Data Science Bootcamp

Capstone Project

on

Find-Default (Prediction of Credit Card fraud)

Table of Contents

Introduction

Credit card fraud has become one of the most pressing concerns for financial institutions, businesses, and consumers worldwide. With the growing dependence on online transactions and digital payments, the need for robust fraud detection mechanisms has never been more critical. Fraudulent activities not only result in significant financial losses but also undermine consumer trust in the financial system. Credit card fraud is the unauthorized use of someone's credit card or credit card information to make purchases or withdraw cash without the cardholder's consent.

In response to this growing threat, financial institutions and payment providers have turned to advanced technologies, including machine learning (ML) and artificial intelligence (AI), to develop predictive models capable of detecting fraudulent transactions in real-time. These models aim to identify suspicious patterns and behaviors in transactional data that deviate from the usual spending habits of cardholders.

One of the primary challenges in building such fraud detection systems is the highly imbalanced nature of the dataset, where fraudulent transactions constitute a tiny fraction of the total transaction volume. As a result, developing accurate predictive models becomes challenging due to the underrepresentation of the minority class (fraudulent transactions) and the need to minimize false positives without overlooking actual fraud cases.

This study leverages a real-world dataset consisting of credit card transactions to build a classification model aimed at detecting fraudulent transactions. The dataset, provided by European cardholders, spans two days in September 2013 and presents an opportunity to examine the complexities of fraud detection in financial transactions.

Problem Statement

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We built a classification model to predict whether a transaction is fraudulent or not.

About the Data

The dataset used in this study contains transaction records for credit card holders in Europe over a two-day period in September 2013. The dataset includes 284,807 transactions, out of which only 492 are fraudulent (0.172% of the total transactions). This imbalance in the dataset poses a significant challenge for building an effective fraud detection model, as the machine learning model must be trained to identify fraud despite the overwhelming presence of non-fraudulent transactions.

The dataset is anonymized, meaning that the personal details of the cardholders, such as their names, card numbers, or addresses, have been removed to protect their privacy. Instead, the features in the dataset are engineered from the raw transactional data, and most of the variables represent transformed or scaled features to protect the confidentiality of the cardholder's identity. This anonymization ensures that the dataset can be used for research purposes without compromising the privacy of individuals.

The features in the dataset include the following:

1. **V1 to V28**: These represent various anonymized features resulting from the transformation of raw transaction data. These features capture different aspects of the transaction, such as time, location, and purchasing patterns. The exact nature of these features has not been disclosed to maintain confidentiality.
2. **Time**: This feature represents the number of seconds elapsed between the transaction and the first transaction in the dataset. This feature can provide insights into the time of day and the temporal patterns of fraudulent transactions.
3. **Amount**: This feature represents the monetary value of the transaction. The amount of a transaction can be an important indicator of fraud, as fraudulent transactions may sometimes involve larger amounts, though this is not always the case.
4. **Class**: This is the target variable, where "1" indicates a fraudulent transaction and "0" indicates a legitimate transaction.

Design Choices and Performance Measures

The design of the credit card fraud detection system involves several key decisions related to the selection of features, model architecture, evaluation metrics, and handling of class imbalance.

1. Feature Engineering

- **Anonymized Features (V1 to V28)**: The dataset contains 28 anonymized features that were pre-processed before being used for model training. These features likely represent transformed versions of original variables, such as spending patterns, transaction behaviour, and possibly user profile information.
- **Time & Amount**: The "Time" and "Amount" features were also retained. "Time" helps capture temporal patterns in transactions, which could provide insights into fraudulent activities (e.g., transactions occurring at unusual hours). "Amount" represents the value of the transaction and may be a key indicator of fraud, as fraudsters may attempt to process larger transactions in a short period.

2. Class Imbalance Handling

- **Class Imbalance Problem**: The dataset is highly imbalanced, with fraudulent transactions representing only 0.172% of the total dataset. Given this, the model is prone to being biased toward predicting the majority class (non-fraudulent transactions). To address this, several techniques were used:
    - **Resampling Methods**: We applied techniques like **oversampling** (e.g., SMOTE – Synthetic Minority Over-sampling Technique) to increase the number of fraudulent transactions in the training dataset. This helps the model learn better representations of fraudulent activity.

