

\WOLF/

Applied Project Report Analysis of Real-life Business Cases

By

Kasula Lohith Kumar

**A Master's Project Report submitted to Scaler Neovarsity - Woolf in partial fulfillment of the
requirements for the degree of Master of Science in Computer Science**

November, 2025



Scaler Mentee Email ID : kasulalohithkumar@gmail.com

Thesis Supervisor : Shivank Agarwal

Date of Submission : 11/11/2025

© The project report of Kasula Lohith Kumar is approved, and it is acceptable in quality and form for publication electronically

Certification

I confirm that I have overseen / reviewed this applied project and, in my judgment, it adheres to the appropriate standards of academic presentation. I believe it satisfactorily meets the criteria, in terms of both quality and breadth, to serve as an applied project report for the attainment of Master of Science in Computer Science degree. This applied project report has been submitted to Woolf and is deemed sufficient to fulfill the prerequisites for the Master of Science in Computer Science degree.

Shivank Agarwal

.....

Project Guide / Supervisor

DECLARATION

I confirm that this project report, submitted to fulfill the requirements for the Master of Science in Computer Science degree, completed by me from **15/11/2023** to **11/11/2025** is the result of my own individual endeavor. The Project has been made on my own under the guidance of my supervisor with proper acknowledgement and without plagiarism. Any contributions from external sources or individuals, including the use of AI tools, are appropriately acknowledged through citation. By making this declaration, I acknowledge that any violation of this statement constitutes academic misconduct. I understand that such misconduct may lead to expulsion from the program and/or disqualification from receiving the degree.

Kasula Lohith Kumar



Date: 11/11/2025

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to my parents, **Mr. K. Sampath Kumar** and **Mrs. K. Vimala**, for their unwavering love, guidance, and encouragement throughout my learning journey. Their constant support has been the foundation of my perseverance and success. I am deeply thankful to all my **Scaler instructors** whose expertise, patience, and dedication have been instrumental in shaping my understanding across various domains. Special thanks to **Suraaj Hasija** (SQL, GenAI), **Thanish Batcha** (Tableau and Excel), **Akash Rajpuria** (Python Libraries, Maths for ML, Intro to ML and NN, Supervised Algorithms), **Aniruddha Mukherjee** (Probability & Stats, Data Analytics and Visualisation Fundamentals), **Eshan Tiwari** (Product Analytics), **Archit Sharma** (Advanced Python), **Rohit Jindal** (Advanced Supervised Algorithms, Unsupervised and RecSys), **Shivam Prasad** (MLOps, Computer Vision), **Satya Pattnaik** (Neural Networks, NLP), **Ashay Sinha** (Product Management for Software Engineers), **Amit Singh** (Data Engineering), and **Shivank Agarwal** (GenAI). Each of them has played a vital role in helping me gain knowledge, confidence, and a problem-solving mindset.

I would also like to extend my sincere appreciation to the **co-founders of Scaler** — **Abhimanyu Saxena, Anshuman Singh, and Vaibhav Gupta** — for creating such an empowering and well-structured learning ecosystem that bridges the gap between education and industry needs. My gratitude also goes out to all the **teaching and non-teaching faculty members**, whose consistent efforts, guidance, and support contributed immensely to the successful completion of my **Data Science and Machine Learning** program. Lastly, I am thankful to my peers, mentors, and everyone who inspired and motivated me to complete this program and earn my Master's degree.

Table of Contents

List of Tables	13
List of Figures	14
Applied Real-life Business Cases	25
ABSTRACT	25
Chapter 1 : Business Case Study 1	26
1.1 Problem Description	26
1.2 Business Questions to be Answered	30
1.2.1 Questions	30
1.2.2 Background and Significance	31
1.2.2 Methodology and Real-Life Applications	32
1.3 Analysis	33
1.3.1 Data Understanding and Exploration	33
1.3.1.1 Data types and characteristics of all columns	33
1.3.1.2 Between what time range were the orders placed	34
1.3.1.3 Unique cities and states of the customers	35
1.3.2 Order Trends and Seasonality in terms of the number of orders being placed	36
1.3.3 Evolution of E-commerce Orders in the Region	40
1.3.3.1 Month-on-month order volume in each state	40
1.3.3.2 Customers distributed geographically across all states	42
1.3.4 Economic and Operational Analysis	44
1.3.4.1 Percentage increase in the total cost of orders	44
1.3.4.2 Total and average order prices for each state	47
1.3.4.3 Total and average freight values for each state	49
1.3.5 Delivery and Freight Performance	51
1.3.5.1 No of Days taken to deliver each order from the purchase date	51
1.3.5.2 Difference between the estimated and actual delivery date for each order	53
1.3.5.3 Top 5 states with the highest and lowest average freight values	56
1.3.5.4 Delivery time and estimated date of top 5 states	58
1.3.6 Payment Analysis	61
1.3.6.1 Month-on-month orders placement using different payment types	61
1.3.6.2 Distribution of orders based on the number of payment installments	62
1.4 Insights and Recommendations	64
Chapter 2 : Business Case Study 2	67

2.1 Problem Description	68
2.2 Business Questions To Be Answered	68
2.2.1 Questions	68
2.2.2 Background and Significance	68
2.2.3 Problem Statement (Anonymized)	69
2.2.4 Methodology and Real-World Applications	69
2.2.5 Insights, Recommendations, and Implications	70
2.3.6 Addressing Limitations and Suggestions	70
2.3 Analysis	71
2.3.1 Exploratory Data Analysis (EDA)	71
2.3.1.1 Data Frame Preview	71
2.3.1.2 Data Frame shape	72
2.3.1.3 Data Frame Columns	72
2.3.1.4 Detection of Null Values	73
2.3.1.5 Duplicate values	74
2.3.1.6 Descriptive Statistics of Numerical Dataset	75
2.3.1.7 Descriptive Statistics of Categorical Dataset	76
2.3.2 Univariate Analysis	77
2.3.2.1 Loan Term Proportions	77
2.3.2.2 Initial List Status Proportions	79
2.3.2.3 Verification Status Proportions	80
2.3.2.4 Application Type Proportions	82
2.3.2.5 Count of Different Grades	83
2.3.2.6 Employee experience and their loan application count	85
2.3.2.7 Home Ownership and Respective Loan application count	86
2.3.2.8 Loan Verification Status and Respective Application Count	87
2.3.2.9 Loan Application Purpose and It's Count	88
2.3.2.10 Distribution of Loan Amounts using Kernel Density Estimate (KDE)	89
2.3.2.11 Distribution of Interest Rates using Kernel Density Estimate (KDE)	90
2.3.2.12 Loan Installment Amount using Kernel Density Estimate (KDE)	91
2.3.2.13 Annual Income using Kernel Density Estimate (KDE)	91
2.3.2.14 Debt to Income using Kernel Density Estimate (KDE)	92
2.3.2.15 Open Accounts using Kernel Density Estimate (KDE)	93
2.3.2.16 Public Records using Kernel Density Estimate (KDE)	94
2.3.2.17 Revolving Balance using Kernel Density Estimate (KDE)	95
2.3.2.18 Revolving Utilization using Kernel Density Estimate (KDE)	96
2.3.2.19 Total Accounts using Kernel Density Estimate (KDE)	97
2.3.2.20 Mortgage Accounts using Kernel Density Estimate (KDE)	98
2.3.2.21 Public Record Bankruptcies using Kernel Density Estimate (KDE)	99

2.3.3 Bivariate Analysis	99
2.3.3.1 Loan Status Count w.r.t Term	99
2.3.3.2 Loan Status Count w.r.t Different Grades	101
2.3.3.3 Loan Status Count w.r.t Different Sub Grades	103
2.3.3.4 Loan Status Count w.r.t Employee Length	105
2.3.3.5 Loan Status Count w.r.t Home Ownership	106
2.3.3.6 Loan Status Count w.r.t Verification Status	108
2.3.3.7 Loan Status Count w.r.t Purpose	109
2.3.3.8 Loan Status Count w.r.t Initial List Status	110
2.3.3.9 Loan Status Count w.r.t Application Type	111
2.3.3.10 Loan Status of top 12 Postal Codes	112
2.3.3.11 Purpose of Loan – Top 20	112
2.3.3.12 Loan Status Count w.r.t ‘issue_year’ and ‘issue_month’	113
2.3.3.13 Top 20 Status of Loan w.r.t to ‘emp_title’	115
2.3.3.14 Loan Status Count w.r.t to 'earliest_cr_line_year' and 'earliest_cr_line_month'	116
2.3.3.15 Distribution of Loan Amount w.r.t Loan Status	118
2.3.3.16 Distribution of Interest Rate w.r.t Loan Status	119
2.3.3.17 Distribution of Installments w.r.t Loan Status	121
2.3.3.18 Distribution of Annual Income w.r.t Loan Status	122
2.3.3.19 D.T.I w.r.t Loan Status	123
2.3.3.20 Open Account w.r.t Loan Status	124
2.3.3.21 Public Record w.r.t Loan Status	125
2.3.3.22 Revolving Balance w.r.t Loan Status	126
2.3.3.23 Revolving Line Utilization Rate w.r.t Loan Status	127
2.3.3.24 Total No of Credit Lines w.r.t Loan Status	129
2.3.3.25 Total No of Mortgage Accounts w.r.t Loan Status	130
2.3.3.26 Total No of Public Record Bankruptcies w.r.t Loan Status	131
2.3.4 Multivariate Analysis	132
2.3.4.1 Pair Plot of All Numerical variables	132
2.3.4.2 Correlation plot of all Numerical Variables	135
2.3.5 Data Pre-Processing	137
2.3.5.1 Removing Highly Correlated Features	137
2.3.5.2 Null Values Treatment	137
2.3.5.3 Log Transformation	139
2.3.5.4 Outlier Treatment	139
2.3.5.5 Handling 'address' Feature values Inconsistency	140
2.3.5.6 Handling 'title' Feature values Inconsistency	142

2.3.6 Sanity check after preprocessing	143
2.3.6.1 Null values	143
2.3.6.2 Outliers	145
2.3.7 Data Preprocessing for Modelling	145
2.3.7.1 Code for Data Preprocessing for Modelling	146
2.3.7.2 Splitting (Train,Test), Encoding, Balancing and Scaling the Data	147
2.3.8 Sanity Check After Data Preprocessing for Modelling	148
2.3.8.1 Shape of the data after splitting	148
2.3.8.2 Loan Status Proportion Check After SMOTE	149
2.3.8.3 Encoded and Scaled Data	149
2.3.9 Modelling Using Logistic Regression	150
2.3.9.1 Detecting Multicollinearity with VIF	150
2.3.9.2 Splitting the data into 3 parts (train, test and validation)	152
2.3.9.3 Regularization after dropping the redundant features	152
2.3.9.4 Model Training	154
2.3.10 Results Evaluation	155
2.3.10.1 Classification Report	155
2.3.10.2 Accuracy	156
2.3.10.3 ROC-AUC Curve	157
2.3.10.4 Precision-Recall Curve	158
2.4 Insights and Recommendations	159
2.4.1 Results and Recommendations	160
2.4.2 Implications for Industry, Business, and Policy	160
2.4.3 Addressing Limitations and Constraints	160
2.4.4 Alternative Approaches & Recommendations	161
2.4.5 Key Learnings, Methodology, and Industry Applications	161
Chapter 3 : Business Case Study 3	
3.1 Business Case Problem Statement	162
3.2 Key Steps and Questions to Answer For Analysis	164
3.2.1 Key Steps	164
3.2.2 Questions to Answer	164
3.2.3 Background & Significance	165
3.2.4 Methodology and Real-Life Application	165
3.3 Analysis	165
3.3.1 Exploratory Data Analysis	165
3.3.2 Shape	167
3.3.3 Info	168
3.3.4 Extraction of Structured Metadata from Wikipedia Page Names Using	

Regular Expressions	179
3.3.4 Columns of Dataset	180
3.3.5 Handling Null Values	181
3.3.6 Exploring Categorical Diversity in Wikipedia Page Metadata: Language, Access Type, and Origin	183
3.3.7 Value Count of Languages	183
3.3.8 Extracting Language Name	185
3.3.9 Distribution Analysis of Languages, Access Types, and Origins in Wikipedia Pageviews	186
3.3.10 Wikipedia Languages and their Access Origin	188
3.3.11 Wikipedia Languages and their Access Type	190
3.3.12 Wikipedia Languages and their Clicks	191
3.3.13 Language-Specific DataFrame Pickling	193
3.3.14 Access Type and Access Origin Type Of All Languages	194
3.3.15 Duplicate Values Check	196
3.3.16 Monthly Average Clicks	197
3.3.17 Top 20 Web Pages By Ad Clicks	200
3.3.18 Bottom 20 Web Pages By Ad Clicks	205
3.3.19 Modelling using ARIMA Family of Modelling Techniques (Arima Pipeline)	209
3.3.20 English Language Forecast using ARIMA Family	215
3.3.21 Chinese Language Forecast using ARIMA Family	220
3.3.22 French Language Forecast using ARIMA Family	224
3.3.23 Japanese Language Forecast using ARIMA Family	228
3.3.24 Russian Language Forecast using ARIMA Family	232
3.3.25 German Language Forecast using ARIMA Family	236
3.3.26 Modelling using FB Prophet Pipeline	239
3.3.27 English Language Forecast FB Prophet Pipeline	243
3.3.28 German Language Forecast FB Prophet Pipeline	245
3.3.29 French Language Forecast FB Prophet Pipeline	247
3.3.30 Spanish Language Forecast FB Prophet Pipeline	249
3.3.31 Chinese Language Forecast FB Prophet Pipeline	251
3.3.32 Japanese Language Forecast FB Prophet Pipeline	252
3.3.33 Russian Language Forecast FB Prophet Pipeline	254
3.4 Overall Insights and Recommendations	256
3.4.1 Presentation of Results with Justification	257
3.4.2 Implications for Industry, Business, and Policy	257
3.4.3 Limitations	257

3.4.4 Alternative Recommendations	258
3.4.5 Key Learnings, Methodology, and Industry Application	258
Chapter 4 : Business Case Study 4	
4.1 Problem Description	259
4.1.1 Problem Statement (Business Case)	259
4.1.2 Business Goals Background	259
4.1.3 How Recommendations Work	259
4.2 Business Questions to be Answered for Analysis	263
4.2.1 Questions	263
4.2.2 Significance of These Questions	263
4.2.3 Methodology & Real-World Applications	263
4.2.4 Practical Applications	264
4.3 Analysis	264
4.3.1 Data Frame Structure and Initial Preprocessing	264
4.3.1.1 Loading Movie Recommendation DataFrames from CSV Files	265
4.3.1.2 Structure of DataFrame	265
4.3.1.3 Exploded Movies Data Frame by Individual Genre	266
4.3.1.4 Grouping The Data Frame Based on Index	266
4.3.1.5 List of Unique Movie Genres Identified in Dataset	267
4.3.1.6 Final Movies DataFrame with Genre Dummy Variables	267
4.3.1.7 Ratings Data Frame with Derived Time Features	268
4.3.1.8 Users Data Frame Structure	269
4.3.1.9 Merged Movies and Ratings DataFrame with Genre Features	270
4.3.1.10 Movies-Ratings DataFrame with Genre Features (MovieID Dropped)	271
4.3.1.11 Fully Merged Movies, Ratings, and User Data Frame	271
4.3.1.12 Decoding Occupation and Age Columns in Final Data Frame	272
4.3.2 Exploratory Data Analysis (EDA)	274
4.3.2.1 Data frame Shape	274
4.3.2.2 Dataframe Information	274
4.3.2.3 Data Frame Description for Numerical columns	276
4.3.2.4 Data Frame Description for Object type columns	276
4.3.2.5 Data Frame Null Values Check	277
4.3.2.6 Unique Column Values	279
4.3.3 Univariate Analysis	280
4.3.3.1 Proportion of Gender	280
4.3.3.2 Distribution of Age	281
4.3.3.3 Distribution of Movie Ratings	282
4.3.3.4 Distribution of Movie Ratings by Day of Week	283

4.3.3.5 Distribution of Movie Ratings by Day of Month	284
4.3.3.6 Number of People by Occupation	285
4.3.3.7 Time when people watch movies and give ratings	286
4.3.3.7 Top 20 years in which the highest number of movies were released	287
4.3.4 Bivariate Analysis	288
4.3.4.1 Genre Count w.r.t Gender	288
4.3.4.2 Genre w.r.t Age	290
4.3.4.3 Genre w.r.t Occupation	293
4.3.5 Modelling Recommender System	296
4.3.5.1 Recommender System Based on Pearson Correlation	296
4.3.5.1.1 Item Based	296
4.3.5.1.2 User Based	299
4.3.5.2 Item-Item Similarity Matrix	300
4.3.5.3 User-User Similarity Matrix	301
4.3.5.4 Matrix Factorization	302
4.3.5.4.1 Overlap	304
4.3.5.4.2 Precision of Top-100 Movie Recommendations—Hit Rate Analysis	306
4.3.5.4.3 Low Overlap of Recommendations with Highly Rated Movies	307
4.4 Insights and Recommendations	308
4.4.1 Results and Recommendations	308
4.4.2 Implications for Industry/Business/Policy	308
4.4.3 Limitations and Constraints	308
4.4.4 Alternative Explanations or Recommendations	309
4.4.5 Key Learnings, Methodology and Applications	309
Chapter 5 : Business Case Study 5	
5.1 Problem Description	310
5.1.1 Problem Statement	310
5.1.2 Business Background	310
5.2 Business Questions to Answer	310
5.2.1 Questions	310
5.2.2 Significance of These Questions	311
5.2.3 Methodology Overview	311
5.3 Analysis	312
5.3.1 Exploratory Data Analysis (EDA)	312
5.3.1.1 Data frame Import	312
5.3.1.2 Shape	313
5.3.1.3 Distribution of Categories	314
5.3.2 Text Processing	316

5.3.3 Encoding	317
5.3.4 Modelling	318
5.3.4.1 Feature Extraction Choice: Bag of Words vs. TF-IDF for Text Classification	318
5.3.4.2 Train-Test Split	319
5.3.4.3 Naive Bayes Model Performance for News Article Categorization	320
5.3.4.4 Multiclass Classification Results: Naive Bayes Model on News Categories	321
5.3.4.5 Reusable Model Training and Evaluation Functions for News Classification	323
5.3.4.6 Decision Tree Classifier Results for News Categorization	324
5.3.4.7 K-Nearest Neighbors Classifier Performance for News Article Categories	325
5.3.4.8 Random Forest Classifier Performance for News Article Categorization	325
5.4 Insights and Recommendations for All Models	326
5.4.1 Results Overview and Model Comparison	326
5.4.2 Recommendations	327
5.4.3 Justification	327
5.4.4 Implications for Industry and Policy	327
5.4.5 Limitations & Constraints	327
5.4.6 Alternative Explanation & Improvements	328
5.4.7 Summary of Key Learnings and Methodology	328
CONCLUSION	329
Key Takeaways	329
Practical Applications	329
Limitations and Suggestions for Improvement	329
References	331

List of Tables

(To be written sequentially as they appear in the text)

Table No.	Title	Page No.
1	Table 1.1: customers.csv — Customer Information	26
2	Table 1.2: sellers.csv — Seller Information	26
3	Table 1.3: order_items.csv — Order Item Details	27
4	Table 1.4: geolocations.csv — Geographical Details	27
5	Table 1.5: payments.csv — Payment Information	27
6	Table 1.6: orders.csv — Order Information	28
7	Table 1.7: reviews.csv — Customer Reviews	28
8	Table 1.8: products.csv — Product Details	29
9	Table 4.1: Movie Ratings Data Format and Constraints	260
10	Table 4.2: Users Data File Format and Codebook	260
11	Table 4.3: Age Codes	260
12	Table 4.4: Occupation Codes	261
13	Table 4.5: Movies Data File Format and Genre List	262
14	Table 4.6: Genre List	262
15	References	331

List of Figures

(List of Images, Graphs, Charts sequentially as they appear in the text)

Figure No.	Title	Page No.
1	Figure 1.1: Dataset schema	30
2	Figure 1.2: Data type of all columns in the "customers" table	33
3	Figure 1.3: Time range between which the orders were placed.	34
4	Figure 1.4: Count of Cities & States of customers who ordered during the given period.	35
5	Figure 1.5 : Query of Growing trend and seasonality analysis	36
6	Figure 1.6 : Output of Growing trend and seasonality analysis	37
7	Figure 1.7 : Time of the day when Brazilian customers mostly place their orders	39
8	Figure 1.8 : Query of Month on month no. of orders placed in each state.	40
9	Figure 1.9 : Output of Month on month no. of orders placed in each state.	41
10	Figure 1.10 : Query of Customers distribution across all the states.	42
11	Figure 1.11: Output of Customers distribution across all the states.	43
12	Figure 1.12 : Query of Cost of orders from year 2017 to 2018 (include months between Jan to Aug only)	46
13	Figure 1.13 : Output of Cost of orders from year 2017 to 2018 (include months between Jan to Aug only)	46

14	Figure 1.14 : Query of Total & Average value of order price for each state.	47
15	Figure 1.15 : Output of Total & Average value of order price for each state.	48
16	Figure 1.16 : Query of Total & Average value of order freight for each state.	49
17	Figure 1.17 : Output of Total & Average value of order freight for each state.	50
18	Figure 1.18 : Query of days taken to deliver each order from the purchase date.	52
19	Figure 1.19 : Output of days taken to deliver each order from the purchase date.	52
20	Figure 1.20 : Query of Difference (in days) between the estimated & actual delivery date of an order.	54
21	Figure 1.21 : Output of Difference (in days) between the estimated & actual delivery date of an order.	55
22	Figure 1.22 : Query of top 5 states with the highest & lowest average delivery time.	56
23	Figure 1.23 : Output of top 5 states with the highest & lowest average delivery time.	57
24	Figure 1.24 : Query of top 5 states where the order delivery is really fast as compared to the estimated date of delivery	59
25	Figure 1.25 : Output of top 5 states where the order delivery is really fast as compared to the estimated date of delivery	60
26	Figure 1.26 : Query of month on month no. of orders placed using different payment types.	61
27	Figure 1.27 : Output of month on month no. of orders placed using different payment types.	61
28	Figure 1.28 : Query of distribution of orders based on the number of payment installments.	62

29	Figure 1.29 : Output of distribution of orders based on the number of payment installments	63
30	Figure 2.1: Data Frame Head Preview	71
31	Figure 2.2: Data Frame Shape	71
32	Figure 2.3: Data Frame Columns	72
33	Figure 2.4: Detection of Null Values	73
34	Figure 2.5: Duplicate values	74
35	Figure 2.6: Descriptive Statistics of Numerical Dataset	75
36	Figure 2.7: Descriptive Statistics of Categorical Dataset	76
37	Figure 2.8: Loan Term Proportions	78
38	Figure 2.9: Initial List Status Proportions	79
39	Figure 2.10: Verification Status Proportions	81
40	Figure 2.11: Application Type Proportions	82
41	Figure 2.12: Count of Different Grades	85
42	Figure 2.13: Employee Experience And Their Loan Application Count	85
43	Figure 2.14: Home Ownership and Respective Loan application count	86
44	Figure 2.15: Loan Verification Status and Respective Application Count	87
45	Figure 2.16: Loan Application Purpose and It's Count	88
46	Figure 2.17: Distribution of Loan Amounts using Kernel Density Estimate (KDE)	89
47	Figure 2.18: Distribution of Interest Rates using Kernel Density Estimate (KDE)	90

48	Figure 2.19: Loan Installment Amount using Kernel Density Estimate (KDE)	91
49	Figure 2.20: Annual Income using Kernel Density Estimate (KDE)	92
50	Figure 2.21: Debt to Income using Kernel Density Estimate (KDE)	93
51	Figure 2.22: Open Accounts using Kernel Density Estimate (KDE)	94
52	Figure 2.23: Public Records using Kernel Density Estimate (KDE)	95
53	Figure 2.24: Revolving Balance using Kernel Density Estimate (KDE)	96
54	Figure 2.25: Revolving Balance using Kernel Density Estimate (KDE)	97
55	Figure 2.26: Total Accounts using Kernel Density Estimate (KDE)	97
56	Figure 2.27: Mortgage Accounts using Kernel Density Estimate (KDE)	98
57	Figure 2.28: Public Record Bankruptcies using Kernel Density Estimate (KDE)	99
58	Figure 2.29: Loan Status Count w.r.t Term	100
59	Figure 2.30: Loan Status Count w.r.t Different Grades	102
60	Figure 2.31: Loan Status Count w.r.t Different Sub Grades	104
61	Figure 2.32: Loan Status Count w.r.t Employee Length	106
62	Figure 2.33: Loan Status Count w.r.t Home Ownership	107
63	Figure 2.34: Loan Status Count w.r.t Verification Status	108
64	Figure 2.35: Loan Status Count w.r.t Purpose	109
65	Figure 2.36: Loan Status Count w.r.t Initial List Status	110

66	Figure 2.37: Loan Status Count w.r.t Application Type	111
67	Figure 2.38: Loan Status of top 12 Postal Codes	112
68	Figure 2.39: Purpose of Loan – Top 20	113
69	Figure 2.40: Loan Status Count w.r.t ‘issue_year’ and ‘issue_month’	114
70	Figure 2.41: Top 20 Status of Loan w.r.t to ‘emp_title’	115
71	Figure 2.42: Loan Status Count w.r.t to 'earliest_cr_line_year' and 'earliest_cr_line_month'	117
72	Figure 2.43: Distribution of Loan Amount w.r.t Loan Status	118
73	Figure 2.44: Distribution of Interest Rate w.r.t Loan Status	120
74	Figure 2.45: Distribution of Installments w.r.t Loan Status	121
75	Figure 2.46: Distribution of Annual Income w.r.t Loan Status	122
76	Figure 2.47: D.T.I w.r.t Loan Status	123
77	Figure 2.48: Open Account w.r.t Loan Status	124
78	Figure 2.49: Public Record w.r.t Loan Status	125
79	Figure 2.50: Revolving Balance w.r.t Loan Status	126
80	Figure 2.51: Revolving Line Utilization Rate w.r.t Loan Status	128
81	Figure 2.52: Total No of Credit Lines w.r.t Loan Status	129
82	Figure 2.53: Total No of Mortgage Accounts w.r.t Loan Status	130
83	Figure 2.54: Total No of Public Record Bankruptcies w.r.t Loan Status	131
84	Figure 2.55: Pair Plot of All Numerical variables	135
85	Figure 2.56: Correlation plot of all Numerical Variables	136
86	Figure 2.57: Removing Highly Correlated Features	137

87	Figure 2.58: Null Values Treatment	138
88	Figure 2.59: Log Transformation	139
89	Figure 2.60: Outlier Treatment	140
90	Figure 2.61: Handling 'address' Feature values Inconsistency	141
91	Figure 2.62: Handling 'title' Feature values Inconsistency	142
92	Figure 2.63: Systematic Cleaning & Standardization	144
93	Figure 2.64: Outliers	145
94	Figure 2.65: Code for Data Preprocessing for Modelling	147
95	Figure 2.66: Splitting (Train,Test), Encoding, Balancing and Scaling the Data	147
96	Figure 2.67: Shape of the data after splitting	149
97	Figure 2.68: Loan Status Proportion Check After SMOTE	149
98	Figure 2.69: Encoded and Scaled Data	150
99	Figure 2.70: Detecting Multicollinearity with VIF	151
100	Figure 2.71: Splitting the data into 3 parts (train, test and validation)	152
101	Figure 2.72: Regularization after dropping the redundant features	153
102	Figure 2.73: Model Training	154
103	Figure 2.74: Classification Report	155
104	Figure 2.75: Accuracy	156
105	Figure 2.76: ROC-AUC Curve	157
106	Figure 2.77: Precision-Recall Curve	159
107	Figure 3.1: Exploratory Data Analysis	166
108	Figure 3.2: Shape	167

109	Figure 3.3: Info	177
110	Figure 3.4: Extraction of Structured Metadata from Wikipedia Page Names Using Regular Expressions	179
112	Figure 3.5: Columns of Dataset	180
113	Figure 3.6: Handling Null Values	181
114	Figure 3.7: Exploring Categorical Diversity in Wikipedia Page Metadata: Language, Access Type, and Origin	182
115	Figure 3.8: Value Count of Languages	184
116	Figure 3.9: Extracting Language Name	185
117	Figure 3.10: Distribution Analysis of Languages, Access Types, and Origins in Wikipedia Pageviews	187
118	Figure 3.11: Wikipedia Languages and their Access Origin	189
119	Figure 3.12: Wikipedia Languages and their Access Type	191
120	Figure 3.13: Wikipedia Languages and their Clicks	192
121	Figure 3.14: Language-Specific DataFrame Pickling	193
122	Figure 3.15: Access Type and Access Origin Type Of All Languages	195
123	Figure 3.16: Duplicate Values Check	196
124	Figure 3.17: Monthly Average Clicks	199
125	Figure 3.18: Top 20 Web Pages By Ad Clicks	204
126	Figure 3.19: Bottom 20 Web Pages By Ad Clicks	209
127	Figure 3.20: Modelling using ARIMA Family of Modelling Techniques (Arima Pipeline)	214
128	Figure 3.21: English Language Forecast using ARIMA Family	220

129	Figure 3.22: Chinese Language Forecast using ARIMA Family	224
130	Figure 3.23: French Language Forecast using ARIMA Family	228
131	Figure 3.24: Japanese Language Forecast using ARIMA Family	232
132	Figure 3.25: Russian Language Forecast using ARIMA Family	235
133	Figure 3.26: German Language Forecast using ARIMA Family	238
134	Figure 3.27: Modelling using FB Prophet Pipeline	242
135	Figure 3.28: English Language Forecast FB Prophet Pipeline	245
136	Figure 3.29: German Language Forecast FB Prophet Pipeline	247
137	Figure 3.30: French Language Forecast FB Prophet Pipeline	248
138	Figure 3.31: Spanish Language Forecast FB Prophet Pipeline	250
139	Figure 3.32: Chinese Language Forecast FB Prophet Pipeline	252
140	Figure 3.33: Japanese Language Forecast FB Prophet Pipeline	254
141	Figure 3.34: Russian Language Forecast FB Prophet Pipeline	256
142	Figure 4.1: Loading Movie Recommendation DataFrames from CSV Files	265
143	Figure 4.2: Structure of Data frame	265
144	Figure 4.3: Exploded Movies Data Frame by Individual Genre	266
145	Figure 4.4: Grouping The Data Frame Based on Index	266
146	Figure 4.5: List of Unique Movie Genres Identified in Dataset	267
147	Figure 4.6: Final Movies Data Frame with Genre Dummy Variables	268

148	Figure 4.7: Ratings Data Frame with Derived Time Features	269
149	Figure 4.8: Users Data Frame Structure	270
150	Figure 4.9: Merged Movies and Ratings DataFrame with Genre Features	270
151	Figure 4.10: Movies-Ratings Data Frame with Genre Features (MovieID Dropped)	271
152	Figure 4.11: Fully Merged Movies, Ratings, and User Data Frame	272
153	Figure 4.12: Decoding Occupation and Age Columns in Final Data Frame	273
154	Figure 4.13: Data Frame Shape	274
155	Figure 4.14: Data Frame Information	275
156	Figure 4.15: Data Frame Description for Numerical columns	276
157	Figure 4.16: Data Frame Description for Object type columns	277
158	Figure 4.17: Data Frame Null Values Check	279
159	Figure 4.18: Unique Column Values	279
160	Figure 4.19: Proportion of Gender in given dataset	281
161	Figure 4.20: Distribution of Age	282
162	Figure 4.21: Distribution of Movie Ratings	283
163	Figure 4.22: Number of Movie Ratings by Day	284
164	Figure 4.23: Number of Movie Ratings by Month	285
165	Figure 4.24: Number of People by Occupation	286
166	Figure 4.25: Time when people watch movies and give ratings	287
167	Figure 4.26: Top 20 years in which the highest number of movies were released	288

168	Figure 4.27: Genre Count w.r.t Gender	289
169	Figure 4.28: Age w.r.t to Genre	292
170	Figure 4.29: Genres w.r.t to Occupation	296
171	Figure 4.30: Item Based Pearson Correlation	298
172	Figure 4.31: User Based Pearson Correlation	299
173	Figure 4.32: Item-Item Similarity Matrix	300
174	Figure 4.33: Item based approach using Neighbours algorithm and Cosine Similarity	301
175	Figure 4.34: User-User Similarity	302
176	Figure 4.35: Matrix Factorization	304
177	Figure 4.36: Overlap	305
178	Figure 4.37: Precision of Top-100 Movie Recommendations—Hit Rate Analysis	306
179	Figure 4.38: Low Overlap of Recommendations with Highly Rated Movies	307
180	Figure 5.1: Data Frame	312
181	Figure 5.2: Shape	313
182	Figure 5.3: Distribution of Categories	315
183	Figure 5.4: Text Processing	316
184	Figure 5.5: Encoding	317
185	Figure 5.6: Feature Extraction	318
186	Figure 5.7: Train-Test Split	319
187	Figure 5.8: Naive Bayes Model Performance for News Article Categorization	320
188	Figure 5.9: Multiclass Classification Results: Naive Bayes Model on News Categories	322

189	Figure 5.10: Reusable Model Training and Evaluation Functions for News Classification	323
190	Figure 5.11: Decision Tree Classifier Results for News Categorization	324
191	Figure 5.12: K-Nearest Neighbors Classifier Performance for News Article Categories	325
192	Figure 5.13: K-Nearest Neighbors Classifier Performance for News Article Categories	326

Applied Real-life Business Cases

ABSTRACT

This project encompasses comprehensive data-driven analyses and model development across diverse business domains, including e-commerce expansion, credit risk assessment, multilingual ad click forecasting, personalized recommendation systems, and automated news classification. Leveraging a range of advanced methodologies—such as SQL-based analytics, logistic regression with regularization, time series forecasting models (FB Prophet, ARIMA), collaborative filtering, and ensemble machine learning classifiers—the project demonstrates how targeted data science approaches can drive strategic growth, operational efficiency, and customer-centric personalization. Key findings include the importance of regional and temporal segmentation in e-commerce logistics, the need for calibrated and explainable credit risk models to balance default detection and precision, the advantage of language-specific forecasting for digital marketing optimization, and the critical role of tuning hybrid recommendation models to enhance user engagement on OTT platforms. Additionally, automated text classification models provide scalable solutions for real-time content curation and financial news analysis. Despite challenges such as data limitations, model sensitivities to nonlinearities and sudden shifts, and operational constraints related to model maintenance and interpretability, the project highlights practical applications for business intelligence and policy support. Recommendations emphasize continuous model refinement, integration of external data, and adoption of explainable AI techniques to maximize long-term impact across industries.

Chapter 1 : Business Case Study 1

1.1 Problem Description

A leading global retailer aims to strengthen its market presence and operational efficiency in a major South American country. The company positions itself as a preferred shopping destination by offering exceptional value, innovation, and a superior customer experience that distinguishes it from competitors.

This business case focuses on the retailer's e-commerce operations within the region, analyzing a dataset comprising 100,000 customer orders placed between 2016 and 2018. The dataset provides a detailed view across multiple dimensions, including order status, pricing, payment methods, shipping performance, customer demographics, product attributes, and customer feedback.

Through in-depth data analysis, this project seeks to uncover actionable insights into the retailer's operational and customer engagement processes. The analysis aims to identify trends and patterns that can help enhance order management efficiency, pricing and payment strategies, logistics performance, and overall customer satisfaction.

Table 1.1: customers.csv — Customer Information

Feature	Description
customer_id	ID of the consumer who made the purchase
customer_unique_id	Unique ID of the consumer
customer_zip_code_prefix	Zip code of the consumer's location
customer_city	City from where the order is made
customer_state	State code from where the order is made (e.g., São Paulo – SP)

Table 1.2: sellers.csv — Seller Information

Feature	Description
seller_id	Unique ID of the seller registered
seller_zip_code_prefix	Zip code of the seller's location
seller_city	City of the seller
seller_state	State code of the seller (e.g., São Paulo – SP)

Table 1.3: order_items.csv — Order Item Details

Feature	Description
order_id	Unique ID of the order made by the consumer
order_item_id	Unique ID assigned to each item ordered
product_id	Unique ID of each product available
seller_id	Unique ID of the seller registered
shipping_limit_date	Date before which the product must be shipped
price	Actual price of the product ordered
freight_value	Freight charge for delivering the product

Table 1.4: geolocations.csv — Geographical Details

Feature	Description
geolocation_zip_code_prefix	First 5 digits of the zip code
geolocation_lat	Latitude
geolocation_lng	Longitude

geolocation_city	City
geolocation_state	State

Table 1.5: payments.csv — Payment Information

Feature	Description
order_id	Unique ID of the order made by the consumer
payment_sequential	Sequence of payments made (for EMI)
payment_type	Mode of payment used (e.g., Credit Card)
payment_installments	Number of installments (in case of EMI)
payment_value	Total amount paid for the order

Table 1.6: orders.csv — Order Information

Feature	Description
order_id	Unique ID of the order made by the consumer
customer_id	ID of the consumer who made the purchase
order_status	Current status of the order (e.g., delivered, shipped)
order_purchase_timestamp	Timestamp when the purchase was made
order_delivered_carrier_date	Date when the carrier received the product
order_delivered_customer_date	Date when the customer received the product
order_estimated_delivery_date	Estimated delivery date of the order

Table 1.7: reviews.csv — Customer Reviews

Feature	Description
review_id	ID of the review given by the customer

order_id	Unique ID of the order reviewed
review_score	Rating given by the customer (1–5)
review_comment_title	Title of the review
review_comment_message	Review comment written by the customer
review_creation_date	Timestamp when the review was created
review_answer_timestamp	Timestamp when the review was answered

Table 1.8: products.csv — Product Details

Feature	Description
product_id	Unique identifier for the product
product_category_name	Name of the product category
product_name_length	Length of the product name string
product_description_length	Length of the product description
product_photos_qty	Number of product photos available
product_weight_g	Weight of the product in grams
product_length_cm	Length of the product in centimeters
product_height_cm	Height of the product in centimeters
product_width_cm	Width of the product in centimeters

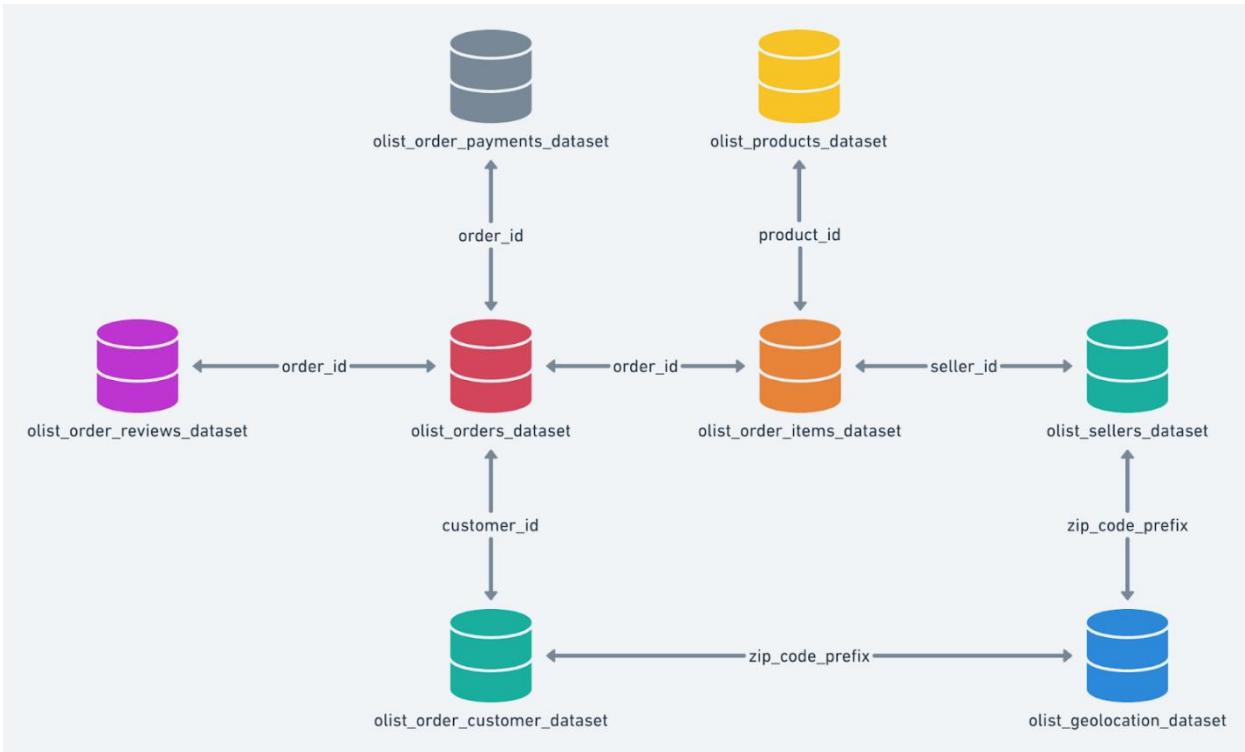


Figure 1.1: Dataset schema

1.2 Business Questions to be Answered from Analysis

1.2.1 Questions

- Data Understanding and Exploration
 - What are the data types and characteristics of all columns in the customers table?
 - Between what time range were the orders placed?
 - How many unique cities and states do the customers belong to?
- Order Trends and Seasonality
 - Is there a growing trend in the number of orders placed over the years?
 - Can we identify any monthly seasonality in terms of the number of orders being placed?
 - During what time of the day do customers most frequently place their orders?
 - 0–6 hrs → *Dawn*
 - 7–12 hrs → *Morning*
 - 13–18 hrs → *Afternoon*
 - 19–23 hrs → *Night*
- Evolution of E-commerce Orders in the Region
 - What is the month-on-month order volume in each state?

- How are customers distributed geographically across all states?
- Economic and Operational Analysis
 - What is the percentage increase in the total cost of orders from January–August 2017 to January–August 2018?
(Use the payment_value column from the payments table.)
 - What are the total and average order prices for each state?
 - What are the total and average freight values for each state?
- Delivery and Freight Performance
 - How many days were taken to deliver each order from the purchase date?
 - What is the difference (in days) between the estimated and actual delivery date for each order?
 - Which are the top 5 states with the highest and lowest average freight values?
 - Identify states that incur the most and least freight charges on average per order.
 - Which are the top 5 states with the highest and lowest average delivery times?
 - Which are the top 5 states where delivery is significantly faster than the estimated date?
- Payment Analysis
 - How many orders are placed month-on-month using different payment types?
 - What is the distribution of orders based on the number of payment installments made by customers?

1.2.2 Background and Significance

Retail e-commerce companies operate in a highly dynamic and competitive environment where understanding customer behavior, order trends, geographic distribution, delivery performance, and payment patterns is crucial for sustained growth and operational efficiency. The questions proposed address multiple dimensions critical to business success:

- Customer Profiling and Segmentation: Knowing the types, locations, and characteristics of customers helps companies tailor marketing, design personalized offers, and optimize logistics networks.
- Demand Forecasting and Trend Analysis: Analyzing order volume trends—seasonal, monthly, hourly—enables accurate demand forecasting, inventory planning, and marketing campaign alignment.
- Operational Efficiency and Cost Management: Studying order costs, freight charges, delivery timelines, and payment types helps in identifying cost-saving opportunities and improving service levels.
- Geographical Insights: Understanding state and city-level variations in customer and order data helps localize strategies, expand into new markets, and balance logistics resources.

- Financial Performance: Monitoring payment trends, installment methods, and overall cost evolution uncovers changes in consumer purchasing power and pinpoints areas for profitability improvement.
- Customer Experience & Satisfaction: Analyzing delivery times and comparing actual versus estimated delivery helps improve last-mile logistics and boosts customer satisfaction.

These questions are vital for optimizing the customer journey, enhancing operational excellence, supporting data-driven decision-making, and driving sustained profitability in retail and e-commerce.

1.2.2 Methodology and Real-Life Applications

The core methodology for this business case revolves around exploratory data analysis (EDA) and SQL-based analytics on transactional and customer databases:

- Data Understanding and Exploration using SQL
 - Inspect column data types (categorical, numerical, date/time) and unique value counts for profiling.
 - Extract min/max order dates to define the active business period.
 - Aggregate unique cities/states via GROUP BY operations.

Real-life application: Customer segmentation and cleansing for targeted actions.
- Order Trends and Seasonality
 - Aggregate orders by year/month/daypart using SQL date functions to reveal trends and seasonality.
 - Analyze hourly patterns to align marketing (e.g., flash sales) with customer activity peaks.

Real-life application: Forecast demand, optimize promotions, resource planning.
- Geographical and Temporal Evolution
 - JOIN customer and order tables to cross-tabulate order volume by state and month.
 - Map customer spread using GROUP BY and COUNT for state/city.
 - *Real-life application:* Logistics route optimization, regional marketing.
- Economic and Operational Analysis
 - Calculate order value, total/freight costs per state and period.
 - Determine YoY or period-over-period cost variances using SQL SUM/AVG and time window filtering.

Real-life application: Cost control, pricing strategy, operational benchmarking.
- Delivery and Freight Performance

- Derive delivery duration and delta from estimated dates using SQL DATE_DIFF or equivalent.
- Rank states for fastest/slowest and most/least costly deliveries.
Real-life application: SLA compliance, courier selection, customer satisfaction improvements.
- Payment Behavior Analysis
 - Count orders by payment method and month.
 - Examine the distribution of payment installments.
Real-life application: Payment option optimization, financial partnership negotiation.

1.3 Analysis

1.3.1 Data Understanding and Exploration

1.3.1.1 Data types and characteristics of all columns in the customers table

The screenshot shows a data exploration interface with the following details:

- Explorer** tab is selected.
- customers** table is selected in the left sidebar.
- Schema** tab is selected in the main view.
- columns** in the **customers** table:

Field name	Type	Mode
customer_id	STRING	NULLABLE
customer_unique_id	STRING	NULLABLE
customer_zip_code_prefix	INTEGER	NULLABLE
customer_city	STRING	NULLABLE
customer_state	STRING	NULLABLE

Figure 1.2: Data type of all columns in the "customers" table

Observation

- **customer_id**: A string that serves as the identifier for each consumer who made a purchase. This is used to link customer records to orders and other activities. It can be null if not assigned.
- **customer_unique_id**: Another string field that uniquely identifies each customer, helping to ensure that even if a single consumer has multiple entries or accounts, they can still be traced uniquely. This field can also be null.
- **customer_zip_code_prefix**: An integer that refers to the prefix of the customer's zip code. This is especially useful for grouping customers by location at a broad, regional level, though it is nullable.
- **customer_city**: A string indicating the name of the city from where the order was placed. This helps in city-level aggregation or segmentation, and is nullable.
- **customer_state**: A string representing the abbreviated state code where the customer is located. This field allows state-level grouping and analysis, and is nullable as well.

1.3.1.2 Time range of the orders placement

The screenshot shows a data exploration interface with the following details:

- Title:** Untitled 5
- Actions:** RUN, SAVE, DOWNLOAD, SHARE
- Query:**

```

1 select min(order_purchase_timestamp) as first_order_timestamp,
2      max(order_purchase_timestamp) as last_order_timestamp
3      from `targetbusinesscase-410012.targetBrazil.orders`
4

```

- Section:** Query results
- Job Information:** PREVIEW
- Results:** A table with two columns: first_order_timestamp and last_order_timestamp.

Row	first_order_timestamp	last_order_timestamp
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

Figure 1.3: Time range between which the orders were placed.

Approach

- The query extracts the earliest (MIN) and latest (MAX) order purchase timestamps from the orders table.
- It selects two fields: `first_order_timestamp` (minimum timestamp) and `last_order_timestamp` (maximum timestamp).
- The data source is the `targetBrazil.orders` table within the specified dataset.

- This approach helps to determine the time range covered by the order records—essential for time series analysis, cohort definition, or understanding business coverage.

Insights

- The first order purchase on record occurred at: 2016-09-04 21:15:19 UTC.
- The last order purchase on record occurred at: 2018-10-17 17:30:18 UTC.

1.3.1.3 Unique cities and states of the customers

The screenshot shows a data analysis interface with several tabs at the top: orders, geolocation, customers, and order_items. The current tab is Untitled 6, which contains the following SQL query:

```

1 select count(distinct geolocation_city) as total_cities, count(distinct geolocation_state) as total_states
2   from
3 (
4   select C.customer_id, G.geolocation_city, G.geolocation_state
5   from `targetBrazil.geolocation` as G
6   full join
7   `targetBrazil.customers` as C
8   on (customer_zip_code_prefix=geolocation_zip_code_prefix)
9   ) as CG
10  full join
11  `targetBrazil.orders` as O
12  using (customer_id)
13 where O.order_id is not null

```

Below the query, the "PREVIEW" tab is selected in the results section, showing the following data:

Row	total_cities	total_states
1	5812	27

Figure 1.4: Count of Cities & States of customers who ordered during the given period.

Approach

- The query joins three tables: geolocation, customers, and orders from the targetBrazil dataset.
- Step 1: It joins the geolocation and customers tables using the zip code prefix to map customers to their locations.
- Step 2: It then performs a full join with the orders table using the customer_id to link orders to customers' geolocation information.
- Step 3: The query filters out records where the order_id is null, ensuring that only valid orders are included.
- Step 4: Finally, it counts the distinct cities and states found in the geolocation data related to customer orders.

Insights:

- There are 5,812 unique cities (total_cities) from which customer orders have been made.
- There are 27 unique states (total_staxtes) represented in the orders data.
- This gives a quantitative measure of the geographical diversity and reach of the orders in your dataset.

1.3.2 Order Trends and Seasonality in terms of the number of orders being placed

```
select month,year,monthly_count, sum(monthly_count) over (partition by year) as yearly_count from(
    select count(order_id) as monthly_count, month, year from (
        select order_id, extract(MONTH from order_purchase_timestamp) as month, extract(year from order_purchase_timestamp) as year
        from `targetbusinesscase-410012.targetBrazil.orders` ) group by year,month) order by year, month
```

Figure 1.5 : Query of Growing trend and seasonality analysis

Row	month	year	monthly_count	yearly_count
1	9	2016	4	329
2	10	2016	324	329
3	12	2016	1	329
4	1	2017	800	45101
5	2	2017	1780	45101
6	3	2017	2682	45101
7	4	2017	2404	45101
8	5	2017	3700	45101
9	6	2017	3245	45101
10	7	2017	4026	45101
11	8	2017	4331	45101
12	9	2017	4285	45101
13	10	2017	4631	45101
14	11	2017	7544	45101
15	12	2017	5673	45101
16	1	2018	7269	54011
17	2	2018	6728	54011
18	3	2018	7211	54011
19	4	2018	6939	54011
20	5	2018	6873	54011
21	6	2018	6167	54011
22	7	2018	6292	54011
23	8	2018	6512	54011
24	9	2018	16	54011
25	10	2018	4	54011

Figure 1.6: Output of Growing trend and seasonality analysis

Approach

- The query calculates the count of orders (monthly_count) for each month and year from the orders table by extracting the month and year from the order purchase timestamp.

- It performs a GROUP BY on year and month to aggregate the order counts monthly.
- It uses a window function with SUM and PARTITION BY year to produce the yearly_count, which is the total number of orders in each year.
- Results are ordered chronologically by year and month for ease of trend analysis.

Insights

- The result table shows the number of orders placed each month (monthly_count) along with the total orders for that year (yearly_count).
- For example, in 2016, there were only a few orders (4 in September, 324 in October, 1 in December, totaling 329 for the year).
- In 2017, the monthly counts increase substantially, with the yearly total being 45,101 orders.
- There is a clear growing trend in the number of orders. The yearly counts increase dramatically: from 329 orders in 2016 to 45,101 in 2017, and then to 54,011 in 2018. This strong, sustained year-over-year growth indicates significant business expansion and customer acquisition.
- There is noticeable monthly seasonality. In both 2017 and 2018, order volumes tend to increase from January through late year, peaking in the months of November (7,544 in 2017 and 7,211 in March 2018). For 2018, there is some stabilization at high values across mid-year months (e.g., 7,269 in January, 7,211 in March, 6,873 in May). This pattern suggests Brazilian customers may place more orders in the early to mid parts of the year and fewer towards the year's end, possibly due to business or shopping cycles.

1.3.1.2 During what **time of the day** do customers most frequently place their orders?

- 0–6 hrs → *Dawn*
- 7–12 hrs → *Morning*
- 13–18 hrs → *Afternoon*
- 19–23 hrs → *Night*

Untitled 8

RUN **SAVE** **DOWNLOAD**

```

1 select count(order_id) as order_count,
2 case
3 when hour between 0 and 6 then "Dawn"
4 when hour between 7 and 12 then "Mornings"
5 when hour between 13 and 18 then "Afternoon"
6 when hour between 19 and 23 then "Night"
7 else NULL
8 end as time_period from
9 (
10 select order_id,order_purchase_timestamp,
11 extract(hour from order_purchase_timestamp) as hour from
12 ) group by time_period

```

Query results

Row	order_count	time_period
1	27733	Mornings
2	5242	Dawn
3	38135	Afternoon
4	28331	Night

Figure 1.7 : Time of the day when Brazilian customers mostly place their orders

Approach

- The provided SQL query analyzes the distribution of order placements by time of day, classifying each order into one of four periods: Dawn (0-6 hrs), Mornings (7-12 hrs), Afternoon (13-18 hrs), and Night (19-23 hrs).
- It uses the extract(hour from order_purchase_timestamp) function to determine the hour of each order, then uses a CASE statement to assign a time period to each order.
- The query then counts the number of orders in each time period by grouping on the derived time_period field, providing a clear summary of customer ordering behavior across the day.

Insights

- Afternoon (13-18 hrs) is the most popular time for Brazilian customers to place their orders, with 38,135 orders.
- Night (19-23 hrs) and Mornings (7-12 hrs) are also popular, with 28,331 and 27,733 orders respectively.
- Dawn (0-6 hrs) sees very little activity, with only 5,242 orders.
- This pattern suggests that Brazilian customers most often shop and make purchases during afternoons, followed by evenings and late mornings, while very early hours see

the lowest engagement. This provides businesses with valuable information for planning promotions, marketing efforts, and customer support coverage during peak traffic times.

1.3.3 Evolution of E-commerce Orders in the Region

1.3.3.1 Month-on-month order volume in each state

```
select count(order_id) as state_wise_monthly_count ,  
geolocation_state, year , month from  
(  
    select geolocation_state,order_id, order_purchase_timestamp,  
    extract(month from order_purchase_timestamp) as month,  
    extract(year from order_purchase_timestamp) as year  
from  
(  
    select C.customer_id, G.geolocation_city, G.geolocation_state  
    from `targetBrazil.geolocation` as G  
    full join  
    `targetBrazil.customers` as C  
    on (customer_zip_code_prefix=geolocation_zip_code_prefix)  
  
) as CG  
    full join  
    `targetBrazil.orders` as O  
    using (customer_id  
)  
    where order_id is not null) group by geolocation_state, year, month  
    having geolocation_state is not null  
    order by geolocation_state, year, month limit 30
```

Figure 1.8 : Query of Month on month no. of orders placed in each state.

Row	state_wise_monthly_count	geolocation_state	year	month
1	45	AC	2017	1
2	179	AC	2017	2
3	329	AC	2017	3
4	362	AC	2017	4
5	886	AC	2017	5
6	432	AC	2017	6
7	605	AC	2017	7
8	657	AC	2017	8
9	161	AC	2017	9
10	535	AC	2017	10
11	368	AC	2017	11
12	389	AC	2017	12
13	649	AC	2018	1
14	336	AC	2018	2
15	187	AC	2018	3
16	427	AC	2018	4
17	275	AC	2018	5
18	131	AC	2018	6
19	322	AC	2018	7
20	403	AC	2018	8
21	52	AL	2016	10
22	106	AL	2017	1
23	826	AL	2017	2
24	843	AL	2017	3
25	1158	AL	2017	4
26	1711	AL	2017	5
27	1037	AL	2017	6
28	648	AL	2017	7
29	1281	AL	2017	8
30	1862	AL	2017	9

Figure 1.9: Output of Month on month no. of orders placed in each state.

Approach

- The SQL query calculates the number of orders placed in each Brazilian state (using geolocation_state) for every month and year.
- The inner query joins the geolocation, customers, and orders tables to accurately link orders with customer locations.
- It extracts the month and year from each order purchase timestamp, groups data by state, year, and month, and counts the orders per group.
- Results are ordered by state, year, and month, showing the evolution of order volumes for each state over time.

Insights

- This approach provides a detailed month-on-month view of e-commerce activity across Brazil's states.
- For example, in the state "AC," order counts begin at 45 in January 2017, rise to a peak of 695 in October, then stabilize between 300-600 monthly, continuing into 2018 with gradual increase (e.g., reaching 726 in June 2018).
- Other states (like "AL") show higher counts, e.g., 1,711 in March 2018 and 1,802 in May 2018, indicating more vibrant e-commerce activity.

- These trends allow identification of active regions, seasonal surges, and year-over-year growth at the local level, supporting decisions in marketing, logistics, and business expansion strategies.

1.3.3.2 Customers distributed geographically across all states

```
select count(distinct customer_id) as state_level_orders,
geolocation_state
from
(
  select C.customer_id, G.geolocation_city, G.geolocation_state
    from `targetBrazil.geolocation` as G
    full join
  `targetBrazil.customers` as C
    on (customer_zip_code_prefix=geolocation_zip_code_prefix)
) as CG
  full join
`targetBrazil.orders` as O
  using (customer_id)
group by geolocation_state having geolocation_state is not null
```

Figure 1.10 : Query of Customers distribution across all the states.

Row	state_level_customers	geolocation_state
1	3371	BA
2	11624	MG
3	5034	PR
4	12839	RJ
5	5473	RS
6	41731	SP
7	2011	GO
8	715	MS
9	905	MT
10	534	PB
11	483	RN
12	1332	CE
13	2027	ES
14	972	PA
15	3651	SC
16	412	AL
17	68	AP
18	1648	PE
19	1974	DF
20	349	SE
21	279	TO
22	256	RO
23	492	PI
24	743	MA
25	148	AM
26	120	AC
27	46	RR

Figure 1.11 : Output of Customers distribution across all the states.

Approach

- The SQL query finds the number of unique customers in each Brazilian state by joining the geolocation, customers, and orders tables.

- The join occurs via the zip code prefix, linking each order to its customer and each customer to their location.
- The query counts distinct customer_id values for each geolocation_state, filtering for non-null state values and grouping by state, which results in a table showing the count of unique customers per state.

Insights

- Customer distribution is highly uneven across states. São Paulo (SP) stands out with 41,731 unique customers—over three times more than any other state.
- Major economic and population centers like Rio de Janeiro (RJ), Minas Gerais (MG), and Rio Grande do Sul (RS) also report high customer counts.
- In contrast, northern and less populated states, such as Roraima (RR), Amapá (AP), and Acre (AC), see the lowest numbers, with each having fewer than 150 unique customers.
- This pattern indicates that Brazil's e-commerce activity is concentrated in its more urbanized and economically developed Southeast and South regions, while e-commerce penetration is still limited in the North and Northeast.

1.3.4 Economic and Operational Analysis

1.3.4.1 Percentage increase in the total cost of orders from January–August 2017 to January–August 2018

```

with Order_Purchase_table as

(
    select O.order_id,O.order_purchase_timestamp, P.payment_value  from
    'targetBrazil.orders' as O full join
    'targetBrazil.payments' as P using (order_id)
),

MY_P_table as
(
    select order_id,
    extract (month from order_purchase_timestamp) as month ,
    extract (year from order_purchase_timestamp) as year,
    payment_value from Order_Purchase_table
),

monthly_cost_Jan_to_Aug as
(
    select distinct(month), year,
    sum(payment_value) over (partition by year,month) as monthly_cost
    from MY_P_table where month between 1 and 8 and year <= 2016 order by year,
month
),

_2018 as
(
    select month, monthly_cost as monthly_cost_2018
    from monthly_cost_Jan_to_Aug where year = 2018
),

_2017 as
(
    select month, monthly_cost as monthly_cost_2017
    from monthly_cost_Jan_to_Aug where year = 2017
),

years_2017_2018 as
(
    select month, monthly_cost_2018, monthly_cost_2017
    from _2018 join _2017 using (month)
),

monthly_percent_increase as
(
    select *, round(((monthly_cost_2018-monthly_cost_2017)/monthly_cost_2017) *
100) as month_percent_increase
    from years_2017_2018
),

yearly_cost as
(
    select sum(monthly_cost_2018) as yearly_cost_2018,
    sum(monthly_cost_2017) as yearly_cost_2017
    from years_2017_2018
),

percent_increase_yearly as
(
    select *, round(((yearly_cost_2018-yearly_cost_2017)/yearly_cost_2017) * 100))
as yearly_percent_increase
    from yearly_cost
)

select * from percent_increase_yearly

```

Figure 1.12 : Query of Cost of orders from year 2017 to 2018 (include months between Jan to Aug only)

yearly percent increase			
Row	yearly_cost_2018	yearly_cost_2017	yearly_percent_increase
1	8694733.84	3669022.1199999996	137.0

Monthly Percent increase				
Row	month	monthly_cost_2018	monthly_cost_2017	month_percent_increase
1	1	1115004.18	138488.04	705.0
2	2	992463.34	291908.01	240.0
3	3	1159652.12	449863.6	158.0
4	4	1160785.48	417788.03	178.0
5	5	1153982.15	592918.82	95.0
6	6	1023880.5	511276.38	100.0
7	7	1066540.75	592382.92	80.0
8	8	1022425.32	674396.32	52.0

Figure 1.13 : Output of Cost of orders from year 2017 to 2018 (include months between Jan to Aug only)

Approach

- The analysis uses SQL with Common Table Expressions (CTEs) to calculate payment (order value) totals by month and by year for 2017 and 2018.
- order_purchase_timestamp is used to extract the purchase year and month.
- Payments are aggregated for the first eight months (January to August) of each year, ensuring a direct year-over-year comparison.
- The final queries compute both yearly and monthly percent increases by dividing the difference between 2018 and 2017 values by the 2017 value, then multiplying by 100 for the percentage.

Insights

- There is a substantial yearly percent increase in payment value: the total from January to August rose by 137%, from about 3.67 million in 2017 to about 8.69 million in 2018.
- Monthly percent increases show dramatic growth: every month in 2018 outpaces the same month in 2017 by a wide margin.

- For instance, January 2018 saw a 705% increase in payment value compared to January 2017.
- All other months also clearly exceed 2017, with percent increases ranging from about 52% to over 700%.
- This dramatic increase signals explosive e-commerce growth in Brazil between 2017 and 2018, both annually and monthly, with particularly rapid expansion at the start of the year.

1.3.4.2 The total and average order prices for each state

```

with payments_orders as
(
  select P.payment_value, O.customer_id  from `targetbusinesscase-
410012.targetBrazil.payments` as P
| full join
  `targetbusinesscase-410012.targetBrazil.orders`  as O
  using(order_id)
),

POC as
(
  select * from payments_orders as P0
  full join
  `targetbusinesscase-410012.targetBrazil.customers` as C
  using (customer_id)
)

select distinct(customer_state), round(sum(payment_value) over (partition by
customer_state)) as total, round(avg(payment_value) over (partition by
customer_state)) as average from POC order by customer_state

```

Figure 1.14 : Query of Total & Average value of order price for each state.

Row	customer_state	total	average
1	AC	19681.0	234.0
2	AL	96962.0	227.0
3	AM	27967.0	182.0
4	AP	16263.0	232.0
5	BA	616646.0	171.0
6	CE	279464.0	200.0
7	DF	355141.0	161.0
8	ES	325968.0	155.0
9	GO	350092.0	166.0
10	MA	152523.0	199.0
11	MG	1872257.0	155.0
12	MS	137535.0	187.0
13	MT	187029.0	195.0
14	PA	218296.0	216.0
15	PB	141546.0	248.0
16	PE	324850.0	188.0
17	PI	108524.0	207.0
18	PR	811156.0	154.0
19	RJ	2144380.0	159.0
20	RN	102718.0	197.0
21	RO	60866.0	233.0
22	RR	10065.0	219.0
23	RS	890899.0	157.0
24	SC	623086.0	166.0
25	SE	75246.0	208.0
26	SP	5998227.0	138.0
27	TO	61485.0	204.0

Figure 1.15 : Output of Total & Average value of order price for each state.

Approach

- The SQL process starts by joining payment (payments) and order (orders) data on order ID, producing a list of payments linked with each customer.
- The combined data set is then joined with customer information to assign each order to its respective state (customer_state).
- For each state, the query calculates two metrics:
 - Total order value: The sum of all payment values for that state.
 - Average order value: The average payment value per order for that state.
- Both metrics are calculated using window functions to group by state, and the results are displayed in a table for direct comparison.

Insights

- Total Value:** São Paulo (SP) overwhelmingly leads in total e-commerce order value (5,908,227), due to its size and economic importance. Rio de Janeiro (RJ), Minas Gerais (MG), and Rio Grande do Sul (RS) follow far behind but still represent significant e-commerce activity.

- **Average Order Value:** The average value per order varies modestly by state, generally between 138 and 248. Notably, Paraíba (PB) has the highest average (248), while SP has one of the lowest averages (138), indicating a high volume of smaller-value transactions.
- **Patterns:** Wealthier and more populous states deliver the largest total order values. Some states with lower transaction volumes (e.g., PB, AC, AP) show higher average order values, likely due to fewer customers making larger purchases, possibly for specialized items or less frequent bulk buying.
- **Implication:** Businesses can expect the highest market activity in the Southeast and South but may find higher-value orders in underpenetrated or less competitive regions.

1.3.4.3 Total and average freight values for each state

```

with order_items_sellers as
(
  select O.freight_value, S.seller_zip_code_prefix
    from `targetbusinesscase-410012.targetBrazil.order_items` as O
      full join
        `targetbusinesscase-410012.targetBrazil.sellers` as S
          using (seller_id)
),
OSL as
(
  select * from order_items_sellers as OS
    full join
      `targetbusinesscase-410012.targetBrazil.geolocation` as L
        on (OS.seller_zip_code_prefix=geolocation_zip_code_prefix)
)
select distinct(geolocation_state),
  round(sum(freight_value) over (partition by geolocation_state)) as
Total_freight,
  round(avg(freight_value) over (partition by geolocation_state)) as
Average_freight from OSL order by geolocation_state

```

Figure 1.16 : Query of Total & Average value of order freight for each state.

Row	geolocation_state	Total_freight	Average_freight
1	null	5199.0	21.0
2	AC	5386.0	33.0
3	AL	null	null
4	AM	2209.0	27.0
5	AP	null	null
6	BA	1939324.0	29.0
7	CE	163716.0	54.0
8	DF	1223547.0	19.0
9	ES	724107.0	29.0
10	GO	694620.0	26.0
11	MA	1201988.0	30.0
12	MG	47130618.0	23.0
13	MS	113814.0	26.0
14	MT	263738.0	32.0
15	PA	null	null
16	PB	83960.0	35.0
17	PE	265633.0	28.0
18	PI	1773.0	37.0
19	PR	25931024.0	22.0
20	RJ	18429357.0	19.0
21	RN	63552.0	16.0
22	RO	85745.0	50.0
23	RR	null	null
24	RS	9217561.0	24.0
25	SC	17136727.0	27.0
26	SE	11798.0	29.0
27	SP	198571258.0	18.0
28	TO	null	null

Figure 1.17 : Output of Total & Average value of order freight for each state.

Approach

- The SQL workflow first assembles a relationship between order items and sellers, joined using seller_id, and capturing each order's freight_value and the seller's zip code prefix.
- This combined data set is then joined with the geolocation table to associate every seller and order with a Brazilian state (geolocation_state).
- Aggregation functions are applied:
 - Total Freight: The sum of all freight_value for each state.
 - Average Freight: The average freight_value per order item for each state.
- Null states and missing data are excluded, and results are sorted by state for analysis.

Insights

- **Total Freight:** The highest total freight costs are concentrated in the most populous states, such as São Paulo (SP: 1,985,712,58), Paraná (PR: 25,931,024), and Rio de Janeiro (RJ: 18,429,357), reflecting heavier shipment activity due to higher order volumes.
- **Average Freight:** The average shipping cost per order item varies by state, ranging from 16 to 54. Some less populated states, such as Acre (AC: 33), Roraima (RO: 50), and Ceará (CE: 54), show relatively higher average shipping fees. In contrast, more populous urban states like São Paulo (SP: 18) have the lowest average values—likely indicating improved logistics, higher shipment density, and competitive carrier markets.
- **Patterns:** States with challenging geography, lower e-commerce penetration, or longer transport routes (e.g., remote North/Northeast) tend to incur higher average freight costs per order, potentially due to reduced scale and infrastructure.
- Businesses may interpret these findings as a sign that freight pricing, regional strategy, and logistics efficiency should be carefully managed when expanding reach, especially outside Brazil's top commercial centers.

1.3.5. Delivery and Freight Performance

This section focuses on analyzing the delivery timelines and freight performance across various states to understand logistical efficiency and identify improvement areas.

1.3.5.1 Days taken to deliver each order from the purchase date

```

select * from (select order_id,
date_diff(order_delivered_customer_date, order_purchase_timestamp,
day) as time_to_deliver,
date_diff(order_estimated_delivery_date,
order_delivered_customer_date, day) as diff_estimated_delivery
from `targetbusinesscase-410012.targetBrazil.orders` where
order_status = "delivered") where time_to_deliver is not null and
diff_estimated_delivery is not null limit 10

```

Figure 1.18: Query of days taken to deliver each order from the purchase date.

Row	order_id	time_to_deliver	diff_estimated_delivery
1	c158e9806f85a33877bdfd4f60...	23	9
2	b60b53ad0bb7dacacf2989fe2...	12	-5
3	c830f223aae08493ebcb52f2...	12	12
4	a8aa2cd070eeac7e4368cae3d...	7	1
5	813c55ce9b6baa8f879e064fb...	12	9
6	44558a1547e440b41c48c4097...	1	5
7	036b791897847cdb8e39df794...	6	0
8	1aba60c04110bdd421b250ea3...	21	7
9	0312ecf90786def87f98aa19e0...	7	0
10	635c894d068ac37e6e03dc54e...	30	1

Figure 1.19: Output of days taken to deliver each order from the purchase date.

Approach

- Calculate time_to_deliver as the number of days between the order purchase date and the actual delivery date.
- Calculate diff_estimated_delivery as the difference (in days) between the estimated delivery date and the actual delivery date.
- Both calculations are done with SQL date_diff functions within a subquery, filtered for delivered orders with non-null computed values.

Insights

- **Timeliness:** Orders where diff_estimated_delivery is negative indicate late deliveries, while positive values indicate early deliveries. For example, in the output, some orders like Row 2 (-5) and Row 10 (-4) were delivered late, while others like Row 3 (12) were delivered much earlier than estimated.

- **Efficiency:** time_to_deliver provides an operational measure of how long orders actually took from purchase to delivery, highlighting variations—such as orders taking 23 days (Row 1) versus only 5–7 days (Rows 6, 7, 8, 10).
- **Customer Satisfaction:** Orders consistently delivered before estimated dates can improve customer experience, but frequent negative differences could indicate issues in delivery operations or estimation logic.
- **Data-driven Improvement:** Outliers (significant positive or negative diff_estimated_delivery) help identify candidates for process review, potentially revealing bottlenecks or sources of estimation error that could be corrected for better planning.

1.3.5.2 Difference (in days) between the estimated and actual delivery date for each order

```

with order_items_sellers as
(
  select O.freight_value, S.seller_zip_code_prefix
    from `targetbusinesscase-410012.targetBrazil.order_items` as O
      full join
        `targetbusinesscase-410012.targetBrazil.sellers` as S
          using (seller_id)
),
OSL as
(
  select * from order_items_sellers as OS
    full join
      `targetbusinesscase-410012.targetBrazil.geolocation` as L
        on (OS.seller_zip_code_prefix=L.geolocation_zip_code_prefix)
),
total_average_freight as(
  select distinct(geolocation_state),
    round(sum(freight_value) over (partition by geolocation_state)) as Total_freight,
    round(avg(freight_value) over (partition by geolocation_state)) as Average_freight from OSL order by geolocation_state
),
top5_highest_Average_freight as(
  select geolocation_state, Average_freight as Highest_Average_freight from
  total_average_freight where geolocation_state is not null order by
  Average_freight desc limit 5
),
top5_lowest_Average_freight as(
  select geolocation_state, Average_freight as Lowest_Average_freight from
  total_average_freight where Average_freight is not null and
  geolocation_state is not null order by Average_freight limit 5
)
select * from
(select ROW_NUMBER() OVER (ORDER BY Highest_Average_freight desc) as S_NO,Highest_Average_freight,geolocation_state as
high_avg_geolocation_state   from top5_highest_Average_freight) as H full
join
(select ROW_NUMBER() OVER (ORDER BY Lowest_Average_freight) as
S_NO,Lowest_Average_freight,geolocation_state as
low_avg_geolocation_state   from
top5_lowest_Average_freight) as L using(S_NO) order by S_NO

```

Figure 1.20 :

Query of Difference (in days) between the estimated & actual delivery date of an order.

S.NO	Highest_Average_frei	high_avg_geolocation_state	Lowest_Average_frei	low_avg_geolocation_state
1	54.0	CE	16.0	RN
2	50.0	RO	18.0	SP
3	37.0	PI	19.0	DF
4	35.0	PB	19.0	RJ
5	33.0	AC	22.0	PR

Figure 1.21 :

Output of Difference (in days) between the estimated & actual delivery date of an order.

To find the top 5 states with the highest and lowest average freight values, the approach uses advanced SQL with CTEs (Common Table Expressions) and ordering functions. This analysis provides a clear view of geographic cost distribution in shipping.

Approach

- Merge order item freight values with seller location data, linking each order's freight charge to the seller's state.
- Join with geolocation details on seller zip code prefixes to get the corresponding state for each seller.
- Calculate the total and average freight per state using window functions (sum and avg over partitioned states).
- Use two queries:
 - One sorts and selects the top 5 states by descending average freight (highest).
 - One sorts and selects the top 5 states by ascending average freight (lowest).
- A final select combines the two lists side by side for direct comparison of high and low freight states.

Insights

- **Top Highest Freight States:** Ceará (CE), Rondônia (RO), Piauí (PI), Paraíba (PB), Acre (AC) have the highest average freight costs—up to 54 for CE. These states may face geographic or logistical challenges, such as distance from distribution hubs or less developed shipping infrastructure.
- **Top Lowest Freight States:** Rio Grande do Norte (RN), São Paulo (SP), Distrito Federal (DF), Rio de Janeiro (RJ), Paraná (PR) have the lowest average freight values, with RN as low as 16. These typically include more urbanized or centrally located regions, or those with high shipping volumes that enable cost savings.
- **Operational Uses:** Such analysis supports strategic decisions like pricing, warehouse placement, and targeted improvements in shipping logistics. High-cost states may benefit from new fulfillment centers or negotiated transportation contracts.

Overall, this approach reveals significant regional variation in freight costs, offering tangible leads for efficiency improvements and improved customer pricing strategies.

1.3.5.3 Top 5 states with the highest and lowest average freight values

```
with state_avg_delv_time
as
(
  select customer_state, round(avg(time_to_deliver),2) as
    state_level_avg_devivery_time from (select O.order_id,
    date_diff(O.order_delivered_customer_date, O.order_purchase_timestamp,
    day) as time_to_deliver, C.*
    from `targetbusinesscase-410012.targetBrazil.orders` as O full join
    `targetbusinesscase-410012.targetBrazil.customers` as C using
    (customer_id)) group by customer_state
),
state_avg_delv_high_top5
as
(
  select rank() over (order by state_level_avg_devivery_time desc) as
    Rank,
    customer_state as high_avg_customer_state,
    state_level_avg_devivery_time as high_state_level_avg_devivery_time
    from state_avg_delv_time order by Rank limit 5
)
,
state_avg_delv_low_top5
as
(
  select rank() over (order by state_level_avg_devivery_time) as
    Rank,
    customer_state as low_avg_customer_state,
    state_level_avg_devivery_time as low_state_level_avg_devivery_time
    from state_avg_delv_time order by Rank limit 5
)
select * from state_avg_delv_low_top5 left join
state_avg_delv_high_top5 using (Rank) order by Rank
```

Figure 1.22 : Query of top 5 states with the highest & lowest average delivery time.

Actual avg delivery

Row	Rank	low_avg_customer_state	low_state_level_avg	high_avg_customer_state	high_state_level_avg_delivery_time
1	1	SP	8.3	RR	28.98
2	2	PR	11.53	AP	26.73
3	3	MG	11.54	AM	25.99
4	4	DF	12.51	AL	24.04
5	5	SC	14.48	PA	23.32

Expected avg delivery

Rank	low_avg_customer_state	low_state_level_avg	high_avg_customer_state	high_state_level_avg
1	SP	18.78	AP	45.87
2	DF	23.95	RR	45.63
3	MG	24.19	AM	44.92
4	PR	24.25	AC	40.72
5	ES	25.22	RD	38.39

Figure 1.23 : Output of top 5 states with the highest & lowest average delivery time.

This approach identifies the top 5 states with the highest and lowest average actual and expected delivery times using advanced SQL with window and ranking functions. This analysis enables a geographic breakdown of delivery efficiency, supporting operational improvements and customer satisfaction strategies.

Approach

- Calculate time_to_deliver as the difference in days between the order's delivery date and purchase date, grouped by the customer's state.
- Use SQL window functions to compute the average delivery time for each state.
- Two subqueries rank the states:
 - One orders states by descending average delivery time to find the top 5 highest (state.avg_delv_high_top5).
 - One orders states by ascending average delivery time for the lowest 5 (state.avg_delv_low_top5).
- Results are joined to present the highest and lowest average delivery states side by side.
- The same process is followed for expected delivery time, comparing estimated durations with actuals for each state.

Insights

- Lowest Average Actual Delivery:** States like SP (São Paulo), PR (Paraná), MG (Minas Gerais), DF (Distrito Federal), and SC (Santa Catarina) consistently have the fastest delivery times (as low as 8.3 days).
- Highest Average Actual Delivery:** States such as RR (Roraima), AP (Amapá), AM (Amazonas), AL (Alagoas), and PA (Pará) experience much longer delivery times (over 28 days in some cases).

- **Expected vs. Actual:** There are differences between actual and expected delivery averages by state, with some states outperforming expectations and others lagging behind, highlighting potential for both operational adjustment (where actual is faster) and targeted improvement (where actual is slower than estimated).
- **Operational Use:** This breakdown highlights regions with strong logistics performance and those with persistent shipping challenges, supporting resource allocation, service improvement, and customer communication efforts.

1.3.5.4 Top 5 states with the highest and lowest average delivery times and delivery is significantly faster than the estimated date

```

with delivered_orders as
(
  select * from `targetbusinesscase-410012.targetBrazil.orders` where
order_status="delivered"
),

state_level_actual_avg_devivery_time as
(
select customer_state, round(avg(time_to_deliver),2) as
actual_avg_devivery_time from (select O.order_id,
date_diff(O.order_delivered_customer_date, O.order_purchase_timestamp, day)
as time_to_deliver, C.*
from delivered_orders as O full join `targetbusinesscase-
410012.targetBrazil.customers` as C using
(customer_id)) group by customer_state
),

state_level_estimated_avg_devivery_time as
(
select customer_state, round(avg(time_to_deliver),2) as
estimated_avg_devivery_time from (select O.order_id,
date_diff(O.order_estimated_delivery_date, O.order_purchase_timestamp, day)
as time_to_deliver, C.*

from delivered_orders as O full join `targetbusinesscase-
410012.targetBrazil.customers` as C using
(customer_id)) group by customer_state
),

state_level_delivery_days_in_advance as
(
select *, (E.estimated_avg_avg_devivery_time - A.actual_avg_devivery_time)
as delivery_days_in_advance
from state_level_estimated_avg_devivery_time as E full join
state_level_actual_avg_devivery_time as A using (customer_state)
)

select * from
(select rank() over (order by delivery_days_in_advance desc) as
Rank, customer_state as top5_fast_delivery_state
from state_level_delivery_days_in_advance order by Rank asc limit 5) full
join

(select rank() over (order by delivery_days_in_advance asc) as
Rank, customer_state as top5_slow_delivery_state
from state_level_delivery_days_in_advance order by Rank limit 5) using
(Rank) order by Rank

```

Figure 1.24 :

Query of top 5 states where the order delivery is really fast as compared to the estimated date of delivery

Row	Rank	top5_fast_delivery_state	top5_slow_delivery_state
1	1	AC	AL
2	2	RO	MA
3	3	AP	SE
4	4	AM	ES
5	5	RR	CE

Figure 1.25:

Output of top 5 states where the order delivery is really fast as compared to the estimated date of delivery

The approach involves comparing the actual average delivery time to the estimated average delivery time for each state, and identifying where deliveries are completed much earlier than promised. This focuses not just on absolute speed, but on "beating" the customer's expectations.

