CREDIT CARD FRAUD DETECTION USING

APPLIED DATA SCIENCE

PHASE-4: DEVELOPMENT PART 2

FEATURE ENGINEERING :

Feature engineering refers to manipulation addition,deletion, edition combination, mutation — of your data set to improve model training, leading to better performance and greater accuracy. Effective feature engineering is based on sound knowledge of the business problem and the available data sources.

SPLIT TRAIN AND TEST DATA :

Splitting the dataset into x and y

Y=df['class']

X=df.drop(['class'],axis=1) Checking some rows of x

X.head()

Checking some rows of y

y. head()

Splitting The Dataset Using Train Test Split :

X_train,x_test,y_train,y_test=train_test_split(x,y,random_state=100,test_size=0.0)

Preserve x_test and y_ test to evaluate

Checking The Spread Of Data Post Split :

Print(np sum(y)

Print(np sum(y_train))

Print(np sum(y_test))

CONFUSION MATRIX :

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

The following 4 are the basic terminology which will help us in determining the metrics we are looking for.

True Positives (TP): when the actual value is Positive and predicted is also Positive.

True Negatives (TN): when the actual value is Negative and prediction is also Negative.

False Positives (FP): When the actual is negative but prediction is Positive. Also known as the Type 1 error

False Negatives (FN): When the actual is Positive but the prediction is Negative. Also known as the Type 2 error.

Fraud detection using in business

Fraud detection is a set of processes and analyses that allow businesses to identify and prevent unauthorized financial activity. This can include fraudulent credit card transactions, identify theft, cyber hacking, insurance scams, and more.

Best practices for fraud detection and prevention

- Identify potential fraud threats. It all starts with a fraud risk profile.
- Implement artificial intelligence.
- Audit and monitor for fraud threats consistently.
- Educate your organization about your fraud detection system.
- Revisit your fraud profiles regularly.

Fraud detection importance in business

Fraud detection is essential for companies to safeguard their customers' transactions and accounts by detecting fraud before or as it happens. The FBI reports that in 2022, elder fraud victims in the US lost an average of $35,101 each, resulting in a total loss of over $3 billion.

Description of Fraud Risks Examples:

while fraud risks vary, examples include

- theft of assets,
- fraudulent disbursements,

- manipulation of expenses,
- inappropriate journal entries.

Asset misappropriation

Asset misappropriation is the most common type of employee fraud, and it's often the easiest to commit. Let's be clear: misappropriating assets means deliberately stealing from an employer.

Effective Strategies for Fraud Detection and Prevention in business

Statistical analysis involves using calculations and models to understand data, while AI methods include data mining, neural networks, machine learning, and pattern recognition to detect fraud patterns.

MODEL SELECTION :

Model selection is the process of selecting the best model from all the available models for a particular business problem on the basis of different criterions such as robustness and model complexity.

TECHNIQUES USED IN MODEL SELECTION:

- Logistic Regression
- KNN
- Decisions tree classifier
- Random forest
- Xgboost
- SVM

LOGISTIC REGRESSION:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

PROGRAM :

Created a common function to fit and predict on a logistic Regression model for bothdefbuildAndRunLogisticModels(dfResults,Methodology,X_train,y_train,x_test, y_test):

L1 and L2

#Logistic Regression

From sklearn import linear_model

From sklearn.model_selection import KFold

Num C = list(np.power(10.8, np.arange(-18, 10)))

Cv_num= KFold (n_splits 10, shuffle True, random_state=42)

searchCV_12 = linear_model.Logistic RegressionCV(

Cs num_C

Penalty=”12”