3. Model Selection

Several machine learning models were considered for the classification task:

- **Logistic Regression**: Logistic regression is a straightforward and interpretable classification model, making it a good baseline. It works well with linear relationships and, when combined with regularization, can avoid overfitting.
- **Decision Tree**: The decision tree algorithm is highly interpretable and captures non-linear relationships in the data. It also helps in handling categorical data and is resistant to outliers, making it suitable for fraud detection.
- **Random Forest**: Random forests, an ensemble method built on multiple decision trees, were chosen to improve model performance. After evaluating these models, both **Decision Tree** and **Logistic Regression** were selected.

4. Evaluation Metrics

Given the highly imbalanced nature of the dataset, traditional metrics like accuracy are not ideal for evaluating the model performance. Instead, the following metrics were prioritized:

- **Precision and Recall**: These metrics were crucial because we need to minimize false positives (legitimate transactions misclassified as fraud) and false negatives (fraudulent transactions missed by the model). Precision measures the proportion of predicted frauds that were truly fraudulent, while recall measures the proportion of actual frauds that were correctly identified by the model.
- **F1 Score**: The F1 score, which is the harmonic mean of precision and recall, was used to balance the trade-off between these two metrics. A high F1 score indicates a good balance between detecting frauds and minimizing false positives.
- **ROC-AUC**: The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) were used to assess the model's ability to discriminate between fraudulent and non-fraudulent transactions. AUC values closer to 1 indicate a better-performing model.

5. Model Deployment

- **Streamlit for Deployment**: Streamlit was chosen as the deployment tool due to its simplicity and ability to quickly deploy machine learning models with interactive web interfaces. The deployed model allows real-time predictions of fraudulent transactions, making it accessible for practical use by stakeholders.
- **Performance Monitoring**: After deployment, the system will be monitored for performance. Any significant changes in fraud (e.g., new fraud patterns emerging) may require model retraining or fine-tuning.

Future Work

There are several potential areas for improvement and future work in building and deploying a fraud detection system:

1. Incorporating More Data

The current dataset represents only two days of transactions, which might not be sufficient to capture the full range of fraudulent behaviors. A larger, more diverse dataset that spans several months or years would help the model learn from a broader set of fraudulent patterns and behaviors.

2. Feature Engineering

The anonymized features in the dataset may still conceal valuable information. Further exploration into more sophisticated feature engineering techniques, such as incorporating external data (e.g., merchant information, geographical data, or time-of-day patterns), may enhance the model's ability to detect fraud.

Additionally, exploring non-linear transformations of the features or using techniques like principal component analysis (PCA) for dimensionality reduction could reveal hidden patterns and improve model performance.

3. Advanced Models

Although Decision Trees and Logistic Regression were chosen for their interpretability and performance, more advanced models, such as **Gradient Boosting Machines (GBM)**, **XGBoost**, or **Deep Learning** models, could be explored. These models have shown success in imbalanced datasets and could further improve fraud detection accuracy.

4. Real-Time Fraud Detection

Future work could focus on integrating the model with real-time transactional systems, allowing financial institutions to flag fraudulent activities as they happen. Continuous monitoring of incoming transactions and retraining the model on fresh data could ensure the system adapts to new fraud trends.

5. Anomaly Detection

In addition to supervised learning approaches, incorporating **unsupervised anomaly detection** techniques could help in identifying new, previously unseen fraudulent behaviors that do not follow established patterns. This is particularly important as fraudsters constantly evolve their tactics to bypass detection systems.

