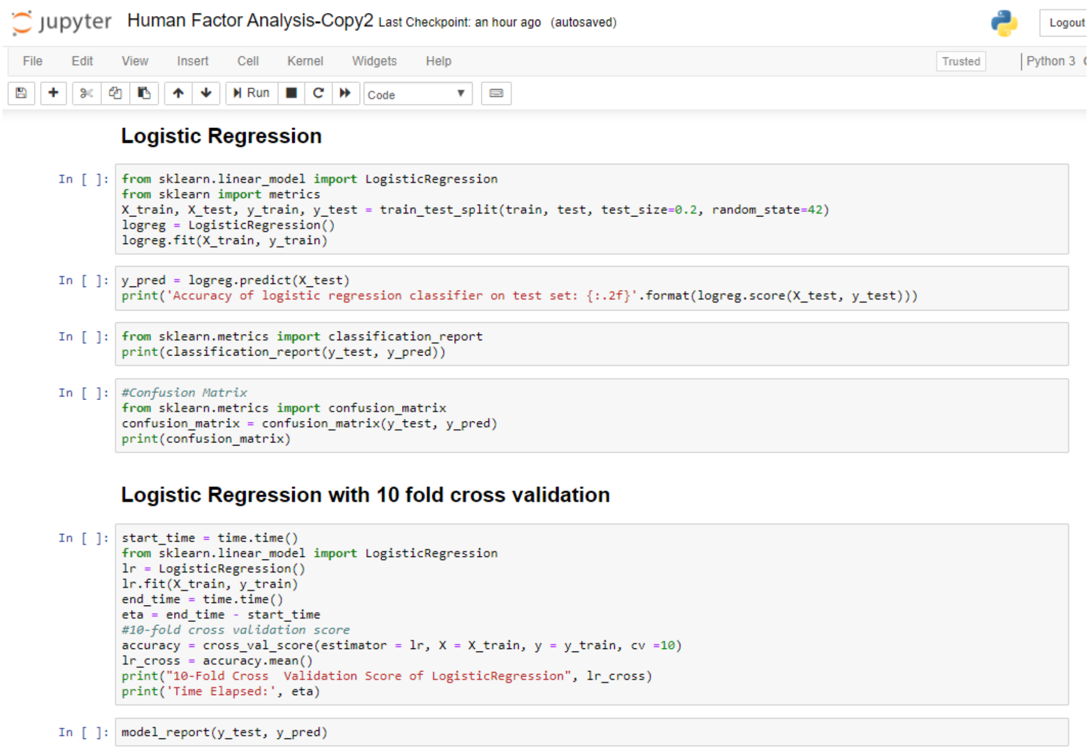
```
In [ ]: start_time = time.time()
        from sklearn.svm import SVC
        svc = SVC(kernel = 'linear', random_state = 0)
        svc.fit(X_train, y_train)
        y_pred = rf.predict(X_test)
        end_time = time.time()
        eta = end_time - start_time
        #10-fold cross validation score
        accuracy = cross_val_score(estimator = svc, X = X_train, y = y_train, cv =2)
        svc_cross = accuracy.mean()
        print("10-Fold Cross  Validation Score of SVC", svc_cross)
        print('Time Elapsed:', eta)
        #printing the output using the model report function
        model_report(y_test, y_pred)
```

from sklearn.metrics import roc_auc_score from sklearn.metrics import roc_curve logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test)) fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1]) plt.figure() plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc) plt.plot([0, 1], [0, 1],'r--') plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('Receiver operating characteristic') plt.legend(loc="lower right") plt.savefig('Log_ROC') plt.show()

Figure 13: Experiment on SVM using Cross Validation

4.7 Experiment 5: Logistic Regression

The most simplest form of model it was implemented using the "sk.learn.linear_model" for the LR model Bazargan and Guzhva (2007), Bazargan and Guzhva (2011). Here the log.reg.predict is set for prediction. As observed from Figure 14. As the 10-Fold CV is implemented with cross val score function Mathur et al. (2017), Kharoufah et al. (2018).


jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Logistic Regression

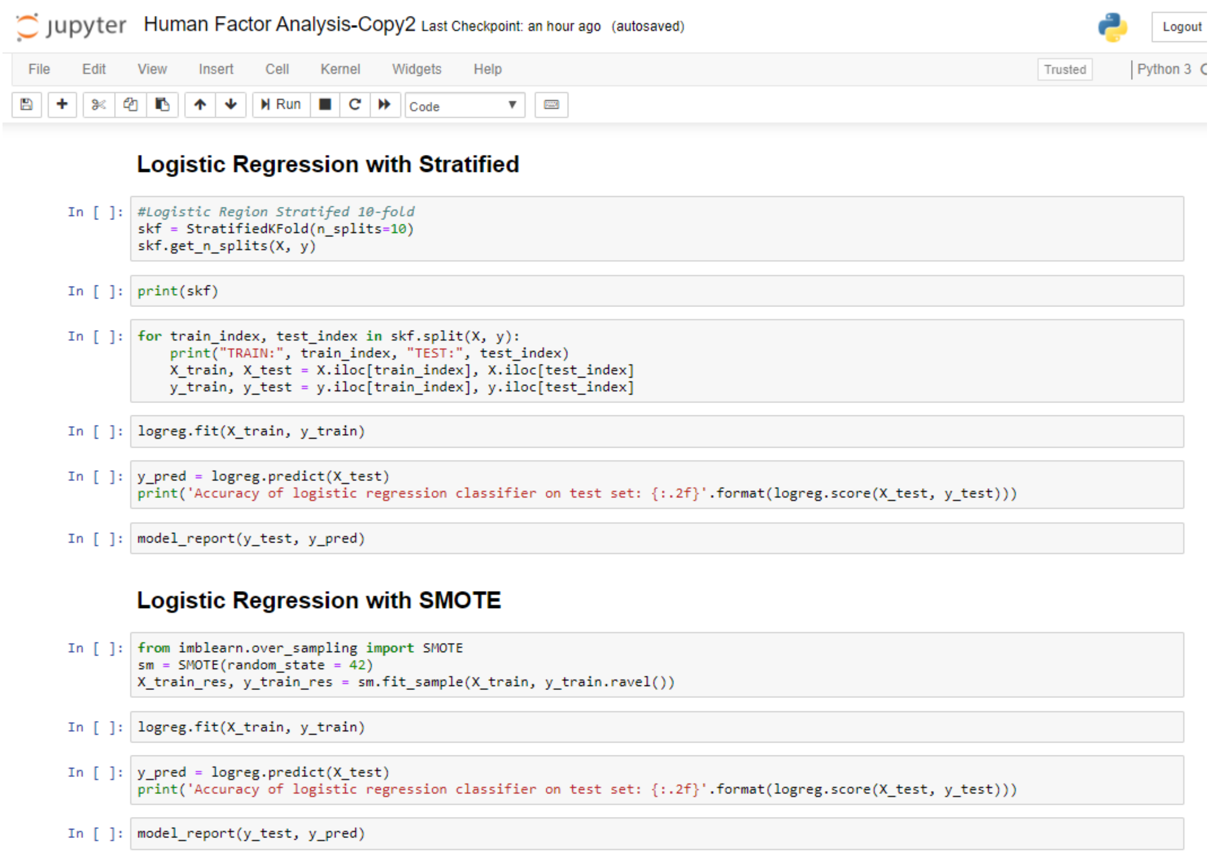
```
In [ ]: from sklearn.linear_model import LogisticRegression
        from sklearn import metrics
        X_train, X_test, y_train, y_test = train_test_split(train, test, test_size=0.2, random_state=42)
        logreg = LogisticRegression()
        logreg.fit(X_train, y_train)

In [ ]: y_pred = logreg.predict(X_test)
        print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

In [ ]: from sklearn.metrics import classification_report
        print(classification_report(y_test, y_pred))

In [ ]: #Confusion Matrix
        from sklearn.metrics import confusion_matrix
        confusion_matrix = confusion_matrix(y_test, y_pred)
        print(confusion_matrix)
```

Logistic Regression with 10 fold cross validation

```
In [ ]: start_time = time.time()
        from sklearn.linear_model import LogisticRegression
        lr = LogisticRegression()
        lr.fit(X_train, y_train)
        end_time = time.time()
        eta = end_time - start_time
        #10-fold cross validation score
        accuracy = cross_val_score(estimator = lr, X = X_train, y = y_train, cv =10)
        lr_cross = accuracy.mean()
        print("10-Fold Cross  Validation Score of LogisticRegression", lr_cross)
        print('Time Elapsed:', eta)

In [ ]: model_report(y_test, y_pred)
```

Figure 14: Experiment on Logistic Regression and using 10 Fold Cross Validation

As observed from the Figure 15 we have implemented the Stratified fold with 10 CV using the function ”StratifiedKFold” using the sciki learn [7]. It helps in exploring the data and finding better outputs Christopher and Appavu (2013). The SMOTE on data the model of Lr is run to handle the class imbalance using the library ”imblearn.over_sampling”. As the number of states for random synthetic sampling are restricted to 42Hofmann (2019).



