Credit Card Fraud Detection

Importing models and libraries

```
# Importing modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
```

```
dataset = pd.read_csv("/content/drive/MyDrive/IndiDrive/creditcard.csv")
dataset.head().append(dataset.tail())
```

```
<ipython-input-3-ed0bc59ac869>:2: FutureWarning: The frame.append method is depre
  dataset.head().append(dataset.tail())
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

10 rows × 31 columns

Data Exploration and visualization

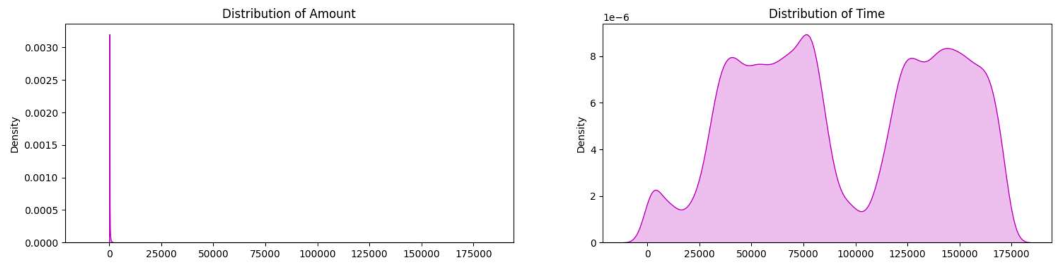
Now we try to find out the relative proportion of valid and fraudulent credit card transactions:

```
print("Fraudulent Cases: " + str(len(dataset[dataset["Class"] == 1])))
print("Valid Transactions: " + str(len(dataset[dataset["Class"] == 0])))
print("Proportion of Fraudulent Cases: " + str(len(dataset[dataset["Class"] == 1])/ dataset.shape[0]))

data_p = dataset.copy()
data_p[" "] = np.where(data_p["Class"] == 1 ,  "Fraud", "Genuine")

# plot a pie chart
data_p[" "].value_counts().plot(kind="pie")
```

```
Fraudulent Cases: 492
Valid Transactions: 284315
Proportion of Fraudulent Cases: 0.001727485630620034
<Axes: ylabel=' '>
```

There is an imbalance in the data, with only 0.17% of the total cases being fraudulent.

```
# plot the named features
f, axes = plt.subplots(1, 2, figsize=(18,4), sharex = True)

amount_value = dataset['Amount'].values # values
time_value = dataset['Time'].values # values

sns.distplot(amount_value, hist=False, color="m", kde_kws={"shade": True}, ax=axes[0]).set_title('Distribution of Amount')
sns.distplot(time_value, hist=False, color="m", kde_kws={"shade": True}, ax=axes[1]).set_title('Distribution of Time')

plt.show()
```



```
print("Average Amount in a Fraudulent Transaction: " + str(dataset[dataset["Class"] == 1]["Amount"].mean()))
print("Average Amount in a Valid Transaction: " + str(dataset[dataset["Class"] == 0]["Amount"].mean()))
```

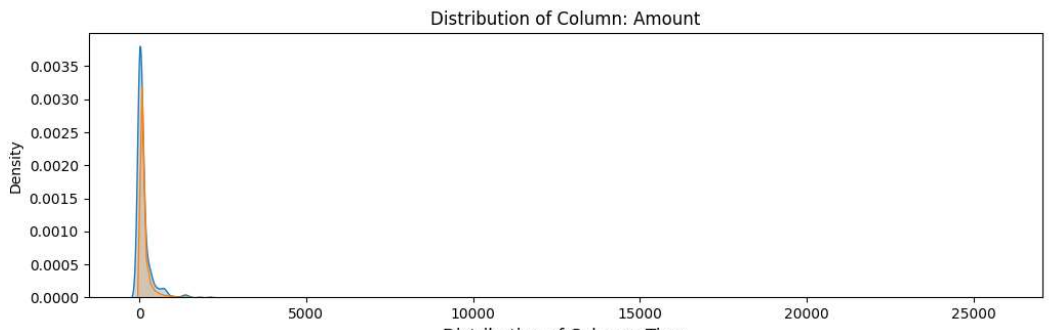
```
Average Amount in a Fraudulent Transaction: 122.21132113821139
Average Amount in a Valid Transaction: 88.29102242231328
```

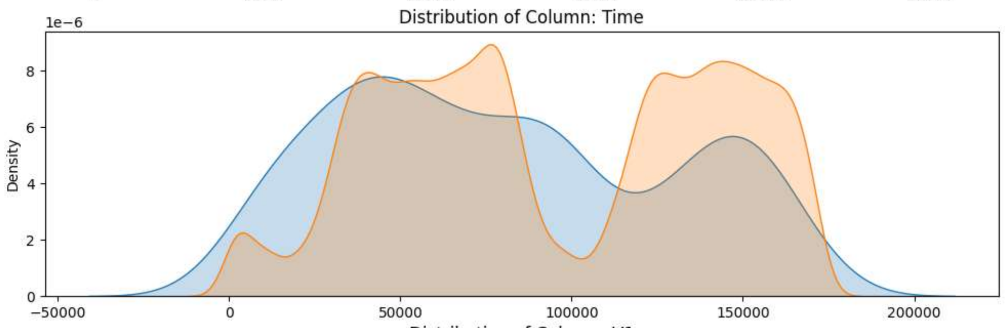
As we can notice from this, the average money transaction for the fraudulent ones is more. It makes this problem crucial to deal with. Now let us try to understand the distribution of values in each feature. Let's start with the Amount:

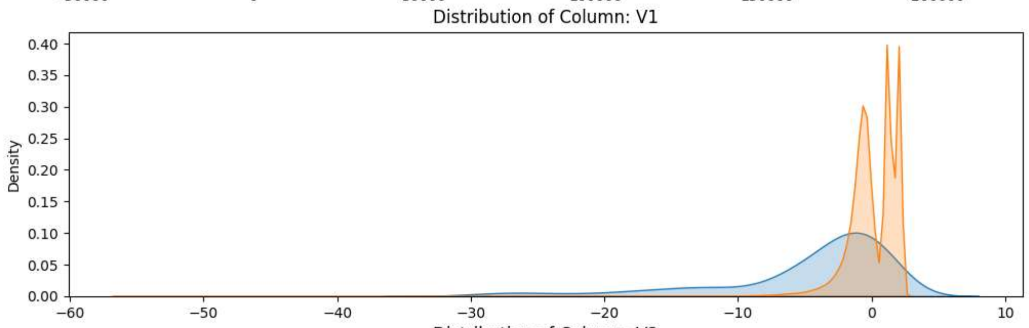
```
print("Summary of the feature - Amount" + "\n-------------------------------")
print(dataset["Amount"].describe())
```

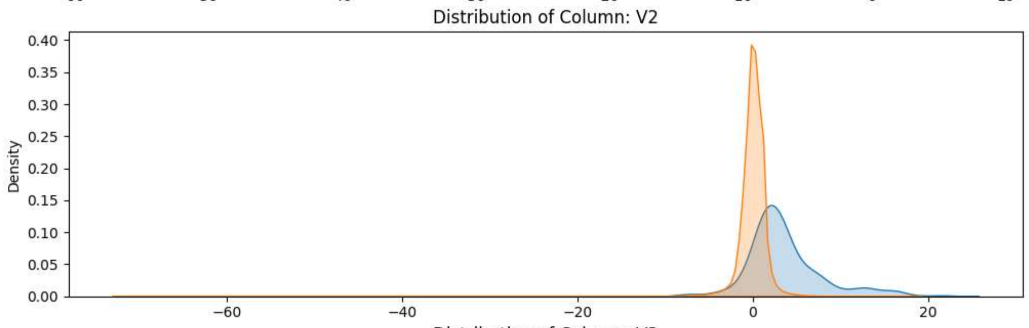
```
Summary of the feature - Amount
-------------------------------
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