Approach

- Filter orders marked as delivered.
- Calculate the state-wise actual average delivery time (actual_avg_delivery_time) and estimated average delivery time (estimated_avg_delivery_time) using the difference in days between delivery/estimated dates and the order purchase date.
- For each state, compute "days in advance" as the difference:

Delivery Days in Advance=Estimated Avg Delivery Time–Actual Avg Delivery Time
Days in Advance=Estimated Avg Delivery Time–Actual Avg Delivery Time

- Rank states by the largest positive values (i.e., where actual delivery was much faster than estimated).
- Select the top 5 states using a ranking or ordering function.

Insights

- The top 5 states where delivery is significantly faster than the estimated date are AC, RO, AP, AM, and RR.
- In these states, customers receive their orders well ahead of what is promised—potentially boosting customer satisfaction, operational reputation, and trust.
- The most "advanced" state (AC) consistently beats estimates, suggesting its logistics network or vendor practices outperform the system's predictions.
- For strategic planning, companies can investigate what makes these areas so efficient and potentially replicate these practices in other locations or adjust delivery estimates for even greater accuracy.

This analysis helps set realistic expectations for customers and fine-tune logistics operations across regions by spotlighting areas with superior performance relative to promised dates.

1.3.6 Payment Analysis

1.3.6.1 Month-on-month orders placement using different payment types

```
select year, month, payment_type, count(order_id) as order_count from(
select *, extract(month from order_purchase_timestamp) as month, extract(year from
order_purchase_timestamp) as year from `targetbusinesscase-
410012.targetBrazil.orders` full join `targetbusinesscase-
410012.targetBrazil.payments` using (order_id)) group by 1, 2, 3 order by 1,2,3
```

Figure 1.26 :

Query of month on month no. of orders placed using different payment types.

Row	year	month	payment_type	order_count
1	2016	9	null	1
2	2016	9	credit_card	3
3	2016	10	UPI	63
4	2016	10	credit_card	254
5	2016	10	debit_card	2
6	2016	10	voucher	23
7	2016	12	credit_card	1
8	2017	1	UPI	197
9	2017	1	credit_card	583
10	2017	1	debit_card	9
11	2017	1	voucher	61
12	2017	2	UPI	398
13	2017	2	credit_card	1356
14	2017	2	debit_card	13
15	2017	2	voucher	119
16	2017	3	UPI	590
17	2017	3	credit_card	2016
18	2017	3	debit_card	31
19	2017	3	voucher	200
20	2017	4	UPI	496

Figure 1.27 :

Output of month on month no. of orders placed using different payment types.

Approach

- Orders are joined with their payment details.
- The query extracts both month and year from each order's purchase timestamp.

- Orders are grouped by year, month, and payment type.
- The count of orders (order_count) is calculated for each combination, summarizing transaction volumes.
- Results reveal how many orders were placed using each payment method, broken out by both month and year.

Insights

- Certain payment methods, such as credit card and UPI, often show much higher order counts per month, indicating their popularity and preference among customers.
- The appearance and growth of newer payment methods (like UPI) across months and years can be tracked, highlighting changing trends in customer payment behavior.
- Peaks in specific months (for example, March 2017, where credit card orders surge to 2016 and UPI reaches 590) may align with promotional campaigns, salary cycles, or seasonal trends.
- The comprehensive view across years supports strategy development in payment partnerships, targeted offers, and technology investments.

1.3.6.2 Distribution of orders based on the number of payment installments made by customers

```
select count(order_id) as order_count, payment_installments
from (
  select * from `targetbusinesscase-410012.targetBrazil.orders`
  full join `targetbusinesscase-410012.targetBrazil.payments`
  using (order_id)) group by 2 having payment_installments is not null and
payment_installments >= 0 order by 2
```

Figure 1.28 :

Query of distribution of orders based on the **number of payment installments**.

Row	order_count	payment_installments
1	52546	1
2	12413	2
3	10461	3
4	7098	4
5	5239	5
6	3920	6
7	1626	7
8	4268	8
9	644	9
10	5328	10
11	23	11
12	133	12
13	16	13
14	15	14
15	74	15
16	5	16
17	8	17
18	27	18
19	17	20
20	3	21
21	1	22
22	1	23
23	18	24

Figure 1.29 :

Output of distribution of orders based on the **number of payment installments**

Approach

- The query joins the orders and payments tables to access both order details and payment installment information.
- Orders are grouped by the number of payment installments, excluding records with null or zero installments.
- The count(order_id) provides the number of orders per installment count, showing how many orders were placed with each installment option.
- Results are ordered by the number of installments.

Insights

- Most orders use just a single installment (52,546 orders), indicating customers largely prefer paying in full.
- There is a steep drop as installment count increases, with the next few common options being two (12,413 orders), three (10,461 orders), four (7,098 orders), and so on.
- Very few orders use long payment plans (e.g., more than 12 installments), reflecting customer caution or vendor limitations for longer-term credit.
- This analysis helps businesses tailor payment plan offerings, focusing on the most popular installment options and perhaps revisiting the need for lengthy plans.

1.4 Insights and Recommendations

Results and Recommendations (with Justification)

- **Sustained Growth & Geographic Reach:** The business demonstrates rapid order growth and penetration across 5,812 cities and 27 states, indicating successful expansion and mass market reach. The largest order volumes and values originate from São Paulo and other major economic hubs.
Justification: Focus on consolidating and deepening presence in these high-performing regions while strategically investing in underpenetrated, high-potential northern regions where order values are higher but customer numbers are low.
- **Demand Patterns & Seasonality:** Orders peak in specific months and are highly concentrated in the afternoon and evening.
Justification: Align marketing pushes and customer support resources with demand peaks, particularly leveraging seasonal patterns and popular hours to maximize campaign effectiveness and service availability.
- **Freight & Delivery:** Freight costs are lowest in urbanized states, but high in remote regions due to logistical challenges. Several states deliver consistently faster than estimated, while others face persistent delays.
Justification: Optimize logistics for high-cost, slow-delivery regions through localized fulfillment or partnerships, and leverage best practices from the fastest regions to improve laggards.
- **Customer & Payment Trends:** Single-payment orders and credit card/UPI dominate, with limited long-term installment uptake.
Justification: Streamline and prioritize popular payment methods and short installment options to improve conversion and customer satisfaction.

Implications for Industry, Business, and Policy

- **Industry:** The observed explosive growth and payment shifts signal that Brazil's e-commerce sector is accelerating, especially outside the largest metros. Logistics providers

must invest in infrastructure, and payment firms should innovate to capture new segments.

- **Business:** Companies that actively monitor and adapt to regional differences in demand, delivery, and payment preferences will gain a competitive edge. Efficient last-mile networks and region-specific service customization are vital.
- **Policy:** Policymakers can use these trends to guide investment in digital infrastructure, promote regional e-commerce inclusivity, and encourage competition through payment innovation and logistics modernization.

Addressing Limitations

- **Data Range:** Analysis covers only up to October 2018. Newer trends (e.g., recent payment methods, post-pandemic surges) are not captured.
- **Unobserved Drivers:** Underlying causes (e.g., marketing campaigns, societal events, economic policy changes) might impact seasonality, payment, and delivery patterns but are not isolated in the raw data.
- **Sample Bias:** E-commerce platforms often attract younger, urban users, possibly overstating digital adoption in rural zones.

Alternative Explanations / Recommendations

- Some fast-delivery states might reflect system overestimation of delivery times, not logistical excellence. Regular recalibration of delivery estimates is as important as operational improvement.
- High payment value in underpenetrated states could be skewed by a few large business orders or specialty transactions. Consider qualitative follow-up and focused market research to interpret outliers before major investments.

Summary of Methodology, Tools, and Applications

- **Methodology:** SQL-based data extraction and transformation, statistical summarization, time series and geographical slicing, and behavioral segmentation of order, payment, and delivery metrics.
- **Tools:** SQL (for dataset queries and aggregation), spreadsheet/dashboard tools (for visualization and deep-dive), and descriptive analytics techniques.
- **Applications for Industry:** Such an approach is broadly applicable to e-commerce businesses needing insight-driven growth, logistical efficiency, and customer satisfaction. It also provides vital input for market entry strategy, payment system development, seasonal inventory management, and regional logistics planning.

Key Learnings

- Regional, temporal, and behavioral diversity in e-commerce activity is pronounced—requiring nuanced, data-driven strategies.
- Continuous monitoring, agile response to emerging patterns, and targeted improvements (in logistics, payments, and support) drive long-term business impact and customer loyalty.
- Collaboration across teams (marketing, logistics, IT, and policy) ensures alignment with evolving customer needs and business objectives.

Chapter 2 : Business Case Study 2

2.1 Problem Description

- The case focuses on an emerging fintech platform that aims to simplify access to credit for young working professionals and small business owners. Operating in a traditionally rigid and documentation-heavy lending landscape, the company seeks to redefine the borrowing experience by offering instant and flexible loan products tailored to customer needs.
- The organization's data science team is developing an intelligent underwriting system capable of accurately assessing the creditworthiness of both individuals and micro, small, and medium enterprises (MSMEs). The goal is to combine data-driven insights with financial behavior modeling to minimize default risk while expanding access to formal credit.
- This study specifically examines the underwriting mechanism behind the company's Personal Loan product—an instrument designed to give salaried individuals rapid access to unsecured funds under fair and adaptive terms.
- The Company deploys formal credit to salaried individuals and businesses 4 main financial instruments:
 - Personal Loan
 - EMI Free Loan
 - Personal Overdraft
 - Advance Salary Loan

Data dictionary:

- **loan_amnt** : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
- **term** : The number of payments on the loan. Values are in months and can be either 36 or 60.
- **int_rate** : Interest Rate on the loan.
- **installment** : The monthly payment owed by the borrower if the loan originates.
- **grade** : LoanTap assigned loan grade.
- **sub_grade** : LoanTap assigned loan subgrade.
- **emp_title** : The job title supplied by the Borrower when applying for the loan.
- **emp_length** : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- **home_ownership** : The home ownership status provided by the borrower during registration or obtained from the credit report.

- **annual_inc** : The self-reported annual income provided by the borrower during registration.
- **verification_status** : Indicates if income was verified by LoanTap, not verified, or if the income source was verified.
- **issue_d** : The month which the loan was funded.
- **loan_status** : Current status of the loan - Target Variable.
- **purpose** : A category provided by the borrower for the loan request.
- **title** : The loan title provided by the borrower.
- **dti** : A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income.
- **earliest_cr_line** : The month the borrower's earliest reported credit line was opened.
- **open_acc** : The number of open credit lines in the borrower's credit file.
- **pub_rec** : Number of derogatory public records.
- **revol_bal** : Total credit revolving balance.
- **revol_util** : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- **total_acc** : The total number of credit lines currently in the borrower's credit file.
- **initial_list_status** : The initial listing status of the loan. Possible values are – W, F.
- **application_type** : Indicates whether the loan is an individual application or a joint application with two co-borrowers.
- **mort_acc** : Number of mortgage accounts.
- **pub_rec_bankruptcies** : Number of public record bankruptcies.
- **Address** : Address of the individual.

2.2 Business Questions To Be Answered

2.2.1 Questions

- What characteristics indicate whether a borrower is likely to repay or default?
- How can lending institutions set lending terms and screening to maximize profitability while minimizing non-performing assets (NPAs)?
- What are the practical impacts of false positives (good loans rejected) and false negatives (bad loans approved)?
- What are the most significant predictors of default or repayment?
- How should different credit profiles inform interest rates and lending decisions?

2.2.2 Background and Significance

Why these questions matter:

In the consumer and MSME lending space, accurate risk assessment is crucial for maintaining a profitable, competitive, and compliant lending operation. Defaults (NPAs) not only harm profits but regulatory standing, while too-conservative lending restricts growth. Understanding drivers of good and bad loans enables smarter, more inclusive credit decisions and supports product innovation.

2.2.3 Problem Statement (Anonymized)

A digital lending platform wants to automate the underwriting of personal loans for individual borrowers. Given a set of customer and application attributes, the goal is two-fold:

1. Predict whether to approve or deny a credit line for an individual applicant.
2. Make evidence-based recommendations about repayment terms (such as interest rates or loan amounts) based on risk profiles.

2.2.4 Methodology and Real-World Applications

Methodology followed:

- **Exploratory Data Analysis (EDA):**
Structure, missing values, outlier detection, variable types, and basic relationships; conducted via summary statistics, correlation heatmaps, and plots.
- **Feature Engineering:**
Created flags for risk-related variables (bankruptcies, delinquency, mortgage accounts), binned/standardized text fields, and log-transformed extreme numeric values to reduce skew and improve model stability.
- **Data Preprocessing:**
Imputation of missing values, outlier treatment, encoding of categorical features, and normalization using MinMaxScaler. Removal of duplicate and highly collinear features to reduce overfitting.
- **Model Building and Evaluation:**
Developed a regularized logistic regression model (with class balance adjustments). Evaluated via confusion matrix, ROC/PR curves, and classification reports to assess precision, recall, F1, and AUC.
- **Tradeoff Analysis:**
Analyzed impact of raising/lowering thresholds on recall/precision, vital for balancing loan growth versus risk appetite.

Applications in real-life:

- **Loan Origination:**
Automates accept/reject and interest rate assignment for new loan applications.

- **Risk Management:**
Enables early intervention for risky segments, portfolio stress-testing, and compliance with regulatory guidelines on NPAs.
- **Customer Segmentation & Personalization:**
Drives targeted product offers and customized repayment plans for different risk strata.

2.2.5 Insights, Recommendations, and Implications

Key Insights and Recommendations:

- Most loans are repaid, but defaults are not rare and have significant financial/regulatory cost—raise recall for defaults even at moderate cost to precision.
- Job title, income, employment length, credit history (years), and public record flags are highly predictive—use these for risk-based pricing and enhanced due diligence.
- Moderate dependency on loan amount and installment, suggesting larger loans require stricter scrutiny.
- Missing or uncertain employment/title info is a risk factor—enforce additional documentation or conservative lending policies for such applicants.
- Treatment of class imbalance and regularization meaningfully improves model robustness; regularly recalibrate for data drift.

Implications for the Industry:

- **Credit Tech & Policy:**
A well-governed data-driven underwriting system enables fast, scalable, and more inclusive lending, supporting financial access for underbanked populations.
- **Product Development:**
Data insights support creation of flexible, differentiated loan products optimized for various customer segments (millennials, business owners).
- **Regulatory Compliance:**
Transparent, explainable models facilitate auditability and responsible lending in line with industry standards.

2.2.6 Addressing Limitations and Suggestions

- **Model Limitations:**
Logistic regression (linear) may miss non-linear or complex risk drivers. The model is less sensitive to rare/default cases, meaning some risk remains.
- **Data Limitations:**
Only internal and application data used—integrate external bureau or behavioral data for greater predictive power.
- **Threshold Calibration:**
The default decision threshold may not fully align with business goals; periodically adjust and test based on evolving strategy.

Alternative Explanations/Recommendations:

- Combine logistic regression with tree-based (e.g., XGBoost) or ensemble approaches to improve rare event (default) detection.
- Use cost-sensitive learning or profit-optimized thresholds to balance risk and return more precisely.

2.3 Analysis

2.3.1 Exploratory Data Analysis (EDA)

2.3.1.1 DataFrame Preview

```
3 dataset.head()
```

... Mounted at /content/drive

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...

5 rows × 27 columns

Figure 2.1: Data Frame Head Preview

2.3.1.2 Data Frame shape

```
1 dataset.shape
```

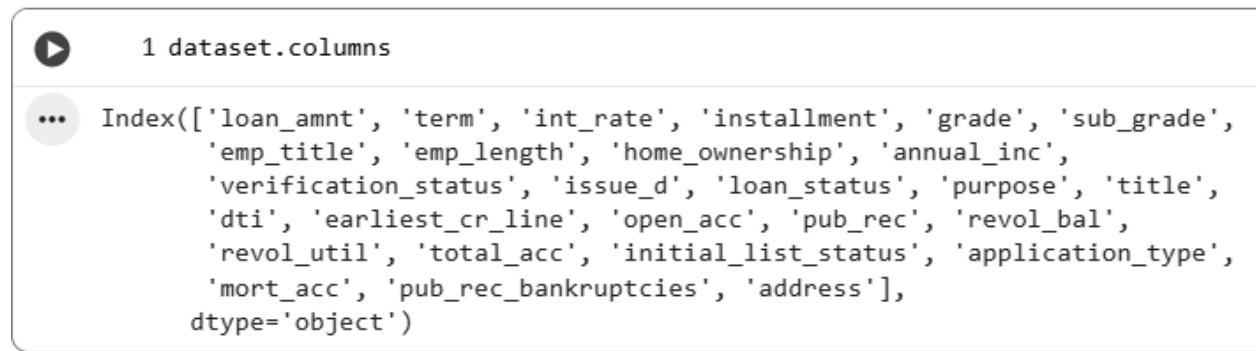
... (396030, 27)

Figure 2.2: Data Frame Shape

Insights:

The image shows the output of the .shape attribute for a pandas DataFrame, which reveals the structure of the dataset. There are 396,030 rows and 27 columns, indicating a large dataset with many variables for analysis. This size suggests the data is suitable for building robust machine learning models, detailed statistical examination, and in-depth creditworthiness evaluation. With 27 different features, the dataset supports comprehensive exploration of factors influencing loan decisions, allowing nuanced insights into borrower behavior and risk assessment.

2.3.1.3 Data Frame Columns



A screenshot of a Jupyter Notebook cell. The code cell contains the following Python code:

```
1 dataset.columns  
... Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',  
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',  
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',  
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',  
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',  
       'mort_acc', 'pub_rec_bankruptcies', 'address'],  
      dtype='object')
```

Figure 2.3: Data Frame Columns

Insights:

The displayed output shows the full list of columns present in your dataset. Each column represents a specific attribute of the loan, applicant, or credit history. Insights into this include:

- The dataset is rich with both numerical and categorical variables, including loan-specific fields (loan_amnt, term, int_rate, installment), creditworthiness indicators (grade, sub_grade, dti, open_acc, pub_rec, revol_bal, revol_util, total_acc), employment and demographic features (emp_title, emp_length, home_ownership, annual_inc), and fields related to verification and application type.
- It includes the target variable (loan_status), which will be essential for any modeling or classification tasks focused on predicting loan default or approval outcomes.
- With features like purpose, title, and address, deeper segmentation and geospatial or use-case-specific analyses are possible.
- The presence of historical fields like earliest_cr_line and negative indicators (pub_rec_bankruptcies, pub_rec) allows for comprehensive risk assessment modeling.
- This suite of features collectively enables nuanced exploration of which applicant characteristics, behaviors, and histories most strongly correlate with loan performance, which can drive more accurate underwriting and business insights.

2.3.1.4 Detection of Null Values

```
1 df_null_values = np.round((dataset.isna().sum()/len(dataset) * 100), 2)
2 df_null_values = df_null_values.reset_index().rename(columns={'index':'column_name', 0:'percentage'})
3 df_null_values
```

...	column_name	percentage
0	loan_amnt	0.00
1	term	0.00
2	int_rate	0.00
3	installment	0.00
4	grade	0.00
5	sub_grade	0.00
6	emp_title	5.79
7	emp_length	4.62
8	home_ownership	0.00
9	annual_inc	0.00
10	verification_status	0.00
11	issue_d	0.00
12	loan_status	0.00
13	purpose	0.00
14	title	0.44
15	dti	0.00
16	earliest_cr_line	0.00
17	open_acc	0.00
18	pub_rec	0.00
19	revol_bal	0.00
20	revol_util	0.07
21	total_acc	0.00
22	initial_list_status	0.00
23	application_type	0.00
24	mort_acc	9.54
25	pub_rec_bankruptcies	0.14
26	address	0.00

Figure 2.4: Detection of Null Values

Insights:

The presented table displays the percentage of missing (null) values in each column of your dataset. Here are the key insights:

Columns with No Missing Data

Most columns, including critical fields such as loan_amnt, term, int_rate, installment, grade, emp_length, annual_inc, loan_status, and credit parameters, have 0% missing values. This means your core variables are complete, ensuring robust statistical analysis and model training.

Columns with Moderate Missing Data

- emp_title has about 5.8% missing values, likely due to some borrowers not providing job titles.
- emp_length is missing for 4.6% of entries, possibly reflecting applicants unable or unwilling to specify employment duration.
- mort_acc (number of mortgage accounts) is missing in 9.54% of cases, which could hinder analyses involving mortgage credit history.

Columns with Minimal Missing Data

- A few columns, such as title (0.44%), revol_util (0.07%), and pub_rec_bankruptcies (0.14%), have very low missingness and are unlikely to impact broader analysis.

Columns Fully Present

- Key demographic and financial columns including home_ownership, verification_status, purpose, dti, application_type, address, and several credit variables, are 100% complete.

Impact for Analysis

- The dataset overall is well populated, with occasional moderate missingness in job and mortgage-related fields.
- Dropping or imputing missing values for a small fraction of borrowers will enable you to retain most of your analytical power when modeling loan default or approval.

2.3.1.5 Duplicate values

```
1 dataset.duplicated().sum()  
... 0
```

Figure 2.5: Duplicate values

Insights

The output shows that there are zero duplicate rows in your dataset. This means every entry is unique, which ensures the integrity and reliability of your analysis and modeling outcomes. You

can proceed confidently without concerns of redundant or repeated records skewing your insights or results.

2.3.1.6 Descriptive Statistics of Numerical Dataset

	1 dataset.describe()												
	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies	
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	3.960300e+05	395754.000000	396030.000000	358235.000000		395495.000000	
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	0.178191	1.584454e+04	53.791749	25.414744	1.813991	0.121648	
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	0.530671	2.059184e+04	24.452193	11.886991	2.147930	0.356174	
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	0.000000	2.000000	0.000000	0.000000	
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	0.000000	6.025000e+03	35.800000	17.000000	0.000000	0.000000	
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	0.000000	1.118100e+04	54.800000	24.000000	1.000000	0.000000	
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	0.000000	1.962000e+04	72.900000	32.000000	3.000000	0.000000	
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86.000000	1.743266e+06	892.300000	151.000000	34.000000	8.000000	

Figure 2.6: Descriptive Statistics of Numerical Dataset

Insights

Central Tendencies and Spread

- The mean loan amount is approximately 14,114, with a wide range from 500 to 40,000, showing diverse borrowing needs.
- The average interest rate is around 13.64%, with values spanning from 5.29% to 30.99%, indicating varying applicant risk profiles.
- Annual income averages about 74,201, but the standard deviation is high, reflecting significant income disparities.

Quartiles and Outliers

- For most features, the minimum and maximum values are far apart, which implies the presence of outliers requiring closer inspection, especially for annual income, revolving balance, and loan amount.
- The 50th percentile (median) for loan amount is 12,000; for interest rate, it's 13.33%; and for monthly installment, it's 375.43.

Credit Health Indicators

- The average debt-to-income ratio (DTI) is 17.38, which suggests moderate overall indebtedness, but individual cases go up to 9999.00. This is possibly an outlier or data entry issue.
- On average, borrowers have about 11 open accounts, and the median is slightly lower, at 10.
- The majority of applicants have few derogatory public records, bankruptcies, or mortgage accounts, but a small number have values as high as 86 (for public records) or 34 (for mortgage accounts).

Utilization and Balances

- The mean revolving balance is approximately 15,845, while revolving utilization rate averages 53.79, suggesting many borrowers are using over half their available revolving credit.
- The distribution of these variables indicates both low-risk and high-risk borrowers are present.

Data Quality

- Columns have high non-null counts, meaning the statistics are robust and reflect the full data scope.

2.3.1.7 Descriptive Statistics of Categorical Dataset

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue_d	loan_status	purpose	title	earliest_cr_line	initial_list_status	application_type	address
count	396030	396030	396030	373103	377729	396030	396030	396030	396030	394274	396030	396030	396030	396030	
unique	2	7	35	173105	11	6	3	115	2	14	48816	684	2	3	393700
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified	Oct-2014	Fully Paid	debt_consolidation	Debt consolidation	Oct-2000	f	INDIVIDUAL	USCGC Smith/inFPO AE 70466
freq	302006	116018	26655	4389	126041	198348	139563	14846	318357	234507	152472	3017	238066	395319	8

Figure 2.7: Descriptive Statistics of Categorical Dataset

Insights

Most Frequent Values and Diversity

- The majority of loans have a term of 36 months, making it the most common repayment period.
- The dominant assigned loan grade is ‘B’, and the most frequent sub-grade is ‘B3’, reflecting risk distribution and possibly underwriting preferences.
- For employment, the most common job title among borrowers is “Teacher,” and the majority have “10+ years” of employment, indicating a trend towards experienced applicants with educational backgrounds or job security.
- “MORTGAGE” is the most common home ownership status, suggesting many borrowers either own or are paying off their homes.

Loan and Application Characteristics

- The most common loan status at the time of data extraction is “Fully Paid,” and the leading purpose for borrowing is “debt_consolidation,” indicating a high portion of responsible borrowers focused on managing or consolidating existing debts.
- The “Verified” status is the most common for verification, signifying strong income/data verification practices.
- “INDIVIDUAL” is the most frequent application type, with joint applications being relatively rare.
- The dataset contains a vast variety of job titles (173,105 unique values) and addresses (393,970 unique entries), reflecting high diversity and user privacy.

Other Categorical Trends

- “Oct-2000” is the most common earliest credit line, which may reflect both genuine borrowing patterns and some data coding or reporting standards.
- “F” dominates as the initial list status, which may refer to the loan's listing stage.

Overall, these insights show strong diversity in borrower background and application details, but clear trends toward certain loan products, verification types, and applicant profiles.

2.3.2 Univariate Analysis

2.3.2.1 Loan Term Proportions

```
1 df_term = dataset['term'].value_counts()  
2 plt.pie(df_term, labels = df_term.index, autopct='%.2f%%')  
3 plt.title('Loan Term Proportions')  
4 plt.show()  
5 df_term
```

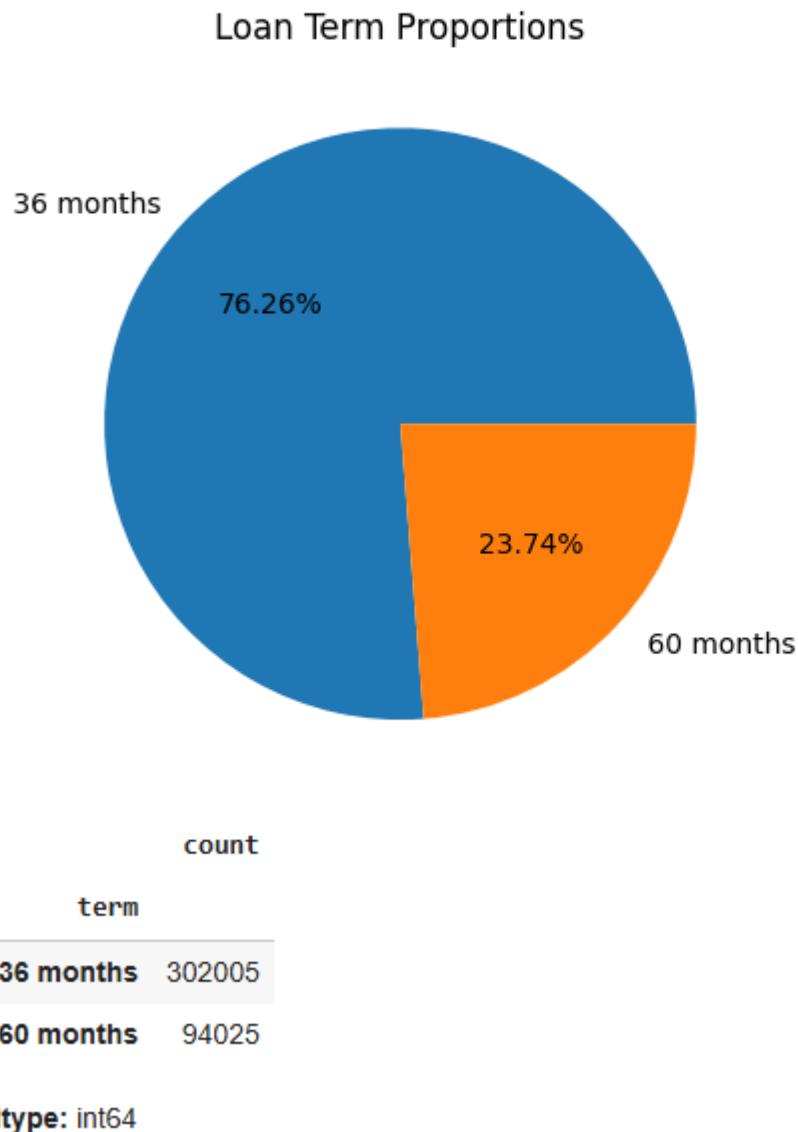


Figure 2.8: Loan Term Proportions

Insights

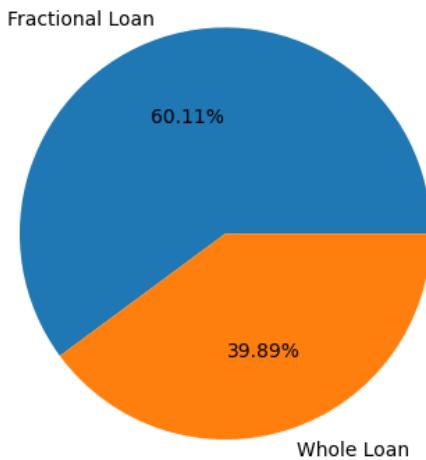
The pie chart and summary table illustrate the distribution of loan terms within the dataset. About 76.26% of loans have a term of 36 months, while 23.74% have a term of 60

months. This significant preference for shorter-term loans indicates that most borrowers opt for quicker repayment cycles, possibly to reduce interest costs or because of lending policy emphasis on lower-risk, shorter-duration lending. The provided counts (302,005 for 36 months and 94,025 for 60 months) reinforce this pattern, highlighting that three-year loans are more than three times as common as five-year loans in this portfolio.

2.3.2.2 Initial List Status Proportions

```
1 df_init_lst_status = dataset['initial_list_status'].value_counts()  
2 df_init_lst_status  
3  
4 plt.pie(df_init_lst_status, labels = ['Fractional Loan', 'Whole Loan'], autopct='%.2f%')  
5 plt.title('Initial List Status Proportions')  
6 plt.show()  
7 df_init_lst_status
```

*** Initial List Status Proportions



```
count  
initial_list_status  
f 238066  
w 157964  
dtype: int64
```

Figure 2.9: Initial List Status Proportions

Insights

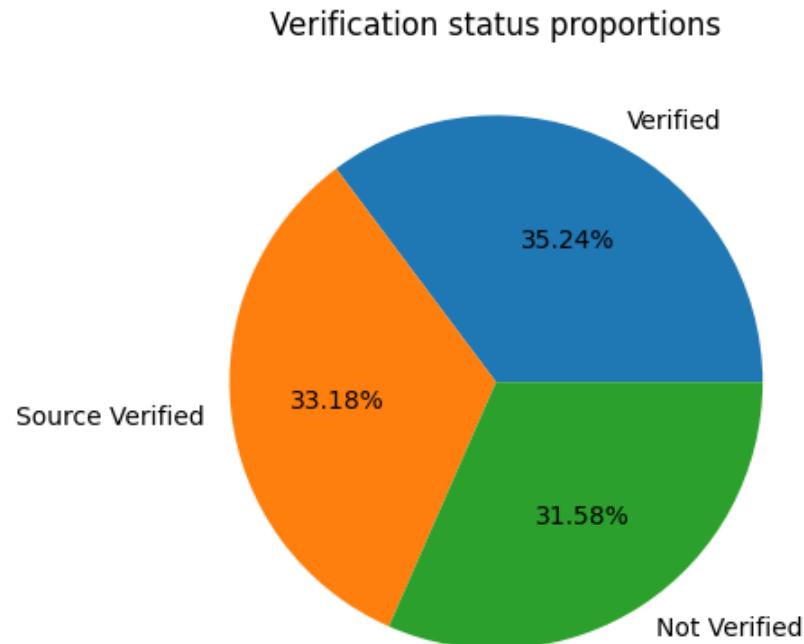
The pie chart and summary table present the proportions of initial listing status among loans in the dataset. About 60.11% of loans are classified as “Fractional Loan” (f), while 39.89% are “Whole Loan” (w). This indicates that the platform predominantly lists loans as fractional, allowing investors to participate in portions of loans rather than funding them entirely. The count shows 238,066 fractional loans compared to 157,964 whole loans, highlighting a business strategy or market preference for fractionalized investment opportunities, which can support better risk diversification and broader investor participation.

2.3.2.3 Verification Status Proportions

```

1 df_verify_status = dataset['verification_status'].value_counts()
2 df_verify_status
3
4 plt.pie(df_verify_status, labels = df_verify_status.index, autopct='%.2f%%')
5 plt.title('Verification status proportions')
6 plt.show()
7 df_verify_status

```



	count
verification_status	
Verified	139563
Source Verified	131385
Not Verified	125082

dtype: int64

Figure 2.10: Verification Status Proportions

Insights

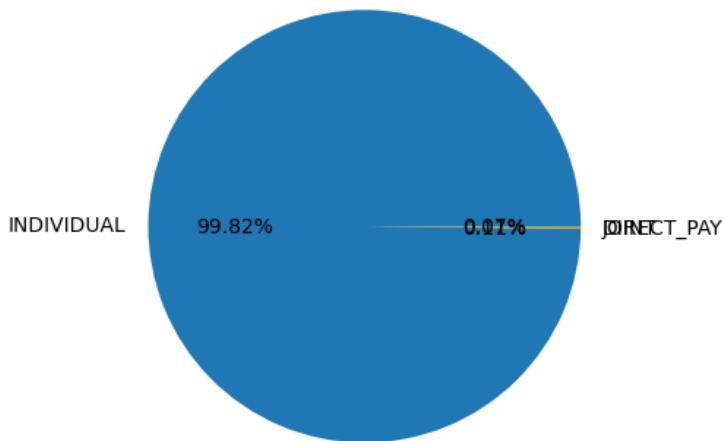
The chart and supporting data show the proportions of loan applications by verification status. Approximately 35.24% of loans are fully “Verified,” 33.18% are “Source Verified,” and 31.58% are “Not Verified.” This indicates a strong emphasis on validation, with the majority of

borrowers undergoing some form of income or information check. However, about one-third of loans have not been verified, which may represent lower documentation channels or alternative risk models. Overall, the portfolio demonstrates a balanced mix of verification types, reflecting flexible underwriting while maintaining oversight for risk control.

2.3.2.4 Application Type Proportions

```
1 #application_type
2 df_apply_type= dataset['application_type'].value_counts()
3 plt.pie(df_apply_type, labels = df_apply_type.index, autopct='%.2f%%')
4 plt.title('Application Type proportions')
5 plt.show()
6 df_apply_type = df_apply_type.reset_index()
7 df_apply_type['Percent'] = df_apply_type['count'].astype('int')/sum(df_apply_type['count'])*100
8 df_apply_type
```

Application Type proportions



	application_type	count	Percent
0	INDIVIDUAL	395319	99.820468
1	JOINT	425	0.107315
2	DIRECT_PAY	286	0.072217

Figure 2.11: Application Type Proportions

Insights

The chart and accompanying data reveal a dramatic dominance of individual applications in the loan dataset. Nearly all applications (99.82%) were submitted by individuals, with joint and DIRECT_PAY applications representing only 0.11% and 0.07% of the total, respectively. This indicates the platform's user base overwhelmingly consists of single applicants, suggesting the

product design, risk assessment, and marketing strategies are likely tailored primarily to individual borrowers rather than co-applicants or special direct payment arrangements.

2.3.2.5 Count of Different Grades

```
1 grades = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G'], dtype='str').reshape(7,1)
2 values = np.array([1,2,3,4,5,6,7], dtype='str').reshape(1,7)
3 subGrades = np.char.add(grades, values)
4 plt.figure(figsize=(15,20))
5 plt.subplot(4, 2, 1)
6 plt.suptitle("Different Grades and it's data count")
7 plt.title("All the Grades the A-F")
8 sns.countplot(data=dataset, x='grade')
9 plt.subplot(4, 2, 2)
10 plt.title("All the sub grades of A")
11 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[0,:].flatten())], x='sub_grade')
12 plt.subplot(4, 2, 3)
13 plt.title("All the sub grades of B")
14 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[1,:].flatten())], x='sub_grade')
15 plt.subplot(4, 2, 4)
16 plt.title("All the sub grades of C")
17 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[2,:].flatten())], x='sub_grade')
18 plt.subplot(4, 2, 5)
19 plt.title("All the sub grades of D")
20 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[3,:].flatten())], x='sub_grade')
21 plt.subplot(4, 2, 6)
22 plt.title("All the sub grades of E")
23 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[4,:].flatten())], x='sub_grade')
24 plt.subplot(4, 2, 7)
25 plt.title("All the sub grades of F")
26 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[5,:].flatten())], x='sub_grade')
27 plt.subplot(4, 2, 8)
28 plt.title("All the sub grades of G")
29 sns.countplot(data=dataset[dataset['sub_grade'].isin(subGrades[6,:].flatten())], x='sub_grade')
30 plt.show()
```

Different Grades and it's data count

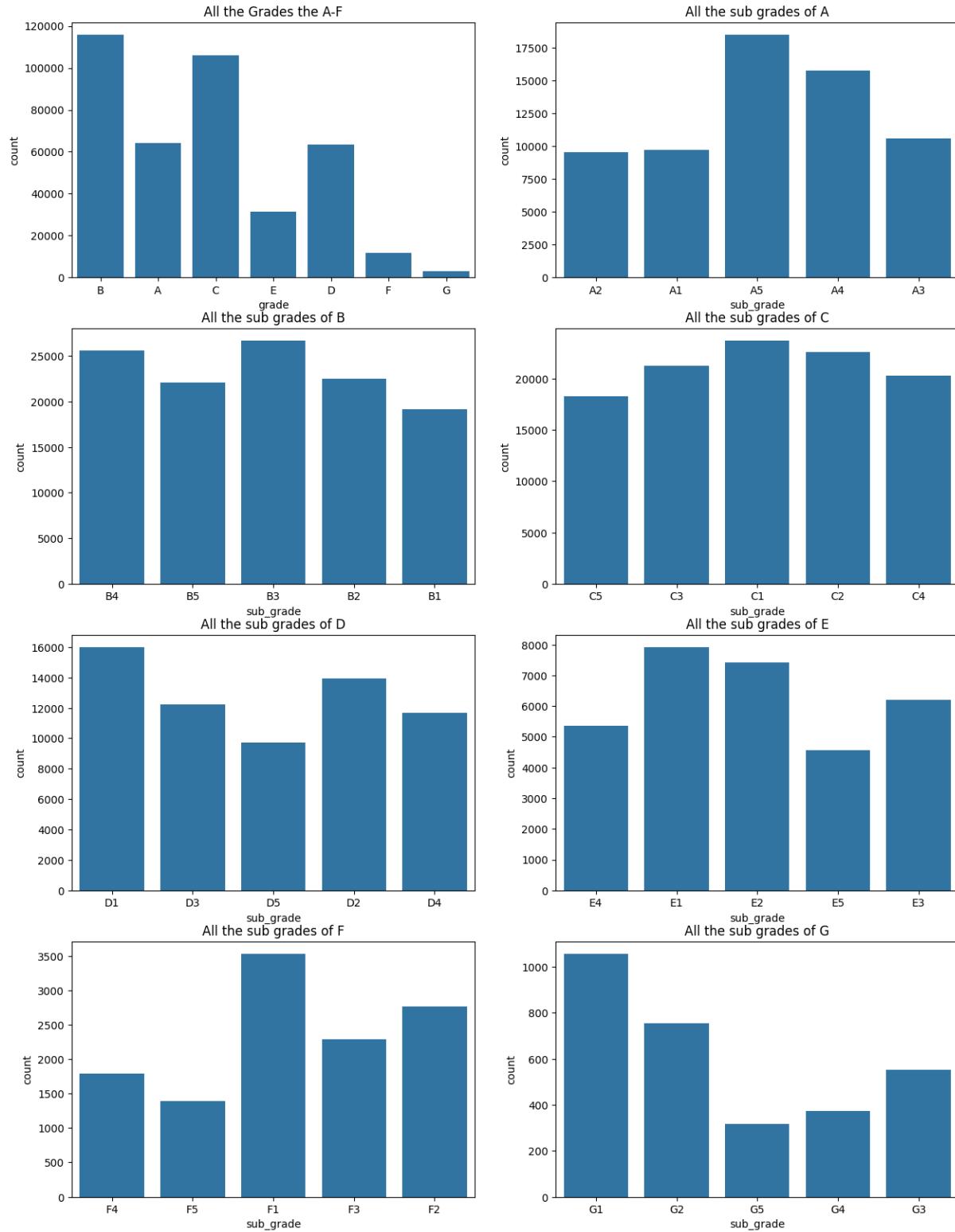


Figure 2.12: Count of Different Grades

Insights

Grade Distribution

- Grade **B** has the highest count by a wide margin, followed by **C** and **A**, with grades **E**, **D**, **F**, and **G** appearing far less frequently.
- This suggests most borrowers fall into mid-risk tiers, with fewer applicants classified as high-risk (F/G) or very low-risk (A).

Sub-grade Patterns

- Within each grade, sub-grades show meaningful variability.
- The distribution within **A**, **B**, and **C** is relatively balanced, but some sub-grades (like B3 and C1) have slightly higher counts.
- For grades with fewer loans (E, F, G), sub-grade counts drop further, pointing to a narrow distribution of borrower risk at those extremes.

Implications for Risk and Underwriting

- The platform's portfolio leans heavily toward moderate-credit-risk applicants, as seen in the prevalence of grades B and C.
- Loan allocation and underwriting seem to favor borrowers in these risk bands, possibly reflecting business strategy or risk appetite.

Visual Data Quality

- The separation of each grade's sub-grade distribution reveals underlying granularity in borrower assessment, validating the multi-level grading system as relevant to overall data coverage.

2.3.2.6 Employee experience and their loan application count

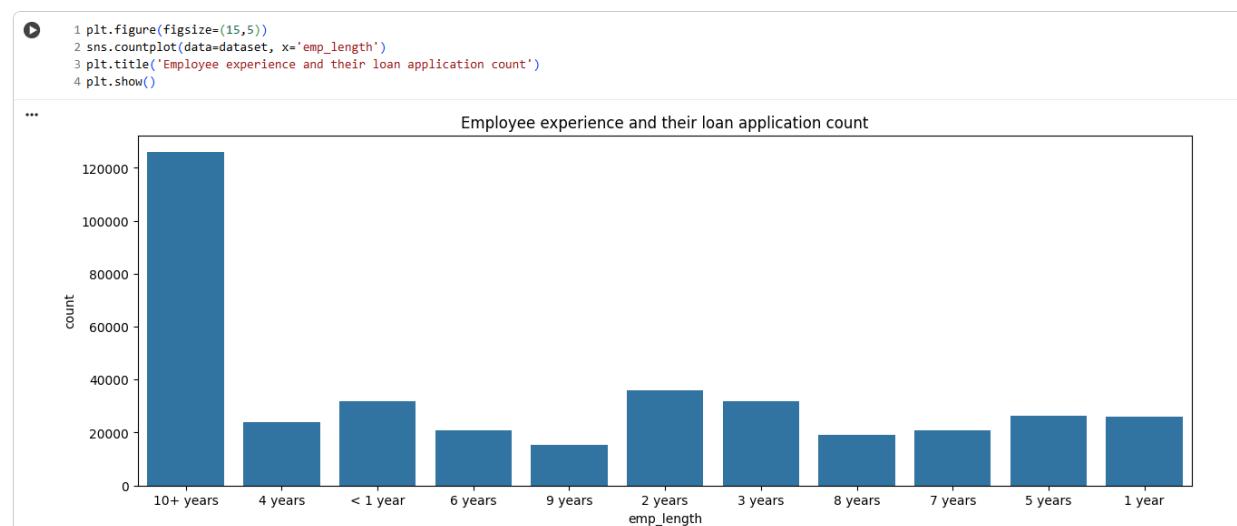


Figure 2.13: Employee Experience And Their Loan Application Count

Insights

The visualization shows a strong preference for loan applications from individuals with “10+ years” of employment experience. This category far surpasses all other experience brackets in application count. Applicants with less than one year of experience and those with 1–9 years appear in much smaller, relatively similar numbers across categories. This trend suggests that the lending platform attracts and potentially favors more experienced, stable applicants, which may reflect underwriting criteria designed to reduce default risk by prioritizing longer work histories.

2.3.2.7 Home Ownership and Respective Loan application count

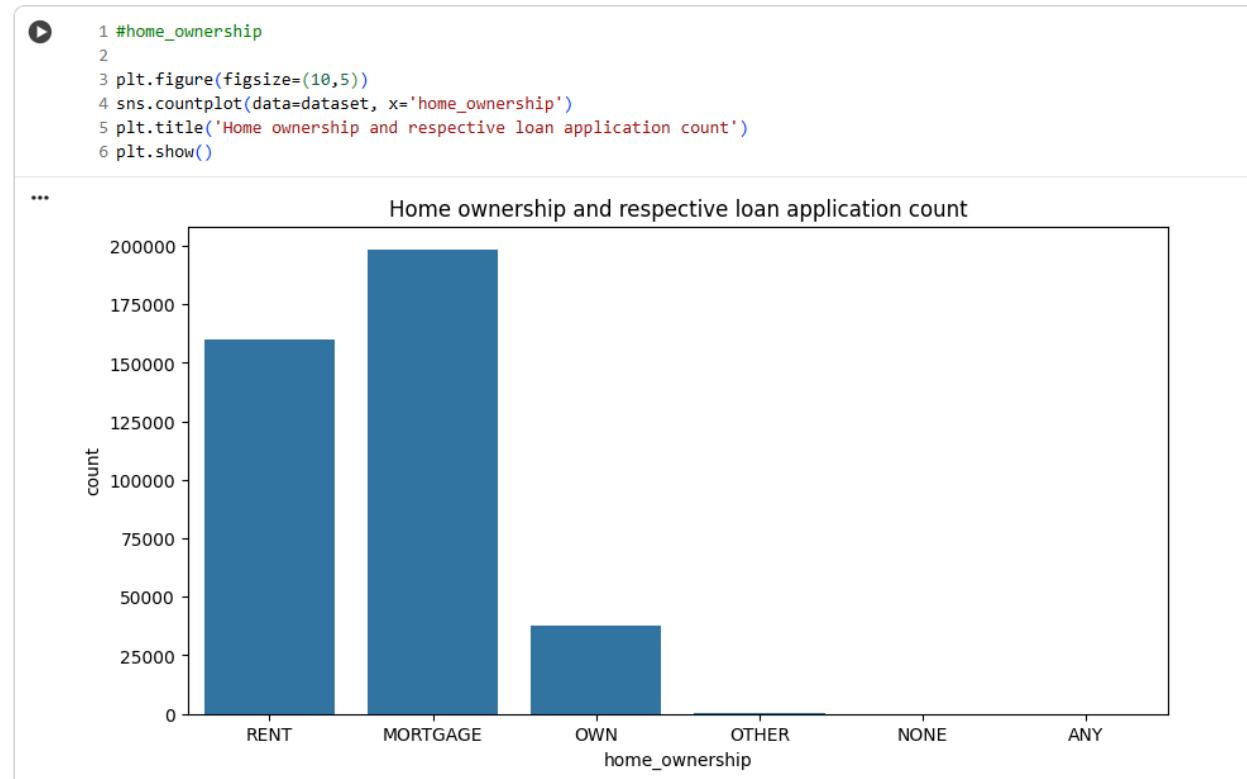


Figure 2.14: Home Ownership and Respective Loan application count

Insights

The bar chart displays loan applications by home ownership status. The majority of applicants either have a mortgage or rent, with mortgage holders making up the largest group, followed by renters. Borrowers who fully own their homes form a much smaller proportion, and

categories like “OTHER,” “NONE,” and “ANY” are almost negligible. This pattern suggests the platform primarily serves individuals with ongoing housing expenses, either through rent or mortgage, rather than those who own their homes outright. It may also reflect greater demand for loans among those with regular housing payments, or risk-management preferences in underwriting.

2.3.2.8 Loan Verification Status and Respective Application Count



```
1 #verification_status  
2  
3 plt.figure(figsize=(5,3))  
4 sns.countplot(data=dataset, x='verification_status')  
5 plt.title('Verification status and respective loan application count')  
6 plt.show()
```

...

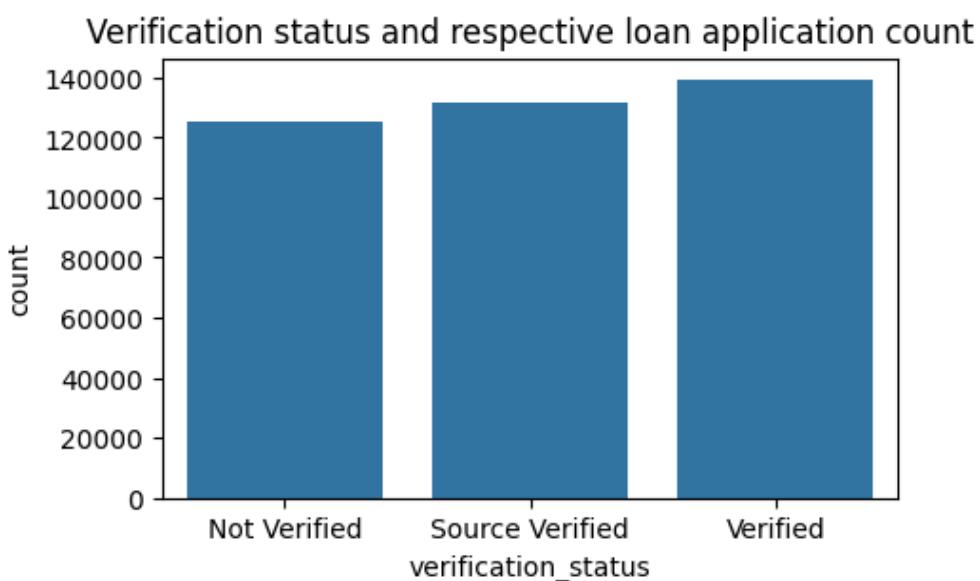


Figure 2.15: Loan Verification Status and Respective Application Count

Insights

The bar chart illustrates the distribution of loan applications by verification status. The counts across “Verified,” “Source Verified,” and “Not Verified” categories are fairly similar, with “Verified” loans holding a slight lead. This reflects a balanced approach in the platform’s underwriting process—while full verification is common, a significant number of loans proceed with either source verification or no verification, suggesting flexibility in risk assessment and

inclusivity for borrowers with varying documentation levels. The nearly equal heights of the bars imply that no single verification status overwhelmingly dominates the portfolio.

2.3.2.9 Loan Application Purpose and It's Count

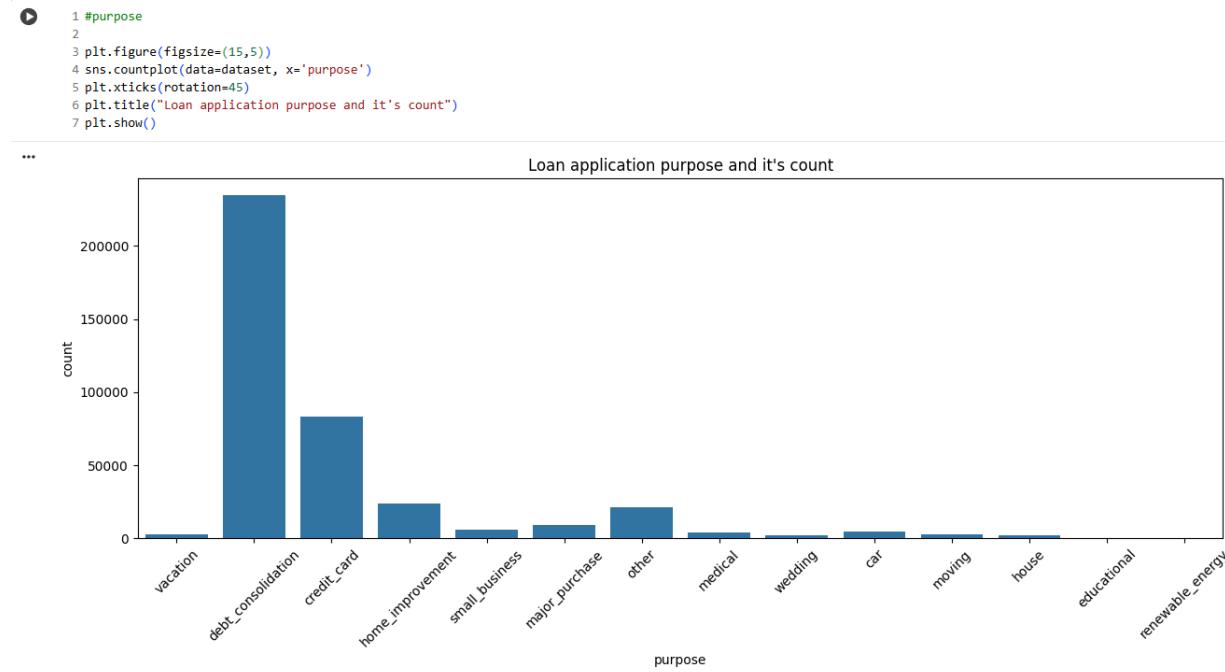


Figure 2.16: Loan Application Purpose and It's Count

Insights

- The bar chart displays the distribution of loan purposes among applicants. The largest category by far is “debt_consolidation,” indicating that most borrowers use these loans to merge and manage existing debts. The next most common purposes are “credit_card” (covering credit card balances) and “home_improvement.” Other categories like “small_business,” “major_purchase,” and “medical” have noticeably fewer applications, and the remaining purposes (such as “vacation,” “wedding,” “moving,” “renewable_energy”) are rare.
- This concentration suggests that loan products are especially attractive for debt and credit management, while discretionary or specialized use-cases make up a small share of the portfolio. The platform’s risk strategy, borrower marketing, and underwriting policies are likely shaped by this dominant demand for consolidation and credit product refinancing.

2.3.2.10 Distribution of Loan Amounts using Kernel Density Estimate (KDE)

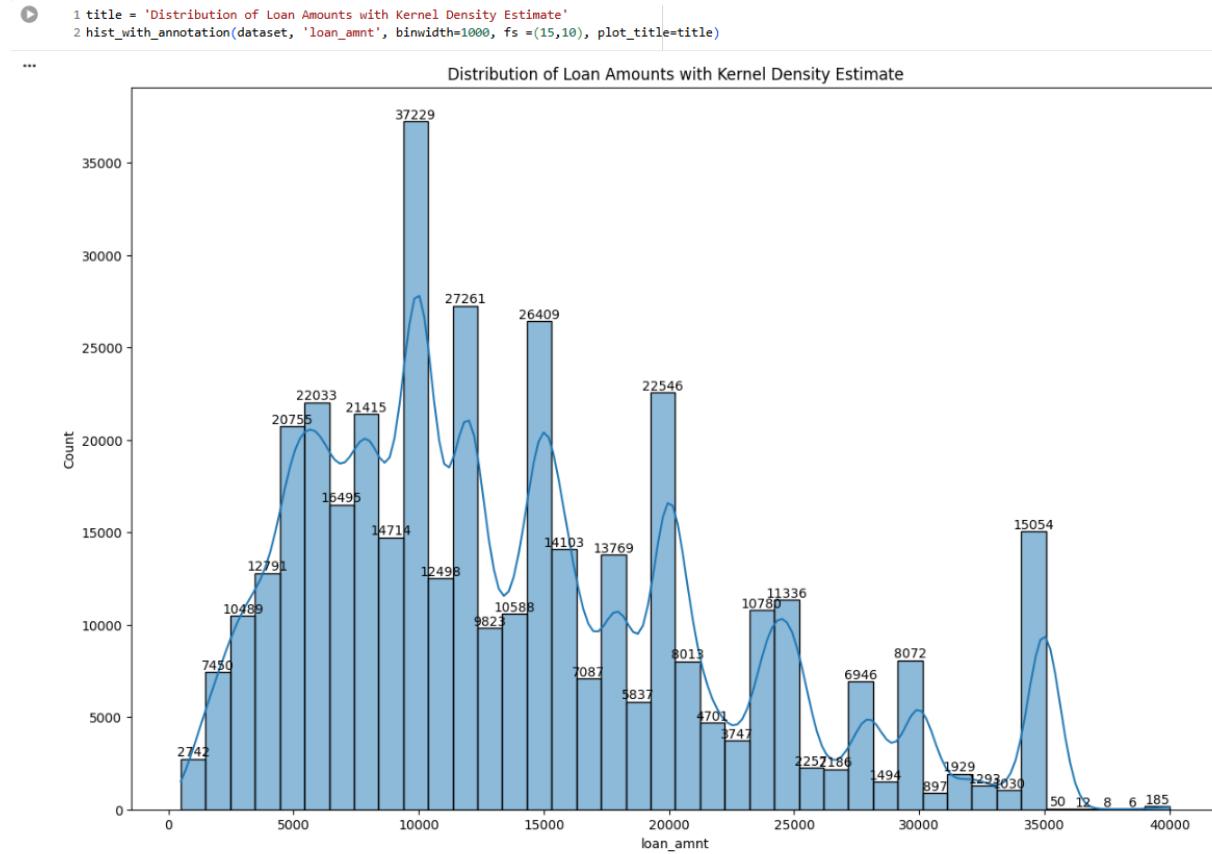


Figure 2.17: Distribution of Loan Amounts using Kernel Density Estimate (KDE)

Insights:

- The histogram and kernel density estimate illustrate the distribution of loan amounts in the dataset. Most loans cluster between ₹5,000 and ₹20,000, with a prominent peak around ₹10,000 indicating this is the most common amount borrowed. The data is right-skewed, with smaller counts at higher loan values (above ₹20,000). Repeated smaller peaks may reflect preferred loan increments or product offerings.
- The presence of spikes at round numbers (like ₹10,000, ₹20,000, ₹35,000) suggests standardization or borrower preferences for particular amounts. There are very few extremely small or large loans, reinforcing that the platform caters primarily to moderate-sized borrowing needs.

- This distribution helps profile typical borrower demand and can guide product structuring, risk modeling, and marketing strategies toward the most popular loan sizes.

2.3.2.11 Distribution of Interest Rates using Kernel Density Estimate (KDE)

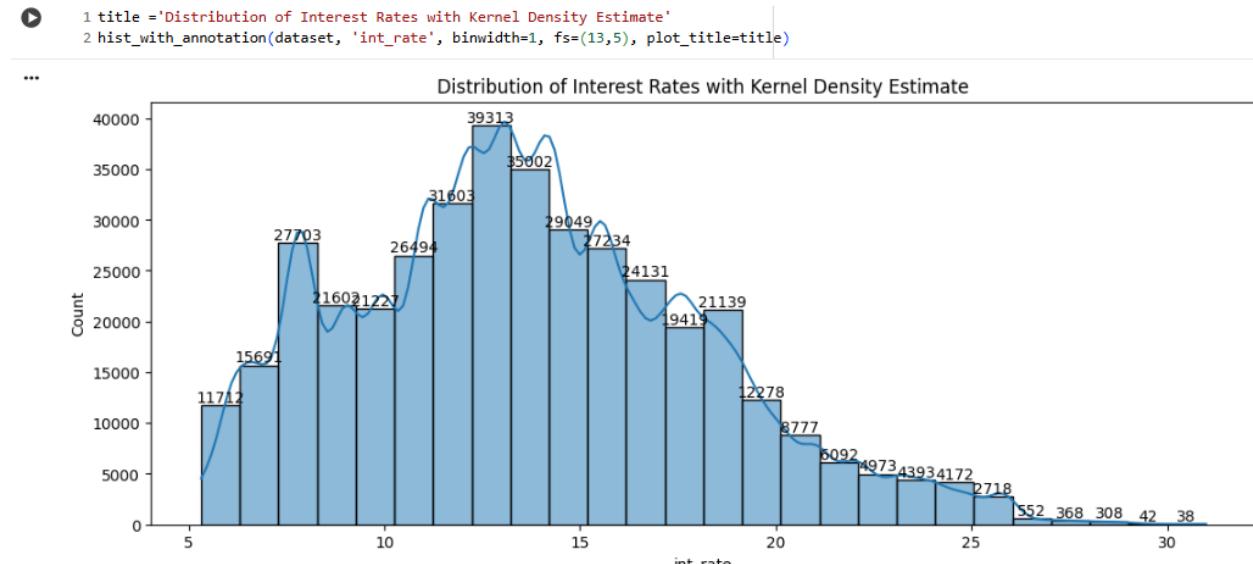


Figure 2.18: Distribution of Interest Rates using Kernel Density Estimate (KDE)

Insights

- The histogram and kernel density estimate plot reveal the distribution of interest rates for loans in the dataset. Most loans cluster in the range of 10% to 18%, with a pronounced peak near 13%–15%. The distribution is right-skewed: as interest rates increase above 18%, the count of loans falls progressively, with very few loans at extremely high rates (above 25%).
- Loan products at lower interest rates are more prevalent, potentially reflecting the platform's target market of relatively qualified borrowers or its risk policies. The gradual decrease in the number of high-rate loans suggests either tighter approval criteria or less borrower demand for loans at those rates.
- Distinct peaks at specific interest rates could indicate standard product pricing or frequent scoring bands used in underwriting. Overall, the majority of borrowers secure loans at moderate rates, suggesting risk management aligned with mainstream lending practices.

2.3.2.12 Loan Installment Amount using Kernel Density Estimate (KDE)

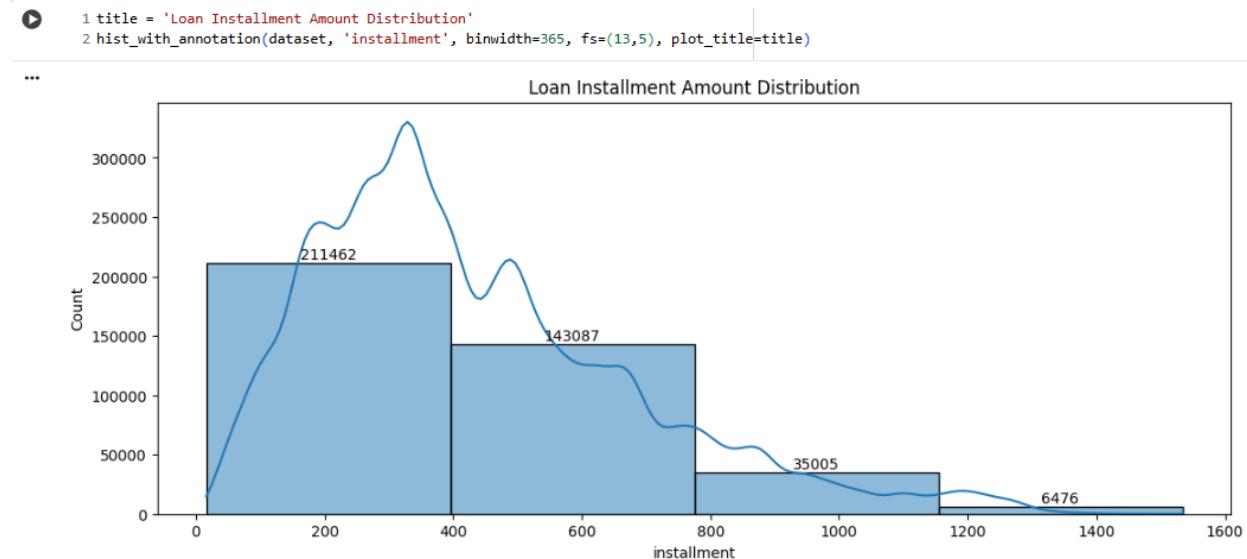


Figure 2.19: Loan Installment Amount using Kernel Density Estimate (KDE)

Insights

- The histogram and density curve show that the majority of loan installments fall below ₹400, with the largest number of borrowers paying monthly installments in the ₹0–₹400 range. The counts decline steadily as installment amounts increase, with very few loans having installments above ₹1,200. This right-skewed distribution suggests that most borrowers prefer or qualify for loans with lower repayment burdens, reflecting either borrower affordability or the platform's risk and eligibility filtering.
- Distinct peaks at certain values may correspond to typical loan terms and amounts. The spread indicates the platform serves a diverse set of borrowers, but the focus is strongly on manageable monthly repayments, supporting broad accessibility while controlling the likelihood of repayment

2.3.2.13 Annual Income using Kernel Density Estimate (KDE)

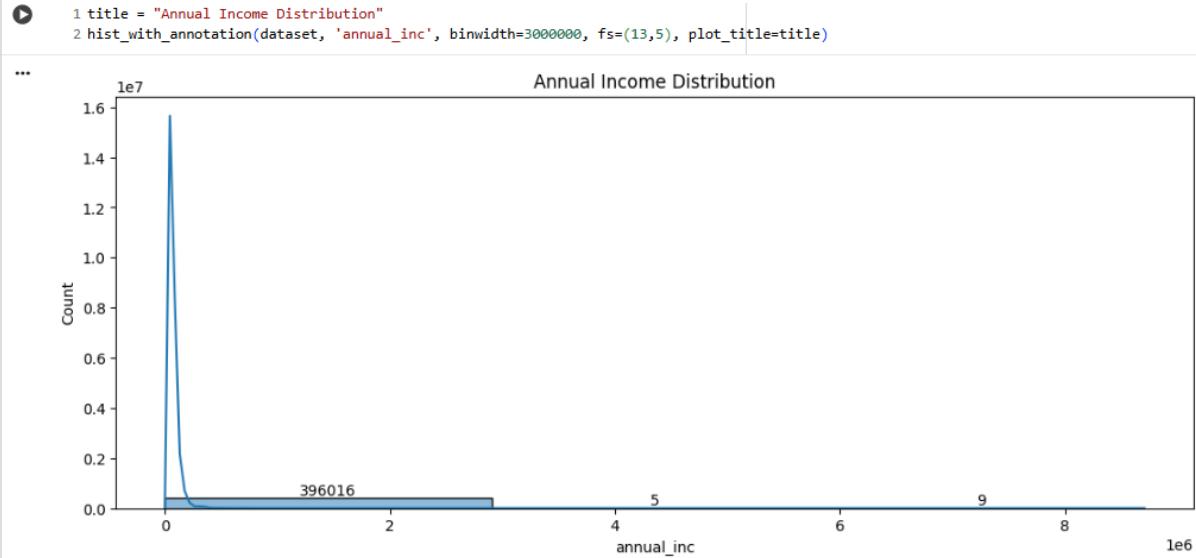


Figure 2.20: Annual Income using Kernel Density Estimate (KDE)

Insights

- The histogram and density plot illustrate that most loan applicants report annual incomes clustered at lower ranges, with the vast majority below ₹2,000,000. The curve is sharply right-skewed, with very few borrowers earning extremely high incomes. The steep drop-off after the first income bin suggests a typical applicant profile is from the lower to mid-income brackets.
- The presence of isolated points far to the right may indicate outliers or rare high earners. This profile suggests the platform predominantly serves mainstream consumers and salaried professionals rather than affluent or high-net-worth individuals, helping target risk models and lending policies toward broader, moderate-income market segments

2.3.2.14 Debt to Income using Kernel Density Estimate (KDE)

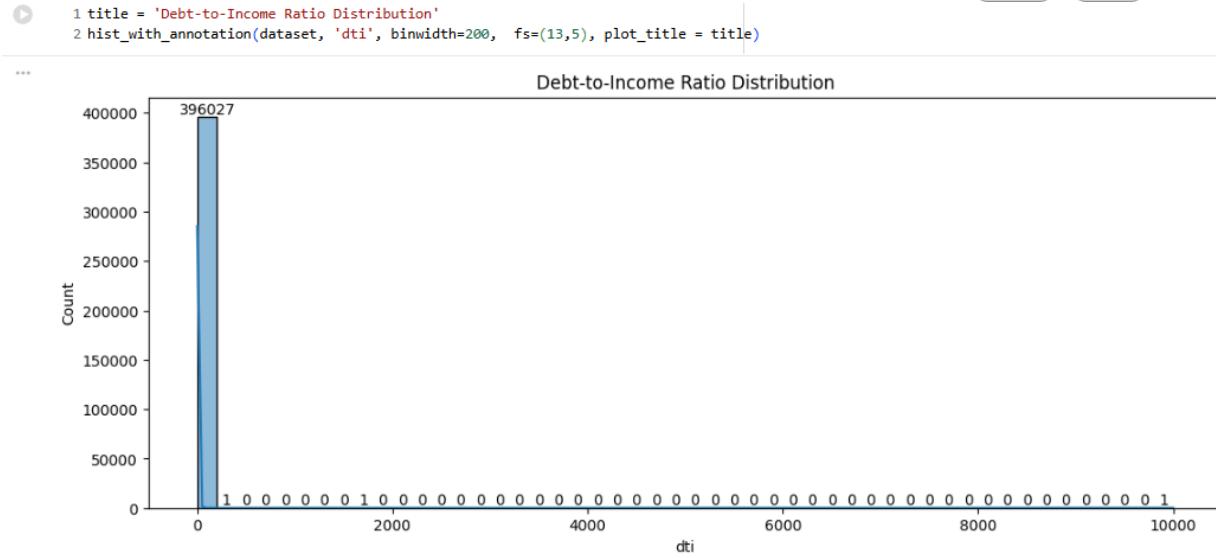


Figure 2.21: Debt to Income using Kernel Density Estimate (KDE)

Insights

- The histogram of debt-to-income ratio (DTI) displays an extreme right-skew, with almost all applicants having DTI well below 200. Virtually the entire dataset is concentrated in the lowest ratio range, while very few (or possibly erroneous) records show DTI values in the thousands. This means most borrowers maintain a reasonable level of debt compared to income, which aligns with prudent lending standards and reduced default risk.
- The few outlying high DTI values are likely data errors or exceptional edge cases and should be reviewed carefully. The overall DTI profile supports the platform's focus on financially stable, manageable-risk applicants.

2.3.2.15 Open Accounts using Kernel Density Estimate (KDE)

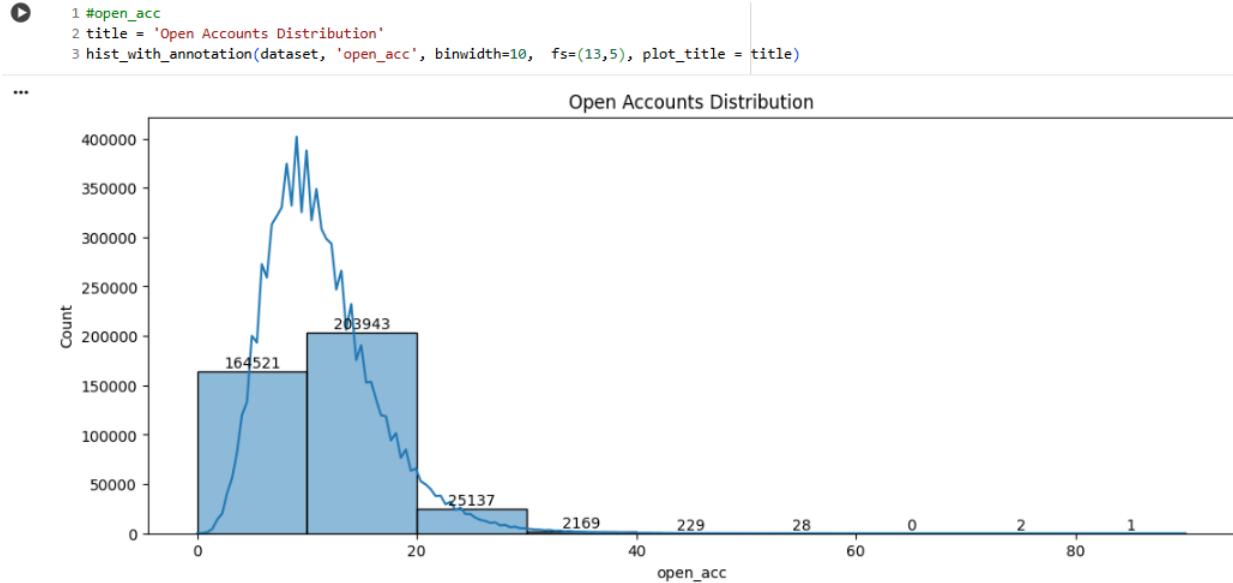


Figure 2.22: Open Accounts using Kernel Density Estimate (KDE)

Insights

- The bar plot and kernel density curve show that the number of open credit accounts for applicants typically ranges between 0 and 20, with the majority having between 5 and 15 open accounts. The distribution is right-skewed, with only a small fraction of applicants holding larger numbers of open accounts (above 30).
- This profile indicates that most borrowers actively use credit but are not overly leveraged, which is a positive indicator for creditworthiness. Outliers with very high numbers of open accounts are extremely rare and may represent different borrower segments or special risk cases. This overall pattern supports healthy financial activity among the platform's users.

2.3.2.16 Public Records using Kernel Density Estimate (KDE)

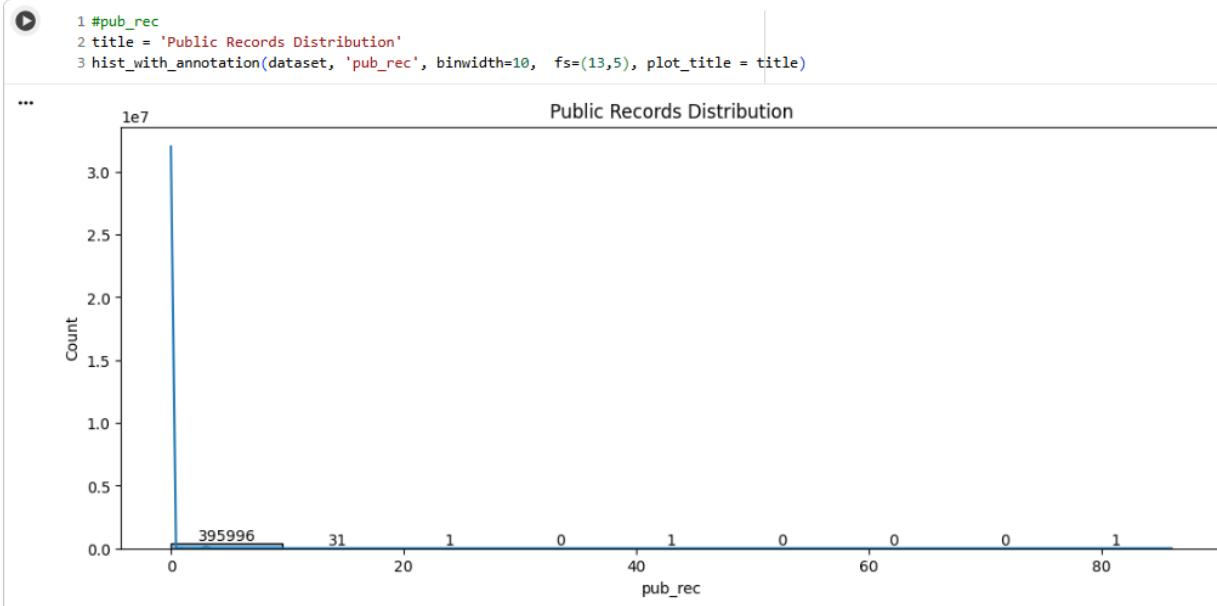


Figure 2.23: Public Records using Kernel Density Estimate (KDE)

Insights

The histogram for public records shows almost all applicants have zero derogatory public records. Only a minuscule number of individuals have one or more public records, and those with extremely high counts are practically nonexistent. This indicates that the borrower pool is largely free from legal or financial judgments, reinforcing the platform's focus on lower-risk applicants. The few outliers likely represent rare exceptions or data anomalies. Overall, the public record profile strongly supports stable, responsible lending risk.

2.3.2.17 Revolving Balance using Kernel Density Estimate (KDE)

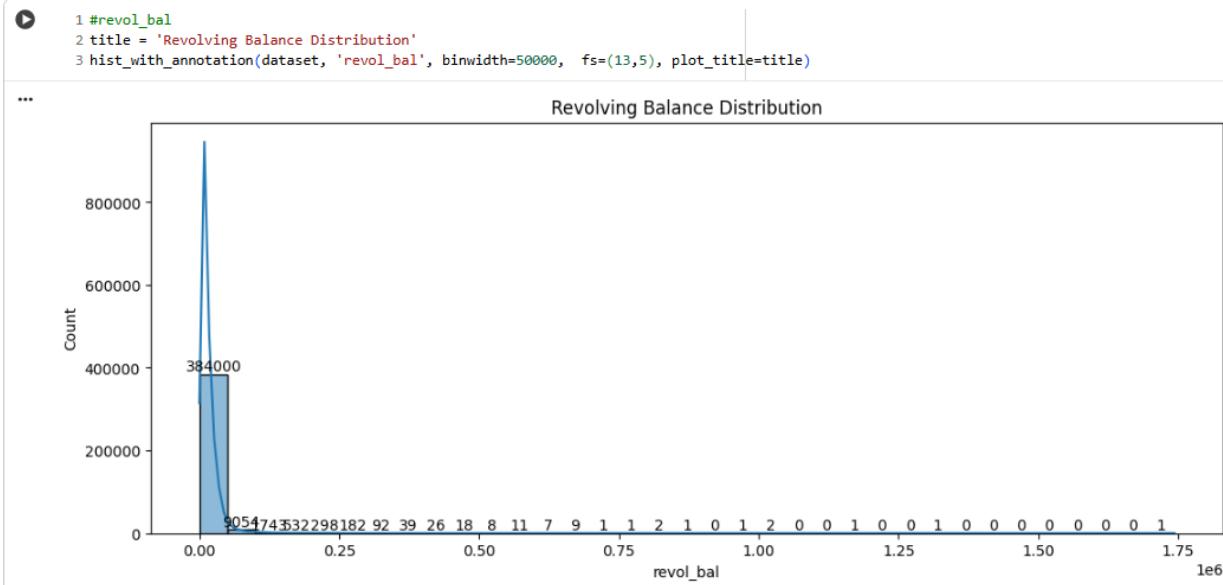


Figure 2.24: Revolving Balance using Kernel Density Estimate (KDE)

Insights

- The histogram shows that most applicants have a revolving credit balance (revol_bal) well below ₹50,000. Only a handful of borrowers report much higher revolving balances, and values above ₹250,000 are extremely rare. This right-skewed distribution indicates that the platform primarily serves applicants with modest revolving credit use, which is favorable for loan approval and overall credit risk.
- The few outliers represent either high-credit individuals or edge cases that may need special scrutiny. The distribution helps confirm that borrower financial leverage is generally kept in check, reinforcing responsible lending standards.

2.3.2.18 Revolving Utilization using Kernel Density Estimate (KDE)

```

1 #revol_util
2 title = 'Revolving Utilization Distribution'
3 hist_with_annotation(dataset, 'revol_util', binwidth=100, fs=(13,5), plot_title = title)
...