,scoring=’roc auc

Cv=cv_num

Random_state=42

Max_iter-10000

,fit intercept-True

,solver newton-cg

Tol-10

searchCV_11-linear_model.Logistic RegressionCV(Cs=num CPenalty=”11”,scoring=roc auc,random_state=42,max_iter-10000,fit intercept=True,solver newton-cg”,tol-10

searchCV 11.fit(x_train, y_train)

searchCV 12.fit(x_train, y_train)

print (“Max auc roc for 12:, searchCV_12.scores [1].mean(axis=0).max())

print(“Parameters for 11 regularisations”)

```
print(searchCV_11.coef)

print(searchCV 11.intercept_)

print(searchCV 11.scores_)

print("Parameters for 12 regularisations")

print(searchCV_12.coef)

print(searchCV_12.intercept_)

print(searchCV 12.scores_)

#find predicted vallues

Y_pred 11

searchCV_11.predict(X_test)

v pred 12 = searchCV 12.predict(x test)

#Find predicted probabilities

Y_pred_probs 11 = searchCV_11.predict_proba(X_test)[:,1]

Y_pred_probs 12 = searchCV_12.predict_proba (X_test) [:,1]

#Accuaracy of L2/L1 models

Accuracy_12 = metrics.accuracy_score (y_preday_pred_12, y_true=y_test)
Accuracy_11 = metrics.accuracy_score(y_preday_pred_11, y_true=y_test)

Print("Accuarcy of Logistic model with 12 regularisation :
(0)".format(Accuracy_12))

Print("Confusion Matrix")

Plot confusion matrix(y_test, y_pred_12)

Print("classification Report")

Print(classification_report(y_test, y_pred_12))

Print("Accuarcy of Logistic model with 11 regularisation €".format(Accuracy_11))

Print("Confusion Matrix")
```

Plot confusion matrix(y_test, y_pred_11)

Print(“classification Report”)

Print(classification_report(y_test, y_pred_11))

12_roc_value = roc_auc_score (y_test, y_pred_probs_12)

Print(“12 roc_value: € format(12_roc_value)) for, tpr, thresholds metrics.roc_curve (y_test, y_pred_probs_12) print(“12 threshold: (e}”.format(threshold))

Threshold thresholds [np.argmax(tpr-fpr)]

Roc auc metrics. Auc (fpr, tpr)

Print(“ROC for the test dataset”, :.1%).format(roc_auc))

Plt.plot(fpr,tpr,label=”Test, auc=”+str(roc_auc))

Plt.legend (loc=4)

Plt.show()

Df Results = df Results. Append(pd.DataFrame({ “Methodology: Methodology, ‘Model”: “Logistic Regression with 12 Regularisation,

11 roc_value roc_auc_score(y_test, y_pred_probs_11)

Print(“11 roc_value: €” .format(11_roc_value))

Fpr, tpr, thresholds metrics.roc_curve (y_test, y_pred probs_11)

Print(“11 threshold: €”.format(threshold))

Threshold thresholds[np.argmax(tpr-fpr)]

Roc auc metrics.auc(fpr, tpr)

Print(“ROC for the test dataset, f.1%).format(roc__auc))

Plt.plot(fpr,tpr,label=”Test, auce”+str(roc_auc))

Plt.legend(loc=4)

Plt.show()

```
Df Results = df_Results.append(pd.DataFrame({ “Methodology: Methodology,
‘Model’: ‘Logistic Regression with L2 Regularisation”,”

11_roc_value roc_auc_score (y_test, y pred_probs_11)

Print(“11 roc_value: €”.format(11_roc_value))

 Fpr, tpr, thresholds metrics.roc_curve (y_test, y_pred_probs_11)

Print(“11 threshold: € format(threshold))

Threshold thresholds [npargmax(tpr-fpr)]

Roc_auc metrics.auc (fpr, tpr)

Print(“ROC for the test dataset”, :.1%).format(roc_auc))

Plt.plot(for, tor, label=”Test, aucaestr(roc_auc))

Plt.legend(loc=4)

Plt.show()

Df Results = df_Results.append(pd.DataFrame({ “Methodology:
Methodology,‘Model”: “Logistic Regression with L1 Regularisation”,”

Return df_Results
```

KNN :

The abbreviation KNN stands for “K-Nearest Neighbour”. It is a supervised machine learning algorithm. The algorithm can be used to solve both classification and regression problem statements. The number of nearest neighbours to a new unknown variable that has to be predicted or classified is denoted by the symbol “K”.