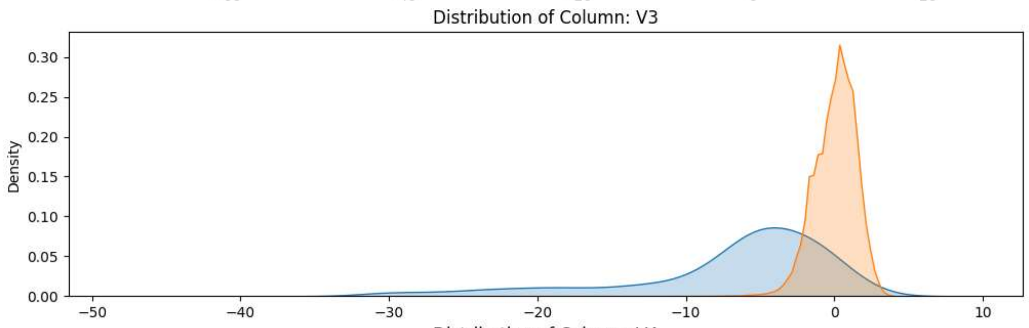
```
# Reorder the columns Amount, Time then the rest
data_plot = dataset.copy()
amount = data_plot['Amount']
data_plot.drop(labels=['Amount'], axis=1, inplace = True)
data_plot.insert(0, 'Amount', amount)

# Plot the distributions of the features
columns = data_plot.iloc[:,0:30].columns
plt.figure(figsize=(12,30*4))
grids = gridspec.GridSpec(30, 1)
for grid, index in enumerate(data_plot[columns]):
 ax = plt.subplot(grids[grid])
 sns.distplot(data_plot[index][data_plot.Class == 1], hist=False, kde_kws={"shade": True}, bins=50)
 sns.distplot(data_plot[index][data_plot.Class == 0], hist=False, kde_kws={"shade": True}, bins=50)
 ax.set_xlabel("")
 ax.set_title("Distribution of Column: "  + str(index))
plt.show()
```

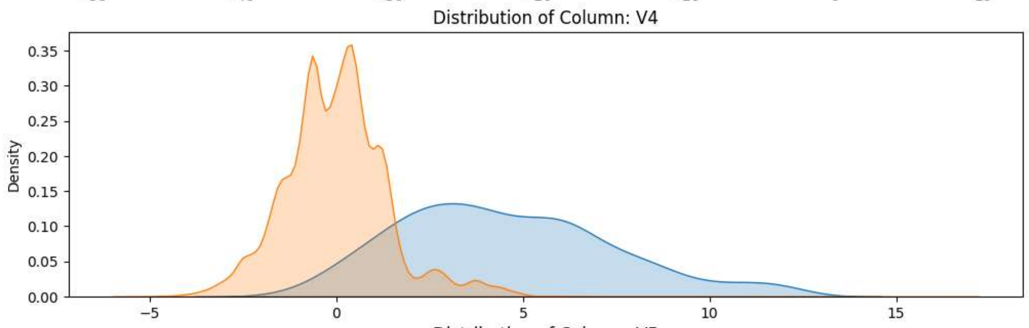

Distribution of Column: Amount
Density
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
5000
10000
15000
20000
25000

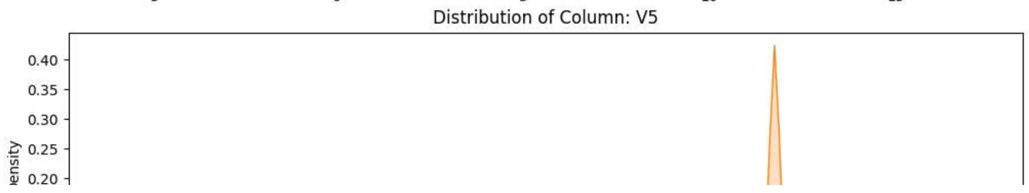

Distribution of Column: Time
1e−6
Density
8
6
4
2
0
−50000
0
50000
100000
150000
200000

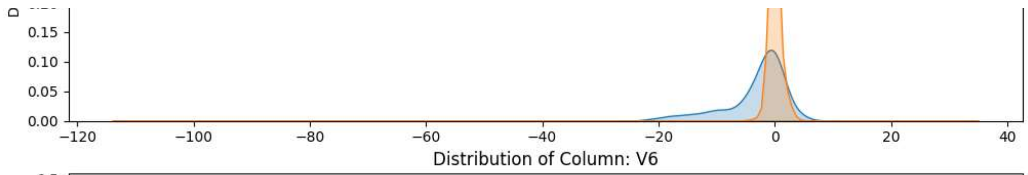

Distribution of Column: V1
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−50
−40
−30
−20
−10
0
10

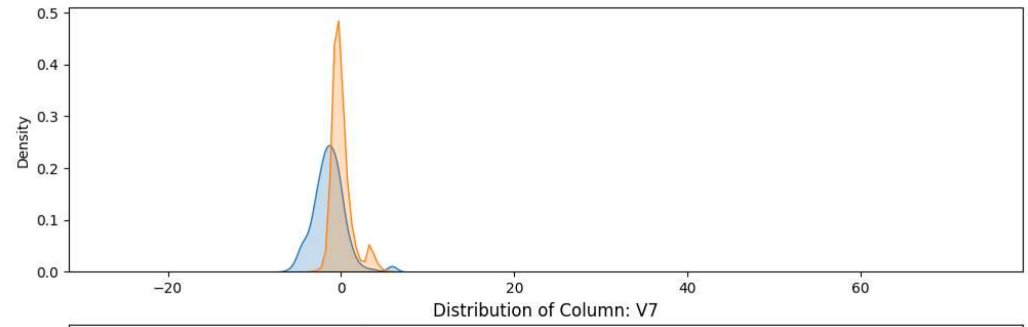

Distribution of Column: V2
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20