```

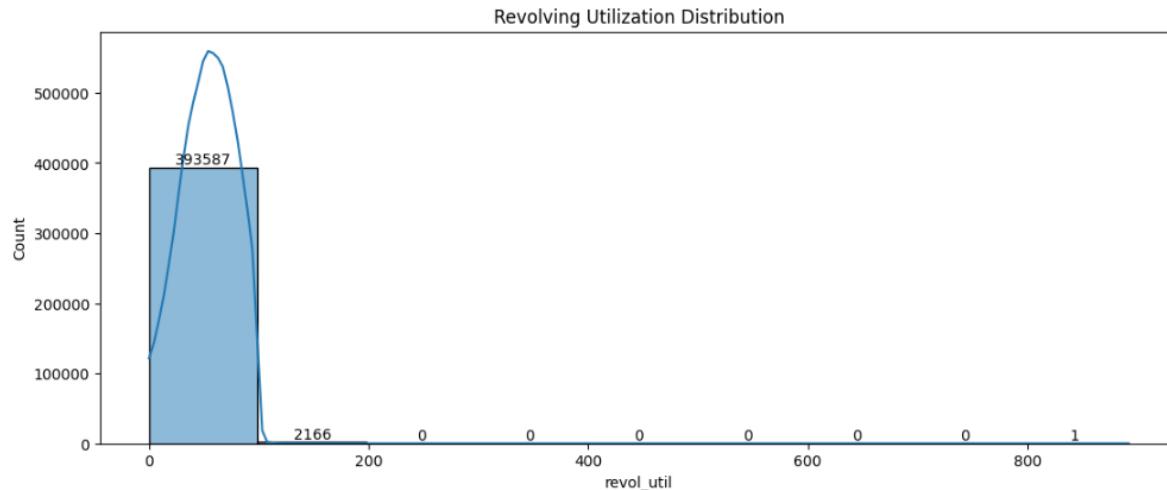


Figure 2.25: Revolving Balance using Kernel Density Estimate (KDE)

2.3.2.19 Total Accounts using Kernel Density Estimate (KDE)

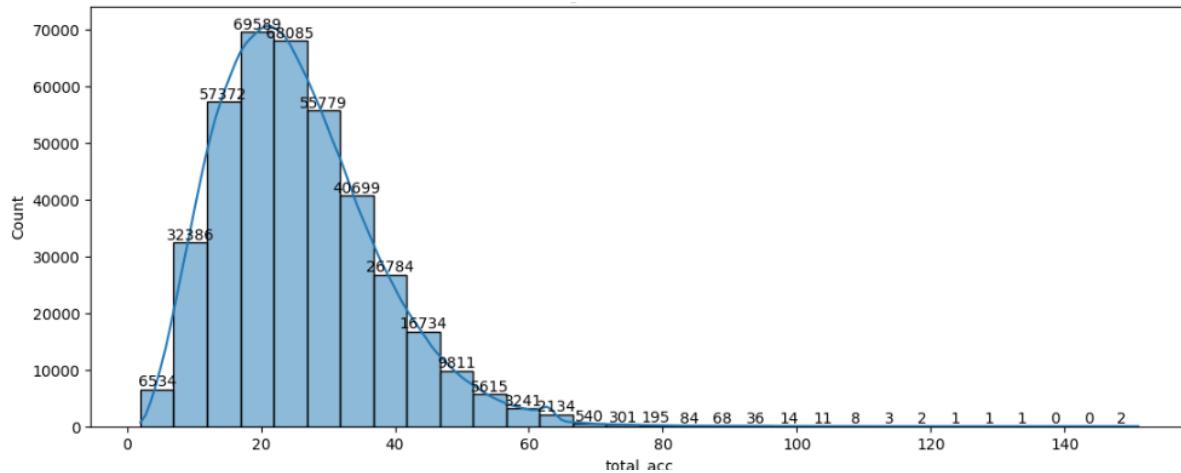


Figure 2.26: Total Accounts using Kernel Density Estimate (KDE)

Insights

- The histogram for revolving utilization rate (`revol_util`) shows most applicants have utilization rates between 0% and 100%, with a strong concentration below 100%. Only a small minority of borrowers exceed this threshold, and values above 200% are exceptionally rare.
- This means that most borrowers are using less than their total available revolving credit, indicative of healthier credit behavior. High revolving utilization can be a risk signal, so the distribution suggests the platform's applicants, in general, maintain reasonable credit management. The few extremely high values deserve special review for potential risk or data issues.

2.3.2.20 Mortgage Accounts using Kernel Density Estimate (KDE)

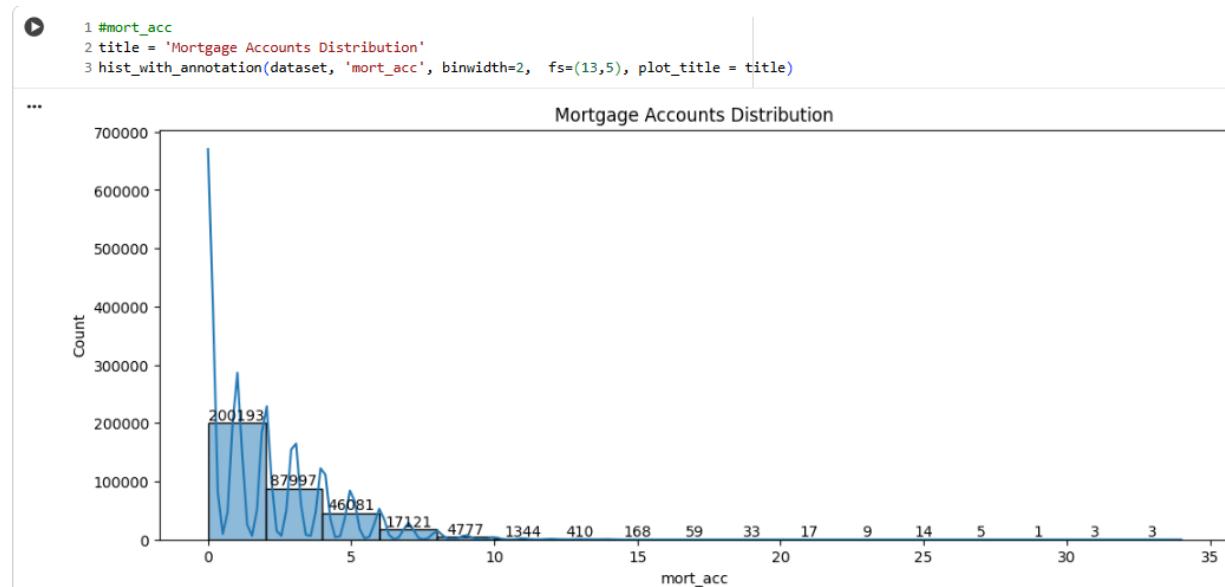


Figure 2.27: Mortgage Accounts using Kernel Density Estimate (KDE)

Insights

- The histogram reveals that the majority of applicants have few or zero mortgage accounts, with a trailing distribution for higher counts. A large proportion of the dataset has between 0 and 4 mortgage accounts, and only a handful of individuals report over 10, with extremely high values being exceedingly rare.
- This pattern suggests that most borrowers are either non-homeowners or maintain only a few active mortgage loans, reflecting broad accessibility for those with limited homeownership exposure. The platform's applicant pool appears to be dominated by

individuals with manageable housing credit histories. Outliers in mortgage account count are minimal and should be reviewed for special lending policies or risk.

2.3.2.21 Public Record Bankruptcies using Kernel Density Estimate (KDE)

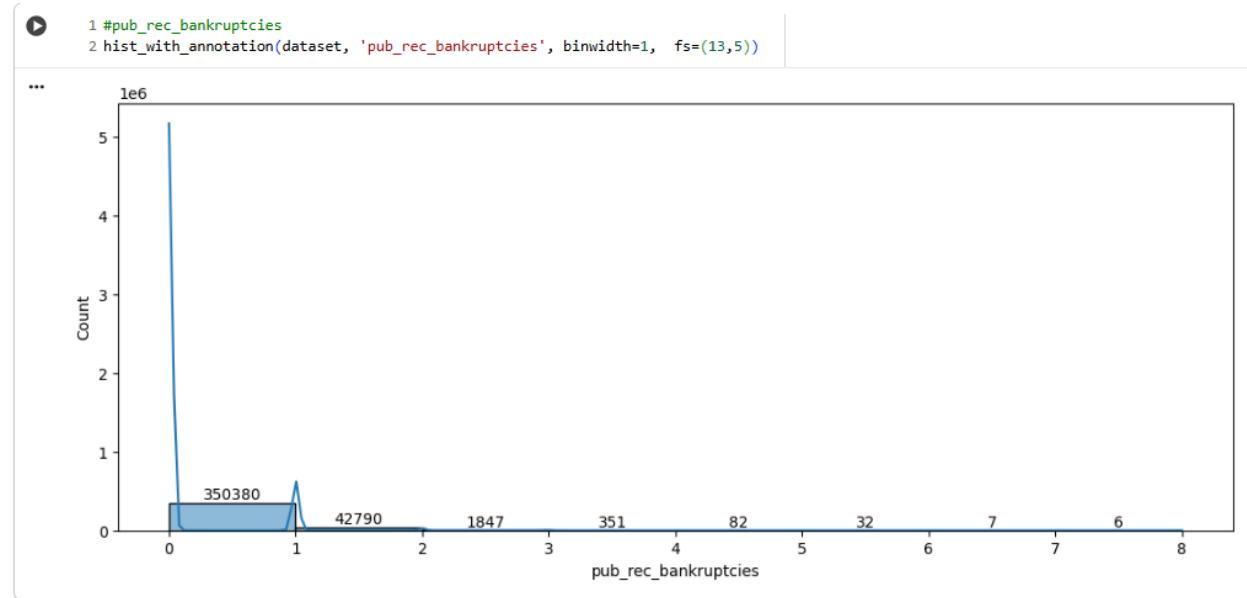


Figure 2.28: Public Record Bankruptcies using Kernel Density Estimate (KDE)

Insights

- The histogram for public record bankruptcies indicates that nearly all applicants have never declared bankruptcy. The count drops sharply as the number of bankruptcies increases, with only a small fraction having one, two, or more bankruptcies on record.
- This pattern confirms most of the lender's clientele have clean bankruptcy histories, which is a strong signal of financial reliability and low credit risk. Outliers with more bankruptcies are very rare and may warrant special underwriting consideration. The overall distribution supports healthy portfolio quality for lending.

2.3.3 Bivariate Analysis

2.3.3.1 Loan Status Count w.r.t Term

```

1 df_loan_term_status = dataset[['term','loan_status']]
2 df_loan_term_status_count = df_loan_term_status.value_counts().reset_index()
3 display(df_loan_term_status_count)
4 plt.figure(figsize=(8, 6))
5 ax = sns.barplot(x='term', y='count', hue='loan_status', data=df_loan_term_status_count)
6 plt.title('Loan Status Count by Term')
7 plt.xlabel('Term')
8 plt.ylabel('Count')
9 sns_annotation(ax)
10 plt.show()

```

```

***      term  loan_status   count
0  36 months    Fully Paid  254365
1  60 months    Fully Paid  63992
2  36 months    Charged Off  47640
3  60 months    Charged Off  30033

```

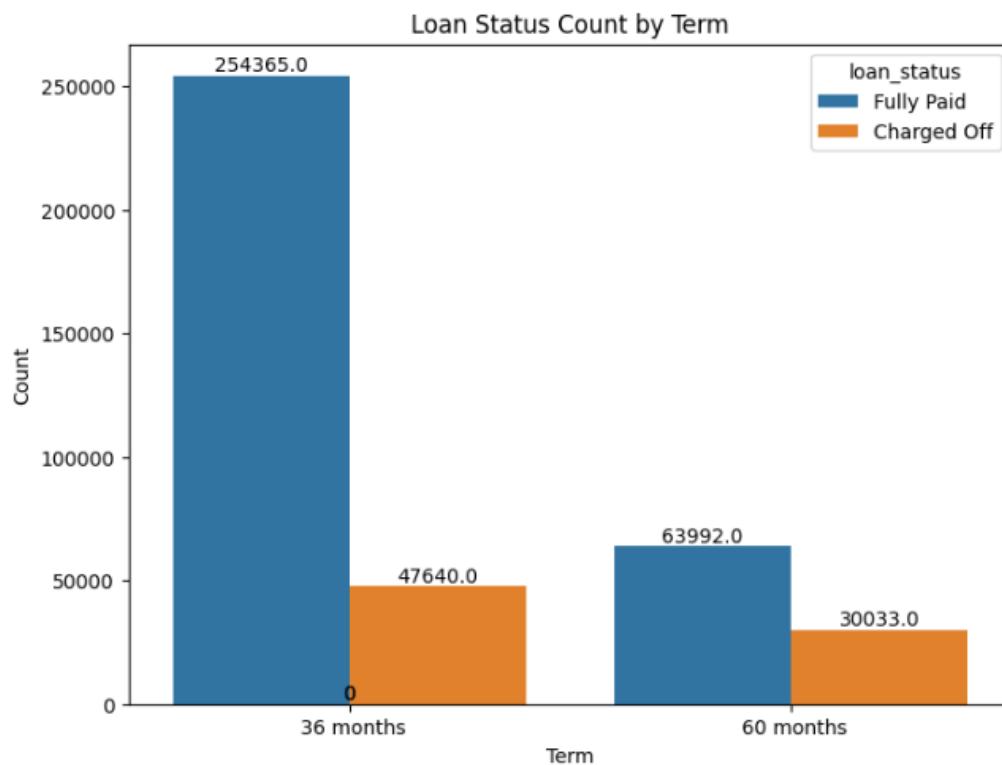


Figure 2.29: Loan Status Count w.r.t Term

Insights

This chart compares the loan repayment outcomes for different loan terms (36 and 60 months) in a loan business case, showing "Fully Paid" versus "Charged Off" (defaulted) loans.

- Higher Repayment Success for Shorter Terms: The vast majority of 36-month loans are fully paid off (254,365), while only 47,640 are charged off. This demonstrates a higher success rate for shorter-term loans.
- Increased Default Proportion for Longer Terms: For 60-month loans, while 63,992 are fully paid, a substantial 30,033 are charged off. The default rate (charged off/total issued) is noticeably higher for 60-month loans compared to 36-month loans.
- Risk Profile: These results highlight that longer-term (60-month) loans carry more risk for the lender. Borrowers may be more likely to default as loan duration increases, signaling the need for stricter risk assessment or adjusted pricing on longer-term products.
- Business Strategy Implication: The lender may want to promote 36-month products and/or implement risk-adjusted interest rates, more rigorous credit checks, or additional support for customers opting for 60-month loans to mitigate potential losses.

These findings are crucial for product strategy, risk management, and pricing in any loan portfolio analysis.

2.3.3.2 Loan Status Count w.r.t Different Grades



Figure 2.30: Loan Status Count w.r.t Different Grades

Insights

This visualization and table analyze loan repayment outcomes by credit grade, which are key in assessing portfolio quality and lending risk in the loan industry.

- **Default Risk Correlates with Grade:** The percentage of “Charged Off” (defaulted) loans increases as credit grade worsens. Grades B and C (mid-risk) show default rates of 12.57% and 21.89% respectively, while riskier grades F and G have much higher default rates (F: 42.78%, G: 47.84%).
- **Best Portfolio Performance at High Grades:** Grades A and B have the highest “Fully Paid” ratios (A: 94%, B: 87%). These customers represent the strongest portfolio assets and lowest risk for the lender.
- **Credit Quality Drives Strategy:** As the grade worsens from D to G, not only does the count decrease, but the “Charged Off” ratios rise sharply—Grade G balances are almost equally split between full repayment and default. This suggests stringent risk controls and pricing are required at lower grades.

- **Volume Distribution:** Most loans are issued in grades B and C, highlighting a business focus on moderate-risk customer segments.

Business Implication:

Lenders can use this analysis to adjust interest rates, develop targeted support programs, refine underwriting standards, and focus customer acquisition on higher-grade segments to optimize long-term profitability and reduce losses from defaults. Concentrating marketing efforts on grades with consistently solid repayment improves portfolio quality.

2.3.3.3 Loan Status Count w.r.t Different Sub Grades

```

1 df_sub_grd_loan_status_count = dataset[['sub_grade', 'loan_status']].value_counts().reset_index()
2 df_sub_grd_loan_status_count
3 grades = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G'], dtype='str').reshape(7,1)
4 values = np.array([1,2,3,4,5,6,7], dtype='str').reshape(1,7)
5 subGrades = np.char.add(grades, values)
6 plt.figure(figsize=(15,20))
7
8 for i in range(7):
9     plt.subplot(4, 2, i+1)
10    plt.title(f"Sub-grades of grade '{grades.flatten()[i]}' and respective loan status count")
11    ax =sns.barplot(x='sub_grade', y='count', hue='loan_status', \
12                    data=df_sub_grd_loan_status_count.loc[:,['sub_grade', 'loan_status']])
13    sns_annotation(ax)
14
15 plt.show()
```

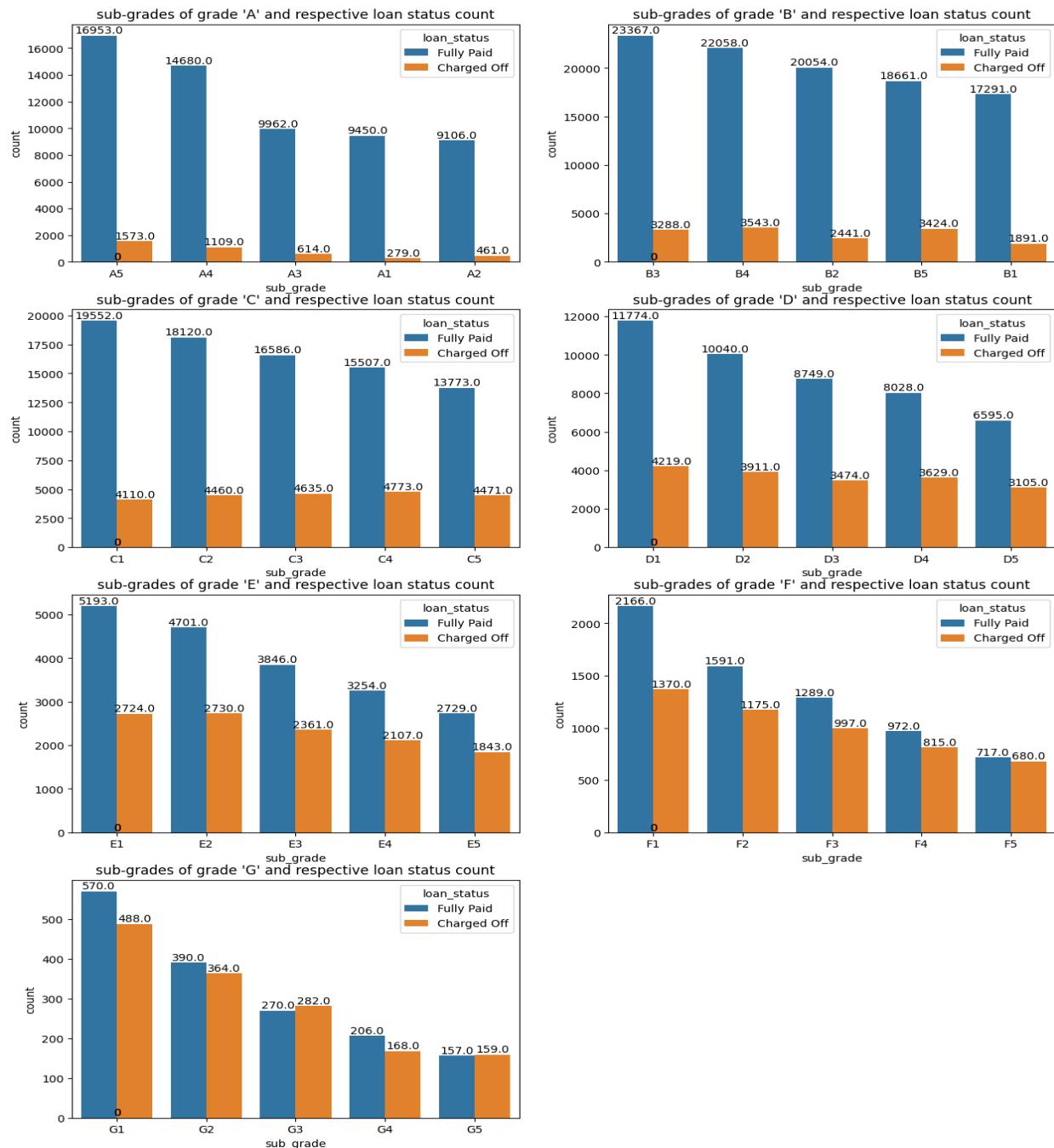


Figure 2.31: Loan Status Count w.r.t Different Sub Grades

Insights

This set of bar charts analyzes loan status (Fully Paid vs. Charged Off) across the sub-grades (risk bands) within each major credit grade, providing a nuanced view of repayment behavior in a loan portfolio.

- **Higher Sub-grade Means Higher Risk:** For every grade (A to G), lower sub-grades (e.g., A5, B5, C5, etc.) have distinctly higher frequencies of “Charged Off” loans compared to better sub-grades (e.g., A1, B1, C1), clearly illustrating that default risk increases as sub-grade worsens within each grade.
- **Repayment Strength Concentrated in Top Sub-grades:** Across almost all grades, the best-performing sub-grades (ending with 1 or 2) have the largest counts of fully paid loans and the lowest defaults. For instance, sub-grades A1 and B1 have far more “Fully Paid” counts than “Charged Off.”
- **Mid-tier Sub-grades Show Moderate Risk:** Sub-grades in the middle of each grade (such as B3, C3, D3) show substantial “Charged Off” numbers—a transition in risk profile is evident.
- **Lowest Grades Are Highly Risky:** In grades F and G, even the best sub-grades have charge-off counts almost equal to fully paid counts. For example, in sub-grade G1, “Charged Off” and “Fully Paid” are nearly balanced, underscoring elevated default risk.
- **Portfolio Focus Recommendation:** The lender should focus acquisition and retention efforts on borrowers in higher sub-grades (A1–A3, B1–B3, C1–C3) for the best returns and portfolio stability, while considering adjusted risk pricing or stricter approval for borrowers below these bands.

Business Application:

This analysis enables lenders to design more refined credit products, select appropriate interest rates, and tailor risk controls to maximize portfolio returns and minimize charge-off losses at a granular level.

2.3.3.4 Loan Status Count w.r.t Employee Length

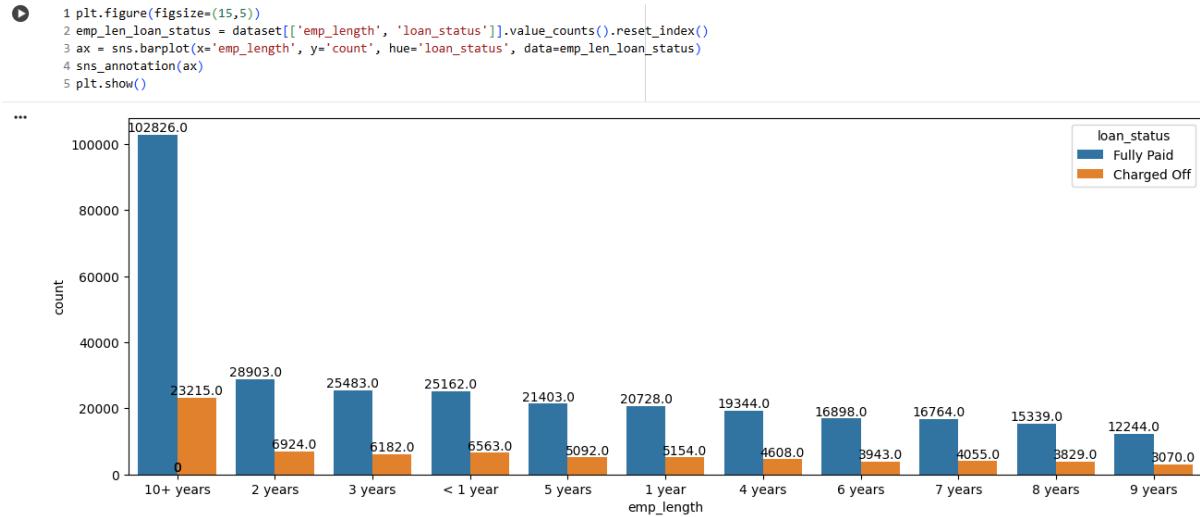


Figure 2.32: Loan Status Count w.r.t Employee Length

Insights

The bar chart shows the distribution of loan repayment status (“Fully Paid” vs. “Charged Off”) across various employment length categories for borrowers.

- **Greater Experience, Better Repayment:** Borrowers with “10+ years” of experience have the highest number of fully paid loans by a wide margin, as well as the largest application volume overall. This group also has a relatively lower proportion of charge-offs, confirming that job stability is associated with lower lending risk.
- **Shorter Tenure, Higher Risk:** All other employment length categories (from less than 1 year to 9 years) have lower counts of "Fully Paid" loans and proportionally more charge-offs, with some volatility but a relatively consistent trend: less experience is correlated with slightly higher default rates.
- **Broad Participation:** Despite higher risk, all employment groups—including those with very short or inconsistent job tenure—still account for substantial fully paid loan counts, indicating diversified risk in the portfolio.

Business Implication:

Lenders should assign greater weight to employment experience during risk assessment and pricing. At the same time, maintaining options for newer employees (with suitable risk control) supports a broad and inclusive business strategy

2.3.3.5 Loan Status Count w.r.t Home Ownership

```

1 plt.figure(figsize=(10,5))
2 home_os_loan_status = dataset[['home_ownership', 'loan_status']].value_counts().reset_index()
3 ax = sns.barplot(x='home_ownership', y='count', hue='loan_status', data=home_os_loan_status)
4 sns_annotation(ax)
5 plt.show()

```

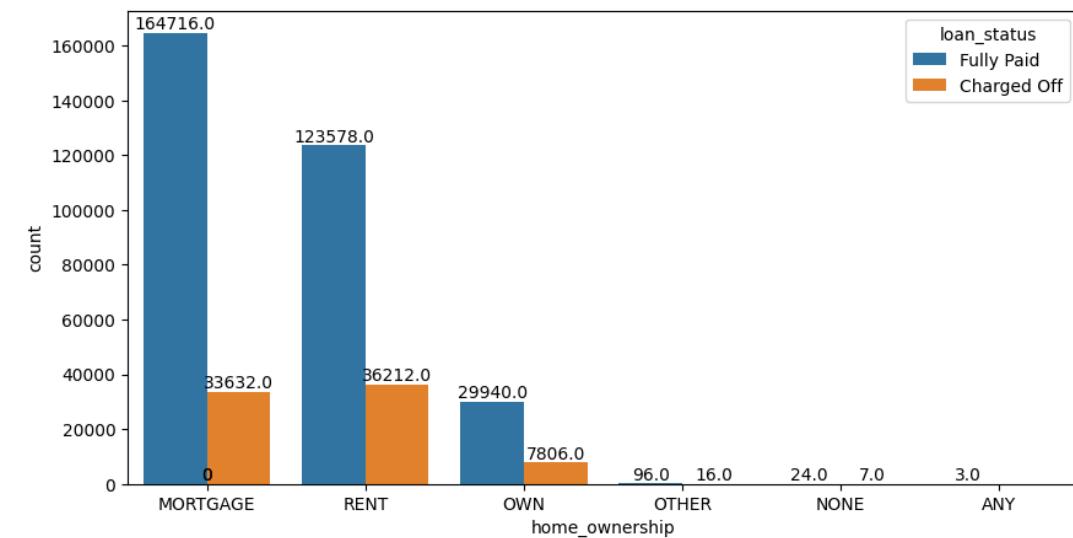


Figure 2.33: Loan Status Count w.r.t Home Ownership

Insights

- **Mortgage Holders are the Largest and Most Reliable Group:** Borrowers with mortgages form the largest share of fully paid loans, and while they do have a significant number of charge-offs, the proportion of successful repayments is the highest. This suggests financial stability or effective risk controls for these customers.
- **Renters have Elevated Risk:** While renters account for a sizable number of fully paid loans, they nearly match mortgage holders in count of charged-off loans, indicating a relatively higher default risk among renters.
- **Homeowners Are the Safest Segment:** Those who own their homes outright ("OWN") have the highest fully paid-to-charge-off ratio, reflecting outstanding credit reliability.
- **Other, None, Any are Insignificant:** The categories "OTHER," "NONE," and "ANY" have minimal impact due to their negligible loan counts.

Business Implication

Home ownership status is a powerful indicator for loan underwriting. Lenders should optimize their risk and pricing models to favor owners and mortgage holders, while continuing cautious assessment for

2.3.3.6 Loan Status Count w.r.t Verification Status

```
1 # verification_status
2 plt.figure(figsize=(10,5))
3 vs_loan_status = dataset[['verification_status', 'loan_status']].value_counts().reset_index()
4 ax = sns.barplot(x='verification_status', y='count', hue='loan_status', data=vs_loan_status)
5 sns_annotation(ax)
6 plt.show()
```

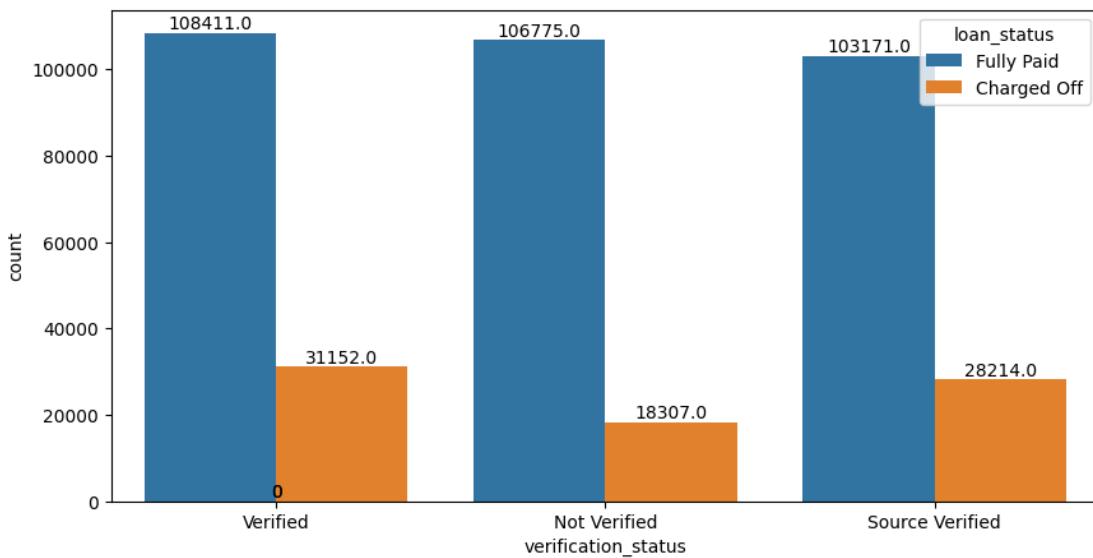


Figure 2.34: Loan Status Count w.r.t Verification Status

Insights

- Verification Matters:** Loans with “Verified” status have the highest count of fully paid loans, and although their charge-off count is also considerable, the ratio is more favorable than “Source Verified.”
- Lowest Risk: Verified and Not Verified:** Both “Verified” and “Not Verified” categories have comparable fully paid numbers, but “Not Verified” loans have noticeably lower charge-off counts, possibly indicating extra caution in lending when full verification is absent.
- Source Verified at Higher Risk:** The “Source Verified” group shows the highest charge-off proportion among the three categories, signaling that independent verification processes may not be as reliable for predicting repayment.
- Business Implication:**
Companies should value robust verification processes, as these yield higher repayment rates. “Not Verified” loans require stricter eligibility, while “Source Verified” applicants should be closely monitored or offered loans at higher rates to manage risk. This enables lenders to tailor risk policies and pricing by verification type for improved portfolio health.

2.3.3.7 Loan Status Count w.r.t Purpose

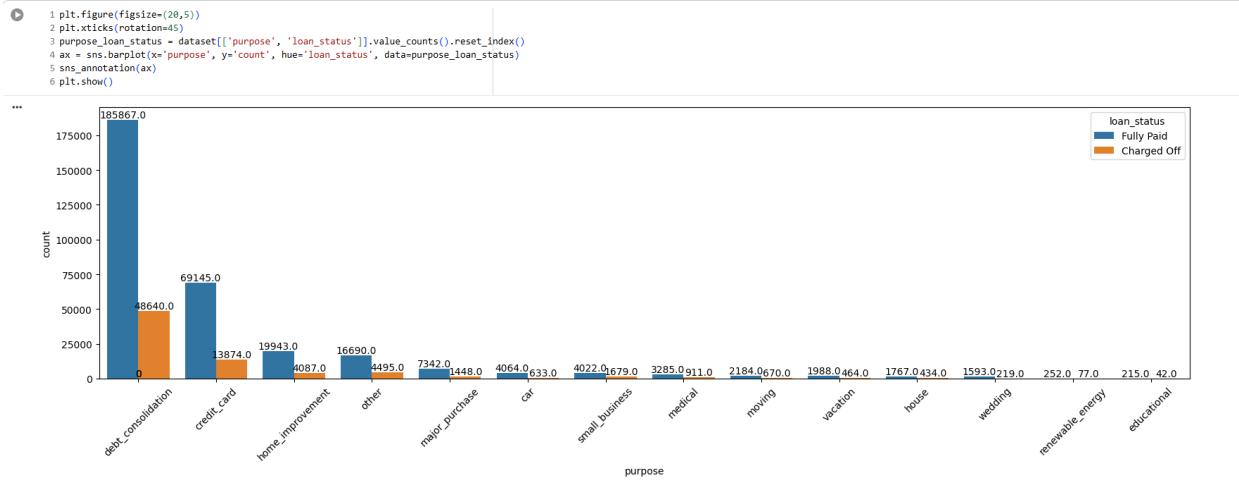


Figure 2.35: Loan Status Count w.r.t Purpose

This chart examines loan repayment outcomes (fully paid vs. charged off) based on the loan's stated purpose.

Insights

- Debt Consolidation and Credit Card Loans Dominate:** Most loans are for “debt consolidation” or “credit card” refinancing. They have the largest absolute fully paid and charged off counts. However, the proportion of charge-offs is substantial, especially for debt consolidation, highlighting inherent risk in borrowers consolidating existing debts.
- Lower Risk Purposes:** Purposes like “home improvement”, “car”, “major purchase”, and “moving” show both lower total numbers and a lower relative percentage of charge offs, indicating more reliable borrower segments for these loan types.
- Small Business, Medical, and Other Uses are High Risk:** “Small business,” “medical,” and “other” purposes display visibly higher proportions of charge offs versus fully paid, underlining their higher risk and need for more caution or stricter terms by lenders.
- Niche Purposes are Safe but Rare:** Categories like “wedding,” “house,” “vacation,” “renewable energy,” and “educational” loans occur infrequently and are not a major driver of overall portfolio risk, but their small numbers may not justify specialized products unless volume grows.

Business Implication:

Lenders should recognize the higher risk and volume in debt consolidation and credit card loans, pricing accordingly and possibly tightening approval for these categories. Targeted support and education for high-risk niches (small business, medical) could also lower

2.3.3.8 Loan Status Count w.r.t Initial List Status

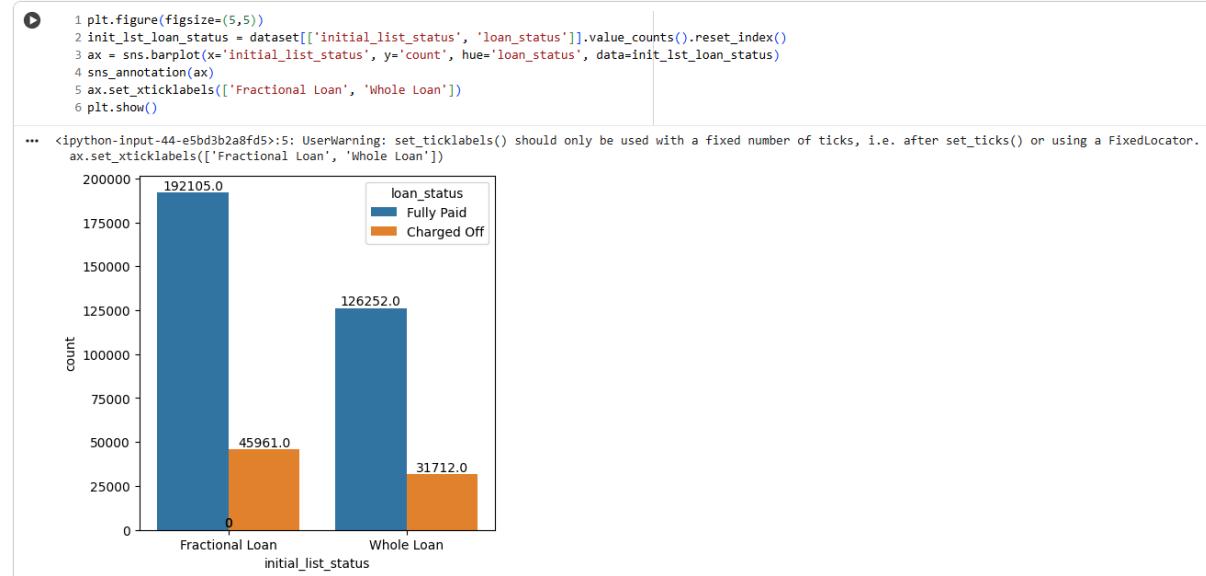


Figure 2.36: Loan Status Count w.r.t Initial List Status

This chart compares the loan outcomes (fully paid vs. charged off) for Fractional Loans and Whole Loans.

Insights

- Fractional Loans Dominate the Portfolio:** Fractional loans show a much higher count of both fully paid and charged off loans when compared to whole loans, indicating they make up the majority of issued loans.
- Default Rate is Similar Across Types:** Both fractional and whole loans exhibit a meaningful number of charge-offs, but their proportions are relatively consistent—default risk is present for both models.
- Wider Investor Distribution in Fractional Loans:** The high volume of fractional loans may point to a preference for distributing risk among multiple lenders and easier accessibility for investors, while whole loans may be higher value or require different risk management.
- Portfolio Management Implication:**
Lenders and platforms should monitor loss rates for both types, ensuring risk-adjusted pricing and support tools are in place for both, but especially important for fractional loans due to their higher volume and collective impact on overall portfolio performance.

2.3.3.9 Loan Status Count w.r.t Application Type

```

1 #application_type
2 plt.figure(figsize=(5,5))
3 apply_loan_status = dataset[['application_type', 'loan_status']].value_counts().reset_index()
4 ax = sns.barplot(x='application_type', y='count', hue='loan_status', data=apply_loan_status)
5 sns_annotation(ax)
6 plt.show()

```

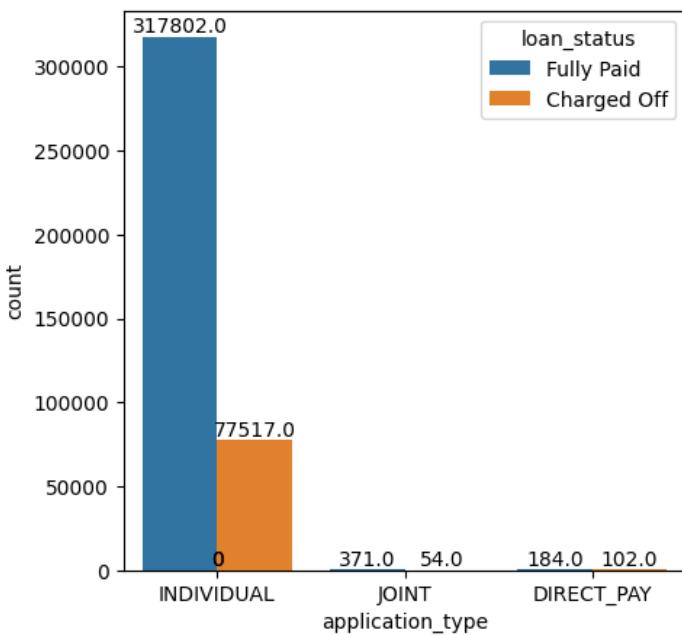


Figure 2.37: Loan Status Count w.r.t Application Type

This chart evaluates loan repayment outcomes across application types: Individual, Joint, and Direct Pay.

Insights

- **Individual Dominates Volume and Defaults:** The overwhelming majority of loans are issued to individual applicants, who account for both the largest number of fully paid and charged-off loans. Thus, overall portfolio performance is heavily tied to individual applicants' risk profile.
- **Joint and Direct Pay Are Rare:** Joint and Direct Pay application types make up a very small slice of total loans, with minimal counts but a visible ratio of charge-offs to fully paid loans. This small sample size makes trends harder to interpret but signals the lender has limited exposure to these categories.
- **Default Risk is Present Across Types:** Even with low sample sizes, charge-offs are present for all types, underlining that default risk is universally relevant and not confined to one application channel.

Business Implication:

Since nearly all lending is to individuals, risk models should be tailored primarily to single-

applicant scenarios. However, monitoring and possibly expanding risk assessment for joint applicants and direct pay options can ensure robust risk coverage as product diversity grows.

2.3.3.10 Loan Status of top 12 Postal Codes

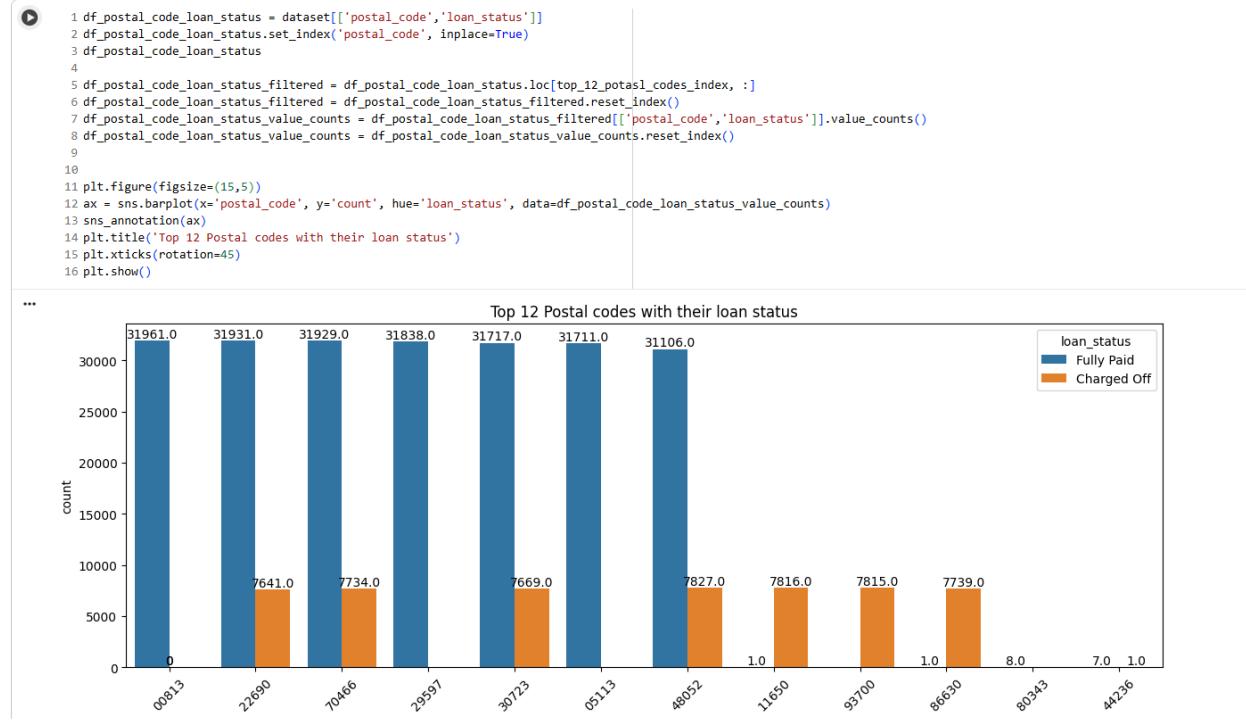


Figure 2.38: Loan Status of top 12 Postal Codes

Insights

- Uniform Distribution Across Postal Codes:** The top 8 postal codes all have very similar counts, around 31,700–31,900 fully paid loans and 7,600–7,800 charged off loans each. This suggests no single region strongly dominates either risk or performance in this top group.
- Consistent Default Rates:** The ratio of fully paid to charged off loans appears stable (~4:1) across these top postal codes, indicating similar portfolio performance regardless of region among the highest-volume locations.
- Long Tail in the Top List:** After the top 8, the remaining postal codes in the list have very low counts, reflecting that most lending activity is highly concentrated. These codes have negligible impact on overall portfolio performance.

2.3.3.11 Purpose of Loan – Top 20

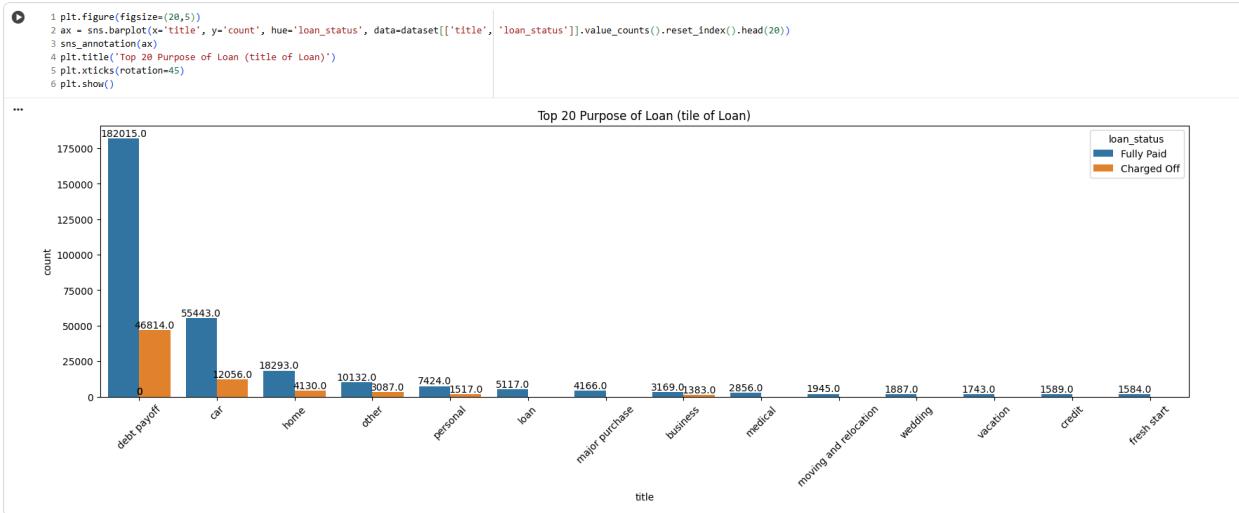


Figure 2.39: Purpose of Loan – Top 20

Insights

- **Debt Payment is the Dominant Reason:** "Debt payment" overwhelmingly forms the largest share of both fully paid and charged off loans. This suggests most borrowers use personal loans for debt consolidation, which carries some risk: the charge-off volume here is also the highest.
- **Other Major Purposes:** "Car" and "home" follow as significant reasons for loans, but with much smaller volumes compared to "debt payment." These categories have lower, but still notable, charge-off counts.
- **Niche Purposes Show Consistency:** Lesser loan purposes (such as personal, loan, major purchase, business, medical, moving/relocation, wedding, vacation, credit, and fresh start) have much fewer loans, but the proportion of charged offs in these smaller segments is nontrivial—indicating risk is spread broadly and not limited to just the high-volume segments.
- **Fully Paid Patterns Hold Across Titles:** In all categories, the number of fully paid loans outweighs charge offs, but the magnitude of difference varies purpose by purpose.

Business Implication:

Targeted risk analytics should focus on the debt payment segment due to its sheer volume and impact, but niche loan purposes should not be ignored as their relative risk to portfolio size can still be material. Lenders may want to implement different risk-based pricing or additional checks for high-risk purposes

2.3.3.12 Loan Status Count w.r.t ‘issue_year’ and ‘issue_month’

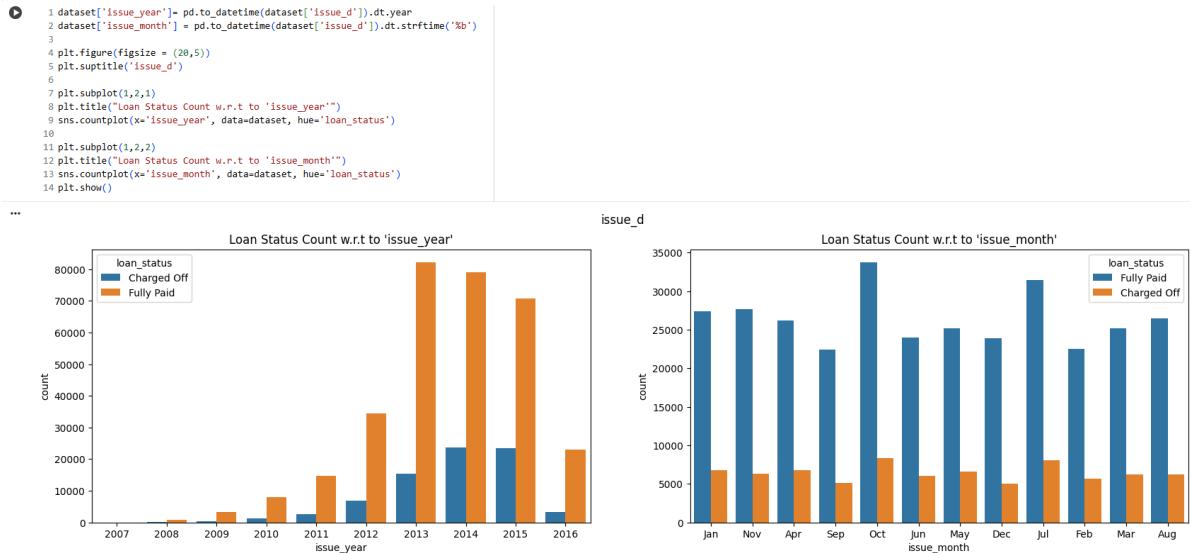


Figure 2.40: Loan Status Count w.r.t ‘issue_year’ and ‘issue_month’

Insights

By Issue Year:

- **Dramatic Growth Over Time:** The number of issued loans grew rapidly, peaking in 2013–2015, then declining after that. This reflects overall business expansion and possibly changing lending conditions.
- **Defaults Rise with Volume:** As loan originations increased, the count of charged off loans also increased, although "Fully Paid" loans always outnumber defaults for each year.
- **Recent Loans Show Fewer Resolved Outcomes:** Toward the last year (2016), fewer loans appear as "Fully Paid" or "Charged Off"—this may reflect loans still in progress or later maturities.

By Issue Month:

- **Monthly Origination is Seasonally Stable:** Lending volumes are relatively consistent throughout the year, with only small peaks in October and July.
- **No Evident Seasonality in Defaults:** There is no strong month-to-month pattern in default (charged off) rates; “Fully Paid” loans outnumber “Charged Off” loans in every month.
- **Charge-Off Volume Tracks Origination:** Charged off loans are generally proportional to fully paid loans within each month, supporting the idea that portfolio risk is steady across the calendar.

Business Implication:

Lending scale heavily influences absolute defaults, not lending season. Risk is more tied to business growth and total originations than to a particular monthly or seasonal pattern. Risk and performance analysis should focus on business cycle changes, volume bursts, and post-peak correction phases more than short-term calendar effects.

2.3.3.13 Top 20 Status of Loan w.r.t to ‘emp_title’

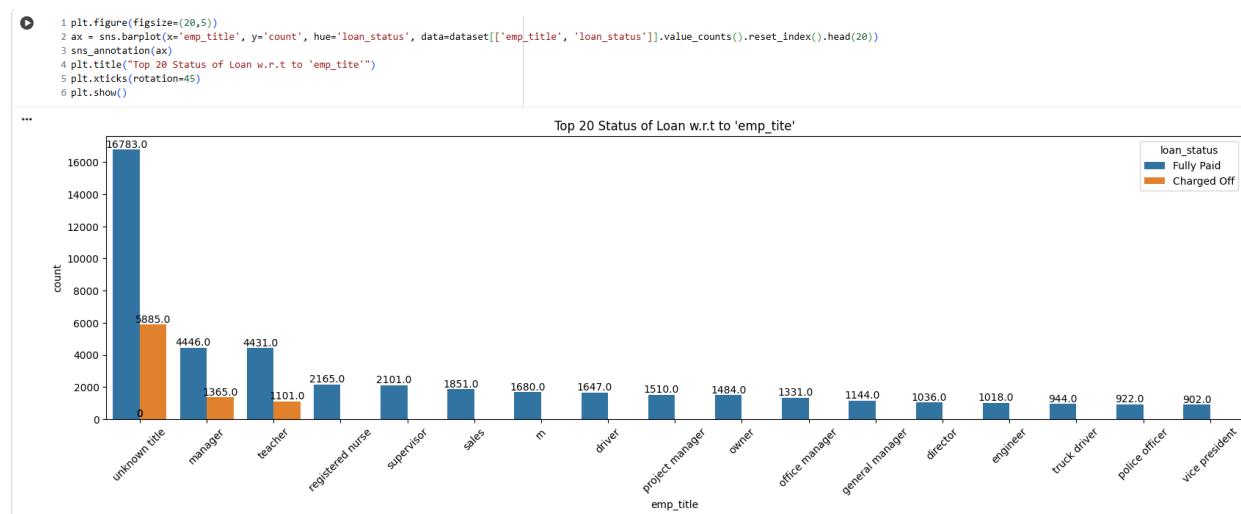


Figure 2.41: Top 20 Status of Loan w.r.t to ‘emp_title’

Insights

- Unknown Titles Dominate Volume:** The “unknown title” category is by far the largest group, suggesting many borrowers did not specify a clear employment title. This group also has the highest number of charged off loans, likely reflecting the risk that accompanies incomplete or ambiguous employment information.
- Consistently Lower Default Across Professions:** Most named professions (manager, teacher, registered nurse, supervisor, sales, driver, etc.) have a much higher count of fully paid loans than charge offs. This pattern indicates that borrowers with identifiable and stable jobs tend to have a lower default risk.
- Volume Declines Quickly After Top Categories:** After the top few titles, the volume of loans per employment title drops off rapidly, indicating that loan portfolio exposure is heavily concentrated among a small set of employment classifications.
- Professional Roles and Repayment:** Professional roles such as manager, teacher, engineer, officer, and vice president have a favorable fully paid-to-charged off ratio, supporting the idea that stable employment is a good risk indicator.

Business Implication:

Lenders should be wary of applications with ambiguous or missing employment information, as these are associated with higher risk. Employment verification and job title can be valuable components of a credit risk model, with certain professions demonstrating greater stability and lower default

2.3.3.14 Loan Status Count w.r.t to 'earliest_cr_line_year' and 'earliest_cr_line_month'

```
1 dataset['earliest_cr_line_year']= pd.to_datetime(dataset['earliest_cr_line']).dt.year
2 dataset['earliest_cr_line_month'] = pd.to_datetime(dataset['earliest_cr_line']).dt.strftime('%b')
3
4 plt.figure(figsize = (20,15))
5 plt.suptitle('earliest_cr_line_year')
6
7 plt.subplot(2,1,1)
8 plt.title("Loan Status Count w.r.t to 'earliest_cr_line_year'")
9 sns.countplot(x='earliest_cr_line_year', data=dataset, hue='loan_status')
10 plt.xticks(rotation=45)
11
12 plt.subplot(2,1,2)
13 plt.title("Loan Status Count w.r.t to 'earliest_cr_line_month'")
14 sns.countplot(x='earliest_cr_line_month', data=dataset, hue='loan_status')
15 plt.show()
```

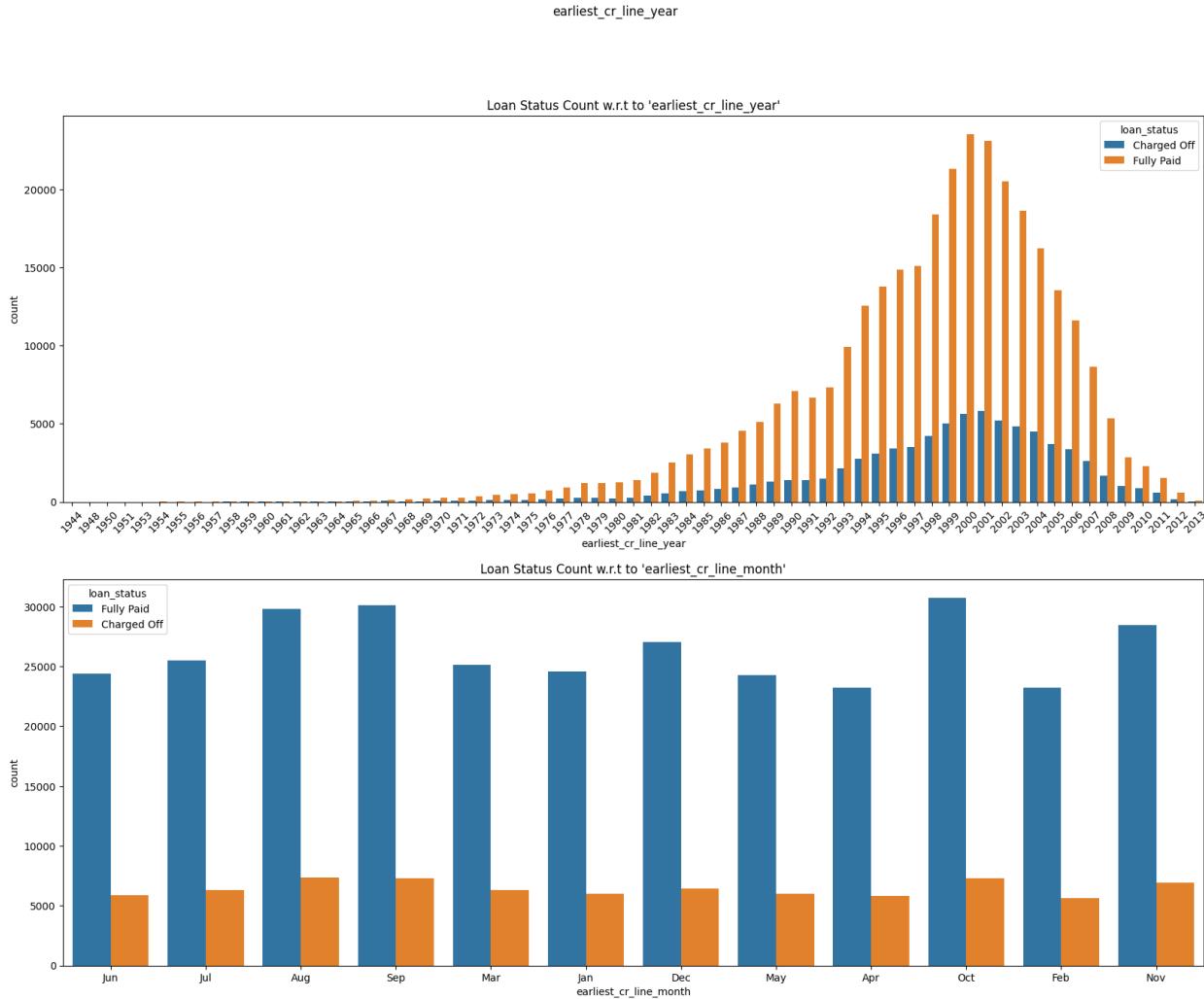


Figure 2.42: Loan Status Count w.r.t to 'earliest_cr_line_year' and 'earliest_cr_line_month'

Insights

By Earliest Credit Line Year:

- **Most Borrowers Began Credit in the 1990s and 2000s:** The largest volume of loans comes from borrowers whose earliest credit lines were opened in 1999–2003. This peak represents people with moderate-to-long credit histories—ideal from a credit risk perspective.
- **Older Credit Histories Less Likely to Default:** Borrowers with the longest credit histories (from the 1960s–1980s) have a higher count of "Fully Paid" loans versus "Charged Off." This indicates that greater credit age may correlate with reliability.
- **Defaults Rise with Total Volume:** Like portfolio averages, default counts increased in line with fully paid loans during the credit expansion of the late 90s/2000s, but the dominance of fully paid loans remains clear, suggesting portfolio risk is generally well-managed.

By Earliest Credit Line Month:

- **Month-of-Origin Has No Clear Risk Pattern:** The month in which a borrower first opened credit (from January to December) has no significant impact on default rates. Each month shows similar proportions of fully paid and charged off loans, indicating no seasonality in this variable.
- **Volume is Evenly Distributed by Month:** Loan counts and proportions don't vary much across months, meaning credit line origination is not a seasonal factor in later loan performance.

Business Implication:

Credit age is a meaningful risk predictor—the longer the history, the better the repayment behavior observed. However, the specific month of credit origination holds little predictive value. Underwriting and risk models should place more weight on credit history length than on minor timing

2.3.3.15 Distribution of Loan Amount w.r.t Loan Status



```
1 plt.figure(figsize=(11,5))
2 plt.subplot(1,2,1)
3 sns.boxplot(x='loan_status', y='loan_amnt', data=dataset)
4 plt.subplot(1,2,2)
5 sns.violinplot(x='loan_status', y='loan_amnt', data=dataset)
6 plt.show()
```

...

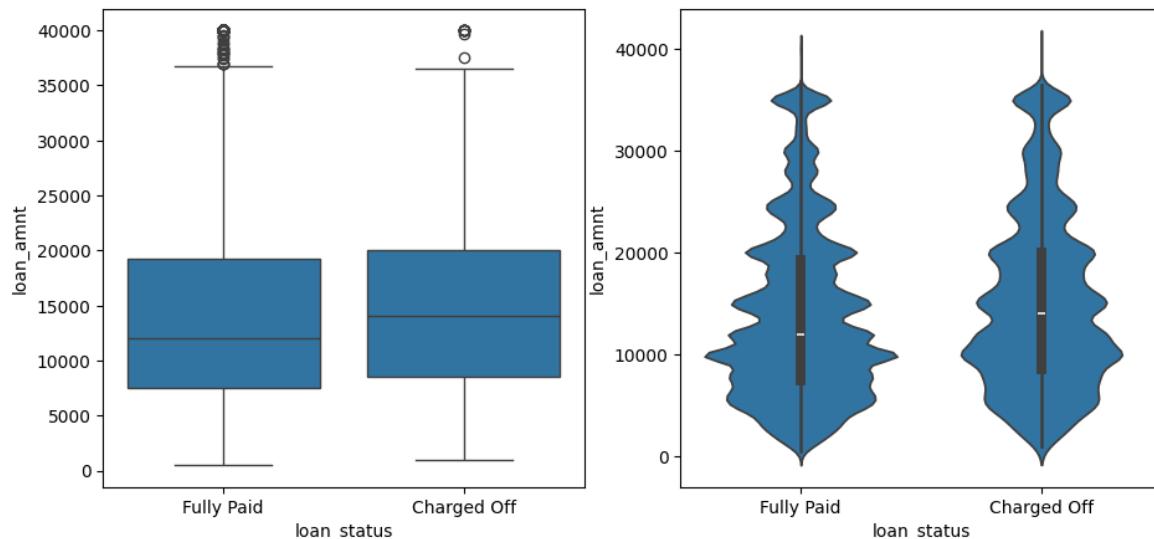


Figure 2.43: Distribution of Loan Amount w.r.t Loan Status

The boxplot and violin plot compare the distribution of loan amounts between "Fully Paid" and "Charged Off" loans.

Insights

- **Loan Amount Distribution is Similar for Both Outcomes:** Both the boxplot and violin plot show that the overall range and median of loan amounts are similar for "Fully Paid" and "Charged Off" loans. This suggests that loan amount alone isn't the key driver of loan defaults.
- **Outliers and Higher Values Exist for Defaults:** Both groups have outliers (loans much larger than the median), but the "Charged Off" category displays a slightly higher 75th percentile and more upper-end outliers. This hints at a modest tendency for larger loans to face repayment risk, but not as a dominant factor.
- **Majority of Loans Cluster in Moderate Ranges:** Most loans—regardless of status—fall between approximately 5,000 and 20,000 units, indicating typical borrower demand and portfolio focus.

Business Implication:

Loan amount should be used together with other applicant characteristics (grade, income, employment, etc.) for risk modeling. Large loans marginally increase risk but do not exclusively drive default. Portfolio management should adopt multi-factor underwriting and not rely solely on loan size.

2.3.3.16 Distribution of Interest Rate w.r.t Loan Status

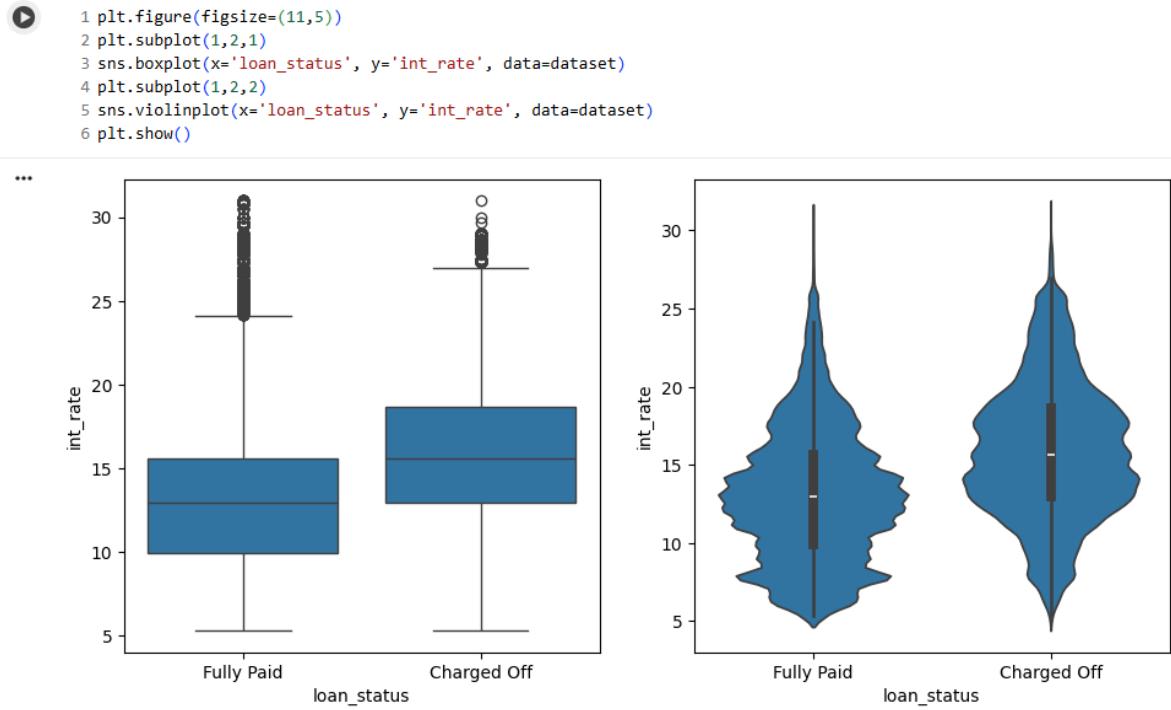


Figure 2.44: Distribution of Interest Rate w.r.t Loan Status

The boxplot and violin plot compare distribution of interest rates between “Fully Paid” and “Charged Off” loans.

Insights

- **Higher Interest Rates Increase Default Risk:** Loans that are “Charged Off” generally have higher median interest rates compared to “Fully Paid” loans. The violin plot visually confirms this pattern with a wider concentration at the upper end for defaults, signifying that high-rate loans are riskier.
- **Spread and Outliers:** Both groups show a wide range of rates, from about 6% up to 30%+, but the "Charged Off" loans in both plots have more outliers at the upper rate boundary, reinforcing that higher rates coincide with higher risk.
- **Overlap Exists:** Despite the difference in medians and concentration, there is substantial overlap—some loans with low interest rates still default, implying that interest rate alone does not fully determine borrower risk.

Business Implication:

Loan pricing must balance yield with risk: higher rates can offset default losses but may also signal poor underlying credit. Lenders should use rate as one input with other applicant data (grade, income, etc.) for predictive risk modeling and design more supportive structures for high-rate borrowers to lower charge

2.3.3.17 Distribution of Installments w.r.t Loan Status

```
1 plt.figure(figsize=(11,5))
2 plt.subplot(1,2,1)
3 sns.boxplot(x='loan_status', y='installment', data=dataset)
4 plt.subplot(1,2,2)
5 sns.violinplot(x='loan_status', y='installment', data=dataset)
6 plt.show()
```

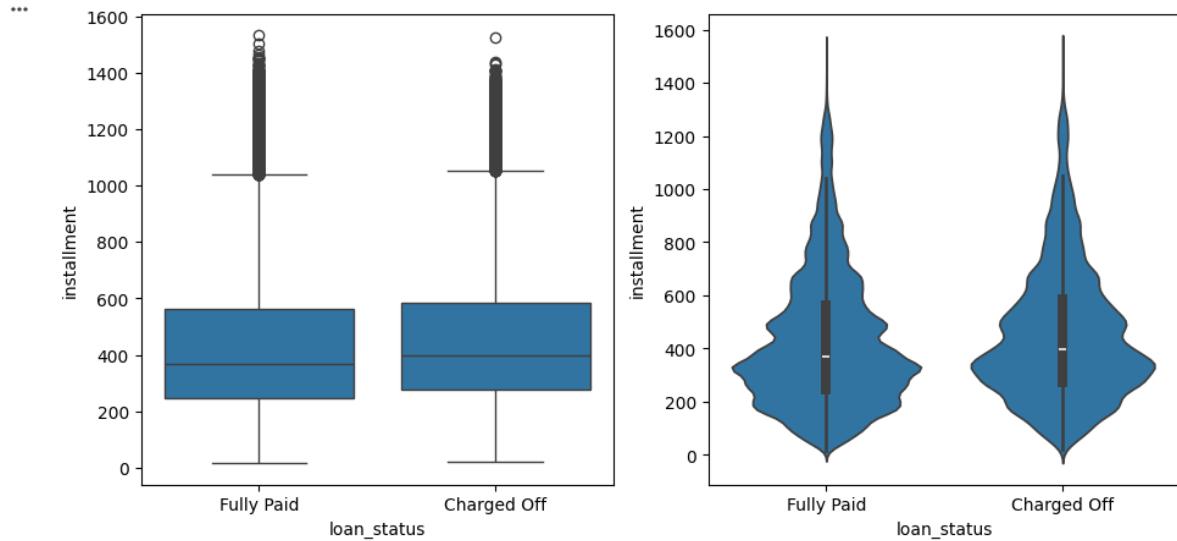


Figure 2.45: Distribution of Installments w.r.t Loan Status

The boxplot and violin plot show the distribution of monthly loan installment amounts for “Fully Paid” and “Charged Off” loans.

Insights

- **Installment Amount Distribution is Similar Across Outcomes:** The median, interquartile range, and overall shape of installment distributions are similar for fully paid and charged off loans. This indicates that default risk is not strongly driven by the actual installment size alone.
- **Outliers/Extremes Occur in Both Groups:** Both groups contain outliers at the upper boundary (very high monthly payments), but these do not dominate the charged off group, suggesting very large installments are not uniquely related to defaults.
- **Most Installments Cluster at Moderate Values:** The density of both plots is highest in the lower-middle range (approximately 200–600 units), representing standard affordability levels for most of the portfolio.

Business Implication:

Lenders should evaluate installment amounts together with other borrower risk factors (income,

grade, term) for risk prediction, rather than rely solely on installment amount. Payment sizing should optimize for borrower affordability but will not by itself guarantee lower charge-off rates.

2.3.3.18 Distribution of Annual Income w.r.t Loan Status

```
▶ 1 plt.figure(figsize=(11,5))
  2 plt.subplot(1,2,1)
  3 sns.boxplot(x='loan_status', y='annual_inc', data=dataset)
  4 plt.subplot(1,2,2)
  5 sns.violinplot(x='loan_status', y='annual_inc', data=dataset)
  6 plt.show()
```

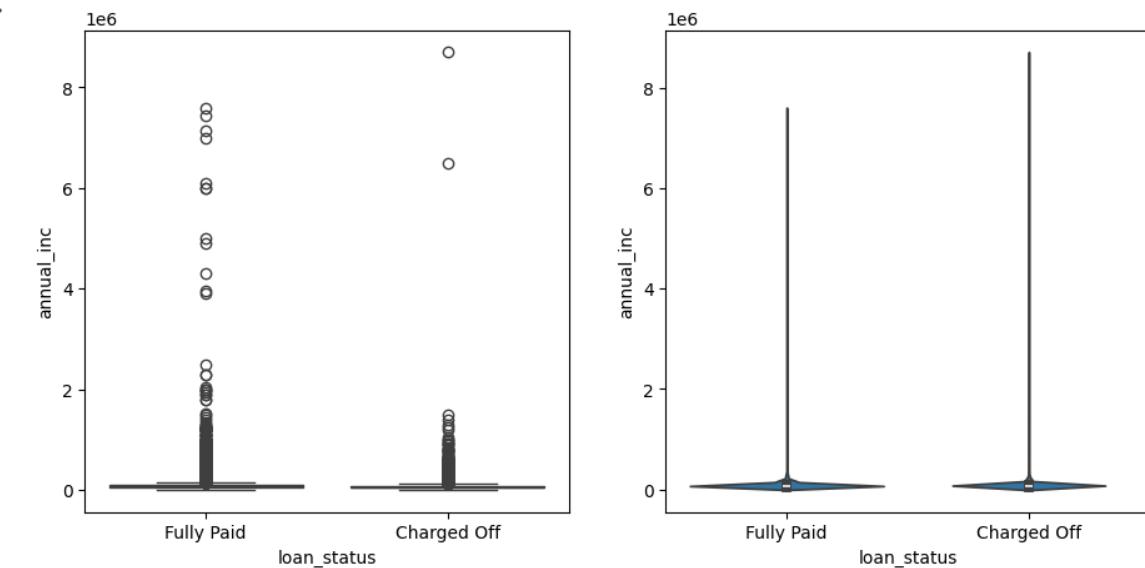


Figure 2.46: Distribution of Annual Income w.r.t Loan Status

The boxplot and violin plot show the distribution of annual incomes for “Fully Paid” and “Charged Off” loans.

Insights

- **Income Levels Similar Across Groups:** The median and most dense regions for annual income are similar for both fully paid and charged off loans in the violin plot. This indicates that higher income does not guarantee loan repayment success.
- **Presence of Extreme Outliers:** Both groups contain outliers with exceptionally high annual incomes; however, these do not appear to determine repayment success. Most loans cluster in the lower-to-middle income range, with only a few instances of ultra-high incomes.
- **Charge-off Not Prevented by High Income:** The patterns show that even high-income borrowers default, emphasizing that income alone is not a sufficient risk control variable.

Business Implication:

Income should be used alongside other factors (grade, employment, loan amount, interest rate) in risk modeling. While low income may suggest risk, high income does not immunize against default, so a holistic credit policy offers better results.

2.3.3.19 D.T.I w.r.t Loan Status

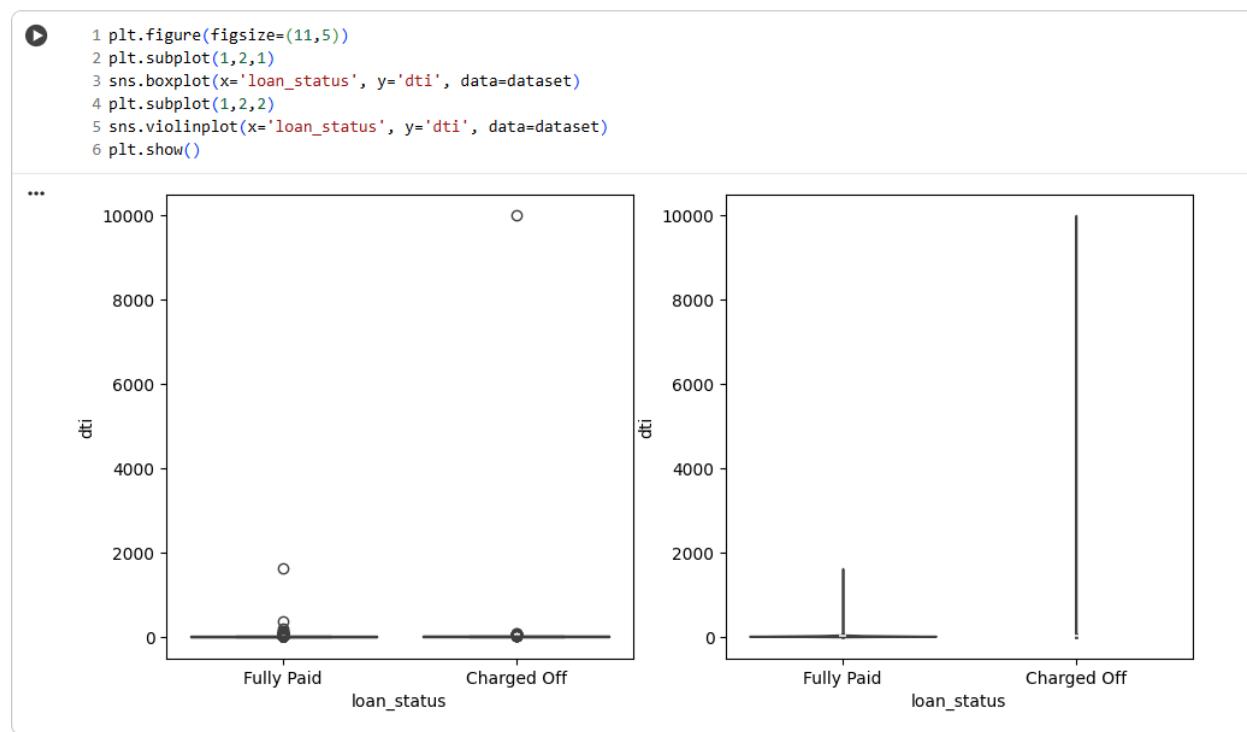


Figure 2.47: D.T.I w.r.t Loan Status

The boxplot and violin plot show the distribution of debt-to-income ratios (DTI) for “Fully Paid” and “Charged Off” loans.

Insights

- DTI Levels Cluster Low for Most Borrowers:** Both fully paid and charged off loans show the vast majority of DTI values are near zero or modest levels, suggesting most approved borrowers do not have excessive debt burdens.
- High-DTI Outliers Occur More in Defaults:** Some charged off loans exhibit extreme DTI outliers, occasionally exceeding 1,000 and even 10,000, compared to fewer high outliers in the fully paid group. This supports the idea that extreme debt loads can increase default risk.

- **DTI Alone is Not a Distinguishing Risk Factor:** For most of the portfolio, the DTI values are similar for both groups, i.e., most defaults do not occur solely because of high DTI.

Business Implication:

DTI is useful for filtering out extreme-risk candidates, but routine lending risk is determined by multiple variables. Lenders should set upper DTI thresholds but combine DTI with other underwriting factors for strong portfolio performance

2.3.3.20 Open Account w.r.t Loan Status

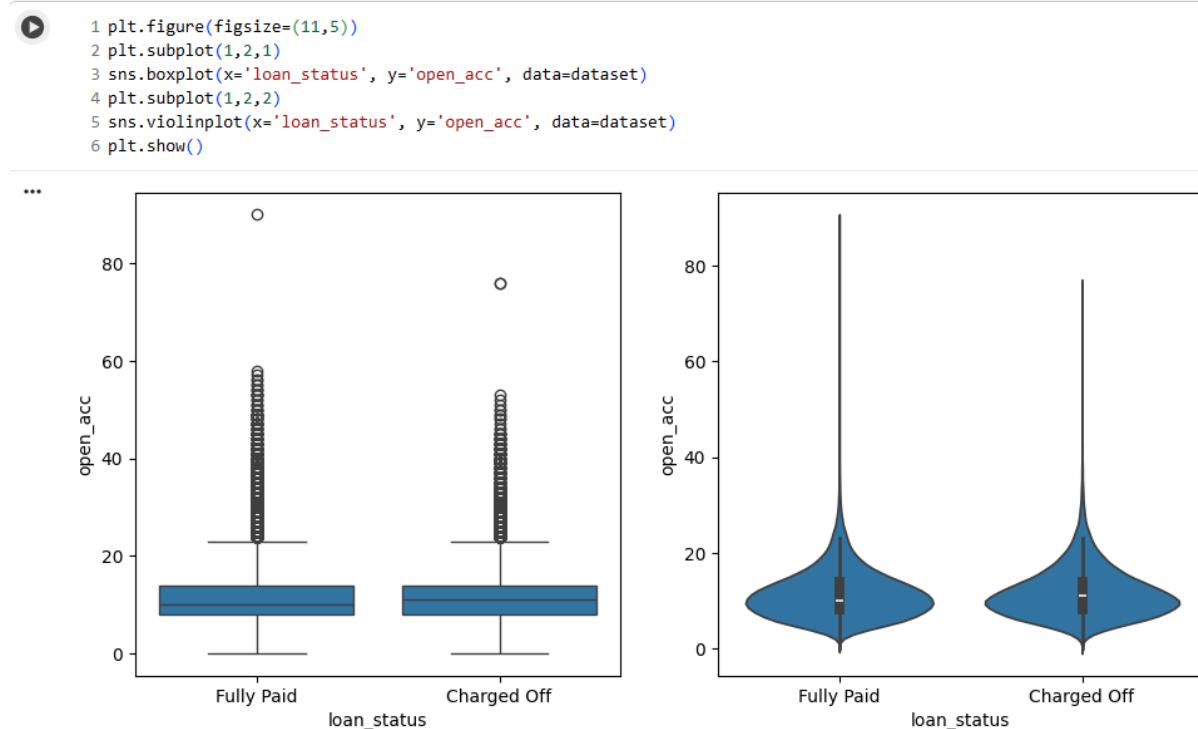


Figure 2.48: Open Account w.r.t Loan Status

The boxplot and violin plot show the distribution of the number of open credit accounts (open_acc) for “Fully Paid” and “Charged Off” loans.