Source Code

Importing Required Libraries import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns !pip install imbalanced-learn Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.12.4) Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.26.4) Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.13.1) Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2) Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.4.2) Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0) !pip install XGBoost Requirement already satisfied: XGBoost in /usr/local/lib/python3.10/dist-packages (2.1.2) Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from XGBoost) (1.26.4) Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from XGBoost) (2.23.4) Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from XGBoost) (1.13.1) from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve from sklearn.preprocessing import StandardScaler from sklearn.ensemble import RandomForestClassifier from xgboost import XGBClassifier from imblearn.over_sampling import SMOTE import warnings warnings.filterwarnings('ignore') from sklearn.linear_model import LogisticRegression from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier from sklearn.svm import SVC from sklearn.neighbors import KNeighborsClassifier from sklearn.naive_bayes import GaussianNB from xgboost import XGBClassifier from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # Load the dataset df = pd.read_csv('creditcard.csv') I. Exploratory Data Analysis Data Quality Check df.head() {"type":"dataframe","variable_name":"df"} df.info() RangeIndex: 59511 entries, 0 to 59510 Data columns (total 31 columns): # Column Non-Null Count Dtype --- ------ -------------- ----- 0 Time 59511 non-null int64 1 V1 59511 non-null float64 2 V2 59511 non-null float64 3 V3 59511 non-null float64 4 V4 59511 non-null float64 5 V5 59511 non-null float64 6 V6 59511 non-null float64 7 V7 59511 non-null float64 8 V8 59511 non-null float64 9 V9 59511 non-null float64 10 V10 59511 non-null float64 11 V11 59511 non-null float64 12 V12 59511 non-null float64 13 V13 59511 non-null float64 14 V14 59511 non-null float64 15 V15 59511 non-null float64 16 V16 59510 non-null float64 17 V17 59510 non-null float64 18 V18 59510 non-null float64 19 V19 59510 non-null float64 20 V20 59510 non-null float64 21 V21 59510

non-null float64 22 V22 59510 non-null float64 23 V23 59510 non-null float64 24 V24 59510 non-null float64 25 V25 59510 non-null float64 26 V26 59510 non-null float64 27 V27 59510 non-null float64 28 V28 59510 non-null float64 29 Amount 59510 non-null float64 30 Class 59510 non-null float64 dtypes: float64(30), int64(1) memory usage: 14.1 MB print(df.dtypes) Time int64 V1 float64 V2 float64 V3 float64 V4 float64 V5 float64 V6 float64 V7 float64 V8 float64 V9 float64 V10 float64 V11 float64 V12 float64 V13 float64 V14 float64 V15 float64 V16 float64 V17 float64 V18 float64 V19 float64 V20 float64 V21 float64 V22 float64 V23 float64 V24 float64 V25 float64 V26 float64 V27 float64 V28 float64 Amount float64 Class float64 dtype: object df.isnull().sum() Time 0 V1 0 V2 0 V3 0 V4 0 V5 0 V6 0 V7 0 V8 0 V9 0 V10 0 V11 0 V12 0 V13 0 V14 0 V15 0 V16 1 V17 1 V18 1 V19 1 V20 1 V21 1 V22 1 V23 1 V24 1 V25 1 V26 1 V27 1 V28 1 Amount 1 Class 1 dtype: int64 df.describe() {"type":"dataframe"} fraud_percentage = df['Class'].value_counts(normalize=True) * 100 print(fraud_percentage) Class 0.0 99.727777 1.0 0.272223 Name: proportion, dtype: float64 Statistical Analysis using Visualizations sns.countplot(x='Class', data=df) plt.title('Class Distribution') plt.show() for col in ['Amount', 'Time']: plt.figure() sns.histplot(df[col], bins=50, kde=True) plt.title(f'Distribution of {col}') plt.show() for col in ['Amount', 'V1', 'V2', 'V3']: # Adjust the feature list as needed plt.figure() sns.kdeplot(df[df['Class'] == 0][col], label='Non-Fraud', shade=True) sns.kdeplot(df[df['Class'] == 1][col], label='Fraud', shade=True) plt.title(f'{col} Distribution by Class') plt.legend() plt.show() plt.figure(figsize=(10, 5)) sns.boxplot(x='Class', y='Amount', data=df) plt.title('Amount Distribution by Class') plt.show() df.hist(figsize=(20,20),color='lime') plt.show() Correlation Analysis plt.figure(figsize=(12,10)) sns.heatmap(df.corr(), annot=False) plt.title("Correlation Heatmap of Features") plt.show() # Bin 'Amount' into categories amount_bins = [-1, 0, 50, 100, 500, 1000, 5000, 10000] # Define bin ranges df['Amount_Bin'] = pd.cut(df['Amount'], bins=amount_bins) # Crosstab between 'Class' and 'Amount_Bin' amount_class_crosstab = pd.crosstab(df['Amount_Bin'], df['Class'], rownames=['Amount Range'], colnames=['Fraud Class']) print(amount_class_crosstab) Fraud Class 0.0 1.0 Amount Range (-1, 0] 473 4 (0, 50] 37574 99 (50, 100] 8675 30 (100, 500] 10591 20 (500, 1000] 1364 7 (1000, 5000] 655 2 (5000, 10000] 13 0 Hypothesis Testing from scipy.stats import ttest_ind # Perform a t-test on 'Amount' between fraud and non-fraud transactions fraud = df[df['Class'] == 1]['Amount'] non_fraud = df[df['Class'] == 0]['Amount'] fraud.shape (162,) non_fraud.shape (59348,) t_stat, p_val = ttest_ind(fraud, non_fraud) print("T-test results: T-statistic = {:.3f}, P-value = {:.3f}".format(t_stat, p_val)) T-test results: T-statistic = -0.087, P-value = 0.930 II. Data Cleaning and Transformation df.shape (59511, 32) Missing Value Treamtment df = df.dropna() df.isnull().sum() Time 0 V1 0 V2 0 V3 0 V4 0 V5 0 V6 0 V7 0 V8 0 V9 0 V10 0 V11 0 V12 0 V13 0 V14 0 V15 0 V16 0 V17 0 V18 0 V19 0 V20 0 V21 0 V22 0 V23 0 V24 0 V25 0 V26 0 V27 0 V28 0 Amount 0 Class 0 Amount_Bin 0