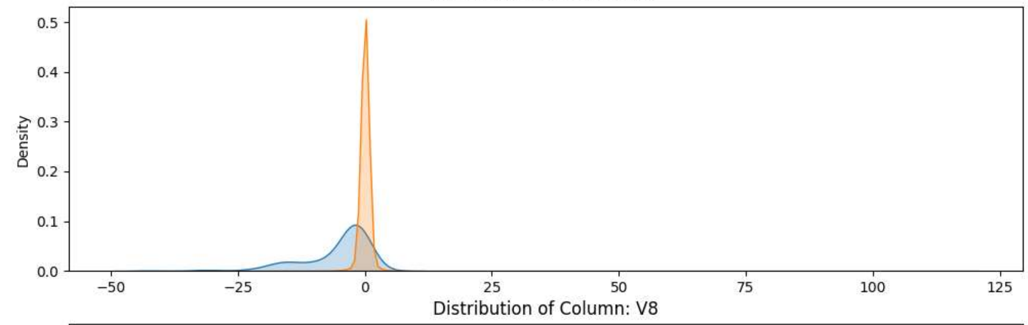

Distribution of Column: V3
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
10

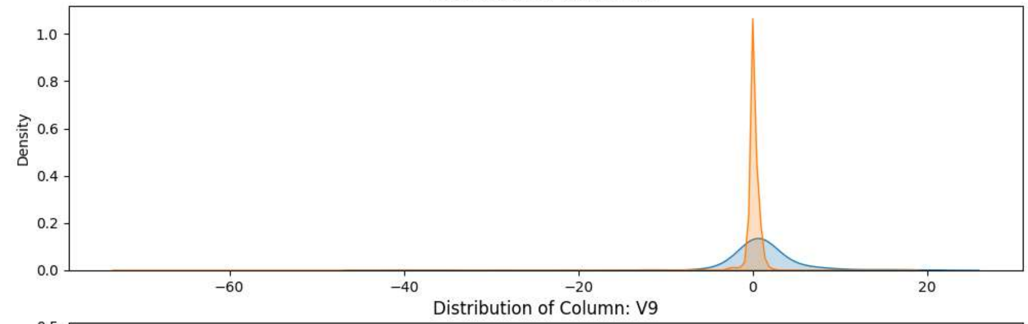

Distribution of Column: V4
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15