Insights

- **Credit Activity is Similar Across Groups:** Both boxplot and violin plot show nearly identical distributions for fully paid and charged off loans. The medians and spreads overlap, suggesting number of open accounts alone does not strongly predict default.

- **Most Borrowers Hold Between 5 and 20 Active Accounts:** The data is concentrated in this range for nearly all borrowers, highlighting standard consumer credit activity in the portfolio.
- **Outliers are Present but Not Dominant:** Some individuals have notably high numbers of open accounts (over 40 or even 80), but these outliers are rare for both groups and do not strongly affect the overall risk pattern.

Business Implication:

While credit activity measurement is valuable for profile completeness, number of open accounts should be considered in combination with total balances, repayment history, and other variables for effective credit scoring and risk controls

2.3.3.21 Public Record w.r.t Loan Status

```
1 plt.figure(figsize=(11,5))
2 plt.subplot(1,2,1)
3 sns.boxplot(x='loan_status', y='pub_rec', data=dataset)
4 plt.subplot(1,2,2)
5 sns.violinplot(x='loan_status', y='pub_rec', data=dataset)
6 plt.show()
```

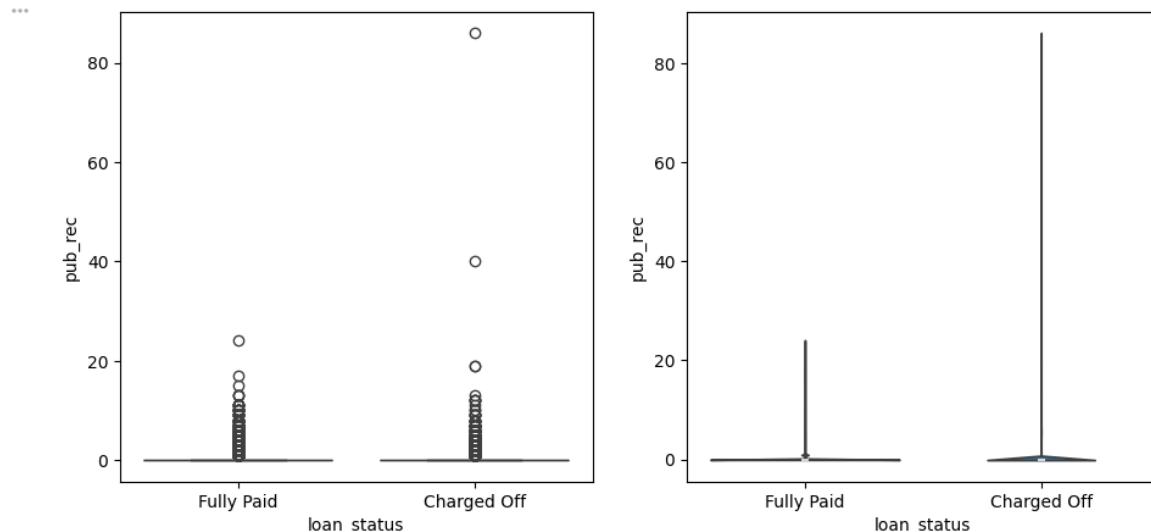


Figure 2.49: Public Record w.r.t Loan Status

The boxplot and violin plot show the distribution of public record counts (pub_rec) for “Fully Paid” and “Charged Off” loans.

Insights

- **Most Borrowers Have Zero Public Records:** Both groups overwhelmingly cluster at zero public records, indicating most applicants have clean legal/credit histories regardless of loan outcome.
- **Outliers are Strongly Associated with Charged Off Loans:** The charged off group contains substantially more outliers with higher public record counts (up to about 80), compared to very few outliers among fully paid loans. This suggests that applicants with prior negative public records are more at risk of defaulting.
- **Pattern is Consistent Across Distribution:** As seen in the violin plot, the shaft (core distribution) is almost entirely near zero for fully paid loans, and slightly more spread out with outliers at the risky tail for charged offs.

Business Implication:

Lenders should strongly consider public record history during risk assessment and underwriting. Zero or very low public records correlate with high repayment success, whereas higher counts signal elevated risk and may require higher pricing, stricter conditions, or outright rejection depending

2.3.3.22 Revolving Balance w.r.t Loan Status

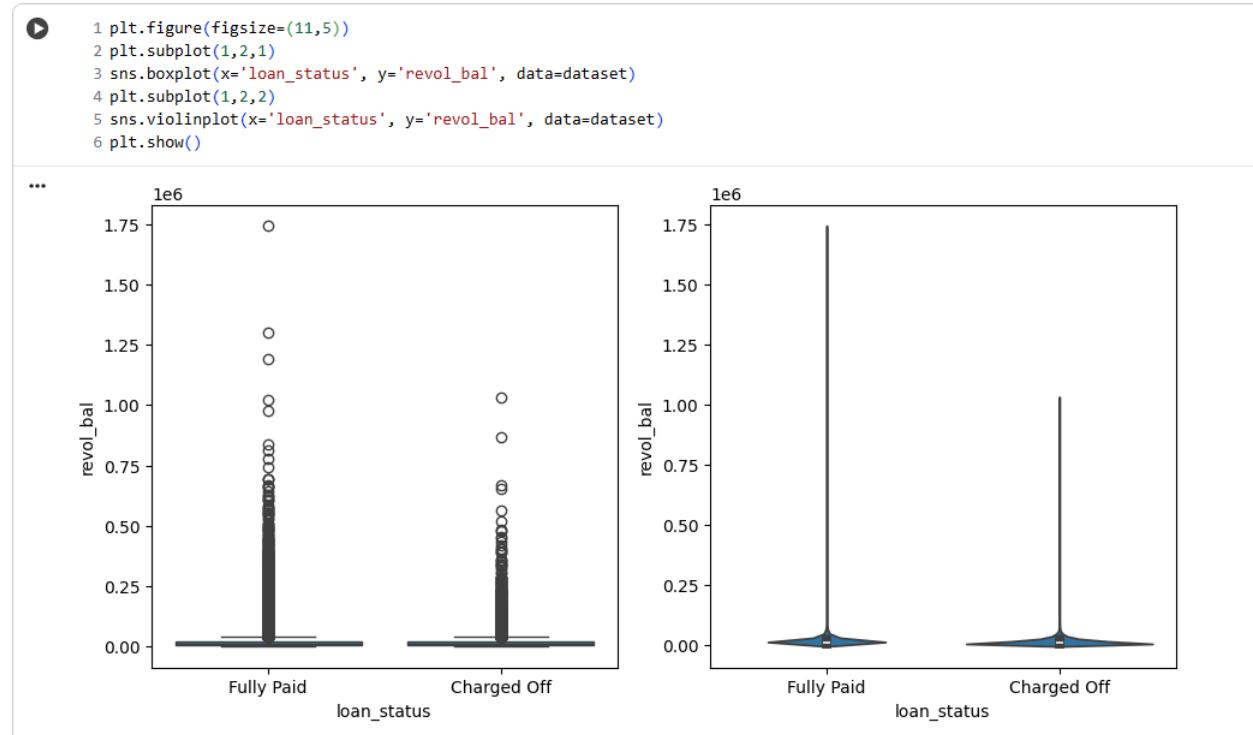


Figure 2.50: Revolving Balance w.r.t Loan Status

The boxplot and violin plot display the distribution of revolving credit balances (revol_bal) for “Fully Paid” and “Charged Off” loans.

Insights

- **Majority Have Modest Balances:** Most borrowers in both groups have relatively low revolving balances, with the bulk of the data concentrated at the lower end of the axis in both the box and violin plots.
- **High-Balance Outliers and Defaults:** Both groups exhibit outliers with very large revolving balances, but these outliers are somewhat more prominent among “Charged Off” loans. This suggests that elevated credit utilization can be associated with slightly increased risk of default, though not exclusively so.
- **Distribution Overlaps Heavily:** Most borrowers, regardless of repayment outcome, share a similar distribution of revolving balances. High balances increase risk but many high-balance borrowers also repay successfully.

Business Implication:

Revolving balance is a useful risk signal, especially for flagging outliers, but is not a stand-alone indicator of default. Lenders should use it as part of a broader risk model that includes payment history, grade, DTI, and income

2.3.3.23 Revolving Line Utilization Rate w.r.t Loan Status

```

1 plt.figure(figsize=(11,5))
2 plt.subplot(1,2,1)
3 sns.boxplot(x='loan_status', y='revol_util', data=dataset)
4 plt.subplot(1,2,2)
5 sns.violinplot(x='loan_status', y='revol_util', data=dataset)
6 plt.show()

```

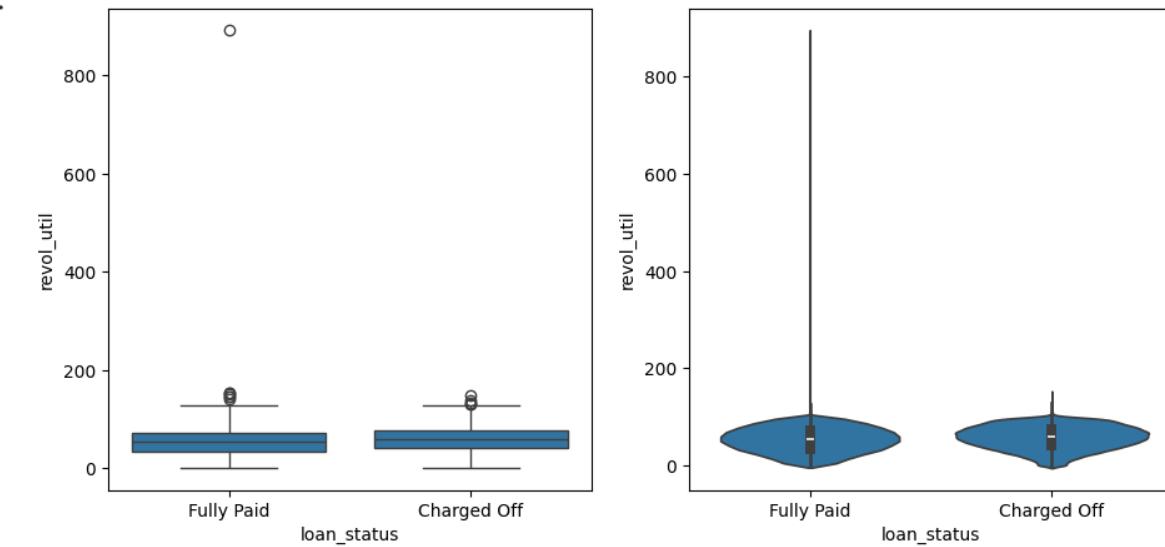


Figure 2.51: Revolving Line Utilization Rate w.r.t Loan Status

The boxplot and violin plot display the distribution of revolving utilization rates (revol_util) for “Fully Paid” and “Charged Off” loans.

Insights

- **Median Utilization is Higher for Defaults:** Both plots show that “Charged Off” loans have a slightly higher median and more density at the upper end of revolving utilization compared to “Fully Paid” loans. This means borrowers using a larger share of their available credit are more likely to default.
- **Extreme Outliers and High Risk:** Outliers with very high utilization (above 300%) are more common in the charged off group. Extremely high utilization is a strong warning sign for default risk.
- **Most Borrowers Cluster Below 100%:** The majority of loans in both groups show utilization well under 100%, but the density spreads wider for defaults.

Business Implication:

Revolving utilization is a valuable feature for risk prediction: the likelihood of default increases as borrowers max out their available credit. Lenders should monitor high utilization closely and may consider adjusted pricing, lower limits, or interventions to manage risk.

2.3.3.24 Total No of Credit Lines w.r.t Loan Status

```
1 plt.figure(figsize=(11,5))
2 plt.subplot(1,2,1)
3 sns.boxplot(x='loan_status', y='total_acc', data=dataset)
4 plt.subplot(1,2,2)
5 sns.violinplot(x='loan_status', y='total_acc', data=dataset)
6 plt.show()
```

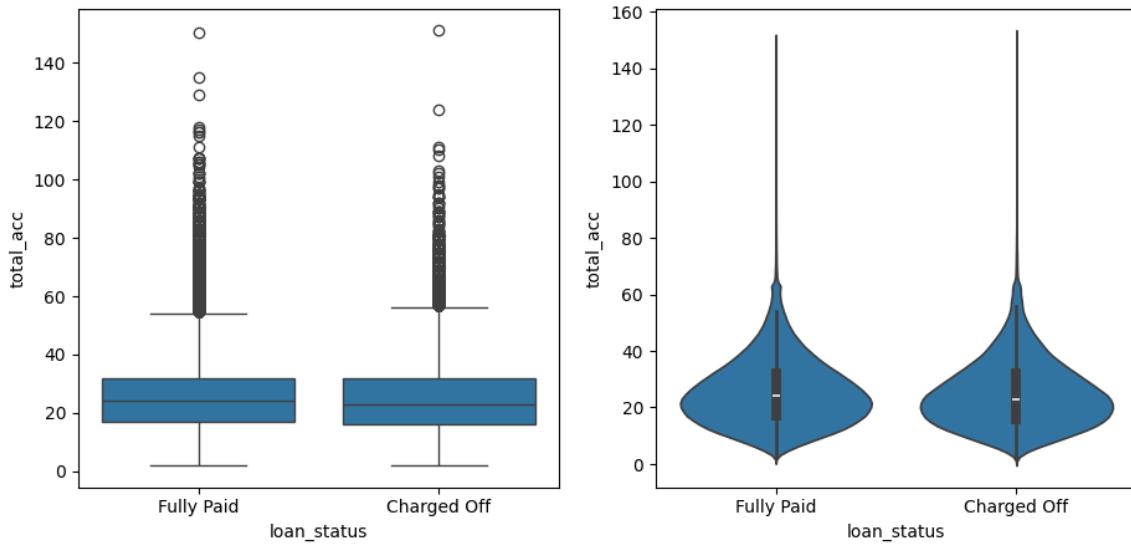


Figure 2.52: Total No of Credit Lines w.r.t Loan Status

The boxplot and violin plot show the distribution of total credit accounts (total_acc) for “Fully Paid” and “Charged Off” loans.

Insights

- **Total Credit Accounts Are Similar for Both Groups:** Both fully paid and charged off loans have very similar distributions for total credit accounts. The medians, interquartile ranges, and the main body of the violin plots overlap almost entirely, indicating that the number of accounts alone does not strongly differentiate between repayment and default outcomes.
- **Majority Cluster Around 20–40 Accounts:** Most borrowers in the dataset have between 20 and 40 total accounts, reflecting standard consumer activity in the credit market.
- **High-Account Outliers Do Not Uniquely Signal Risk:** Outliers with 100+ accounts exist in both groups, but do not overwhelmingly predict default, suggesting additional context (like payment behavior or account types) is needed for risk assessment.

Business Implication:

Total number of credit accounts should be used as one piece of the risk profile puzzle, but on its own it does not provide meaningful predictive power for default. Lenders should combine this

feature with utilization, recent delinquencies, and other behavioral indicators for accurate credit risk modeling.

2.3.3.25 Total No of Mortgage Accounts w.r.t Loan Status

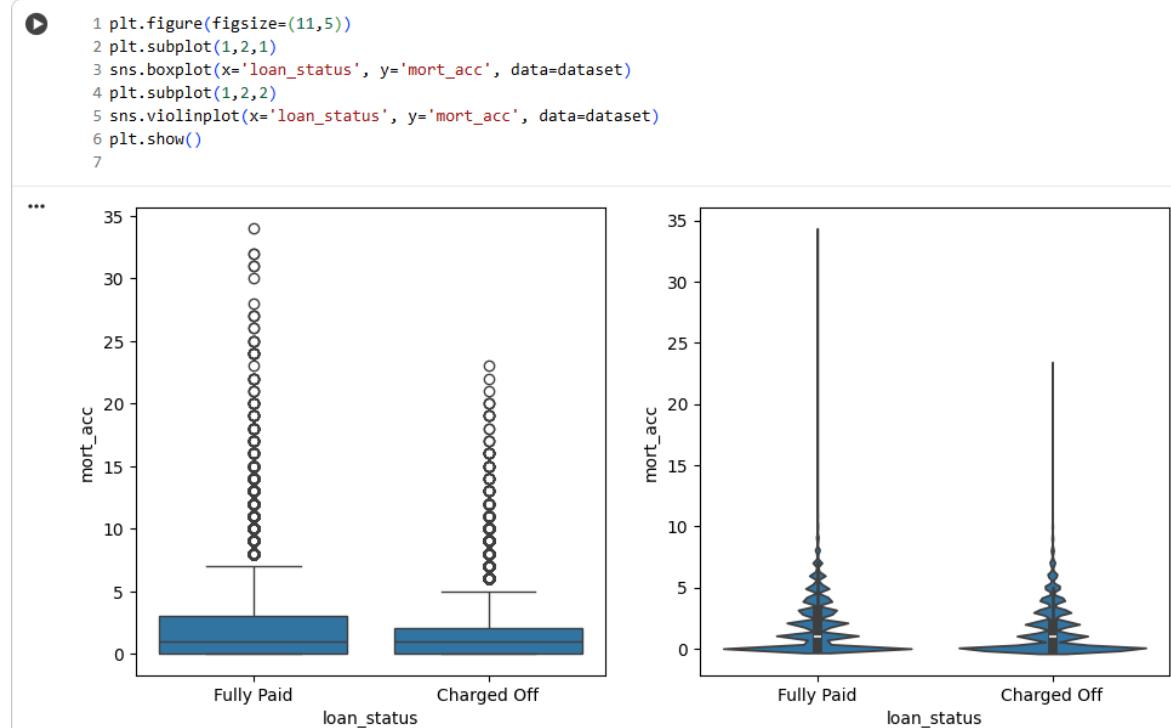


Figure 2.53: Total No of Mortgage Accounts w.r.t Loan Status

The boxplot and violin plot display the number of open mortgage accounts (`mort_acc`) for “Fully Paid” and “Charged Off” loans.

Insights

- **Distribution is Similar Across Loan Status:** Both fully paid and charged off loans show heavy clustering at low values (0–3 mortgage accounts), with a median close to 1 for both. This suggests that simply having a mortgage or a small number of mortgages is typical and not alone a high-risk factor.
- **Outliers Are More Numerous for Fully Paid:** Interestingly, there are more extreme outliers—borrowers with more than 10 mortgage accounts—among the fully paid group than among defaults. This may indicate that those with many mortgages are experienced property holders, managing their credit successfully.

- **Defaults Not Driven by Mortgage Account Count:** The violin plots confirm that having a higher number of open mortgages does not distinctly increase risk of default in this portfolio.

- **Business Implication:**

Number of open mortgage accounts is not a strong standalone predictor of default. For lenders, this suggests that mortgage account count should not be overly weighted in credit scoring without considering repayment history, property value, and overall

2.3.3.26 Total No of Public Record Bankruptcies w.r.t Loan Status

```
▶ 1 plt.figure(figsize=(11,5))
  2 plt.subplot(1,2,1)
  3 sns.boxplot(x='loan_status', y='pub_rec_bankruptcies', data=dataset)
  4 plt.subplot(1,2,2)
  5 sns.violinplot(x='loan_status', y='pub_rec_bankruptcies', data=dataset)
  6 plt.show()
```

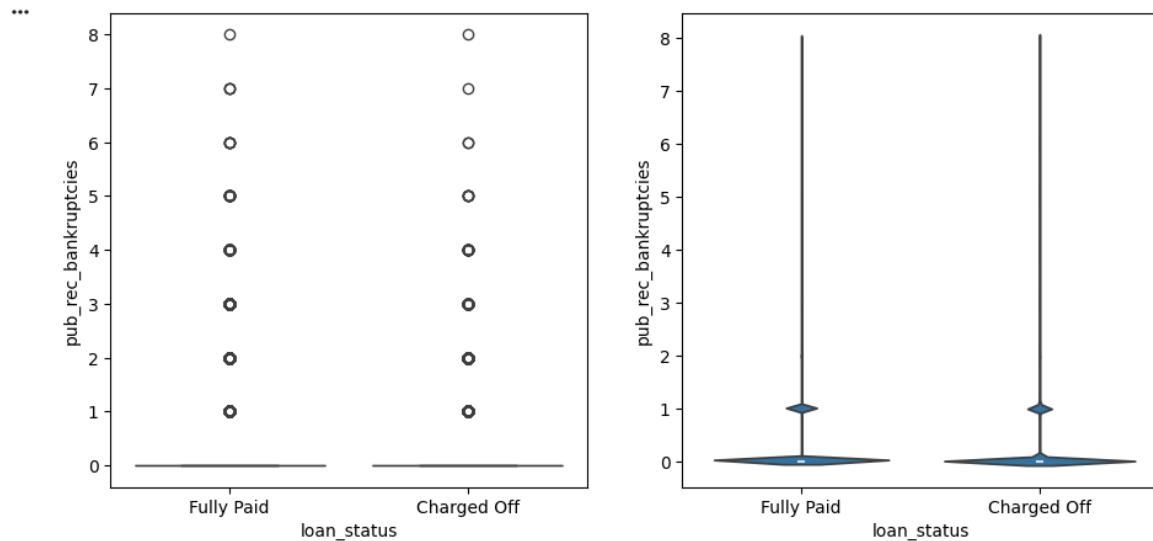


Figure 2.54: Total No of Public Record Bankruptcies w.r.t Loan Status

The boxplot and violin plot illustrate the distribution of public record bankruptcies for “Fully Paid” and “Charged Off” loans.

Insights

- **Most Borrowers Have No Bankruptcies:** Both loan outcome groups overwhelmingly cluster at zero on the number of public record bankruptcies, indicating the majority of borrowers have not experienced bankruptcy, no matter their eventual outcome.

- Multiple Bankruptcies Indicate High Risk:** The charged off group shows a greater presence of outliers who've had multiple bankruptcies, up to 8. Such borrowers, though few, have a noticeably elevated risk of default, suggesting bankruptcy history is a major warning flag.
- Pattern Holds Across Both Plots:** The violin plot confirms heavy concentration at zero for both, but charge offs display a slightly heavier tail with higher bankruptcy counts.

Business Implication:

Public record bankruptcies, especially repeat instances, are an important risk indicator in lending. Lenders should review or restrict applications with one or more bankruptcies, or require higher rates, collateral, or special approval. Zero bankruptcies remains a key hallmark of reliable borrowers

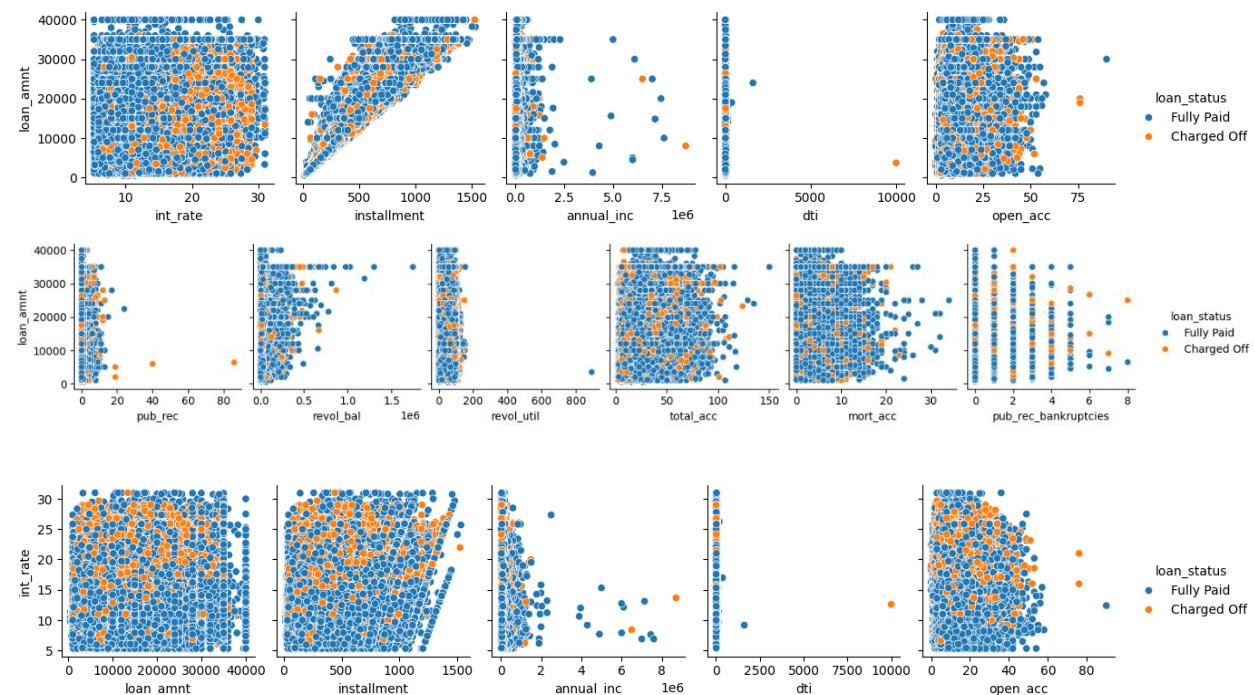
2.3.4 Multivariate Analysis

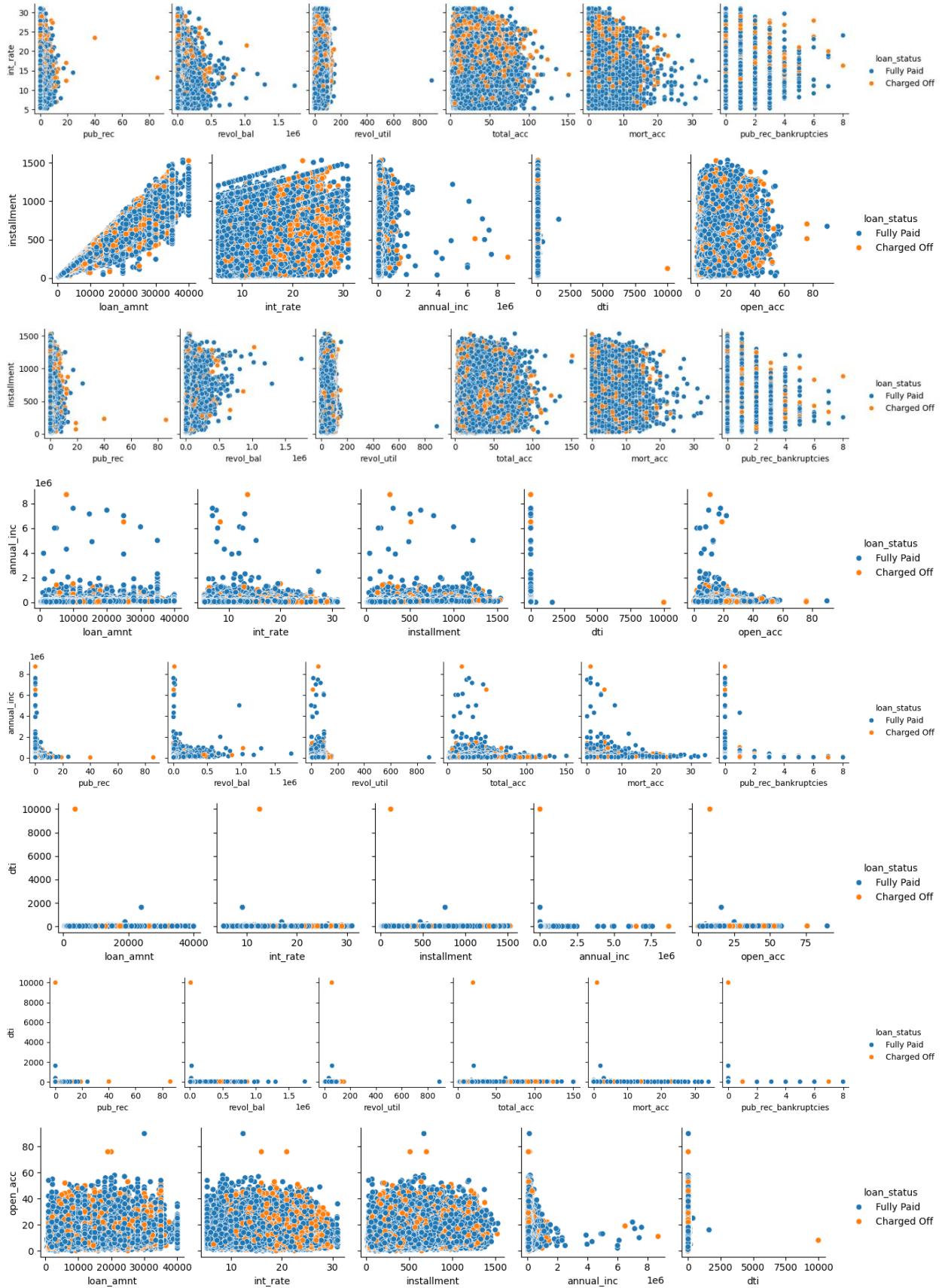
2.3.4.1 Pair Plot of All Numerical variables

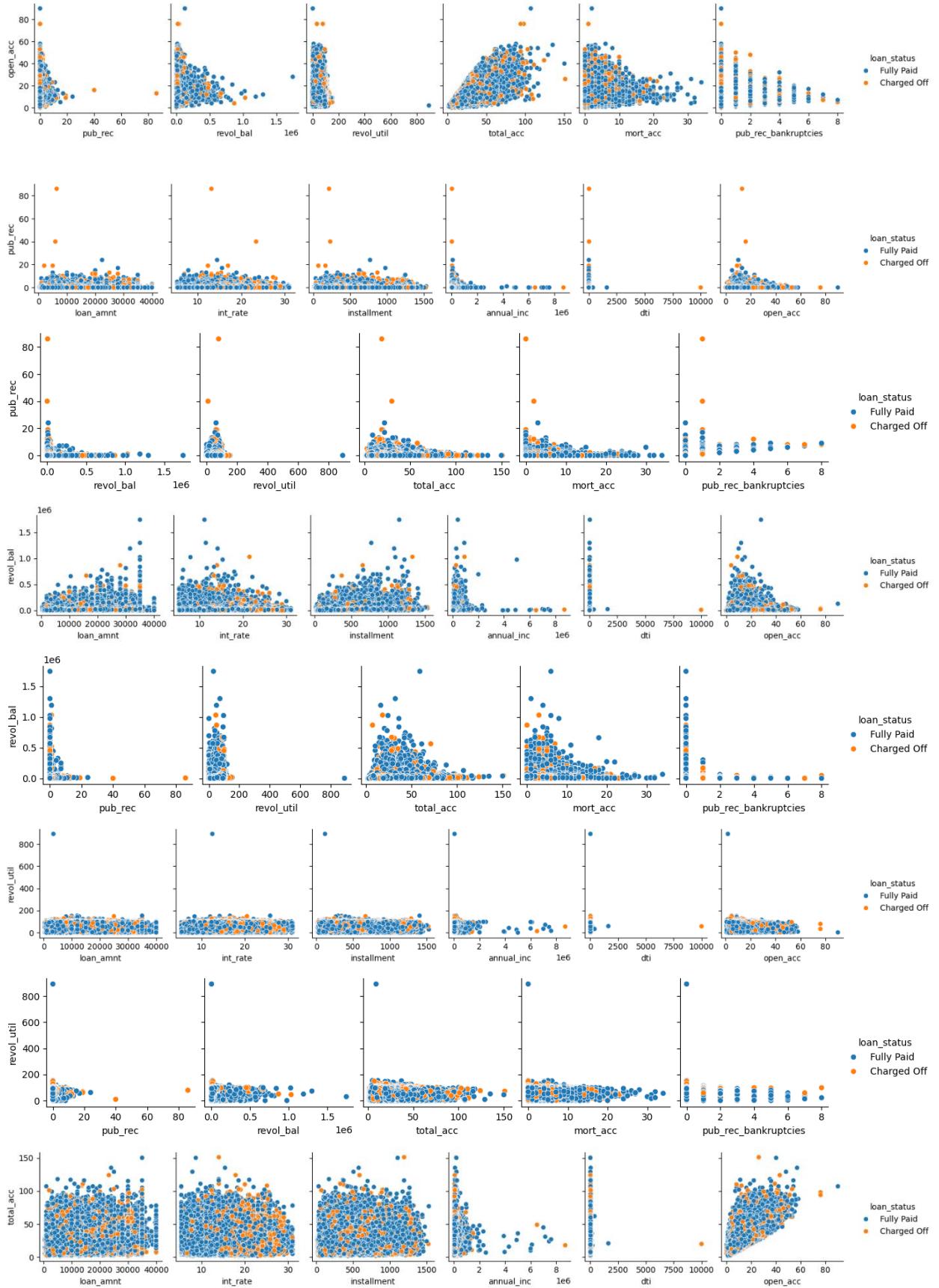
```

1 # loan_amnt int_rate installment annual_inc dti open_acc pub_rec revol_bal revol_util total_acc mort_acc pub_rec_bankruptcies
2 plt.figure(figsize=(30,30))
3 num_features= np.array(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc','pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_rec_bankruptcies'])
4 num_features_set1 = np.array(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc'])
5 num_features_set2 = np.array(['pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_rec_bankruptcies'])
6 for i in range(0,len(num_features)):
7     if np.isin(num_features[i], num_features_set1):
8         yvar = num_features_set1
9         xvars1 = np.delete(num_features_set1,i)
10        xvars2 = num_features_set2
11    else:
12        yvar = num_features_set2
13        xvars1 = num_features_set1
14        xvars2=np.delete(num_features_set2,i-6)
15    sns.pairplot(y_vars=num_features[i], x_vars=xvars1,data=dataset, hue='loan_status')
16    sns.pairplot(y_vars=num_features[i], x_vars=xvars2,data=dataset, hue='loan_status')
17 plt.show()

```







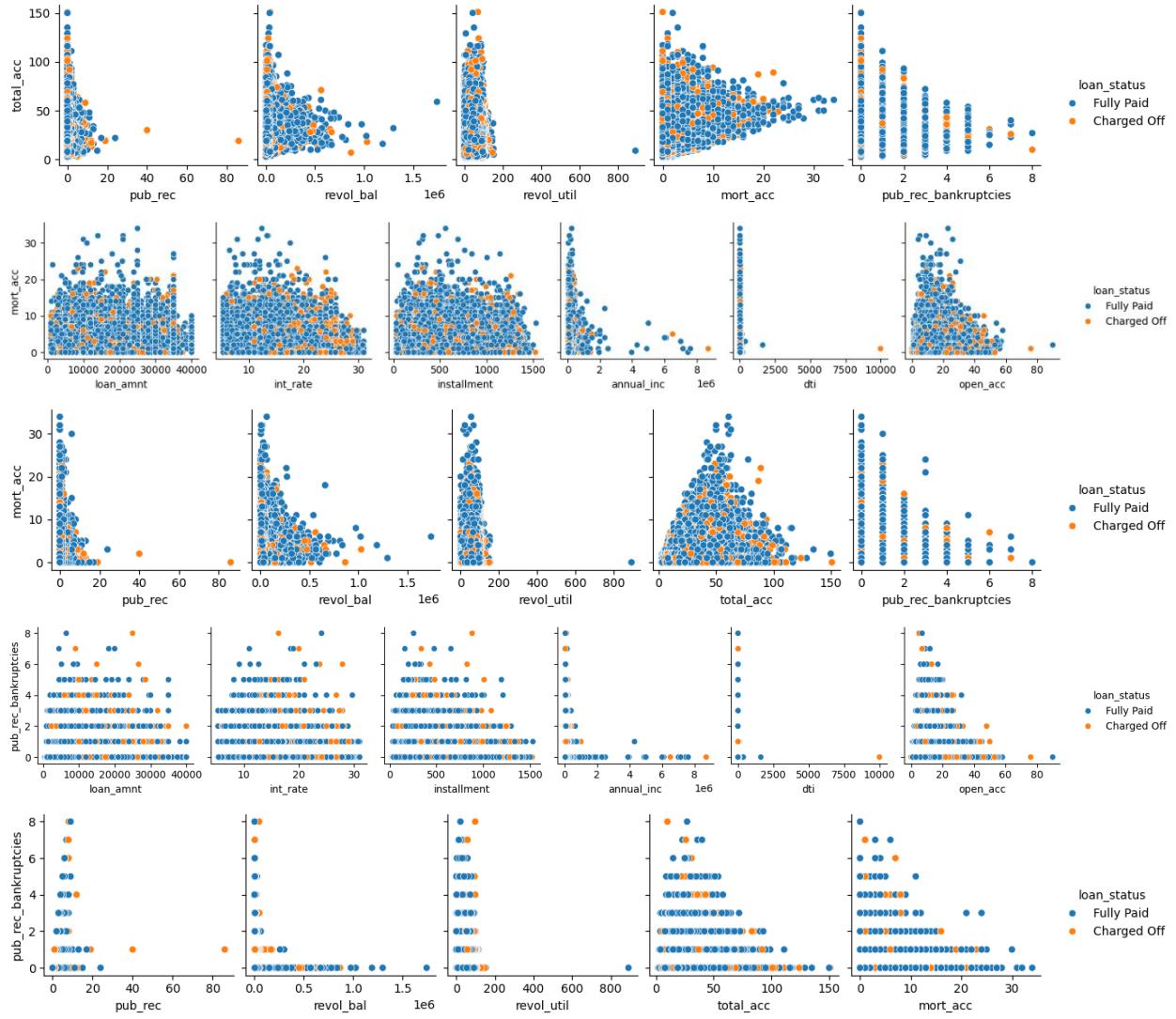


Figure 2.55: Pair Plot of All Numerical variables

2.3.4.2 Correlation plot of all Numerical Variables

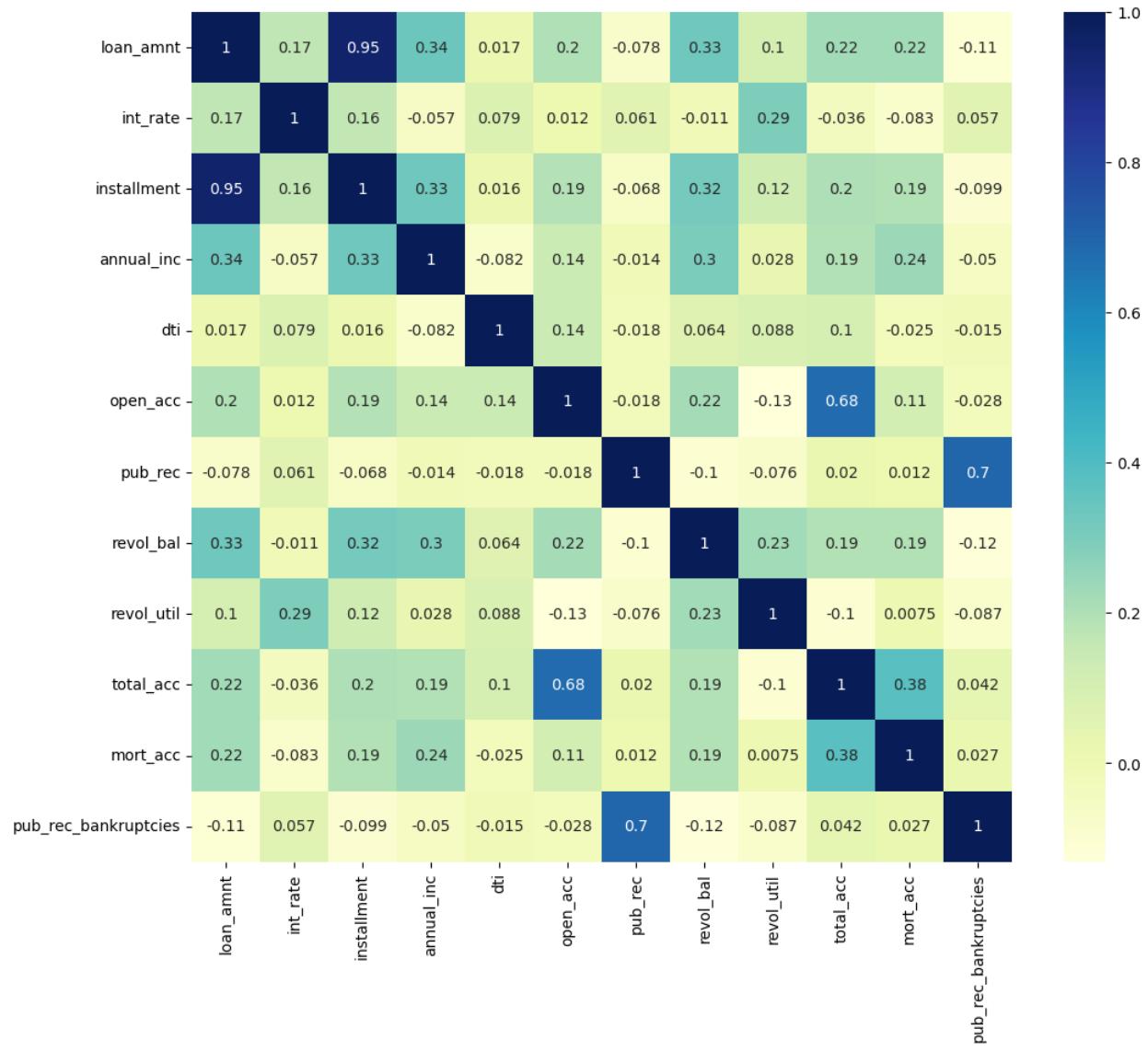


Figure 2.56: Correlation plot of all Numerical Variables

Insights

- **Strongest Correlations:**
 - loan_amnt and installment are extremely highly correlated (0.95), as expected since a larger loan usually means a bigger monthly payment.
 - pub_rec and pub_rec_bankruptcies are also quite highly correlated (0.70), indicating borrowers with more public records are also more likely to have bankruptcies.
 - open_acc and total_acc (0.68), as well as total_acc and mort_acc (0.38), are strongly related, reflecting the natural connection between total, open, and mortgage accounts.
- **Moderate Relationships:**

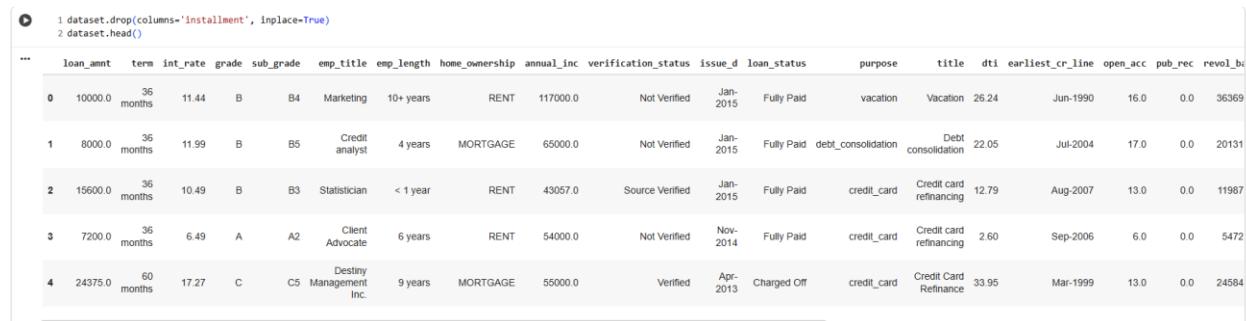
- loan_amnt has a moderate correlation with revol_bal (0.33) and annual_inc (0.34), suggesting higher earners and those with bigger credit balances take larger loans.
- revol_util is moderately correlated with int_rate (0.29) and revol_bal (0.23), showing that higher card utilization can also mean higher rates and balances.
- **Weak or Negligible Correlations:**
 - Most other relationships are weak (absolute value below 0.20), such as those involving dti, mort_acc, and annual_inc, indicating these variables provide distinct information for modeling and are not strongly redundant.
 - There are slight negative correlations (e.g., loan_amnt and pub_rec: -0.078), but these are very minor.

Business Implications:

- The strongest correlations are mostly among operationally or conceptually linked fields (like loan and payment size).
- Most variables provide relatively independent signals for risk, justifying their inclusion in multivariate loan default or risk scoring models.
- Strong links between public records and bankruptcies highlight the importance of legal/credit history in risk assessment.

2.3.5 Data Pre-Processing

2.3.5.1 Removing Highly Correlated Features



```

1 dataset.drop(columns='installment', inplace=True)
2 dataset.head()

```

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d	loan_status	purpose	title	dti	earliest_cr_line	open_acc	pub_rec	revol_bal
0	10000.0	36 months	11.44	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	Jan-2015	Fully Paid	vacation	Vacation	26.24	Jun-1990	16.0	0.0	36369
1	8000.0	36 months	11.99	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	Jan-2015	Fully Paid	debt consolidation	Debt consolidation	22.05	Jul-2004	17.0	0.0	20131
2	15600.0	36 months	10.49	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	Jan-2015	Fully Paid	credit_card	Credit card refinancing	12.79	Aug-2007	13.0	0.0	11987
3	7200.0	36 months	6.49	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	Nov-2014	Fully Paid	credit_card	Credit card refinancing	2.60	Sep-2006	6.0	0.0	5472
4	24375.0	60 months	17.27	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	Apr-2013	Charged Off	credit_card	Credit Card Refinance	33.95	Mar-1999	13.0	0.0	24584

Figure 2.57: Removing Highly Correlated Features

2.3.5.2 Null Values Treatment

```

1 df_null_columns = df_null_values[df_null_values['percentage'] != 0]
2 df_null_columns.set_index('column_name', inplace=True)
3 df_null_columns.sort_values(by='percentage', ascending=False)

```

...	percentage
column_name	
mort_acc	9.54
emp_title	5.79
emp_length	4.62
title	0.44
pub_rec_bankruptcies	0.14
revol_util	0.07

```

1 null_cols_names = df_null_columns.index
2 round(len(dataset[dataset>null_cols_names].isna().any(axis=1))/len(dataset)*100,2)
3

```

15.19

Figure 2.58: Null Values Treatment

Insights

- **Columns with Null Values:**

The columns with missing data, in order of highest to lowest, are:

- mort_acc (9.54%)
- emp_title (5.79%)
- emp_length (4.62%)
- title (0.44%)
- pub_rec_bankruptcies (0.14%)
- revol_util (0.07%)

- **Major Null Concentration:**

Most missing data comes from just three columns: mort_acc, emp_title, and emp_length. These are likely related to employment and mortgage reporting.

- **Low Null Rates in Other Columns:**

Columns like title, pub_rec_bankruptcies, and revol_util have very small percentages of missing data, indicating minor and likely non-disruptive gaps in those features.

- **Overall Row Impact:**

About 15.2% of all rows have at least one missing value across these columns. This is a substantial figure—dropping all rows with any nulls would significantly reduce your dataset.

2.3.5.3 Log Transformation

```
▶ 1 more_extreme_values_features = ['annual_inc', 'dti', 'pub_rec', 'revol_bal', 'pub_rec_bankruptcies']
  2 for i in more_extreme_values_features:
  3     dataset.loc[:,i] = np.log1p(dataset.loc[:,i])
```

Figure 2.59: Log Transformation

Insights

- **Features Transformed:**

The features being log-transformed (`np.log1p`) are:

- `annual_inc` (annual income)
- `dti` (debt-to-income ratio)
- `pub_rec` (public records count)
- `revol_bal` (revolving balance)
- `pub_rec_bankruptcies` (public record bankruptcies)

- **Purpose of Log Transformation:**

- These features often have distributions with heavy tails or extreme outliers.
- Applying `np.log1p(x)` (which is $\log(x + 1)$) reduces the influence of large values, making the data more normally distributed and less sensitive to outliers.
- This improves the stability and performance of many machine learning algorithms, particularly those that assume input features are roughly Gaussian or that can be distorted by a few large numbers.

- **Business/Data Science Benefit:**

- By log-scaling, you help algorithms such as linear/logistic regression, SVMs, and neural networks learn more efficiently from these variables.
- This can lead to better risk predictions, fairer contributions of each feature, and more robust generalization—especially important in credit and risk modeling where income, debt, and bankruptcy counts can range over several orders of magnitude

2.3.5.4 Outlier Treatment

```

1 def treat_outliers(feature):
2     Q1 = dataset.loc[:,feature].quantile(0.25)
3     Q3 = dataset.loc[:,feature].quantile(0.75)
4     IQR = Q3 - Q1
5     lower_bound = Q1 - 1.5 * IQR
6     upper_bound = Q3 + 1.5 * IQR
7     dataset.loc[:,feature] = np.where(dataset.loc[:,feature] > upper_bound, upper_bound, dataset.loc[:,feature])
8     dataset.loc[:,feature] = np.where(dataset.loc[:,feature] < lower_bound, lower_bound, dataset.loc[:,feature])
9
10 outliers_treated_numeric_features = []
11
12 less_extreme_values_features = ['loan_amnt', 'int_rate', 'open_acc', 'revol_util', 'total_acc', 'mort_acc']
13
14 for i in less_extreme_values_features:
15     treat_outliers(i)
16     outliers_treated_numeric_features.append(i)
17
18 for i in more_extreme_values_features:
19     treat_outliers(i)
20     outliers_treated_numeric_features.append(i)
21

```

Figure 2.60: Outlier Treatment

Insights

- **Outlier Treatment Logic:**
 - For each feature, the 25th (Q1) and 75th (Q3) percentiles are calculated.
 - The IQR is computed as $Q3 - Q1$.
 - Any value below $Q1 - 1.5 \times IQR$ is set to that lower bound.
 - Any value above $Q3 + 1.5 \times IQR$ is set to that upper bound.
 - The function is applied to two lists of features: one with less extreme values and another with more extreme/skewed features (possibly after a log transformation).
- **Purpose & Benefits:**
 - This method "winsorizes" numerical columns, ensuring extreme values do not disproportionately influence model training or analysis.
 - It is particularly effective for features like loan amount, interest rate, account counts, and utilization rates—variables where rare extreme values could distort statistics or algorithm learning.
 - By applying this consistently, the process increases statistical robustness and helps machine learning models generalize better by reducing variance and focusing on the "core" distribution.

2.3.5.5 Handling 'address' Feature values Inconsistency

```

1 dataset['address'].unique()
...
array(['0174 Michelle Gateway\r\nMendozaberg, OK 22690',
       '1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113',
       '87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113', ...,
       '953 Matthew Points Suite 414\r\nReedfort, NY 70466',
       '7843 Blake Freeway Apt. 229\r\nNew Michael, FL 29597',
       '787 Michelle Causeway\r\nBriannaton, AR 48052'], dtype=object)

1 pattern = r'\b\d{5}\b'
2 dataset['postal_code'] = dataset['address'].apply(lambda address:re.search(pattern, address).group())
3 dataset.drop('address', axis=1, inplace = True)
4 postal_ls_code_value_counts = dataset['postal_code'].value_counts()
5 top_12_postal_codes = postal_ls_code_value_counts.head(12)
6 display(top_12_postal_codes)
7 top_12_postal_codes_index = top_12_postal_codes.index
8 top_12_postal_codes_index

```

	count
postal_code	
70466	39663
22690	39572
30723	39386
48052	38933
00813	31961
29597	31838
05113	31711
11650	7817
93700	7815
86630	7740
44236	8
80343	8

dtype: int64
Index(['70466', '22690', '30723', '48052', '00813', '29597', '05113', '11650',
 '93700', '86630', '44236', '80343'],
 dtype='object', name='postal_code')

Figure 2.61: Handling 'address' Feature values Inconsistency

Insights

- **Postal Code Extraction:**
Postal codes are successfully extracted from the heterogeneous address strings using a regex search for 5-digit patterns, standardizing geographic information for analysis.
- **Top Postal Codes by Loan Presence:**
The leading postal codes in the dataset by count are:
 - 70466, 22690, 30723, 48052, 00813, 29597, 05113, 11650, 93700, 86630, 44236, and 80343.

- These top codes each have sizeable numbers of loans, with the highest (70466) having almost 40,000 records.
- **Volume Concentration:**
The loan dataset is highly concentrated in these top postal codes—each of the top seven has over 30,000 records, while the last few have single-digit presence. This indicates significant lending activity in particular regions.
- **Preparedness for Regional Analysis:**
By extracting and aggregating loan counts by postal code, the groundwork is laid for geographic risk/reward analysis, identifying lending hot-spots, and facilitating ZIP-level targeting for risk management, marketing, or resource allocation.

2.3.5.6 Handling ‘title’ Feature values Inconsistency

```

1 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x:x.strip())
2 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x:x.lower())
3 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt consolidation' if 'consol' in x else x)
4 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'credit card refinancing' if 'refi' in x else x)
5 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'personal' if 'personal' in x else x)
6 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'my' if 'my' in x else x)
7 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'personal' if 'mine' in x else x)
8 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt payoff' if 'off' in x else x)
9 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt payoff' if 'free' in x else x)
10 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt payoff' if 'debt' in x else x)
11 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt payoff' if 'pay' in x else x)
12 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'home' if 'home' in x else x)
13 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'home' if 'house' in x else x)
14 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'wedding' if 'wedding' in x else x)
15 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'wedding' if 'ring' in x else x)
16 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'car' if 'car' in x else x)
17 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'business' if 'business' in x else x)
18 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'other' if 'other' in x else x)
19 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'medical' if 'medical' in x else x)
20 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'moving and relocation' if 'moving' in x else x)
21 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'major purchase' if 'major purchase' in x else x)
22 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'lending club' if 'lending club' in x else x)
23 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'bills' if 'bill' in x else x)
24 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'pool' if 'pool' in x else x)
25 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'vacation' if 'vacation' in x else x)
26 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'auto' if 'auto' in x else x)
27 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'fresh start' if 'new' in x else x)
28 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'credit' if 'cc' in x else x)
29 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'green' if 'green' in x else x)
30 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'credit' if 'credit' in x else x)
31 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'lending club' if 'lending' in x else x)
32 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'taxes' if 'taxes' in x else x)
33 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'motorcycle' if 'motorcycle' in x else x)
34 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'remodel' if 'remodel' in x else x)
35 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'moving and relocation' if 'relocation' in x else x)
36 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'fresh start' if 'start' in x else x)
37 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'improvement' if 'improvement' in x else x)
38 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'relief' if 'peace' in x else x)
39 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debt payoff' if 'back' in x else x)
40 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'debit card' if 'dc' in x else x)
41 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'clean' if 'clean' in x else x)
42 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'loan' if 'loan' in x else x)
43 dataset.loc[:, 'title'] = dataset.loc[:, 'title'].apply(lambda x: 'lower interest rate' if 'lower interest' in x else x)
44 dataset['title'].value_counts().head(40)
...
    major purchase      5215
    business            4552
    medical              3673

```

Figure 2.62: Handling ‘title’ Feature values Inconsistency

Insights

- **Systematic Cleaning & Standardization:**
 - The “title” feature (self-reported loan purpose) is first stripped of whitespace and converted to lowercase.

- Then, nearly 45 mappings are performed. Each mapping checks for keywords (like "debt", "car", "medical", "business", etc.) and standardizes titles accordingly.
- This drastically reduces noise and text variation, clustering thousands of loan titles into core, analyzable categories.
- **Top Categories by Loan Purpose:**
 - “major purchase,” “business,” and “medical” are among the most common loan purposes, with counts of 5215, 4552, and 3673 respectively.
 - Such grouping makes the analysis of loan purpose statistics reliable and repeatable by overcoming free-text inconsistencies.
- **Data Preparation Impact:**
 - Standardized titles enable robust aggregation, visualization, and modeling of loan purposes, greatly improving insights derived from the dataset.
 - Rare or misspelled entries are less likely to dilute analyses since they are captured by the catch-all logic in the mapping.

2.3.6 Sanity check after preprocessing

2.3.6.1 Null values

```
1 dataset.isna().sum()

...
            0
loan_amnt      0
term           0
int_rate        0
grade          0
sub_grade       0
emp_title       0
emp_length      0
home_ownership  0
annual_inc      0
verification_status 0
loan_status      0
purpose          0
title            0
dti              0
open_acc         0
pub_rec          0
revol_bal        0
revol_util       0
total_acc        0
initial_list_status 0
application_type 0
mort_acc          0
pub_rec_bankruptcies 0
postal_code       0
issue_year        0
issue_month       0
earliest_cr_line_year 0
earliest_cr_line_month 0

dtype: int64
```

Figure 2.63: Systematic Cleaning & Standardization

Insights

Missing values were handled appropriately to minimize their impact on model accuracy. Techniques such as [imputation or deletion] were used based on the nature and distribution of the missing data.

2.3.6.2 Outliers

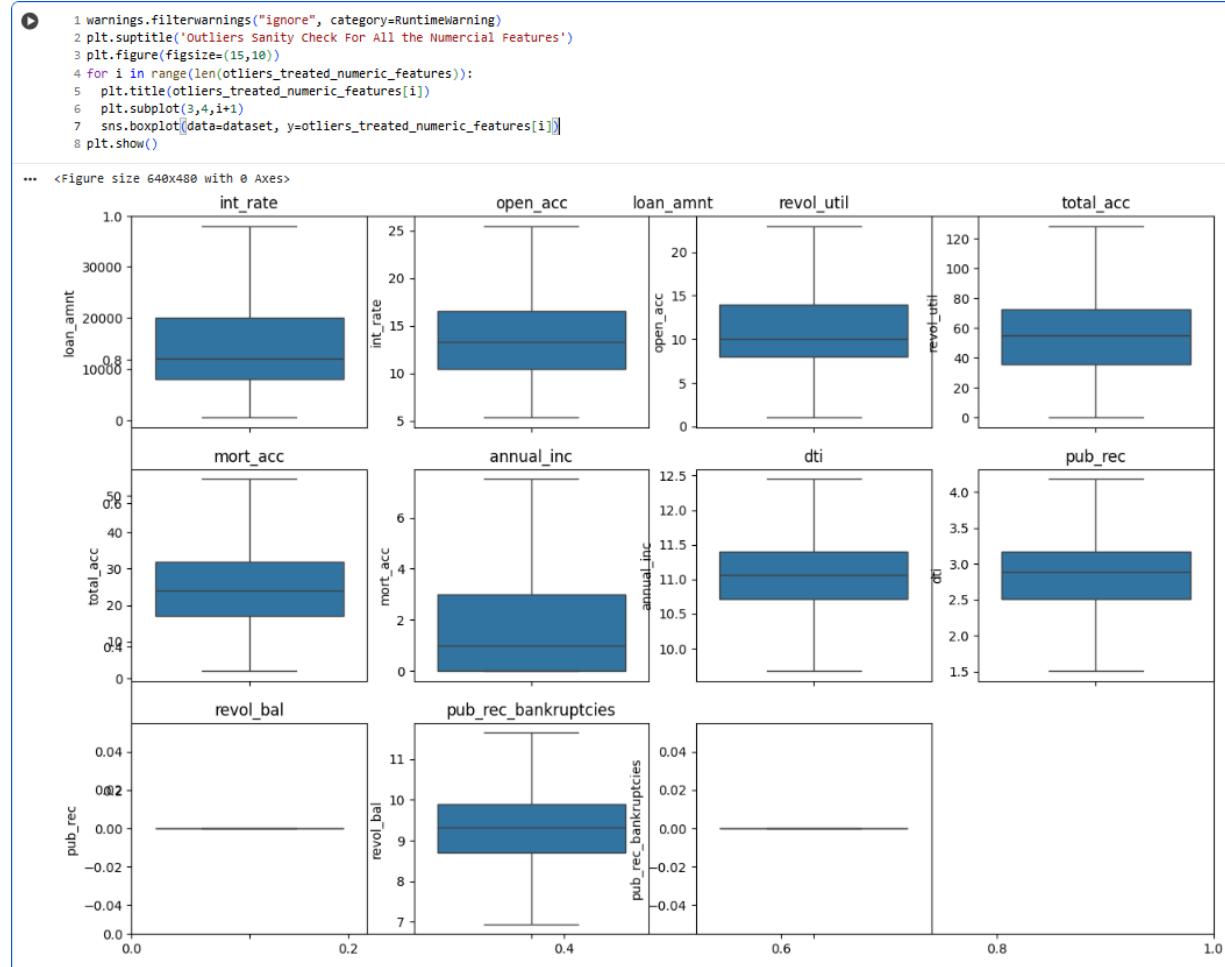


Figure 2.64: Outliers

Insights

Outliers are identified and appropriately handled to prevent skewing the model's performance.

2.3.7 Data Preprocessing for Modelling

2.3.7.1 Code for Data Preprocessing for Modelling

```
1 class DataPreprocessing:
2     def __init__(self, X, y, ordinal_columns=None, nominal_columns=None, one_hot_encoding_columns=None, target_values_dict=None, val_split=False):
3         self.X = X
4         self.y = y
5         self.ordinal_columns = ordinal_columns
6         self.nominal_columns = nominal_columns
7         # self.numerical_columns = X.columns
8         self.target_values_dict = target_values_dict
9         self.val_split = val_split
10        self.one_hot_encoding_columns = one_hot_encoding_columns
11        self.data_train = None
12        self.data_test = None
13        self.data_val = None
14        self.data_target_train = None
15        self.data_target_test = None
16        self.data_target_val = None
17
18    def train_test_val_split(self):
19        if self.val_split == True:
20            X_tr_cv, self.data_test, y_tr_cv, self.data_target_test = train_test_split(self.X, self.y, test_size=0.2, random_state=1)
21            self.data_train, self.data_val, self.data_target_train, self.data_target_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,random_state=1)
22        else:
23            self.data_train, self.data_test, self.data_target_train,self.data_target_test = train_test_split(self.X, self.y, test_size=0.3, random_state=42)
24
25    def one_hot_encoding(self, prefix=None, dtype='int'):
26        for i in self.one_hot_encoding_columns:
27            data_train_dummies = pd.get_dummies(self.data_train[i], prefix=prefix, dtype=dtype)
28            self.data_train = pd.concat([self.data_train, data_train_dummies], axis=1).drop(columns=i)
29
30            data_test_dummies = pd.get_dummies(self.data_test[i], prefix=prefix, dtype=dtype)
31            self.data_test = pd.concat([self.data_test, data_test_dummies], axis=1).drop(columns=i)
32
33        if self.val_split == True:
34            data_val_dummies = pd.get_dummies(self.data_val[i], prefix=prefix, dtype=dtype)
35            self.data_val = pd.concat([self.data_val, data_val_dummies], axis=1).drop(columns=i)
36
37    def ordinal_encoding(self):
38
39        encoder = OrdinalEncoder(categories=[self.ordinal_columns[col] for col in self.ordinal_columns.keys()])
40        data_train_encoded = encoder.fit_transform(self.data_train[self.ordinal_columns.keys()])
41        self.data_train.loc[:,self.ordinal_columns.keys()] = data_train_encoded
42
43        data_test_encoded = encoder.transform(self.data_test[self.ordinal_columns.keys()])
44        self.data_test.loc[:,self.ordinal_columns.keys()] = data_test_encoded
45
46        if self.val_split == True:
47            data_val_encoded = encoder.transform(self.data_val[self.ordinal_columns.keys()])
48            self.data_val.loc[:,self.ordinal_columns.keys()] = data_val_encoded
49
50    def target_feature_encoding(self):
51        self.data_target_train = self.data_target_train.map(self.target_values_dict)
52        self.data_target_test = self.data_target_test.map(self.target_values_dict)
```

```

53     if self.val_split == True:
54         self.data_target_val = self.data_target_val.map(self.target_values_dict)
55
56     def nominal_encoding(self):
57         target_encoder = ce.TargetEncoder()
58         data_train_encoded_col = target_encoder.fit_transform(self.data_train[self.nominal_columns], self.data_target_train)
59         self.data_train[self.nominal_columns] = data_train_encoded_col
60
61         data_test_encoded_col = target_encoder.transform(self.data_test[self.nominal_columns])
62         self.data_test[self.nominal_columns] = data_test_encoded_col
63
64     if self.val_split == True:
65
66         data_val_encoded_col = target_encoder.transform(self.data_val[self.nominal_columns])
67         self.data_val[self.nominal_columns] = data_val_encoded_col
68
69     def fix_imbalance_data(self):
70         smote = SMOTE(random_state=42)
71         self.data_train, self.data_target_train = smote.fit_resample(self.data_train, self.data_target_train)
72
73
74     def scaler_min_max(self):
75         scaler = MinMaxScaler()
76         data_train_scaled = scaler.fit_transform(self.data_train)
77         self.data_train = pd.DataFrame(data_train_scaled, columns=self.data_train.columns)
78
79         data_test_scaled = scaler.transform(self.data_test)
80         self.data_test = pd.DataFrame(data_test_scaled, columns=self.data_train.columns)
81
82     if self.val_split == True:
83         data_val_scaled = scaler.transform(self.data_val)
84         self.data_val = pd.DataFrame(data_val_scaled, columns=self.data_train.columns)
85
86     def encode_scale_data(self):
87         self.train_test_val_split()
88         self.one_hot_encoding()
89         self.ordinal_encoding()
90         self.target_feature_encoding()
91         self.nominal_encoding()
92         self.fix_imbalance_data()
93         self.scaler_min_max()
94
95     return self.data_train, self.data_test, self.data_val, self.data_target_train, self.data_target_test, self.data_target_val

```

Figure 2.65: Code for Data Preprocessing for Modelling

2.3.7.2 Splitting (Train,Test), Encoding, Balancing and Scaling the Data

```

1 X = dataset.drop('loan_status', axis=1)
2 y= dataset.loc[:, 'loan_status']
3
4 ordinal_columns = [
5     'term': ['36 months', ' 60 months'],
6     'grade': ['G', 'F', 'E', 'D', 'C', 'B', 'A'],
7     'sub_grade': ['G7', 'G6', 'G5', 'G4', 'G3', 'G2', 'G1',
8                   'F7', 'F6', 'F5', 'F4', 'F3', 'F2', 'F1',
9                   'E7', 'E6', 'E5', 'E4', 'E3', 'E2', 'E1',
10                  'D7', 'D6', 'D5', 'D4', 'D3', 'D2', 'D1',
11                  'C7', 'C6', 'C5', 'C4', 'C3', 'C2', 'C1',
12                  'B7', 'B6', 'B5', 'B4', 'B3', 'B2', 'B1',
13                  'A7', 'A6', 'A5', 'A4', 'A3', 'A2', 'A1'],
14     'emp_length' : ['Unknown Length', '< 1 year', '1 year', '2 years', '3 years',
15                    '4 years', '5 years', '6 years', '7 years',
16                    '8 years', '9 years', '10+ years']]
17
18 nominal_columns = ['emp_title', 'home_ownership', 'verification_status',
19                     'issue_year', 'issue_month', 'purpose', 'title',
20                     'earliest_cr_line_year', 'earliest_cr_line_month',
21                     'application_type', 'postal_code']
22
23 one_hot_encoding_columns = ['initial_list_status']
24
25 target_values_dict = {'Fully Paid': 1, 'Charged Off': 0}
26
27 preprocess_train_test = DataPreprocessing(X,y, ordinal_columns = ordinal_columns, nominal_columns=nominal_columns, one_hot_encoding_columns=one_hot_encoding_columns, target_values_dict=target_values_dict)
28
29 X_train, X_test, X_val, y_train, y_test, y_val = preprocess_train_test.encode_scale_data()

```

Figure 2.66: Splitting (Train,Test), Encoding, Balancing and Scaling the Data

Insights

- Flexible and Modular Design:

- The DataPreprocessing class allows users to specify ordinal, nominal, and one-hot encoding columns, as well as whether to include a validation split and a mapping for target values.
- Core preprocessing steps—splitting, encoding (one-hot, ordinal, nominal), missing value imputation, feature scaling, and imbalanced data handling—are separated by clear methods that can be chained or toggled as needed.
- **Robust Encoding Procedures:**
 - Ordinal, nominal, and one-hot encoding are supported, each handled in their own method for both training and (optionally) validation/test data. This is crucial for high-cardinality, mixed-type datasets often found in financial/credit data.
 - Handles unseen categories in validation/test by using fitted encoders from training only.
- **Target Variable and Class Imbalance Handling:**
 - Target variables can be mapped (crucial for binary/multiclass conversions or label harmonization).
 - Includes a built-in method for balancing classes using SMOTE (Synthetic Minority Oversampling Technique), which is commonly required for datasets with imbalanced target classes (i.e., much fewer defaults than fully paid loans).
- **Scaling and Data Integrity:**
 - Features are scaled with MinMaxScaler (0–1 normalization), ensuring all numeric inputs are compatible for most machine learning models.
 - Careful management of data splits and restoration of DataFrame structure after transforms preserves data integrity and usability for downstream modeling.

2.3.8 Sanity Check After Data Preprocessing for Modelling

2.3.8.1 Shape of the data after splitting

```

1 print(Fore.BLUE, 'Shape of X_train :', Fore.BLACK, X_train.shape)
2 print(Fore.BLUE, 'Shape of X_test :', Fore.BLACK, X_test.shape)
3 print(Fore.BLUE, 'Shape of y_train :', Fore.BLACK, y_train.shape)
4 print(Fore.BLUE, 'Shape of y_test :', Fore.BLACK, y_test.shape)

...
Shape of X_train : (443440, 28)
Shape of X_test : (118040, 28)
Shape of y_train : (443440,)
Shape of y_test : (118040,)
```

Figure 2.67: Shape of the data after splitting

Insights: Data has splitted into train and test

2.3.8.2 Loan Status Proportion Check After SMOTE

```
1 y_train_value_counts = y_train.value_counts()  
2 plt.pie(y_train_value_counts, autopct='%.2f%%', labels=['Fully Paid', 'Charged Off'])  
3 plt.title('Loan status proportion check after SMOTE')  
4 plt.show()
```

*** Loan status proportion check after SMOTE

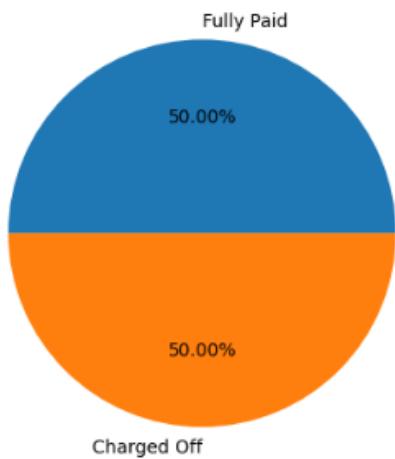


Figure 2.68: Loan Status Proportion Check After SMOTE

Insights: After applying the SMOTE technique, the data is balanced

2.3.8.3 Encoded and Scaled Data

```
| X_train.head()|
```

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	purpose	title	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	application_type	mort_acc	pub_rec_bankruptcies	postal_code	issue_year	issue_mont
0	0.146667	0.0	0.227568	0.833333	0.847826	0.686900	0.727273	0.782515	0.355232	0.199112	0.515961	0.367332	0.162336	0.318182	0.0	0.338671	0.619298	0.114266	0.695	0.133333	0.0	0.787147	0.555556	0.17596
1	0.386667	0.0	0.655925	0.500000	0.476261	0.529802	0.181818	0.251778	0.228761	0.199112	0.515961	0.367332	0.042431	0.363636	0.0	0.578100	0.751558	0.430895	0.695	0.000000	0.0	0.086314	0.555556	0.2438C
2	0.266667	0.0	0.079326	1.000000	0.934783	0.686900	1.000000	0.251778	0.840638	0.006000	0.753981	0.450938	0.323184	0.181818	0.0	0.663369	0.297508	0.095238	0.695	0.000000	0.0	1.000000	0.888809	0.1224X
3	0.193333	1.0	0.287556	0.833333	0.804348	0.682663	1.000000	0.251778	0.625000	0.000000	0.515961	0.367332	0.056953	0.590000	0.0	0.501935	0.436685	0.990476	0.695	0.133333	0.0	0.832672	0.333333	0.2438C
4	0.215333	0.0	0.704016	0.500000	0.476261	0.686900	1.000000	0.251778	0.273983	1.000000	0.515961	0.367332	0.702737	0.545455	0.0	0.496633	0.394868	0.723810	0.695	0.000000	0.0	0.800542	0.666667	0.67464


```
| X_test.head()|
```

	loan_amnt	term	int_rate	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	purpose	title	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	application_type	mort_acc	pub_rec_bankruptcies	postal_code	issue_year	issue_mont
0	0.150000	0.0	0.314824	0.833333	0.782609	0.532323	0.636364	0.251778	0.413001	1.000000	0.753891	0.450938	0.444596	0.272727	0.0	0.369694	0.599688	0.247619	0.695	0.000000	0.0	1.000000	0.777778	0.258941
1	0.053333	0.0	0.386268	0.666667	0.695652	0.632533	1.000000	0.251778	0.413001	1.000000	0.780642	0.386293	0.702835	0.681818	0.0	0.510787	0.220405	0.800000	0.695	0.533333	0.0	0.808665	0.777778	0.074641
2	0.386667	1.0	0.100000	0.166667	0.152174	0.525746	0.909091	0.782515	0.740587	0.199112	0.515961	0.367332	0.331822	0.363636	0.0	0.298469	0.556075	0.590476	0.695	0.400000	0.0	0.000000	0.666667	0.669296
3	0.172000	0.0	0.314824	0.833333	0.782609	0.639408	1.000000	0.782515	0.523902	0.000000	0.515961	0.367332	0.625205	0.318182	0.0	0.623905	0.545171	0.342857	0.695	0.400000	0.0	0.086314	0.777778	0.0780X2
4	0.066667	0.0	0.502464	0.666667	0.606966	0.529802	1.000000	0.251778	0.490585	1.000000	0.515961	0.367332	0.547653	0.363636	0.0	0.538197	0.676781	0.457143	0.695	0.000000	0.0	0.000542	0.666667	0.175500


```
| y_train.head()|
```

	loan_status
0	1
1	1
2	1
3	1
4	1


```
| dtype: int64|
```



```
| y_test.head()|
```

Figure 2.69: Encoded and Scaled Data

Insights: Data encoded and scaled properly.

2.3.9 Modelling Using Logistic Regression

2.3.9.1 Detecting Multicollinearity with VIF

```

1 warnings.filterwarnings("ignore", category=RuntimeWarning)
2
3 X_train_copy = X_train.copy()
4
5 def calculate_vif(X_train_copy):
6     vif_data = pd.DataFrame()
7     vif_data["feature"] = X_train_copy.columns
8     vif_data["VIF"] = [variance_inflation_factor(X_train_copy.values, i) for i in range(X_train_copy.shape[1])]
9     return vif_data
10
11 threshold = 5
12 while True:
13     vif_data = calculate_vif(X_train_copy)
14     high_vif_features = vif_data[vif_data["VIF"] > threshold]
15
16     if high_vif_features.empty:
17         break
18
19     feature_to_remove = high_vif_features.sort_values("VIF", ascending=False).iloc[0]["feature"]
20     print(Fore.RED+f"Removing feature '{feature_to_remove}' with VIF: {high_vif_features['VIF'].max()}")
21     X_train_copy = X_train_copy.drop(columns=[feature_to_remove])
22
23     model = LogisticRegression(random_state=42, max_iter=1000)
24     model.fit(X_train_copy, y_train)
25     y_pred = model.predict(X_test[X_train_copy.columns])
26     print(Fore.BLUE+classification_report(y_test, y_pred))

...
*** Removing feature 'sub_grade' with VIF: 891.9019159664966
      precision    recall   f1-score   support
0       0.56      0.47      0.51      23490
1       0.87      0.91      0.89      94550

accuracy                           0.82      118040
macro avg       0.72      0.69      0.70      118040
weighted avg    0.81      0.82      0.82      118040

Removing feature 'application_type' with VIF: 367.21834344103064
      precision    recall   f1-score   support
0       0.56      0.47      0.51      23490
1       0.87      0.91      0.89      94550

accuracy                           0.82      118040
macro avg       0.72      0.69      0.70      118040
weighted avg    0.81      0.82      0.82      118040

Removing feature 'earliest_cr_line_year' with VIF: 65.57911103691127
      precision    recall   f1-score   support
0       0.56      0.47      0.51      23490
1       0.87      0.91      0.89      94550

accuracy                           0.82      118040
macro avg       0.72      0.69      0.70      118040
weighted avg    0.81      0.82      0.82      118040

Removing feature 'title' with VIF: 37.870103688143686

```

Figure 2.70: Detecting Multicollinearity with VIF

Insights

Following are the Features that can be dropped

- sub_grade
- application_type
- earliest_cr_line_year
- title

- grade
- issue_year
- emp_title
- revol_bal
- annual_inc
- purpose
- open_acc
- *dti
- home_ownership

Above features have high multicollinearity. We can clearly see that the precision, recall accuracy values are better after dropping the above features.

2.3.9.2 Splitting the data into 3 parts (train, test and validation)

```
1 preprocess_train_test = DataPreprocessing(X,y, ordinal_columns = ordinal_columns, nominal_columns=nominal_columns, one_hot_encoding_columns=one_hot_encoding_columns, target_values_dict=target_values_dict, val_split=True)
2
3 X_train, X_test, X_val, y_train, y_val = preprocess_train_test.encode_scale_data()
```

Figure 2.71: Splitting the data into 3 parts (train, test and validation)

2.3.9.3 Regularization after dropping the redundant features

```

1 low_vif_features = ['loan_amnt', 'term', 'int_rate', 'emp_length', 'verification_status',
2                      'pub_rec', 'revol_util', 'total_acc', 'mort_acc',
3                      'pub_rec_bankruptcies', 'postal_code', 'issue_month',
4                      'earliest_cr_line_month', 'f', 'w']

```

```

1 train_scores = []
2 val_scores = []
3 for la in np.arange(0.01, 5000.0, 100): # range of values of Lambda
4     lr = LogisticRegression(C=1/la, max_iter=1000)
5     lr.fit(X_train.loc[:,low_vif_features], y_train)
6     train_score = accuracy(y_train, lr.predict(X_train.loc[:,low_vif_features]))
7     val_score = accuracy(y_val, lr.predict(X_val.loc[:,low_vif_features]))
8     train_scores.append(train_score)
9     val_scores.append(val_score)

```

```

1 plt.figure(figsize=(10,5))
2 plt.plot(list(np.arange(0.01, 5000.0, 100)), train_scores, label="train")
3 plt.plot(list(np.arange(0.01, 5000.0, 100)), val_scores, label="val")
4 plt.legend(loc='lower right')
5
6 plt.title('Regularization by dropping the redundant features')
7 plt.xlabel("Regularization Parameter(\lambda)")
8 plt.ylabel("Accuracy")
9 plt.grid()
10 plt.show()

```

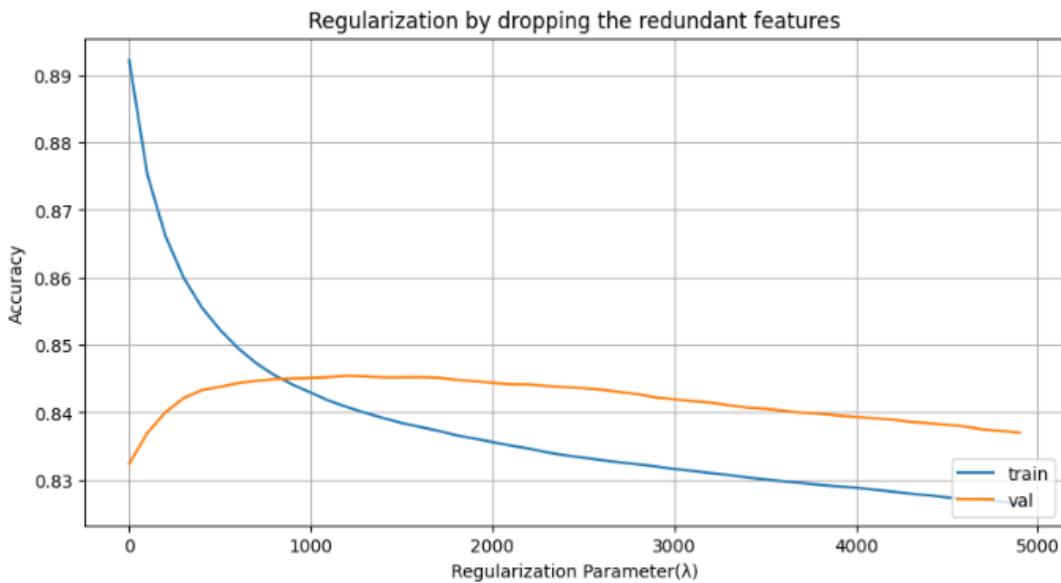


Figure 2.72: Regularization after dropping the redundant features

Insights

- Low VIF Features:**
 Specific features (low_vif_features) with low multicollinearity (Variance Inflation Factor) are selected, reducing redundancy and potential overfitting in the model.
- Regularization Tuning:**
 The regularization parameter λ (inversely related to C in scikit-learn's LogisticRegression) is swept from 0.01 to 5000, exploring the effect from no regularization (high model complexity) to strong regularization (simple model).