Logistic Regression with Stratified

```
In [ ]: #Logistic Region Stratifed 10-fold
        skf = StratifiedKFold(n_splits=10)
        skf.get_n_splits(X, y)

In [ ]: print(skf)

In [ ]: for train_index, test_index in skf.split(X, y):
            print("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

In [ ]: logreg.fit(X_train, y_train)

In [ ]: y_pred = logreg.predict(X_test)
        print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

In [ ]: model_report(y_test, y_pred)
```

Logistic Regression with SMOTE

```
In [ ]: from imblearn.over_sampling import SMOTE
        sm = SMOTE(random_state = 42)
        X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())

In [ ]: logreg.fit(X_train, y_train)

In [ ]: y_pred = logreg.predict(X_test)
        print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

In [ ]: model_report(y_test, y_pred)
```

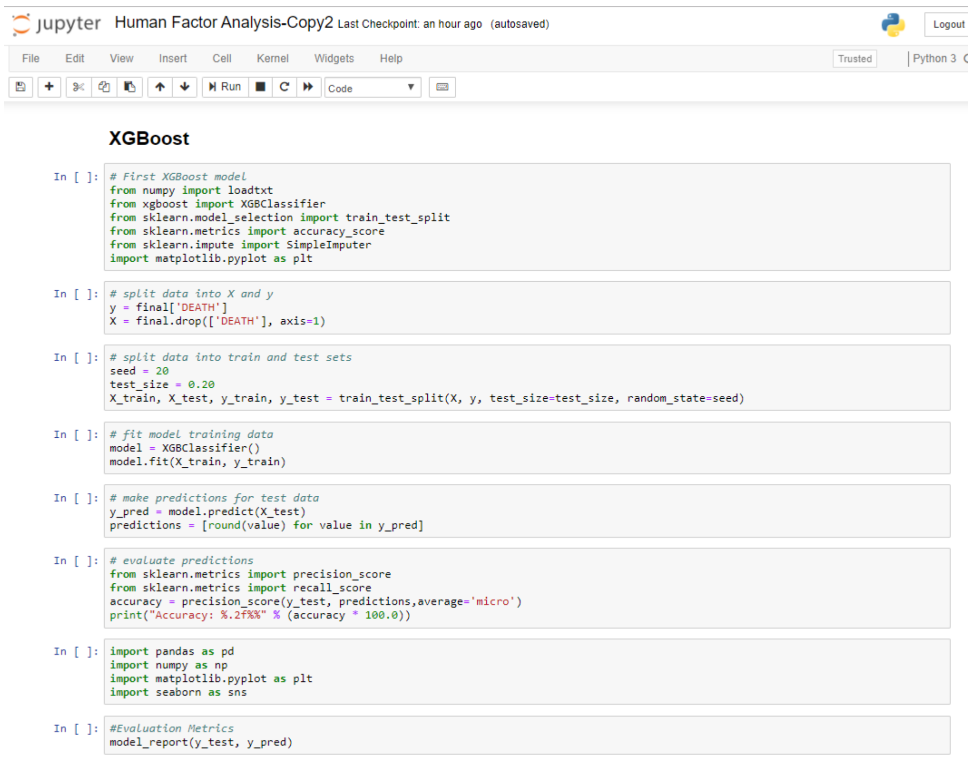
Figure 15: Experiment on Logistic Regression Stratified K-Fold and SMOTE

4.8 Experiment 6: XGBoost

It is one of the best performing models as the implementation was done using the ”XGBClassifier” function [8]. So were the SimpleImputer function being imported from sklearn.impute as it handles if any missing values, which in this study has been deal in the initial stages of this research. Prediction were rounded of using the round() of the values for the predictors. As observed in the Figure 16.