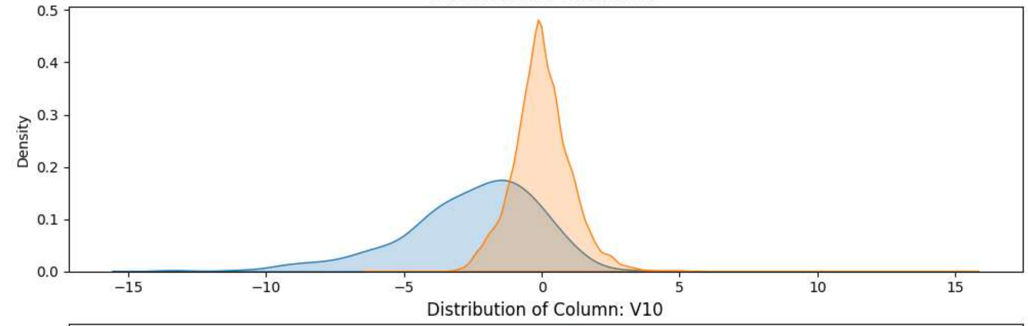

Distribution of Column: V5
Density
0.40
0.35
0.30
0.25
0.20

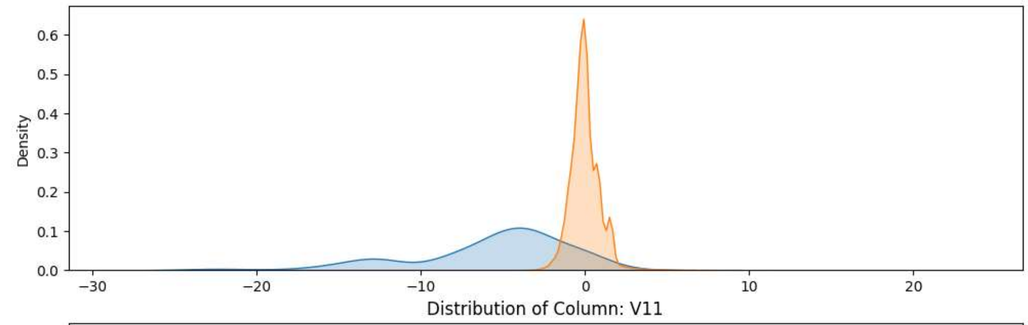

Distribution of Column: V6

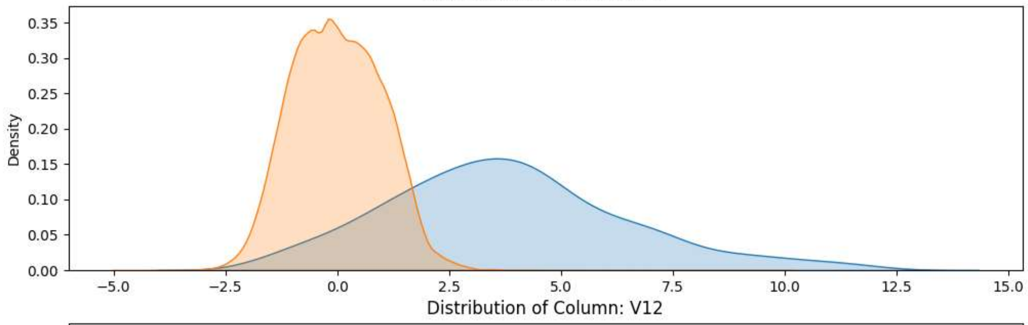

Density
Distribution of Column: V7

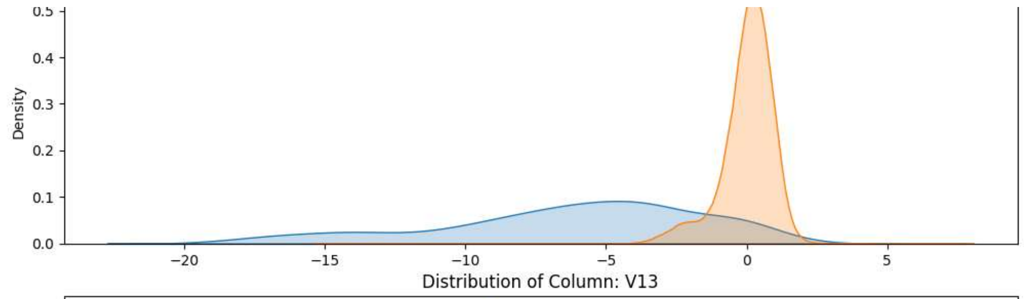

Density
Distribution of Column: V8

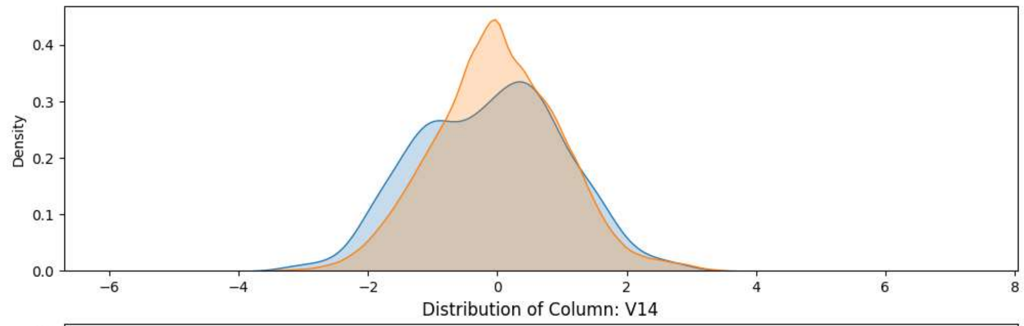

Density
Distribution of Column: V9

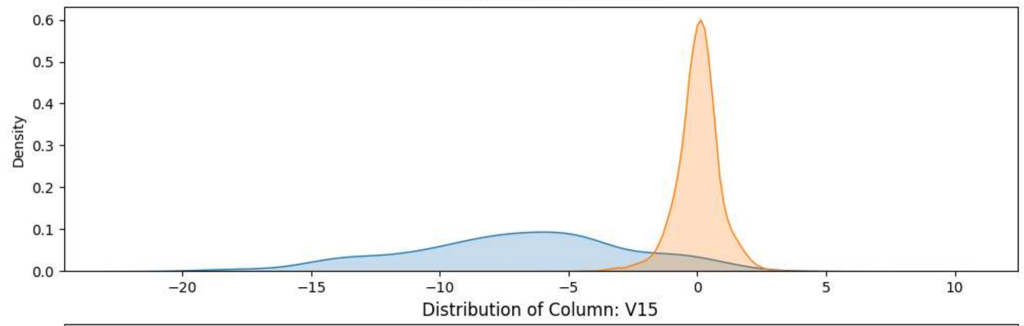

Density
Distribution of Column: V10

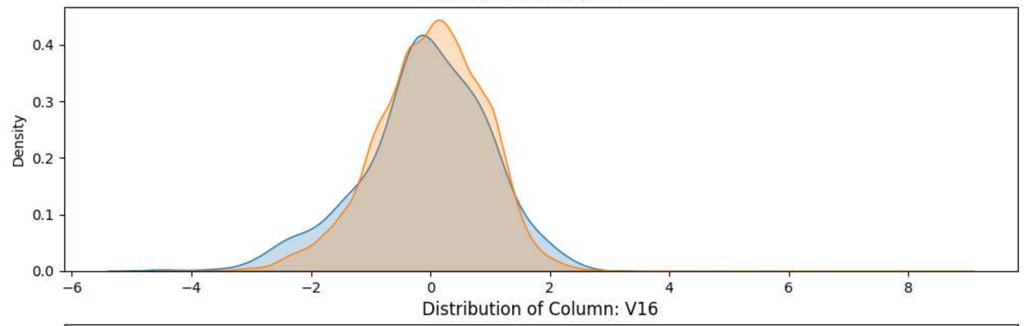

Density
Distribution of Column: V11

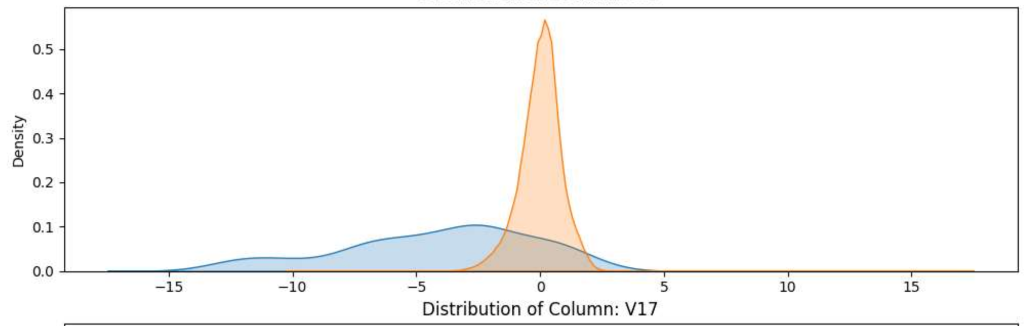

Density
Distribution of Column: V12

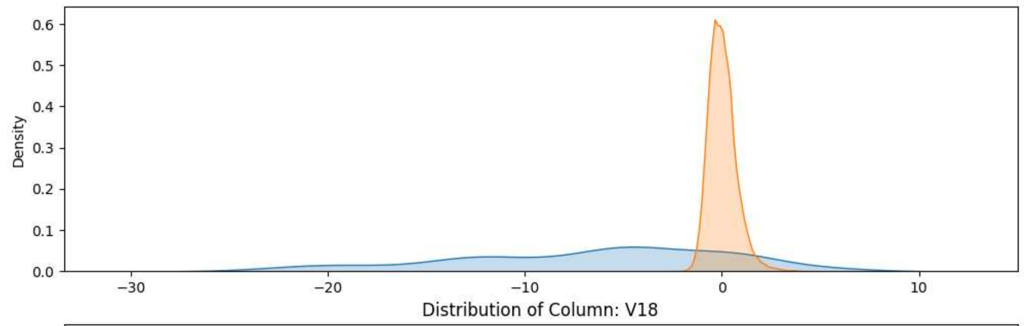

Density
Distribution of Column: V13

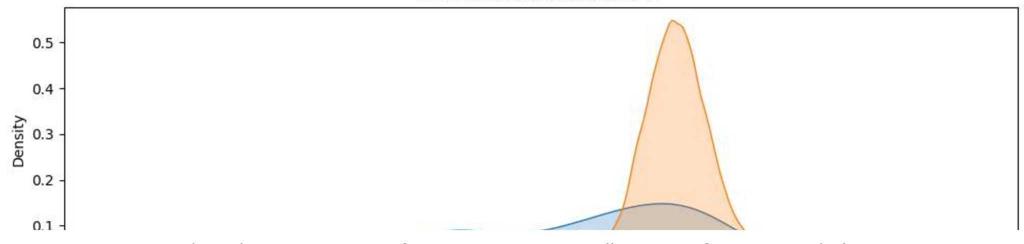

Density
Distribution of Column: V14

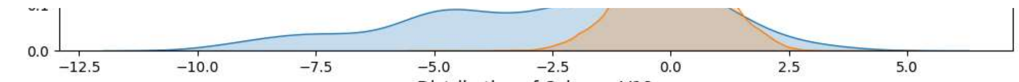

Density
Distribution of Column: V15


Density
Distribution of Column: V16

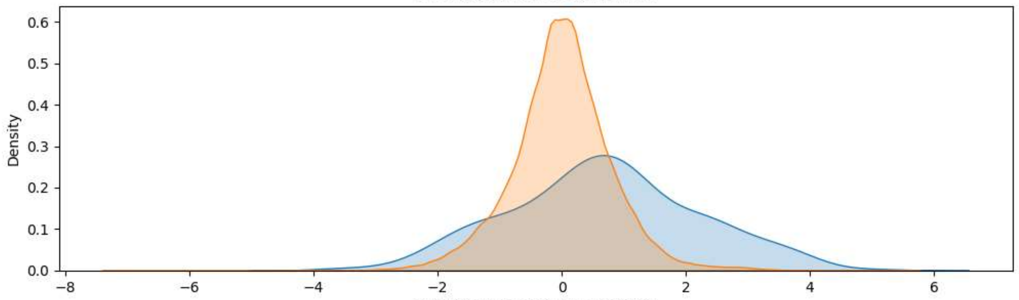

Density
Distribution of Column: V17


Density
Distribution of Column: V18

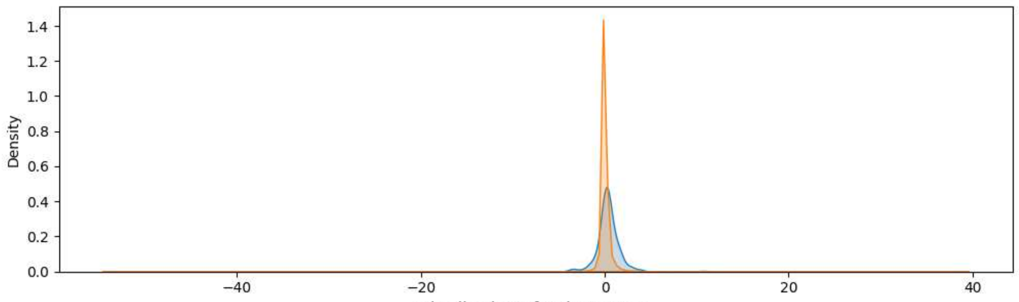

Density



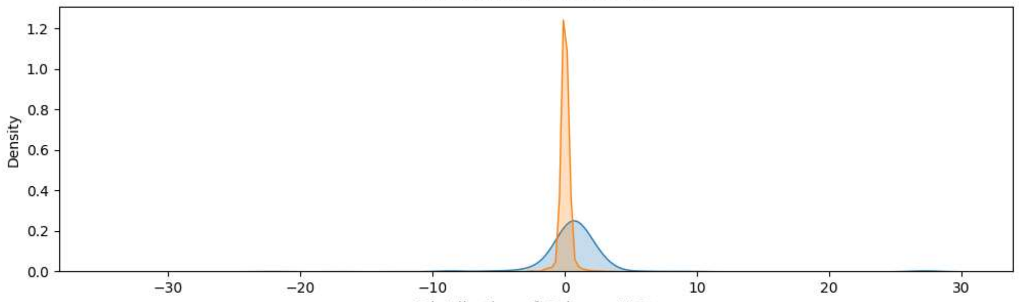
Distribution of Column: V19



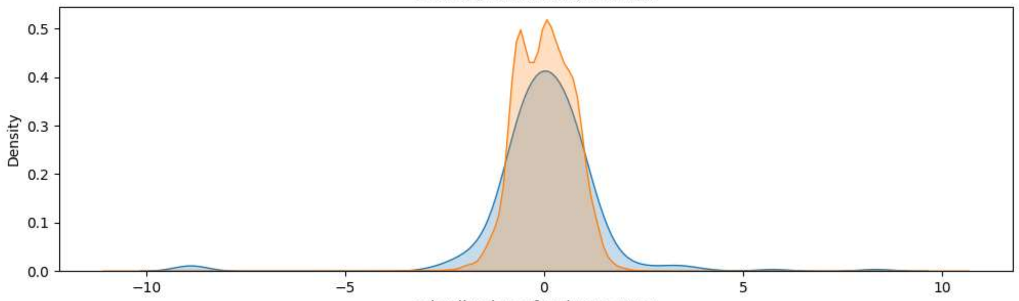
Distribution of Column: V20



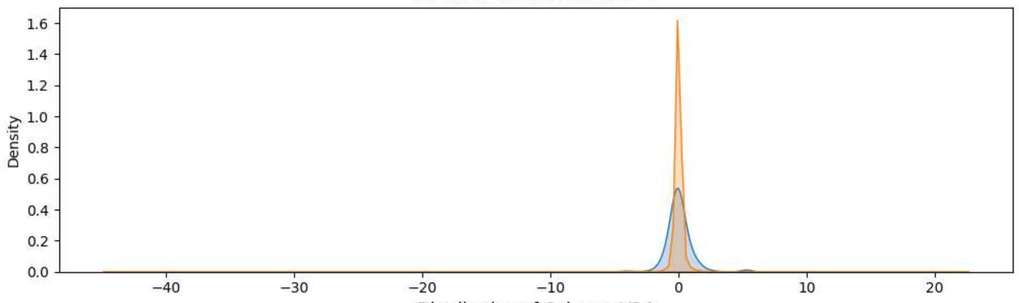
Distribution of Column: V21



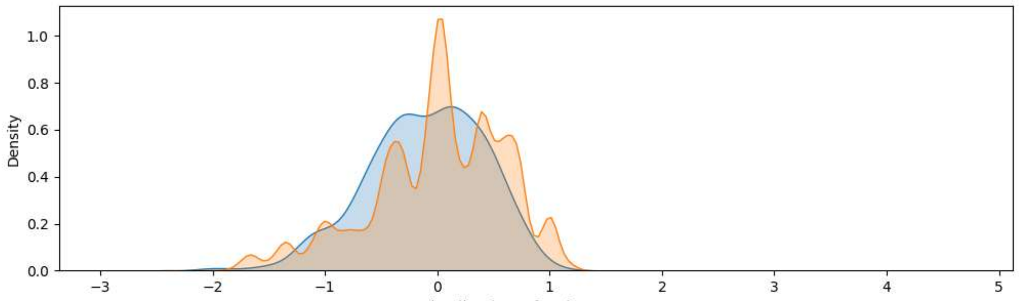
Distribution of Column: V22



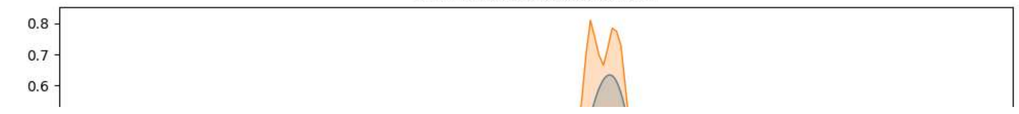
Distribution of Column: V23

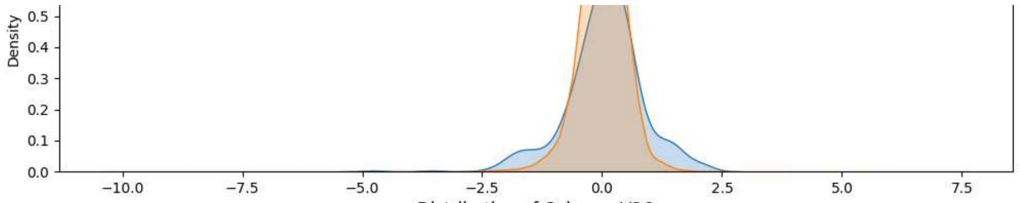


Distribution of Column: V24

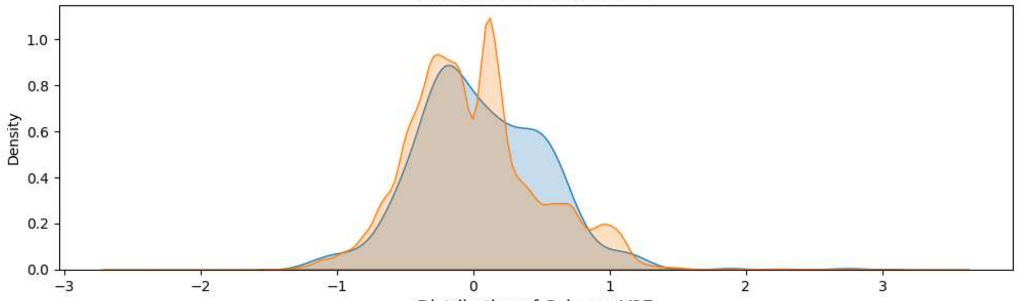


Distribution of Column: V25

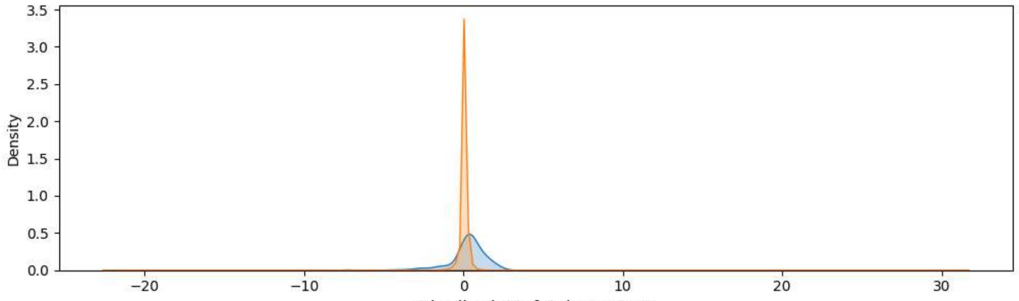