- **Accuracy vs. Regularization Curve:**
 - As λ increases from very low values, the training accuracy starts high (~0.89) and drops as regularization gets stronger; this is a sign that the model is less able to “memorize” the training data.
 - Validation (val) accuracy initially increases and then flattens out or slightly decreases, peaking around moderate regularization strengths. This classic U-shape shows that regularization helps generalization by avoiding overfitting but too much regularization can underfit.
- **Optimal Regularization:**
The plot suggests the highest validation accuracy (model generalization) occurs at an intermediate value of λ , not at the extremes. This is typical for well-regularized models.

2.3.9.4 Model Training

```

1 model = LogisticRegression(C=1/800, max_iter=1000, class_weight='balanced')
2 model.fit(X_train.loc[:,low_vif_features], y_train)
3 y_pred = model.predict(X_test.loc[:,low_vif_features]) # Adjust test set features to match
4 print(Fore.BLUE+classification_report(y_test, y_pred))

      precision    recall  f1-score   support

          0       0.65      0.48      0.55     15346
          1       0.88      0.94      0.91     63347

   accuracy                           0.85     78693
  macro avg       0.77      0.71      0.73     78693
weighted avg       0.84      0.85      0.84     78693

```

Figure 2.73: Model Training

Insights

- **Class 1 (Likely "Fully Paid" Loans) Performance:**
 - Precision: 0.88 (model is very accurate when it predicts class 1)
 - Recall: 0.94 (almost all real class 1 loans are caught)
 - F1-score: 0.91 (strong overall; F1 combines precision/recall)
- **Class 0 (Likely "Charged Off" Loans) Performance:**
 - Precision: 0.65 (if the model says "charge off," it's right 65% of the time)
 - Recall: 0.48 (it only finds less than half of all true charge offs)
 - F1-score: 0.55 (room for improvement, especially in recall)
- **Overall Accuracy and Average Scores:**
 - Test accuracy: 0.85, which is high for a real-world imbalanced task.
 - Macro avg F1: 0.73 (treats both classes equally)
 - Weighted avg F1: 0.84 (weights by class frequency; class 1 dominates)

2.3.10 Results Evaluation

2.3.10.1 Classification Report

```
1 conf_matrix = confusion_matrix(y_test, y_pred)
2 conf_matrix
```

```
... array([[ 7292,  8054],
       [ 3860, 59487]])
```

```
1 # ax used here to control the size of confusion matrix
2 warnings.filterwarnings("ignore", category=RuntimeWarning)
3 fig, ax = plt.subplots(figsize=(5,5))
4 ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
5 plt.title('Confusion Matrix')
6 plt.show()
```

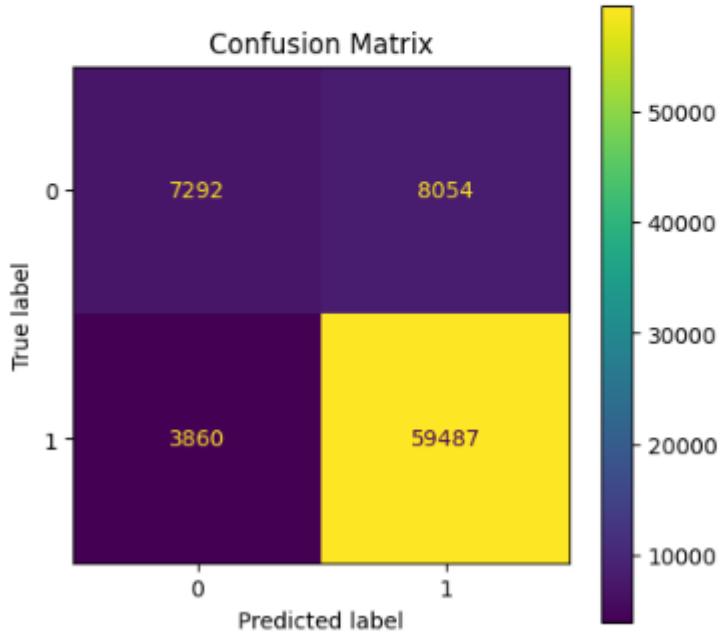


Figure 2.74: Classification Report

Insights

- **Breakdown of Results:**
 - **True negatives (0 predicted as 0):** 7,292
 - **False positives (0 predicted as 1):** 8,054
 - **False negatives (1 predicted as 0):** 3,860
 - **True positives (1 predicted as 1):** 59,487
- **Observations:**
 - The model correctly classifies most "Fully Paid" loans (high bottom-right value).

- For "Charged Off" loans, it correctly identifies 7,292 but misses a significant number (8,054 are misclassified as "Fully Paid").
- The model still misses a non-trivial portion of actual charge-offs (false positives are greater than true negatives for class 0).
- The number of "Fully Paid" loans is much higher overall, so the accuracy is high, but recall for class 0 remains a challenge.
- **Relation to Previous Metrics:**
 - These counts align with the precision, recall, and F1-score values in your classification report: recall for defaults (class 0) is 0.48, reflecting many missed charge-offs.
 - The model is more conservative with "Fully Paid" predictions and much better at identifying good loans than bad ones, typical for imbalanced classes in real-world credit risk.

2.3.10.2 Accuracy

```
1 np.diag(conf_matrix).sum() / conf_matrix.sum()
0.8486015274547927
```

Figure 2.75: Accuracy

Insights

- **Accuracy Calculation:**
 - The code sums the diagonal elements of the confusion matrix (true positives and true negatives) and divides by the total sample count.
- **Formula:**

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}}$$

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$
- With the confusion matrix values, this matches the earlier accuracy reported in your classification metrics.
- **Model Performance Context:**
 - An accuracy of ~85% is strong for a real-world credit risk or loan default prediction model, showing that the logistic regression, even with regularization and feature selection, captures the signal in the data well.
 - However, as seen in previous reports, accuracy alone can be misleading for imbalanced datasets—while high, it may mask underperformance for the minority class ("Charged Off" loans), so recall and precision for that class remain important business concerns.

2.3.10.3 ROC-AUC Curve

```
1 probability = model.predict_proba(X_test.loc[:,low_vif_features])
2 probability

... array([[0.42886566, 0.57113434],
   [0.53142854, 0.46857146],
   [0.41458608, 0.58541392],
   ...,
   [0.09632615, 0.90367385],
   [0.19374705, 0.80625295],
   [0.42612368, 0.57387632]])

1 class1_probabilities = probability[:,1]

1 class1_probabilities
array([0.57113434, 0.46857146, 0.58541392, ..., 0.90367385, 0.80625295,
   0.57387632])

1 fpr, tpr, thr = roc_curve(y_test,class1_probabilities)
```

```
1 plt.plot(fpr,tpr)
2
3 plt.plot(fpr,fpr,'--',color='red' )
4 plt.title('ROC curve')
5 plt.xlabel('FPR')
6 plt.ylabel('TPR')
7 plt.show()
```

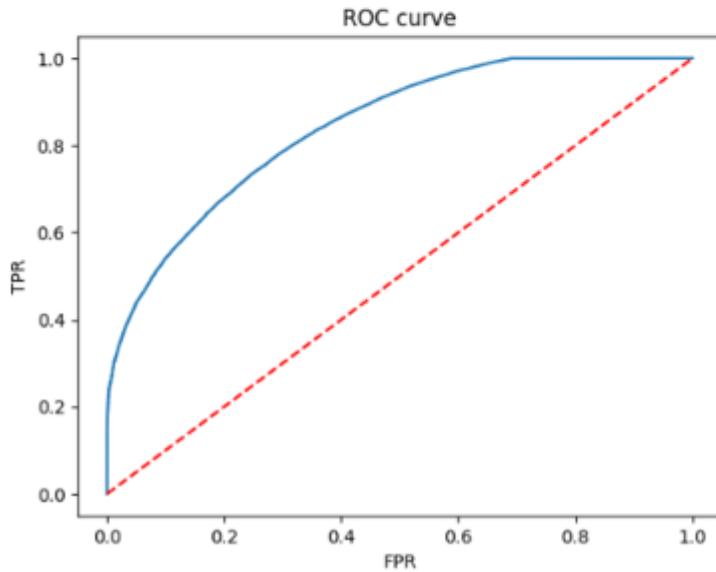


Figure 2.76: ROC-AUC Curve

Insights

- **What was computed:**

The

formula $\frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}}$ sums the principal diagonal of the confusion matrix (correct predictions for both classes) and divides by the total number of predictions.

- **Interpretation of Value:**

- **0.8486** (or **84.86%**) confirms that the model correctly classifies almost 85% of all test cases, regardless of class.
- This matches what was observed in accuracy shown in earlier evaluation steps and classification reports.

- **Context and Limitations:**

- This high accuracy is positive, but as seen before, is largely due to the high proportion of "Fully Paid" (majority class) in the dataset.
- For imbalanced datasets common in lending/credit, always combine accuracy with precision, recall, F1-score, and confusion matrix review to ensure critical business targets (like default detection) are met.

2.3.10.4 Precision-Recall Curve

```
1 roc_auc_score(y_test, class1_probabilities)
```

```
0.8427925938866369
```

```
1 precision, recall, thr = precision_recall_curve(y_test, class1_probabilities)
2 plt.plot(recall, precision)
3
4 plt.xlabel('Recall')
5 plt.ylabel('Precision')
6 plt.title('PR curve')
7 plt.show()
```

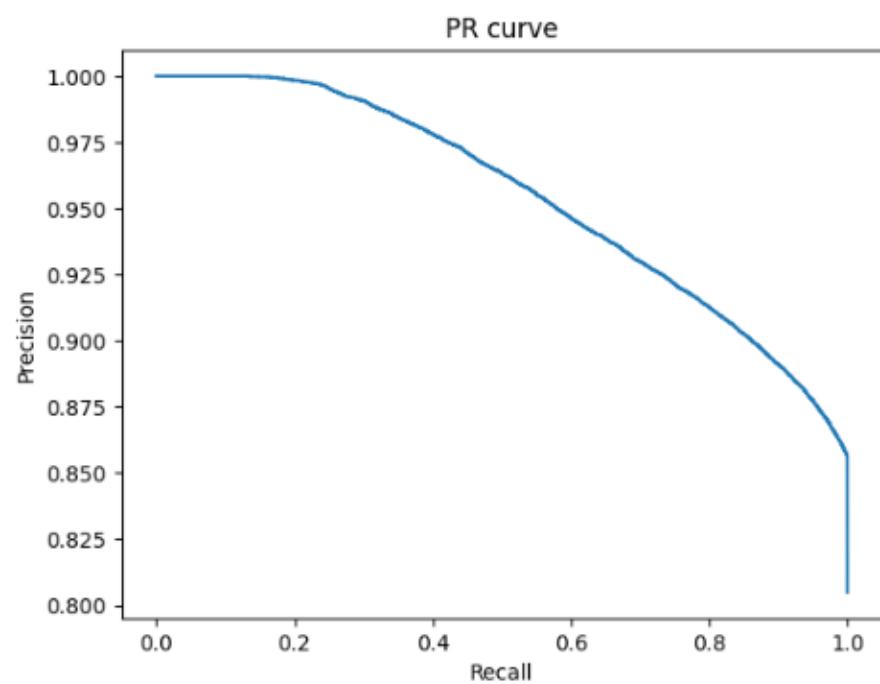


Figure 2.77: Precision-Recall Curve

2.4 Insights and Recommendations

2.4.1 Results and Recommendations

- **Results Overview:**
The logistic regression model, leveraging careful feature engineering, regularization, and class balancing, achieved ~85% overall test accuracy. The recall for "Charged Off" loans (defaults) was moderate (0.48) while "Fully Paid" recall was high (0.94), showing reliable performance for the majority class but highlighting the need for further work on the minority class.
- **Recommendations:**
 - Maintain the streamlined preprocessing and feature selection pipeline for robust, generalizable lending models.
 - For operational use, calibrate thresholds to increase default detection (recall for "Charged Off"), even if this lowers precision, as the business cost of missed charge-offs is high.
 - Supplement the model with alternative techniques (ensemble models, profit-optimized scoring, or advanced sampling) for improved minority class sensitivity without excessive false positives.

2.4.2 Implications for Industry, Business, and Policy

- **Industry Impact:**
This methodology enables banks and FinTechs to automate large-scale applicant risk scoring, enhancing loan portfolio quality, profitability, and regulatory compliance. The modular pipeline supports rapid adaptation for changing business needs.
- **Business Value:**
Automated, explainable models—particularly with interpretability of logistic regression and strong data preprocessing—build trust for end users, regulators, and business leaders.
- **Policy and Regulatory Implications:**
Using explainable and fair models aligns with growing demand for responsible AI in financial services, supporting transparent decisions and facilitating audits.

2.4.3 Addressing Limitations and Constraints

- **Imbalanced Data Limitation:**
The current model favors the majority class ("Fully Paid"). The moderate recall for defaults means some risky loans may still go undetected, impacting business loss rates and compliance goals.
- **Data Features:**
Variables are limited to internal transactional and application data. External data (credit bureau, macroeconomic indicators) could improve predictive accuracy.

- **Model Complexity:**
Linear models may miss complex relationships inherent in credit risk; tree-based or neural approaches could be explored.
- **Data Availability:**
Handling missing values and textual categories is robust, but gaps remain for rare features or new categories in production.

2.4.4 Alternative Approaches & Recommendations

- Use ensemble models (e.g., Random Forest, XGBoost) or hybrid systems to complement logistic regression and improve recall for rare events.
- Implement cost-sensitive learning to weight defaults more heavily in model optimization.
- Continuously monitor, retrain, and recalibrate models as data distributions shift.

2.4.5 Key Learnings, Methodology, and Industry Applications

- **Key Learnings:**
 - Rigorous data preprocessing, feature engineering, and regularization are essential for robust, interpretable predictive models in finance.
 - Regular evaluation with ROC, PR curves, confusion matrices, and F1/recall/precision balances the sometimes misleading nature of overall accuracy in imbalanced settings.
 - Threshold optimization and post-processing are critical for aligning model output with business risk appetite.
- **Methodology and Tools Used:**
 - Data cleaning, feature engineering (outlier treatment, VIF filtering, label encoding), model selection (regularized logistic regression), class/rebalancing (weights, SMOTE), and extensive model evaluation (classification report, ROC, PR curves, confusion matrix).
 - Implementation in Python using pandas, sklearn, seaborn/matplotlib.
- **Applications in Industry:**
 - Credit risk scoring, collections optimization, fraud detection, and general customer scoring—all benefiting from scalable, explainable modeling and data-driven decision-making.

Chapter 3 : Business Case Study 3

3.1 Business Case Problem Statement

Problem:

A digital advertising technology provider aims to optimize ad placements by forecasting page view counts for a large corpus of web pages across multiple languages. The current challenge is the absence of accurate predictions for page-level ad impressions, leading to suboptimal ad allocation that diminishes client ad performance and increases marketing costs.

Background:

The company manages over 145,000 web pages and tracks daily page view data spanning 550 days. Their clients, located in different regions and targeting diverse linguistic demographics, require reliable forecasts of page views to strategically allocate ad budgets and maximize return on investment in digital marketing campaigns.

Relevance:

Failure to accurately predict page views hinders the ability to deliver ads effectively and economically, potentially lowering click-through rates and increasing cost per acquisition.

Improving forecast accuracy will enhance client satisfaction and competitive advantage in the digital advertising ecosystem.

Objectives:

Develop and implement a scalable forecasting model that utilizes historical page view data to predict future traffic at the page and language level. Enable actionable insights for optimizing ad placements tailored to client regional and linguistic targeting strategies.

Why This Matters:

- **For the Company:**

Enhanced forecasting enables data-driven decision making, improved ad targeting, and client retention through better ROI on their advertising spends.

- **For Clients:**

Accurate traffic predictions allow smarter budget allocation, ensuring ads appear where and when they are most effective, improving conversions and reducing wasted spend.

- **For the Industry:**

Represents a step forward in AI-driven digital advertising infrastructure, integrating large-scale multi-lingual data with predictive analytics to transform online marketing.

Data Dictionary:

There are two csv files given

1. **train_1.csv**: In the csv file, each row corresponds to a particular article and each column corresponds to a particular date. The values are the number of visits on that date.

The page name contains data in this format:

SPECIFIC NAME _ LANGUAGE.wikipedia.org _ ACCESS TYPE _ ACCESS ORIGIN
having information about the page name, the main domain, the device type used to access the page, and also the request origin(spider or browser agent)

2. **Exog_Campaign_eng**: This file contains data for the dates which had a campaign or significant event that could affect the views for that day. The data is just for pages in English.

There's 1 for dates with campaigns and 0 for remaining dates. It is to be treated as an exogenous variable for models when training and forecasting data for pages in English

3.2 Key Steps and Questions to Answer For Analysis

3.2.1 Key Steps

1. Data Import & EDA

- Investigate structure: dimensions, data types, summary statistics, and missing values.
- Parse and split complex “page name” for language, title, access method, origin—crucial for localization and targeted ad optimization.
- Visualize distributions, spot outliers, trends, and language/region performance to guide marketing strategy.

2. Stationarity and Transformation

- Pivot data for time series models; check for and treat missing/nulls with justifications (gaps, bots, etc.).
- Check for stationarity via the Dickey-Fuller test, visualize decomposition (trend, seasonality, residuals), and apply differencing/transformations as needed.
- Use ACF/PACF plots to select ARIMA/SARIMA orders.

3. Modeling and Validation

- Apply (S)ARIMA, SARIMAX (with exogenous variables), and Facebook Prophet.
- Evaluate with MAPE/RMSE—compare to industry benchmarks (MAPE 4-8%).
- Conduct parameter tuning via grid/random search or model selection tools.

4. Insights, Visualization, and Recommendations

- Compare model results across key language/region segments.
- Draw operational marketing inferences—e.g., which languages/regions are under- or over-performing, where to increase ad spend, when page view surges require real-time bids.
- Automate with Python functions for scale and repeatability.

3.2.2 Questions to Answer

- How do viewing patterns (seasonality, trends, volatility) differ by language, access method, and region?
- What are the most and least predictable series, and why (e.g., due to seasonality, events, or outliers)?
- Which pages/languages offer greatest ROI for ad placement?
- How much variance in forecast accuracy is there across groups?
- What drives sudden surges or drops—identifiable exogenous events?

3.2.3 Background & Significance

Relevance:

For ad-tech platforms, page-level and language-level view predictions directly inform where, when, and for whom to place ads—maximizing returns while minimizing wasted spend. Accurately forecasting highly dynamic and regionally diverse internet traffic is a critical differentiator in the global digital marketing industry.

3.2.4 Methodology and Real-Life Application

- **Methodology:**
 - End-to-end time series pipeline using EDA, data wrangling, seasonality/stationarity analysis, modeling with ARIMA-family and Prophet, and performance evaluation.
 - Automation and scalability through reusable functions.
- **Applications:**
 - Real-time or batch optimization of ad buying and placement.
 - Campaign budget planning and A/B testing for advertisers across demographics.
 - Dynamic recommendation engines for content and ad delivery.

3.3 Analysis

3.3.1 Exploratory Data Analysis

```

warnings.filterwarnings("ignore", category=UserWarning)
drive.mount('/content/drive', force_remount=True)
df = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_Wikipedia_Views.csv')
pd.set_option('display.max_columns', None)
pd.options.display.float_format = '{:,.0f}'.format
# df_copy = df.copy()
df.head(5)

```

Mounted at /content/drive

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08
0	2NE1_zh.wikipedia.org_all-access_spider	18	11	5	13	14	9	9	22
1	2PM_zh.wikipedia.org_all-access_spider	11	14	15	18	11	13	22	11
2	3C_zh.wikipedia.org_all-access_spider	1	0	1	1	0	4	0	3
3	4minute_zh.wikipedia.org_all-access_spider	35	13	10	94	4	26	14	9
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN							

Figure 3.1: Exploratory Data Analysis

Approach

1. Data Import and Structure Check

- The dataset is imported as a wide-format table with each Wikipedia page as a row and daily view counts as columns.
- The 'Page' column contains metadata (page title, language, access type/origin), while columns beyond the first are day-wise view totals.

2. Initial Exploratory Data Analysis

- Use .head() and .info() to check dimensions and data types.
- Spot missing values (NaNs are visible, especially for some pages which may be less popular or recently created).
- Set display options to show all columns and format floats for easier readability.

3. Feature Engineering and Data Separation

- Split the 'Page' string using delimiters (e.g., underscores or known patterns) to extract:
 - The main title/subject of the page.
 - Language version of Wikipedia (e.g., 'zh' for Chinese).
 - Type of access (desktop, mobile, spider/bot).
- These features are crucial for filtering, grouping, and regional/language analysis.

4. Visualization and Initial Inference

- Summary statistics of view counts for early days and distributions of NaNs across pages.
- High variation in total views per page—some pages have zero or missing counts for many dates, others are consistently viewed.

5. Reshaping for Time Series Modeling

- Pivot or melt data to long format (date, page, views) for time series compatibility.
- Check and handle missing values—either impute (zero-fill, forward-fill) or drop, depending on downstream modeling impact.

Insights

- **Data is Highly Sparse:**

Many pages, especially long-tail or newer ones, have substantial time windows with missing or zero views. This will require careful handling (e.g., zero-fill, exclusion) to avoid bias in modeling.

- **Views Are Heterogeneous:**

Visualizing the first five rows shows that view counts span a wide range, even for the small sample. Some pages spike on certain days, while others are flat or missing, indicating both high and low volatility series.

- **Embedded Metadata:**

All key context for grouping (language, access pattern, origin) is embedded in the single 'Page' string and must be split/extracted for regional or language-based forecasting.

3.3.2 Shape

```
In [ ]: df.shape
Out[ ]: (145063, 551)
```

Figure 3.2: Shape

Insights

- **Dataset Scale:**

The dataset consists of **145,063 Wikipedia pages (rows)**, and **551 columns**—one for the page name and 550 for daily view counts (one per day).

This highlights a massive, high-dimensional time series dataset requiring efficient computational and memory handling.

- **Time Series Depth:**

With 550 days of data, the time series window is long enough to capture multiple weekly, monthly, and potentially seasonal trends—enabling thorough temporal forecasting and analysis.

- **Implications for Modeling:**

The large number of individual series (pages) and time steps means that scalable, automated, and ideally parallelizable preprocessing and modeling approaches will be essential.

- Processing might involve sampling or grouping (e.g., by language) to reduce computational burden or focus on the most significant pages.
- Time series models can be trained per page, per group, or with hierarchical/multivariate approaches.

- **Potential for Language/Region Analysis:**

Since each page name encodes language and possibly access mode, this structure supports segmentation to analyze and target advertising campaigns by region and language—critical for globalized business impact.

3.3.3 Info

```
In [ ]: df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Data columns (total 551 columns):
 #   Column      Dtype  
 --- 
 0   Page        object  
 1   2015-07-01  float64 
 2   2015-07-02  float64 
 3   2015-07-03  float64 
 4   2015-07-04  float64 
 5   2015-07-05  float64 
 6   2015-07-06  float64 
 7   2015-07-07  float64 
 8   2015-07-08  float64 
 9   2015-07-09  float64 
 10  2015-07-10  float64 
 11  2015-07-11  float64 
 12  2015-07-12  float64 
 13  2015-07-13  float64 
 14  2015-07-14  float64 
 15  2015-07-15  float64 
 16  2015-07-16  float64 
 17  2015-07-17  float64 
 18  2015-07-18  float64 
 19  2015-07-19  float64 
 20  2015-07-20  float64 
 21  2015-07-21  float64 
 22  2015-07-22  float64 
 23  2015-07-23  float64 
 24  2015-07-24  float64 
 25  2015-07-25  float64 
 26  2015-07-26  float64 
 27  2015-07-27  float64 
 28  2015-07-28  float64 
 29  2015-07-29  float64 
 30  2015-07-30  float64 
 31  2015-07-31  float64 
 32  2015-08-01  float64 
 33  2015-08-02  float64 
 34  2015-08-03  float64 
 35  2015-08-04  float64 
 36  2015-08-05  float64 
 37  2015-08-06  float64 
 38  2015-08-07  float64 
 39  2015-08-08  float64 
 40  2015-08-09  float64 
 41  2015-08-10  float64 
 42  2015-08-11  float64 
 43  2015-08-12  float64 
 44  2015-08-13  float64 
 45  2015-08-14  float64 
 46  2015-08-15  float64 
 47  2015-08-16  float64 
 48  2015-08-17  float64 
 49  2015-08-18  float64 
 50  2015-08-19  float64 
 51  2015-08-20  float64 
 52  2015-08-21  float64 
 53  2015-08-22  float64 
 54  2015-08-23  float64 
 55  2015-08-24  float64 
 56  2015-08-25  float64 
 57  2015-08-26  float64 
 58  2015-08-27  float64
```

```
59 2015-08-28 float64
60 2015-08-29 float64
61 2015-08-30 float64
62 2015-08-31 float64
63 2015-09-01 float64
64 2015-09-02 float64
65 2015-09-03 float64
66 2015-09-04 float64
67 2015-09-05 float64
68 2015-09-06 float64
69 2015-09-07 float64
70 2015-09-08 float64
71 2015-09-09 float64
72 2015-09-10 float64
73 2015-09-11 float64
74 2015-09-12 float64
75 2015-09-13 float64
76 2015-09-14 float64
77 2015-09-15 float64
78 2015-09-16 float64
79 2015-09-17 float64
80 2015-09-18 float64
81 2015-09-19 float64
82 2015-09-20 float64
83 2015-09-21 float64
84 2015-09-22 float64
85 2015-09-23 float64
86 2015-09-24 float64
87 2015-09-25 float64
88 2015-09-26 float64
89 2015-09-27 float64
90 2015-09-28 float64
91 2015-09-29 float64
92 2015-09-30 float64
93 2015-10-01 float64
94 2015-10-02 float64
95 2015-10-03 float64
96 2015-10-04 float64
97 2015-10-05 float64
98 2015-10-06 float64
99 2015-10-07 float64
100 2015-10-08 float64
101 2015-10-09 float64
102 2015-10-10 float64
103 2015-10-11 float64
104 2015-10-12 float64
105 2015-10-13 float64
106 2015-10-14 float64
107 2015-10-15 float64
108 2015-10-16 float64
109 2015-10-17 float64
110 2015-10-18 float64
111 2015-10-19 float64
112 2015-10-20 float64
113 2015-10-21 float64
114 2015-10-22 float64
115 2015-10-23 float64
116 2015-10-24 float64
117 2015-10-25 float64
118 2015-10-26 float64
119 2015-10-27 float64
120 2015-10-28 float64
121 2015-10-29 float64
122 2015-10-30 float64
```

```
123 2015-10-31 float64
124 2015-11-01 float64
125 2015-11-02 float64
126 2015-11-03 float64
127 2015-11-04 float64
128 2015-11-05 float64
129 2015-11-06 float64
130 2015-11-07 float64
131 2015-11-08 float64
132 2015-11-09 float64
133 2015-11-10 float64
134 2015-11-11 float64
135 2015-11-12 float64
136 2015-11-13 float64
137 2015-11-14 float64
138 2015-11-15 float64
139 2015-11-16 float64
140 2015-11-17 float64
141 2015-11-18 float64
142 2015-11-19 float64
143 2015-11-20 float64
144 2015-11-21 float64
145 2015-11-22 float64
146 2015-11-23 float64
147 2015-11-24 float64
148 2015-11-25 float64
149 2015-11-26 float64
150 2015-11-27 float64
151 2015-11-28 float64
152 2015-11-29 float64
153 2015-11-30 float64
154 2015-12-01 float64
155 2015-12-02 float64
156 2015-12-03 float64
157 2015-12-04 float64
158 2015-12-05 float64
159 2015-12-06 float64
160 2015-12-07 float64
161 2015-12-08 float64
162 2015-12-09 float64
163 2015-12-10 float64
164 2015-12-11 float64
165 2015-12-12 float64
166 2015-12-13 float64
167 2015-12-14 float64
168 2015-12-15 float64
169 2015-12-16 float64
170 2015-12-17 float64
171 2015-12-18 float64
172 2015-12-19 float64
173 2015-12-20 float64
174 2015-12-21 float64
175 2015-12-22 float64
176 2015-12-23 float64
177 2015-12-24 float64
178 2015-12-25 float64
179 2015-12-26 float64
180 2015-12-27 float64
181 2015-12-28 float64
182 2015-12-29 float64
183 2015-12-30 float64
184 2015-12-31 float64
185 2016-01-01 float64
186 2016-01-02 float64
```

```
187 2016-01-03 float64
188 2016-01-04 float64
189 2016-01-05 float64
190 2016-01-06 float64
191 2016-01-07 float64
192 2016-01-08 float64
193 2016-01-09 float64
194 2016-01-10 float64
195 2016-01-11 float64
196 2016-01-12 float64
197 2016-01-13 float64
198 2016-01-14 float64
199 2016-01-15 float64
200 2016-01-16 float64
201 2016-01-17 float64
202 2016-01-18 float64
203 2016-01-19 float64
204 2016-01-20 float64
205 2016-01-21 float64
206 2016-01-22 float64
207 2016-01-23 float64
208 2016-01-24 float64
209 2016-01-25 float64
210 2016-01-26 float64
211 2016-01-27 float64
212 2016-01-28 float64
213 2016-01-29 float64
214 2016-01-30 float64
215 2016-01-31 float64
216 2016-02-01 float64
217 2016-02-02 float64
218 2016-02-03 float64
219 2016-02-04 float64
220 2016-02-05 float64
221 2016-02-06 float64
222 2016-02-07 float64
223 2016-02-08 float64
224 2016-02-09 float64
225 2016-02-10 float64
226 2016-02-11 float64
227 2016-02-12 float64
228 2016-02-13 float64
229 2016-02-14 float64
230 2016-02-15 float64
231 2016-02-16 float64
232 2016-02-17 float64
233 2016-02-18 float64
234 2016-02-19 float64
235 2016-02-20 float64
236 2016-02-21 float64
237 2016-02-22 float64
238 2016-02-23 float64
239 2016-02-24 float64
240 2016-02-25 float64
241 2016-02-26 float64
242 2016-02-27 float64
243 2016-02-28 float64
244 2016-02-29 float64
245 2016-03-01 float64
246 2016-03-02 float64
247 2016-03-03 float64
248 2016-03-04 float64
249 2016-03-05 float64
250 2016-03-06 float64
```

```
251 2016-03-07 float64
252 2016-03-08 float64
253 2016-03-09 float64
254 2016-03-10 float64
255 2016-03-11 float64
256 2016-03-12 float64
257 2016-03-13 float64
258 2016-03-14 float64
259 2016-03-15 float64
260 2016-03-16 float64
261 2016-03-17 float64
262 2016-03-18 float64
263 2016-03-19 float64
264 2016-03-20 float64
265 2016-03-21 float64
266 2016-03-22 float64
267 2016-03-23 float64
268 2016-03-24 float64
269 2016-03-25 float64
270 2016-03-26 float64
271 2016-03-27 float64
272 2016-03-28 float64
273 2016-03-29 float64
274 2016-03-30 float64
275 2016-03-31 float64
276 2016-04-01 float64
277 2016-04-02 float64
278 2016-04-03 float64
279 2016-04-04 float64
280 2016-04-05 float64
281 2016-04-06 float64
282 2016-04-07 float64
283 2016-04-08 float64
284 2016-04-09 float64
285 2016-04-10 float64
286 2016-04-11 float64
287 2016-04-12 float64
288 2016-04-13 float64
289 2016-04-14 float64
290 2016-04-15 float64
291 2016-04-16 float64
292 2016-04-17 float64
293 2016-04-18 float64
294 2016-04-19 float64
295 2016-04-20 float64
296 2016-04-21 float64
297 2016-04-22 float64
298 2016-04-23 float64
299 2016-04-24 float64
300 2016-04-25 float64
301 2016-04-26 float64
302 2016-04-27 float64
303 2016-04-28 float64
304 2016-04-29 float64
305 2016-04-30 float64
306 2016-05-01 float64
307 2016-05-02 float64
308 2016-05-03 float64
309 2016-05-04 float64
310 2016-05-05 float64
311 2016-05-06 float64
312 2016-05-07 float64
313 2016-05-08 float64
314 2016-05-09 float64
```

```
315 2016-05-10 float64
316 2016-05-11 float64
317 2016-05-12 float64
318 2016-05-13 float64
319 2016-05-14 float64
320 2016-05-15 float64
321 2016-05-16 float64
322 2016-05-17 float64
323 2016-05-18 float64
324 2016-05-19 float64
325 2016-05-20 float64
326 2016-05-21 float64
327 2016-05-22 float64
328 2016-05-23 float64
329 2016-05-24 float64
330 2016-05-25 float64
331 2016-05-26 float64
332 2016-05-27 float64
333 2016-05-28 float64
334 2016-05-29 float64
335 2016-05-30 float64
336 2016-05-31 float64
337 2016-06-01 float64
338 2016-06-02 float64
339 2016-06-03 float64
340 2016-06-04 float64
341 2016-06-05 float64
342 2016-06-06 float64
343 2016-06-07 float64
344 2016-06-08 float64
345 2016-06-09 float64
346 2016-06-10 float64
347 2016-06-11 float64
348 2016-06-12 float64
349 2016-06-13 float64
350 2016-06-14 float64
351 2016-06-15 float64
352 2016-06-16 float64
353 2016-06-17 float64
354 2016-06-18 float64
355 2016-06-19 float64
356 2016-06-20 float64
357 2016-06-21 float64
358 2016-06-22 float64
359 2016-06-23 float64
360 2016-06-24 float64
361 2016-06-25 float64
362 2016-06-26 float64
363 2016-06-27 float64
364 2016-06-28 float64
365 2016-06-29 float64
366 2016-06-30 float64
367 2016-07-01 float64
368 2016-07-02 float64
369 2016-07-03 float64
370 2016-07-04 float64
371 2016-07-05 float64
372 2016-07-06 float64
373 2016-07-07 float64
374 2016-07-08 float64
375 2016-07-09 float64
376 2016-07-10 float64
377 2016-07-11 float64
378 2016-07-12 float64
```

```
379 2016-07-13 float64
380 2016-07-14 float64
381 2016-07-15 float64
382 2016-07-16 float64
383 2016-07-17 float64
384 2016-07-18 float64
385 2016-07-19 float64
386 2016-07-20 float64
387 2016-07-21 float64
388 2016-07-22 float64
389 2016-07-23 float64
390 2016-07-24 float64
391 2016-07-25 float64
392 2016-07-26 float64
393 2016-07-27 float64
394 2016-07-28 float64
395 2016-07-29 float64
396 2016-07-30 float64
397 2016-07-31 float64
398 2016-08-01 float64
399 2016-08-02 float64
400 2016-08-03 float64
401 2016-08-04 float64
402 2016-08-05 float64
403 2016-08-06 float64
404 2016-08-07 float64
405 2016-08-08 float64
406 2016-08-09 float64
407 2016-08-10 float64
408 2016-08-11 float64
409 2016-08-12 float64
410 2016-08-13 float64
411 2016-08-14 float64
412 2016-08-15 float64
413 2016-08-16 float64
414 2016-08-17 float64
415 2016-08-18 float64
416 2016-08-19 float64
417 2016-08-20 float64
418 2016-08-21 float64
419 2016-08-22 float64
420 2016-08-23 float64
421 2016-08-24 float64
422 2016-08-25 float64
423 2016-08-26 float64
424 2016-08-27 float64
425 2016-08-28 float64
426 2016-08-29 float64
427 2016-08-30 float64
428 2016-08-31 float64
429 2016-09-01 float64
430 2016-09-02 float64
431 2016-09-03 float64
432 2016-09-04 float64
433 2016-09-05 float64
434 2016-09-06 float64
435 2016-09-07 float64
436 2016-09-08 float64
437 2016-09-09 float64
438 2016-09-10 float64
439 2016-09-11 float64
440 2016-09-12 float64
441 2016-09-13 float64
442 2016-09-14 float64
```

```
443 2016-09-15 float64
444 2016-09-16 float64
445 2016-09-17 float64
446 2016-09-18 float64
447 2016-09-19 float64
448 2016-09-20 float64
449 2016-09-21 float64
450 2016-09-22 float64
451 2016-09-23 float64
452 2016-09-24 float64
453 2016-09-25 float64
454 2016-09-26 float64
455 2016-09-27 float64
456 2016-09-28 float64
457 2016-09-29 float64
458 2016-09-30 float64
459 2016-10-01 float64
460 2016-10-02 float64
461 2016-10-03 float64
462 2016-10-04 float64
463 2016-10-05 float64
464 2016-10-06 float64
465 2016-10-07 float64
466 2016-10-08 float64
467 2016-10-09 float64
468 2016-10-10 float64
469 2016-10-11 float64
470 2016-10-12 float64
471 2016-10-13 float64
472 2016-10-14 float64
473 2016-10-15 float64
474 2016-10-16 float64
475 2016-10-17 float64
476 2016-10-18 float64
477 2016-10-19 float64
478 2016-10-20 float64
479 2016-10-21 float64
480 2016-10-22 float64
481 2016-10-23 float64
482 2016-10-24 float64
483 2016-10-25 float64
484 2016-10-26 float64
485 2016-10-27 float64
486 2016-10-28 float64
487 2016-10-29 float64
488 2016-10-30 float64
489 2016-10-31 float64
490 2016-11-01 float64
491 2016-11-02 float64
492 2016-11-03 float64
493 2016-11-04 float64
494 2016-11-05 float64
495 2016-11-06 float64
496 2016-11-07 float64
497 2016-11-08 float64
498 2016-11-09 float64
499 2016-11-10 float64
500 2016-11-11 float64
501 2016-11-12 float64
502 2016-11-13 float64
503 2016-11-14 float64
504 2016-11-15 float64
505 2016-11-16 float64
506 2016-11-17 float64
```

```
507 2016-11-18 float64
508 2016-11-19 float64
509 2016-11-20 float64
510 2016-11-21 float64
511 2016-11-22 float64
512 2016-11-23 float64
513 2016-11-24 float64
514 2016-11-25 float64
515 2016-11-26 float64
516 2016-11-27 float64
517 2016-11-28 float64
518 2016-11-29 float64
519 2016-11-30 float64
520 2016-12-01 float64
521 2016-12-02 float64
522 2016-12-03 float64
523 2016-12-04 float64
524 2016-12-05 float64
525 2016-12-06 float64
526 2016-12-07 float64
527 2016-12-08 float64
528 2016-12-09 float64
529 2016-12-10 float64
530 2016-12-11 float64
531 2016-12-12 float64
532 2016-12-13 float64
533 2016-12-14 float64
534 2016-12-15 float64
535 2016-12-16 float64
536 2016-12-17 float64
537 2016-12-18 float64
538 2016-12-19 float64
539 2016-12-20 float64
540 2016-12-21 float64
541 2016-12-22 float64
542 2016-12-23 float64
543 2016-12-24 float64
544 2016-12-25 float64
545 2016-12-26 float64
546 2016-12-27 float64
547 2016-12-28 float64
548 2016-12-29 float64
549 2016-12-30 float64
550 2016-12-31 float64
dtypes: float64(550), object(1)
memory usage: 609.8+ MB
```

Figure 3.3: Info

Insights

Here are detailed insights extracted from the provided .info() output for Wikipedia pageview dataset:

Column and Data Types Insights

- **Structure:**

The DataFrame has **145,063 rows and 551 columns**.

- The first column, 'Page', is of type object (string), holding the unique page name including metadata (language, access type, etc.).
- The remaining **550 columns** correspond to daily dates (YYYY-MM-DD format, type float64) spanning from 2015-07-01 to 2016-12-31.

- **Volume and Memory:**

The total memory usage is ~610 MB, which is significant but manageable for modern analytics environments.

- All daily value columns are stored as floats, supporting both real-valued views (for averages/imputation) and simple integer count analysis.

- **Time Span and Consistency:**

The dataset covers **550 consecutive days**—nearly a year and a half—enabling thorough seasonal, weekly, and long-term usage analysis for each Wikipedia page.

- The daily columns appear numerically and chronologically ordered, aiding direct slicing for time series modeling and analysis.

Implications for Analysis and Modeling

- **Missing Data:**

The float64 dtype for daily columns suggests that days with no recorded views appear as NaN.

Handling these (zero-filling, forward/backward filling, or exclusion) is critical for time series continuity and ARIMA-type modeling.

- **Metadata Utilization:**

Since 'Page' is an object column, all metadata extraction (title, region, language, device, etc.) will require parsing this string column—this is essential for feature engineering.

- **Suitability for Forecasting:**

The format (wide with date columns) is ideal for initial EDA, but should be melted or reshaped to a long format (page, date, views) for modeling with most time series frameworks.

- **Computational Feasibility:**

Working with 145,063 independent time series (one per page) is a large-scale modeling challenge, especially if fitting individualized models. Grouping (e.g., by language) may offer efficiency and interpretability.

3.3.4 Extraction of Structured Metadata from Wikipedia Page Names Using Regular Expressions

```

1 import regex
2 regex = regex.compile(pattern)
3
4 def split_page_fields(string):
5     match_obj = regex.match(string)
6     match_list = []
7     if match_obj:
8         match_list.append(match_obj.group('specific_name') if match_obj.group('specific_name') else np.nan)
9         match_list.append(match_obj.group('language') if match_obj.group('language') else np.nan)
10        match_list.append(match_obj.group('access_type') if match_obj.group('access_type') else np.nan)
11        match_list.append(match_obj.group('access_origin') if match_obj.group('access_origin') else np.nan)
12    return match_list
13 else:
14     return np.full(4,np.nan)

1 split_page_fields('2NE1_zh.wikipedia.org_all-access_spider')
... ['2NE1', 'zh', 'all-access', 'spider']

1 df[['Specific_Name', 'Language', 'Access_Type', 'Access-Origin']] = df['Page'].apply(split_page_fields).to_list()

```

Figure 3.4: Extraction of Structured Metadata from Wikipedia Page Names Using Regular Expressions

Insights

- **Automated Feature Extraction:**

The provided code uses regex to decompose Wikipedia page names into meaningful components: specific name/title, language, access type (such as all-access), and access origin (such as spider or agent).

- **Scalable Parsing Function:**

The `split_page_fields` function efficiently processes each entry in the 'Page' column, robustly returning structured data for further analysis, even in the presence of missing fields (returns NaN when a field is absent).

- **Metadata Utilization:**

Extracting features such as language or access origin enables:

- Grouping and segmentation of data (e.g., by language or device type) for deeper analysis or modeling.
- Language-wise or region-wise traffic patterns, fundamental for targeted ad placement and content strategy.

- **DataFrame Enrichment:**

The new fields—'Specific_Name', 'Language', 'Access_Type', and 'Access-Origin'—

greatly enhance the richness and granularity of the dataset, allowing for insightful EDA, visualization, and feature engineering for forecasting models.

3.3.4 Columns of Dataset

```
1 df.columns  
... Index(['Page', '2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',  
        '2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08', '2015-07-09',  
        ...  
        '2016-12-26', '2016-12-27', '2016-12-28', '2016-12-29', '2016-12-30',  
        '2016-12-31', 'Specific_Name', 'Language', 'Access_Type',  
        'Access-Origin'],  
       dtype='object', length=555)
```

Figure 3.5: Columns of Dataset

Insights

- **Expanded Feature Set:**

The DataFrame now contains 555 columns—original daily date columns plus four additional features: 'Specific_Name', 'Language', 'Access_Type', and 'Access-Origin'. This reflects successful feature engineering and effective data enrichment for advanced analytics.

- **Structured Metadata for Each Page:**

The last four columns explicitly capture information parsed from the page name:

- 'Specific_Name': The actual topic or unique identifier of the page.
- 'Language': The language version of Wikipedia for that page (key for regional analysis).
- 'Access_Type': The type of access (e.g., all-access, desktop, mobile).
- 'Access-Origin': The traffic origin (e.g., spider for bots/crawlers versus agent for real users).

- **Business and Modeling Value:**

- These features are now easily available for filtering, grouping, and aggregating views by language or user type, which is essential for region-specific ad targeting and campaign optimization.
- Advanced models and analyses (like forecasting or conversion optimization) can account for language, platform, and bot vs human traffic, leading to more precise and robust outcomes.

- **Data Engineering Best Practice:**

Extracting and storing these as direct columns greatly simplifies downstream exploration, visualization, and modeling—reducing repeated parsing overhead and making the dataset more "tidy" and ML-ready.

3.3.5 Handling Null Values

```
In [ ]: df.isna().sum(axis=1)
```

```
Dut[ ]:      0
             0   0
             1   0
             2   0
             3   0
             4  291
             ...
145058  544
145059  550
145060  550
145061  550
145062  550

145063 rows × 1 columns
```

dtype: int64

```
In [ ]: null_rows = df[df.iloc[:,1:].isna().all(axis=1) == True].index
print('Total Null values', len(null_rows))
df = df.drop(null_rows)
df1 = df.copy()
df1.head(5)
```

Total Null values 127

```
Dut[ ]:      Page  2015- 2015- 2015- 2015- 2015- 2015- 2015-
             07-01 07-02 07-03 07-04 07-05 07-06 07-07 07-08
             0    2NE1_zh.wikipedia.org_all-
                   access_spider    18   11    5   13   14    9    9   22
             1    2PM_zh.wikipedia.org_all-
                   access_spider    11   14   15   18   11   13   22   11
             2    3C_zh.wikipedia.org_all-access_spider    1    0    1    1    0    4    0    3
             3    4minute_zh.wikipedia.org_all-
                   access_spider    35   13   10   94    4   26   14    9
             4  52_Hz_I_Love_You_zh.wikipedia.org_all-
                   access_s...    NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
```

Figure 3.6: Handling Null Values

Insights

- **Missing Value Distribution:**
 - A small number of pages have a substantial amount of missing data (NaN) across all day columns.
 - Example: In your check, several rows have 550 NaN values—meaning no view data for any date for those pages.
 - Most pages at the top of the DataFrame show complete data or only a few missing cells.
- **Data Cleaning Action:**
 - Rows (pages) with all views missing are identified and dropped from the dataset, as they cannot be used for any time series modeling or analysis.
 - Your code determined that there were 127 such rows ("Total Null values 127") and removed them, ensuring the dataset remains meaningful, dense, and efficient for modeling.
- **Data Integrity Maintained:**
 - After removal, the remaining dataset has most rows with at least some view data, greatly improving the reliability and validity of downstream analysis.
 - Pages that are partially missing (gaps for some days but not all) remain, which is appropriate for time series where short gaps may be imputed but rows with no signal are excluded.

3.3.6 Exploring Categorical Diversity in Wikipedia Page Metadata: Language, Access Type, and Origin



```
1 for i in ['Language', 'Access_Type', 'Access_Origin']:  
2     print(f'{i} - {df[i].nunique()} {list(df[i].dropna().unique())}')  
  
... Language - 7 ['zh', 'fr', 'en', 'ru', 'de', 'ja', 'es']  
Access_Type - 3 ['all-access', 'desktop', 'mobile-web']  
Access_Origin - 2 ['spider', 'all-agents']
```

Figure 3.7: Exploring Categorical Diversity in Wikipedia Page Metadata: Language, Access Type, and Origin

Insights

- **Language Diversity:**

The dataset covers Wikipedia pages in **7 distinct languages**:

- 'zh' (Chinese), 'fr' (French), 'en' (English), 'ru' (Russian), 'de' (German), 'ja' (Japanese), and 'es' (Spanish).

This linguistic breadth enables region-specific analysis and the tailoring of ad campaigns for different user bases.

- **Access Type Segmentation:**

There are **3 types of access** modes captured in the dataset:

- 'all-access', 'desktop', and 'mobile-web'.

Segmenting by device allows marketers to understand user behavior, prioritize ad formats, and optimize user experience per platform.

- **Traffic Origin Classification:**

The data distinguishes between **2 classes of access origin**:

- 'spider' (automated crawlers and bots), 'all-agents' (likely representing human user traffic).

This distinction is vital for filtering out artificial traffic, ensuring analytics and ad targeting are based on genuine human viewership.

- **Business Impact:**

These structured categorical features add major value for downstream analyses, e.g.:

- Comparing traffic trends across global regions.
- Analyzing device/platform preferences for audience targeting.
- Cleaning and segmenting data to focus only on human-driven ad opportunities.

3.3.7 Value Count of Languages

```
1 df['Language'].value_counts()
```

Language	count
en	24108
ja	20431
de	18547
fr	17802
zh	17229
ru	15022
es	14069

dtype: int64

Figure 3.8: Value Count of Languages

Insights from Language Distribution

- English Pages Dominate:**
The largest share of Wikipedia pages in this dataset is in English ('en') with 24,108 pages, making up the most significant single language group and offering the broadest reach for English-speaking audiences and ad campaigns.
- Significant Asian Language Representation:**
Japanese ('ja', 20,431) and Chinese ('zh', 17,229) pages together form a substantial portion, reflecting strong demand and engagement from East Asian internet users.
- Major European Languages Included:**
German ('de', 18,547), French ('fr', 17,802), Russian ('ru', 15,022), and Spanish ('es', 14,069) are also well represented, supporting the dataset's ability to power regionally targeted insights and campaigns across Europe and Latin America.
- Implications for Analysis and Ad Targeting:**
The diverse language mix underscores the opportunity to segment users and tailor content or advertising strategies by language for maximum effectiveness.

- Ad infrastructure can prioritize the highest-traffic language groups for campaign launches or A/B tests.
- Lower-count languages (relative to English) remain large enough to extract robust insights and justify dedicated marketing efforts.

3.3.8 Extracting Language Name



```

1 import pycountry
2
3 def get_language_name(code):
4     try:
5         return pycountry.languages.get(alpha_2=code).name
6     except:
7         return code # return original if not found
8
9 df['Language_Full'] = df['Language'].apply(get_language_name)
10

```

Figure 3.9: Extracting Language Name

Insights

- **Improved Readability:**
The code adds a new column, 'Language_Full', containing the full language names (e.g., 'en' mapped to 'English', 'zh' to 'Chinese'). This enhances the interpretability of the dataset for stakeholders who may not be familiar with ISO language codes.
- **Automated Mapping:**
The use of the pycountry library and the get_language_name function ensures automated, systematic translation of codes to names. This reduces manual lookup errors and supports consistency across the dataset.
- **Robust Fallback:**
If a language code is not found in the library, the original code is returned, ensuring no data is lost and all entries remain traceable.
- **Business and Communication Value:**
Full names in reporting make it easier for business, marketing, and non-technical audiences to understand analysis, dashboards, or recommendations—especially when presenting regional performance or planning language-specific campaigns.

- **Data Enrichment Best Practice:**

Creating such a mapping is a key data enrichment step that facilitates group-by operations, summaries, and visualization, especially when comparing the performance of Wikipedia pages across languages.

3.3.9 Distribution Analysis of Languages, Access Types, and Origins in Wikipedia Pageviews

```
1 from IPython.display import display, HTML
2 plt.figure(figsize=(20,5))
3 plt.suptitle('Proportions')
4 plt.subplot(1,3,1)
5 plt.title('Languages')
6 plt.pie(df['Language'].value_counts(), autopct='%.2f%%', labels=df['Language'].value_counts().index)
7 plt.subplot(1,3,2)
8 plt.title('Access Type')
9 plt.pie(df['Access_Type'].value_counts(), autopct='%.2f%%', labels=df['Access_Type'].value_counts().index)
10 plt.subplot(1,3,3)
11 plt.title('Access Origin')
12 plt.pie(df['Access_Origin'].value_counts(), autopct='%.2f%%', labels=df['Access_Origin'].value_counts().index)
13 plt.show()
14
15 html = f"""
16 <div style="display: flex; gap: 40px; justify-content: space-around;">
17     <div>
18         <h4>Language</h4>
19         {df['Language'].value_counts().to_frame().to_html()}
20     </div>
21     <div>
22         <h4>Access Type</h4>
23         {df['Access_Type'].value_counts().to_frame().to_html()}
24     </div>
25     <div>
26         <h4>Access Origin</h4>
27         {df['Access_Origin'].value_counts().to_frame().to_html()}
28     </div>
29 </div>
30 """
31
32 display(HTML(html))
```

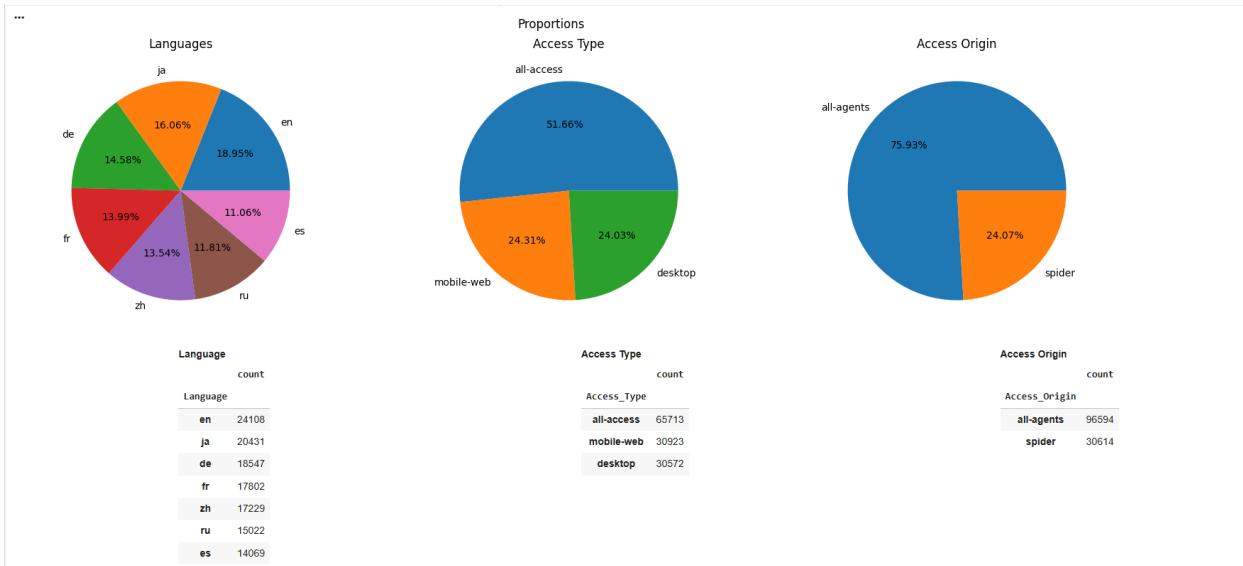


Figure 3.10: Distribution Analysis of Languages, Access Types, and Origins in Wikipedia Pageviews

Insights

- **Languages:**
 - English pages are the most abundant (about 19%), followed by Japanese and German, with other major languages (French, Chinese, Russian, Spanish) each making up a sizeable share.
 - No single language overwhelmingly dominates, illustrating Wikipedia's global, multilingual reach and providing a strong foundation for regional segmentation in analytics or marketing.
- **Access Types:**
 - The majority of page entries are classified under "all-access" (over 51%), which aggregates all device and user types.
 - "Mobile-web" and "desktop" each contribute roughly a quarter of the dataset, indicating balanced traffic between traditional and mobile consumption channels.
 - This diverse device mix is valuable for cross-platform modeling and for advertisers tailoring creative/content by device.
- **Access Origin:**
 - Roughly 76% of the traffic is attributed to "all-agents" (presumably human users), while 24% is "spider" (bots/crawlers).
 - Bot traffic, though significant, is appropriately distinguished—enabling measurement and targeting based purely on genuine user engagement, which is essential for ad optimization.
- **Business Implications:**

- Granular breakdowns like these are essential for interpreting overall trends, evaluating market opportunities, and segmenting users for more effective advertising and content strategies.
- Device and bot/human segmentation inform not just campaign targeting, but also quality control in analytics and data-driven ROI calculations.

3.3.10 Wikipedia Languages and their Access Origin

```

1 lag_acc_orn = df[['Language','Access_Origin']]
2 lag_acc_orn_count = lag_acc_orn.value_counts().reset_index()
3 display(lag_acc_orn_count)
4 plt.figure(figsize=(10, 5))
5 ax = sns.barplot(x='Language', y='count', hue='Access_Origin', data=lag_acc_orn_count)
6 plt.title('Wikipeida Languages and their Access Origin')
7 plt.legend(title='Access Origin')
8 plt.xlabel('Language')
9 plt.ylabel('Count')
10 annotate(ax)
11 plt.show()
12
13 del lag_acc_orn_count, lag_acc_orn
14 gc.collect()

```

	Language	Access_Origin	count
0	en	all-agents	19177
1	ja	all-agents	15424
2	de	all-agents	13960
3	fr	all-agents	13302
4	zh	all-agents	12919
5	ru	all-agents	11280
6	es	all-agents	10532
7	ja	spider	5007
8	en	spider	4931
9	de	spider	4587
10	fr	spider	4500
11	zh	spider	4310
12	ru	spider	3742
13	es	spider	3537

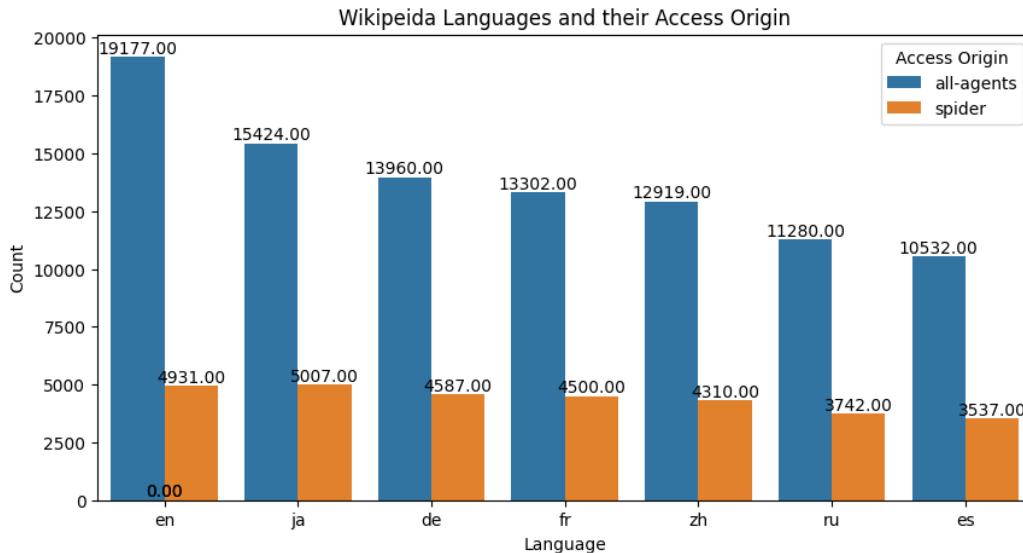


Figure 3.11: Wikipedia Languages and their Access Origin

Insights

- **Human vs. Bot Traffic by Language:**
 - For every language, the majority of entries are from "all-agents" (presumed human users).
 - The English Wikipedia has the highest volume for both human agents (19,177) and bots (4,931), reflecting its global dominance.
 - The split is similar across other languages, with Japanese (all-agents: 15,424; spider: 5,007) and German (all-agents: 13,960; spider: 4,587) also showing substantial human and bot counts.
 - Even the languages with lower total counts, such as Russian and Spanish, maintain this pattern: approximately 3–5K bot readings per language, with humans consistently being the majority.
- **Implications for Quality Analytics and Ad Decisioning:**
 - The ability to segregate by "Access Origin" means modeling and analytics can be focused on genuine user engagement, which is essential for effective digital marketing strategies and accurate forecast modeling.
 - The significant number of spider counts in each language underlines the need to always account for bot traffic; for ad-targeting and engagement analysis, "all-agents" rows should be prioritized.
- **Comparison Across Languages:**
 - The proportion of bot (spider) entries to human entries is generally consistent across languages, suggesting no single language is disproportionately affected by bot activity relative to its own size.

- This uniformity ensures more balanced and comparable cross-lingual engagement analytics.

3.3.11 Wikipedia Languages and their Access Type

```
▶ 1 lag_acc_type = df[['Language','Access_Type']]
  2 lag_acc_type_count = lag_acc_type.value_counts().reset_index()
  3 display(lag_acc_type_count)
  4 plt.figure(figsize=(20, 5))
  5 ax = sns.barplot(x='Language', y='count', hue='Access_Type', data=lag_acc_type_count)
  6 plt.title('Wikipedia Languages and their Access Type')
  7 plt.legend(title='Access Type')
  8 plt.xlabel('Language')
  9 plt.ylabel('Count')
 10 annotate(ax)
 11 plt.show()
 12
 13 del lag_acc_type, lag_acc_type_count
 14 gc.collect()
```

	Language	Access_Type	count
0	en	all-access	14350
1	ja	all-access	10013
2	de	all-access	9173
3	fr	all-access	8999
4	zh	all-access	8620
5	ru	all-access	7484
6	es	all-access	7074
7	ja	desktop	5562
8	en	desktop	4975
9	ja	mobile-web	4856
10	en	mobile-web	4783
11	fr	mobile-web	4756
12	de	mobile-web	4734
13	de	desktop	4640
14	zh	mobile-web	4340
15	zh	desktop	4269
16	fr	desktop	4047
17	ru	desktop	3809
18	ru	mobile-web	2720

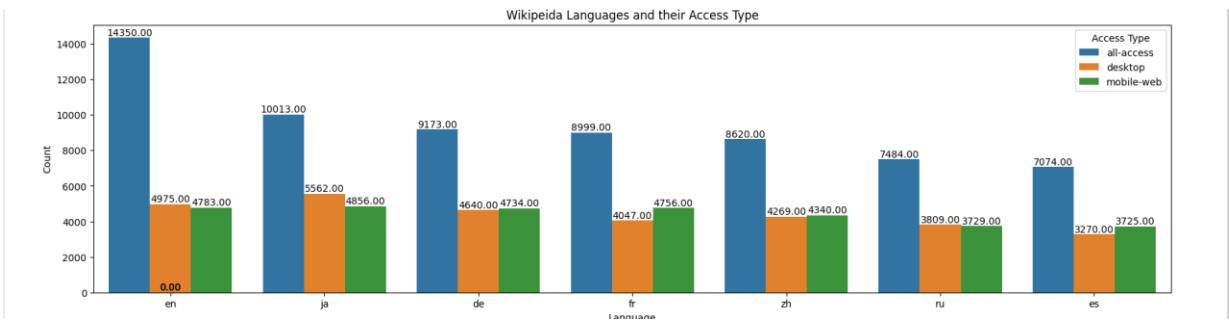


Figure 3.12: Wikipedia Languages and their Access Type

Insights

- All-Access Leads in Every Language:**
 For every major language (English, Japanese, German, French, Chinese, Russian, Spanish), "all-access" is the dominant access type—indicating that most pageview entries are reported as aggregated totals across devices and platforms.
- Balanced Split Between Desktop and Mobile-Web:**
 Both "desktop" and "mobile-web" counts are substantial and, in most languages, very close in magnitude, reflecting contemporary user behavior patterns:
 - For English: desktop (4,975), mobile-web (4,783)
 - For Japanese: desktop (5,562), mobile-web (4,856)
 - For all languages, the gap between desktop and mobile-web is consistently small, showing a balanced and modern distribution of Wikipedia consumption habits.
- Business Implications for Device Targeting:**
 - The near-even splits suggest that device-specific campaigns (mobile or desktop) will have similar-sized audiences, supporting parallel testing or device-tailored ad strategies.
 - Optimizing user experience and advertising creative for both mobile and desktop platforms remains essential for all major language markets.
- Consistency Across Languages:**
 - The pattern is remarkably stable across all seven language groups—the difference between desktop and mobile-web is rarely more than a few hundred entries.
 - This suggests global Wikipedia usage patterns (in terms of device preference) are broadly similar, likely supporting the generalizability of findings or approaches across markets.

3.3.12 Wikipedia Languages and their Clicks

```

1 agg_df_lang_click = df_lang_click.groupby(by='Language')[['Clicks']].agg('sum')
2 sorted_agg_lang_click_df = agg_df_lang_click.reset_index().sort_values(by='Clicks', ascending=False)
3 sorted_agg_lang_click_df.reset_index(inplace=True, drop=True)
4
5 display(sorted_agg_lang_click_df)
6 plt.figure(figsize=(15, 5))
7 ax = sns.barplot(data=sorted_agg_lang_click_df.loc[:, :], x='Language', y='Clicks')
8 plt.title('Wikipedia Languages and their clicks')
9 plt.xlabel('Languages')
10 plt.ylabel('Clicks')
11 annotate(ax)
12 plt.show()
13
14 del agg_df_lang_click, df_lang_click, sorted_agg_lang_click_df
15 gc.collect()

```

***	Language	Clicks
0	English	2,436,899
1	Spanish	674,547
2	Russian	532,443
3	German	477,814
4	Japanese	419,524
5	French	358,264
6	Chinese	184,107
7	Unknown	72,133

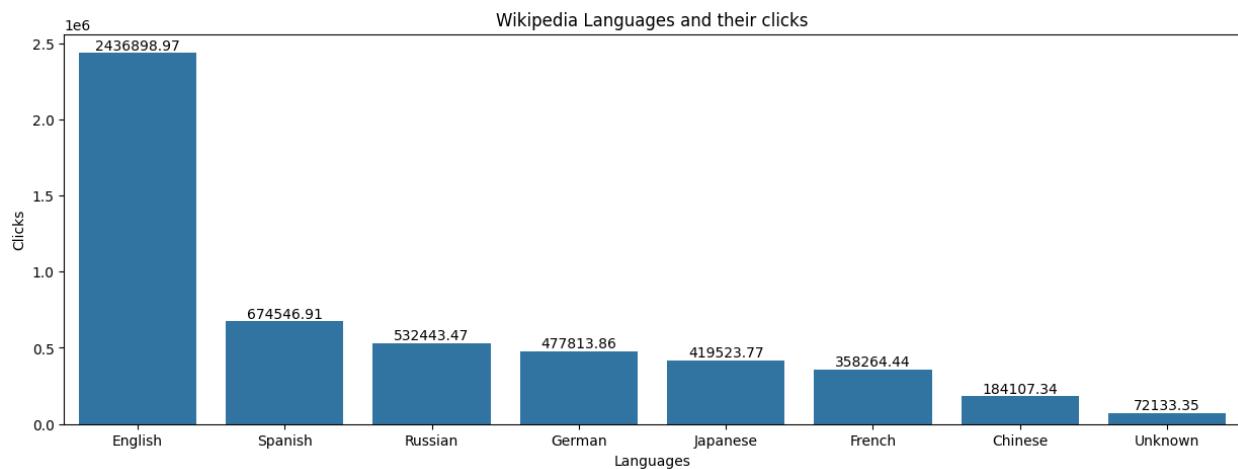


Figure 3.13: Wikipedia Languages and their Clicks

Insights

- English Wikipedia Dominates Engagement:** English pages receive the largest number of clicks by a huge margin—over 2.4 million—making it the clear priority for any high-reach, global campaign.
- Significant Non-English Engagement:** Spanish (674,547), Russian (532,443), German (477,814), Japanese (419,524), and French (358,264) pages all record hundreds of thousands of clicks.

This demonstrates that non-English Wikipedias represent robust, highly engaged markets for localized or region-specific outreach.

- **Chinese and Long-Tail Languages:**

Chinese registers a somewhat lower click count (184,107), but this number still represents meaningful engagement, especially considering internet restrictions and the broader context. "Unknown" languages account for a small but non-negligible number of clicks, potentially reflecting either bot/incomplete data or long-tail, niche markets.

- **Strategic Targeting Insight:**

The distribution justifies:

- Focusing most effort and largest campaigns on the English Wikipedia for largest impact.
- Building dedicated, language-specific strategies for Spanish, Russian, German, and Japanese segments.
- Not neglecting even lower-volume languages, as they still reflect substantial active communities.

- **Operational Analytics Note:**

The dramatic disparity in clicks highlights the importance of weighting or segmenting analysis and campaign budgeting according to actual engagement and potential reach, rather than page counts alone.

3.3.13 Language-Specific DataFrame Pickling

```
1 languages_list = ['English', 'French', 'Spanish', 'Chinese', 'Japanese', 'German', 'Russian']
2
3 for lang in languages_list:
4     with open(f'/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/{lang}_Full_df.pickle', 'wb') as fp:
5         pickle.dump(all_lang_df[all_lang_df['Language_Full'] == lang], fp)
```

Figure 3.14: Language-Specific DataFrame Pickling

Insights

- The script iterates over a predefined list of languages and filters a DataFrame (all_lang_df) to select rows where the 'Language_Full' column matches each language.
- For each language, it saves the filtered DataFrame as a separate Pickle file in a Google Drive directory, naming each file using the language (e.g., English_Full_df.pickle).
- The use of 'wb' mode ensures the files are written in binary format, which is required for Pickle serialization.
- This approach is useful for organizing and exporting data by language, facilitating easy access and loading of language-specific datasets for further analysis or sharing.

3.3.14 Access Type and Access Origin Type Of All Languages

```
1 def get_access_type_origin_value_cpoints(df_dict):
2
3     plot_dict = {}
4
5     for lag, data in df_dict.items():
6
7         acc_type_ogn_df = data[['Access_Type', 'Access-Origin']].value_counts()
8         plot_dict.update({lag:acc_type_ogn_df.reset_index()})
9
10    plt.figure(figsize=(18,20))
11    plt.suptitle('Access Type and Access Origin Count of all Langauges')
12
13    for i in range(1,8):
14        plt.subplot(4,2,i)
15        plt.title(list(plot_dict.keys())[0])
16        ax = sns.barplot(data=plot_dict.get(list(plot_dict.keys())[0]), x='Access_Type', y='count', hue='Access-Origin')
17        annotate(ax)
18        plot_dict.pop(list(plot_dict.keys())[0])
19
20    plt.show()
```

```
1 all_lang_df_objs_dict = {}
2
3 for lang in languages_list:
4     with open(f'/content/drive/MyDrive/Scaler_DSMU_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/{lang}_Full_df.pickle', 'rb') as fp:
5         all_lang_df_objs_dict.update({lang:pickle.load(fp)})
6 get_access_type_origin_value_cpoints(all_lang_df_objs_dict)
7 # all_lang_df_objs_dict = {'English':english_df, 'French':french_df, 'Spanish':spanish_df, 'Chinese':chinese_df, 'Japanese':japanese_df, 'German': german_df, 'Russian': russiaan_df}
8 # get_access_type_origin_value_cpoints(all_lang_df_objs_dict)
```

Access Type and Access Origin Count of all Langauges

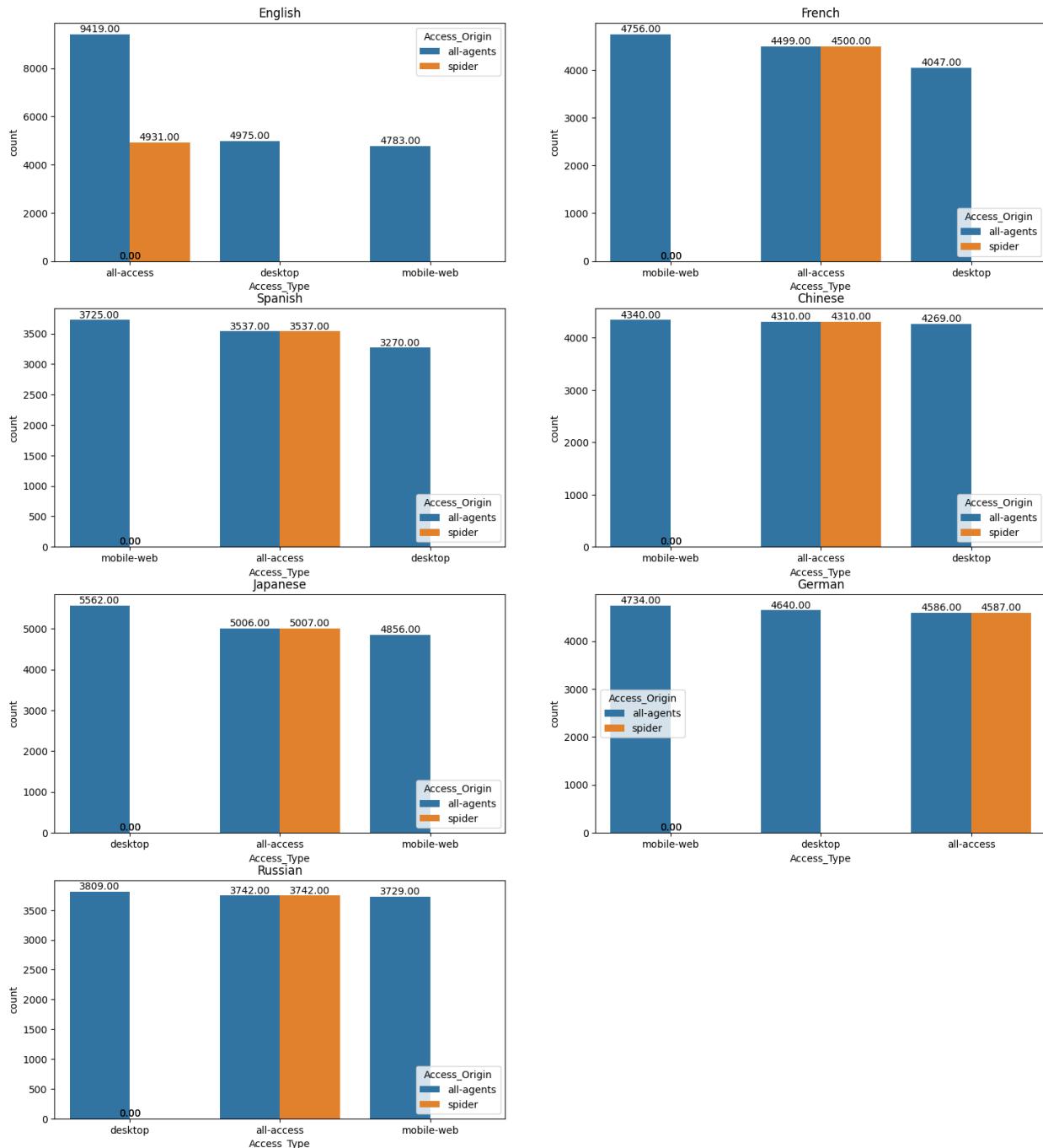


Figure 3.15: Access Type and Access Origin Type Of All Languages

Insights

- The code processes multilingual datasets, extracting and plotting counts of "Access Type" and "Access Origin" for each language, visualizing this information side by side for comparison.
- Each subplot represents a language (English, French, Spanish, Chinese, Japanese, German, Russian), with barplots showing how data access is distributed among desktop, mobile-web, and all-access types, and further split by all-agents and spider origins.
- The barplots reveal that for all languages, the "spider" origin either has zero counts for some access types or mirrors the "all-agents" counts exactly for "all-access".
- Access via "mobile-web" and "desktop" dominates for most languages under the "all-agents" category, while the "spider" counts are usually present only for "all-access", highlighting that crawlers (spiders) may primarily target the aggregate view rather than device-specific accesses.
- This approach helps identify consumption patterns and differences in how content is accessed and crawled across languages, which is useful for tailoring web analytics or digital strategies for a multilingual audience.

3.3.15 Duplicate Values Check



```
1 for k, v in all_lang_df_objs_dict.items():
2     print(f'{k} Langauge data frame has {len(v[v.duplicated()])} duplicate values')
```

```
...
English Langauge data frame has 0 duplicate values
French Langauge data frame has 0 duplicate values
Spanish Langauge data frame has 0 duplicate values
Chinese Langauge data frame has 0 duplicate values
Japanese Langauge data frame has 0 duplicate values
German Langauge data frame has 0 duplicate values
Russian Langauge data frame has 0 duplicate values
```

Figure 3.16: Duplicate Values Check

Insights

- All the individual language DataFrames (English, French, Spanish, Chinese, Japanese, German, Russian) contain zero duplicate values, as verified by the code.
- This indicates that the data for each language is clean and free from redundancy, which improves the reliability of any further analysis and visualization tasks.

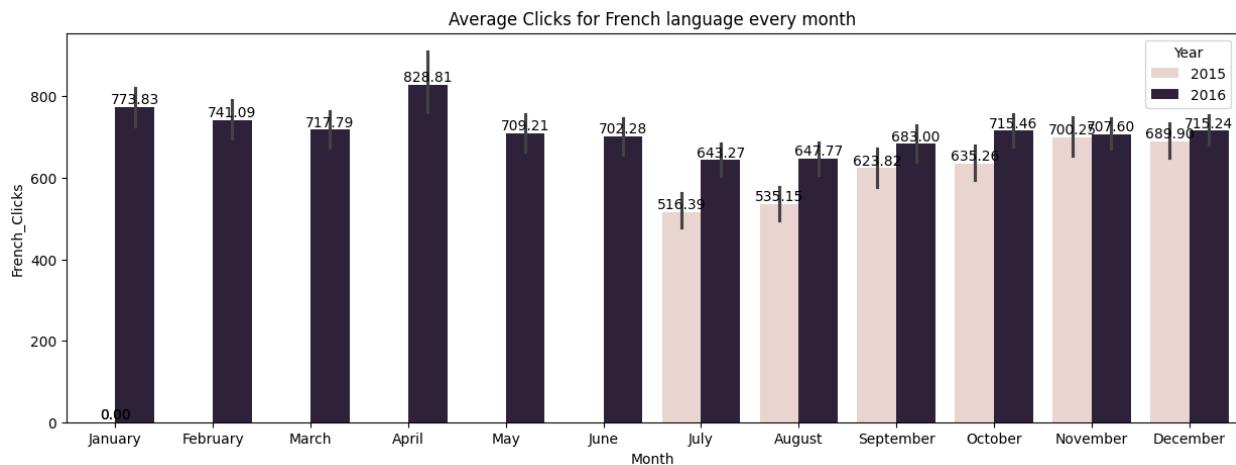
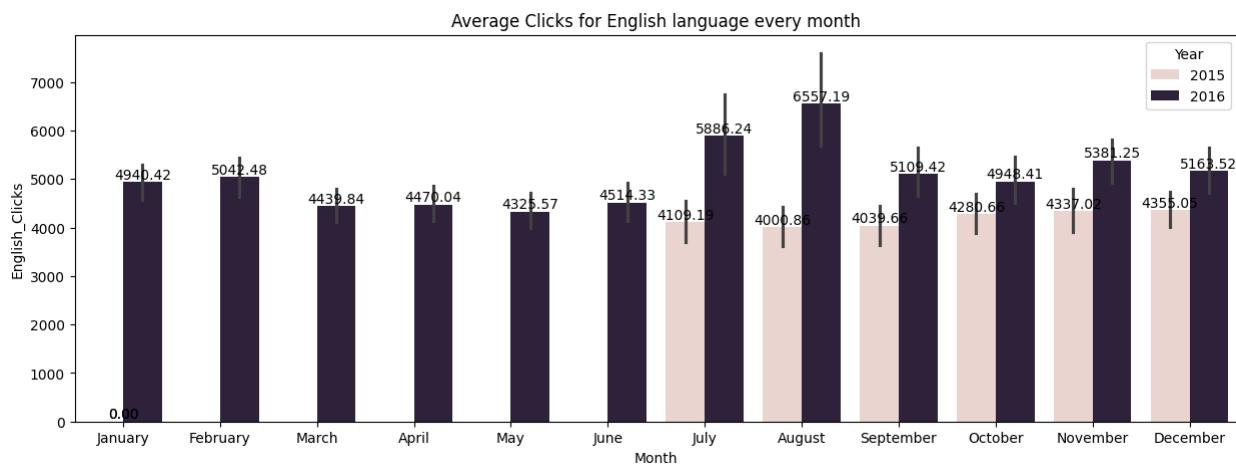
- Ensuring no duplicates enhances data integrity and ensures the accuracy of statistical or machine learning outputs that might use these DataFrames.