dtype: int64 # Impute missing values in 'Amount' with the median df['Amount'].fillna(df['Amount'].median(), inplace=True) df.head() {"type":"dataframe","variable_name":"df"} Outlier Treatment Q1 = df['Amount'].quantile(0.25) Q3 = df['Amount'].quantile(0.75) IQR = Q3 - Q1 lower_bound = Q1 - 1.5 * IQR upper_bound = Q3 + 1.5 * IQR df = df[(df['Amount'] >= lower_bound) & (df['Amount'] <= upper_bound)] df.shape (53256, 32) # Quantile-based binning df['Amount_Bin'] = pd.qcut(df['Amount'], q=4, labels=['low', 'medium', 'high', 'very_high']) df.head() {"type":"dataframe","variable_name":"df"} Feature Engineering # Standardize the 'Amount' feature using z-score normalization df['Amount'] = (df['Amount'] - df['Amount'].mean()) / df['Amount'].std() start_datetime = pd.to_datetime("2013-09-01 00:00:00") # Convert 'Time' to datetime by adding seconds to start_datetime df['Transaction_DateTime'] = start_datetime + pd.to_timedelta(df['Time'], unit='s') df['Transaction_DateTime'].head() 0 2013-09-01 00:00:00 1 2013-09-01 00:00:00 3 2013-09-01 00:00:01 4 2013-09-01 00:00:02 5 2013-09-01 00:00:02 Name: Transaction_DateTime, dtype: datetime64[ns] # Convert 'Time' to hours df['Hour'] = df['Time'] / 3600 # Now 'Hour' ranges from 0 to 48 df['Hour'] = df['Hour'].astype(int) # Convert to integer hours for grouping # Filter fraud transactions (Class == 1) and count per hour fraud_counts = df[df['Class'] == 1].groupby('Hour').size() fraud_counts = fraud_counts.reindex(range(48), fill_value=0) print(fraud_counts.head(10)) Hour 0 1 1 1 2 20 3 13 4 5 5 10 6 3 7 22 8 5 9 14 dtype: int64 Fraud Transactions per hour is seen above # Convert Time to hours (0-48) and create cyclic features df['Hour'] = df['Time'] / 3600 % 24 df['Hour_sin'] = np.sin(2 * np.pi * df['Hour'] / 24) df['Hour_cos'] = np.cos(2 * np.pi * df['Hour'] / 24) df.head() {"type":"dataframe","variable_name":"df"} III. Balancing Data # Separate features and target X = df.drop('Class', axis=1) y = df['Class'] # Check data types in X print(X.dtypes) Time int64 V1 float64 V2 float64 V3 float64 V4 float64 V5 float64 V6 float64 V7 float64 V8 float64 V9 float64 V10 float64 V11 float64 V12 float64 V13 float64 V14 float64 V15 float64 V16 float64 V17 float64 V18 float64 V19 float64 V20 float64 V21 float64 V22 float64 V23 float64 V24 float64 V25 float64 V26 float64 V27 float64 V28 float64 Amount float64 Amount_Bin category Transaction_DateTime datetime64[ns] Hour float64 Hour_sin float64 Hour_cos float64 dtype: object # Drop the 'Transaction_DateTime' column X = X.drop(columns=['Transaction_DateTime']) # Define a mapping dictionary mapping = {'Low': 1, 'Medium': 2, 'High': 3} # Apply the mapping X['Amount_Bin'] = X['Amount_Bin'].map(mapping) from sklearn.preprocessing import LabelEncoder # Initialize the encoder label_encoder = LabelEncoder() # Fit and transform the 'Amount_Bin' column X['Amount_Bin'] = label_encoder.fit_transform(X['Amount_Bin']) + 1 # Apply SMOTE smote = SMOTE(random_state=42) X_resampled, y_resampled = smote.fit_resample(X, y) from sklearn.model_selection import train_test_split # Split into training and testing sets X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,