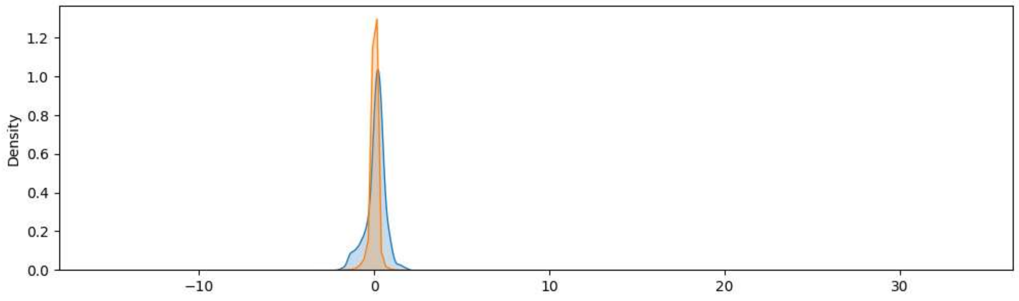
Distribution of Column: V26



Distribution of Column: V27



Distribution of Column: V28



Data Preparation

Since the features are created using PCA, feature selection is unnecessary as many features are tiny. Let's see if there are any missing values in the dataset:

```
# check for null values
dataset.isnull().shape[0]
print("Non-missing values: " + str(dataset.isnull().shape[0]))
print("Missing values: " + str(dataset.shape[0] - dataset.isnull().shape[0]))
```

```
Non-missing values: 284807
Missing values: 0
```

As there are no missing data, we turn to standardization. We standardize only Time and Amount using RobustScaler:

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler().fit(dataset[["Time", "Amount"]])
dataset[["Time", "Amount"]] = scaler.transform(dataset[["Time", "Amount"]])

dataset.head().append(dataset.tail())
```

```
<ipython-input-13-0c03c5a915c0>:5: FutureWarning: The frame.append method is deprecated and will be
  dataset.head().append(dataset.tail())
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533
284802	1.034951	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334
284803	1.034963	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869
284804	1.034975	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417
284805	1.034975	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145
284806	1.035022	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650

10 rows × 31 columns

Next, let's divide the data into features and targets. We also make the train-test split of the data:

```
# Separate response and features  Undersampling before cross validation will lead to overfiting
y = dataset["Class"] # target
X = dataset.iloc[:,0:30]

# Use SKLEARN for the split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size = 0.2, random_state = 42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((227845, 30), (56962, 30), (227845,), (56962,))
```

```
# Create the cross validation framework
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV, cross_val_score, RandomizedSearchCV

kf = StratifiedKFold(n_splits=5, random_state = None, shuffle = False)
```

```
# Import the imbalance Learn module
from imblearn.pipeline import make_pipeline ## Create a Pipeline using the provided estimators .
from imblearn.under_sampling import NearMiss  ## perform Under-sampling  based on NearMiss methods.
from imblearn.over_sampling import SMOTE  ## PerformOver-sampling class that uses SMOTE.
# import the metrics
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, recall_score, precision_score, f1_score
# Import the classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Building and Training the model

Let's run RandomForestClassifier on the dataset and see the performance:

```
# Fit and predict
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)

# For the performance let's use some metrics from SKLEARN module
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print("The accuracy is", accuracy_score(y_test, y_pred))
print("The precision is", precision_score(y_test, y_pred))
print("The recall is", recall_score(y_test, y_pred))
print("The F1 score is", f1_score(y_test, y_pred))
```