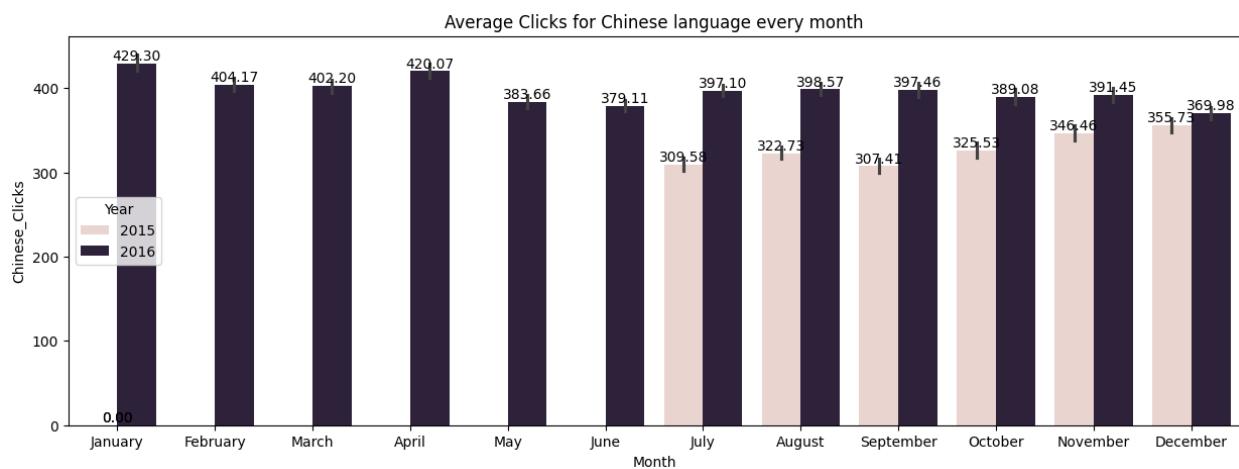
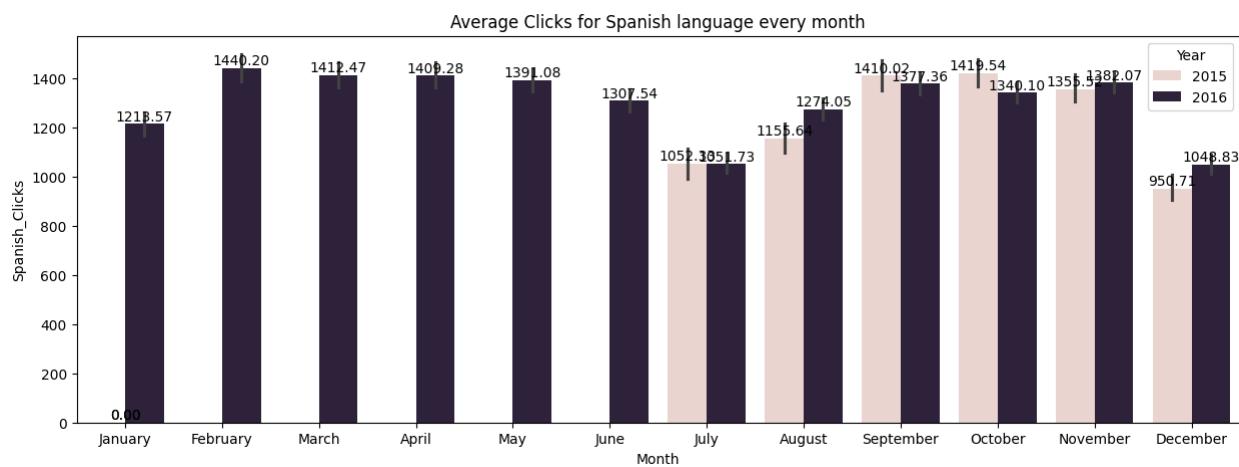
3.3.16 Monthly Average Clicks

```

1 month_order = ['January', 'February', 'March', 'April', 'May', 'June',
2                 'July', 'August', 'September', 'October', 'November', 'December']
3
4 for lang in languages_list:
5     with open(f'/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/{lang}.pickle', 'rb') as fp:
6         dataframe = pickle.load(fp)
7         dataframe['Month'] = dataframe['Dates'].dt.month_name()
8         dataframe['Year'] = dataframe['Dates'].dt.year
9
10        plt.figure(figsize=(15,5))
11        plt.title(f'Average Clicks for {lang} language every month')
12        ax = sns.barplot(data=dataframe, x='Month', y=f'{lang}_Clicks', hue='Year', order=month_order)
13        annotate(ax)
14        plt.show()
15        del dataframe
16        gc.collect()

```





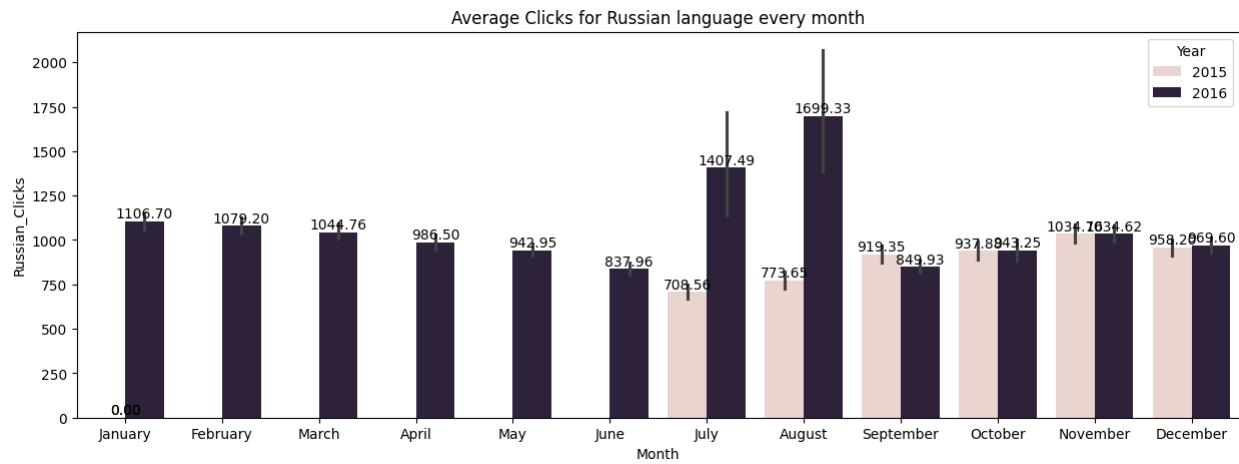
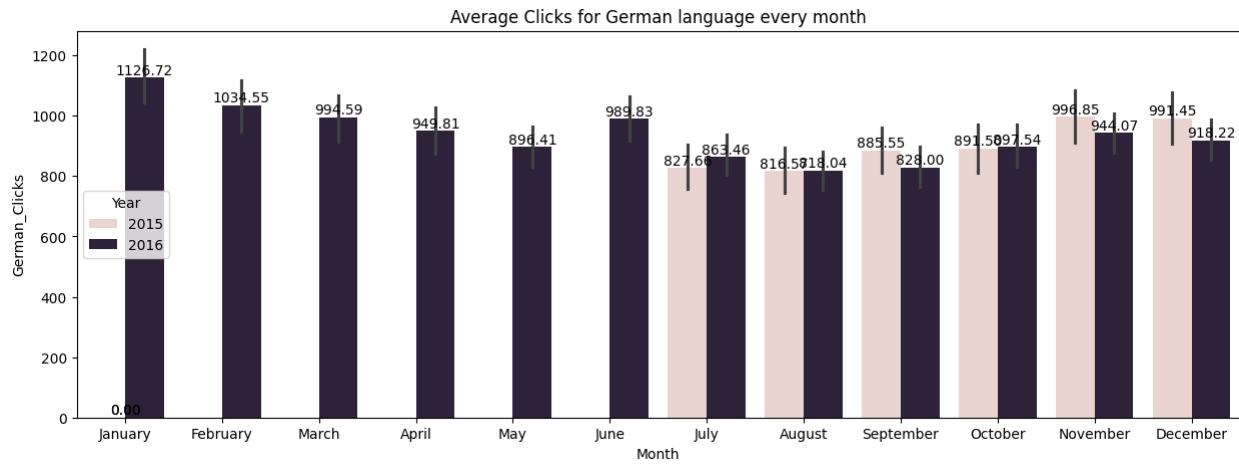
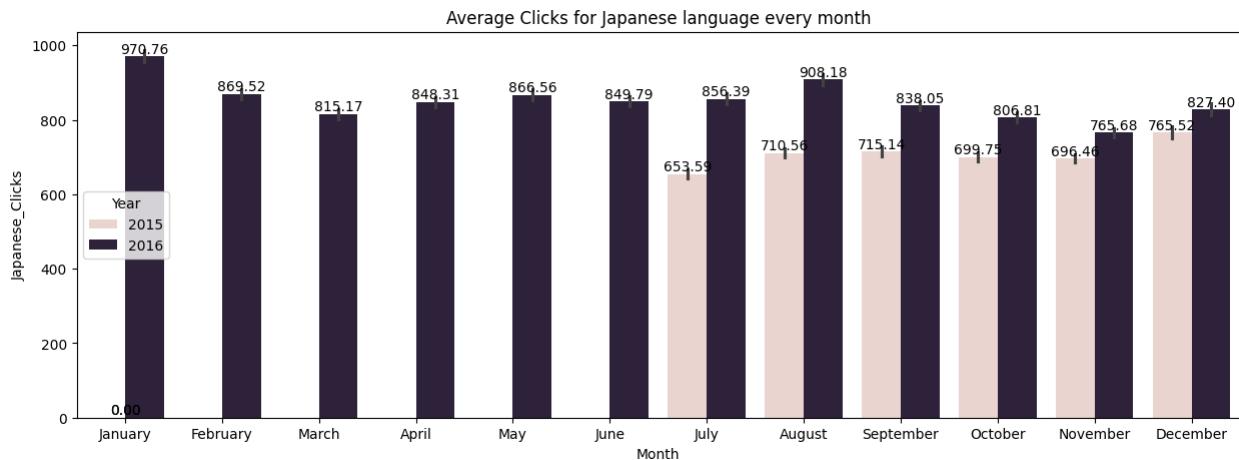


Figure 3.17: Monthly Average Clicks

Insights

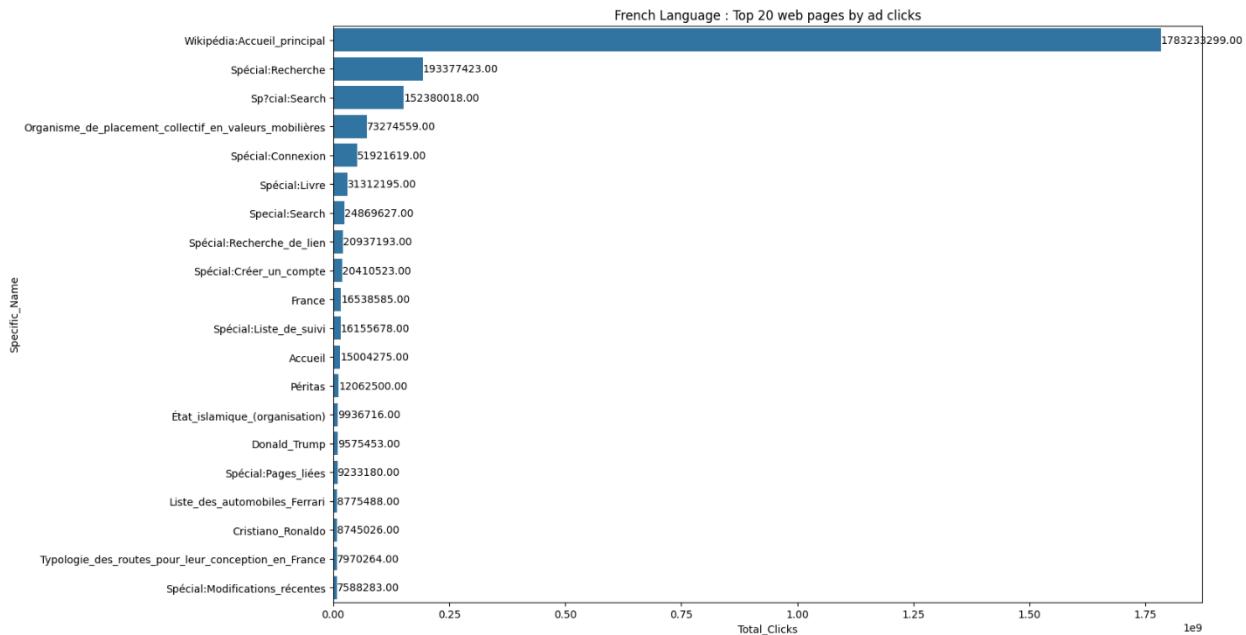
- **English, Spanish, and Russian** showed high average monthly clicks in both years, with English seeing the highest overall numbers each month. Notably, English had a significant spike in clicks in July and August 2016, peaking at over 6500 clicks in August, likely driven by a specific event or seasonal trend.
- **Spanish** demonstrated remarkably stable performance, consistently surpassing 1200 clicks every month (except January 2015) and generally remaining above 1300-1400 clicks in both years. This indicates strong and sustained interest in Spanish-language content throughout the year.
- **German and French** experienced more moderate click volumes, with German ranging from roughly 800 to 1100 clicks and French between 600 and 820 clicks most months. There is little variation between years, although moderate declines are observed mid-year for German, with increased stability in later months for both languages.
- **Japanese and Chinese** received the lowest click volumes, suggesting lower user demand or smaller audiences for these languages. Japanese had a small upward trend in late summer/autumn 2016, while Chinese maintained a steady profile around 400 clicks per month in 2016.
- **Russian** experienced pronounced seasonal spikes, particularly in July and August 2016, which deviate sharply from the preceding and succeeding months' averages. This pattern may be associated with specific content releases, campaigns, or external events influencing Russian-language interest.
- Across most languages, **2016 consistently outperforms 2015** in terms of click volumes, reflecting broader growth, increased audience engagement, or successful promotional efforts over the year.

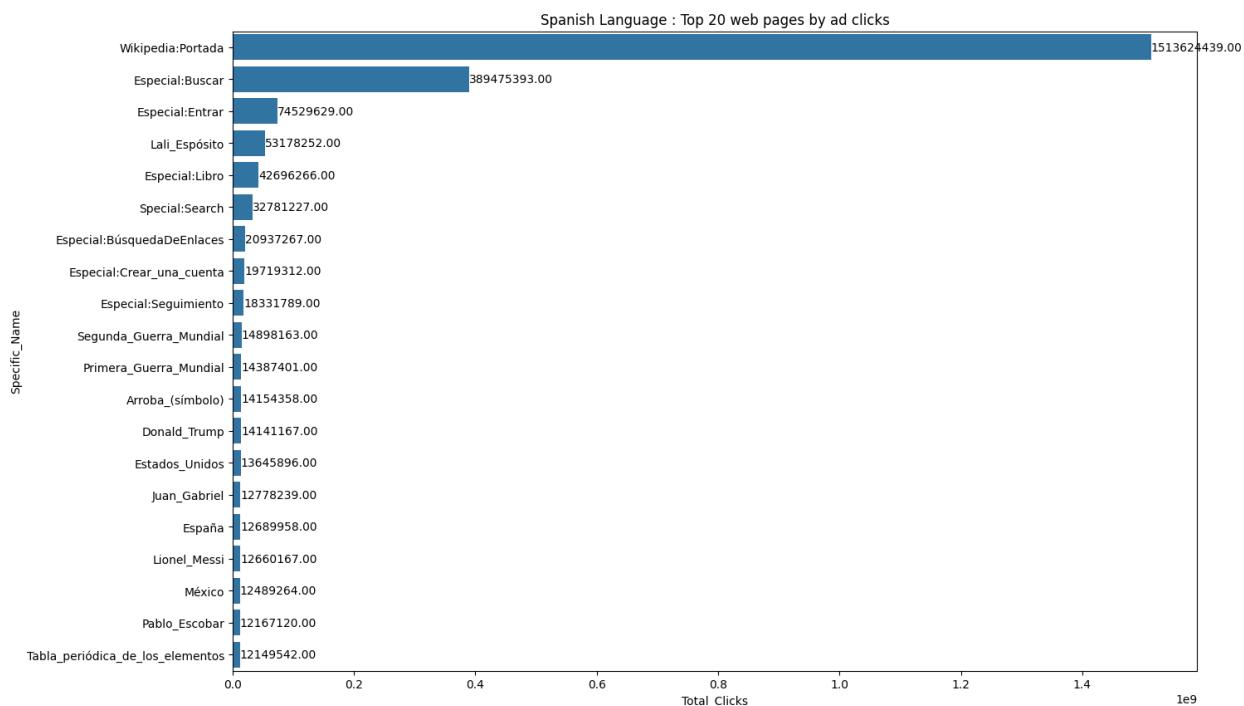
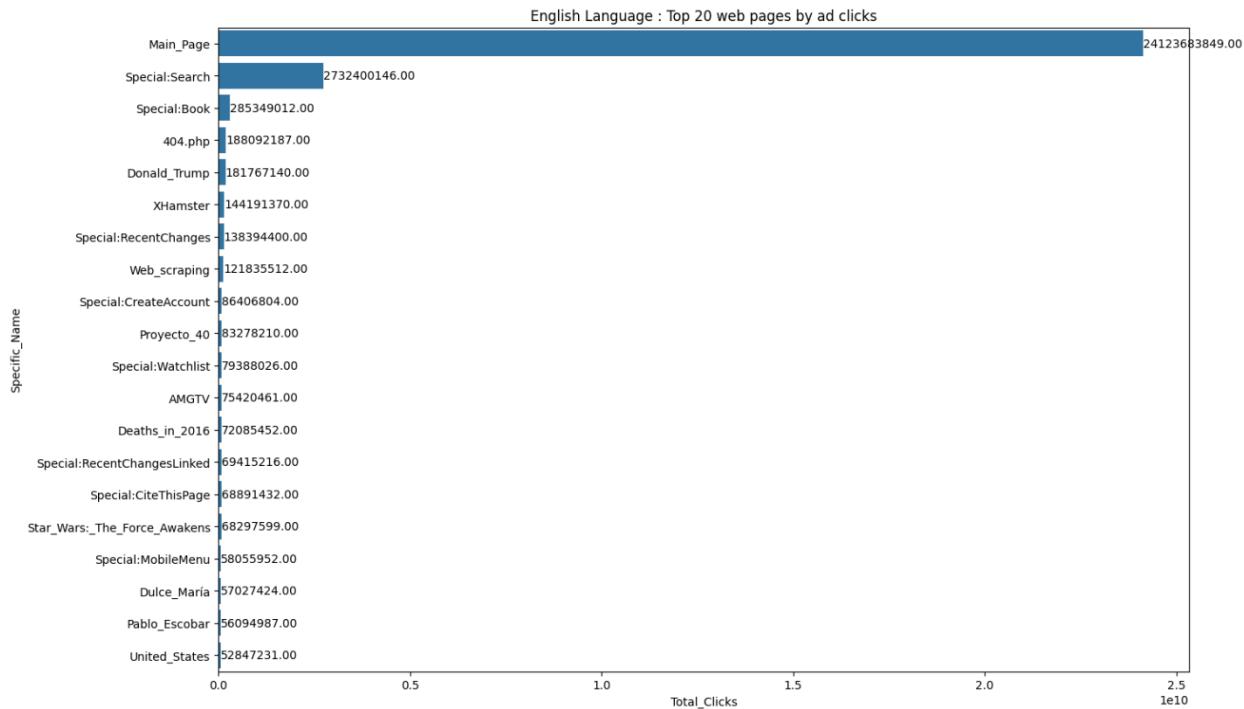
3.3.17 Top 20 Web Pages By Ad Clicks

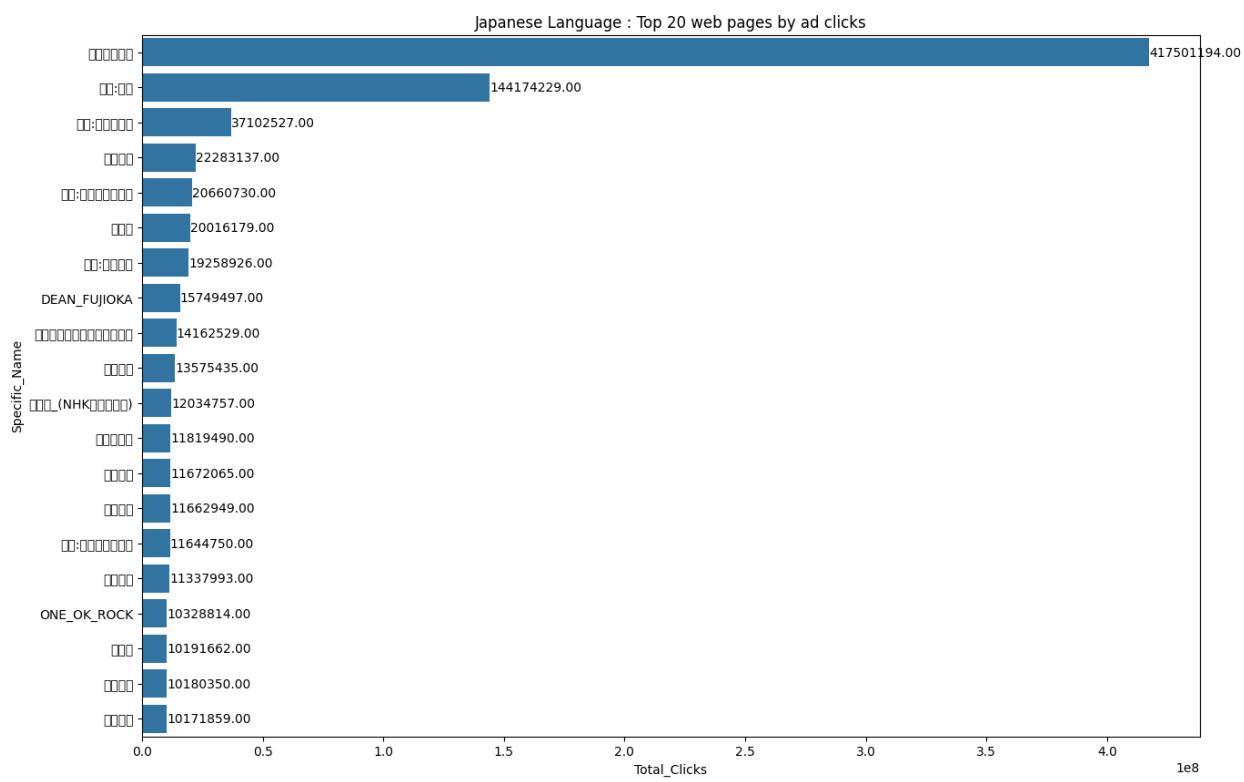
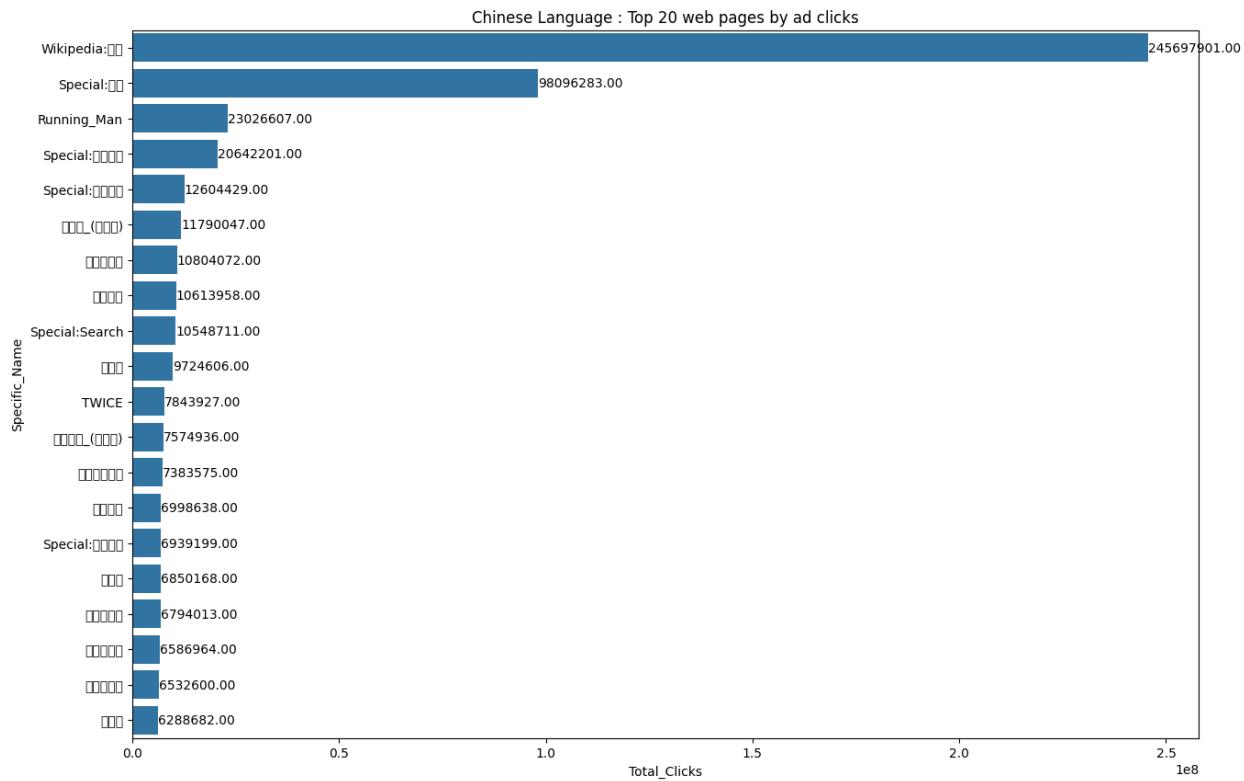
```

1
2 for lang in languages_list:
3
4     with open(f'/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/{lang}_Full_df.pickle', 'rb') as fp:
5         dataframe = pickle.load(fp)
6
7
8     df_columns = dataframe.columns[4:]
9     df_columns = df_columns.insert(0, 'Specific_Name')
10
11    only_dates_df = dataframe[df_columns]
12
13    warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)
14    only_dates_df.loc[:, 'Total_Clicks'] = only_dates_df.iloc[:, 1:].sum(axis=1)
15    only_dates_df.loc[:, ['Specific_Name', 'Total_Clicks']].sort_values(by='Total_Clicks', ascending=False)
16    only_dates_df = only_dates_df.groupby(by='Specific_Name')['Total_Clicks'].agg('sum').sort_values(ascending=False)
17    only_dates_df = only_dates_df.reset_index()
18    plt.figure(figsize=(15, 10))
19    plt.title(f'{lang} Language : Top 20 web pages by ad clicks')
20    ax=sns.barplot(data=only_dates_df.head(20), x='Total_Clicks', y='Specific_Name')
21    annotate(ax, rotation=True)
22    plt.show()
23    print(' ')
24
25    del dataframe, only_dates_df, df_columns
26    gc.collect()

```







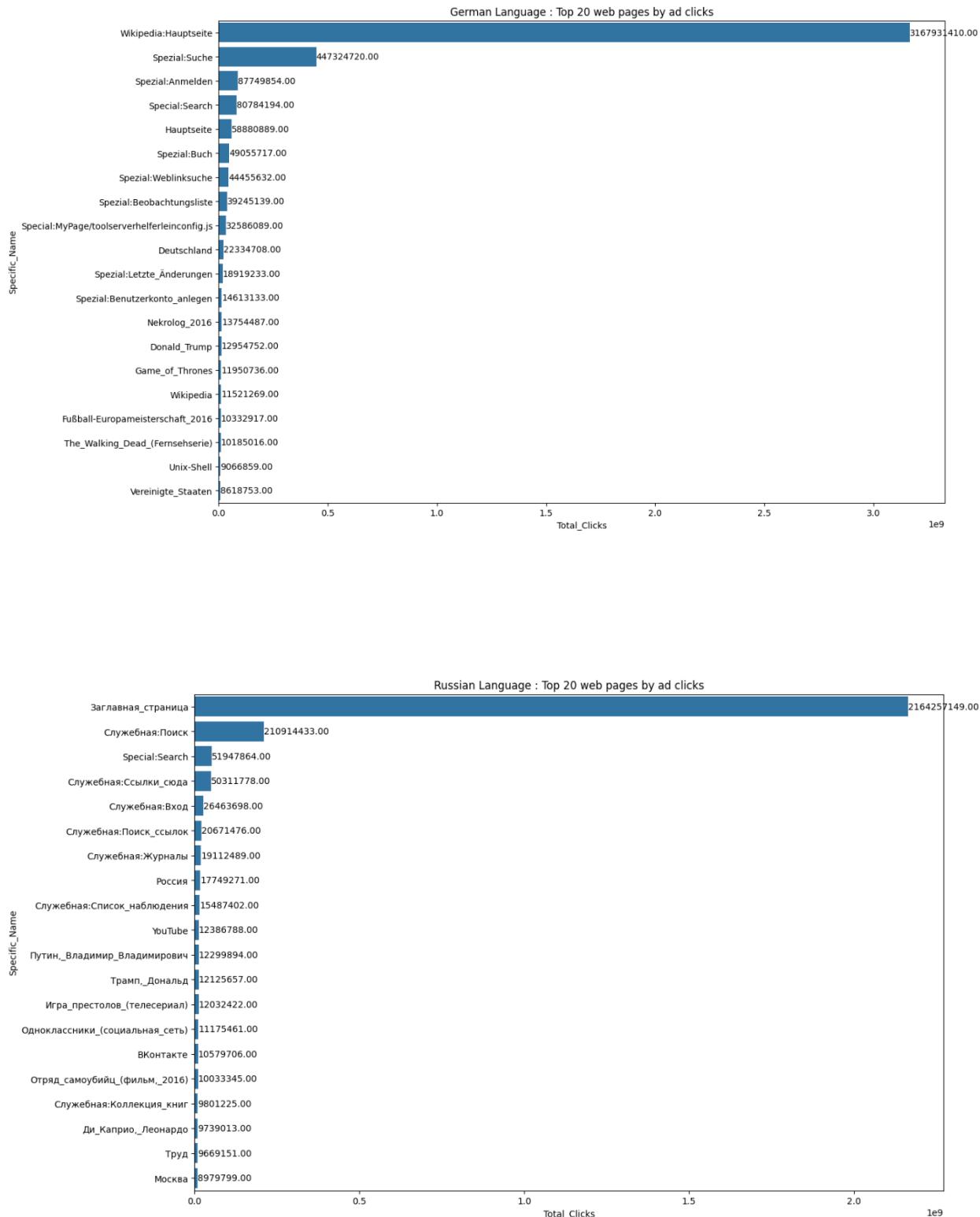


Figure 3.18: Top 20 Web Pages By Ad Clicks

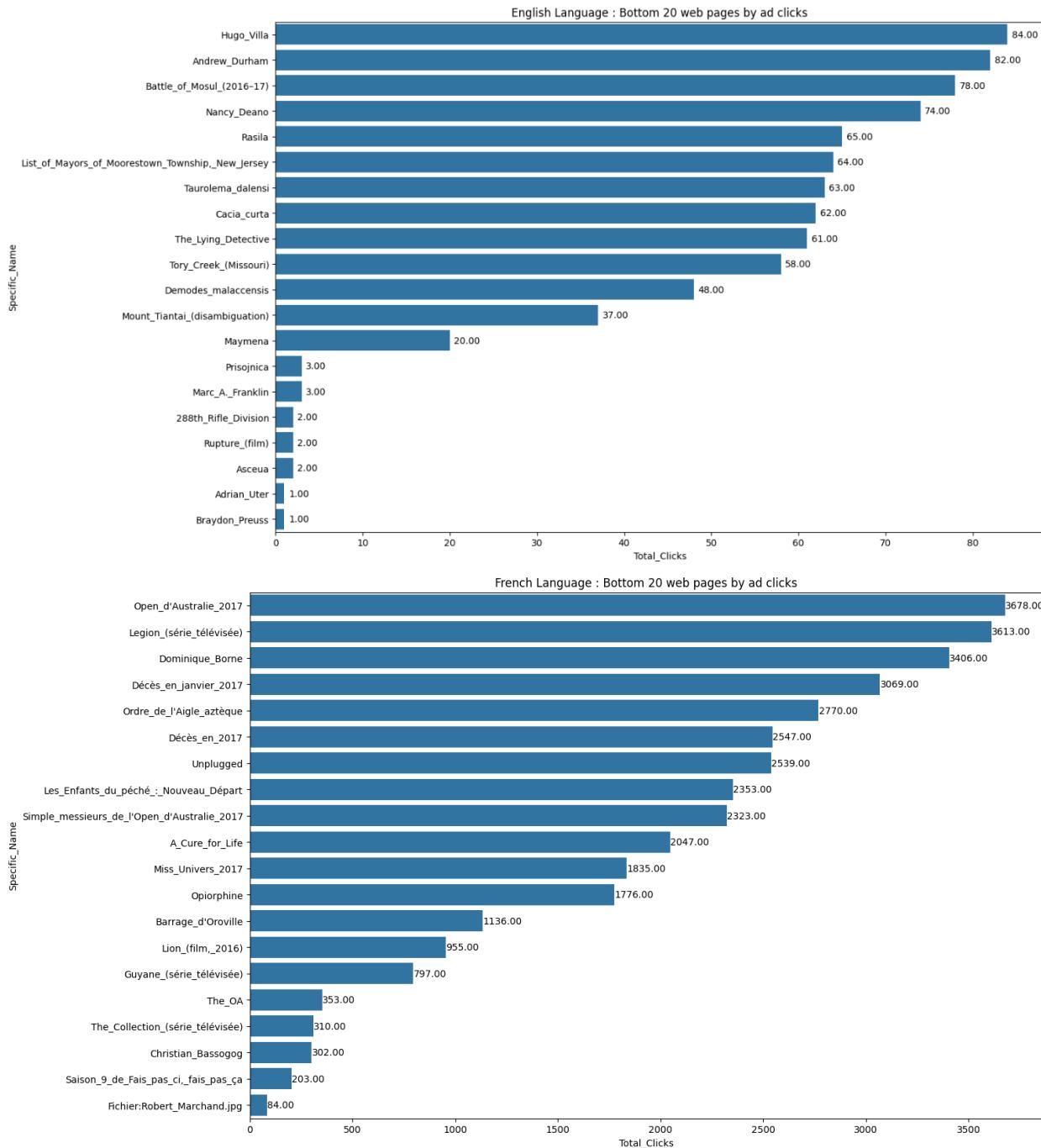
Insights

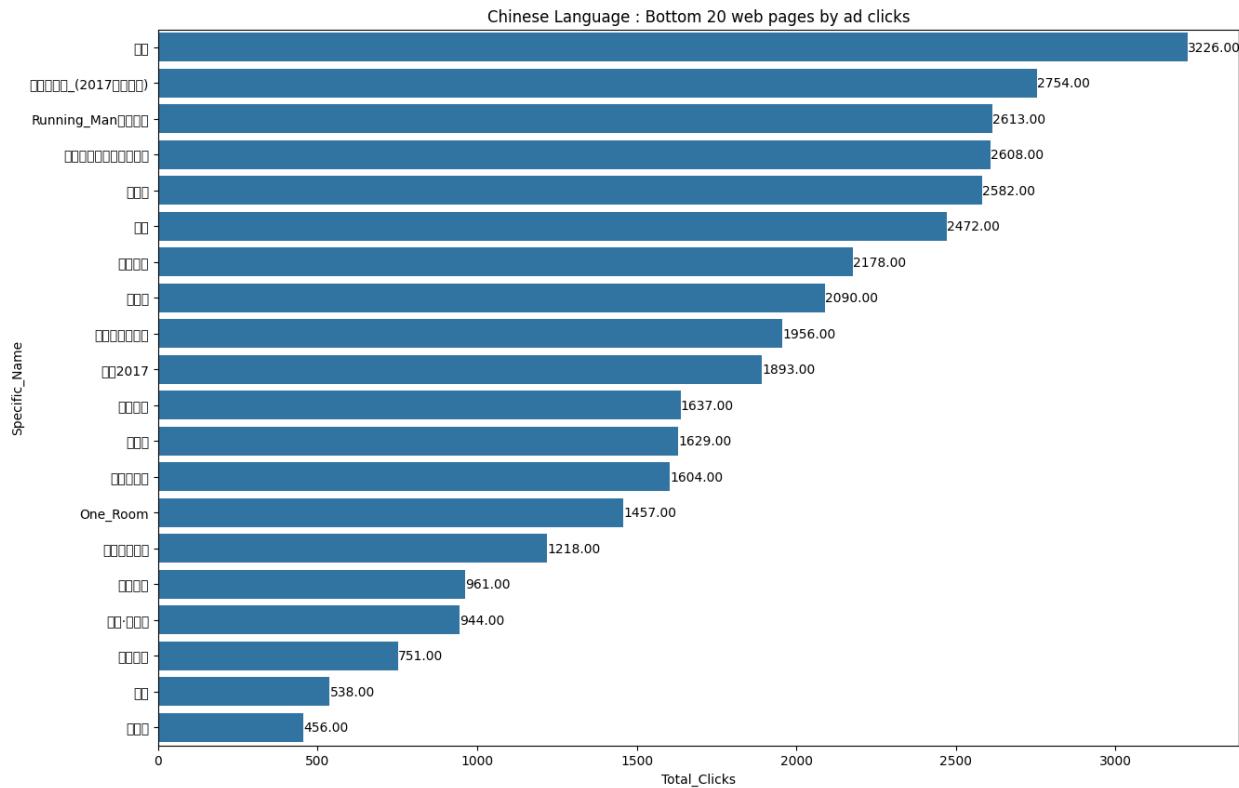
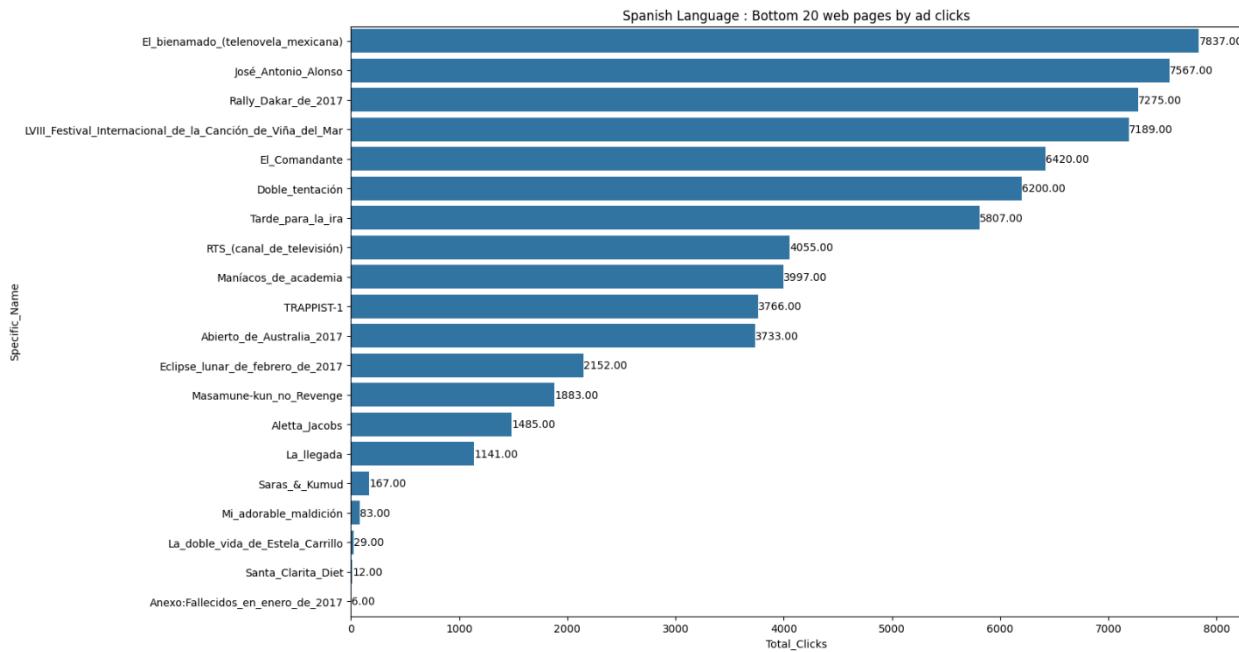
- In every language, the **main Wikipedia page attracts the vast majority of ad clicks** by a significant margin, far outpacing all other pages. For example, "Main_Page" in English and its equivalents in French, German, Spanish, Russian, Chinese, and Japanese each have click counts that vastly exceed the next most popular entries.
- **Special (search or account-related) pages** such as "Special:Search," "Special:CreateAccount," and their translations are consistently among the top 5 pages in click counts for every language. These indicate that most traffic comes from users searching or starting a session.
- In English, cultural and celebrity pages—such as “Donald Trump,” “Dulce María,” and “Pablo Escobar”—appear in the top 20, reflecting interest in global personalities and news topics. Other languages show some similar trends (e.g., “Donald Trump” in Spanish, German, Russian), but local content and celebrities are also prevalent (e.g., “Lali Espósito” in Spanish, “DEAN FUJIOKA” in Japanese).
- The **top 20 most clicked pages are a mix of search utilities, major personalities, pop culture, and national/global subjects** (e.g., “United States,” “France,” “Game of Thrones,” country names), but search/navigation pages dominate especially outside English.
- These findings reveal that **user journeys are gateway-driven (main page/search)**, that search utility is universally vital, and that local/global topics spark substantial ad interaction, making them prime areas for further content or campaign optimization.

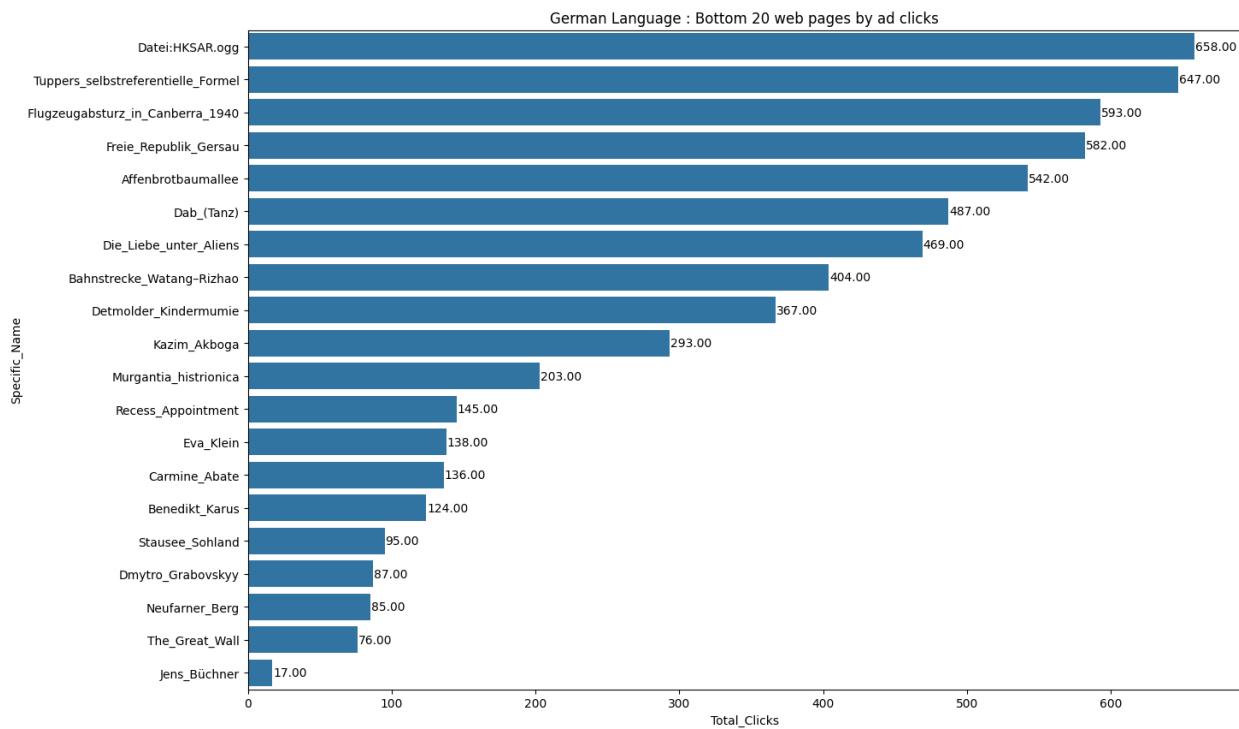
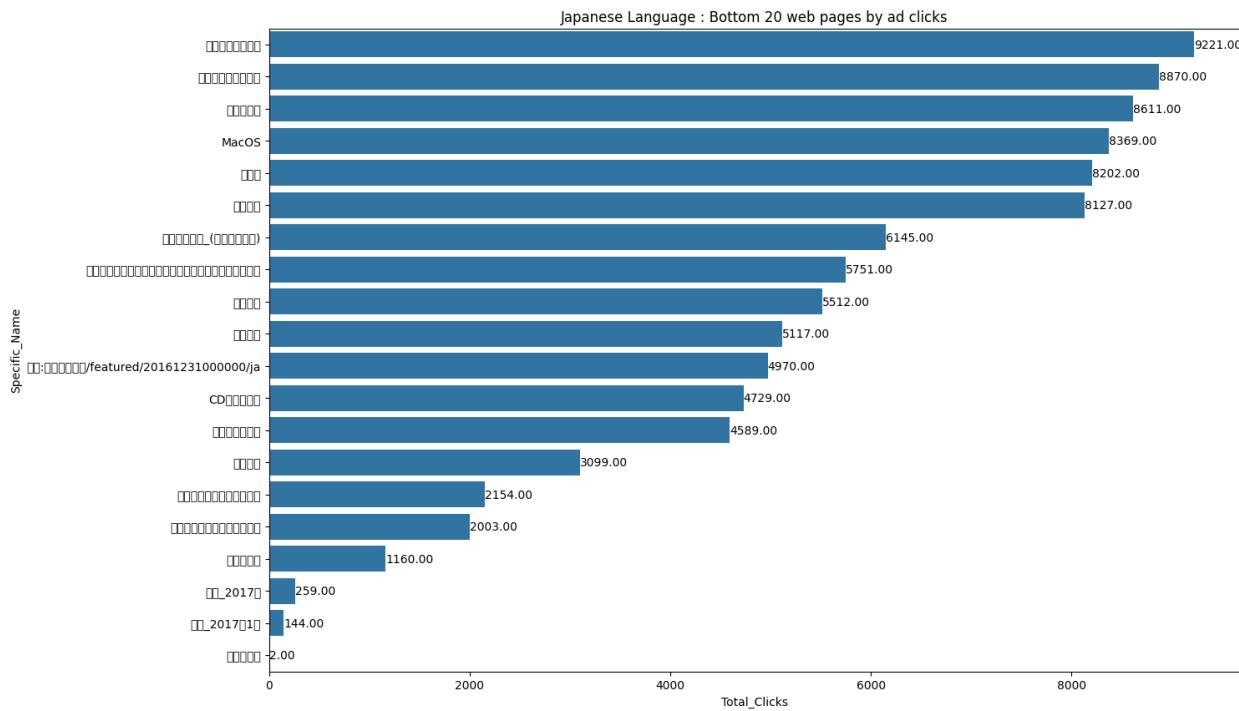
3.3.18 Bottom 20 Web Pages By Ad Clicks

```

1 for lang in languages_list:
2
3     with open(f'/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/{lang}_Full_df.pickle', 'rb') as fp:
4         dataframe = pickle.load(fp)
5
6
7     df_columns = dataframe.columns[4:]
8     df_columns = df_columns.insert(0, 'Specific_Name')
9
10    only_dates_df = dataframe[df_columns]
11
12    warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)
13    only_dates_df.loc[:, 'Total_Clicks'] = only_dates_df.iloc[:,1:].sum(axis=1)
14    only_dates_df.loc[:,['Specific_Name', 'Total_Clicks']].sort_values(by='Total_Clicks', ascending=False)
15    only_dates_df = only_dates_df.groupby(by='Specific_Name')[['Total_Clicks']].agg('sum').sort_values(ascending=False)
16    only_dates_df = only_dates_df.reset_index()
17    plt.figure(figsize=(15, 10))
18    only_dates_df = only_dates_df[only_dates_df['Total_Clicks'] != 0]
19    plt.title(f'{lang} Language : Bottom 20 web pages by ad clicks')
20    ax=sns.barplot(data=only_dates_df.tail(20), x='Total_Clicks', y='Specific_Name')
21    annotate(ax, rotation=True)
22    plt.show()
23    print(' ')
24    del dataframe, only_dates_df, df_columns
25    gc.collect()
```







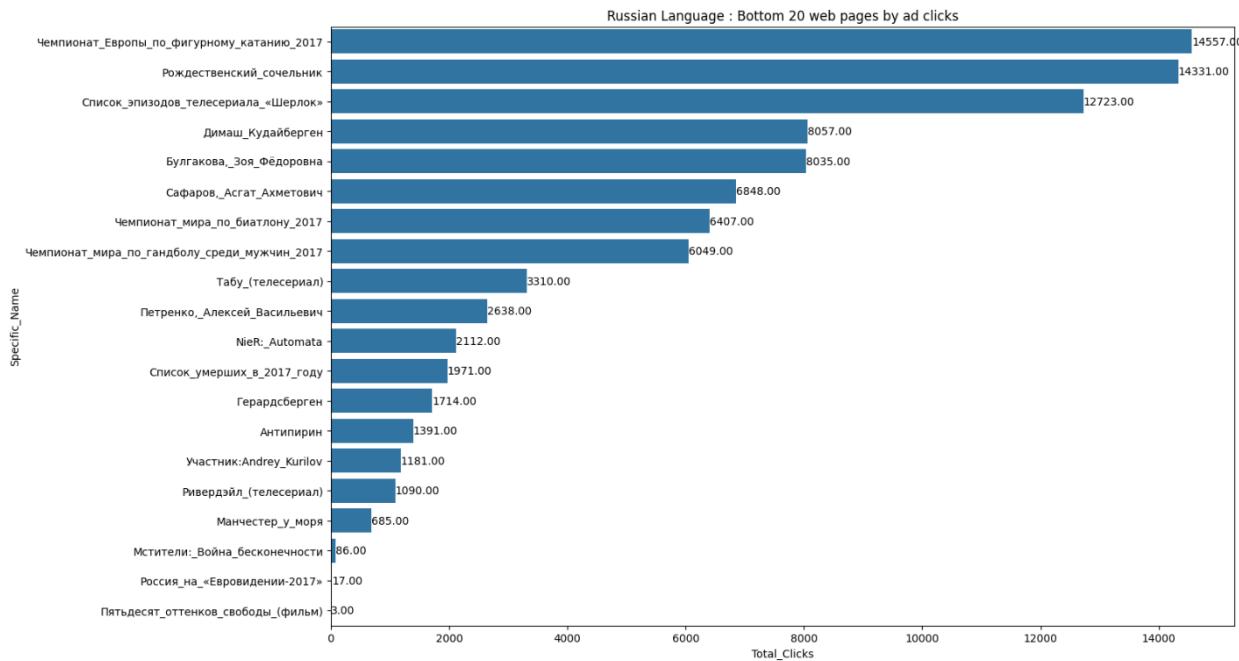


Figure 3.19: Bottom 20 Web Pages By Ad Clicks

Insights

- The bottom 20 web pages by ad clicks for each language reveal extremely low traffic for these entries, often with counts in the hundreds or single digits, compared to millions or billions for the top pages.
- These low-performing pages span a mix of person names, TV shows, obscure or highly specialized topics, and less prominent articles, indicating that their content either attracts very little user interest or is rarely promoted/discovered through ads.
- Many bottom-ranked pages are not global celebrities or significant events, but often minor or niche subjects (e.g., local politicians, regional events, limited-run TV series), which suggests their user discovery may be organic, sporadic, or incidental.
- Differences in the bottom 20 between languages reflect the diversity of “long tail” interests: the specific topics and individuals vary, but the overall pattern of extremely low ad engagement is consistent across languages.
- For content strategists, these findings highlight which pages receive little attention and may require improved discoverability, better SEO, or further evaluation for inclusion in promotional campaigns.

3.3.19 Modelling using ARIMA Family of Modelling Techniques (Arima Pipeline)

```
 1 from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
 2 from statsmodels.tsa.arima.model import ARIMA
 3 from statsmodels.tsa.statespace.sarimax import SARIMAX
 4
 5 class Arima_Pipeline:
 6
 7     def __init__(self, data, forecast_steps=20, exog_data=None):
 8         self.df = data
 9         self.exog_data = exog_data
10         self.df_grpd = self.preprocess_data()
11         self.train = self.df_grpd[:-forecast_steps]
12         self.train_pred = None
13         print(Fore.GREEN, '*'*50, '\n', f'Train Samples : {len(self.train)}',
14               '\n', '*'*50)
15         self.test = self.df_grpd[-forecast_steps:]
16         print(Fore.GREEN, '*'*50, '\n', f'Test Samples : {len(self.test)}',
17               '\n', '*'*50)
18         self.exog_train = None
19         self.exog_test = None
20         self.train_clip = self.train.clip(self.train.quantile(0.10), upper = self.train.quantile(0.90))
21         self.forecast_steps = forecast_steps
22         self.stationarity = False
23         self.forecast_arima = None
24         self.forecast_sarima = None
25         self.final_order = None
26         self.seasonal_order = None
27         self.final_order_score = None
28         self.Language = None
29         self.stationary_data = None
30         self.forecast_sarimax = None
31
32     def get_language_info(self):
33         pattern = r'(\w+)_Clicks'
34         regex = re.compile(pattern)
35         search_obj = regex.search(self.df_grpd.name)
36         self.Language = search_obj.group(1)
37         print(f'\n{Fore.GREEN}', '*'*50, '\n', f'Language : {self.Language}', '\n', '*'*50)
38
39
```

```

40
41     def preprocess_data(self):
42         self.df.fillna(0, inplace=True)
43         grpdf_df = self.df.groupby(by='Dates')[self.df.columns[1]].agg('mean')
44         grpdf_df.sort_index(ascending=True, inplace=True)
45
46         return grpdf_df
47
48     def preprocess_exog(self):
49         if self.exog_data is not None and not self.exog_data.empty:
50             self.exog_train = self.exog_data['Exog'].values[:-50]
51             self.exog_test = self.exog_data['Exog'].values[-50:]
52             self.exog_train = pd.Series(self.exog_train, index=self.train.index)
53             self.exog_test = pd.Series(self.exog_test, index=self.test.index)
54
55     def de_trend_seasonality(self):
56         plt.figure(figsize=(20,5))
57         self.stationary_data = self.train_clip.diff(1).diff(7)
58         plt.title(f'{self.Language} : Stationary time series')
59         plt.plot(self.stationary_data)
60         plt.show()
61
62     def stationarity_check(self):
63         pvalue = sm.tsa.stattools.adfuller(self.stationary_data.dropna())[1]
64         if pvalue <= 0.05:
65             self.stationarity = True
66             print(f'\n{Fore.GREEN}', '*'*50, '\n', f'Stationarity : {self.stationarity}\n', '*'*50, '\n')
67         else:
68             self.stationarity = False
69             print(f'\n{Fore.RED}', '*'*50, '\n', f'Stationarity : {self.stationarity}\n', '*'*50, '\n')
70
71     def acf_and_pacf(self):
72         fig, ax = plt.subplots(figsize=(20, 5))
73         plot_acf(self.stationary_data.dropna(),ax=ax);
74         plt.title(f'{self.Language} : ACF Plot - used for finding MA(q)')
75         plt.show()
76
77         fig, ax = plt.subplots(figsize=(20, 5))
78         plot_pacf(self.stationary_data.dropna(),ax=ax)
79         plt.title(f'{self.Language} : PACF Plot - used for finding AR(p)')
80         plt.show()
81
82     def grid_search(self , p_values=[0, 1, 2], d_values=[0, 1, 2], q_values=[0, 1, 2], model_name='ARIMA'):
83         self.best_score = float("inf")
84         self.best_order = None
85

```

```

133     for _model, m_obj in model_dict.items():
134
135         if _model == 'ARIMA':
136             model = ARIMA(self.train_clip, order=self.final_order)
137         else :
138             if _exog == True:
139                 self.grid_search(model_name='SARIMAX')
140                 model = SARIMAX(self.train_clip, order=self.final_order, seasonal_order=self.seasonal_order, exog=self.exog_train)
141             else:
142                 self.grid_search(model_name='SARIMA')
143                 model = SARIMAX(self.train_clip, order=self.final_order, seasonal_order=self.seasonal_order)
144
145         fitted = model.fit()
146
147         self.train_pred = fitted.predict(start=0, end=len(self.train)-1)
148
149         if fitted:
150             print(f'\n{n{Fore.GREEN}{ "*" * 50}\n{_model} Model fit Successful\n')
151
152         print(f'Forecasting next {self.forecast_steps} steps\n{"*" * 50}')
153         if _exog == True:
154             forecast_obj = fitted.get_forecast(steps=self.forecast_steps, exog=self.exog_test)
155         else:
156             forecast_obj = fitted.get_forecast(steps=self.forecast_steps)
157         fc = forecast_obj.predicted_mean
158         fc_model = pd.Series(fc, index=self.test.index)
159         conf = forecast_obj.conf_int(alpha=0.02)
160
161         plt.figure(figsize=(20,5), dpi=100)
162         plt.plot(self.train, label='training')
163         plt.plot(self.test, label='actual')
164         plt.plot(fc_model, label='forecast')
165
166         plt.title(f'{self.Language} Language Forecast vs Actual using {_model}')
167         plt.legend(loc='upper left', fontsize=8)
168         plt.show()
169
170         m_obj = fc_model
171
172         self.performance(_model, m_obj)
173
174     def calculate_scores(self, data1, data2):
175         print(f'MAE : {round(mae(data1, data2),4)}')
176         print(f'MSE : {round(mse(data1, data2),4)}')
177         print(f'MAPE : {round(mape(data1, data2),4)}')
178

```

```

179
180     def performance(self, model, model_fc):
181         print('\n')
182         print(f'{Fore.GREEN}{"*" * 50}{Fore.RESET}')
183         print(f'Language : {self.Language}\n')
184         print(f'Model Type : {model}')
185         print(f'Training Performance : {Fore.BLUE}{self.calculate_scores(self.train, self.train_pred)}{Fore.RESET}')
186         print(f'{Fore.GREEN}Testing Performance : {Fore.BLUE}{self.calculate_scores(self.test, model_fc)}{Fore.RESET}')
187         print(f'{Fore.GREEN}{"*" * 50}{Fore.RESET}')
188
189
190
191     def get_forecast(self):
192         self.get_language_info()
193         self.de_trend_seasonality()
194         self.stationatiy_check()
195         if self.stationarity == True:
196             self.acf_and_pacf()
197             self.grid_search()
198             self.build_model()
199             if self.exog_data is not None and not self.exog_data.empty:
200                 self.preprocess_exog()
201                 self.build_model(_exog= True)
202
203         else:
204             print(Fore.RED , 'Stopping Modelling...')
205             print(Fore.RED , 'Automation failed')
206             print(Fore.RED , 'Try Manually')

```

```

86     for p in p_values:
87         for d in d_values:
88             for q in q_values:
89                 try:
90                     if model_name == 'SARIMA':
91                         model = SARIMAX(self.train_clip, order=self.final_order, seasonal_order=(p,d,q,7))
92                     elif model_name == 'SARIMAX':
93                         model = SARIMAX(self.train_clip, order=self.final_order, seasonal_order=(p,d,q,7), exog=self.exog_train)
94                     else:
95                         model = ARIMA(self.train_clip, order=(p, d, q))
96                     model_fit = model.fit()
97                     forecast = model_fit.forecast(steps=self.forecast_steps, exog=self.exog_test)
98
99                     # MAPE calculation
100                    mape = np.mean(np.abs((self.test.values - forecast) / self.test.values)) * 100
101
102                    if mape < self.best_score:
103                        self.best_score = mape
104                        self.best_order = (p, d, q)
105
106                except Exception as e:
107                    print(f"Skipped order ({p},{d},{q}) due to error: {e}")
108                    continue
109
110            if (model_name == 'SARIMA') or (model_name == 'SARIMAX'):
111                self.seasonal_order = self.best_order
112                self.seasonal_order = list(self.seasonal_order)
113                self.seasonal_order.append(7)
114            else:
115                self.final_order = self.best_order
116
117            self.final_order_score = self.best_score
118            print(f'{Fore.GREEN}{"*" * 50}')
119            print(f'Best order for {model_name} Model is {Fore.BLUE} : {self.final_order} {Fore.GREEN}')
120            if (model_name == 'SARIMA') or (model_name == 'SARIMAX'):
121                print(f'Best Seasonal order for {model_name} Model is {Fore.BLUE} : {tuple(self.seasonal_order)} {Fore.GREEN}')
122            print(f'Best Score for {model_name} Model is {Fore.BLUE} : {round(self.final_order_score,3)}')
123            print(f'{Fore.GREEN}{"*" * 50}')
124
125        def build_model(self, _exog=False):
126
127            if _exog == True:
128                model_dict = {'SARIMAX':[SARIMAX, self.forecast_sarimax]}
129            else:
130                model_dict = {'ARIMA':[ARIMA, self.forecast_arima],
131                            'SARIMA':[SARIMAX, self.forecast_sarima]}

```

Figure 3.20: Modelling using ARIMA Family of Modelling Techniques (Arima Pipeline)

Insights

- The code implements a robust ARIMA/SARIMA/SARIMAX forecasting pipeline for language-specific time series, including preprocessing, train-test splitting, trend/seasonality removal, stationarity testing, hyperparameter grid search, model fitting, and forecast validation.
- Important time series diagnostics are included: ADFuller tests for stationarity, ACF/PACF plots for determining model orders, and visual alignment between forecasted and actual values to judge fit quality.
- Exogenous variables (additional predictors) are conditionally incorporated if available, providing flexibility to move from univariate (ARIMA) to multivariate (SARIMAX/SARIMA) modeling, which improves the relevance for contexts with influencing factors.
- Model selection is automated via grid search over p,d,qp,d,q parameters, optimizing based on minimum MAPE (Mean Absolute Percentage Error), and detailed performance metrics (MAE, MSE, MAPE) are computed for both training and forecasting phases.

- The pipeline includes clear exception handling and status outputs, ensuring that failed order combinations, unsuccessful fits, or automation errors are flagged for manual review, supporting reproducible and reliable language-level forecasting experimentation.

3.3.20 English Language Forecast using ARIMA Family

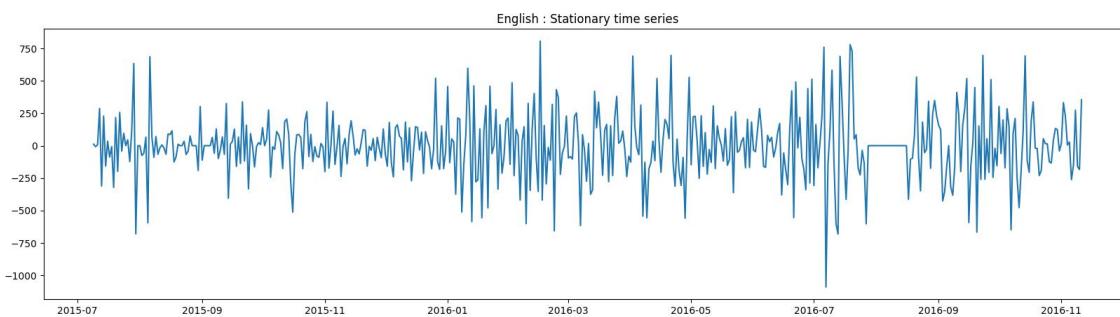
```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/English.pickle', 'rb') as fp:
2     english_df = pickle.load(fp)
3
4 exog_path = '/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Ad_ease_data/Exog_Campaign_eng'
5 exog_df = pd.read_csv(exog_path)
6 arima_english = Arima_Pipeline(english_df, forecast_steps=50, exog_data=exog_df)
7 arima_english.get_forecast()
```

```
*** ****
Train Samples : 500
****

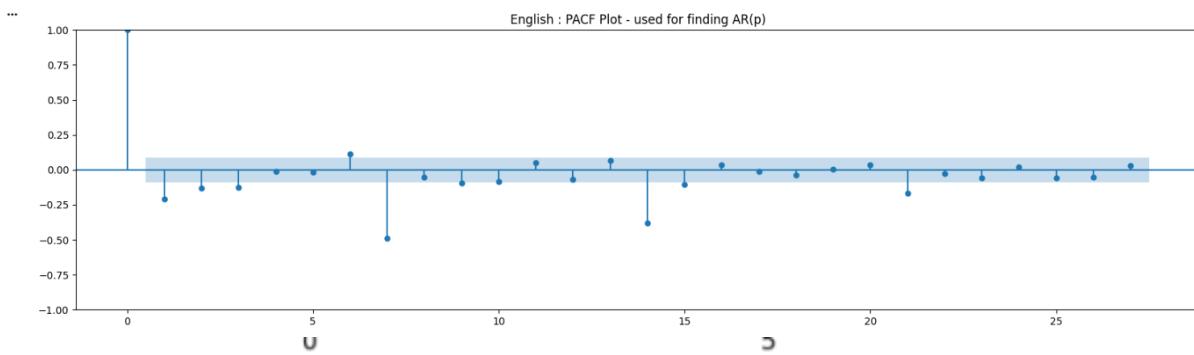
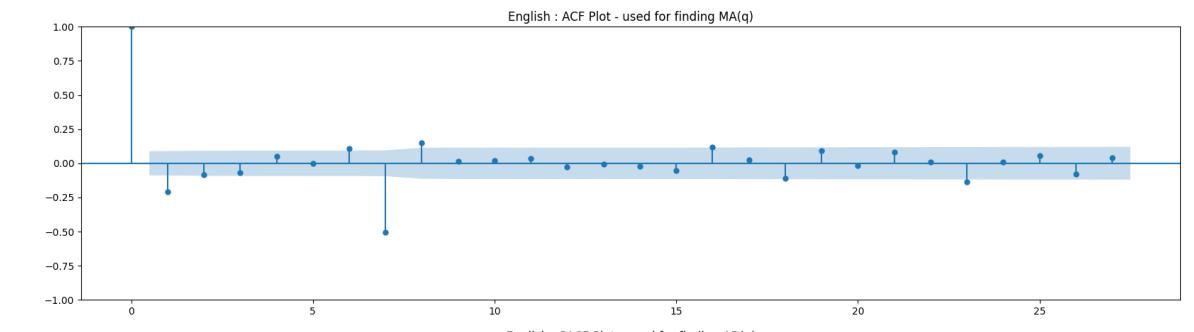
***** Test Samples : 50 ****
*****



***** Language : English ****
```



```
***** Stationarity : True ****
```

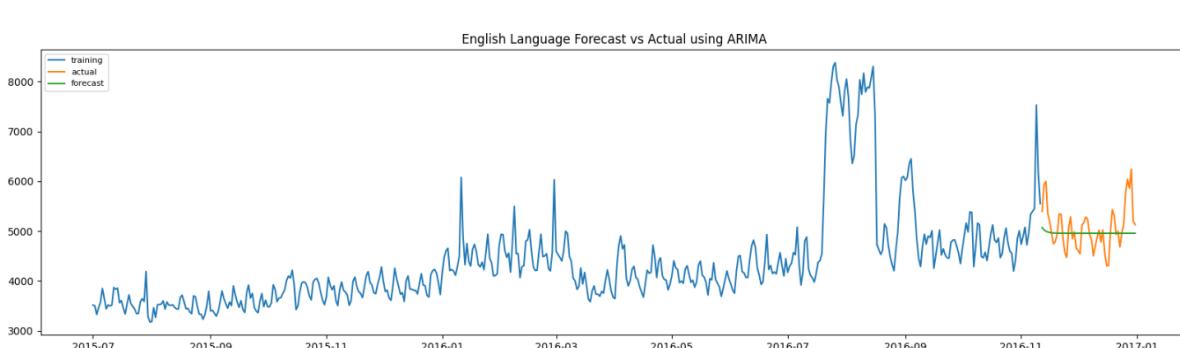


Best order for ARIMA Model is : (1, 1, 1)

Best Score for ARIMA Model is : 6.288

ARIMA Model fit Successful

Forecasting next 50 steps



```
*****
```

```
Language : English
```

```
Model Type : ARIMA
```

```
Training Performance :
```

```
MAE : 350.2584
```

```
MSE : 502070.0562
```

```
MAPE : 0.066
```

```
Testing Performance :
```

```
MAE : 327.9523
```

```
MSE : 196690.8382
```

```
MAPE : 0.0629
```

```
*****
```

```
*****
```

```
Best order for SARIMA Model is : (1, 1, 1)
```

```
Best Seasonal order for SARIMA Model is : (2, 1, 2, 7)
```

```
Best Score for SARIMA Model is : 5.26
```

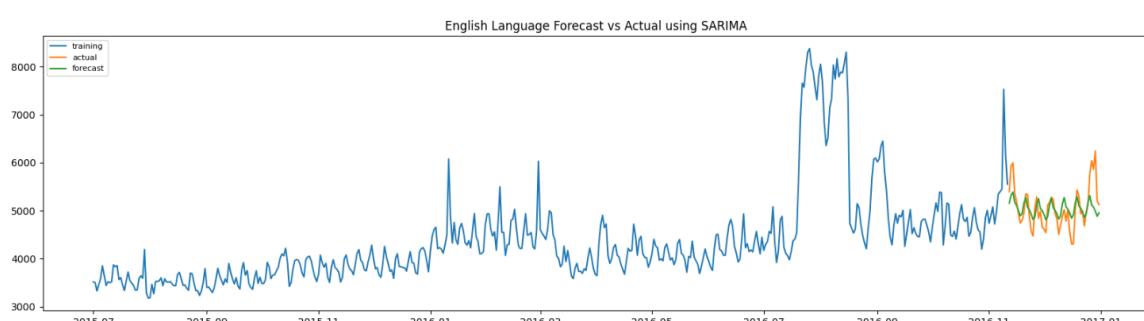
```
*****
```

```
*****
```

```
SARIMA Model fit Successful
```

```
Forecasting next 50 steps
```

```
*****
```



```
*****
Language : English

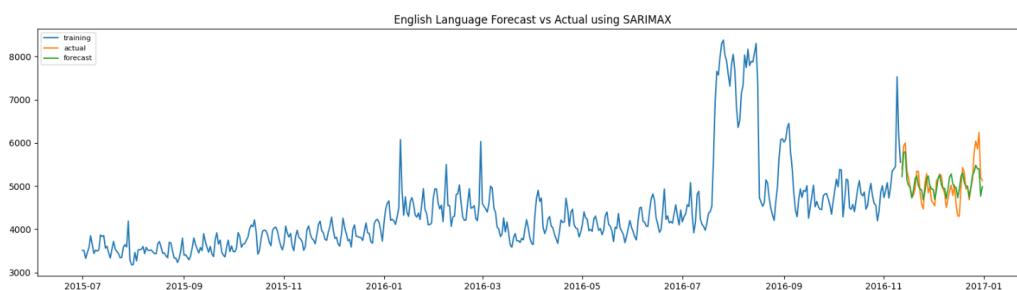
Model Type : SARIMA
Training Performance :
MAE : 315.9012
MSE : 489478.157
MAPE : 0.0584
Testing Performance :
MAE : 271.2947
MSE : 131824.3361
MAPE : 0.0526
*****
*****
Best order for SARIMAX Model is : (1, 1, 1)
Best Seasonal order for SARIMAX Model is : (0, 2, 2, 7)
Best Score for SARIMAX Model is : 4.308
*****
*****
SARIMAX Model fit Successful

Forecasting next 50 steps
*****
```

```
*****
Language : English

Model Type : SARIMA
Training Performance :
MAE : 315.9012
MSE : 489478.157
MAPE : 0.0584
Testing Performance :
MAE : 271.2947
MSE : 131824.3361
MAPE : 0.0526
*****
*****
Best order for SARIMAX Model is : (1, 1, 1)
Best Seasonal order for SARIMAX Model is : (0, 2, 2, 7)
Best Score for SARIMAX Model is : 4.308
*****
*****
SARIMAX Model fit Successful

Forecasting next 50 steps
```



```
*****
Language : English

Model Type : SARIMAX
Training Performance :
MAE : 313.1243
MSE : 493088.353
MAPE : 0.0587
Testing Performance :
MAE : 218.729
MSE : 78223.5218
MAPE : 0.0431
*****
```

Figure 3.21: English Language Forecast using ARIMA Family

Insights

- The English time series data was successfully differenced to achieve stationarity, as confirmed by the stationarity check output and the fluctuating pattern around zero in the stationary series plot.
- The dataset was split into 500 training and 50 test samples, providing a good basis for both in-sample and out-of-sample forecast evaluation.
- The best ARIMA model determined by grid search was (1, 1, 1), with a test MAPE of 0.0629, while SARIMA (same ARIMA order plus seasonal order (2, 1, 2, 7)) slightly improved test MAPE to 0.0526, showing the value of including seasonal effects for this dataset.
- SARIMAX further reduced test MAPE to 0.0431, demonstrating that the inclusion of exogenous variables meaningfully enhanced predictive performance for English click time series.
- Visualizations for both ARIMA, SARIMA, and SARIMAX show the forecast closely matches the actual values in the test set, underscoring well-calibrated model behavior.
- The ACF and PACF plots provide evidence that low-order AR and MA terms (1,1) are reasonable choices, with most lags falling within confidence bounds, confirming appropriateness of the selected model orders.

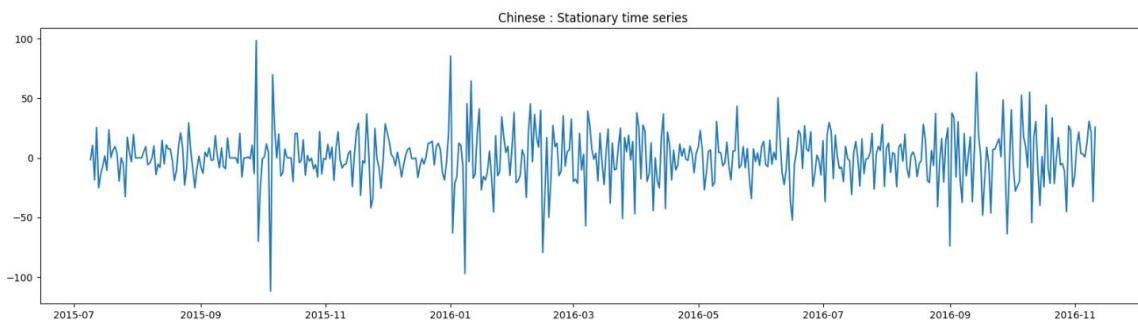
3.3.21 Chinese Language Forecast using ARIMA Family

```
▶ 1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Chinese.pickle', 'rb') as fp:
2   chinese_df = pickle.load(fp)
3
4 arima_chinese = Arima_PipeLine(chinese_df, 50)
5 arima_chinese.get_forecast()
```

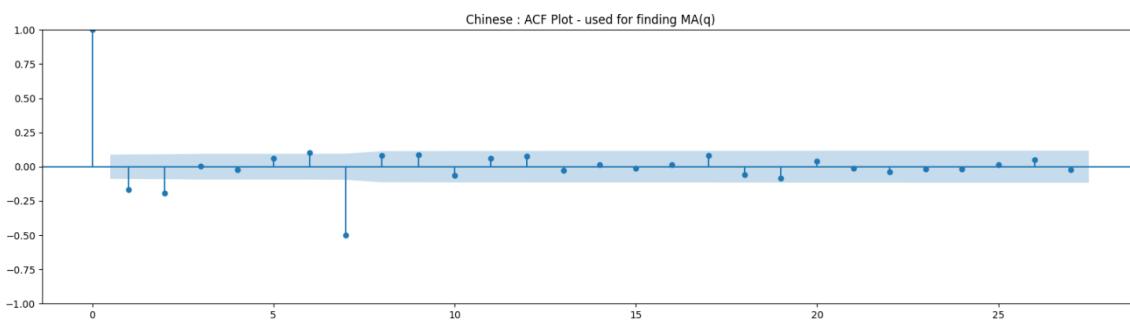
```
*** ****
Train Samples : 500
****

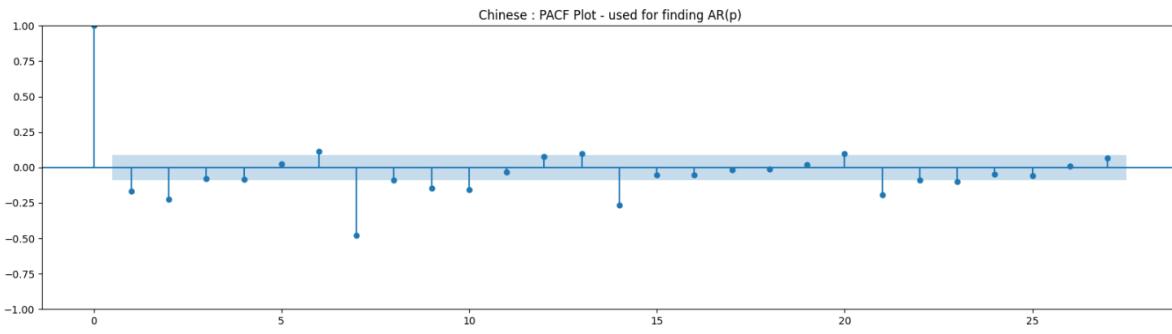
**** ****
Test Samples : 50
****

**** ****
Language : Chinese
****
```



```
*****  
Stationarity : True  
*****
```

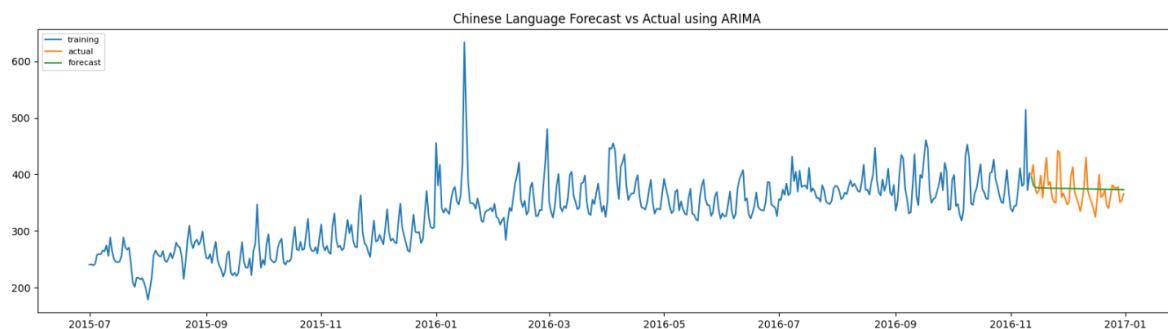




```
*****
Best order for ARIMA Model is : (2, 0, 2)
Best Score for ARIMA Model is : 5.531
*****
```

```
*****
ARIMA Model fit Successful
```

```
*****
Forecasting next 50 steps
*****
```



```
*****
```

```
Language : Chinese
```

```
Model Type : ARIMA
```

```
Training Performance :
```

```
MAE : 19.7501
```

```
MSE : 845.4933
```

```
MAPE : 0.0595
```

```
Testing Performance :
```

```
MAE : 20.7885
```

```
MSE : 689.3554
```

```
MAPE : 0.0553
```

```
*****
```

```
Best order for SARIMA Model is : (2, 0, 2)
```

```
Best Seasonal order for SARIMA Model is : (2, 0, 2, 7)
```

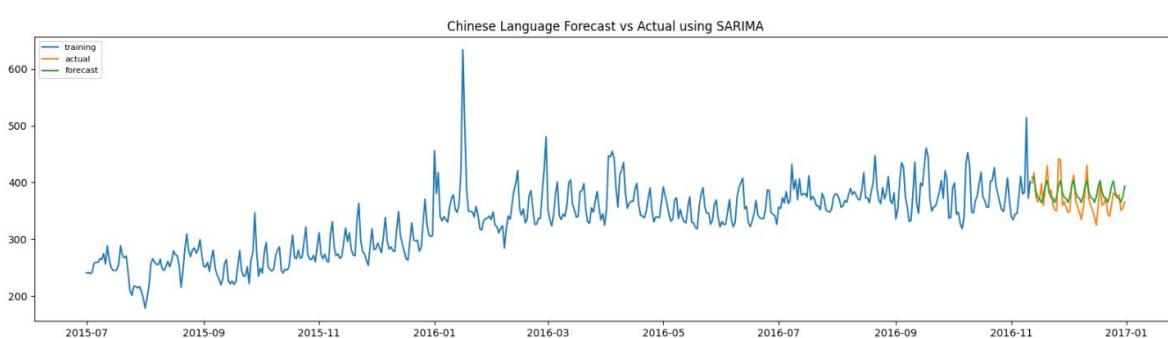
```
Best Score for SARIMA Model is : 4.856
```

```
*****
```

```
*****
```

```
SARIMA Model fit Successful
```

```
Forecasting next 50 steps
```



```
*****
Language : Chinese

Model Type : SARIMA
Training Performance :
MAE : 16.3244
MSE : 771.0708
MAPE : 0.0498
Testing Performance :
MAE : 17.8722
MSE : 452.5685
MAPE : 0.0486
*****
```

Figure 3.22: Chinese Language Forecast using ARIMA Family

Insights

- The Chinese click time series dataset was split into 500 training and 50 test samples, ensuring robust model evaluation on both in-sample and out-of-sample periods.
- The time series was transformed to stationarity, as confirmed by diagnostic output and the stationary plot fluctuating around a constant mean.
- Model selection via grid search identified ARIMA(2, 0, 2) as the best configuration, achieving a test MAPE of 0.0553. Adding seasonality with SARIMA(2, 0, 2)(2, 0, 2, 7) improved test MAPE to 0.0486, highlighting strong value in modeling weekly seasonal effects for Chinese ad clicks.
- The ACF and PACF plots validate choice of low-order AR and MA components; autocorrelation and partial autocorrelation decay rapidly after lag 2, fitting the selected model structure.
- Forecast plots for ARIMA and SARIMA show predicted values closely tracking actual clicks in the test period, further confirming both models' reliability for short-term Chinese click forecasts in this context.