[7]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[8]https://xgboost.readthedocs.io/en/latest/python/python_api.html

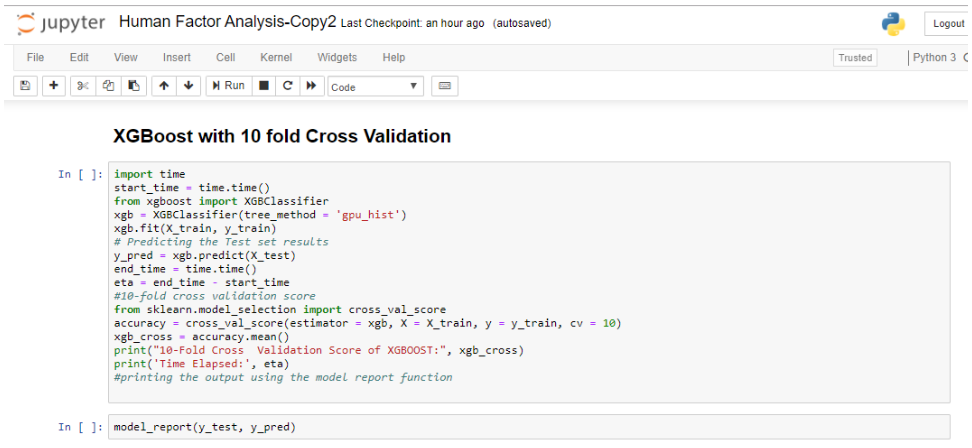

jupyter
Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

XGBoost

```
In [ ]: # First XGBoost model
        from numpy import loadtxt
        from xgboost import XGBClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn.impute import SimpleImputer
        import matplotlib.pyplot as plt

In [ ]: # split data into X and y
        y = final['DEATH']
        X = final.drop(['DEATH'], axis=1)

In [ ]: # split data into train and test sets
        seed = 20
        test_size = 0.20
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)

In [ ]: # fit model training data
        model = XGBClassifier()
        model.fit(X_train, y_train)

In [ ]: # make predictions for test data
        y_pred = model.predict(X_test)
        predictions = [round(value) for value in y_pred]

In [ ]: # evaluate predictions
        from sklearn.metrics import precision_score
        from sklearn.metrics import recall_score
        accuracy = precision_score(y_test, predictions,average='micro')
        print("Accuracy: %.2f%%" % (accuracy * 100.0))

In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

In [ ]: #Evaluation Metrics
        model_report(y_test, y_pred)
```

Figure 16: Experiment on XGBoost

The evaluation of CV in XGBoost is done using the function cross_val_score which is imported from scikit-learn. The folds are set to 10 and for experiment purpose it was set to 5 folds and 15 folds too. In which the best output was received with 10-folds. The function and model fit is seen in theFigure 17.