```
The accuracy is 0.9995962220427653
The precision is 0.9746835443037974
The recall is 0.7857142857142857
The F1 score is 0.8700564971751412
```

As we can see, we had only 0.17% fraud transactions, and a model predicting all transactions to be valid would have an accuracy of 99.83%. Luckily, our model exceeded that to over 99.96%.

As a result, accuracy isn't a suitable metric for our problem. There are three more:

- **Precision**: It is the total number of true positives divided by the true positives and false positives. Precision makes sure we don't spot good transactions as fraudulent in our problem.
- **Recall**: It is the total number of true positives divided by the true positives and false negatives. Recall assures we don't predict fraudulent transactions as all good and therefore get good accuracy with a terrible model.
- **F1 Score**: It is the harmonic mean of precision and recall. It makes a good average between both metrics.

The recall is more important than precision in our problem, as predicting a fraudulent transaction as good is worse than marking a good transaction as fraudulent, you can use fbeta_score() and adjust the beta parameter to make it more weighted towards recall.

Undersampling

In this section, we will perform undersampling to our dataset. One trivial point to note is that we will not undersample the testing data as we want our model to perform well with skewed class distributions.

The steps are as follows:

- Use a 5-fold cross-validation on the training set.
- On each of the folds, use undersampling.
- Fit the model on the training folds and validate on the validation fold.

NearMiss Methods

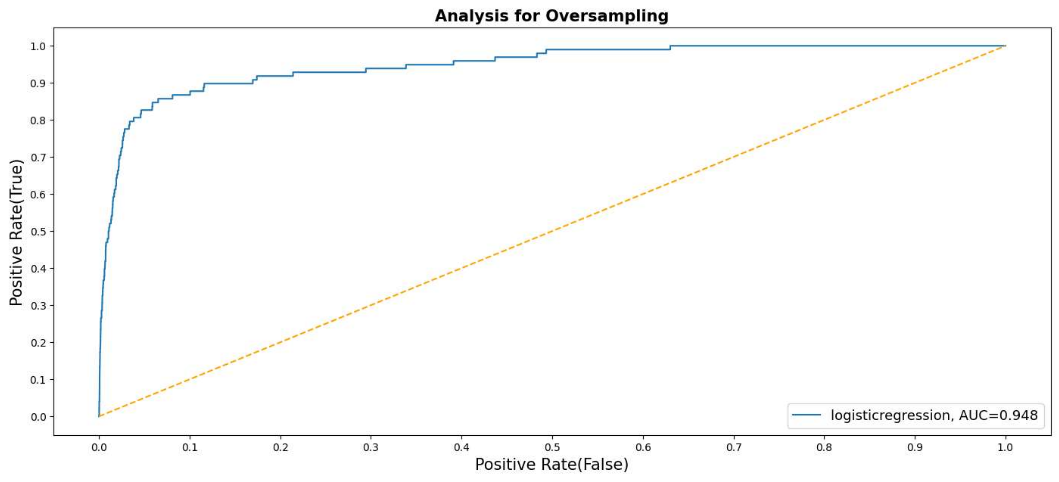
```
# Performing Near Miss Method
def get_model_best_estimator_and_metrics(estimator, params, kf=kf, X_train=X_train,
                                         y_train=y_train, X_test=X_test,
                                         y_test=y_test, is_grid_search=True,
                                         sampling=NearMiss(), scoring="f1",
                                         n_jobs=2):
    if sampling is None:

        pipeline = make_pipeline(estimator)
    else:

        pipeline = make_pipeline(sampling, estimator)

    estimator_name = estimator.__class__.__name__.lower()
    new_params = {f'{estimator_name}__{key}': params[key] for key in params}
    if is_grid_search:

        search = GridSearchCV(pipeline, param_grid=new_params, cv=kf, return_train_score=True, n_jobs=n_jobs, verbose=2)
    else:

        search = RandomizedSearchCV(pipeline, param_distributions=new_params,
                                    cv=kf, scoring=scoring, return_train_score=True,
                                    n_jobs=n_jobs, verbose=1)

    search.fit(X_train, y_train)
    cv_score = cross_val_score(search, X_train, y_train, scoring=scoring, cv=kf)

    y_pred = search.best_estimator_.named_steps[estimator_name].predict(X_test)

    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    y_proba = search.best_estimator_.named_steps[estimator_name].predict_proba(X_test)[::, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc = roc_auc_score(y_test, y_proba)

    return {
        "best_estimator": search.best_estimator_,
        "estimator_name": estimator_name,
        "cv_score": cv_score,
        "recall": recall,
        "accuracy": accuracy,
        "f1_score": f1,
        "fpr": fpr,
        "tpr": tpr,
        "auc": auc,
    }
```

```
## Create the dataframe
res_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

logreg_us_results = get_model_best_estimator_and_metrics(
    estimator=LogisticRegression(),
    params={"penalty": ['l1', 'l2'],
              'C': [ 0.01, 0.1, 1, 100],
              'solver' : ['liblinear']},
    sampling=NearMiss(),
)
print(f"==={logreg_us_results['estimator_name']}===")
print("Model:", logreg_us_results['best_estimator'])
print("Accuracy:", logreg_us_results['accuracy'])
print("Recall:", logreg_us_results['recall'])
print("F1 Score:", logreg_us_results['f1_score'])
res_table = res_table.append({'classifiers': logreg_us_results["estimator_name"],
                                        'fpr': logreg_us_results["fpr"],
                                        'tpr': logreg_us_results["tpr"],
                                        'auc': logreg_us_results["auc"]
                              }, ignore_index=True)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
===logisticregression===
Model: Pipeline(steps=[('nearmiss', NearMiss()),
                ('logisticregression',
                 LogisticRegression(C=0.1, penalty='l1', solver='liblinear'))])
Accuracy: 0.8124012499561111
Recall: 0.9183673469387755
F1 Score: 0.016565433462175594
<ipython-input-19-cc5b683ef9c2>:16: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  res_table = res_table.append({'classifiers': logreg_us_results["estimator_name"],
```

```
# Plot the ROC curve for undersampling
res_table.set_index('classifiers', inplace=True)
fig = plt.figure(figsize=(17,7))

for j in res_table.index:
    plt.plot(res_table.loc[j]['fpr'],
             res_table.loc[j]['tpr'],
             label="{}, AUC={:.3f}".format(j, res_table.loc[j]['auc']))
```

