Regression and Time Series Analysis Course Project

"Credit Card Fraud Detection"

Name	Net ID
Ronit Gandhi	rg1225
Ganesh Arpan Nookala	gn178
Kamala Prerna Nookala	kn491
Aditya Singhal	as4622
Hang Wang	hw598

Date: - December 12, 2023

Contents:

Preface

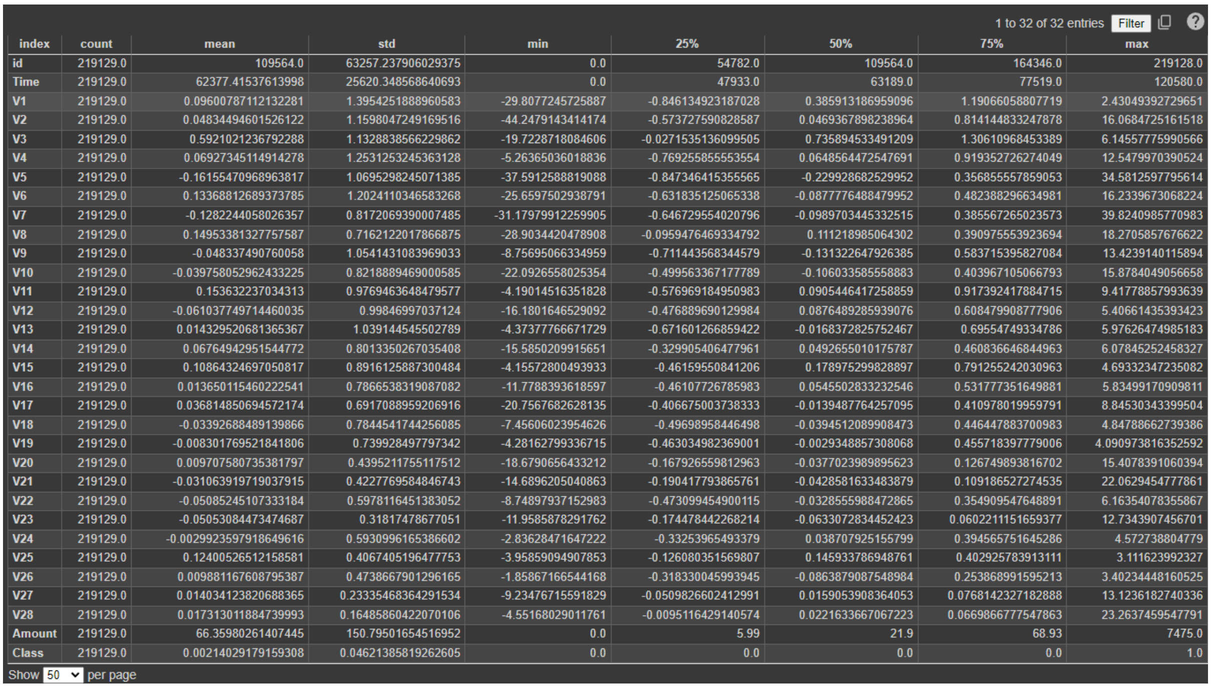
It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

| “The greatest value of a picture is when it forces us to notice what we never expected to see.” **John W. Tukey**

Dataset Description

The dataset comprises of credit card transactions over a two-day period in September 2013, with 492 fraud cases out of 284,807 transactions, resulting in a highly imbalanced dataset (0.172% fraud instances). Features V1 to V28 are derived from PCA transformation, while 'Time' represents elapsed seconds since the first transaction, and 'Amount' denotes the transaction amount. Due to confidentiality constraints, original features and additional background information are undisclosed. The 'Class' feature, indicating fraud (1) or non-fraud (0), serves as the response variable.

Dataset Overview:



index	count	mean	std	min	25%	50%	75%	max
id	219129.0	109564.0	63257.237906029375	0.0	54782.0	109564.0	164346.0	219128.0
Time	219129.0	62377.41537613998	25620.348568640693	0.0	47933.0	63189.0	77519.0	120580.0
V1	219129.0	0.09600787112132281	1.3954251888960583	-29.8077245725887	-0.846134923187028	0.385913186959096	1.19066058807719	2.43049392729651
V2	219129.0	0.04834494601526122	1.1598047249169516	-44.2479143414174	-0.573727590828587	0.0469367898238964	0.814144833247878	16.0684725161518
V3	219129.0	0.5921021236792288	1.1328838566229862	-19.7228718084606	-0.0271535136099505	0.735894533491209	1.30610968453389	6.14557775990566
V4	219129.0	0.06927345114914278	1.2531253245363128	-5.26365036018836	-0.769255855553554	0.0648564472547691	0.919352726274049	12.5479970390524
V5	219129.0	-0.16155470968963817	1.0695298245071385	-37.5912588819088	-0.847346415355565	-0.229928682529952	0.356855557859053	34.5812597795614
V6	219129.0	0.13368812689373785	1.2024110346583268	-25.6597502938791	-0.631835125065338	-0.0877776488479952	0.482388296634981	16.2339673068224
V7	219129.0	-0.1282244058026357	0.8172069390007485	-31.17979912259905	-0.646729554020796	-0.0989703445332515	0.385567265023573	39.8240985770983
V8	219129.0	0.14953381327757587	0.7162122017866875	-28.9034420478908	-0.0959476469334792	0.111218985064302	0.390975553923694	18.2705857676622
V9	219129.0	-0.048337490760058	1.0541431083969033	-8.75695066334959	-0.711443568344579	-0.131322647926385	0.583715395827084	13.4239140115894
V10	219129.0	-0.039758052962433225	0.8218889469000585	-22.0926558025354	-0.499563367177789	-0.106033585558883	0.403967105066793	15.8784049056658
V11	219129.0	0.153632237034313	0.9769463648479577	-4.19014516351828	-0.576969184950983	0.0905446417258859	0.917392417884715	9.41778857993639
V12	219129.0	-0.061037749714460035	0.99846997037124	-16.1801646529092	-0.476889690129984	0.0876489285939076	0.608479908777906	5.40661435393423
V13	219129.0	0.014329520681365367	1.039144545502789	-4.37377766671729	-0.671601266859422	-0.0168372825752467	0.69554749334786	5.97626474985183
V14	219129.0	0.06764942951544772	0.8013350267035408	-15.5850209915651	-0.329905406477961	0.0492655010175787	0.460836646844963	6.07845252458327
V15	219129.0	0.10864324697050817	0.8916125887300484	-4.15572800493933	-0.46159550841206	0.178975299828897	0.791255242030963	4.69332347235082
V16	219129.0	0.013650115460222541	0.7866538319087082	-11.7788393618597	-0.46107726785983	0.0545502833232546	0.531777351649881	5.83499170909811
V17	219129.0	0.036814850694572174	0.6917088959206916	-20.7567682628135	-0.406675003738333	-0.0139487764257095	0.410978019959791	8.84530343399504
V18	219129.0	-0.03392688489139866	0.7844541744256085	-7.45606023954626	-0.49698958446498	-0.0394512089908473	0.446447883700983	4.84788662739386
V19	219129.0	-0.008301769521841806	0.739928497797342	-4.28162799336715	-0.463034982369001	-0.0029348857308068	0.455718397779006	4.090973816352592
V20	219129.0	0.009707580735381797	0.4395211755117512	-18.6790656433212	-0.167926559812963	-0.0377023989895623	0.126749893816702	15.4078391060394
V21	219129.0	-0.031063919719037915	0.4227769584846743	-14.6896205040863	-0.190417793865761	-0.0428581633483879	0.109186527274535	22.0629454777861
V22	219129.0	-0.05085245107333184	0.5978116451383052	-8.74897937152983	-0.473099454900115	-0.0328555988472865	0.354909547648891	6.16354078355867
V23	219129.0	-0.05053084473474687	0.31817478677051	-11.9585878291762	-0.174478442268214	-0.0633072834452423	0.0602211151659377	12.7343907456701
V24	219129.0	-0.0029923597918649616	0.5930996165386602	-2.83628471647222	-0.33253965493379	0.038707925155799	0.394565751645286	4.572738804779
V25	219129.0	0.12400526512158581	0.4067405196477753	-3.95859094907853	-0.126080351569807	0.145933786948761	0.402925783913111	3.111623992327
V26	219129.0	0.009881167608795387	0.4738667901296165	-1.85867166544168	-0.318330045993945	-0.0863879087548984	0.253868991595213	3.40234448160525
V27	219129.0	0.014034123820688365	0.23335468364291534	-9.23476715591829	-0.0509826602412991	0.0159053908364053	0.0768142327182888	13.1236182740336
V28	219129.0	0.017313011884739993	0.16485860422070106	-4.55168029011761	-0.0095116429140574	0.0221633667067223	0.0669866777547863	23.2637459547791
Amount	219129.0	66.35980261407445	150.79501654516952	0.0	5.99	21.9	68.93	7475.0
Class	219129.0	0.00214029179159308	0.04621385819262605	0.0	0.0	0.0	0.0	1.0

- The transaction amount is relatively small. The mean of all the amounts is approximately USD 66.
- There are no "Null" values, so we don't have to work on ways to replace values.
- Most of the transactions were Non-Fraud (99.83%) of the time, while Fraud transactions occurs (0.17%) of the time in the dataframe.
- The target or response variable ‘Class’ has a lesser number of ‘bad’ values, i.e., only 0.172% instances are fraudulent.
- This is where SMOTE (Synthetic Minority Over-Sampling Technique) comes into picture, we apply SMOTE to the dataset to overcome the imbalance

Data Preprocessing:

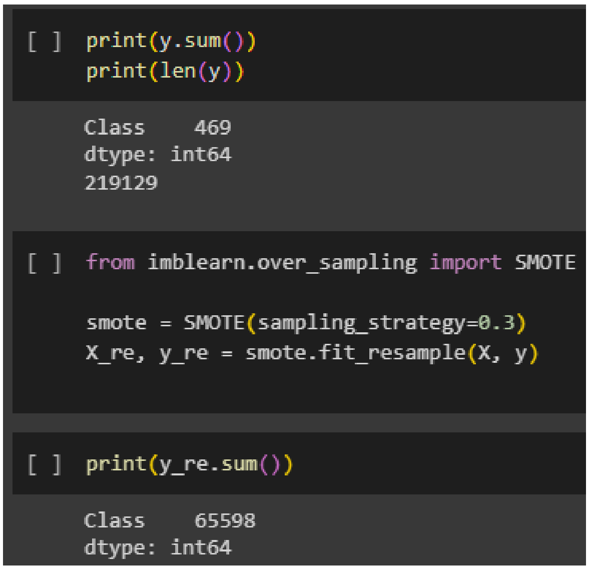
Synthetic Minority Over-sampling Technique, or SMOTE for short, is a preprocessing technique used to address a class imbalance in a dataset.

In the real world, oftentimes we end up trying to train a model on a dataset with very few examples of a given class (e.g. rare disease diagnosis, manufacturing defects, fradulent transactions) which results in poor performance. Due to the nature of the data (occurrences are so rare), it's not always realistic to go out and acquire more. One way of solving this issue is to under-sample the majority class. That is to say, we would exclude rows corresponding to the majority class such that there are roughly the same amount of rows for both the majority and minority classes. However, in doing so, we lose out on a lot of data that could be used to train our model thus improving its accuracy (e.g. higher bias). Another other option is to over-sample the minority class. In other words, we randomly duplicate observations of the minority class. The problem with this approach is that it leads to overfitting because the model learns from the same examples. This is where SMOTE comes in. At a high level, the SMOTE algorithm can be described as follows:

- Take difference between a sample and its nearest neighbour
- Multiply the difference by a random number between 0 and 1
- Add this difference to the sample to generate a new synthetic example in feature space
- Continue on with next nearest neighbour up to user-defined number

Number of 'bad' values before applying SMOTE = 469

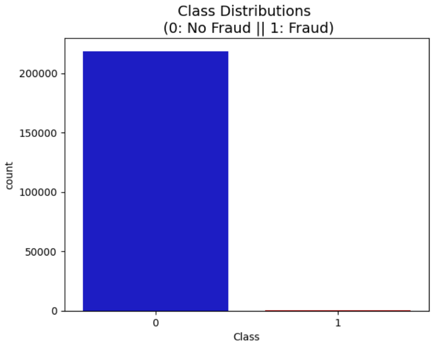
Number of 'bad' values after applying SMOTE = 65598



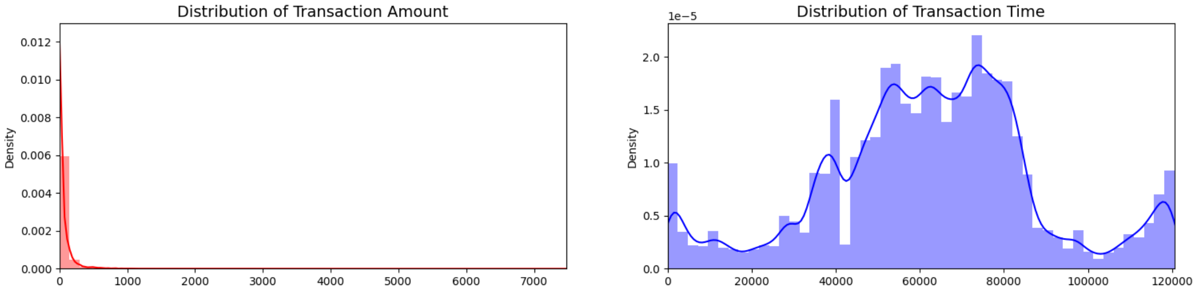
```
[ ]  print(y.sum())
     print(len(y))

     Class    469
     dtype: int64
     219129

[ ]  from imblearn.over_sampling import SMOTE

     smote = SMOTE(sampling_strategy=0.3)
     X_re, y_re = smote.fit_resample(X, y)

[ ]  print(y_re.sum())

     Class    65598
     dtype: int64
```

Exploratory Data Analysis:

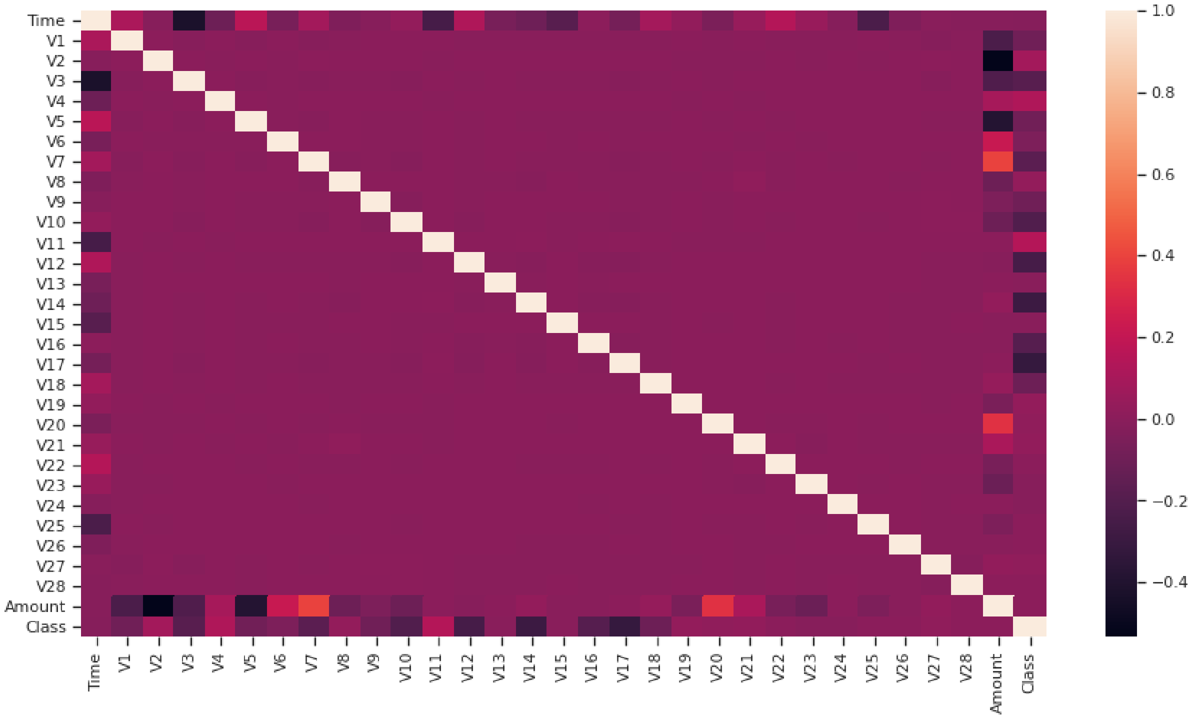
Note: Notice how imbalanced our original dataset is! Most of the transactions are non-fraud. If we use this dataframe as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most transactions are not fraud. But we don't want our model to assume, we want our model to detect patterns that give signs of fraud!



Distributions: By seeing the distributions we can have an idea how skewed are these features, we can also see further distributions of the other features.



Correlation Heatmap (Pearson Correlation):



Feature Engineering:

Weight of Evidence:

The Weight of Evidence (WOE) tells the predictive power of an independent variable in relation to the dependent variable. Since it evolved from the credit scoring world, it is generally described as a measure of the separation of good and bad customers. "Bad Customers" refers to the customers who defaulted on a loan. and "Good Customers" refers to the customers who paid back loan.

$$\text{WOE} = \ln\left(\frac{\text{Distribution of Goods}}{\text{Distribution of Bads}}\right)$$

It's good to understand the concept of WOE in terms of events and non-events. It is calculated by taking the natural logarithm (log to base e) of division of % of non-events and % of events.

$$\text{WOE} = \ln\left(\frac{\text{\% of non-events}}{\text{\% of events}}\right)$$

Steps to calculate WOE:

- For a continuous variable, split data into 10 parts (or lesser depending on the distribution). (For a categorical variable, you do not need to split the data)
- Calculate the number of events and non-events in each group (bin).
- Calculate the % of events and % of non-events in each group.
- Calculate WOE by taking natural log of division of % of non-events and % of events.

Checking for correct binning with WOE:

1. The WOE should be monotonic i.e. either growing or decreasing with the bins. We plot WOE values and check linearity on the graph.
2. Performing the WOE transformation after binning.

Information Value:

Information value (IV) is one of the most useful technique to select important variables in a predictive model. It helps to rank variables on the basis of their importance. The IV is calculated using the following formula :

$$\text{IV} = \sum (\text{\% of non-events} - \text{\% of events}) * \text{WOE}$$

Interpreting IV values:

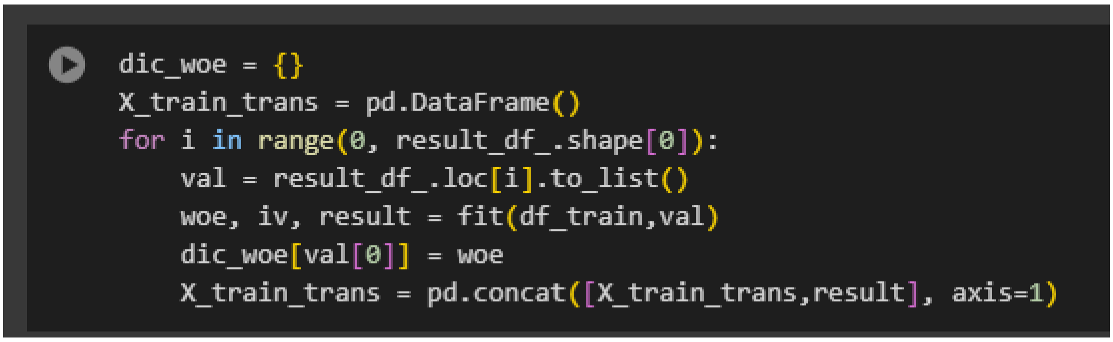
If the IV statistic is:

- Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)
- 0.02 to 0.1, then the predictor has only a weak relationship to the Goods/Bads odds ratio
- 0.1 to 0.3, then the predictor has a medium strength relationship to the Goods/Bads odds ratio
- 0.3 to 0.5, then the predictor has a strong relationship to the Goods/Bads odds ratio.
- > 0.5, suspicious relationship (Check once)

In our model, for selecting the best features we will be using the attributes with IV > 0.3.