test_size=0.2, random_state=42) IV. Predictive Modelling Logistic Regression from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve # Initializing the Logistic Regression model log_reg = LogisticRegression() log_reg.fit(X_train, y_train) y_pred = log_reg.predict(X_test) y_pred_proba = log_reg.predict_proba(X_test)[:, 1] y_pred array([0., 0., 1., ..., 0., 0., 1.]) print("Logistic Regression Model Metrics:") print("Accuracy:", accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred)) print("Recall:", recall_score(y_test, y_pred)) print("F1 Score:", f1_score(y_test, y_pred)) print("ROC-AUC:", roc_auc_score(y_test, y_pred_proba)) Logistic Regression Model Metrics: Accuracy: 0.9800884955752213 Precision: 0.9893812144029347 Recall: 0.9703654610869153 F1 Score: 0.9797810812102672 ROC-AUC: 0.9978128539495447 Decision Tree from sklearn.tree import DecisionTreeClassifier dt = DecisionTreeClassifier(random_state=42) dt.fit(X_train, y_train) y_pred = dt.predict(X_test) print("Decision Tree Model Metrics:") print("Accuracy:", accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred)) print("Recall:", recall_score(y_test, y_pred)) print("F1 Score:", f1_score(y_test, y_pred)) print("ROC-AUC:", roc_auc_score(y_test, y_pred_proba)) Decision Tree Model Metrics: Accuracy: 0.9989173413669742 Precision: 0.9983921308994609 Recall: 0.9994319257716342 F1 Score: 0.9989117577478117 ROC-AUC: 0.9978128539495447 # Plot ROC Curve fpr, tpr, _ = roc_curve(y_test, y_pred_proba) plt.plot(fpr, tpr, label="Decision Tree") plt.xlabel("False Positive Rate") plt.ylabel("True Positive Rate") plt.title("ROC Curve - Decision Tree") plt.legend() plt.show() from sklearn.tree import plot_tree import matplotlib.pyplot as plt # Convert feature names to list format feature_names = X_train.columns.tolist() # Visualize Decision Tree plt.figure(figsize=(15, 10)) plot_tree(dt, feature_names=feature_names, class_names=["Not Fraud", "Fraud"], filled=True) plt.title("Decision Tree Visualization") plt.show() Random Forest rf = RandomForestClassifier(n_estimators=100, # Number of trees max_depth=10, # Maximum depth of the tree random_state=42, # Ensures reproducibility n_jobs=-1) rf.fit(X_train, y_train) RandomForestClassifier(max_depth=10, n_jobs=-1, random_state=42) print("Random Forest Model Metrics:") print("Accuracy:", accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred)) print("Recall:", recall_score(y_test, y_pred)) print("F1 Score:", f1_score(y_test, y_pred)) print("ROC-AUC:", roc_auc_score(y_test, y_pred_proba)) Random Forest Model Metrics: Accuracy: 0.9989173413669742 Precision: 0.9983921308994609 Recall: 0.9994319257716342 F1 Score: 0.9989117577478117 ROC-AUC: 0.9978128539495447 # Plot ROC Curve fpr, tpr, _ = roc_curve(y_test, y_pred_proba) plt.plot(fpr, tpr, label="Random Forest") plt.xlabel("False Positive Rate") plt.ylabel("True Positive Rate") plt.title("ROC Curve - Random Forest") plt.legend() plt.show() importances = rf.feature_importances_