```
plt.plot([0,1], [0,1], color='orange', linestyle='--')
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Positive Rate(False)", fontsize=15)
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("Positive Rate(True)", fontsize=15)
plt.title('Analysis for Oversampling', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')
plt.show()
```



Oversampling with SMOTE

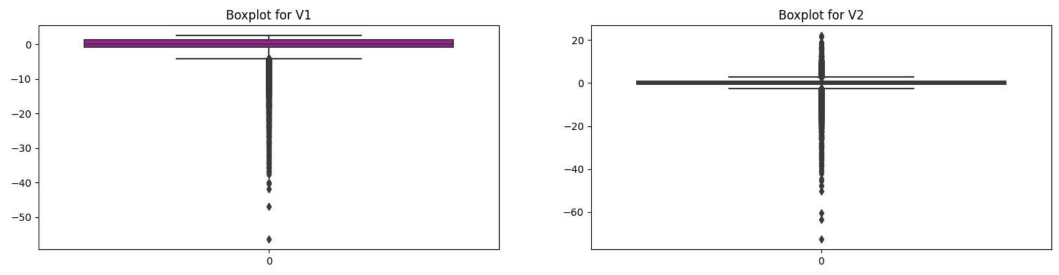
```
# Cumulatively create a table for the ROC curve
res_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

lin_reg_os_results = get_model_best_estimator_and_metrics(
    estimator=LogisticRegression(),
    params={"penalty": ['l1', 'l2'], 'C': [ 0.01, 0.1, 1, 100, 100],
            'solver' : ['liblinear']},
    sampling=SMOTE(random_state=42),
    scoring="f1",
    is_grid_search=False,
    n_jobs=2,
)
print(f"==={lin_reg_os_results['estimator_name']}===")
print("Model:", lin_reg_os_results['best_estimator'])
print("Accuracy:", lin_reg_os_results['accuracy'])
print("Recall:", lin_reg_os_results['recall'])
print("F1 Score:", lin_reg_os_results['f1_score'])
res_table = res_table.append({'classifiers': lin_reg_os_results["estimator_name"],
                                        'fpr': lin_reg_os_results["fpr"],
                                        'tpr': lin_reg_os_results["tpr"],
                                        'auc': lin_reg_os_results["auc"]
                              }, ignore_index=True)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
===logisticregression===
Model: Pipeline(steps=[('smote', SMOTE(random_state=42)),
                ('logisticregression',
                 LogisticRegression(C=0.01, penalty='l1', solver='liblinear'))])
Accuracy: 0.9759664337628594
Recall: 0.9183673469387755
```

```
F1 Score: 0.11620400258231117
<ipython-input-21-cdcc3bcf3229>:18: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  res_table = res_table.append({'classifiers': lin_reg_os_results["estimator_name"],
```

Appendix: Outlier Detection and Removal

```
# boxplot for two example variables in the dataset

f, axes = plt.subplots(1, 2, figsize=(18,4), sharex = True)

variable1 = dataset["V1"]
variable2 = dataset["V2"]

sns.boxplot(variable1, color="m", ax=axes[0]).set_title('Boxplot for V1')
sns.boxplot(variable2, color="m", ax=axes[1]).set_title('Boxplot for V2')

plt.show()
```



```
# Find the IQR for all the feature variables
# Please note that we are keeping Class variable also in this evaluation, though we know using this method no observation
# be removed based on this variable.

quartile1 = dataset.quantile(0.25)
quartile3 = dataset.quantile(0.75)

IQR = quartile3 - quartile1
print(IQR)
```

```
Time      1.000000
V1        2.236015
V2        1.402274
V3        1.917560
V4        1.591981
V5        1.303524
V6        1.166861
V7        1.124512
V8        0.535976
V9        1.240237
V10       0.989349
V11       1.502088
V12       1.023810
V13       1.311044
V14       0.918724
V15       1.231705
V16       0.991333
V17       0.883423
V18       0.999657
V19       0.915248
V20       0.344762
V21       0.414772
V22       1.070904
V23       0.309488
V24       0.794113
V25       0.667861
V26       0.567936
V27       0.161885
V28       0.131240
Amount    1.000000
Class     0.000000
dtype: float64
```

```
# Remove the outliers
constant = 3
datavalid = dataset[~((dataset < (quartile1 - constant * IQR)) |(dataset > (quartile3 + constant * IQR))).any(axis=1)]
deletedrows = dataset.shape[0] - datavalid.shape[0]
print("We have removed " + str(deletedrows) + " rows from the data as outliers")
```

```
We have removed 53376 rows from the data as outliers
```

Conclusion

In conclusion, the project focused on the critical area of credit card fraud detection, a pressing concern in the modern financial landscape. The primary objective of this project was to develop and evaluate various machine learning models and techniques to effectively identify fraudulent transactions and enhance the security of credit card transactions. Through extensive research, data preprocessing, feature engineering, and model evaluation, we have made several key findings and achievements in the field of credit card fraud detection.

Summary

- This credit card fraud detection project aimed to develop an effective fraud detection system by employing a combination of data preprocessing techniques, model building, and advanced sampling methods. The project followed a structured approach with the following key steps:

1. Importing Models and Visualization: We initiated the project by importing essential Python libraries for data manipulation, visualization, and machine learning. Visualization tools like Matplotlib and Seaborn were employed to gain insights into the data distribution and characteristics.
2. Data Preparation: Data preparation was a crucial step to ensure the dataset's quality and compatibility with machine learning models. We handled missing values, removed duplicates, and standardized feature scales as part of this phase. Exploratory Data Analysis (EDA) was also performed to understand the data's patterns and distributions.
3. Building and Training the Models: We constructed a variety of machine learning models, including logistic regression, decision trees, random forests. Each model was trained on the preprocessed data and fine-tuned for optimal performance.
4. UnderSampling: Addressing class imbalance is a fundamental concern in credit card fraud detection. We implemented undersampling techniques to reduce the dominance of the majority class. This helped in training models to recognize fraudulent transactions more effectively by ensuring a balanced dataset.
5. Oversampling with SMOTE: To further enhance the model's ability to detect fraud, we leveraged the Synthetic Minority Over-sampling Technique (SMOTE) to oversample the minority class. SMOTE generated synthetic examples, improving the model's capacity to capture rare fraudulent patterns.
6. Appendix: Outlier Detection and Removal: In an appendix, we discussed the importance of identifying and dealing with outliers in the dataset. Outliers can distort the performance of machine learning models and lead to inaccurate predictions. We explored outlier detection techniques and discussed their potential impact on the fraud detection process.

In summary, this credit card fraud detection project involved a comprehensive approach that encompassed data preparation, model building, and advanced sampling methods. By addressing class imbalance through undersampling and oversampling techniques like SMOTE, we aimed to develop a robust fraud detection system capable of accurately identifying fraudulent transactions while minimizing false alarms. Additionally, the discussion on outlier detection highlighted the importance of data quality in building a reliable fraud detection model. This project contributes to the ongoing efforts to enhance the security of credit card transactions and protect consumers and financial institutions from fraudulent activities.