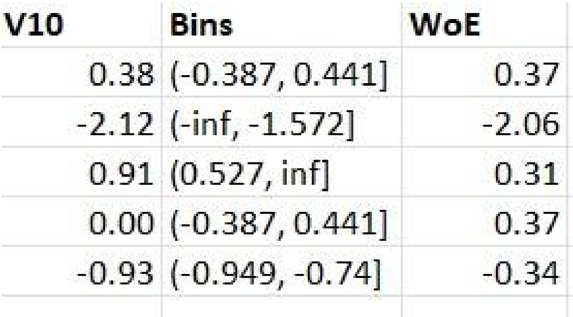
Methodology:

Modelling:



```
dic_woe = {}
X_train_trans = pd.DataFrame()
for i in range(0, result_df_.shape[0]):
    val = result_df_.loc[i].to_list()
    woe, iv, result = fit(df_train,val)
    dic_woe[val[0]] = woe
    X_train_trans = pd.concat([X_train_trans,result], axis=1)
```

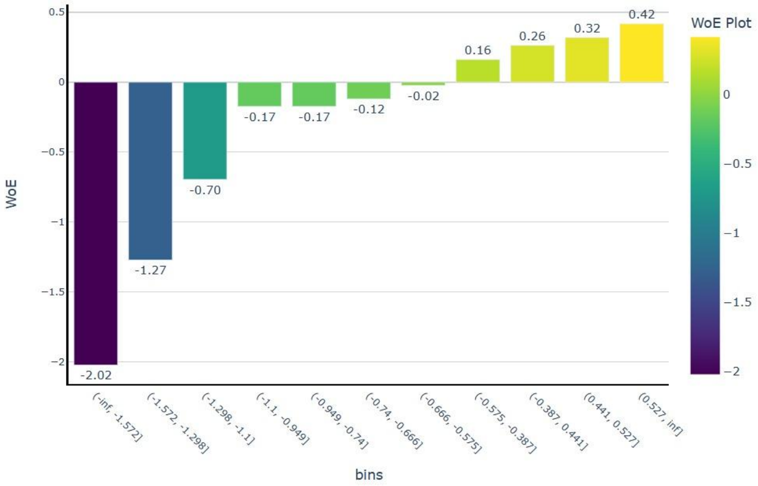
After we successfully identify the features we transform the train and test data before feeding it to the Logistic Regression model.



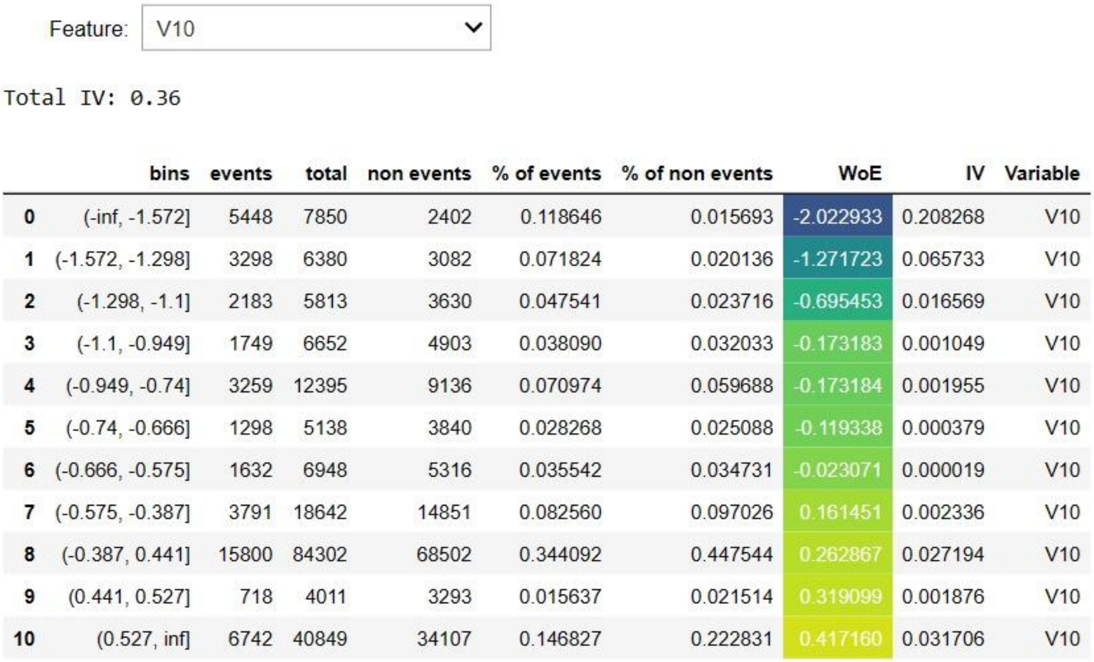
V10	Bins	WoE
0.38	(-0.387, 0.441]	0.37
-2.12	(-inf, -1.572]	-2.06
0.91	(0.527, inf]	0.31
0.00	(-0.387, 0.441]	0.37
-0.93	(-0.949, -0.74]	-0.34

In the above figure we can see the values for the feature V10 before binning then there is a binning range and finally the WOE values are obtained.