PROGRAM :

```
#Created a common function to fit and predict on a KNN model

Def buildAndRunkodels (df_Results, Methodology, X_train,y train, x_test, y_test );

#create KNN model and fit the model with train dataset
```

```
Knn NeighborsClassifier(n_neighbors 5,n jobs-16)
Knn.fit(x train,y_train)
Score knn.score (X_test,y_test)
Print(“model score”)
Print(score)
#Accuracy
Y_pred knn.predict(X_test)
KNN_Accuracy metrics.accuracy score(y_pred y_pred, y_true-y_test)
Print(“Confusion Matrix”)
Plot_confusion_matrix(y_test, y_pred)
Print(“classification Report”)
Print(classification_report(y_test, y_pred))
Knn_probs = knn.predict proba(X_test) [: , 1]
#Calculate roc auc
Knn_roc_value roc_auc_score (y_test, knn_probs)
Print(“Kroc value: (0)”.format(knn_roc_value))
Fpr, tpr, thresholds metrics.roc_curvely_test, knn_probs)
Threshold thresholds[np.argmax(tpr-for)]
Print(“KN threshold: (0)”.format(threshold))
Roc auc metrics.auc (for, tpr)
Print(“ROC for the test dataset”, “(1.1%).format(roc_auc))
Plt.plot(for, tpr, label=”Test, auc+str(roc_suc))
Plt.legend(loc-4)
Plt.show()
```

Df Results = df Results.append(pd.DataFrame({ “Methodology”: Methodology, “Model’s : “KNN”, “Accuracy: score, ‘roc_value’: knn_roc_value ,”threshold” : threshold } index = [0])

Return df Results

Decision Tree Classifier :

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree- structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

PROGRAM :

Def buildAndRunTreeModels(df_Results, Methodology, X_train,y_train, X_test, y_test):

Criteria [‘gini’, ‘entropy’]

Scores = {}

For c in criteria:

Dt DecisionTreeClassifier(criterion = c, random_state=42)

Dt.fit(X_train, y_train)

Y_pred = dt.predict(X_test)

Test_score dt.score(X_test, y_test)

Tree_preds dt.predict_proba(X_test)[:, 1]

Tree_roc_value = roc_auc_score(y_test, tree_preds)

Scores test_score

Print(c + score: {0}”.format(test_score))

Print(“Confusion Matrix”)

Plot_confusion_matrix(y_test, y_pred)

```
Print("classification Report")

Print(classification_report(y_test, y_pred))

Print(c+" tree_roc_value: {e}".format(tree_roc_value))

 Fpr, tpr, thresholds metrics.roc_curve(y_test, tree_preds)

Threshold thresholds [np.argmax(tpr-fpr)]

Print("Tree threshold: {0}".format(threshold))

Roc_auc metrics.auc(fpr, tpr)

Print("ROC for the test dataset", (:.1%)'.format(roc_auc))

Plt.plot(fpr, tpr, label="Test, auc="+str(roc_auc))

Plt.legend(loc=4)

Plt.show()

Df_Results= df_Results.append(pd.DataFrame({'Methodology: Methodology,
'Model': 'Tree Model with (0) criteria'.format©, 'Accuracy ' : test_score,
'roc_value' : tree_roc_value)

Return df Results
```

Random Forest :