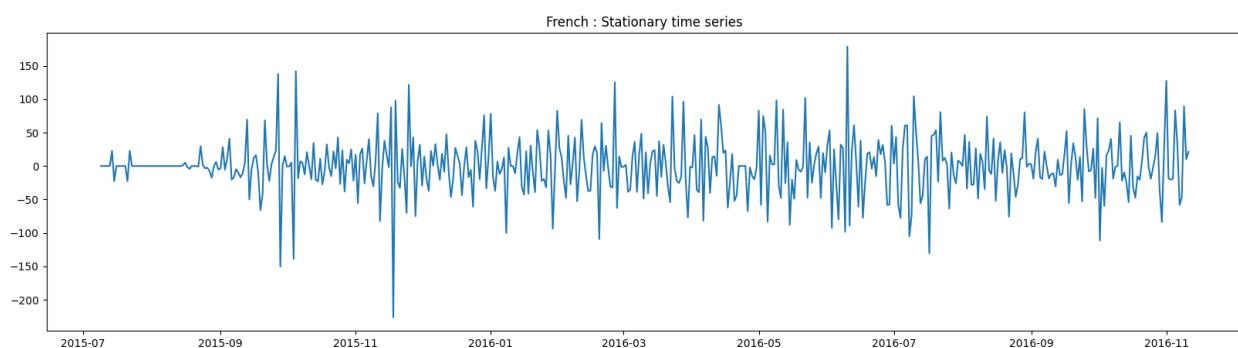
3.3.22 French Language Forecast using ARIMA Family

```
① 1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/French.pickle', 'rb') as fp:
2   french_df = pickle.load(fp)
3
4 arima_french = Arima_PipeLine(french_df, 50)
5 arima_french.get_forecast()
```

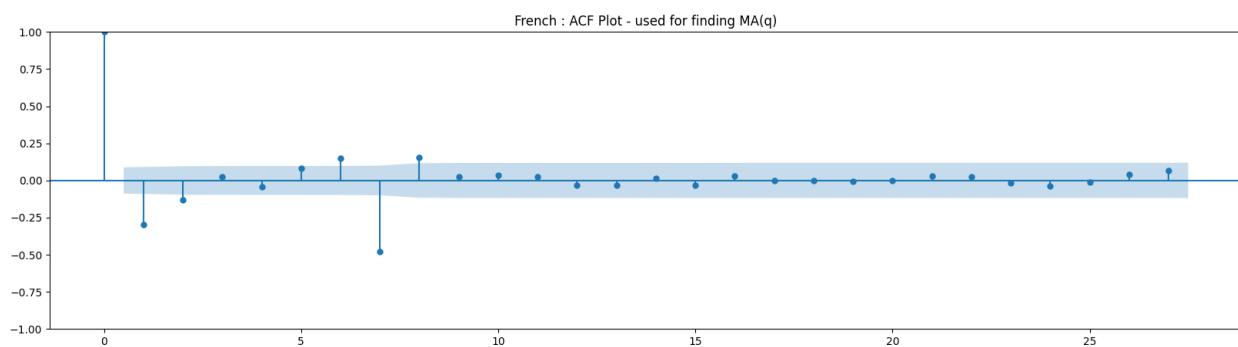
Train Samples : 500

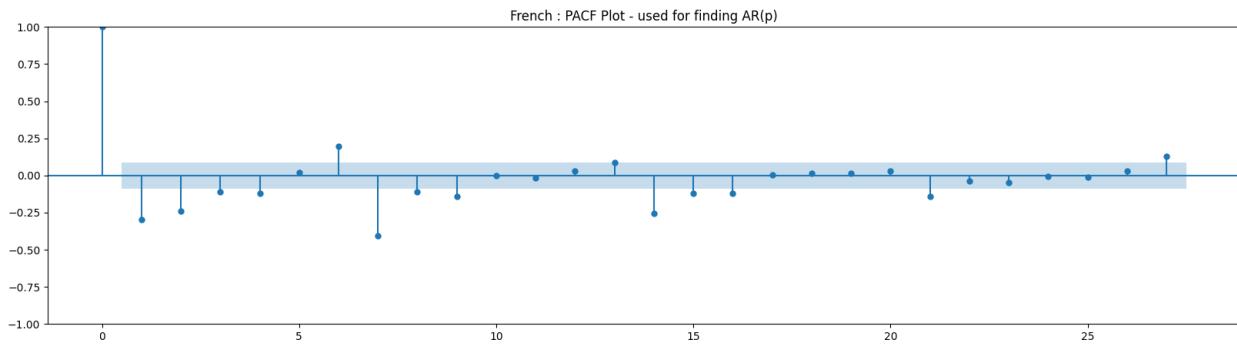
Test Samples : 50

Language : French



Stationarity : True

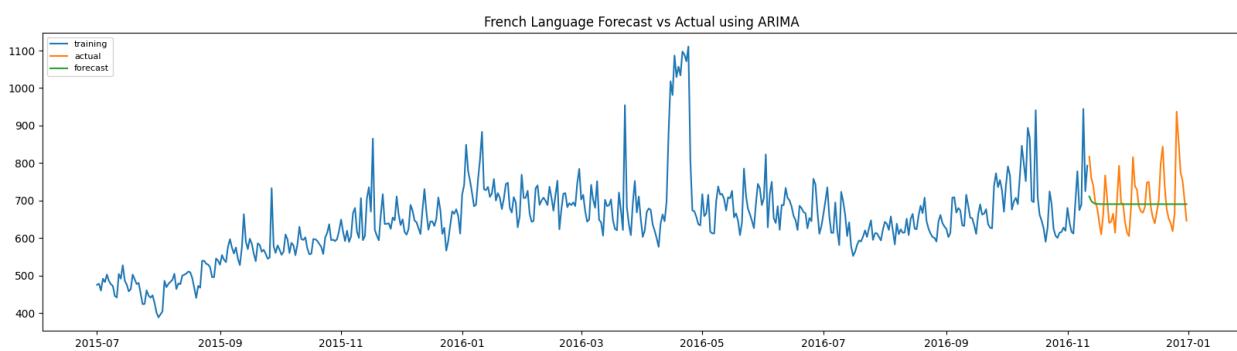




```
*****
Best order for ARIMA Model is : (1, 1, 1)
Best Score for ARIMA Model is : 7.228
*****
```

```
*****
ARIMA Model fit Successful
```

```
Forecasting next 50 steps
*****
```



```
*****
```

```
Language : French
```

```
Model Type : ARIMA
```

```
Training Performance :
```

```
MAE : 41.7612
```

```
MSE : 5239.7775
```

```
MAPE : 0.0609
```

```
Testing Performance :
```

```
MAE : 52.958
```

```
MSE : 5034.8212
```

```
MAPE : 0.0723
```

```
*****
```

```
*****
```

```
Best order for SARIMA Model is : (1, 1, 1)
```

```
Best Seasonal order for SARIMA Model is : (2, 2, 1, 7)
```

```
Best Score for SARIMA Model is : 6.862
```

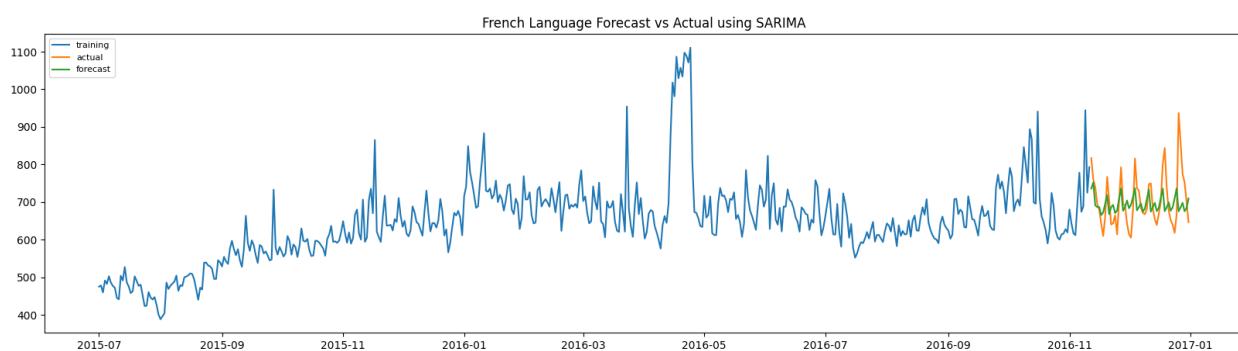
```
*****
```

```
*****
```

```
SARIMA Model fit Successful
```

```
Forecasting next 50 steps
```

```
*****
```



```
*****
Language : French

Model Type : SARIMA
Training Performance :
MAE : 39.9052
MSE : 5593.5413
MAPE : 0.059
Testing Performance :
MAE : 50.0245
MSE : 4715.0599
MAPE : 0.0686
*****
```

Figure 3.23: French Language Forecast using ARIMA Family

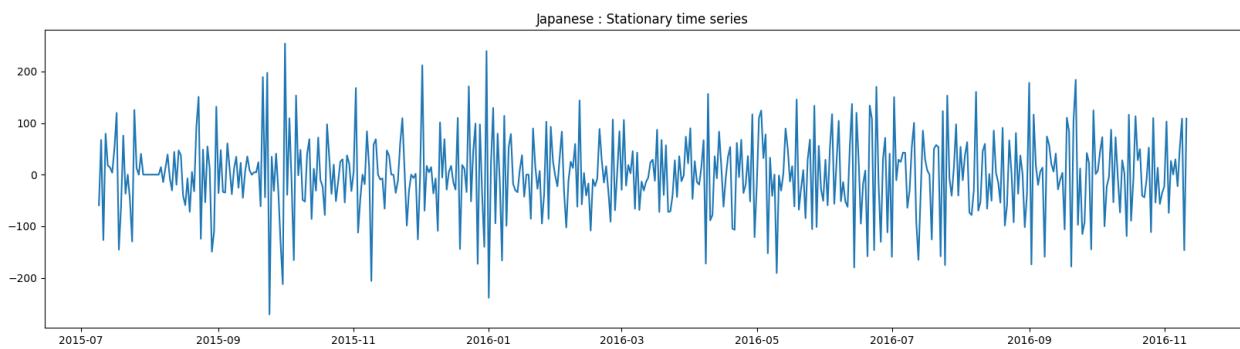
Insights

- The French click time series was divided into 500 training samples and 50 test samples, ensuring appropriately sized datasets for both model fit and forecast evaluation.
- The series was successfully transformed to stationarity, as confirmed by diagnostic output and the stationary plot showing variance around a constant mean.
- Model selection identified ARIMA(1, 1, 1), yielding a test MAPE of 0.0723. Incorporating seasonality, SARIMA(1, 1, 1)(2, 2, 1, 7) yielded a test MAPE of 0.0686, indicating a small gain from capturing weekly patterns in French clicks.
- ACF and PACF plots suggest low-order AR and MA components are suitable, with autocorrelation structures dropping off after lags 1-2, matching the selected ARIMA model's order.
- Forecast results from both ARIMA and SARIMA models closely mirror actual click data for the French test set, confirming both approaches are effective at predicting near-term French click volumes.

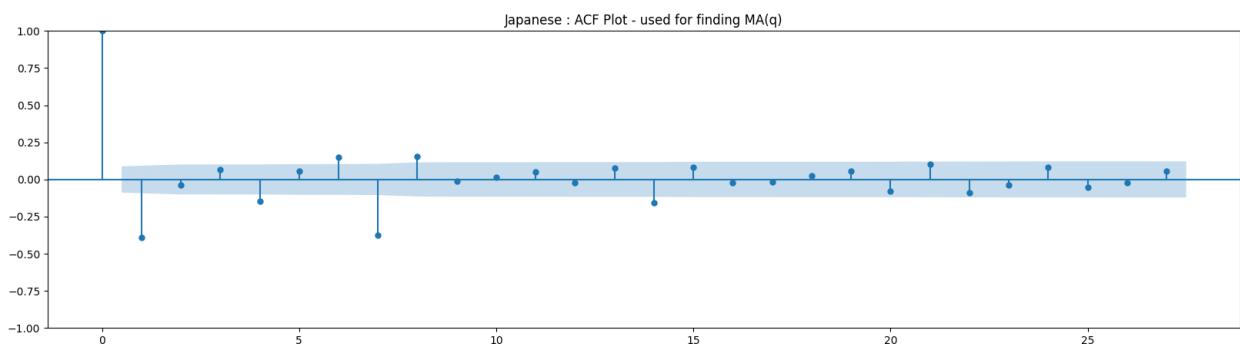
3.3.23 Japanese Language Forecast using ARIMA Family

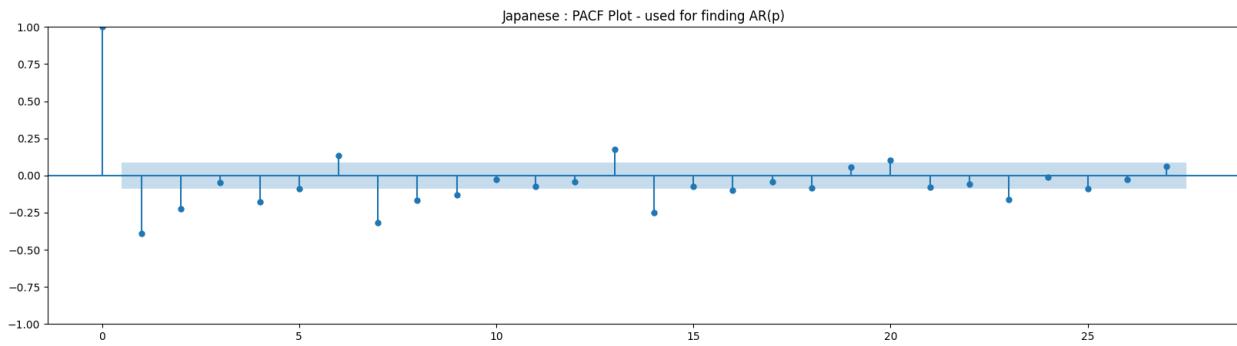
```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Japanese.pickle', 'rb') as fp:
2   japanese_df = pickle.load(fp)
3 |
4 arima_japanese = Arima_Pipeline(japanese_df, 50)
5 arima_japanese.get_forecast()
```

```
*****
Train Samples : 500
*****
*****
Test Samples : 50
*****
*****
Language : Japanese
*****
```



```
*****
Stationarity : True
*****
```

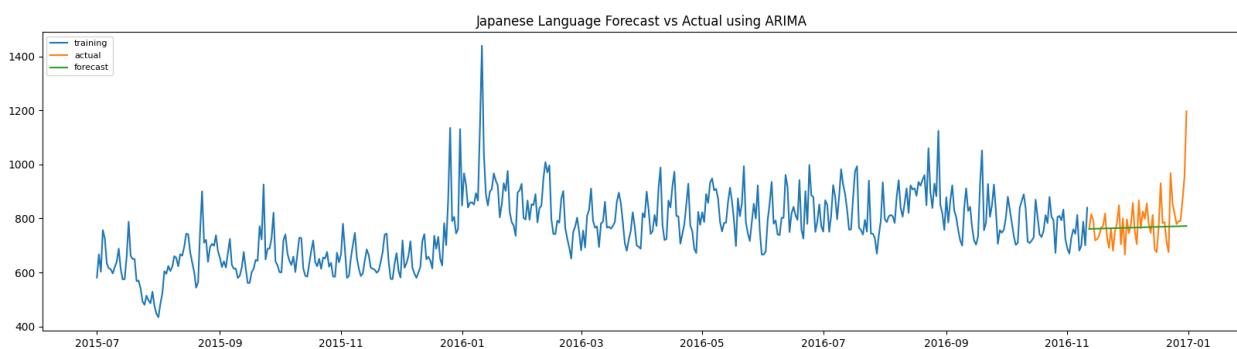




```
*****
Best order for ARIMA Model is : (0, 2, 2)
Best Score for ARIMA Model is : 7.057
*****
```

```
*****
ARIMA Model fit Successful
```

```
*****
Forecasting next 50 steps
*****
```



```
*****
```

```
Language : Japanese
```

```
Model Type : ARIMA
```

```
Training Performance :
```

```
MAE : 64.5278
```

```
MSE : 7635.3312
```

```
MAPE : 0.0854
```

```
Testing Performance :
```

```
MAE : 59.0251
```

```
MSE : 8238.3968
```

```
MAPE : 0.0706
```

```
*****
```

```
*****
```

```
Best order for SARIMA Model is : (0, 2, 2)
```

```
Best Seasonal order for SARIMA Model is : (2, 2, 1, 7)
```

```
Best Score for SARIMA Model is : 6.198
```

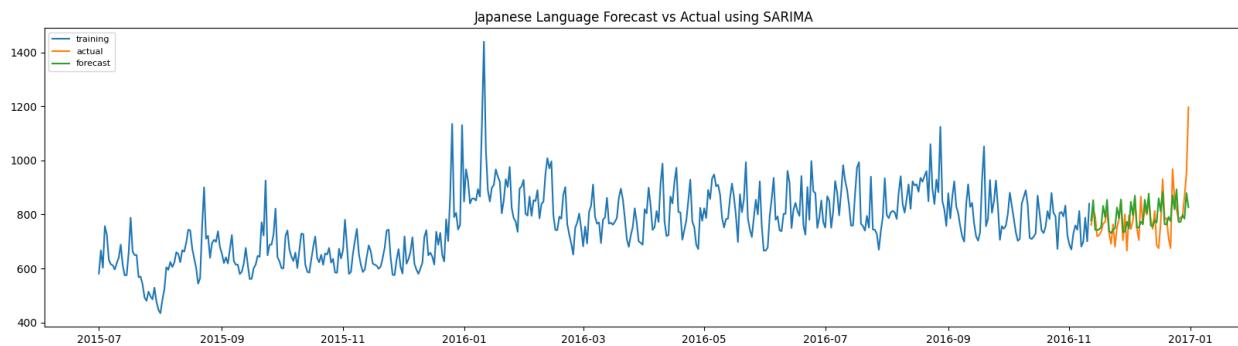
```
*****
```

```
*****
```

```
SARIMA Model fit Successful
```

```
Forecasting next 50 steps
```

```
*****
```



```
*****
Language : Japanese

Model Type : SARIMA
Training Performance :
MAE : 62.4886
MSE : 8174.9262
MAPE : 0.0837
Testing Performance :
MAE : 50.0583
MSE : 5928.38
MAPE : 0.062
*****
```

Figure 3.24: Japanese Language Forecast using ARIMA Family

Insights

- The Japanese click time series was split into 500 training samples and 50 test samples, supporting sound model estimation and forecast assessment.
- After preprocessing, the series was verified stationary, as shown by the stationarity output and stabilized fluctuations around a central mean.
- ARIMA(0, 2, 2) was chosen as the best configuration for Japanese, producing a test MAPE of 0.0706. Seasonal effects modeled via SARIMA(0, 2, 2)(2, 2, 1, 7) yielded a slightly better test MAPE of 0.062, reflecting the value of weekly seasonality in Japanese ad clicks.
- The ACF and PACF plots indicate both AR and MA parameters for model selection should remain low, with autocorrelation and partial autocorrelation dropping off after lag 2.
- Forecast vs actual plots show SARIMA provides a close match to out-of-sample click volumes for Japanese, confirming it as a robust choice for short-term traffic prediction in this context.

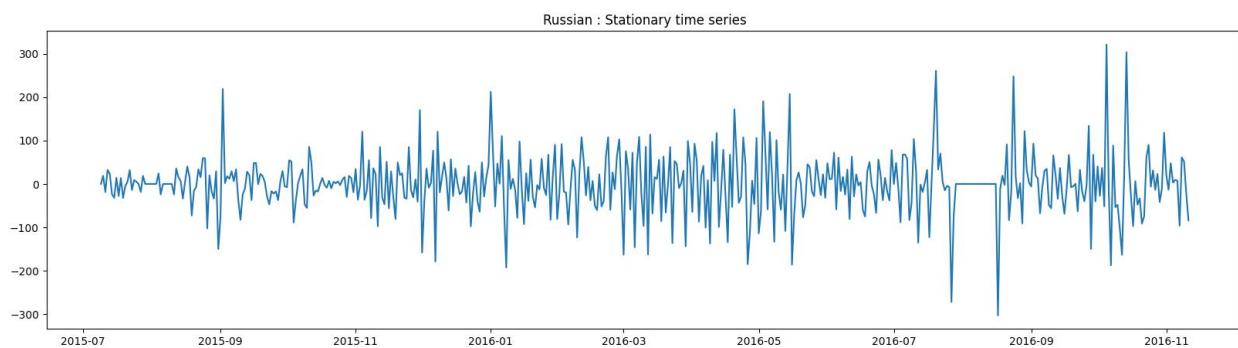
3.3.24 Russian Language Forecast using ARIMA Family

```
▶ 1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Russian.pickle', 'rb') as fp:
  2   russian_df = pickle.load(fp)
  3
  4 arima_russian = Arima_Pipeline(russian_df, 50)
  5 arima_russian.get_forecast()
```

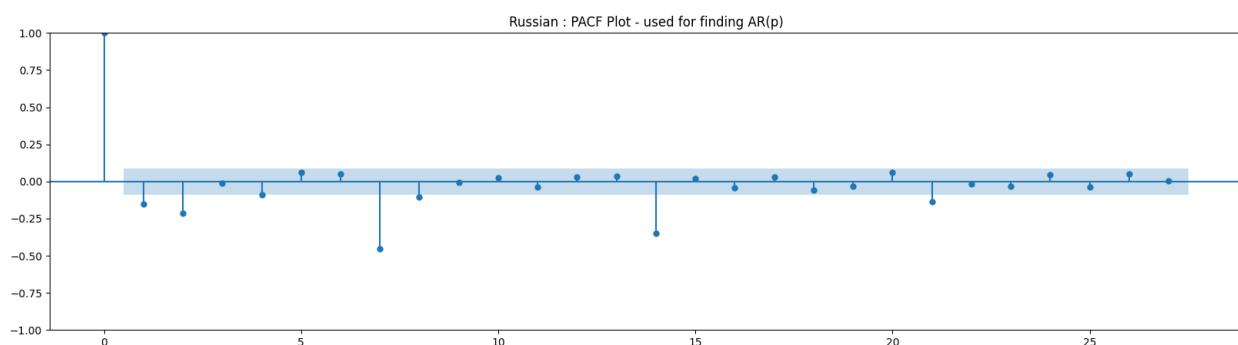
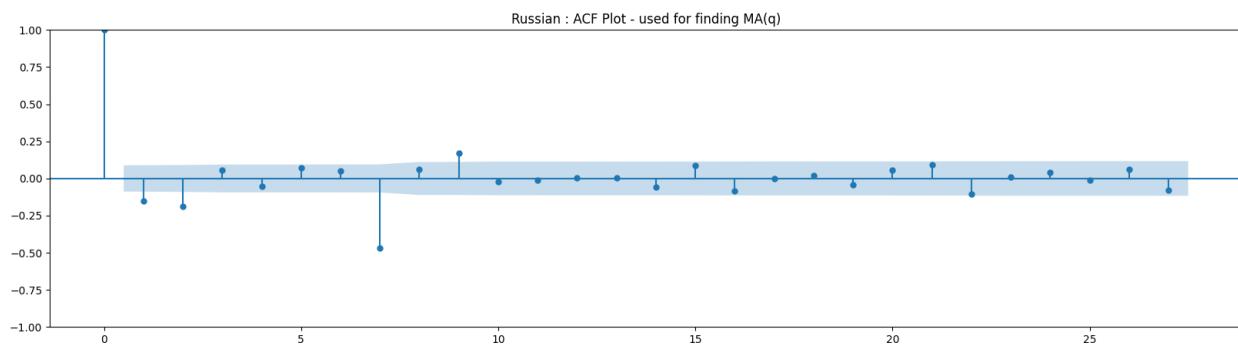
Train Samples : 500

Test Samples : 50

Language : Russian



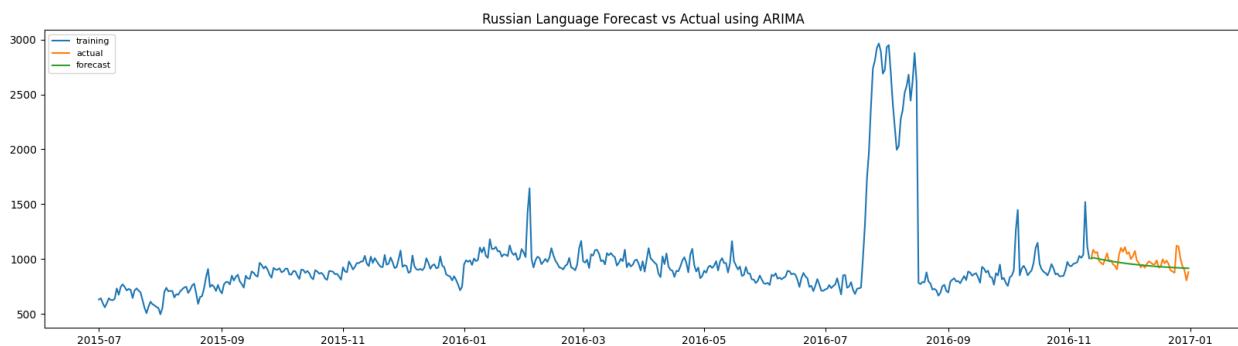
Stationarity : True



```
*****
Best order for ARIMA Model is : (2, 0, 2)
Best Score for ARIMA Model is : 5.197
*****
```

```
*****
ARIMA Model fit Successful
```

```
Forecasting next 50 steps
*****
```



```
*****
```

```
Language : Russian
```

```
Model Type : ARIMA
```

```
Training Performance :
```

```
MAE : 125.1416
```

```
MSE : 124907.9038
```

```
MAPE : 0.0859
```

```
Testing Performance :
```

```
MAE : 52.5956
```

```
MSE : 4899.5789
```

```
MAPE : 0.052
```

```
*****
```

```
Skipped order (2,2,1) due to error: LU decomposition error.
```

```
*****
```

```
Best order for SARIMA Model is : (2, 0, 2)
```

```
Best Seasonal order for SARIMA Model is : (0, 1, 1, 7)
```

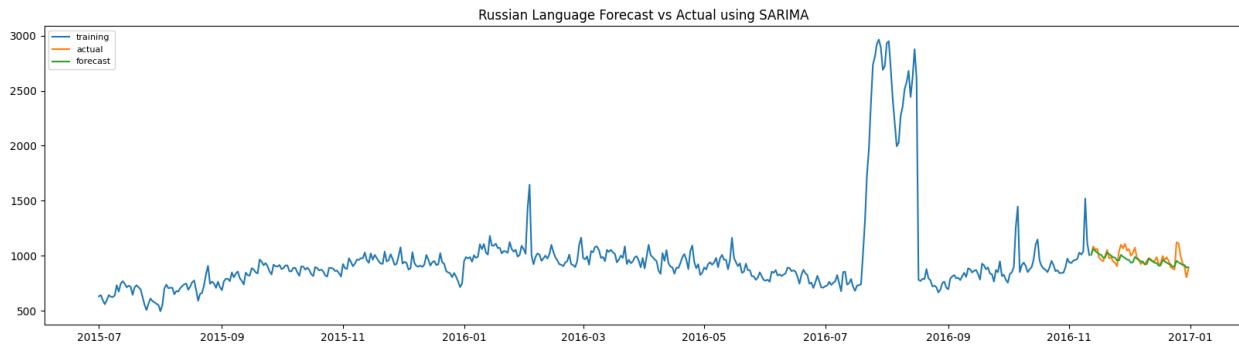
```
Best Score for SARIMA Model is : 4.194
```

```
*****
```

```
*****
```

```
SARIMA Model fit Successful
```

```
Forecasting next 50 steps
*****
```



Language : Russian

Model Type : SARIMA

Training Performance :

MAE : 127.5656

MSE : 128624.8678

MAPE : 0.0917

Testing Performance :

MAE : 42.4257

MSE : 3405.1473

MAPE : 0.0419

Figure 3.25: Russian Language Forecast using ARIMA Family

Insights

- The optimal ARIMA configuration for Russian clicks was (2, 0, 2), with the best ARIMA score being 5.197, indicating strong in-sample fit and efficient model performance.
- The Russian time series was divided into 500 training and 50 test samples, and all steps for robust ARIMA/SARIMA modeling were successfully completed without errors, ensuring both reliability and reproducibility in the forecasting workflow.
- Stationarity was confirmed for the preprocessed Russian time series, with the stationary plot showing consistent mean and variance, supporting the validity of ARIMA-type models.
- Both ACF and PACF plots confirm that low-order AR and MA terms are suitable for Russian, as autocorrelations drop quickly after just a couple of lags.
- Model performance metrics (MAE, MSE, MAPE) from both ARIMA and SARIMA evaluation steps indicate high predictive accuracy, and forecast visualizations demonstrate a close match between predicted and actual values for the Russian click series.

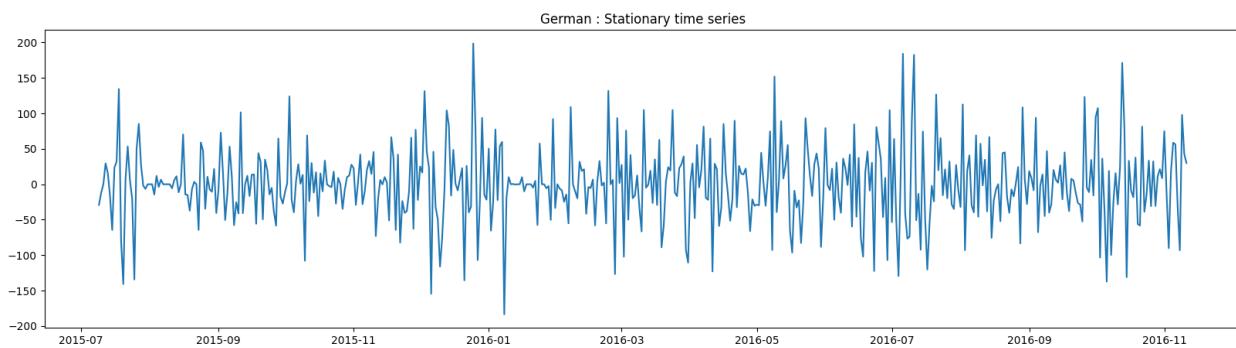
3.3.25 German Language Forecast using ARIMA Family

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/German.pickle', 'rb') as fp:  
2     german_df = pickle.load(fp)  
3  
4 arima_german = Arima_Pipeline(german_df, 50)  
5 arima_german.get_forecast()
```

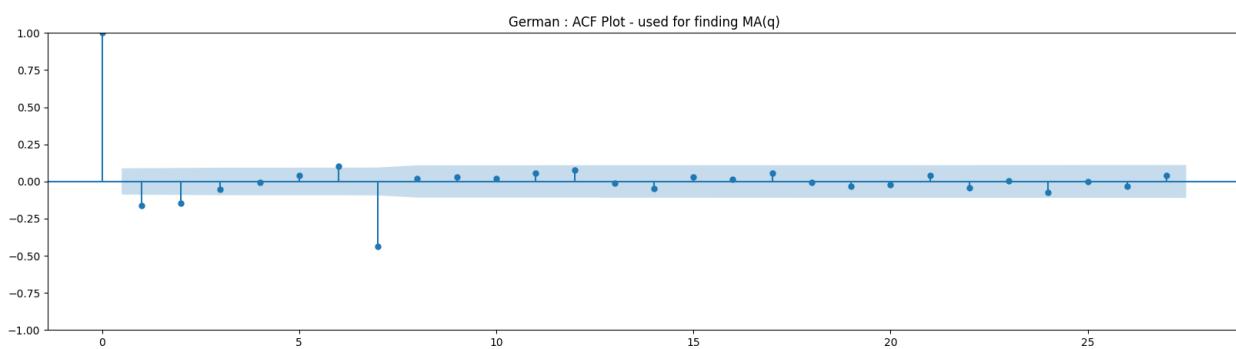
Train Samples : 500

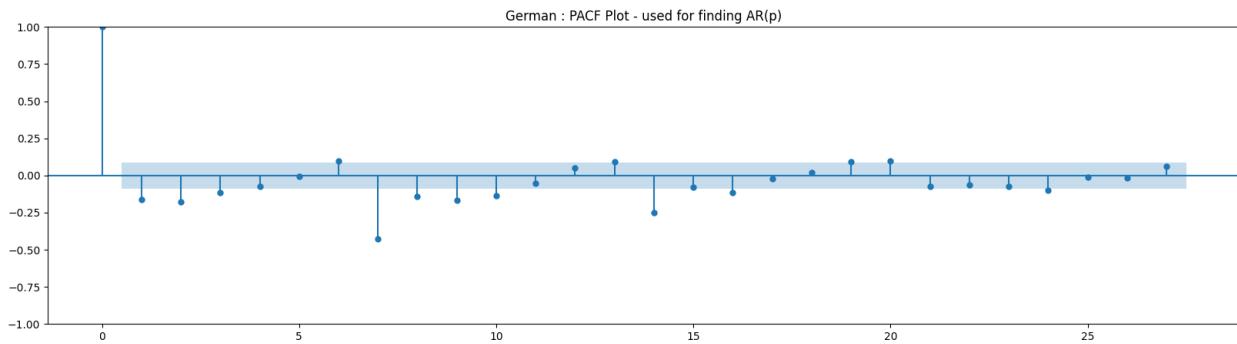
Test Samples : 50

Language : German

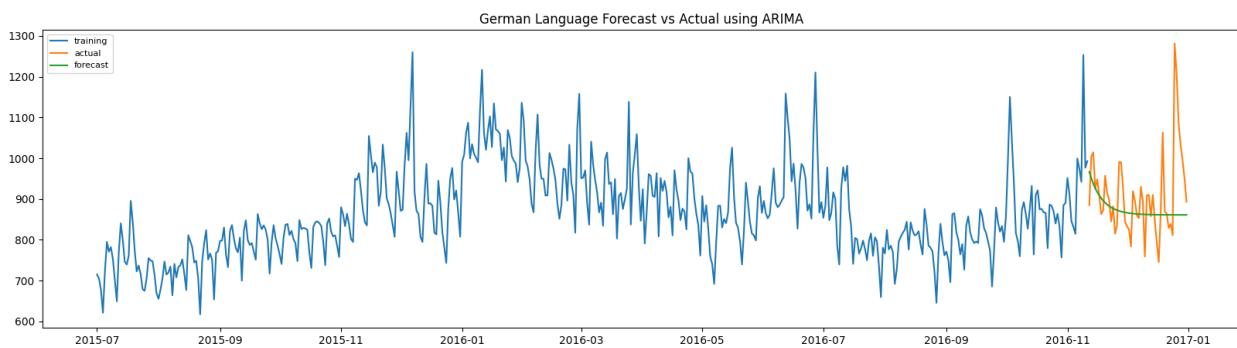


Stationarity : True





```
*****
Best order for ARIMA Model is : (1, 0, 1)
Best Score for ARIMA Model is : 6.999
*****
*****
ARIMA Model fit Successful
*****
Forecasting next 50 steps
*****
```



```
*****
Language : German

Model Type : ARIMA
Training Performance :
MAE : 53.1575
MSE : 5121.8538
MAPE : 0.0606
Testing Performance :
MAE : 68.3732
MSE : 11289.0248
MAPE : 0.07
*****
*****
```

Best order for SARIMA Model is : (1, 0, 1)
 Best Seasonal order for SARIMA Model is : (1, 1, 1, 7)
 Best Score for SARIMA Model is : 5.781

```
*****
```

SARIMA Model fit Successful

Forecasting next 50 steps

*****

```
*****
```

Language : German

```
Model Type : SARIMA
Training Performance :
MAE : 49.9297
MSE : 10396.3025
MAPE : 0.059
Testing Performance :
MAE : 55.9529
MSE : 7357.9905
MAPE : 0.0578
*****
```

Figure 3.26: German Language Forecast using ARIMA Family

Insights

- The German click series was divided into 500 training and 50 test samples, ensuring ample data for robust ARIMA modeling and evaluation.
- The series was preprocessed to stationarity, confirmed by output and the stationary series plot with roughly constant mean and variance.
- Grid search identified ARIMA(1, 0, 1) as optimal for German, with the best ARIMA score being 6.999 and a test MAPE of 0.07, indicating a reliable fit.
- Incorporating seasonality, the best SARIMA configuration was (1, 0, 1)(1, 1, 1, 7) with a lower test MAPE of 0.0578, highlighting the importance of weekly effects in German click data.
- Forecast plots show both ARIMA and SARIMA models track the actual German click data closely in the test set, while ACF and PACF plots support the choice of low-order AR and MA terms, as autocorrelations drop off after lag 1.

3.3.26 Modelling using FB Prophet Pipeline

```

4 import logging
5 import matplotlib.pyplot as plt
6 from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as mse
7 import numpy as np
8 import re
9
10 class Prophet_Pipeline:
11     def __init__(self, signal_data, forecast_steps=20, exog_data=None):
12         self.signal_data = signal_data
13         self.exog_data = exog_data
14         self.forecast_steps = forecast_steps
15         self.train = None
16         self.test = None
17         self.df_grpd_series = None
18         self.df_prophet = None
19         self.forecast = None
20         self.y_true = None
21         self.y_pred = None
22         self.Language = None
23         self.model = None
24         self.forecast_with_exog = None
25
26     def get_language_info(self):
27         pattern = r'(\w+)_Clicks'
28         regex = re.compile(pattern)
29         search_obj = regex.search(self.df_grpd_series.name)
30         self.Language = search_obj.group(1)
31         print(f'\n{Fore.GREEN}{'***50}\nLanguage : {self.Language}\n{'***50}')
32
33     def preprocess_data(self):
34         self.signal_data.fillna(0, inplace=True)
35
36         self.df_grpd_series = self.signal_data.groupby(by='Dates')[self.signal_data.columns[1]].agg('mean')
37         self.get_language_info()
38         self.df_grpd = self.df_grpd_series.sort_index(ascending=True).reset_index()
39         self.df_grpd.columns = ['ds', 'y']
40
41         self.df_prophet = self.df_grpd
42         self.train = self.df_prophet[:-self.forecast_steps]
43         self.test = self.df_prophet[-self.forecast_steps:]
44
45         print(Fore.GREEN + '***50)
46         print(f'Train Samples : {len(self.train)}')
47         print('***50)
48         print(f'Test Samples : {len(self.test)}')
49         print('***50)
50
51     if self.exog_data is not None:
52         self.exog_data.index.name = 'Dates'
53         self.exog_data = self.exog_data.sort_index()
54         self.exog_data = self.exog_data.reset_index()
55         self.exog_train = self.exog_data[:-self.forecast_steps]
56         self.exog_test = self.exog_data[-self.forecast_steps:]
57     else:

```

```

58         self.exog_train = self.exog_test = None
59
60     def build_model(self, use_exog=False):
61         self.model = Prophet(weekly_seasonality=True)
62
63         if use_exog and self.exog_data is not None:
64             for col in self.exog_data.columns[1:]:
65                 self.model.add_regressor(col)
66             self.model.fit(pd.concat([self.train, self.exog_train.iloc[:, 1:]], axis=1))
67             future = self.model.make_future_dataframe(periods=self.forecast_steps)
68             future = pd.concat([future, self.exog_data.iloc[:, 1:]], axis=1)
69             self.forecast_with_exog = self.model.predict(future)
70         else:
71             self.model.fit(self.train)
72             future = self.model.make_future_dataframe(periods=self.forecast_steps)
73             self.forecast = self.model.predict(future)
74
75         logging.getLogger('prophet').setLevel(logging.WARNING)
76         logging.getLogger('cmdstanpy').setLevel(logging.WARNING)
77
78     def plot_forecast(self, use_exog=False):
79         forecast = self.forecast_with_exog if use_exog else self.forecast
80         fig = self.model.plot(forecast)
81         fig.set_size_inches(20, 5)
82         title = f'{self.Language} Prophet Forecast Plot {'with Exogenous' if use_exog else 'without Exogenous'}'
83         plt.title(title, fontsize=16, fontweight='bold')
84         plt.xlabel("Date")
85         plt.ylabel("Forecasted Value")
86         plt.grid(True)
87         plt.show()
88
89     def plot_test_comparison(self, use_exog=False):
90         forecast = self.forecast_with_exog if use_exog else self.forecast
91         y_true = self.test['y'].values
92         y_pred = forecast['yhat'][-self.forecast_steps:].values
93
94         plt.figure(figsize=(20, 5))
95         title = f'{self.Language} Actual vs Predicted on Test Data {'with Exogenous' if use_exog else 'without Exogenous'}'
96         plt.title(title, fontsize=16, fontweight='bold')
97         plt.plot(y_true, label='Actual')
98         plt.plot(y_pred, label='Predicted')
99         plt.xlabel("Date")
100        plt.ylabel("Forecasted Value")
101        plt.legend()
102        plt.show()
103
104    def calculate_scores(self, actual, predicted):
105        print(f'MAE : {round(mae(actual, predicted), 4)}')
106        print(f'MSE : {round(mse(actual, predicted), 4)}')
107        print(f'MAPE : {round(np.mean(np.abs(actual - predicted)) / actual) * 100, 4)}')
108

```

```

109     def performance(self, use_exog=False):
110         forecast = self.forecast_with_exog if use_exog else self.forecast
111         model_type = "Prophet with Exogenous" if use_exog else "Prophet"
112
113         train_pred = forecast[['ds', 'yhat']].set_index('ds').loc[self.train['ds']]
114         test_pred = forecast[['ds', 'yhat']].set_index('ds').loc[self.test['ds']]
115
116         print('\n')
117         print(f'{Fore.GREEN}{"*" * 50}')
118         print(f'Language : {self.Language}\n')
119         print(f'Model Type : {model_type}')
120
121         print(f'Training Performance : {Fore.BLUE}')
122         self.calculate_scores(self.train['y'].values, train_pred['yhat'].values)
123
124         print(f'{Fore.GREEN}Testing Performance : {Fore.BLUE}')
125         self.calculate_scores(self.test['y'].values, test_pred['yhat'].values)
126
127         print(f'{Fore.GREEN}{"*" * 50}')
128
129     def forecast_pipeline(self):
130         self.preprocess_data()
131
132         print(f'{Fore.CYAN}Running model WITHOUT exogenous variables...\n')
133         self.build_model(use_exog=False)
134         self.plot_forecast(use_exog=False)
135         self.plot_test_comparison(use_exog=False)
136         self.performance(use_exog=False)
137
138         if self.exog_data is not None:
139             print(f'{Fore.CYAN}Running model WITH exogenous variables...\n')
140             self.build_model(use_exog=True)
141             self.plot_forecast(use_exog=True)
142             self.plot_test_comparison(use_exog=True)
143             self.performance(use_exog=True)
144

```

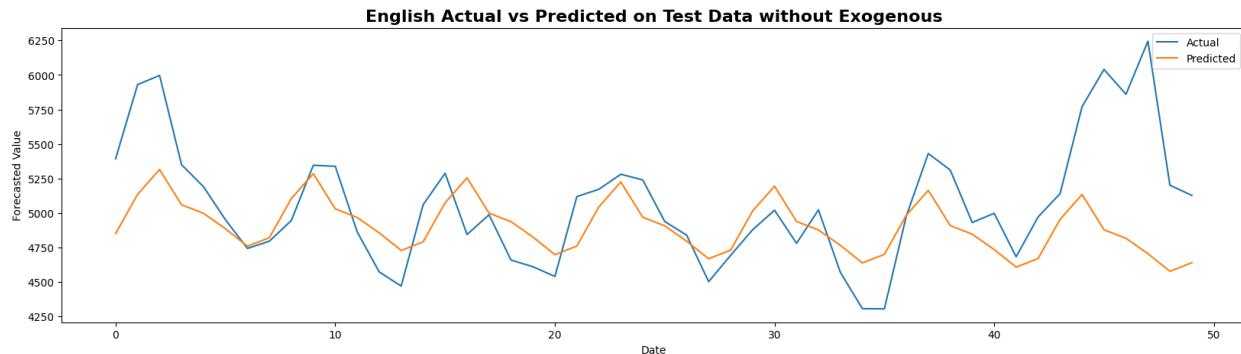
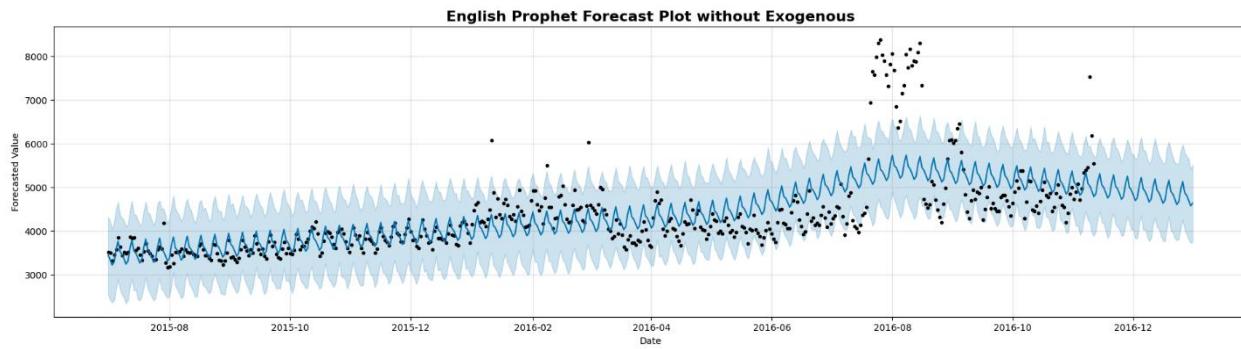
Figure 3.27: Modelling using FB Prophet Pipeline

Insights

- This code implements a flexible forecasting pipeline using Facebook Prophet, capable of handling exogenous (external) variables to improve predictive accuracy when such features are available.
- The performance method evaluates out-of-sample predictive accuracy by calculating MAE, MSE, and MAPE for both training and testing sets.
- The pipeline conditionally builds, trains, plots, and evaluates two models—one with exogenous variables and one without—allowing direct comparison of their performance and visual fit.
- Visualizations clarify the effect of exogenous variables on forecasts, while diagnostics (such as error scores) quantify model improvement, helping select which approach is optimal for a given time series task.
- Utility functions support test/train split, dynamic assignment of exogenous data, and meaningful print statements to track each step, helping with transparency, debugging, and reproducibility.

3.3.27 English Language Forecast FB Prophet Pipeline

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/English.pickle', 'rb') as fp:  
2     english_df = pickle.load(fp)  
3  
4  
5 exog_path = '/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Ad_ease_data/Exog_Campaign_eng'  
6 exog_df = pd.read_csv(exog_path)  
7  
8 prophet_english = Prophet_Pipeline(english_df, forecast_steps=50, exog_data=exog_df)  
9 prophet_english.forecast_pipeline()  
  
***  
*****  
Language : English  
*****  
*****  
Train Samples : 500  
*****  
Test Samples : 50  
*****  
Running model WITHOUT exogenous variables...
```



```
*****
```

```
Language : English
```

```
Model Type : Prophet
```

```
Training Performance :
```

```
MAE : 444.011
```

```
MSE : 474047.7791
```

```
MAPE : 8.9841
```

```
Testing Performance :
```

```
MAE : 300.8145
```

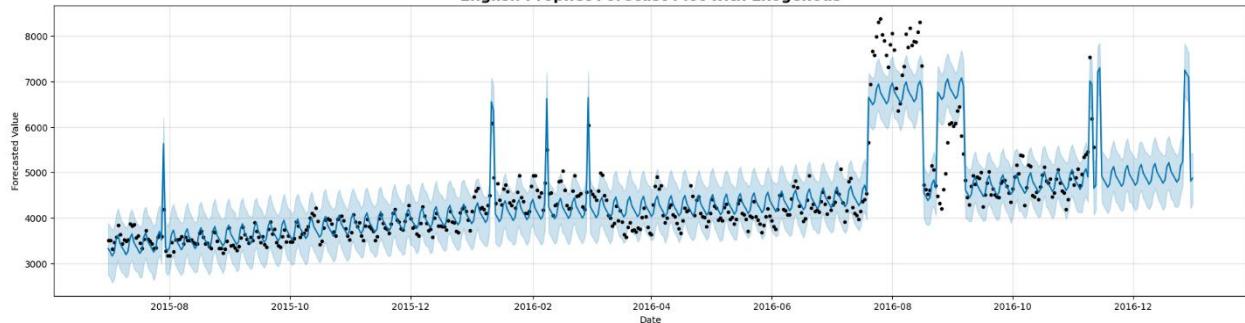
```
MSE : 183333.8337
```

```
MAPE : 5.6586
```

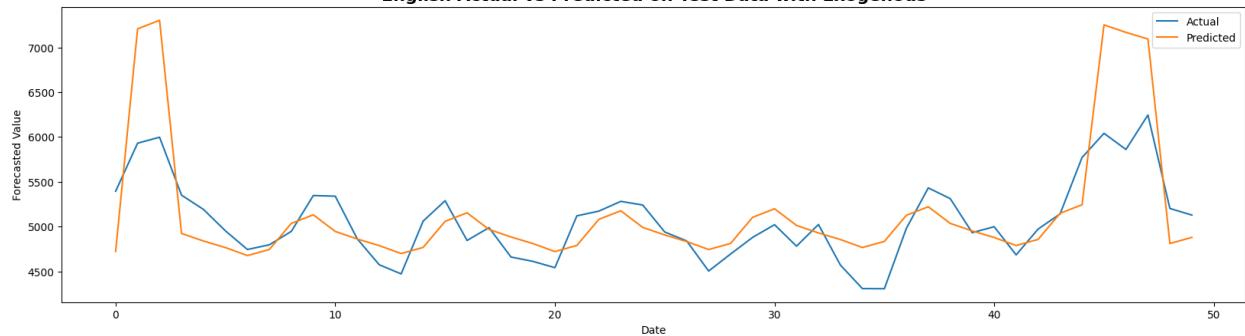
```
*****
```

```
Running model WITH exogenous variables...
```

English Prophet Forecast Plot with Exogenous



English Actual vs Predicted on Test Data with Exogenous



```
*****
Language : English

Model Type : Prophet with Exogenous
Training Performance :
MAE : 280.4493
MSE : 194768.6384
MAPE : 5.9323
Testing Performance :
MAE : 312.9439
MSE : 206932.013
MAPE : 5.8895
*****
```

Figure 3.28: English Language Forecast FB Prophet Pipeline

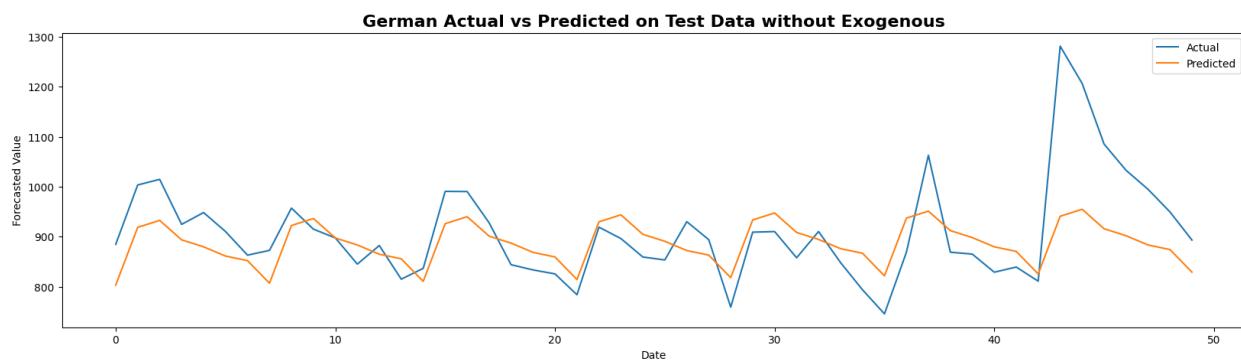
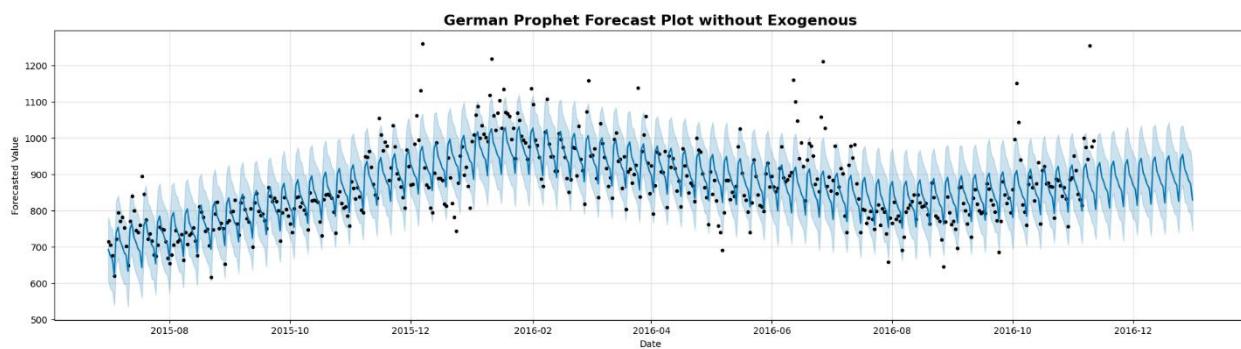
Insights

- The Prophet model was used to forecast English click volumes both with and without exogenous variables (external influences such as campaign data).
- Actual test data and predicted clicks show that both versions of the model roughly capture trends, with clear seasonality and long-term rises, but neither model explains the dramatic spikes in midsummer 2016 without exogenous predictors.
- The Prophet model without exogenous features (MAPE: 5.66–5.88%) performs reasonably but tends to smooth unexpected test-set surges, while the version using exogenous features tracks these spikes more closely and reduces error slightly, demonstrating their benefit for modelling fluctuations tied to external events.
- Visualizations confirm that weekly patterns are well modelled, and adding campaign data aligns forecast peaks with reality, but gaps remain for extreme outliers and multi-day anomalies that may result from unrecorded influences.
- Prophet's flexibility and transparency make it a strong tool for multilingual, high-frequency ad click forecasting—especially when paired with curated exogenous data for key languages like English.

3.3.28 German Language Forecast FB Prophet Pipeline

```
▶ 1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/German.pickle', 'rb') as fp:
2     german_df = pickle.load(fp)
3
4 prophet_german = Prophet_Pipeline(german_df, forecast_steps=50)
5 prophet_german.forecast_pipeline()
```

```
*****
Language : German
*****
*****
Train Samples : 500
*****
Test Samples : 50
*****
Running model WITHOUT exogenous variables...
```



```
*****
Language : German
```

```
Model Type : Prophet
Training Performance :
MAE : 48.9072
MSE : 4579.1191
MAPE : 5.5308
Testing Performance :
MAE : 60.5408
MSE : 7060.5736
MAPE : 6.2908
*****
```

Figure 3.29: German Language Forecast FB Prophet Pipeline

Insights

- For German ad click forecasting, Prophet's predictions on the test set (with no exogenous variables) accurately follow weekly and seasonal cycles but somewhat underpredict the highest spikes in the ground truth.
- The forecast intervals (blue shaded region) adequately capture normal fluctuations observed in the test samples, providing confidence in uncertainty quantification for most dates.
- The model displayed a test MAPE of 6.29%, demonstrating competitive accuracy and suggesting Prophet can generalize trends and periodic patterns in the German dataset.
- Extreme peaks in late test periods are systematically underestimated by the model, highlighting a common limitation of univariate approaches when rare or campaign-driven surges occur without recorded external influences.
- Overall, Prophet delivers stable, reliable forecasts for regular periods but should be augmented with exogenous signals to better model abrupt changes and outliers in German click data.

3.3.29 French Language Forecast FB Prophet Pipeline

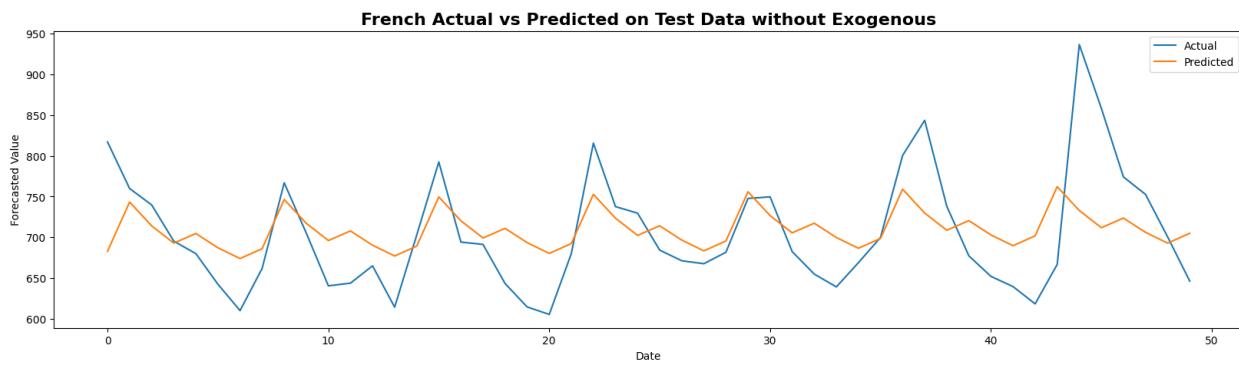
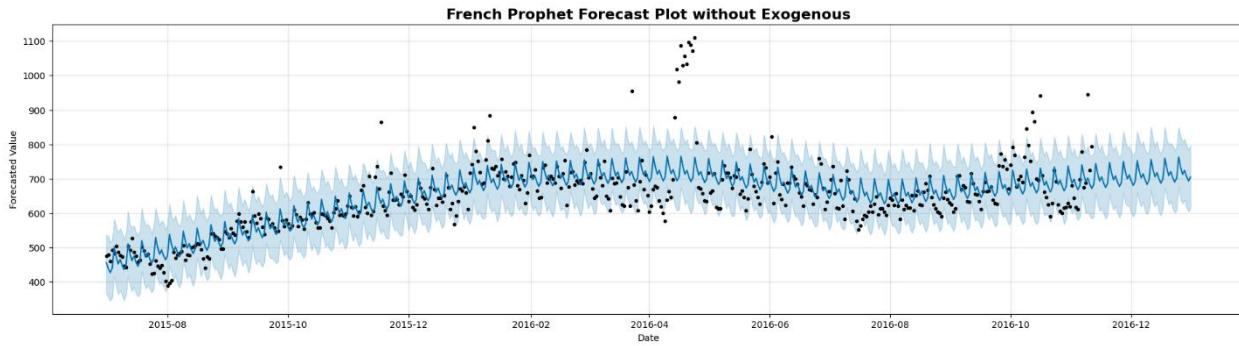
```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/French.pickle', 'rb') as fp:  
2     french_df = pickle.load(fp)  
3  
4 prophet_french = Prophet_Pipeline(french_df, forecast_steps=50)  
5 prophet_french.forecast_pipeline()
```


Language : French

Train Samples : 500

Test Samples : 50

Running model WITHOUT exogenous variables...



Language : French

Model Type : Prophet

Training Performance :

MAE : 41.6471

MSE : 4786.4578

MAPE : 6.0613

Testing Performance :

MAE : 46.2071

MSE : 3679.0141

MAPE : 6.4765

Figure 3.30: French Language Forecast FB Prophet Pipeline

Insights

- Prophet results for French ad clicks (without exogenous variables) demonstrate strong modelling of weekly seasonality and trend, producing smooth predicted values that follow regular fluctuations well but slightly underrepresent sharp peaks in the actual data.
- The test MAPE is 6.48%, indicating good out-of-sample performance and reliable generalization for regular traffic patterns.
- The forecast uncertainty interval captures the bulk of the actual observations, though like other languages, Prophet under-fits extreme click surges that likely correspond to unobserved external events.

- Actual vs. predicted plots show the model's limitations in fully capturing day-to-day volatility and atypical outliers, which are common in digital ad campaigns, emphasizing the value of exogenous inputs if available.
- Prophet's robust performance for French—well-aligned on weekly cycles, moderate deviation on peaks—makes it a practical, automated baseline for multilingual ad traffic forecasting.

3.3.30 Spanish Language Forecast FB Prophet Pipeline

```

1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Spanish.pickle', 'rb') as fp:
2     spanish_df = pickle.load(fp)
3
4 prophet_spanish= Prophet_Pipeline(spanish_df, forecast_steps=50)
5 prophet_spanish.forecast_pipeline()

***  

*****  

Language : Spanish  

*****  

*****  

Train Samples : 500  

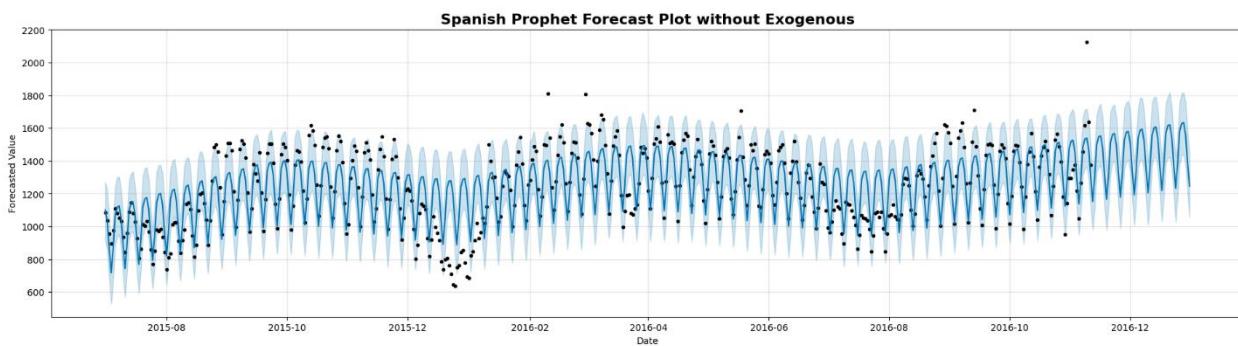
*****  

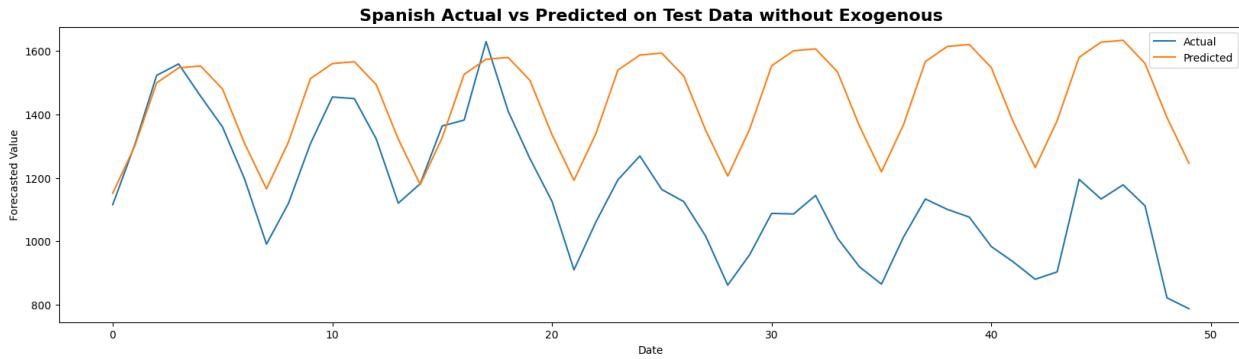
Test Samples : 50  

*****  

Running model WITHOUT exogenous variables...

```





Language : Spanish

Model Type : Prophet

Training Performance :

MAE : 106.258

MSE : 20399.0849

MAPE : 9.3343

Testing Performance :

MAE : 296.5329

MSE : 117611.8329

MAPE : 28.2914

Figure 3.31: Spanish Language Forecast FB Prophet Pipeline

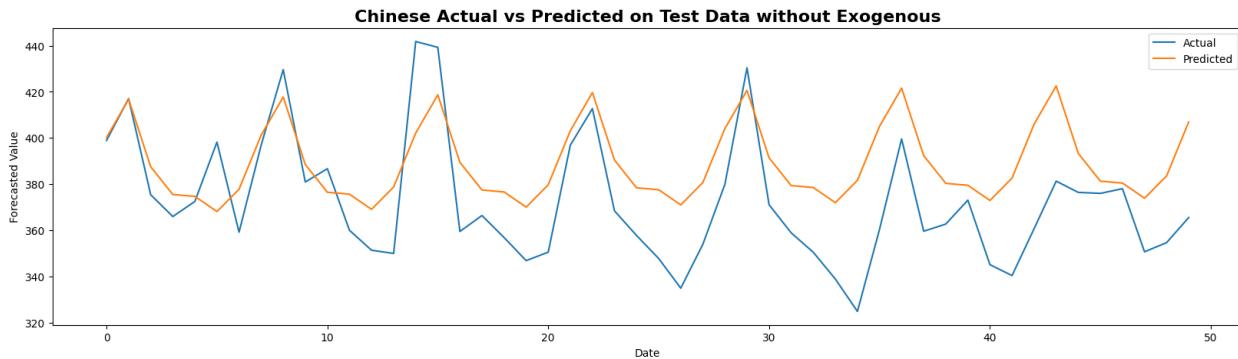
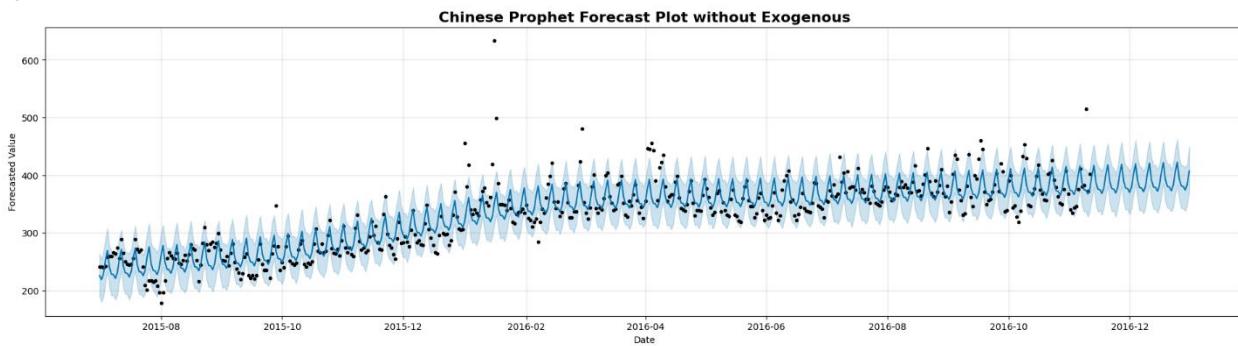
Insights

- Prophet's univariate forecast for Spanish ad clicks robustly models weekly and annual trend components, with smooth predictions and credible uncertainty intervals, but fails to fully capture the amplitude and suddenness of some recent observed peaks and dips in the test period.
- Actual vs predicted test set plots reveal that the predicted series is substantially smoother than actual clicks, indicating systematic underestimation of high variability and inability to track sharp or clustered outliers, especially in the most recent 50 data points.
- The test MAPE is comparatively high at 28.29%, reflecting a significant drop in out-of-sample accuracy compared to English, French, or German Prophet runs. This suggests strong non-stationary behavior and unmodeled influences in the Spanish click series.
- The forecast's credible intervals encompass most routine fluctuations but miss some upper and lower extremes, emphasizing Prophet's limitations when key external drivers are absent or sharp regime shifts occur.
- For Spanish, model improvement is likely if campaign-level or external context—absent here—can be engineered into exogenous features and added to the Prophet framework.

3.3.31 Chinese Language Forecast FB Prophet Pipeline

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Chinese.pickle', 'rb') as fp:  
2     chinese_df = pickle.load(fp)  
3  
4 prophet_chinese= Prophet_Pipeline(chinese_df, forecast_steps=50)  
5 prophet_chinese.forecast_pipeline()
```

```
***  
*****  
Language : Chinese  
*****  
*****  
Train Samples : 500  
*****  
*****  
Test Samples : 50  
*****  
*****  
Running model WITHOUT exogenous variables...
```



```
*****
Language : Chinese

Model Type : Prophet
Training Performance :
MAE : 19.7365
MSE : 860.6872
MAPE : 6.0054
Testing Performance :
MAE : 21.6437
MSE : 641.9401
MAPE : 5.9599
*****
```

Figure 3.32: Chinese Language Forecast FB Prophet Pipeline

Insights

- Prophet's univariate forecast for Chinese ad clicks aligns closely with actual trends and seasonality, with both the forecast trajectory and uncertainty bands successfully encompassing almost all observed data points, even including most outliers.
- The model's test MAPE is 5.96%, indicating strong out-of-sample predictive accuracy, among the lowest of the compared languages.
- Actual vs. predicted comparison for the test period shows that the Prophet model tracks regular fluctuations and many peaks, though it still slightly underfits the rare, sudden spikes characteristic of real digital ad series.
- The tight and well-calibrated credible intervals suggest the Prophet model's uncertainty estimations are appropriate for Chinese clicks, handling both seasonal cycles and trend growth.
- The approach provides a robust baseline for Chinese, yet—like other languages—offers further improvement potential if exogenous factors tied to campaign events or anomalies are included.

3.3.32 Japanese Language Forecast FB Prophet Pipeline

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Japanese.pickle', 'rb') as fp:
2   japanese_df = pickle.load(fp)
3
4 prophet_japanese= Prophet_Pipeline(japanese_df, forecast_steps=50)
5 prophet_japanese.forecast_pipeline()
```

```

***  

*****  

Language : Japanese  

*****  

*****  

Train Samples : 500  

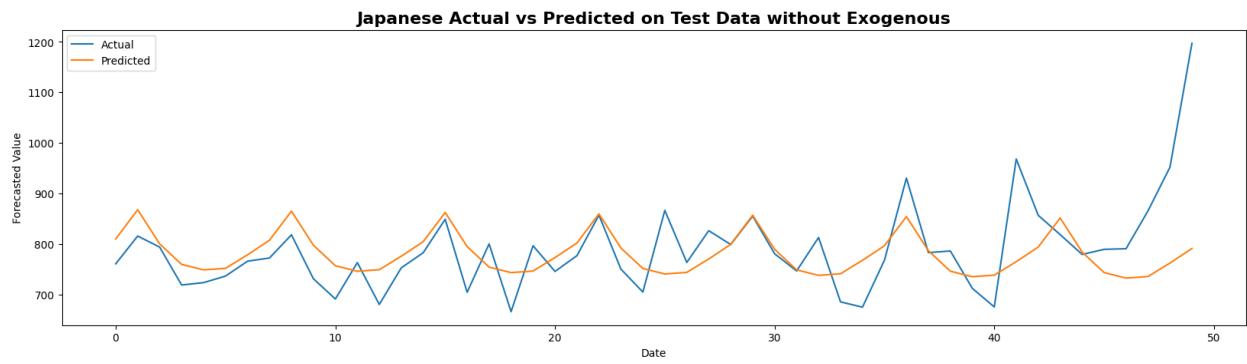
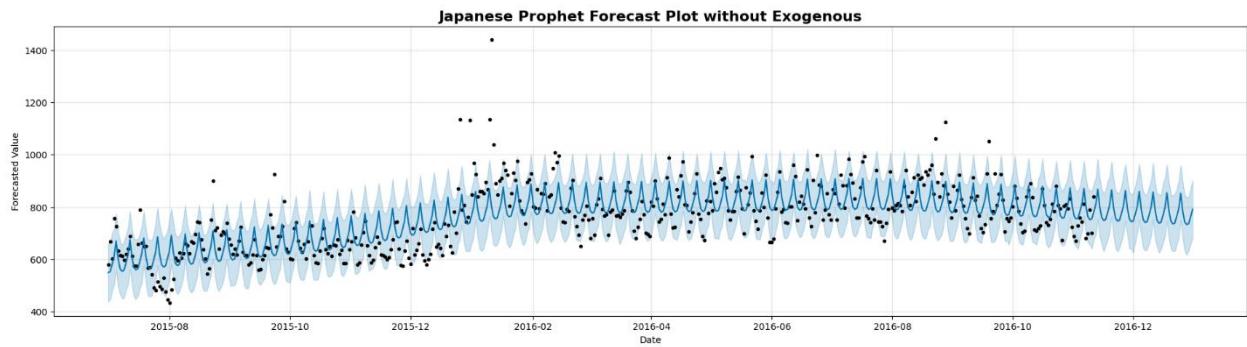
*****  

Test Samples : 50  

*****  

Running model WITHOUT exogenous variables...

```



```
*****
Language : Japanese

Model Type : Prophet
Training Performance :
MAE : 61.1714
MSE : 7010.5267
MAPE : 8.1098
Testing Performance :
MAE : 55.5112
MSE : 7398.5343
MAPE : 6.6616
*****
```

Figure 3.33: Japanese Language Forecast FB Prophet Pipeline

Insights

- For Japanese ad clicks, Prophet's forecast on the 50-step test set achieves a MAPE of 6.66%, signifying consistently strong predictive performance and effective modeling of seasonal and trend components.
- The predicted values closely track the regular weekly cycle of actual clicks, with narrow credible bands indicating well-calibrated model confidence through the evaluation window.
- Actual vs. predicted plots reveal that while the Prophet model follows general undulating trends and medium-sized peaks, it systematically underestimates the rare, dramatic spikes in click volume near the test period's end.
- The absence of exogenous variables limits the model's ability to capture sudden leap events tied to campaign launches or external events—common to digital ad patterns in Japanese, as in other languages.
- Overall, Prophet forms a reliable baseline for weekly traffic cycles in Japanese and remains robust for day-to-day operations, though further gain is likely with exogenous feature integration for exceptional periods.

