Firstly, to run the project please run the main.py file.
Or you can also use the eda.ipynb, training.ipynb, and evaluation.ipynb notebooks to run the project.

Project Structure:

```
├── CreditCardFraudDetection
│   ├── Dataset(contains the creditcard.csv dataset)
│   ├── models(contains all the saved models)
│   ├── Plots(contains all the plots and figures)
│   └── SrC(contains the source code of the project)
│       ├── _pycache_
│       ├── __init__.py(empty file)
│       ├── evaluate_models.py(Contains code to evaluate the models)
│       ├── logreg_gd.py(Contains code to train and evaluate the Gradient Descent model)
│       ├── plotting.py(Contains code to plot the confusion matrix and curves)
│       ├── Preprocessing.py (Contains code to load and preprocess the data, scaling and
│                                 sampling is applied here)
│       ├── train_models.py(Contains code to train,load,and save the models)
│   └── _init_.py(empty file)
│   └── eda.ipynb(Exploratory Data Analysis is performed here)
│   └── training.ipynb(Model training is done here)
│   └── evaluation.ipynb(Loads saved models and evaluates them, plots figures.)
│   ├── main.py(Main file to run the project, contains a sample run.)
    └── Readme.md(This file.)
```

In this task I have performed Credit Card Fraud Detection using a few different Machine Learning Models. The main aim was to detect as many fraud cases as possible.

I have used the Credit Card Fraud Detection dataset from Kaggle for this task. The dataset was collected in 2013 and contains the transaction information of European card holders. The transactions were made over 48 hours and about 284,407 transactions were made. Only 492 were fraud transactions, which are only 0.17% of the total transactions.

Firstly, I have explored the data and have performed data analysis, which revealed that the data is hugely imbalanced. I have also performed analysis to understand the relationship between the transaction time and fraud transactions, transaction amount and fraud transactions, and similar other attributes. The correlation heatmap revealed that most of the variables were not very strongly correlated. I have also performed analysis to catch outliers.

Feature Scaling:
The values for the time and amount data were very high so I have performed Standard Scaling using Sklearn's StandardScaler. This was important to make sure that all of our data was equally scaled and no variable had more impact.

Data Split and Sampling:
After scaling the data I have separated it into training and testing data. Now the train data is sampled using different sampling techniques such as RandomUnderSampling, RandomOverSampling, and SMOTE. Test data was not sampled and left as it is to make sure that we get our predictions on realistic data.

Approach:
In order to get the best possible predictions, I have used multiple sampling techniques which give different sets of training data. I have then trained all the different data using multiple Machine Learning models. The models were all trained on different hyperparameters to select the best versions of all the models.

Architecture:
I have used 5 different models. Those include Sklearn's LogisticRegression, RandomForestClassifier, DecisionTreeClassifier, and GradientBoostingClassifier. I have also created a LogisticRegression with Gradient Descent classifier from scratch. I have used SKlearn's GridSearchCV and StratifiedKFold to perform cross validation.

GridSearchCV was used to select the best available hyperparameters for all the models and StratifiedKFold was used to select the best learning rate for the GradientDescent model. All models were trained multiple times with different combinations of data and GridSearchCV scoring.

For example, Undersampled,Oversampled, and original data were used with combinations of GridSearchCV scoring(f1_score, precision, recall, roc_auc_score) to train multiple models each with the best hyperparameters.

This gave an objective look at how all these models performed with how the data was sampled and what the scoring rate was.

For a task like this the tradeoff between precision and recall is very important to reduce errors. F1_score is then obviously the best choice. For all the trained models, RandomForestClassifier took the longest to train but the model trained with best hyperparameters also gave the best results. The results are shared in detail in the evaluation.ipynb file.

Key Decisions:

After training the Logistic Regression and Decision Tree Classifier on all possible permutations of sampling techniques and scoring rates, a clear picture could be seen that the undersampled data along with f1_scoring was producing the most optimal results. Also, RandomForest and GradientBoosting models take way too long to train so they were trained on undersampled and original data with f1 scoring.