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

PROGRAM :

```
Def buildAndRunRandomForest Models(df desults, Methodology, X_train,y_train,
X_test, y_test):

RF_model RandomForestClassifier(n_estimators=100,Bootstrap =  True,
Max_features 'sqrt', random_state=42)

# Fit on training data
```

```
RF_model.fit(X_train, y_train)
RF_test_score
 RF_model.score(X_test, y_test)
RF_model.predict(X_test)
Print('Model Accuracy: (0}'.format(RF_test_score))
# Actual class predictions
Rf_predictions = RFTmodel.predict(X_test)
Print("Confusion Matrix")
Plot_confusion_matrix(y_test, rf_predictions)
Print(classification_report(y_test, rf_predictions))
Print("classification Report")
#Probabilities for each class
Rf_probs RF_model.predict_proba(X_test)[:, 1]
# Calculate roc auc
Roc_value roc_auc_score(y_test, rf_probs)
Print("Random Forest roc_value: (0)" .format(roc_value))
 Fpr, tpr, thresholds metrics.roc_curve(y_test, rf_probs)
Threshold thresholds[np.argmax(tpr-fpr)]
 Print("Random Forest threshold: (0)".format(threshold))
Roc_auc metrics.auc(fpr, tpr)
Print("ROC for the test dataset", (:.1%)'.format(roc_auc))
 Plt.plot(fpr, tor, label="Test, auc="+str(roc_auc))
Plt.legend(loc=4)
Plt.show()
```

Df_Results= df_Results.append(pd.DataFrame({‘Methodology’: Methodology, Model’: ‘Random Forest’, ‘Accuracy: RF_test_score, roc, ‘roc_value’: roc_value, ‘threshold’: threshold), index=[0])

Return df_Results

XGBoost :

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements Machine Learning algorithms under the Gradient Boosting framework. It provides a parallel tree boosting to solve many data science problems in a fast and accurate way.

PROGRAM :

#Created a common function to fit and predict on a x5Boost model

Def buildAndRunXGBoost Models (df Results, Methodology,x_train,y train, X_test,y_test):

#Evaluate XGboost model

XOBmodel XGBClassifier(random_state=42)

XOBmodel.fit(x train, y_train)

Y_pred XG8model.predict(X_test)

XGB test score XG8model.score (X_test, y_test)

Print(“Model Accuracy: ().format(XG8_test_score))

Print(“Confusion Matrix”)

Plot confusion matrix(y_test, y_pred)

Print(“classification Report”)

Print(classification_report(y_test, y_pred))

#Probabilities for each class

XGB_probs XGBmodel.predict_proba(X_test) [1, 1]

#Calculate roc auc

```
XGB_roc_value = roc_auc_scorely test, XGB_probs)

Print(“xoboost roc value: (0) format(XGB roc_value))

Fpr, tpr, thresholds metrics.roc curvely test, X08_probs)

Threshold thresholds[np.argmax(tor-for)]

Print(“XGBoost threshold: (0)”.format(threshold))

 Roc auc metrics.auc(fpr, tpr)

Print(“ROC for the test dataset, (:.1%) format(roc_auc))

 Plt.plot(for, tor,label=”Test, auce+str(roc auc))

Plt.legend(loc-4)

Plt.show()

Df Results df_Results.append(pd.DataFrame({‘Methodology:Methodology ,
“Model” : “XoBoost”, ‘Accuracy:XG8_test_score, ‘Roc_value’ : XG8_roc_value,
’threshold’ : threshold }index = [0] ,ignore_index = True )

Return df Results
```

SVM :

“Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

PROGRAM :

```
#Created a common function to fit and predict on a so

Def buildAndRundels (of Results, Methodology, x trait,y train, x_test, test):

# Evaluate S model with signold hernel model

 From sklearn.sve import SVC
```

```
From sklearn.netrics import accuracy score

From sklearn.metrics Import roc_ac_score

Clf SVC(kernel sigmoid, random_state=42)

Cif.fit(x train,y_train)

Y pred Self.predict(X_test) SV Score accuracy scorely test,y pred SVM)

Print("accuracy score €".format(SVM_Score)) print("Confusion Matrix")

Plot confusion matrisly test, y pred SVM)

Print("classification Report")

 Print(classification reportly test, y_pred SVM))

#Run classifier

Classifier = SVC(kernel signold, probability=True)

Svm_probs classifier.fit(x train, y_train).predict proba(X_test)[5, 1]

#Calculate rec auc

Rec value roc auc scorely test, sim probs)

Print("SVM rac_value: (0)".format(roc_value))

Fpr, tpr, thresholds metrics.roc curvely test, svm probs)

 Threshold thresholds[np.argmax(tor-for)]

Print("svm threshold: (0)".format(threshold))

Roc_auc metrics. Auc(for, tpr)

Print("SVM threshold: €".format(threshold)) print("ROC for the test dataset", :.1%)
format(roc_auc))

Plt.plot(fpr, tpr, label="Test, auca"+str(roc_auc))

Plt.legend (loc=4)

Plt.show()
```

Df Results = df_Results.append(pd.DataFrame({‘Methodology:Methodology, ‘ Model’: ‘SVM’,’Accuracy’:SVM_Score, ‘roc_value’:roc_value , ‘threshold’ : threshold } index = [0] , ignore_index = True)

Return df Results

Model evaluation:

Model evaluation is the process of figuring out how well the model performs at guessing something.

This evaluation is usually handled with a test dataset. What we do is is we take some proportion of the total data, set it aside so it never gets involved with the model training and then we pass that data through the model once we’ve trained it.

We have the known outcomes of that data that’s been set aside. We take the independent variables from that test dataset, and feed it through the model. The model should then spit out a bunch of guesses for the dependent variable.

We can then compare the guesses against the known dependent variable values. Depending on how the actual outcomes and the guessed outcomes match up, we can come up with all sorts of measures of model performance.

Accuracy

Accuracy is a simple and common measure for whether or not predictions were correct compared to known outcomes.Oftentimes though, it’s not sufficient.

Let’s say we know that 90% of the population does not snore. We could easily nail 90% accuracy on average without a model by simply saying everyone does not snore. Full stop. We’ll get it wrong 10% of the time.

Using confusion matrix:

A confusion matrix is a way to compare known outcomes and guessed outcomes.

If it’s a binary classifier where there are two possible guesses (True/False, Yes/No, etc.), then the matrix ends up looking like a 2x2 grid. One axis represents the true outcome, and another axis represents the guessed or predicted outcome.

The combinations include:

True Positive – the things that are true, that were predicted true

True Negative – false, predicted false

False Positive – actually false, but predicted true

False Negative – actually true, predicted false

Usually each box of the confusion matrix gets a count, percentage, or proportion reflecting the outcomes of the different predictions from the model.

Precision

Precision is the ratio of the true positives over the sum of the true positive and false positive.You can think of this as of all the things predicted to be true, how many were actually true?

It's a way of assessing how reliable the true guesses are.

Recall

Recall is the ratio of the true positive divided by the sum of the true positive and the false positive, or _everything that is true.

Think of recall as how many of the true cases were actually found?

Recall is especially important where the cost of missing a true case is high. For instance, recall is especially important in disease diagnosis, where failing to catch a disease can have huge negative impacts on a patient.