feature_names = X_train.columns indices = np.argsort(importances)[-10:] # Top 10 features plt.figure(figsize=(10, 6)) plt.title("Top 10 Feature Importances") plt.barh(range(len(indices)), importances[indices], align="center") plt.yticks(range(len(indices)), [feature_names[i] for i in indices]) plt.xlabel("Relative Importance") plt.show() V. Model Validation and Hyperparameter Tuning Boosting from sklearn.ensemble import GradientBoostingClassifier from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score from sklearn.model_selection import train_test_split gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.05, max_depth=3, subsample=0.8, min_samples_split=10, min_samples_leaf=5, random_state=42) gb_model.fit(X_train, y_train) y_pred = gb_model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) precision = precision_score(y_test, y_pred) recall = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) print("Gradient Boosting Performance:") print(f"Accuracy: {accuracy:.4f}") print(f"Precision: {precision:.4f}") print(f"Recall: {recall:.4f}") print(f"F1 Score: {f1:.4f}") Gradient Boosting Performance: Accuracy: 0.9964 Precision: 0.9966 Recall: 0.9962 F1 Score: 0.9964 Cross Validation from sklearn.model_selection import cross_val_score # Logistic Regression Cross-Validation lr_cv_scores = cross_val_score(log_reg, X, y, cv=5, scoring='f1') print("Logistic Regression CV F1 Scores:", lr_cv_scores) print("Mean F1 Score:", lr_cv_scores.mean()) Logistic Regression CV F1 Scores: [0.07509387 0.58227848 0.12903226 0.75 0.37837838] Mean F1 Score: 0.38295659695794415 # Decision Tree Cross-Validation dt_cv_scores = cross_val_score(dt, X, y, cv=5, scoring='f1') print("Decision Tree CV F1 Scores:", dt_cv_scores) print("Mean F1 Score:", dt_cv_scores.mean()) Decision Tree CV F1 Scores: [0.00591133 0.32727273 0.54166667 0.8 0.70833333] Mean F1 Score: 0.4766368114643977 # Random Forest Cross-Validation rf_cv_scores = cross_val_score(rf, X, y, cv=5, scoring='f1') print("Random Forest CV F1 Scores:", rf_cv_scores) print("Mean F1 Score:", rf_cv_scores.mean()) Random Forest CV F1 Scores: [0.00591366 0.63157895 0.66666667 0.91803279 0.69565217] Mean F1 Score: 0.5835688470778522 Hyperparameter Tuning # Logistic Regression lr_param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'saga']} lr_grid_search = GridSearchCV(LogisticRegression(), lr_param_grid, cv=5, scoring='f1') lr_grid_search.fit(X_train, y_train) print("Best parameters for Logistic Regression:", lr_grid_search.best_params_) best_lr_model = lr_grid_search.best_estimator_ Best parameters for Logistic Regression: {'C': 1, 'solver': 'liblinear'} y_pred = best_lr_model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) precision = precision_score(y_test, y_pred) recall = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) cm = confusion_matrix(y_test, y_pred) print(accuracy,precision,recall,f1,cm) 0.973404255319149 0.9860935524652339 0.9600454459382692 0.972895178699928 [[10539 143] [422 10140]] The Best Model model_predictions = { "Logistic Regression": log_reg.predict(X_test), "Decision Tree":