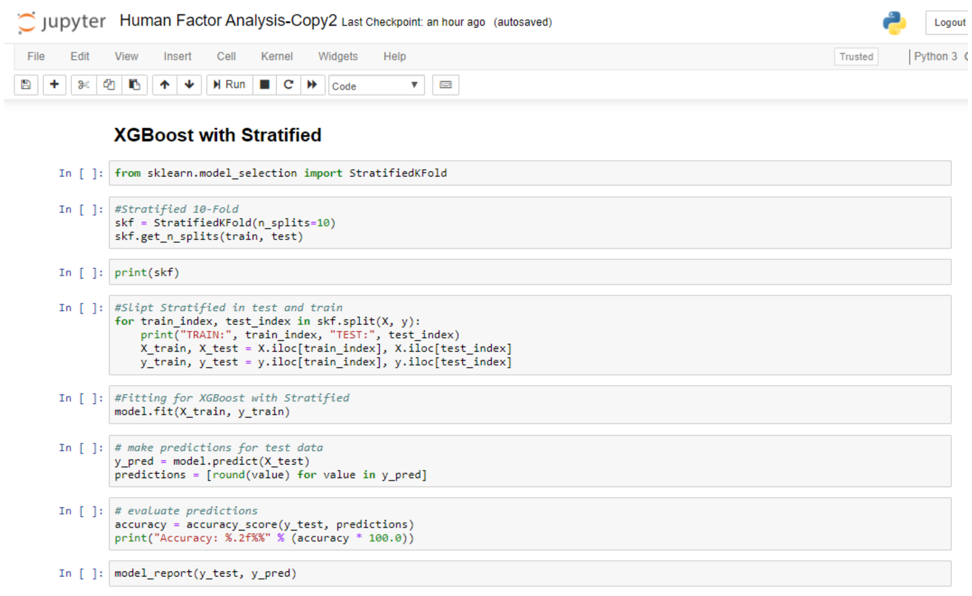

jupyter
Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

XGBoost with 10 fold Cross Validation

```
In [ ]: import time
        start_time = time.time()
        from xgboost import XGBClassifier
        xgb = XGBClassifier(tree_method = 'gpu_hist')
        xgb.fit(X_train, y_train)
        # Predicting the Test set results
        y_pred = xgb.predict(X_test)
        end_time = time.time()
        eta = end_time - start_time
        #10-fold cross validation score
        from sklearn.model_selection import cross_val_score
        accuracy = cross_val_score(estimator = xgb, X = X_train, y = y_train, cv = 10)
        xgb_cross = accuracy.mean()
        print("10-Fold Cross  Validation Score of XGBOOST:", xgb_cross)
        print('Time Elapsed:', eta)
        #printing the output using the model report function

In [ ]: model_report(y_test, y_pred)
```

Figure 17: Experiment on XGBoost with 10 Fold Cross Validation

A similar approach has been taken for the stratified as the other models to explore the data, using stratified k fold function with the splits of 10 as observed in the Figure 18 XGBoost is one of the best performing models using CV we managed to get a very accurate result. The data is spilt and the trained and tested.

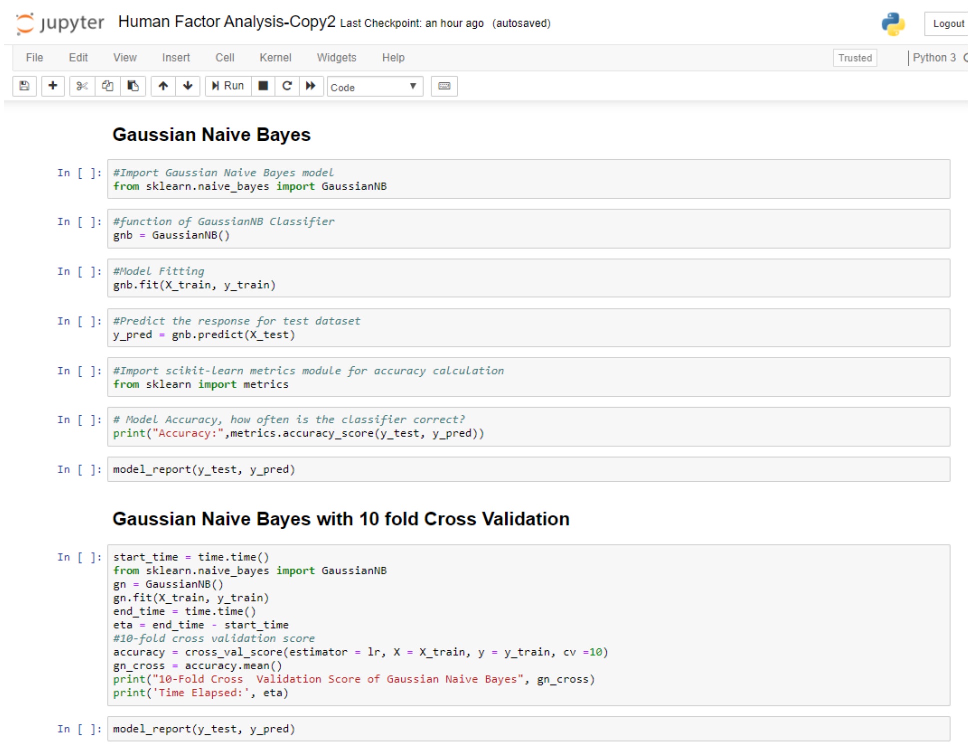

jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

XGBoost with Stratified

```
In [ ]: from sklearn.model_selection import StratifiedKFold

In [ ]: #Stratified 10-Fold
        skf = StratifiedKFold(n_splits=10)
        skf.get_n_splits(train, test)

In [ ]: print(skf)

In [ ]: #Slipt Stratified in test and train
        for train_index, test_index in skf.split(X, y):
            print("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

In [ ]: #Fitting for XGBoost with Stratified
        model.fit(X_train, y_train)

In [ ]: # make predictions for test data
        y_pred = model.predict(X_test)
        predictions = [round(value) for value in y_pred]

In [ ]: # evaluate predictions
        accuracy = accuracy_score(y_test, predictions)
        print("Accuracy: %.2f%%" % (accuracy * 100.0))

In [ ]: model_report(y_test, y_pred)
```