WOE Plot:



WOE Dataframe:


Feature: V10

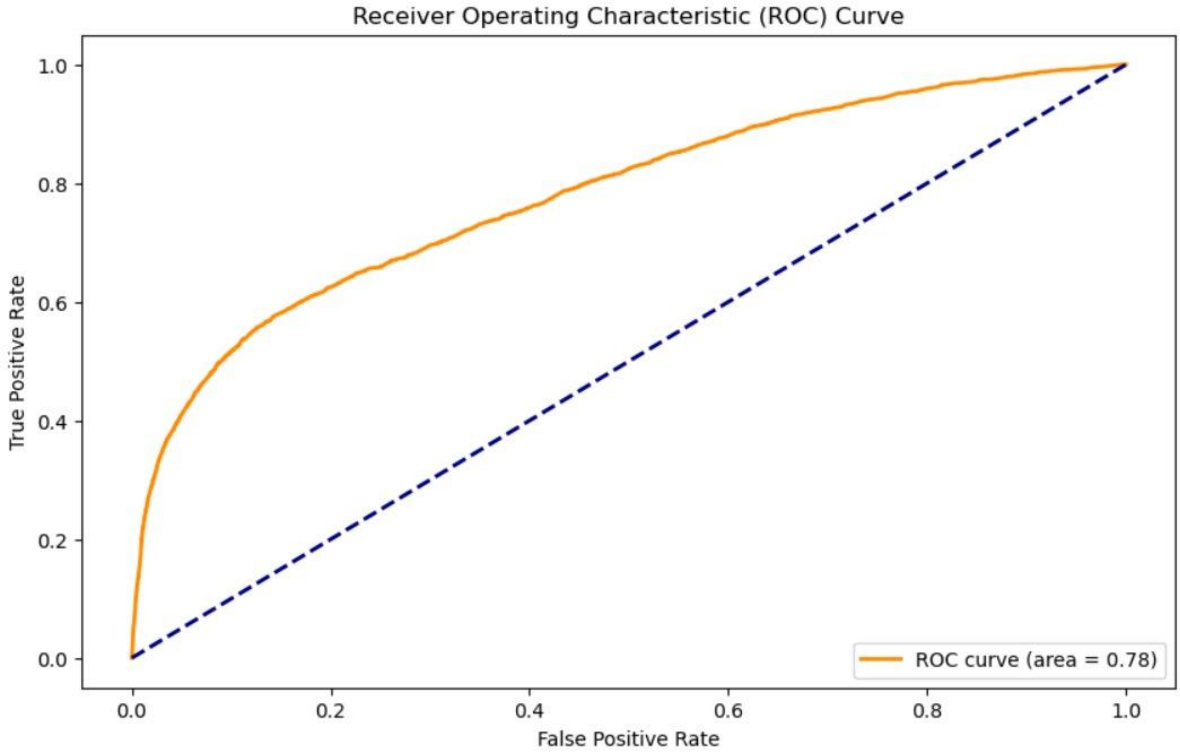
```
Total IV: 0.36
```

	bins	events	total	non events	% of events	% of non events	WoE	IV	Variable
0	(-inf, -1.572]	5448	7850	2402	0.118646	0.015693	-2.022933	0.208268	V10
1	(-1.572, -1.298]	3298	6380	3082	0.071824	0.020136	-1.271723	0.065733	V10
2	(-1.298, -1.1]	2183	5813	3630	0.047541	0.023716	-0.695453	0.016569	V10
3	(-1.1, -0.949]	1749	6652	4903	0.038090	0.032033	-0.173183	0.001049	V10
4	(-0.949, -0.74]	3259	12395	9136	0.070974	0.059688	-0.173184	0.001955	V10
5	(-0.74, -0.666]	1298	5138	3840	0.028268	0.025088	-0.119338	0.000379	V10
6	(-0.666, -0.575]	1632	6948	5316	0.035542	0.034731	-0.023071	0.000019	V10
7	(-0.575, -0.387]	3791	18642	14851	0.082560	0.097026	0.161451	0.002336	V10
8	(-0.387, 0.441]	15800	84302	68502	0.344092	0.447544	0.262867	0.027194	V10
9	(0.441, 0.527]	718	4011	3293	0.015637	0.021514	0.319099	0.001876	V10
10	(0.527, inf]	6742	40849	34107	0.146827	0.222831	0.417160	0.031706	V10

Logistic Regression:

Logistic regression estimates the probability of an event occurring, such as voted or didn’t vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds. We apply logistic regression on the transformed train and test data.