F1 Score

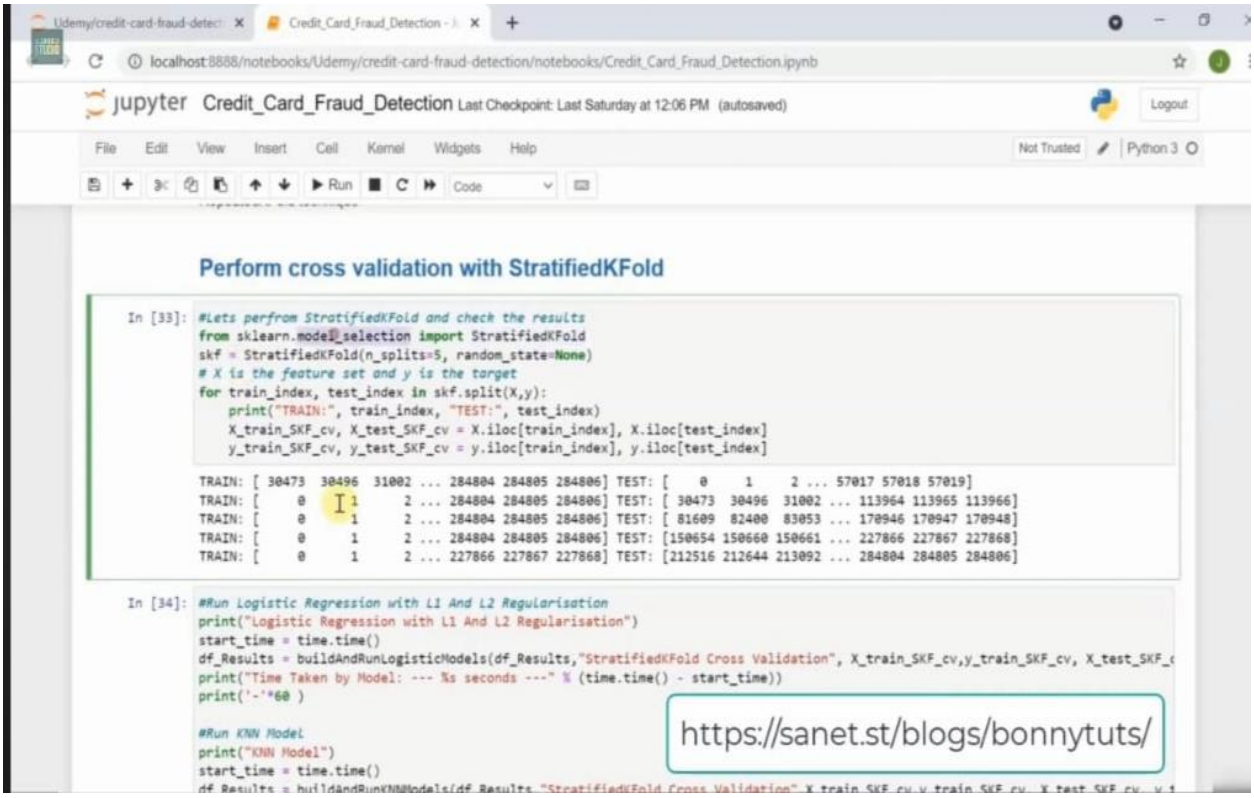
This is a unified measure that combines both the precision and recall.A higher F1 score is generally deemed better.This is a handy metric to rely on when comparing multiple different models.

Stratifiedkfold

In machine learning, When we want to train our ML model we split our entire dataset into training_set and test_set using train_test_split() class present in sklearn. Then we train our model on training_set and test our model on test_set. The problems that we are going to face in this method are:

Whenever we change the random_state parameter present in train_test_split(), We get different accuracy for different random_state and hence we can't exactly point out the accuracy for our model.

The train_test_split() splits the dataset into training_test and test_set by random sampling. But stratified sampling is performed.