Figure 18: Experiment on XGBoost with Stratified K-Fold

4.9 Experiment 7: Gaussian Naïve Bayes


jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Gaussian Naive Bayes

```
In [ ]: #Import Gaussian Naive Bayes model
        from sklearn.naive_bayes import GaussianNB

In [ ]: #function of GaussianNB Classifier
        gnb = GaussianNB()

In [ ]: #Model Fitting
        gnb.fit(X_train, y_train)

In [ ]: #Predict the response for test dataset
        y_pred = gnb.predict(X_test)

In [ ]: #Import scikit-learn metrics module for accuracy calculation
        from sklearn import metrics

In [ ]: # Model Accuracy, how often is the classifier correct?
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

In [ ]: model_report(y_test, y_pred)
```

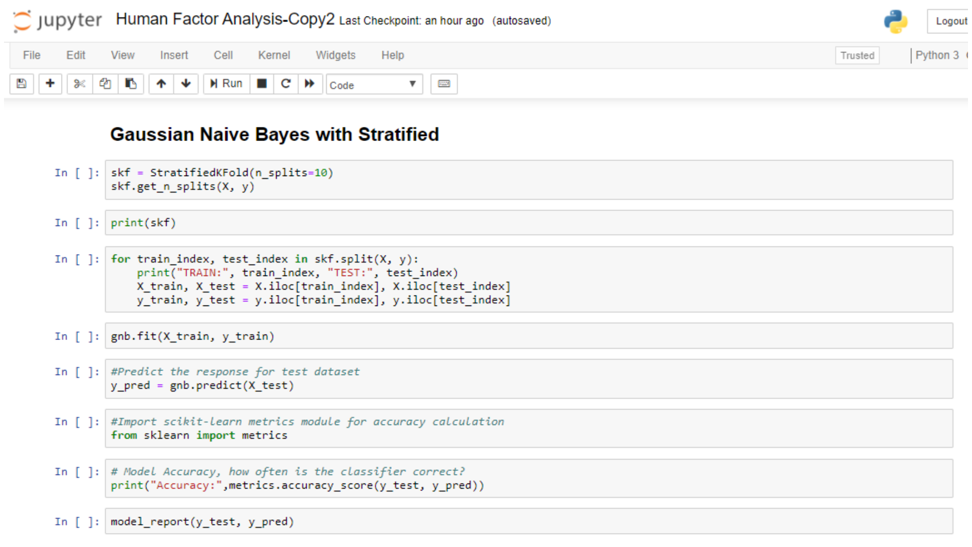
Gaussian Naive Bayes with 10 fold Cross Validation

```
In [ ]: start_time = time.time()
        from sklearn.naive_bayes import GaussianNB
        gn = GaussianNB()
        gn.fit(X_train, y_train)
        end_time = time.time()
        eta = end_time - start_time
        #10-fold cross validation score
        accuracy = cross_val_score(estimator = lr, X = X_train, y = y_train, cv =10)
        gn_cross = accuracy.mean()
        print("10-Fold Cross  Validation Score of Gaussian Naive Bayes", gn_cross)
        print('Time Elapsed:', eta)

In [ ]: model_report(y_test, y_pred)
```

Figure 19: Experiment on Gaussian Naïve Bayes and with Using 10-Fold Cross Validation

in Figure 19 the model is a linear classifier which is best fit for supervised learning. As our data is big it is very much the best consideration. Using the function "GaussianNB()"

function for predictions. The library is from sklearn. And also the 10-Fold CV is implemented in this research using the cross val score() and also the implementation of stratified fold Figure 20.


jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

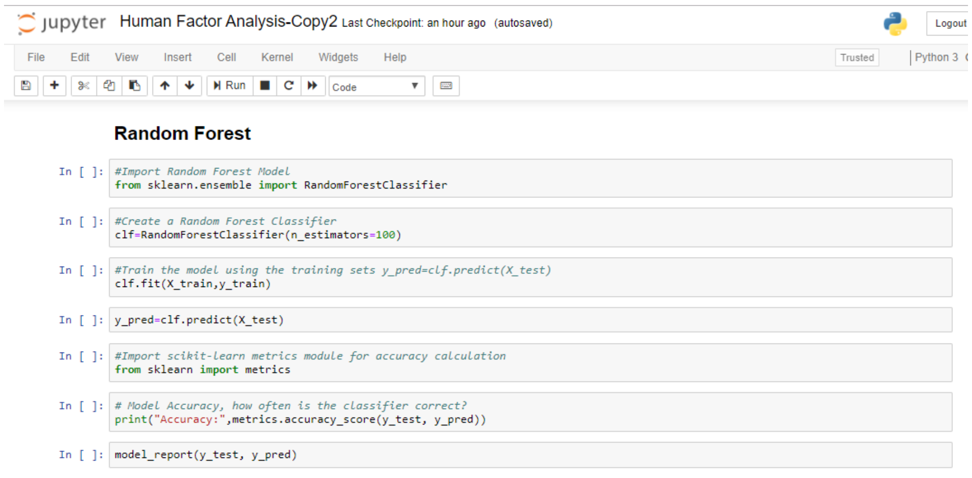
Gaussian Naive Bayes with Stratified

```
In [ ]: skf = StratifiedKFold(n_splits=10)
        skf.get_n_splits(X, y)

In [ ]: print(skf)

In [ ]: for train_index, test_index in skf.split(X, y):
            print("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

In [ ]: gnb.fit(X_train, y_train)

In [ ]: #Predict the response for test dataset
        y_pred = gnb.predict(X_test)

In [ ]: #Import scikit-learn metrics module for accuracy calculation
        from sklearn import metrics

In [ ]: # Model Accuracy, how often is the classifier correct?
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

In [ ]: model_report(y_test, y_pred)
```

Figure 20: Experiment on Gaussian Naïve Bayes and with Using Stratified K-Fold

4.10 Experiment 8: Random Forest

One of the model common and widely used models is the random forest as observed from the Figure 21 the model uses the ensemble approach which is used from the ”sklearn.ensemble” to get the ”RandomForestClassifier” function as the number of classifier decision trees and it would take an average in order to improve the accuracy and have control over the model being over-fit [9]. We have not defined any max depth as we wanted the model to learn on all the basis and not be restrictive in nature Burnett and Si (2017).


jupyter Human Factor Analysis-Copy2 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

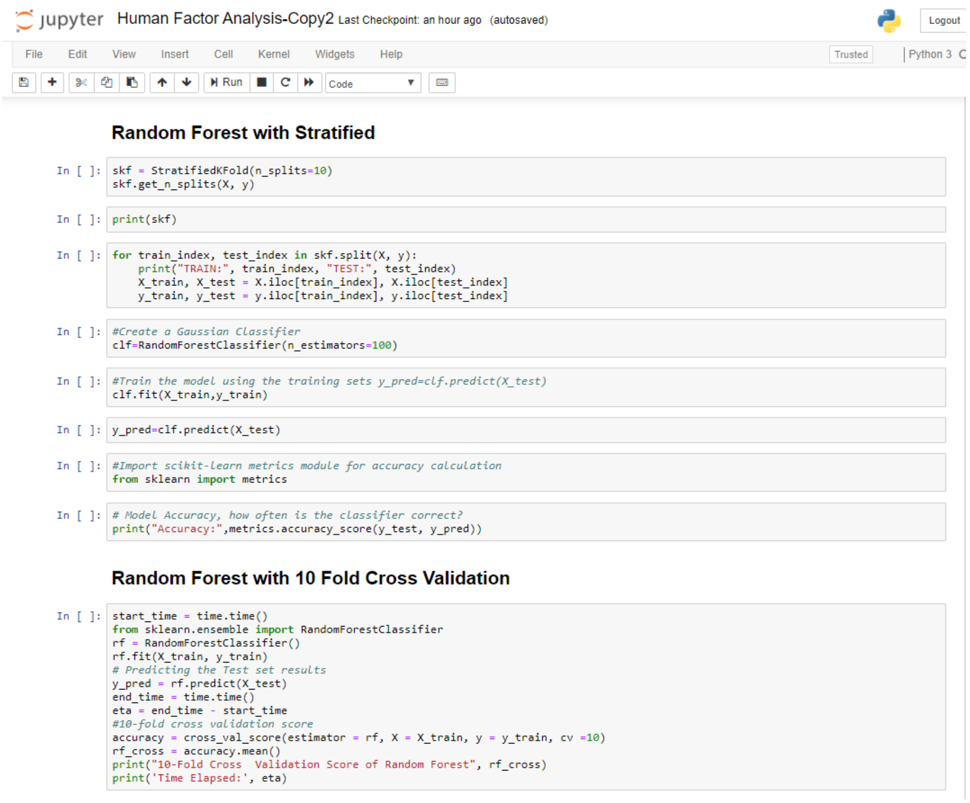
Random Forest

```
In [ ]: #Import Random Forest Model
        from sklearn.ensemble import RandomForestClassifier

In [ ]: #Create a Random Forest Classifier
        clf=RandomForestClassifier(n_estimators=100)

In [ ]: #Train the model using the training sets y_pred=clf.predict(X_test)
        clf.fit(X_train,y_train)

In [ ]: y_pred=clf.predict(X_test)

In [ ]: #Import scikit-learn metrics module for accuracy calculation
        from sklearn import metrics

In [ ]: # Model Accuracy, how often is the classifier correct?
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

In [ ]: model_report(y_test, y_pred)
```