ROC Curve: -

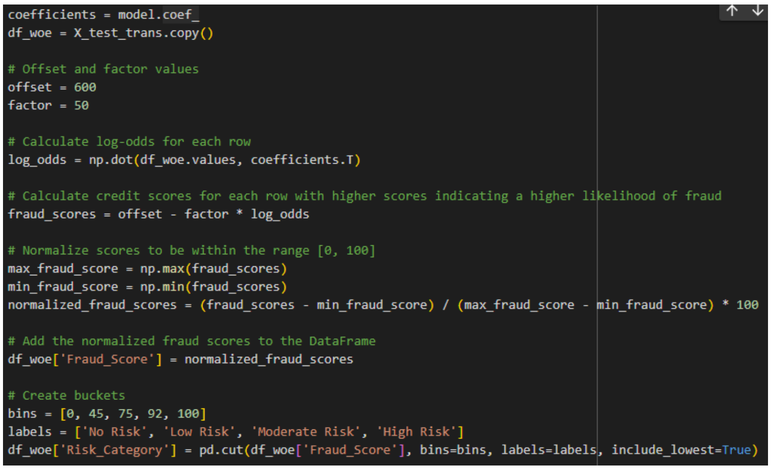


Scoring the Model:

Mechanism:

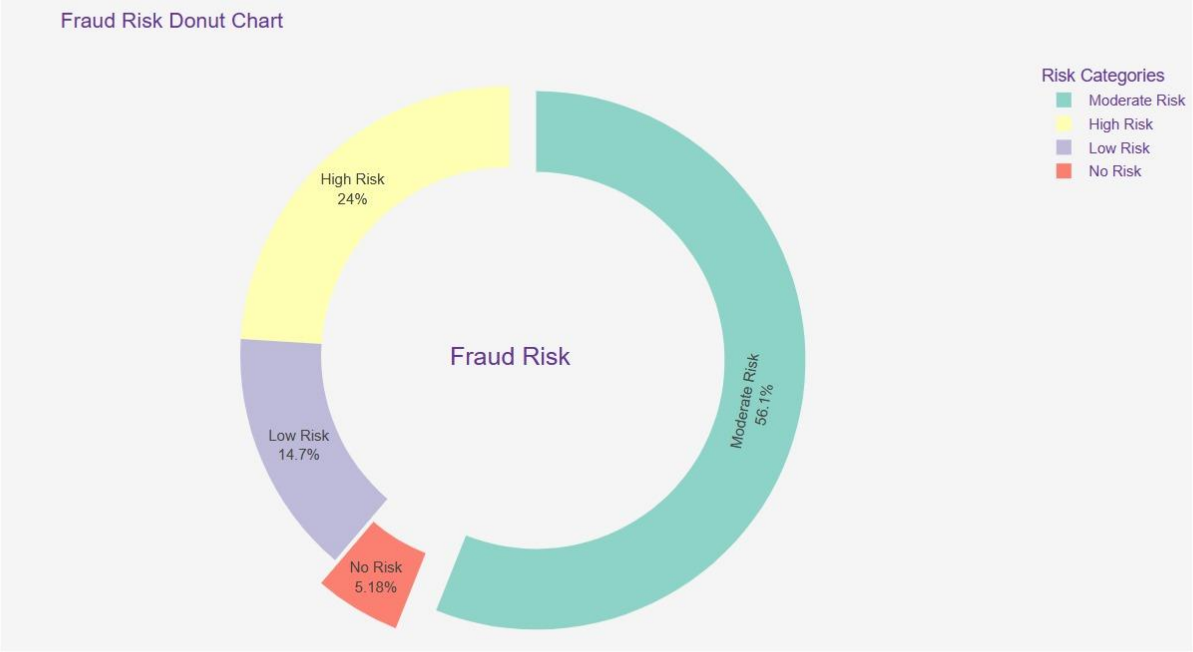
The logistic regression model's coefficients are applied to the test data, and the resulting log-odds are transformed into fraud scores through a linear combination involving an offset and a factor. These scores are then normalized to a range between 0 and 100. The code appends the normalized scores to the test data and categorizes the risk into four levels: "No Risk," "Low Risk," "Moderate Risk," and "High Risk." A donut chart is generated to visually represent the distribution of risk categories.

