Model Traning File and Testing File

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Online Payments Fraud Detection</title>
  <style>
    body {
      font-family: sans-serif;
      background-color: #f0f0f0;
      padding: 20px;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      background-image: url(""C:\Users\Skamalesh\Downloads\OIP (10).jpg"");position: full;
    }
    .container {
      max-width: 400px;
    }
    .top-buttons {
      position: absolute;
      top: 20px;
      right: 20px;
    }
    .top-buttons button {
      margin-left: 10px;
    }
    .login-box, .transaction-box {
      background-color: #fff;
      padding: 30px;
      border: 1px solid #ddd;
      border-radius: 4px;
      margin-bottom: 20px;
    }
    h1, h2 {
      margin-bottom: 20px;
      text-align: center;
    }
    label {
      display: block;
      margin-bottom: 10px;
    }
    input[type="text"],
    input[type="password"],
    input[type="number"],
    input[type="date"],
    input[type="time"],
    color: white;
    padding: 10px 20px;
```

```
/* Modal styles */
    .modal-overlay {
      position:  fixed;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background-color: rgba(0, 0, 0, 0.5);
      display: none;
      justify-content: center;
      align-items: center;
      z-index: 9999;
    }
    .modal-content {
      background-color: #fff;
      padding: 20px;
      border-radius: 4px;
      text-align: center;
      animation-name: bounceIn;
      animation-duration: 1s;
    }
    @keyframes bounceIn {
      0% {
        transform: scale(0);
      }
      50% {
        transform: scale(1.1);
      }
      100% {
        transform: scale(1);
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="top-buttons">
      <button onclick="location.href='#';">Home</button>
      <button onclick="location.href='#';">Contact Us</button>
    </div>

    <div class="login-box">
      <h1>Online Payments Fraud Detection</h1>
      <form id="loginForm">
        <input type="text" name="username" placeholder="Username" value="admin" required>
        <input type="password" name="password" placeholder="Password" value="Vijay" required>
        <br>
        <button type="submit">Login</button>
      </form>
      <!-- Forgot password link -->
```

```
<select id="transactionType" name="transactionType" required>
        <option value="">Select Type</option>
        <option value="online">Online</option>
        <option value="offline">Offline</option>
       </select>
       <label for="transactionTime">Time of Transaction:</label>
       <input type="time" id="transactionTime" name="transactionTime" required>

       <label for="transactionDate">Date of Transaction:</label>
       <input type="date" id="transactionDate" name="transactionDate" required>

       <label for="balanceBefore">Balance Before the Transaction:</label>
       <input type="number" id="balanceBefore" name="balanceBefore" required>

       <label for="balanceAfter">Balance After the Transaction:</label>
       <input type="number" id="balanceAfter" name="balanceAfter" required>

       <label for="currentBalance">Current Balance:</label>
       <input type="number" id="currentBalance" name="currentBalance" required>

       <button type="submit">Submit</button>
     </form>
   </div>
 </div>

 <!-- Fraud Detected modal -->
 <div class="modal-overlay" id="fraudModal">
   <div class="modal-content">
     <h2>Warning: Fraud Detected!</h2>
   </div>
 </div>

 <script>
   document.getElementById('loginForm').addEventListener('submit', function(event) {
     event.preventDefault(); // Prevent form submission
     var username = this.username.value;
     var password = this.password.value;
     // Perform login validation here
     // For demo purpose, let's assume the login is successful
     if (username === 'admin' && password === 'reddy') {
       document.querySelector('.login-box').style.display = 'none'; // Hide login box
       document.querySelector('.transaction-box').style.display = 'block'; // Show transaction details box
     } else {
       alert('Invalid username or password. Please try again.');
     }
   });

   document.getElementById('transactionForm').addEventListener('submit', function(event) {
     event.preventDefault(); // Prevent form submission
     // Process transaction details here
     // For demo purpose, let's assume fraud is detected
```

```
// Implement your forgot password functionality here
    alert('Forgot password feature is not implemented yet.');
  });
 </script>
</body>
</html>
```

Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Random Forest Classifier

A function named RandomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

1.Random Forest classifier¶

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)

y_test_predict1=rfc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict1)
test_accuracy
```

```
0.9958847736625515
```

```
y_train_predict1=rfc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict1)
train_accuracy
```

```
1.0
```

```
pd.crosstab(y_test,y_test_predict1)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	232	2
is not Fraud	0	252

```
print(classification_report(y_test,y_test_predict1))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      1.00       234
is not Fraud       0.99      1.00      1.00       252

    accuracy                           1.00       486
   macro avg       1.00      1.00      1.00       486
weighted avg       1.00      1.00      1.00       486
```

Decision Tree Classifier

A function named Decisiontree is created and train and test data are passed as the parameters. Inside the function, the DecisiontreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train, y_train)

y_test_predict2=dtc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict2)
test_accuracy
```

```
0.9917695473251029
```

```
y_train_predict2=dtc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict2)
train_accuracy
```

```
1.0
```

```
pd.crosstab(y_test,y_test_predict2)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	231	3
is not Fraud	1	251