Figure 21: Experiment on Random Forest

[9]https://scikit-learn.org/stable/modules/ensemble.html

As seen in the Figure 22 stratified fold would create a test set in way that each of the class would have same class distribution and would be close connection. It is invariant to the class Li (2014). Using both the approaches of fold i.e. 10-Fold CV and stratified fold.



Random Forest with Stratified

```
In [ ]: skf = StratifiedKFold(n_splits=10)
        skf.get_n_splits(X, y)

In [ ]: print(skf)

In [ ]: for train_index, test_index in skf.split(X, y):
            print("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

In [ ]: #Create a Gaussian Classifier
        clf=RandomForestClassifier(n_estimators=100)

In [ ]: #Train the model using the training sets y_pred=clf.predict(X_test)
        clf.fit(X_train,y_train)

In [ ]: y_pred=clf.predict(X_test)

In [ ]: #Import scikit-learn metrics module for accuracy calculation
        from sklearn import metrics

In [ ]: # Model Accuracy, how often is the classifier correct?
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Random Forest with 10 Fold Cross Validation

```
In [ ]: start_time = time.time()
        from sklearn.ensemble import RandomForestClassifier
        rf = RandomForestClassifier()
        rf.fit(X_train, y_train)
        # Predicting the Test set results
        y_pred = rf.predict(X_test)
        end_time = time.time()
        eta = end_time - start_time
        #10-fold cross validation score
        accuracy = cross_val_score(estimator = rf, X = X_train, y = y_train, cv =10)
        rf_cross = accuracy.mean()
        print("10-Fold Cross  Validation Score of Random Forest", rf_cross)
        print('Time Elapsed:', eta)
```

Figure 22: Experiment on Random Forest using Stratified and 10-Fold Cross Validation

References

Bazargan, M. and Guzhva, V. S. (2007). Factors contributing to fatalities in General Aviation accidents, *World Review of Intermodal Transportation Research* **1**(2): 170.

Bazargan, M. and Guzhva, V. S. (2011). Impact of gender , age and experience of pilots on general aviation accidents, *Accident Analysis and Prevention* **43**(3): 962–970.
URL: *http://dx.doi.org/10.1016/j.aap.2010.11.023*

Burnett, R. A. and Si, D. (2017). Prediction of Injuries and Fatalities in Aviation Accidents through Machine Learning, pp. 60–68.

Christopher, A. B. and Appavu, S. (2013). Data mining approaches for aircraft accidents prediction: An empirical study on Turkey airline, *2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology, ICE-CCN 2013* (Iceccn): 739–745.

Hofmann, M. (2019). Analysis of aviation accidents data, (October 2018).

Kharoufah, H., Murray, J., Baxter, G. and Wild, G. (2018). Progress in Aerospace Sciences A review of human factors causations in commercial air transport accidents and incidents : From to 2000 – 2016, **99**(March): 1–13.

Kumar, P., Gupta, S., Agarwal, M. and Singh, U. (2016). Categorization and standardization of accidental risk-criticality levels of human error to develop risk and safety management policy, *Safety Science* **85**: 88–98.
URL: *http://dx.doi.org/10.1016/j.ssci.2016.01.007*

Li, G. (2014). Pilot-Related Factors in Aircraft Crashes : A Review of Epidemiologic Studies, (November 1994).

Management, E., College, S. E. and Force, A. (2014). Study on the Aviation Accidents Due to Human Factors Based on Improved Support Vector Machine 2 The selection of key indicator based on Analytic Hierarchy Process (AHP), pp. 278–283.

Mathur, P., Khatri, S. K. and Sharma, M. (2017). Prediction of Aviation Accidents using Logistic Regression Model, pp. 1–4.

Zhang, X. and Mahadevan, S. (2019). Ensemble machine learning models for aviation incident risk prediction, *Decision Support Systems* **116**(October 2018): 48–63.
URL: *https://doi.org/10.1016/j.dss.2018.10.009*