Repeated K-fold Cross Validation

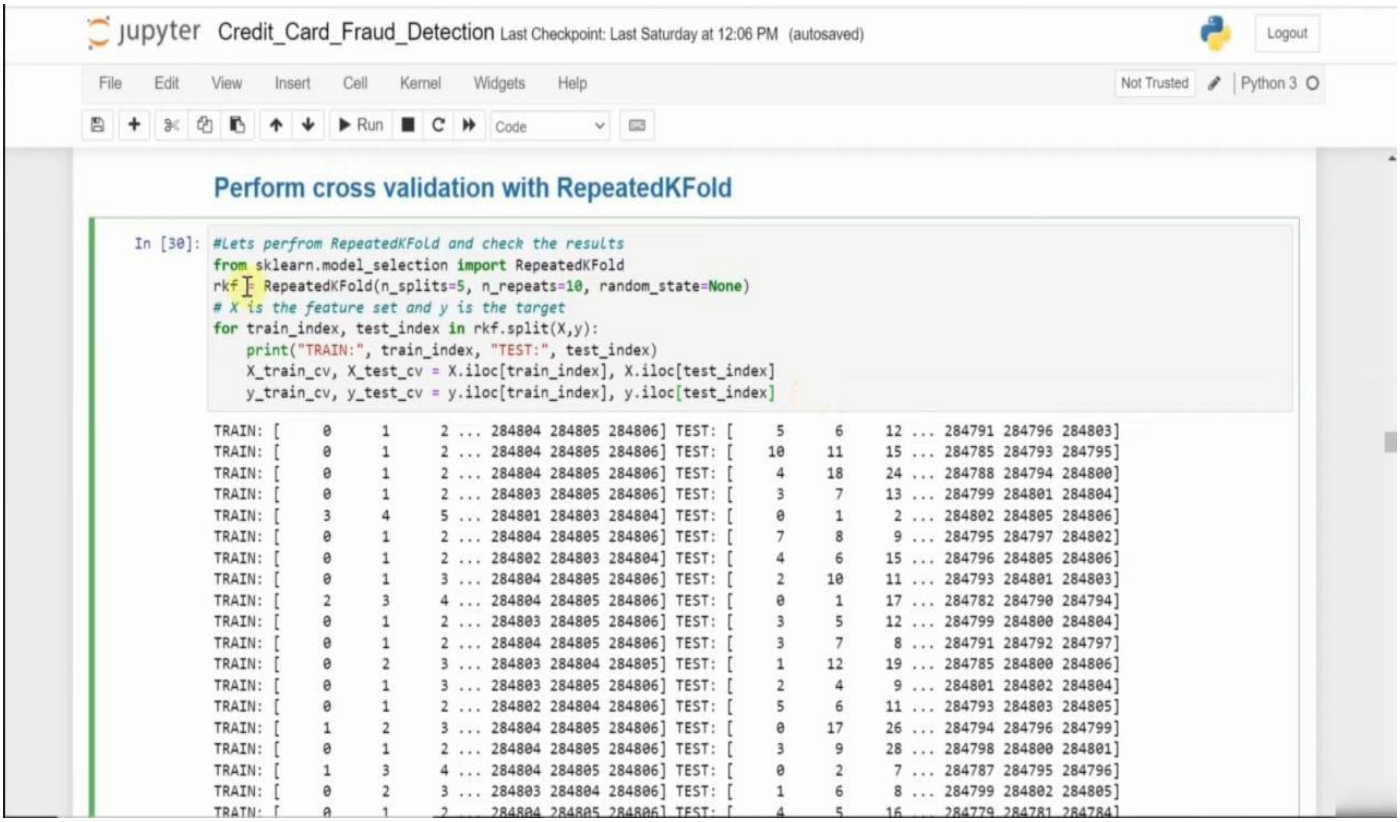
Repeated K-fold is the most preferred Cross-Validation technique for both classification and regression Machine Learning models. Shuffling and random sampling of the data set multiple times is the core procedure of repeated K-fold algorithm and it results in making a robust model as it covers the maximum training and testing operations.

Steps involved in the repeated K-fold cross-validation:

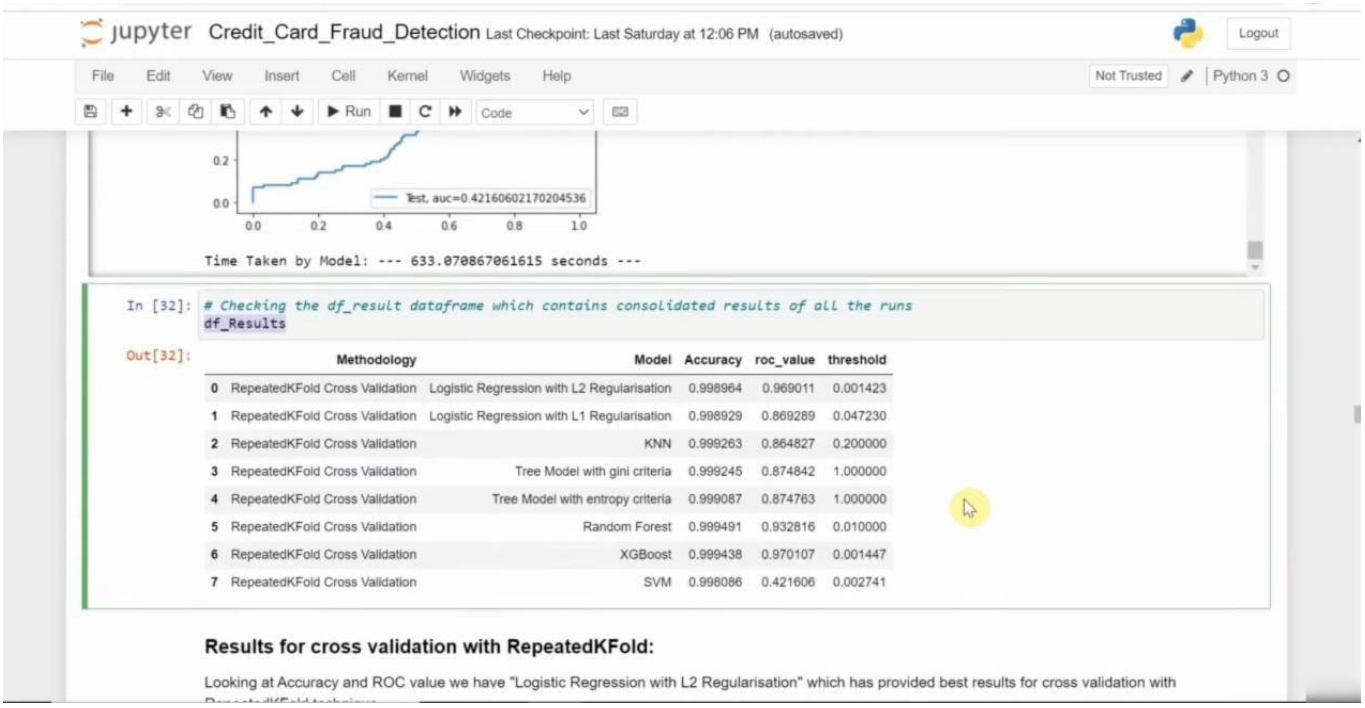
1. Split the data set into K subsets randomly

2.For each one of the developed subsets of data points

- Treat that subset as the validation set
- Use all the rest subsets for training purposes
- Training of the model and evaluate it on the validation set or test set
- Calculate prediction error.



By validating all the algorithms mentioned above.we have created a table which includes methodology,model, accuracy,roc_value, threshold value we can say that among these models logistic regression with L2 regularization is more accurate and precise and this model shows and perfect for our project credit card fraud deduction



	Methodology	Model	Accuracy	roc_value	threshold
0	RepeatedKFold Cross Validation	Logistic Regression with L2 Regularisation	0.998964	0.969011	0.001423
1	RepeatedKFold Cross Validation	Logistic Regression with L1 Regularisation	0.998929	0.869289	0.047230
2	RepeatedKFold Cross Validation	KNN	0.999263	0.864827	0.200000
3	RepeatedKFold Cross Validation	Tree Model with gini criteria	0.999245	0.874842	1.000000
4	RepeatedKFold Cross Validation	Tree Model with entropy criteria	0.999087	0.874763	1.000000
5	RepeatedKFold Cross Validation	Random Forest	0.999491	0.932816	0.010000
6	RepeatedKFold Cross Validation	XGBoost	0.999438	0.970107	0.001447
7	RepeatedKFold Cross Validation	SVM	0.998086	0.421606	0.002741