```
print(classification_report(y_test,y_test_predict2))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      0.99       234
is not Fraud       0.99      1.00      0.99       252

    accuracy                           0.99       486
   macro avg       0.99      0.99      0.99       486
weighted avg       0.99      0.99      0.99       486
```

ExtraTrees Classifier

A function named ExtraTree is created and train and test data are passed as the parameters. Inside the function, ExtraTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train,y_train)

y_test_predict3=etc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict3)
test_accuracy
```

```
0.9938271604938271
```

```
y_train_predict3=etc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict3)
train_accuracy
```

```
1.0
```

```
pd.crosstab(y_test,y_test_predict3)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	231	3
is not Fraud	0	252

```
print(classification_report(y_test,y_test_predict3))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      0.99       234
is not Fraud       0.99      1.00      0.99       252

    accuracy                           0.99       486
   macro avg       0.99      0.99      0.99       486
weighted avg       0.99      0.99      0.99       486
```

Support Vector Machine Classifier

A function named SupportVector is created and train and test data are passed as the parameters. Inside the function, the SupportVectorClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.7901234567901234
```

```
y_train_predict4=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict4)
train_accuracy
```

```
0.8009259259259259
```

```
pd.crosstab(y_test,y_test_predict4)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	132	102
is not Fraud	0	252

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_test_predict4))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.56      0.72       234
is not Fraud       0.71      1.00      0.83       252

    accuracy                           0.79       486
   macro avg       0.86      0.78      0.78       486
weighted avg       0.85      0.79      0.78       486
```

```
df.columns
```

```
Index(['step', 'type', 'amount', 'oldbalanceOrg', 'newbalanceOrig',
       'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder

la = LabelEncoder()
y_train1 = la.fit_transform(y_train)
```

```
y_test1=la.transform(y_test)
```

preprocessing class of sklearn. LabelEncoder[source] 0 to n classes-1 as the range for the target labels to be encoded. Instead of encoding the input X, the target values, i.e. y, should be encoded using this transformer.

```
y_test1=la.transform(y_test)
```

```
y_test1
```

```
array([0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1])
```

```
y_train1
```

```
array([0, 1, 0, ..., 1, 1, 0])
```

Xgboost Classifier

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, the xgboostClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

```
import xgboost as xgb
xgb1 = xgb.XGBClassifier()
xgb1.fit(x_train, y_train1)

y_test_predict5=xgb1.predict(x_test)
test_accuracy=accuracy_score(y_test1,y_test_predict5)
test_accuracy
```

```
0.9979423868312757
```

```
y_train_predict5=xgb1.predict(x_train)
train_accuracy=accuracy_score(y_train1,y_train_predict5)
train_accuracy
```

```
1.0
```

```
pd.crosstab(y_test1,y_test_predict5)
```

col_0	0	1
row_0		
0	233	1
1	0	252

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test1,y_test_predict5))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       234
           1       1.00      1.00      1.00       252

    accuracy                           1.00       486
   macro avg       1.00      1.00      1.00       486
weighted avg       1.00      1.00      1.00       486
```

Compare The Models

For comparing the above four models, the compareModel function is defined.

After calling the function, the results of models are displayed as output. From the five models, the svc is performing well. From the below image, We can see the accuracy of the model is 79% accuracy. .

Compare Models

```
def compareModel():
    print("train accuracy for rfc",accuracy_score(y_train_predict1,y_train))
    print("test accuracy for rfc",accuracy_score(y_test_predict1,y_test))
    print("train accuracy for dtc",accuracy_score(y_train_predict2,y_train))
    print("test accuracy for dtc",accuracy_score(y_test_predict2,y_test))
    print("train accuracy for etc",accuracy_score(y_train_predict3,y_train))
    print("test accuracy for etc",accuracy_score(y_test_predict3,y_test))
    print("train accuracy for svc",accuracy_score(y_train_predict4,y_train))
    print("test accuracy for svcc",accuracy_score(y_test_predict4,y_test))
    print("train accuracy for xgb1",accuracy_score(y_train_predict5,y_train1))
    print("test accuracy for xgb1",accuracy_score(y_test_predict5,y_test1))
```

```
compareModel()
```

```
train accuracy for rfc 1.0
test accuracy for rfc 0.9958847736625515
train accuracy for dtc 1.0
test accuracy for dtc 0.9917695473251029
train accuracy for etc 1.0
test accuracy for etc 0.9938271604938271
train accuracy for svc 0.8009259259259259
test accuracy for svcc 0.7901234567901234
train accuracy for xgb1 1.0
test accuracy for xgb1 0.9979423868312757
```

Evaluating Performance Of The Model And Saving The Model

From sklearn, accuracy_score is used to evaluate the score of the model. On the parameters, we have given svc (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model is svc by pickle.dump().

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.7901234567901234
```

```
y_train_predict4=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict4)
train_accuracy
```

```
0.8009259259259259
```

```
import pickle
pickle.dump(svc,open('payments.pkl','wb'))
```