dt.predict(X_test), "Random Forest": rf.predict(X_test) } results = [] for model_name, y_pred in model_predictions.items(): accuracy = accuracy_score(y_test, y_pred) precision = precision_score(y_test, y_pred) recall = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) # Append results results.append({ "Model": model_name, "Accuracy": accuracy, "Precision": precision, "Recall": recall, "F1 Score": f1, }) for result in results: print(result) {'Model': 'Logistic Regression', 'Accuracy': 0.9800884955752213, 'Precision': 0.9893812144029347, 'Recall': 0.9703654610869153, 'F1 Score': 0.9797810812102672} {'Model': 'Decision Tree', 'Accuracy': 0.9989173413669742, 'Precision': 0.9983921308994609, 'Recall': 0.9994319257716342, 'F1 Score': 0.9989117577478117} {'Model': 'Random Forest', 'Accuracy': 0.9993409903972886, 'Precision': 0.9992427110942824, 'Recall': 0.9994319257716342, 'F1 Score': 0.9993373094764745} results_df = pd.DataFrame(results) results_melted = results_df.melt(id_vars="Model", var_name="Metric", value_name="Score") sns.set(style="whitegrid") plt.figure(figsize=(10, 6)) sns.barplot(data=results_melted, x="Metric", y="Score", hue="Model", palette="viridis") As Random Forest is seen to have high evaluation metrics, there is a chance of overfitting. Thus, we choose the Decision Tree and Logistic Regression models. Among the two, Logistic Regression model best classifies the data into fraud and non-fraud. Time Series Analysis from statsmodels.tsa.arima.model import ARIMA import matplotlib.pyplot as plt # Fit ARIMA model model = ARIMA(fraud_counts, order=(1, 1, 1)) model_fit = model.fit() forecast = model_fit.forecast(steps=48) print(forecast) 48 0.000010 49 0.000008 50 0.000008 51 0.000008 52 0.000008 53 0.000008 54 0.000008 55 0.000008 56 0.000008 57 0.000008 58 0.000008 59 0.000008 60 0.000008 61 0.000008 62 0.000008 63 0.000008 64 0.000008 65 0.000008 66 0.000008 67 0.000008 68 0.000008 69 0.000008 70 0.000008 71 0.000008 72 0.000008 73 0.000008 74 0.000008 75 0.000008 76 0.000008 77 0.000008 78 0.000008 79 0.000008 80 0.000008 81 0.000008 82 0.000008 83 0.000008 84 0.000008 85 0.000008 86 0.000008 87 0.000008 88 0.000008 89 0.000008 90 0.000008 91 0.000008 92 0.000008 93 0.000008 94 0.000008 95 0.000008 Name: predicted_mean, dtype: float64 plt.figure(figsize=(12, 6)) plt.plot(fraud_counts, label='Observed Fraud Counts') plt.plot(range(48, 96), forecast, color='red', label='Forecasted Fraud Counts') plt.xlabel('Hour') plt.ylabel('Fraud Count') plt.legend() plt.title('Fraud Forecast for the Next Two Days') plt.show() Here, we forecast the number of frauds occouring in the next 48 hours. Model Deployment import joblib joblib.dump(best_lr_model, 'logreg_model.pkl') ['logreg_model.pkl'] joblib.dump(dt, 'dt_model.pkl') ['dt_model.pkl'] pip install streamlit joblib Collecting streamlit Downloading streamlit-1.40.1-py2.py3-none-any.whl.metadata (8.5 kB) Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (1.4.2) Requirement already satisfied: altair=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)

Requirement already satisfied: blinker=1.0.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.9.0) Requirement already satisfied: cachetools=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (5.5.0) Requirement already satisfied: click=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7) Requirement already satisfied: numpy=1.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.26.4) Requirement already satisfied: packaging=20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (24.2) Requirement already satisfied: pandas=1.4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.2.2) Requirement already satisfied: pillow=7.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (11.0.0) Requirement already satisfied: protobuf=3.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.25.5) Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (17.0.0) Requirement already satisfied: requests=2.27 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.32.3) Requirement already satisfied: rich=10.14.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (13.9.4) Requirement already satisfied: tenacity=8.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (9.0.0) Requirement already satisfied: toml=0.10.1 in /usr/local/lib/python3.10/dist-packages (from streamlit) (0.10.2) Requirement already satisfied: typing-extensions=4.3.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.12.2) Requirement already satisfied: gitpython!=3.1.19,=3.0.7 in /usr/local/lib/python3.10/dist-packages (from streamlit) (3.1.43) Collecting pydeck=0.8.0b4 (from streamlit) Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB) Requirement already satisfied: tornado=6.0.3 in /usr/local/lib/python3.10/dist-packages (from streamlit) (6.3.3) Collecting watchdog=2.1.5 (from streamlit) Downloading watchdog-6.0.0-py3-nonemanylinux2014_x86_64.whl.metadata (44 kB) ━━ 44.3/44.3 kB 2.1 MB/s eta 0:00:00 ent already satisfied: entrypoints in /usr/local/lib/python3.10/distpackages (from altair=4.0->streamlit) (0.4) Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from altair=4.0- >streamlit) (3.1.4) Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair=4.0- >streamlit) (4.23.0) Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair=4.0- >streamlit) (0.12.1) Requirement already satisfied: gitdb=4.0.1 in /usr/local/lib/python3.10/dist-packages (from gitpython! =3.1.19,=3.0.7->streamlit) (4.0.11) Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas=1.4.0- >streamlit) (2.8.2) Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas=1.4.0- >streamlit) (2024.2) Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas=1.4.0- >streamlit) (2024.2) Requirement already satisfied:

charset-normalizer=2 in /usr/local/lib/python3.10/dist-packages (from requests=2.27- >streamlit) (3.4.0) Requirement already satisfied: idna=2.5 in /usr/local/lib/python3.10/dist-packages (from requests=2.27- >streamlit) (3.10) Requirement already satisfied: urllib3=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests=2.27- >streamlit) (2.2.3) Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests=2.27->streamlit) (2024.8.30) Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich=10.14.0- >streamlit) (3.0.0) Requirement already satisfied: pygments=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich=10.14.0- >streamlit) (2.18.0) Requirement already satisfied: smmap=3.0.1 in /usr/local/lib/python3.10/dist-packages (from gitdb=4.0.1->gitpython!=3.1.19,=3.0.7->streamlit) (5.0.1) Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->altair=4.0- >streamlit) (3.0.2) Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0- >altair=4.0->streamlit) (24.2.0) Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0- >altair=4.0->streamlit) (2024.10.1) Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0- >altair=4.0->streamlit) (0.35.1) Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair=4.0->streamlit) (0.21.0) Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich=10.14.0->streamlit) (0.1.2) Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas=1.4.0->streamlit) (1.16.0) Downloading streamlit-1.40.1-py2.py3-none-any.whl (8.6 MB) ━━ 8.6/8.6 MB 35.0 MB/s eta 0:00:00 ━━ 6.9/6.9 MB 45.1 MB/s eta 0:00:00 anylinux2014_x86_64.whl (79 kB) ━━ 79.1/79.1 kB 5.2 MB/s eta 0:00:00 lit Successfully installed pydeck-0.9.1 streamlit-1.40.1 watchdog-6.0.0 import streamlit as st # Load the saved models logreg_model = joblib.load('logreg_model.pkl') dt_model = joblib.load('dt_model.pkl') # App Title st.title("Credit Card Fraud Detection") 2024-11-17 04:56:09.471 WARNING streamlit.runtime.scriptrunner_utils.script_run_context: Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.704 Warning: to view this Streamlit app on a browser, run it with the following command: streamlit run /usr/local/lib/python3.10/dist-packages/colab_kernel_launcher.py [ARGUMENTS] 2024-11-17 04:56:09.705 Thread

'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. DeltaGenerator() # Select Model model_choice = st.selectbox("Select Model:", ("Logistic Regression", "Decision Tree")) # Input Features st.subheader("Enter Transaction Features:") feature1 = st.number_input("Feature 1 (e.g., Time):", min_value=0.0, step=0.1) feature2 = st.number_input("Feature 2 (e.g., V1):", min_value=-10.0, max_value=10.0, step=0.1) feature3 = st.number_input("Feature 3 (e.g., V2):", min_value=-10.0, max_value=10.0, step=0.1) 2024-11-17 04:56:09.730 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.733 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.735 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.738 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.743 Session state does not function when running a script without `streamlit run` 2024-11-17 04:56:09.746 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.748 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.750 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.751 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.753 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.755 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.756 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.758 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.760 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.762 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.764 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.765 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.767 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.768 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.769 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.771 Thread

'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.772 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.773 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.775 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.776 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.777 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode. 2024-11-17 04:56:09.779 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.