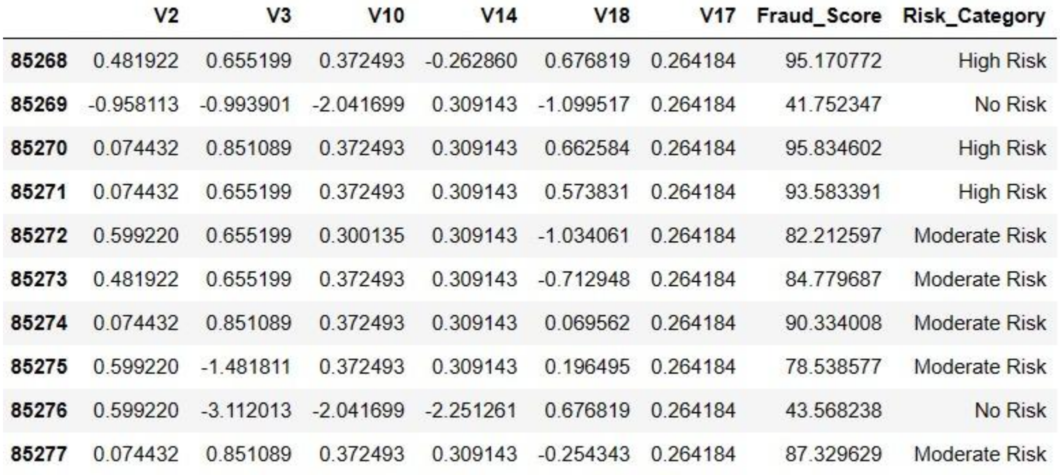
Formula:



```
coefficients = model.coef_
df_woe = X_test_trans.copy()

# Offset and factor values
offset = 600
factor = 50

# Calculate log-odds for each row
log_odds = np.dot(df_woe.values, coefficients.T)

# Calculate credit scores for each row with higher scores indicating a higher likelihood of fraud
fraud_scores = offset - factor * log_odds

# Normalize scores to be within the range [0, 100]
max_fraud_score = np.max(fraud_scores)
min_fraud_score = np.min(fraud_scores)
normalized_fraud_scores = (fraud_scores - min_fraud_score) / (max_fraud_score - min_fraud_score) * 100

# Add the normalized fraud scores to the DataFrame
df_woe['Fraud_Score'] = normalized_fraud_scores

# Create buckets
bins = [0, 45, 75, 92, 100]
labels = ['No Risk', 'Low Risk', 'Moderate Risk', 'High Risk']
df_woe['Risk_Category'] = pd.cut(df_woe['Fraud_Score'], bins=bins, labels=labels, include_lowest=True)
```

Results:





	V2	V3	V10	V14	V18	V17	Fraud_Score	Risk_Category
85268	0.481922	0.655199	0.372493	-0.262860	0.676819	0.264184	95.170772	High Risk
85269	-0.958113	-0.993901	-2.041699	0.309143	-1.099517	0.264184	41.752347	No Risk
85270	0.074432	0.851089	0.372493	0.309143	0.662584	0.264184	95.834602	High Risk
85271	0.074432	0.655199	0.372493	0.309143	0.573831	0.264184	93.583391	High Risk
85272	0.599220	0.655199	0.300135	0.309143	-1.034061	0.264184	82.212597	Moderate Risk
85273	0.481922	0.655199	0.372493	0.309143	-0.712948	0.264184	84.779687	Moderate Risk
85274	0.074432	0.851089	0.372493	0.309143	0.069562	0.264184	90.334008	Moderate Risk
85275	0.599220	-1.481811	0.372493	0.309143	0.196495	0.264184	78.538577	Moderate Risk
85276	0.599220	-3.112013	-2.041699	-2.251261	0.676819	0.264184	43.568238	No Risk
85277	0.074432	0.851089	0.372493	0.309143	-0.254343	0.264184	87.329629	Moderate Risk

Conclusion:

In conclusion, this project represents a significant stride in advancing credit card transaction analysis through the meticulous development of innovative regression models. The emphasis on real-time fraud detection, dynamic credit management, and user-centric security measures underscores the commitment to staying ahead of evolving challenges. The potential for cross-industry applications and the exploration of blockchain technology further solidify the project's relevance and impact. As the financial landscape continues to evolve, these strategic future scopes position the research to make enduring contributions to fraud prevention, credit management, and security measures in credit card transactions. The dedication to continuous improvement, collaboration, and ethical considerations ensures that this endeavor remains at the forefront of addressing the complexities inherent in the dynamic domain of financial transactions.

Future Scope:

1. **Real-Time Fraud Detection Integration:**
   - Adapt regression models for real-time fraud detection, leveraging advanced anomaly detection techniques and ensuring responsiveness to evolving fraudulent patterns.

2. **Dynamic Credit Management Strategies:**
   - Develop dynamic credit management practices that adapt to changing economic conditions and user behaviors, incorporating external factors for predicting credit limits in a dynamic financial landscape.

3. **User-Centric Security Measures:**
   - Integrate user-centric security measures, such as biometrics and behavioral analysis, to complement regression models and provide an additional layer of protection.

4. **Cross-Industry Applicability:**
   - Assess the applicability of regression models to other industries, fostering collaborations to share insights and improve fraud prevention practices across diverse domains.

5. **Blockchain for Enhanced Security:**
   - Explore the integration of blockchain or distributed ledger technologies to enhance the security and transparency of credit card transactions, potentially reducing the risk of fraud and unauthorized activities.

References:

1. https://www.researchgate.net/publication/322520135_Monotone_optimal_binning_algorithm_for_credit_risk_modeling - Monotone optimal binning algorithm for credit risk modeling
2. https://arxiv.org/abs/1611.06439 - Credit Card Fraud Detection: A Review of the State of the Art
3. https://towardsdatascience.com/credit-card-fraud-detection-using-machine-learning-python-5b098d4a8edc - Building a Machine Learning Model for Credit Card Fraud Detection
4. https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html - Weight of Evidence & Information Value
5. https://medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c - SMOTE
6. https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets - Exploratory Data Analysis
7. LLMS - Chat GPT, Bard