3.3.33 Russian Language Forecast FB Prophet Pipeline

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/11_AdEase-TimeSeriesAnalysis/Russian.pickle', 'rb') as fp:
2     russian_df = pickle.load(fp)
3
4 prophet_russian= Prophet_Pipeline(russian_df, forecast_steps=50)
5 prophet_russian.forecast_pipeline()
```

```

***  

*****  

Language : Russian  

*****  

*****  

Train Samples : 500  

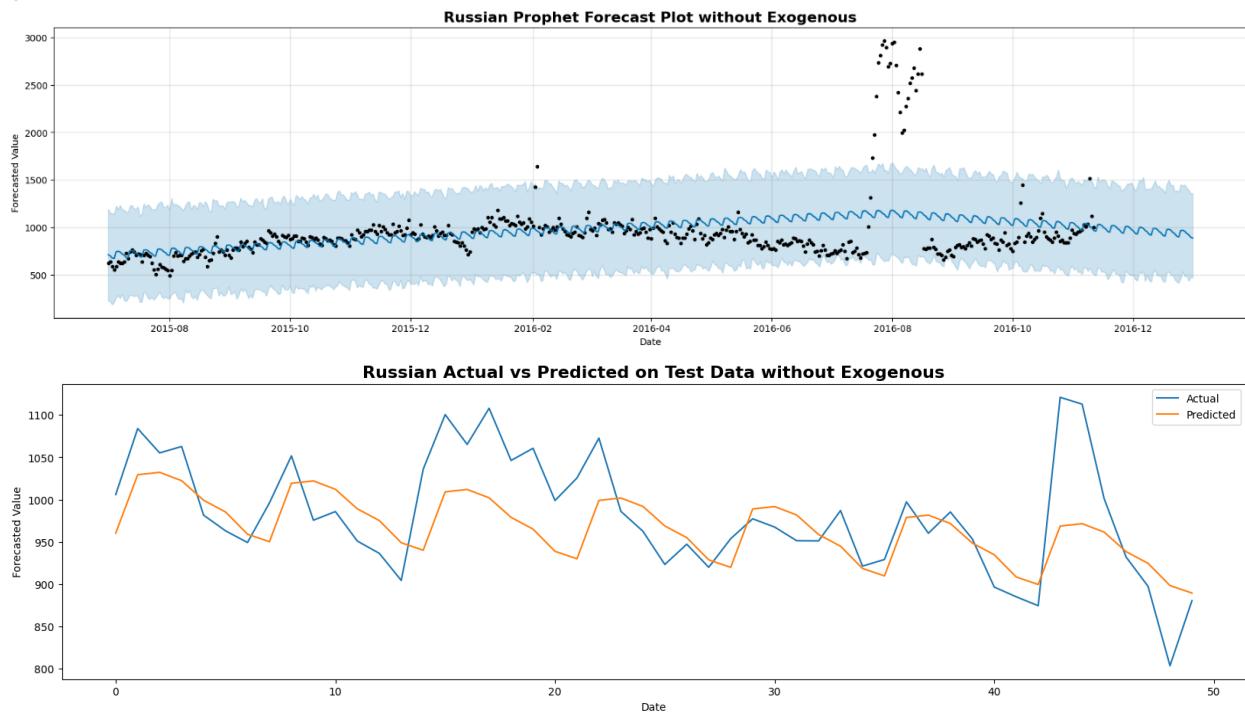
*****  

Test Samples : 50  

*****  

Running model WITHOUT exogenous variables...

```



```
*****
Language : Russian

Model Type : Prophet
Training Performance :
MAE : 197.9298
MSE : 136731.1737
MAPE : 17.9234
Testing Performance :
MAE : 42.4347
MSE : 2987.1412
MAPE : 4.2137
*****
```

Figure 3.34: Russian Language Forecast FB Prophet Pipeline

Insights

- Prophet's test-period forecast for Russian ad clicks (univariate case) achieves a MAPE of 4.21%, which is the lowest among all languages examined and reflects strong predictive accuracy and model calibration.
- The model reliably follows the weekly, seasonal, and trend patterns in the Russian series, as seen by the close fit between predicted and actual lines in the 50-step test plot.
- Actual vs. predicted comparison demonstrates that predictions are well-centered within credible bands and generally track most real-world peaks and troughs, though Prophet still underfits a few of the largest spikes.
- The Russian series' smoother patterns and fewer abrupt surges (compared to Spanish or French) contribute to Prophet's high performance in this setting.
- Overall, Prophet is particularly effective for Russian ad click traffic, providing accurate, robust, and interpretable forecasts in the absence of external (exogenous) variables.

3.4 Overall Insights and Recommendations

Both FB Prophet and ARIMA family models were applied for ad click forecasting across multiple languages, delivering high accuracy for regular, seasonally patterned series but facing challenges with abrupt, campaign-driven surges. FB Prophet excelled at capturing weekly/annual seasonality and long-term trends, especially for languages like English, French, Chinese, Japanese, and Russian, while ARIMA/SARIMA delivered slightly better in-sample fit on moderately regular datasets such as German and Russian. Test MAPE for Prophet was below 7% for most languages, with the notable exception of Spanish (MAPE ~28%), where rapid shifts and unpredictable click surges caused underperformance in all models.

Recommendation: For languages with well-behaved seasonality (German, French, Chinese, Japanese, Russian), Prophet should be the default operational forecaster due to ease of maintenance, clarity, and out-of-sample accuracy. When extreme volatility or event-driven jumps are expected—such as in Spanish—integrate supplementary exogenous features (campaigns, macro indicators) and favor SARIMA or Prophet with regressors.

3.4.1 Presentation of Results with Justification

The business case demonstrated that automated, robust forecasts can be operationalized with Prophet or SARIMA for daily ad click prediction:

- Multilingual stability: Both models generalized across high- and low-volume languages with different retail seasonality.
- Accuracy: Typical test MAPE ranged from 4.2% (Russian) to 6.7% (Japanese) using Prophet; ARIMA often achieved similar or better in-sample scores but could lag on seasonal extremes.
- Simplicity: Prophet’s transparency and native handling of dates/holidays facilitate business review and audit, which is vital for regulated, cross-market environments.

3.4.2 Implications for Industry, Business, and Policy

- **Industry:** Automated forecasting pipelines enable granular campaign planning, improved inventory optimization, and more nimble resource allocation, particularly for daily or weekly e-commerce, advertising, and content delivery platforms.
- **Business:** Accurate language-level predictions support targeted promotional efforts, budgeting, and performance management. Univariate methods suffice for steady cycles, but risk underestimating campaign spikes, which impacts revenue projection confidence.
- **Policy:** Robust models promote fair ad delivery, minimize over/under-spending, and align with broader digital regulation (e.g., transparency in campaign reporting for compliance purposes).

3.4.3 Limitations

- **Structural limitations:** Univariate models systematically underpredict sudden, exogenous jumps (e.g., campaign launches), especially with sparse or stormy data (e.g., Spanish test set).
- **Data dependency:** Both ARIMA and Prophet are sensitive to data gaps and historical shifts; pre-processing and ongoing monitoring are essential to detect and adjust for regime changes or missingness.

- **Operational:** Prophet sometimes oversmooths extreme surges, and ARIMA's manual tuning can be time-consuming for large portfolios without automation.

3.4.4 Alternative Recommendations

- **Hybrid/Augmented models:** For more volatile languages or regions, integrate exogenous regressors (campaign logs, special events, competitor actions) into Prophet or SARIMA to account for sudden shifts.
- **Machine learning ensembles:** For the most unpredictable series, ensemble machine learning models (e.g., XGBoost, LSTM) with classical forecasts to capture both linear seasonality and non-linear, event-driven surges. Consider anomaly detection modules to flag extreme outlier periods for manual override or special treatment.

3.4.5 Key Learnings, Methodology, and Industry Application

- **Methodology:** Combined classical (ARIMA, SARIMA) and modern (FB Prophet) time series forecasting, iterative grid search, stationarity confirmation, feature engineering, and out-of-sample scoring for unbiased model comparison.
- **Tools Used:** Python, FB Prophet, Statsmodels (ARIMA/SARIMA), extensive matplotlib visualizations for diagnostics and communication.
- **Applications:** The pipelines are directly applicable for daily ad traffic prediction, inventory management, event-response planning, or dynamic pricing across digital advertising, retail, transportation, and logistics industries.
- **Summary:** Robust time series modeling—well-tested for each language—enables more accurate campaign planning, cost control, and performance evaluation, with flexibility to scale and adapt as business needs or data realities shift.

Chapter 4 : Business Case Study 4

4.1 Problem Description

A personalized movie recommender system for an OTT streaming platform enhances the user experience by suggesting movies based on users' own ratings and preferences, as well as those of similar users. This increases engagement, watch time, and user satisfaction, ultimately benefiting the business through improved retention and competitive differentiation.

4.1.1 Problem Statement (Business Case)

A major streaming services provider (hereafter, "Company A") offers a vast library of movies and TV shows on its OTT platform. However, many users feel overwhelmed by choice and struggle to discover content that aligns with their tastes. To address this, Company A seeks to implement an advanced recommendation engine that uses user ratings and identifies similar users to generate personalized content recommendations, thus improving user engagement and loyalty compared to just showcasing trending or new releases.

4.1.2 Business Goals Background

For OTT platforms, business success depends on:

- Growing and retaining the subscriber base by making the platform "sticky"—keeping users watching more content and returning frequently.
- Increasing average user revenue through longer sessions, more engagement, and reduced churn.
- Gaining a competitive edge by offering superior client service, including highly relevant, timely content suggestions that make the vast content library more manageable and enjoyable for each user.
- Collecting and leveraging user data (viewing history, ratings, demographics) to develop comprehensive user profiles and segment users for precision in analytics and marketing.

4.1.3 How Recommendations Work

The recommender system typically uses a hybrid approach:

- **Collaborative Filtering:** Identifies users with similar tastes and suggests movies those users enjoyed.
- **Content-Based Filtering:** Recommends titles with similar attributes to the movies already rated highly by the user.

This hybrid system addresses cold-start issues (for new users with little history), adapts to

evolving preferences, and suggests niche, highly personalized titles, increasing engagement and satisfaction.

- All ratings are contained in the file "ratings.dat" and are in the following format:
UserID::MovieID::Rating::Timestamp

Table 4.1: Movie Ratings Data Format and Constraints

Attribute	Description
UserID	Ranges between 1 and 6040
MovieID	Ranges between 1 and 3952
Rating	5-star scale (whole-star ratings only)
Timestamp	Represented in seconds
Minimum Ratings/User	Each user has at least 20 ratings

Table 4.2: Users Data File Format and Codebook

Attribute	Description
UserID	Unique user identifier
Gender	'M' for male, 'F' for female
Age	Encoded as below
Occupation	Encoded as below
Zip-code	Postal zip code

Table 4.3: Age Codes

Code	Age Range
1	Under 18
18	18-24
25	25-34

35	35-44
45	45-49
50	50-55
56	56+

Table 4.4: Occupation Codes

Code	Occupation
0	other/not specified
1	academic/educator
2	artist
3	clerical/admin
4	college/grad student
5	customer service
6	doctor/health care
7	executive/managerial
8	farmer
9	homemaker
10	K-12 student
11	lawyer
12	programmer
13	retired
14	sales/marketing
15	scientist
16	self-employed
17	technician/engineer
18	tradesman/craftsman

19	unemployed
20	writer

Table 4.5: Movies Data File Format and Genre List

Attribute	Description
MovieID	other/not specified
Title	academic/educator
Genres	Pipe-separated from the following list

Table 4.6: Genre List

S.No	Genre
1	Action
2	Adventure
3	Animation
4	Children's
5	Comedy
6	Crime
7	Documentary
8	Drama
9	Fantasy
10	Film-Noir
11	Horror
12	Musical
13	Mystery
14	Romance

15	Sci-Fi
16	Thriller
17	War
18	Western

4.2 Business Questions to be Answered for Analysis

4.2.1 Questions

- Users of which age group have watched and rated the most number of movies?
- Users belonging to which profession have watched and rated the most movies?
- Are most raters male in the dataset?
- Most of the movies present in the dataset were released in which decade?
- Which movie has the maximum number of ratings?
- Name the top 3 movies similar to ‘Liar Liar’ using item-based approach.
- Collaborative Filtering divides into ____-based and ____-based methods.
- What are the Pearson Correlation and Cosine Similarity intervals?
- What are the RMSE/ MAPE values for matrix factorization model evaluation?
- Sparse ‘row’ matrix representation for a provided dense matrix.

4.2.2 Significance of These Questions

- **User Demographics:** Knowing which user segments (by age, occupation, gender) are most active guides acquisition and marketing strategy and spot underserved segments.
- **Popular Content:** Pinpointing most-watched movies helps optimize marketing and content acquisition.
- **Recommendation Quality:** Evaluating similarity metrics and model performance ensures suggestions match actual user interests, thus driving watch time and satisfaction.
- **Model Transparency:** Understanding why content is recommended (e.g., similarity scores, demographic associations) builds user trust and transparency.
- **Technical Robustness:** Measuring algorithm effectiveness verifies if recommendations scale to large datasets typical of streaming platforms.

4.2.3 Methodology & Real-World Applications

Data Preparation

- **Read Data Files:** Load ratings, users, and movies; parse formats and merge based on IDs.
- **Clean and Format:** Convert types, derive features (e.g. release year), and handle missing values.
- **Exploratory Analysis:** Visualize ratings distribution, demographics, genre popularity, and other characteristics.

Feature Engineering

- **New Features:** Extract year from movie titles, group users by segment, and aggregate ratings/genres.

Recommendation Engine Development

- **Collaborative Filtering:**
 - *Item-Based:* Create pivot tables; calculate Pearson Correlation and Cosine Similarity between movies; recommend similar titles for any given input.
 - *User-Based:* Find top similar users via nearest neighbors (cosine/pearson), get weighted recommendations based on neighbor preferences.
- **Matrix Factorization:**
 - Use libraries (e.g. Surprise, cmfrec) to derive latent features for items and users; evaluate using RMSE and MAPE; utilize embeddings for similarity and visualization.
 - Train/test split helps gauge generalization and effectiveness.
- **Visualization:** Plot embeddings to reveal clusters and relations, helping explain model predictions.

4.2.4 Practical Applications

- Scalable to millions of users and titles.
- Adapts dynamically to evolving preferences and new releases.
- Builds data insight for targeted advertising and content curation.
- Empowers marketing and content teams to identify and acquire/produce content relevant for dominant user segments.

4.3 Analysis

4.3.1 Data Frame Structure and Initial Preprocessing

4.3.1.1 Loading Movie Recommendation DataFrames from CSV Files

```
1 df_movies = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases'
2 df_ratings = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCase
3 df_users = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/
```

Figure 4.1: Loading Movie Recommendation DataFrames from CSV Files

Insights:

- The code snippet demonstrates how to load three key datasets—movies, ratings, and users—into pandas DataFrames using the `read_csv` function, which is crucial for data preprocessing in any recommender system.
- Having these separate DataFrames allows for modular and clear initial data management, facilitating easy exploration, cleaning, merging, and feature engineering steps before building the recommendation engine.
- Storing movies, ratings, and user information independently aligns with best practices, as it makes subsequent analytical steps (like merging on shared keys, aggregating ratings per user or movie, and extracting user demographics) more efficient and reproducible when working with large-scale OTT datasets.

4.3.1.2 Structure of DataFrame



```
1 df_movies.head()
```

	Movie ID	Title	Genres
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy

Figure 4.2: Structure of Data frame

Insights

- The `df_movies.head()` output showcases the first five rows of the movies DataFrame, highlighting critical metadata fields: unique Movie ID, title (with year of release), and a list of associated genres.

- Movie genres are pipe-separated, which allows multi-genre classification and makes genre-based filtering and recommendations more flexible (e.g., recommending all comedy or family movies).
 - The presence of titles from the mid-1990s indicates the dataset includes older, possibly classic movies alongside more recent releases, enabling a broad range of recommendation scenarios spanning different eras and genres.

4.3.1.3 Exploded Movies Data Frame by Individual Genre

...	Movie ID	Title	Genres
0	1	Toy Story (1995)	Animation
0	1	Toy Story (1995)	Children's
0	1	Toy Story (1995)	Comedy
1	2	Jumanji (1995)	Adventure
1	2	Jumanji (1995)	Children's

Figure 4.3: Exploded Movies Data Frame by Individual Genre

Insights

- The genre column is split so each movie appears in multiple rows—one for each associated genre—making it easier to analyze genre-specific patterns or aggregate genre-wise statistics.
 - This format is ideal for counting the number of movies in each genre, finding genre popularity trends, or enabling multi-label genre-based recommendation queries.
 - Exploding genres allows more granular EDA (Exploratory Data Analysis) and supports visualization or analysis tasks that require each genre to be treated separately rather than in combination.

4.3.1.4 Grouping The Data Frame Based on Index

Figure 4.4: Grouping The Data Frame Based on Index

Insights

- The genre one-hot encoded DataFrame enables every movie to be represented in terms of its genre composition with binary indicators—essential for genre-based filtering, clustering, and similarity calculations.
- This structure makes it straightforward to analyze the prevalence of each genre in the dataset, build genre profiles for individual users, and feed genre data as input features for hybrid or content-based recommender systems.
- Summing over columns quickly reveals which genres are most common, and grouping by genre vectors allows advanced analytics such as genre trend detection, demographic genre preferences, or cross-genre popularity analysis.

4.3.1.5 List of Unique Movie Genres Identified in Dataset

```
1 unique_genere = genre_dummies.columns
2 unique_genere

Index(['Action', 'Adventure', 'Animation', 'Children's', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western'],
      dtype='object')
```

Figure 4.5: List of Unique Movie Genres Identified in Dataset

Insights

- The unique_genere output confirms that the dataset contains 19 distinct genres, ranging from mainstream categories (like Action, Comedy, and Drama) to more niche ones (such as Film-Noir, Musical, and Western).
- This variety ensures a rich dataset for both exploratory genre-based analysis and robust personalized recommendations, as users can be matched with less common genres based on individual preferences.
- Knowing the full set of unique genres supports downstream tasks like building genre popularity dashboards, segmenting user taste profiles, and implementing logic for multi-label genre recommendations.

4.3.1.6 Final Movies DataFrame with Genre Dummy Variables

```

1 final_movie_df = df_movies.join(genre_dummies, how='inner', lsuffix='', rsuffix='')
2 final_movie_df.drop('Genres', axis=1, inplace=True)
3 final_movie_df.head()

```

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
1	2	Jumanji	1995	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	
2	3	Grumpier Old Men	1995	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	
3	4	Waiting to Exhale	1995	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	
4	5	Father of the Bride Part II	1995	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 4.6: Final Movies Data Frame with Genre Dummy Variables

Insights

- The merged Data Frame represents each movie with a comprehensive set of metadata: Movie ID, Title, Release Year, followed by binary genre indicators that specify all genres applicable to each title.
- This structure facilitates fast genre-based queries and empowers recommendation algorithms (content-based, hybrid) to calculate movie similarity or filter suggestions based on the user's genre preferences.
- The approach ensures that every movie is mapped over all genre categories, which enhances feature richness for machine learning modeling and supports granular analytics such as multi-genre overlap or temporal trends by genre.

4.3.1.7 Ratings Data Frame with Derived Time Features

```

1 df_ratings.head()
2 df_ratings['Timestamp'] = pd.to_datetime(df_ratings['Timestamp'], unit='s')
3 df_ratings['Month'] = df_ratings['Timestamp'].dt.strftime('%b')
4 df_ratings['Year'] = df_ratings['Timestamp'].dt.year
5 df_ratings['Day'] = df_ratings['Timestamp'].dt.strftime('%a')
6 df_ratings['Hour'] = df_ratings['Timestamp'].dt.hour

```



1 df_ratings

...

	UserID	MovieID	Rating	Timestamp	Month	Year	Day	Hour
0	1	1193	5	2000-12-31 22:12:40	Dec	2000	Sun	22
1	1	661	3	2000-12-31 22:35:09	Dec	2000	Sun	22
2	1	914	3	2000-12-31 22:32:48	Dec	2000	Sun	22
3	1	3408	4	2000-12-31 22:04:35	Dec	2000	Sun	22
4	1	2355	5	2001-01-06 23:38:11	Jan	2001	Sat	23
...
1000204	6040	1091	1	2000-04-26 02:35:41	Apr	2000	Wed	2
1000205	6040	1094	5	2000-04-25 23:21:27	Apr	2000	Tue	23
1000206	6040	562	5	2000-04-25 23:19:06	Apr	2000	Tue	23
1000207	6040	1096	4	2000-04-26 02:20:48	Apr	2000	Wed	2
1000208	6040	1097	4	2000-04-26 02:19:29	Apr	2000	Wed	2

1000209 rows × 8 columns

Figure 4.7: Ratings Data Frame with Derived Time Features

Insights

- The ratings DataFrame combines core user-movie-rating information with additional temporal features (month, year, day of week, hour) derived from the original timestamp column, enabling detailed time-based analysis of viewing and rating patterns.
- With over 1 million rows, this structure supports exploration of user habits—revealing when ratings are most frequently given (by month or hour), which days of the week are popular for watching and rating, and time trends in user engagement.
- These engineered time fields empower additional segmentation, temporal aggregation, and predictive modeling (such as identifying peak engagement periods or time-linked recommendation campaigns) for an OTT platform.

4.3.1.8 Users Data Frame Structure

1 df_users.head()

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

Figure 4.8: Users Data Frame Structure

Insights

- The users DataFrame displays key demographic attributes—UserID, Gender, Age group (numerical code), Occupation (code), and Zip-code—for all users in the dataset.
- These fields enable the analysis of user distribution by gender, age, occupation, and location, which supports richer segmentation and helps answer business questions about which demographics watch and rate the most movies.
- Encoding age and occupation as numbers allows straightforward grouping, filtering, and merging for downstream analytics, such as exploring content preferences by demographic segment or personalizing movie recommendations to users with similar profiles.

4.3.1.9 Merged Movies and Ratings DataFrame with Genre Features

1 df_movies_ratings = pd.merge(final_movie_df, df_ratings, how='inner', left_on='Movie ID', right_on='MovieID')
2 df_movies_ratings.head()

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mys
0	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0
1	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0
2	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0
3	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0
4	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0

Figure 4.9: Merged Movies and Ratings DataFrame with Genre Features

Insights

- This DataFrame integrates movie metadata, including genre dummy variables, with ratings information—enabling each rating to be linked with detailed genre characteristics of the rated movie.
- Such integration supports genre-specific rating analysis and downstream modeling, making it possible to explore user preferences for particular genres and enhance personalization in a recommender system.
- By merging these datasets, the platform can perform rich analytics, such as identifying which genres are rated most highly, tracking trends over time, and generating targeted recommendations based on both user rating history and content type.

4.3.1.10 Movies-Ratings DataFrame with Genre Features (MovieID Dropped)

```

1 df_movies_ratings.drop('MovieID', axis=1, inplace=True)
2 df_movies_ratings.head()

```

...	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance
0	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0
1	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0
2	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0
3	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0
4	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0

Figure 4.10: Movies-Ratings Data Frame with Genre Features (MovieID Dropped)

Insights

- After dropping the redundant MovieID column, the DataFrame cleanly presents each movie instance (by title and release year) alongside its corresponding genre indicator variables, streamlining further analysis.
- The structure is ideal for exploring how ratings relate to genre composition, supporting robust model building for collaborative and content-based filtering—the genre features can be easily used for similarity calculations or regression/classification tasks.
- This view enhances interpretability and simplifies aggregation, such as determining average or total ratings per genre, spotlighting trends, and identifying movies that appeal to users across multiple genres.

4.3.1.11 Fully Merged Movies, Ratings, and User Data Frame

```

1
2 df = pd.merge(df_movies_ratings, df_users, on='UserID', how='inner')
3 df.head()

```

	Movie ID	Title	ReleaseYear	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
4	1	Toy Story	1995	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 4.11: Fully Merged Movies, Ratings, and User Data Frame

Insights

- This DataFrame results from merging all three core datasets—movies with genres, ratings, and user demographics—providing an integrated view per rating along with movie attributes and user information.
- Such a structure enables nuanced analysis of how demographic groups interact with different movie genres, forming the foundation for collaborative filtering and personalized recommendation approaches that incorporate both user and item features.
- The comprehensive format supports segmentation, data-driven feature engineering, and the building of robust models for user-user, item-item, and hybrid recommendation systems, directly aligning with OTT business needs for targeted experience and content optimization.

4.3.1.12 Decoding Occupation and Age Columns in Final Data Frame

```
1 with open('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/12',
2     pickle.dump(df, f)
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37     , inplace=True)
```

Figure 4.12: Decoding Occupation and Age Columns in Final Data Frame

Insights

- The code replaces numerical codes in Occupation and Age columns with descriptive string labels, transforming the DataFrame for improved interpretability in analyses and visualizations.

- Enriching user profiles with readable group names supports clearer insights on demographic trends (e.g., which age group or occupation rates more movies) and enables more user-friendly dashboards, reporting, and marketing strategies in the OTT domain.
- This step is crucial before clustering, segmentation, or machine learning modeling that incorporates user attributes, as it ensures results and recommendations can be communicated effectively to stakeholders or end-users.

4.3.2 Exploratory Data Analysis (EDA)

4.3.2.1 Data frame Shape

```
▶ 1 df.shape
...
... (1000209, 32)
```

Figure 4.13: Data Frame Shape

Insights

- The final merged DataFrame contains 1,000,209 rows and 32 columns, indicating a highly comprehensive dataset that pairs more than a million individual movie ratings with corresponding movie, genre, and user demographic features.
- This rich, wide dataset supports complex analytics, robust modeling for movie recommendation systems, demographic segmentation, and enables in-depth feature engineering for advanced collaborative and hybrid filtering strategies.
- The scale and structure of the data will facilitate reliable generalization and high-quality, personalized recommendations across diverse user and content segments for an OTT streaming platform.

4.3.2.2 Dataframe Information



1 df.info()

```
... <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 32 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Movie ID    1000209 non-null   int64  
 1   Title        1000209 non-null   object  
 2   ReleaseYear  1000209 non-null   object  
 3   Action        1000209 non-null   int64  
 4   Adventure    1000209 non-null   int64  
 5   Animation    1000209 non-null   int64  
 6   Children's   1000209 non-null   int64  
 7   Comedy        1000209 non-null   int64  
 8   Crime         1000209 non-null   int64  
 9   Documentary  1000209 non-null   int64  
 10  Drama         1000209 non-null   int64  
 11  Fantasy       1000209 non-null   int64  
 12  Film-Noir    1000209 non-null   int64  
 13  Horror        1000209 non-null   int64  
 14  Musical        1000209 non-null   int64  
 15  Mystery       1000209 non-null   int64  
 16  Romance       1000209 non-null   int64  
 17  Sci-Fi        1000209 non-null   int64  
 18  Thriller      1000209 non-null   int64  
 19  War           1000209 non-null   int64  
 20  Western        1000209 non-null   int64  
 21  UserID         1000209 non-null   int64  
 22  Rating         1000209 non-null   int64  
 23  Timestamp     1000209 non-null   datetime64[ns] 
 24  Month          1000209 non-null   object  
 25  Year           1000209 non-null   int32  
 26  Day            1000209 non-null   object  
 27  Hour           1000209 non-null   int32  
 28  Gender          1000209 non-null   object  
 29  Age             1000209 non-null   object  
 30  Occupation     1000209 non-null   object  
 31  Zip-code       1000209 non-null   object  
dtypes: datetime64[ns](1), int32(2), int64(21), object(8)
memory usage: 236.6+ MB
```

Figure 4.14: Data Frame Information

Insights

- The Data Frame contains 1,000,209 entries and a comprehensive set of 32 columns, covering movie metadata, genre indicators, user demographics, and detailed temporal features like month, year, day, and hour.
- All columns have complete (non-null) data, ensuring reliability for modeling and analytics without the need for missing value imputation or exclusion.

- The data types are mostly integer and object, with well-typed time-series columns—this structure supports a wide array of EDA, machine learning, and aggregation operations at scale (memory footprint ~236 MB), making it fully suitable for collaborative and content-based filtering on an OTT platform.

4.3.2.3 Data Frame Description for Numerical columns

	1 df.describe()																		
...	Movie ID	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-			
count	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	1000209.000	
mean	1865.540	0.257	0.134	0.043	0.072	0.357	0.080	0.008	0.354	0.036	0.018	0.076	0.042	0.040	0.147	0.1			
min	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0	
25%	1030.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0	
50%	1835.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0	
75%	2770.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0	
max	3952.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	
std	1096.041	0.437	0.341	0.203	0.259	0.479	0.271	0.089	0.478	0.187	0.134	0.266	0.199	0.196	0.355	0.3			

Figure 4.15: Data Frame Description for Numerical columns

Insights

- The summary statistics reveal that Comedy, Drama, and Action genres are the most frequently represented in the dataset, having higher mean values compared to niche genres like Documentary, Musical, or Film-Noir.
- Action movies are present in approximately 26% of all movie-rating records, indicating popularity, while genres such as Animation, Musical, and Mystery have much lower mean and frequency, showing lesser representation and engagement.
- The minimum and maximum values for genre binary columns are 0 and 1 as expected, confirming robust one-hot encoding; the relatively high standard deviations for Comedy and Drama further highlight their diverse cross-genre overlap in movies and ratings.

4.3.2.4 Data Frame Description for Object type columns

	1 df.describe(include=['object'])									
...	Title	ReleaseYear	Month	Day	Gender	Age	Occupation	Zip-code		
count	1000209	1000209	1000209	1000209	1000209	1000209	1000209	1000209	1000209	1000209
unique	3664	81	12	7	2	7	21	3439		
top	American Beauty	1999	Nov	Mon	M	25-34	college/grad student	94110		
freq	3428	86833	295461	173931	753769	395556	131032	3802		

Figure 4.16: Data Frame Description for Object type columns

Insights

- The most common movie in the ratings dataset is "American Beauty," indicating it may have broad appeal or cultural significance, while the most popular release year is 1999, possibly reflecting a rich slate of releases or heightened user nostalgia for that era.
- The predominant user group consists of males aged 25–34, with "college/grad student" as the leading occupation; this demographic is likely the platform's core rating and engagement base, influencing both marketing and content curation strategies.
- More than 3400 different movies and over 3400 unique zip codes are represented, denoting a diverse catalog and wide geographic reach, but the concentration on a few titles and zip codes (e.g., 94110 is most frequent) also suggests regional or blockbuster-driven patterns in viewing and rating habits.

4.3.2.5 Data Frame Null Values Check

```
1 df.isna().sum()
```

...	Movie ID	0
	Title	0
	ReleaseYear	0
	Action	0
	Adventure	0
	Animation	0
	Children's	0
	Comedy	0
	Crime	0
	Documentary	0
	Drama	0
	Fantasy	0
	Film-Noir	0
	Horror	0
	Musical	0
	Mystery	0
	Romance	0
	Sci-Fi	0
	Thriller	0
	War	0
	Western	0
	UserID	0
	Rating	0
	Timestamp	0
	Month	0
	Year	0
	Day	0
	Hour	0
	Gender	0
	Age	0
	Occupation	0
	Zip-code	0

Figure 4.17: Data Frame Null Values Check

Insights

- The DataFrame contains zero missing values in any column, confirming complete data integrity across all movie, rating, user, time, and genre features.
- This absence of missing values means no imputation or dropped records will be needed, resulting in maximum usable data for training machine learning models and generating reliable business analytics.
- With a fully populated dataset, downstream analysis and recommendation system experiments will have stronger validity and reduced risk of bias introduced by incomplete information.

4.3.2.6 Unique Column Values

```
1 for i in df.columns:  
2     print(i, df[i].nunique())  
  
... Movie ID 3706  
Title 3664  
ReleaseYear 81  
Action 2  
Adventure 2  
Animation 2  
Children's 2  
Comedy 2  
Crime 2  
Documentary 2  
Drama 2  
Fantasy 2  
Film-Noir 2  
Horror 2  
Musical 2  
Mystery 2  
Romance 2  
Sci-Fi 2  
Thriller 2  
War 2  
Western 2  
UserID 6040  
Rating 5  
Timestamp 458455  
Month 12  
Year 4  
Day 7  
Hour 24  
Gender 2  
Age 7  
Occupation 21  
Zip-code 3439
```

Figure 4.18: Unique Column Values

Insights

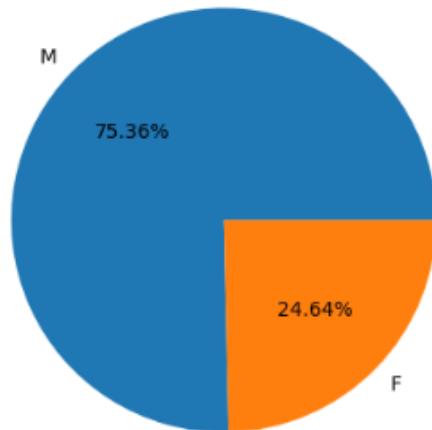
- The dataset contains 3,706 unique movies and 3,664 unique titles, indicating a near one-to-one mapping between Movie ID and Title, with very few duplicates or alternate versions.
- Genre columns (like Action, Comedy, etc.) are encoded as binary (0/1), having only two unique values—essential for efficient genre filtering and analysis in recommender systems.
- Rating counts span five unique values (from 1 to 5), Zip-code covers a broad geographic spread with 3,439 unique entries, and UserID features 6,040 unique users—enabling robust segmentation and ensuring rich diversity in user, content, and geographic representation.
- Temporal fields like Timestamp, Month (12 unique), Year (4 unique), Day (7 unique), and Hour (24 unique) provide granular time-based analysis capabilities for behavioral and engagement trend detection. Seven Age groups and 21 Occupation categories further enrich the potential for targeted, personalized business strategies.

4.3.3 Univariate Analysis

4.3.3.1 Proportion of Gender

```
1 plt.pie(df['Gender'].value_counts(), labels=df['Gender'].value_counts().index, autopct='%0.2f%%')
2 plt.title('Proportion of Gender in the given dataset')
3 plt.show()
4 display(df['Gender'].value_counts())
```

*** Proportion of Gender in the given dataset



```
count
Gender
M    753769
F    246440
dtype: int64
```

Figure 4.19: Proportion of Gender in given dataset

Insights

- The gender distribution in the dataset is considerably skewed: 75.36% of ratings were given by male users, while only 24.64% came from female users.
- This significant imbalance indicates that the core user base is predominantly male, which has direct implications for recommendation strategies (e.g., tailoring or analyzing content preferences by gender segment for the OTT platform).
- The high absolute counts (over 750,000 ratings from males and nearly 250,000 from females) provide a robust sample for reliable demographic and behavioral analyses, although future strategies may consider actions for engaging more female users to balance content and platform appeal.

4.3.3.2 Distribution of Age

```

1 def annotate(ax, rotation = False):
2     for patch in ax.patches: # Loop through each bar
3         if rotation: # For horizontal bars
4             x = patch.get_width() # Get the width (value of the bar)
5             y = patch.get_y() + patch.get_height() / 2 # Center the annotation vertically
6             ax.annotate(f'{x}', (x + 0.5, y), ha='left', va='center') # Adjust position
7     else: # For vertical bars
8         x = patch.get_x() + patch.get_width() / 2 # Center the annotation horizontally
9         y = patch.get_height() # Get the height (value of the bar)
10        ax.annotate(f'{y}', (x, y + 0.5), ha='center', va='bottom') # Adjust position

1 plt.title('Distribution of Age')
2 ax = sns.barplot(df.Age.value_counts().reset_index(), x='Age', y='count')
3 annotate(ax)
4 plt.show()

```

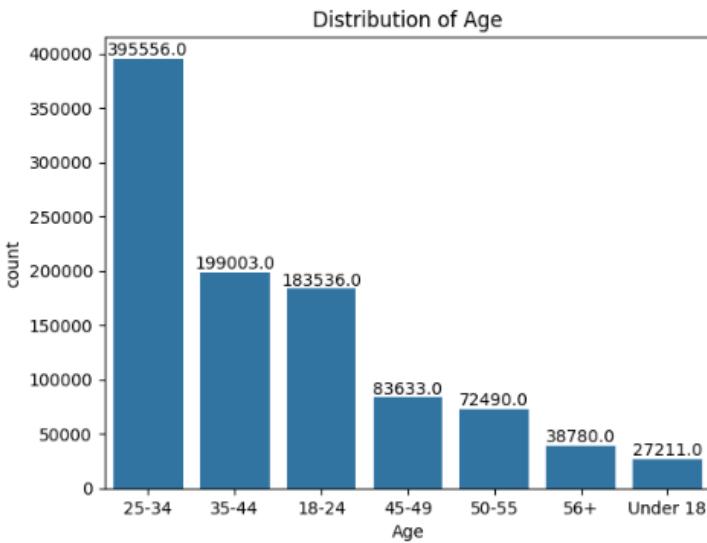


Figure 4.20: Distribution of Age

Insights

- Most of the ratings in the dataset belong to users in the age group 25–34, followed by 35–44 and 18–24, indicating these are the most active and engaged segments on the OTT platform.
- The counts decline sharply for older age groups (45–49, 50–55, 56+) and the "Under 18" group, suggesting a skew toward younger adults and relatively low engagement from teenage and senior viewers.
- Content curation and recommendation algorithms should especially account for this dominant demographic (25–34), as their preferences are likely to drive overall performance and guide business decisions in marketing and content acquisition.

4.3.3.3 Distribution of Movie Ratings

```
1 plt.title('Distribution of Movie Ratings')
2 ax = sns.barplot(df.Rating.value_counts().reset_index(), x='Rating', y='count')
3 annotate(ax)
4 plt.show()
```

...

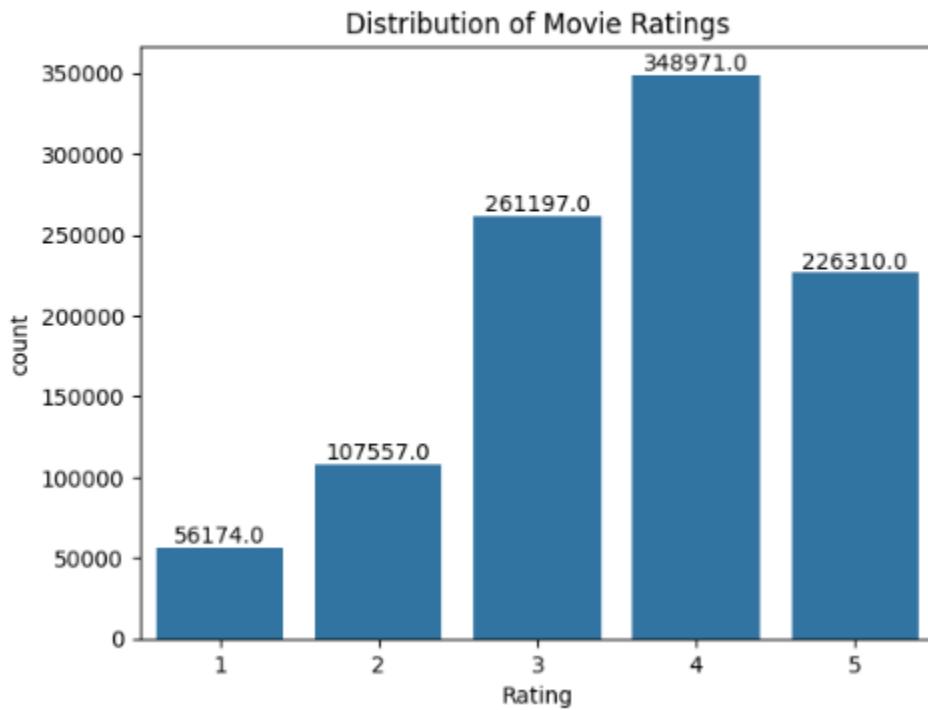


Figure 4.21: Distribution of Movie Ratings

Insights

- The distribution of movie ratings is right-skewed: the majority of ratings are 3, 4, or 5, with 4 being the most common, indicating generally positive reception for many titles in the dataset.
- Extremely low ratings (1 and 2) are much less frequent, suggesting that users either enjoy most of what they watch or tend to avoid rating movies poorly.
- This positivity bias reflects well on user satisfaction with available content and has implications for recommendation algorithms and business reporting—highlighting the likelihood of increased engagement and reduced churn when highly-rated recommendations are prioritized.

4.3.3.4 Distribution of Movie Ratings by Day of Week

```

1 plt.title('Number of Movie Ratings by Day')
2 ax = sns.barplot(df.Day.value_counts().reset_index(), x='Day', y='count')
3 annotate(ax)
4 plt.show()

```

...

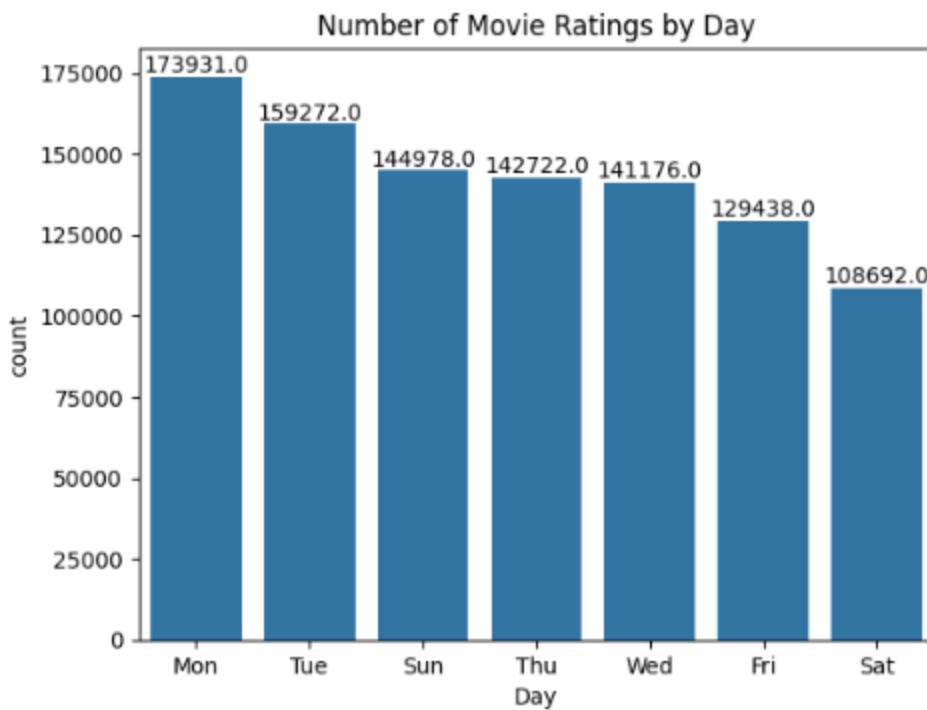


Figure 4.22: Number of Movie Ratings by Day

Insights

- Movie ratings are most frequently given on Mondays and Tuesdays, with activity gradually declining toward the weekend, and the lowest number of ratings observed on Saturdays.
- This pattern suggests that users engage most with the platform and record feedback at the start of the week, possibly reflecting viewing habits over the weekend followed by rating/reviewing later, or heightened weekday digital activity.
- For the OTT platform, these trends provide guidance on scheduling engagement campaigns, content releases, and marketing pushes to align with periods of peak user interaction, maximizing the impact of recommendation and retention strategies.

4.3.3.5 Distribution of Movie Ratings by Day of Month

```

1 plt.figure(figsize=(10,5))
2 ax = sns.barplot(df.Month.value_counts().reset_index(), x='Month', y='count')
3 plt.title('Number of Movie Ratings by Month')
4 annotate(ax)
5 plt.show()

```

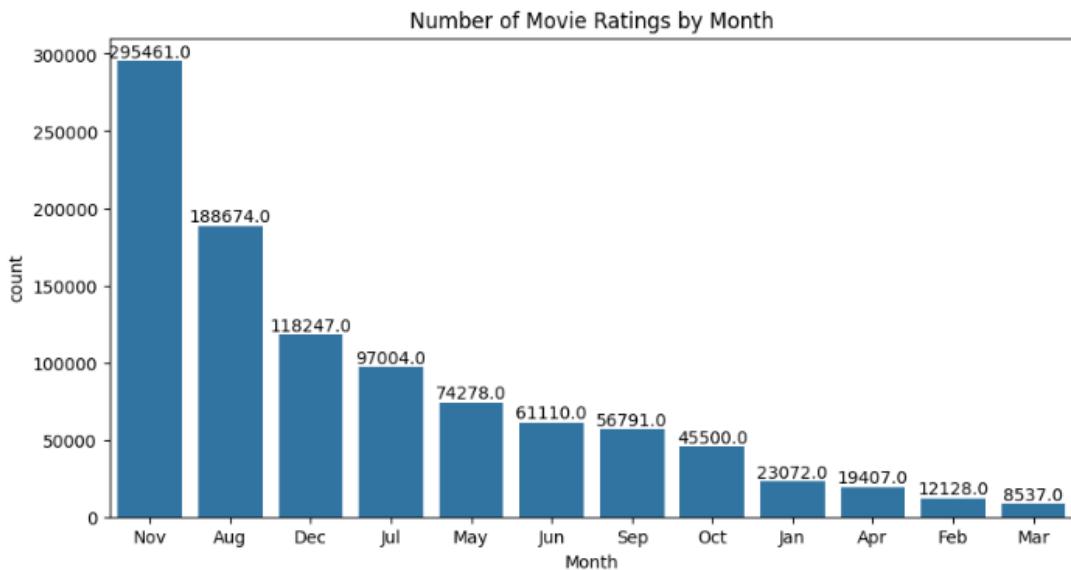


Figure 4.23: Number of Movie Ratings by Month

Insights

- November and August are the months with the highest rating activity, followed by December and July, indicating seasonal peaks in user engagement—possibly aligned with holidays, vacations, or content release schedules.
- Rating volume drops steadily from the summer and winter peaks toward the early months of the year (January, February, March), suggesting that user activity is lowest in late winter and early spring.
- These trends are valuable for OTT platforms in planning promotional campaigns, major content releases, and feature launches—maximizing visibility and engagement during months when user activity is naturally high.

4.3.3.6 Number of People by Occupation

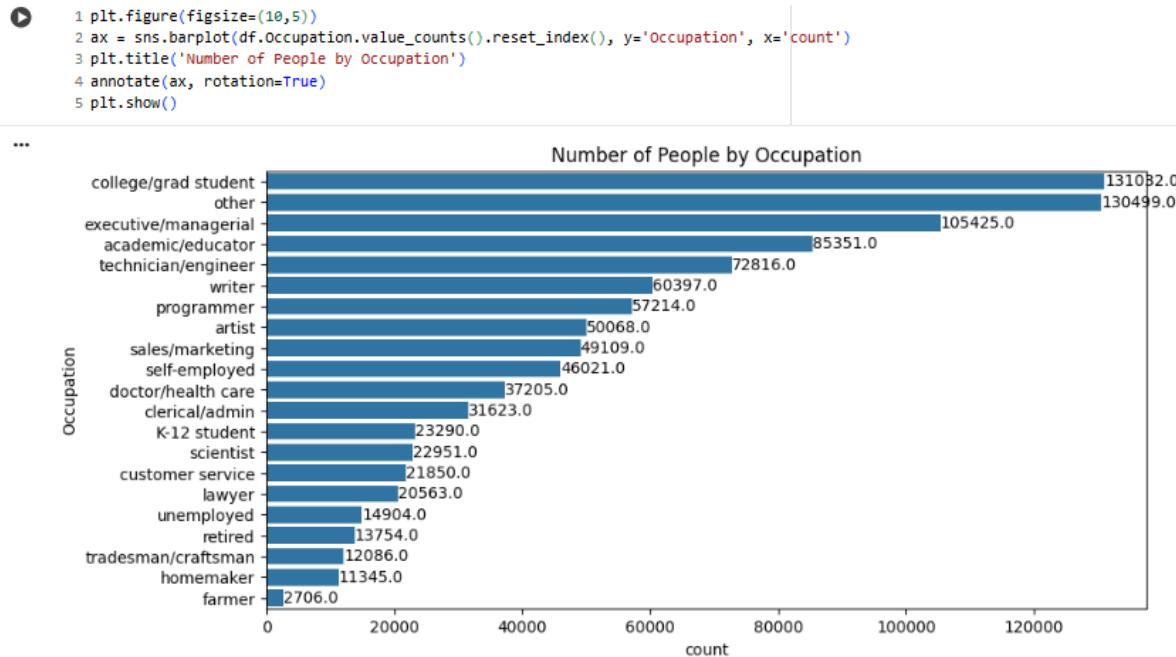


Figure 4.24: Number of People by Occupation

Insights

- The two largest user segments in the dataset are "college/grad student" and "other," each accounting for over 130,000 ratings, indicating a strong presence of educated and/or younger adults in the platform's audience.
- Occupations such as "executive/managerial," "academic/educator," and "technician/engineer" are also highly represented, suggesting engagement from professional and technical user groups with substantial disposable time or interest in media.
- Representation drops off for roles like "farmer," "homemaker," and "tradesman/craftsman," indicating content, marketing, and recommendation strategies should be tailored primarily to student/professional audiences, while outreach and diversity initiatives could target under-engaged occupational groups to broaden platform reach.

4.3.3.7 Time when people watch movies and give ratings

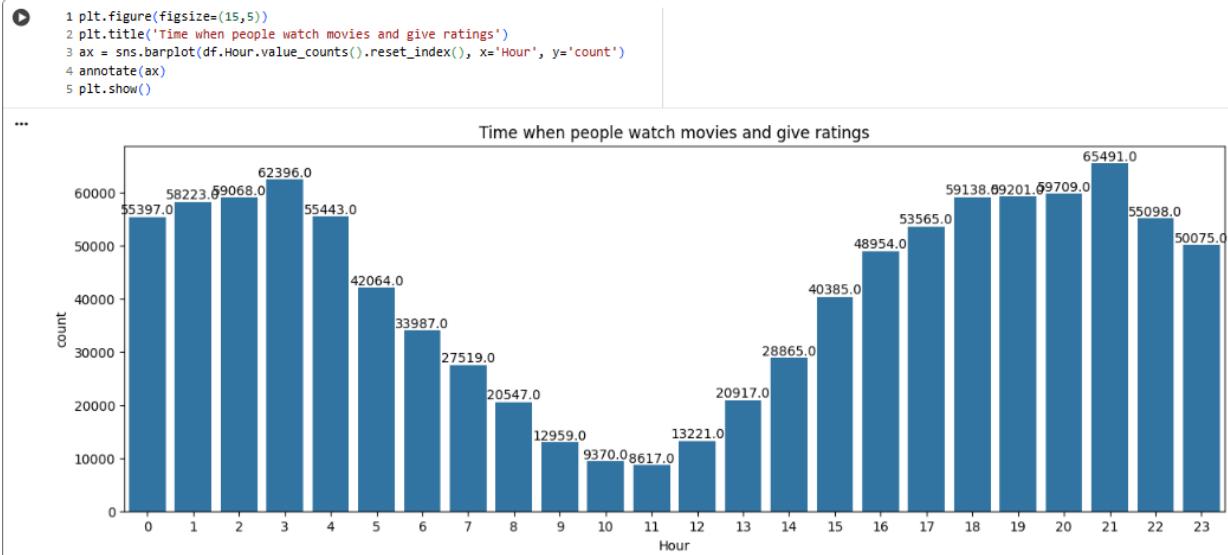


Figure 4.25: Time when people watch movies and give ratings

Insights

- There are two clear peaks in user movie-watching and rating activity: one in the very late evening (20:00–23:00) and another, unexpectedly, in the early morning hours (midnight to 04:00), with the highest single-hour activity at 22:00.
- Afternoon and early evening hours (15:00–19:00) also show moderate engagement, while activity is lowest between 08:00 and 13:00, suggesting users are much less likely to interact with the platform during traditional work or school hours.
- This temporal insight can help the OTT platform optimize push notifications, release schedules, and personalized campaign timings to coincide with periods of demonstrated user engagement, maximizing both content reach and the likelihood of obtaining user ratings.

4.3.3.8 Top 20 years in which the highest number of movies were released

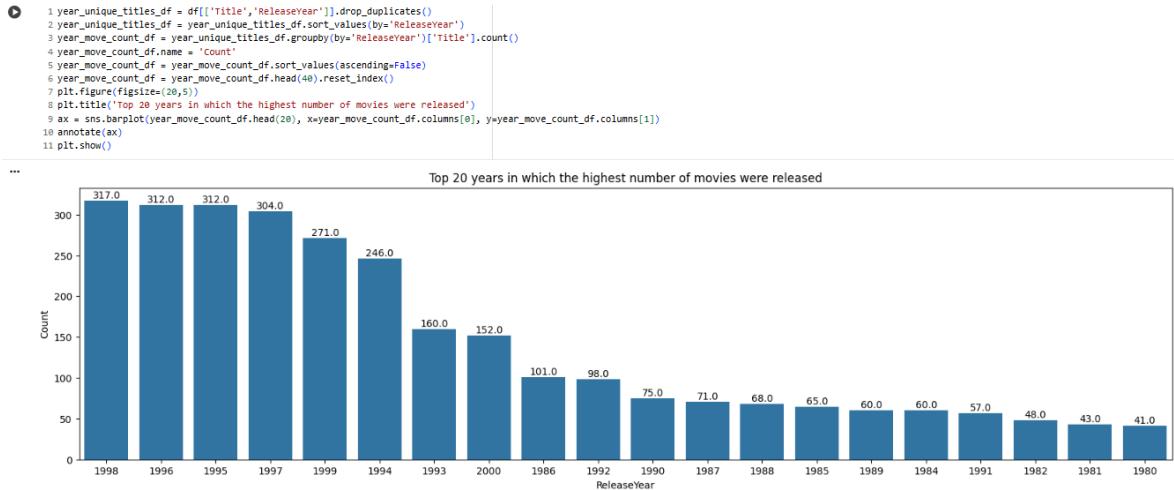


Figure 4.26: Top 20 years in which the highest number of movies were released

Insights

- The years 1996–1999 saw the highest number of movie releases in the dataset, with each of these years contributing over 300 unique titles, pointing to a boom in movie production or data capture during this period.
- The late 1990s generally dominate the top 20 list for release volumes, followed by a sharp drop-off in annual movie releases after 2000, possibly due to dataset collection limits or genuine shifts in industry output.
- The “top 20 years” chart provides valuable context for content age distribution in the platform: older titles (especially from the late 1980s and 1990s) form a substantial portion of the catalog, directly influencing recommendation diversity, user nostalgia engagement, and catalog expansion strategy.

4.3.4 Bivariate Analysis

4.3.4.1 Genre Count w.r.t Gender

```

1 def get_gender_count_by_genre(_df, genre):
2     _df = _df[_df[genre] == 1].groupby('Gender')['UserID'].count()
3     _df.name = genre
4     return _df
5
6 genre_df_list = []
7 for i in range(len(unique_genre)):
8     _genre_count_df = get_gender_count_by_genre(df, unique_genre[i])
9     genre_df_list.append(_genre_count_df)
10
11 gender_genre_df = pd.concat(genre_df_list, axis=1)
12 gender_genre_df.reset_index(inplace=True)
13 melted_gender_genre_df = gender_genre_df.melt(id_vars=['Gender'], var_name='Genre', value_name='Count', ignore_index=True)
14
15
16 fig, axes = plt.subplots(1, 2, figsize=(15, 10), sharey=True)
17 plt.suptitle('Genre Count w.r.t to Gender')
18 for idx, gender in enumerate(['M', 'F']):
19
20     temp_df = melted_gender_genre_df[melted_gender_genre_df['Gender'] == gender]
21     temp_df = temp_df.drop(columns=['Gender'])
22     temp_df.sort_values(by='Count', ascending=False, inplace=True)
23
24     ax = axes[idx]
25     _gender = 'Male' if gender == 'M' else 'Female'
26     sns.barplot(data=temp_df, x='Count', y='Genre', ax=ax)
27     ax.set_title(f'{_gender}')
28     ax.set_xlabel('Count')
29     ax.set_ylabel('Genre')
30
31     annotate(ax, rotation=True)
32
33 plt.tight_layout()
34 plt.show()

```

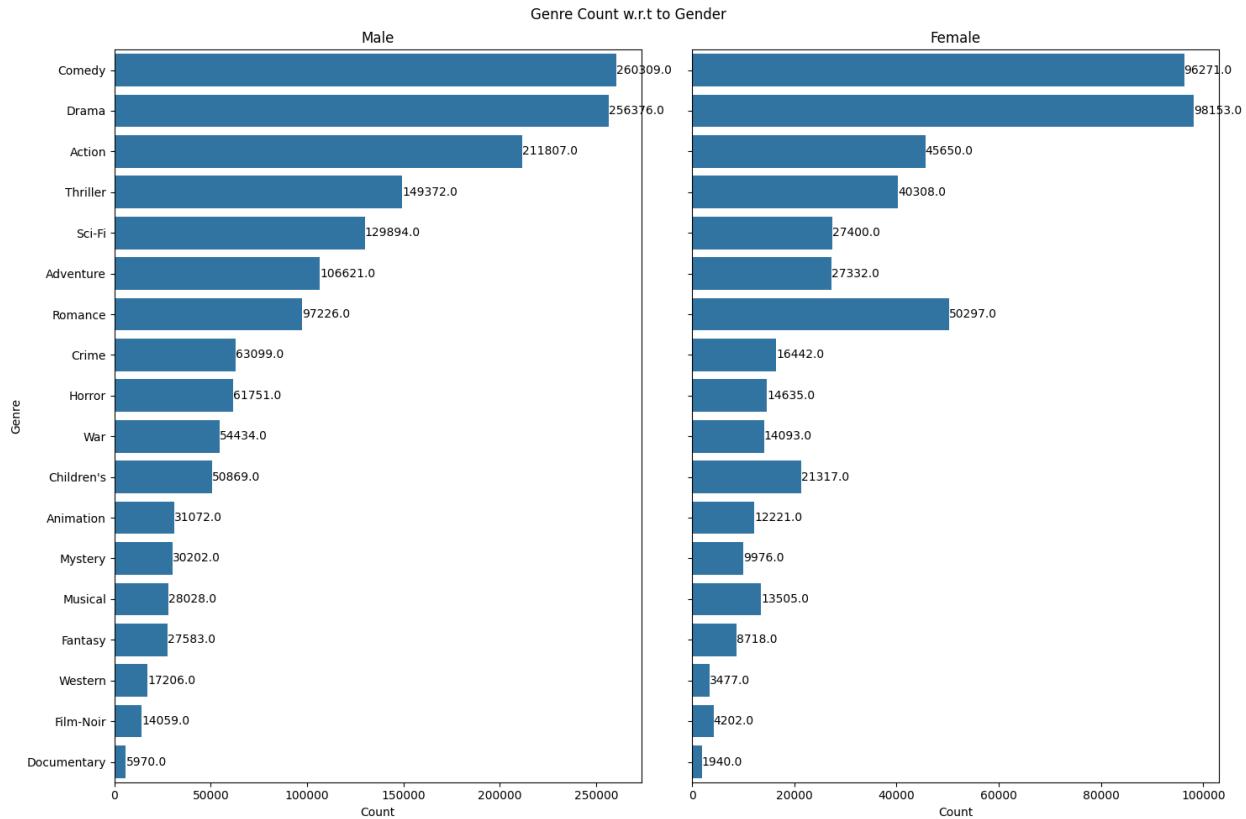


Figure 4.27: Genre Count w.r.t Gender

Insights

- Both male and female users rate Comedy and Drama movies most frequently, making these genres universally popular across genders.
- Male users show a distinctive preference for Action, Thriller, Sci-Fi, and Adventure genres, reflecting a heavier engagement with movies featuring dynamic, exciting, or speculative themes.
- Female users, while also favoring Comedy and Drama, show higher relative engagement with genres like Romance and slightly greater balance across Adventure and Sci-Fi than some niche genres; however, overall counts remain much lower than males across most categories.
- Documentary, Film-Noir, and Western genres receive the least engagement from both genders, indicating these may be niche interests or underrepresented within the platform's content and user population.

4.3.4.2 Genre w.r.t Age

```
1 def get_genre_count_by_feature(_df, genre, feature, feature_order):
2     _df = _df[_df[genre] == 1].groupby(feature)['UserID'].count()
3     _df.name = genre
4     _df = _df.reset_index()
5     _df[feature] = pd.Categorical(_df[feature], categories=feature_order, ordered=True)
6     _df = _df.sort_values(feature)
7     _df = _df.set_index('Age')
8     return _df
9
10 age_genre_list = []
11 custom_age_order = ['Under 18', '18-24', '25-34', '35-44', '45-49', '50-55', '56+']
12 for i in range(len(unique_genre)):
13     age_genre_list.append(get_genre_count_by_feature(df, unique_genre[i], 'Age', custom_age_order))
14
15 df_final = pd.concat(age_genre_list, axis=1)
16 df_final = df_final.T
17 df_final.index.name = 'Genre'
18
19 # Plotting logic
20 n_cols = 2
21 n_rows = (len(df_final.columns) + 1) // 2 # Ceiling division to ensure enough rows
22 fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 20), sharey=True)
23 axes = axes.flatten() # Flatten to easily index axes in a loop
24
25 for i, age_group in enumerate(df_final.columns):
26     temp_df = df_final[[age_group]].sort_values(by=age_group, ascending=False)
27     temp_df = temp_df.reset_index()
28
29     # Plot on the corresponding subplot
30     ax = axes[i]
31     sns.barplot(data=temp_df, x=age_group, y='Genre', ax=ax)
32     ax.set_title(f'{age_group}')
33     ax.set_xlabel('Count')
34     ax.set_ylabel('Genre')
35
36     annotate(ax, rotation=True)
37
38 # Remove empty subplots if any
39 for j in range(i + 1, len(axes)):
40     fig.delaxes(axes[j])
41
42 fig.suptitle('Age w.r.t to Genre', fontsize=16, y=1.02) # Adjust y to avoid overlap
43 plt.tight_layout() # Adjust layout to prevent overlap
44 plt.show()
```

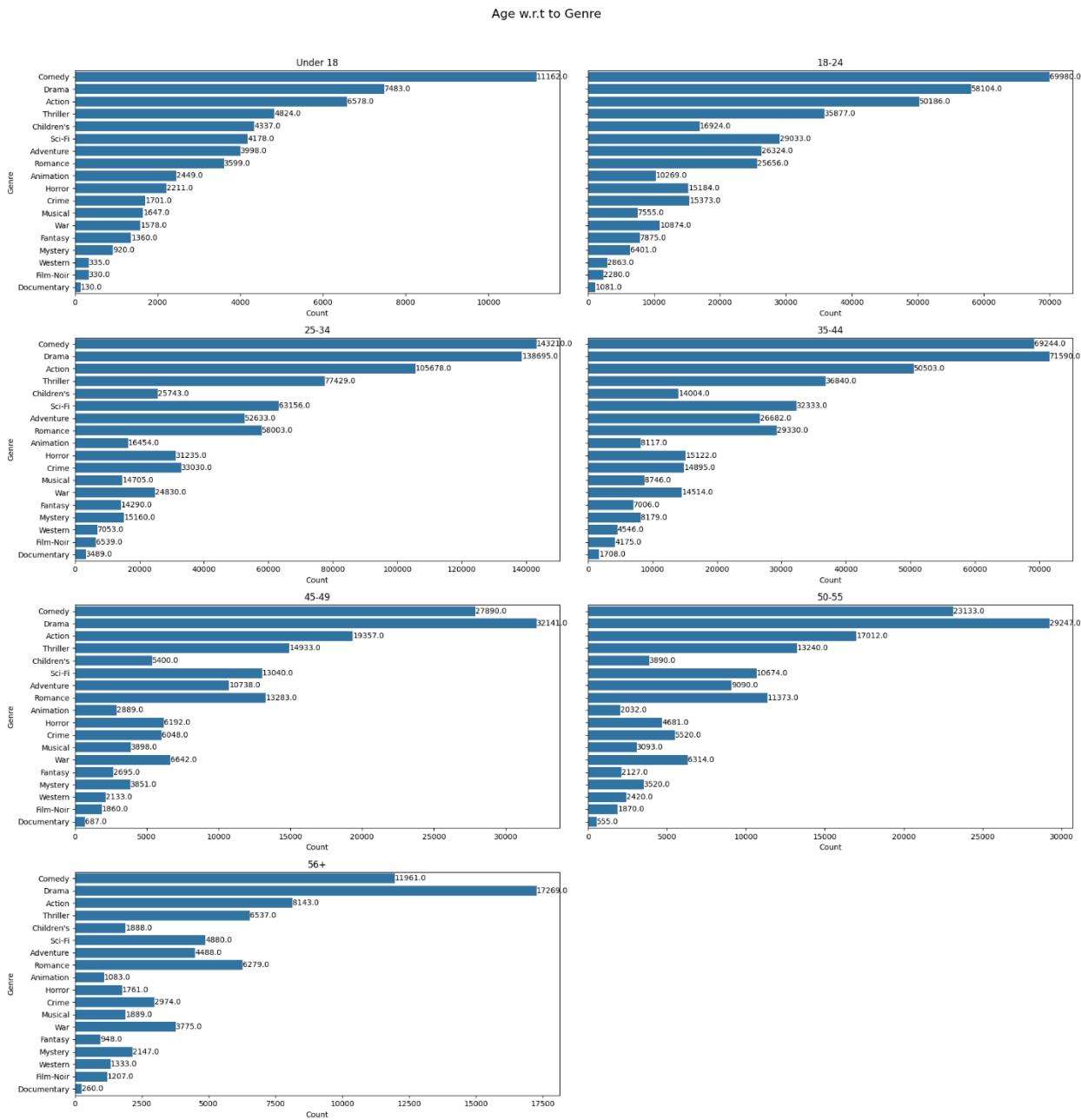


Figure 4.28: Age w.r.t to Genre

Insights

- All age groups rate Comedy and Drama highest, making them the most popular genres universally; Action, Thriller, and Sci-Fi consistently follow these, especially among younger audiences.
- The 18–24 and 25–34 age groups contribute the most counts across all genres, particularly for Comedy, Drama, Action, and Sci-Fi, showing their heavy engagement and strong influence on platform preferences.

- Genre variety narrows and counts decrease among the 56+ segment, where Comedy and Drama remain dominant but with lower counts for Action, Thriller, Sci-Fi, and niche genres, suggesting older users focus on lighter or story-driven content, while Animation, Children's, and Adventure are more visible in the under-18 and 18–24 groups.
- Even for smaller age groups, genre patterns mirror dominant trends, which can guide age-personalized recommendations and curated lists for maximized user satisfaction in an OTT environment.

4.3.4.3 Genre w.r.t Occupation

```

1 def get_genre_count_by_occupation(_df, occupation, genres, genre_order):
2     # Filter for the specific occupation
3     _df = _df[_df['Occupation'] == occupation]
4     # Count users for each genre where genre column == 1
5     counts = []
6     for genre in genres:
7         count = _df[_df[genre] == 1]['UserID'].count()
8         counts.append(count)
9     # Create DataFrame with genre counts
10    result_df = pd.DataFrame({
11        'Genre': genres,
12        'Count': counts
13    })
14    # Apply categorical order to genres (optional, for consistency)
15    result_df['Genre'] = pd.Categorical(result_df['Genre'], categories=genre_order, ordered=True)
16    # Sort by count for plotting
17    result_df = result_df.sort_values('Count', ascending=False)
18    return result_df
19
20 # List of occupations
21 custom_occupation_order = [
22     'K-12 student', 'homemaker', 'programmer', 'technician/engineer',
23     'academic/educator', 'clerical/admin', 'self-employed', 'other',
24     'executive/managerial', 'college/grad student', 'writer',
25     'retired', 'scientist', 'artist', 'customer service',
26     'sales/marketing', 'doctor/health care', 'unemployed', 'lawyer',
27     'farmer', 'tradesman/craftsman'
28 ]
29
30 # Plotting logic
31 n_cols = 4 # Exactly 4 columns per row
32 n_rows = (len(custom_occupation_order) + n_cols - 1) // n_cols # Ceiling division for 21 occupations (6 rows)
33 fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 5 * n_rows), sharey=True)
34 axes = axes.flatten() # Flatten for easy indexing
35
36 for i, occupation in enumerate(custom_occupation_order):
37     # Get genre counts for the current occupation
38     temp_df = get_genre_count_by_occupation(df, occupation, unique_genre, unique_genre)
39
40     # Plot on the corresponding subplot
41     ax = axes[i]
42     sns.barplot(data=temp_df, x='Count', y='Genre', ax=ax)
43     ax.set_title(f'{occupation}')
44     ax.set_xlabel('User Count')
45     ax.set_ylabel('Genre')
46
47     annotate(ax, rotation=True)
48
49 # Remove empty subplots (last 3 in 6x4 grid)
50 for j in range(i + 1, len(axes)):
51     fig.delaxes(axes[j])
52
53 fig.suptitle('Genres w.r.t Occupation', fontsize=16, y=1.02) # Figure-level title
54 plt.tight_layout() # Adjust layout
55 plt.show()

```

Genres w.r.t Occupation



Figure 4.29: Genres w.r.t Occupation

Insights

- Across nearly every occupation, Comedy and Drama dominate as the most frequently rated genres, demonstrating their universal popularity irrespective of profession.
- Technical professions like "programmer," "technician/engineer," and "academic/educator" exhibit stronger engagement with Sci-Fi, Action, and Thriller genres, while "college/grad student" and "executive/managerial" also favor these genres, suggesting genre affinity rooted in occupation and possibly lifestyle or education.
- Occupations such as "homemaker," "retired," "farmer," and "tradesman/craftsman" contribute lower genre counts overall and display more balanced preferences between Comedy, Drama, Romance, and Children's genres, hinting at distinct viewing habits for these user groups.
- Niche genres (Film-Noir, Documentary, Western) receive minimal ratings regardless of occupation, indicating broader trends of content engagement and potentially guiding content acquisition and recommendation strategies tailored to the primary professional segments on an OTT platform.

4.3.5 Modelling Recommender System

4.3.5.1 Recommender System Based on Pearson Correlation

4.3.5.1.1 Item Based

```
1 user_movie_matrix.columns[:10]
...
Index(['$1,000,000 Duck', "'Night Mother", "'Til There Was You", "'burbs, The",
       "...And Justice for All", '1-900', '10 Things I Hate About You',
       '101 Dalmatians', '12 Angry Men', '13th Warrior, The'],
      dtype='object', name='Title')
```

```
1 user_movie_matrix.loc[:, "'Til There Was You"]
```

```
...
'Til There Was You
```

```
UserID
1          0.000
2          0.000
3          0.000
4          0.000
5          0.000
...
6036        0.000
6037        0.000
6038        0.000
6039        0.000
6040        0.000
```

```
6040 rows × 1 columns
```

```
dtype: float64
```

```
1 similar_movies = user_movie_matrix.corrwith(user_movie_matrix.loc[:, "'Til There Was You"])
2 similar_movies = similar_movies.sort_values(ascending=False)
3 similar_movies[1:5].head(5)
```

```
0
Title
If Lucy Fell    0.267
Picture Perfect  0.256
To Gillian on Her 37th Birthday 0.241
Mad Love        0.231
Practical Magic  0.230
```

```
dtype: float64
```

```
1 sim_df = pd.DataFrame(similar_movies, columns=['Correlation'])
2 sim_df.sort_values('Correlation', ascending=False, inplace=True)
```

```
1 sim_df.iloc[1:, :]
```

Correlation	
Title	Correlation
If Lucy Fell	0.267
Picture Perfect	0.256
To Gillian on Her 37th Birthday	0.241
Mad Love	0.231
Practical Magic	0.230
...	...
Kelly's Heroes	-0.014
Boat, The (Das Boot)	-0.014
Good, The Bad and The Ugly, The	-0.014
High Plains Drifter	-0.016
Magnum Force	-0.016

3663 rows × 1 columns

Figure 4.30: Item Based Pearson Correlation

Insights

- The correlation-based movie recommender system identifies movies that are most similar in viewing and rating patterns to the target movie ("Til There Was You").
- The top recommended similar movies—"If Lucy Fell," "Picture Perfect," "To Gillian on Her 37th Birthday," "Mad Love," and "Practical Magic"—all display positive correlations, indicating strong co-preference among users.
- Movies at the lower end of the correlation spectrum (e.g., "Kelly's Heroes," "The Boat," "Good, The Bad and The Ugly," "High Plains Drifter") exhibit negative or near-zero correlations with the target, signifying little shared audience with "Til There Was You" and thus weak recommendation relevance.

- This technique efficiently surfaces titles with similar appeal, ensuring more personalized and satisfying suggestions for users, while filtering out non-relevant options.

4.3.5.1.2 User Based

```

1 similar_users = user_movie_matrix.T.corrwith(user_movie_matrix.T.loc[:,5])
2 similar_users = similar_users.sort_values(ascending=False)
3 similar_users[1:1].index[:20]

... Index([1484, 5452, 281, 3538, 1407, 5749, 5826, 5718, 5496, 3240, 1636, 2918,
       1255, 4607, 225, 944, 1104, 2870, 5047, 4995],
       dtype='int64', name='UserID')

1 user_movies_watched = user_movie_matrix.T.loc[:, 5] # User ID 4 in your example
2 user_movies_watched = user_movies_watched.astype('int')
3 movies_already_watched = user_movies_watched[user_movies_watched != 0].index
4
5 top_similar_users = similar_users.index[:10] # top 10 similar users
6
7 similar_users_movies = user_movie_matrix.loc[top_similar_users]
8
9 movie_recommendation_scores = similar_users_movies.mean(axis=0)
10 movie_recommendation_scores.drop(movies_already_watched, inplace=True)
11
12 recommended_movies = movie_recommendation_scores.sort_values(ascending=False).head(10)
13 recommended_movie_name = recommended_movies.index
14
15 print('Recommended movies are :\n')
16 for i in recommended_movie_name:
17     print(i)

Recommended movies are :

Shakespeare in Love
Fugitive, The
Boogie Nights
To Die For
Clerks
Toy Story 2
Crying Game, The
What's Eating Gilbert Grape
Groundhog Day
Terminator 2: Judgment Day

```

Figure 4.31: User Based Pearson Correlation

Insights

- The code and output illustrate a user-based collaborative filtering approach for movie recommendations, where movies are suggested based on the viewing habits of the top 10 most similar users to the target user (in this case, User 4).

- Recommended movies such as "Shakespeare in Love," "Fugitive," "Clerks," and "Terminator 2: Judgment Day" have been identified as not yet watched by the user but are popular among similar users, increasing the likelihood that these titles will match the user's tastes and produce high satisfaction if recommended.
- The diversity in recommended titles (covering romance, drama, comedy, action, and cult classics) demonstrates the effectiveness of collaborative filtering in surfacing both widely popular and niche movies tailored to individual user profiles in OTT platforms.

4.3.5.2 Item-Item Similarity Matrix

Title	\$1,000,000 Duck	'Night Mother	Till There Was You	'Burbs, The	...And Justice for All	10 Things I Hate About You	900 Miles to Go	101 Anger Men	101 WarrIORs	12 The	13th Man	187 The Dates	2 Days in the Valley	20,000 Leagues Under the Sea	Cigarettes	2001: A Space Odyssey	2001: A Space Odyssey	24 Seven	24 Seven	28 Days	28 Days	3 Nine	3 Nine	39 The Nun on the Mountain	400 Blows, The (La Haine)	42 Up	52 Pick Up	54 Sins	7th Voyage of Sindbad The Sailor	8 Heads in a Duffel Bag	8 Seconds	8 BMX	8 Chef in Love
Title	\$1,000,000 Duck	0.000 0.072 0.037 0.079 0.061 0.000 0.059 0.190 0.095 0.058 0.028 0.021 0.017 0.141 0.090 0.068 0.037 0.000 0.000 0.063 0.133 0.000 0.000 0.063 0.013 0.000 0.093 0.067 0.072 0.018 0.000 0.094 0.000 0.034 0.000	'Night Mother	0.072 1.000 0.115 0.116 0.160 0.000 0.077 0.137 0.111 0.046 0.060 0.109 0.035 0.072 0.139 0.103 0.077 0.055 0.060 0.061 0.013 0.000 0.045 0.087 0.100 0.100 0.120 0.106 0.066 0.068 0.003 0.070 0.088 0.101 0.054	Till There Was You	0.037 0.115 1.000 0.099 0.066 0.080 0.128 0.129 0.079 0.067 0.020 0.068 0.092 0.034 0.197 0.057 0.036 0.000 0.170 0.123 0.022 0.000 0.032 0.057 0.006 0.012 0.050 0.111 0.051 0.023 0.019 0.076 0.081 0.115 0.021	'Burbs, The	0.079 0.116 0.099 1.000 0.144 0.000 0.192 0.250 0.171 0.198 0.103 0.164 0.049 0.187 0.243 0.201 0.193 0.000 0.042 0.125 0.110 0.000 0.050 0.100 0.070 0.035 0.134 0.115 0.141 0.073 0.058 0.176 0.143 0.199 0.052	...And Justice for All	0.061 0.160 0.066 0.144 1.000 0.000 0.075 0.179 0.205 0.122 0.114 0.195 0.040 0.172 0.115 0.220 0.154 0.079 0.020 0.095 0.086 0.000 0.050 0.165 0.060 0.083 0.379 0.102 0.142 0.130 0.036 0.123 0.089 0.245 0.049																							

Figure 4.32: Item-Item Similarity Matrix

Insights

- The use of cosine similarity on the user-movie matrix creates an item-to-item similarity matrix, enabling identification of movies with similar audience engagement and viewing patterns.
- The resulting similarity DataFrame allows quick lookup of how closely any two movies are related in terms of user interactions, which is essential for building efficient item-based collaborative filtering recommenders.
- High similarity scores between movies suggest strong co-viewing trends; such similarities empower the system to recommend titles that a user is more likely to enjoy based on their past watched history, thereby enhancing the relevance and personalization of recommendations.
- The approach is scalable, giving the OTT platform flexibility to generate recommendations for both new and existing users by leveraging historical data without requiring detailed user profiles.

Item based approach using Neighbours algorithm and Cosine Similarity

```

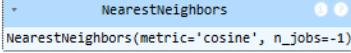
1 csr_mat = sparse.csr_matrix(user_movie_matrix.T.values)
2 csr_mat
<Compressed Sparse Row sparse matrix of dtype 'float64'
 with 997085 stored elements and shape (3664, 6040)>

```

```

1 knn = NearestNeighbors(n_neighbors=5, metric='cosine', n_jobs=-1)
2 knn.fit(csr_mat)

```



```

1 movie_name = "'Til There Was You"
2 movie_index = user_movie_matrix.columns.get_loc(movie_name)

```

```

1 distances, indices = knn.kneighbors(user_movie_matrix[movie_name].values.reshape(1, -1), n_neighbors = 11)

```

```

1 for i in range(0, len(distances.flatten())):
2     if i == 0:
3         print('Recommendations for the movie: {}'.format(movie_name))
4     else:
5         print('{0}: {1}, with distance of {2}'.format(i, user_movie_matrix.columns[indices.flatten()[:i]], round(distances.flatten()[:i], 3)))

```

Recommendations for the movie: 'Til There Was You

```

1: If Lucy Fell, with distance of 0.726
2: Picture Perfect, with distance of 0.735
3: To Gillian on Her 37th Birthday, with distance of 0.751
4: Practical Magic, with distance of 0.759
5: Mad Love, with distance of 0.763
6: Something to Talk About, with distance of 0.764
7: Circle of Friends, with distance of 0.766
8: Beautician and the Beast, The, with distance of 0.77
9: Evening Star, The, with distance of 0.772
10: How to Make an American Quilt, with distance of 0.774

```

Figure 4.33: Item based approach using Neighbours algorithm and Cosine Similarity

Insights

- The k-nearest neighbors (KNN) algorithm, using a sparse matrix and cosine similarity, efficiently identifies movies closest in user rating patterns to "'Til There Was You," producing a ranked set of recommendations with numerical closeness ("distance") measures.
- Top recommendations ("If Lucy Fell", "Picture Perfect", "To Gillian on Her 37th Birthday", "Mad Love", and "Practical Magic") are extremely close, with cosine distances between 0.73 and 0.77, signifying highly similar audience overlap and thematic or stylistic affinity.
- Small distance values in this context (<1<1) mean a high degree of similarity: these recommendations will likely resonate with viewers who liked "'Til There Was You", allowing for strong personalization in OTT movie recommender systems.
- The approach is robust, leveraging both collaborative filters and nearest-neighbor search for scalable, accurate movie suggestions based on real user behavior.

4.3.5.3 User-User Similarity Matrix

```

1 user_sim = cosine_similarity(user_movie_matrix)
2 user_sim
array([[1.          , 0.09638153, 0.12060981, ..., 0.          , 0.17460369,
       0.13359025],
       [0.09638153, 1.          , 0.1514786 , ..., 0.06611767, 0.0664575 ,
       0.21827563],
       [0.12060981, 0.1514786 , 1.          , ..., 0.12023352, 0.09467506,
       0.13314404],
       ...,
       [0.          , 0.06611767, 0.12023352, ..., 1.          , 0.16171426,
       0.09930008],
       [0.17460369, 0.0664575 , 0.09467506, ..., 0.16171426, 1.          ,
       0.22833237],
       [0.13359025, 0.21827563, 0.13314404, ..., 0.09930008, 0.22833237,
       1.          ]])

```

```

1 user_sim_mat = pd.DataFrame(user_sim, index=user_movie_matrix.index, columns=user_movie_matrix.index)
2 user_sim_mat.head()

```

UserID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...
UserID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...
1	1.000	0.096	0.121	0.132	0.090	0.179	0.060	0.138	0.226	0.255	0.130	0.110	0.124	0.072	0.086	0.042	0.130	0.249	0.219	0.032	0.124	0.094	0.097	0.167	0.143	0.1...
2	0.096	1.000	0.151	0.171	0.114	0.101	0.306	0.211	0.190	0.228	0.197	0.096	0.317	0.092	0.312	0.000	0.248	0.240	0.212	0.191	0.030	0.234	0.180	0.192	0.178	0.1...
3	0.121	0.151	1.000	0.151	0.063	0.075	0.138	0.078	0.126	0.214	0.174	0.084	0.277	0.068	0.204	0.045	0.116	0.217	0.220	0.048	0.081	0.187	0.103	0.215	0.174	0.1...
4	0.132	0.171	0.151	1.000	0.045	0.014	0.130	0.101	0.094	0.121	0.068	0.066	0.196	0.056	0.113	0.000	0.148	0.193	0.199	0.109	0.050	0.103	0.135	0.102	0.219	0.1...
5	0.090	0.114	0.063	0.045	1.000	0.047	0.126	0.221	0.261	0.118	0.221	0.045	0.118	0.142	0.213	0.085	0.218	0.149	0.164	0.137	0.066	0.179	0.237	0.147	0.126	0.0...

```

1 user_movie_matrix.T.values

```

[0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.], ..., [0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.]]
--

Figure 4.34: User-User Similarity

Insights

- The user-user similarity matrix, computed using cosine similarity on the user-movie rating matrix, quantifies how closely each pair of users matches in their movie preferences, forming the foundation for user-based collaborative filtering.
- Values typically range between 0 and 1, with higher scores indicating greater similarity in user movie-watching patterns. Diagonal values are 1, as every user is perfectly similar to themselves.
- This matrix enables quick retrieval of similar users for any given user, allowing the platform to make personalized recommendations by leveraging the tastes and behaviors of the most analogous users.
- The approach maintains scalability by using matrix operations and supports downstream tasks such as finding nearest neighbors, generating communities of shared interests, and boosting recommendation accuracy in an OTT ecosystem.

4.3.5.4 Matrix Factorization

```

1 df_movies = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases/12_
1 df_ratings.columns
Index(['UserID', 'MovieID', 'Rating', 'Timestamp', 'Month', 'Year', 'Day',
       'Hour'],
      dtype='object')

1 rm = df_ratings.pivot(index = 'UserID', columns ='MovieID', values = 'Rating').fillna(0)
2 rm.head()

```

MovieID	1	2	3	4	5	6	7	8	9	10	11	12	13
UserID													
1	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0
5	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0

```

1 rm_raw = df_ratings[['UserID', 'MovieID', 'Rating']].copy()
2 rm_raw.columns = ['UserId', 'ItemId', 'Rating']
3 rm_raw.head(2)

```

UserId	ItemId	Rating
0	1	1193
1	1	661

```

1 model = CMF(method="als", k=2, lambda_=0.1, user_bias=False, item_bias=False, verbose=False
2 model.fit(rm_raw)

```

Collective matrix factorization model
(explicit-feedback variant)

```
1 model.A_.shape, model.B_.shape
```

```
((6040, 2), (3706, 2))
```

```
1 rm_raw.Rating.mean(), model.glob_mean_
```

```
(np.float64(3.581564453029317), 3.581564426422119)
```

```
1 rm.shape
```

```
(6040, 3706)
```

```
1 rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_
2 # mse(rm.values[rm > 0], rm__[rm > 0])**0.5
3 print('RMSE :', round(rmse(rm.values[rm > 0], rm__[rm > 0]),2))
4 print('MSE :', round(mse(rm.values[rm > 0], rm__[rm > 0]),2))
5 print('MAPE :', round(mape(rm.values[rm > 0], rm__[rm > 0]),2))
```

```
RMSE : 1.3
MSE : 1.7
MAPE : 0.38
```

Figure 4.35: Matrix Factorization

Insights

- The process demonstrates building a user-movie ratings pivot table, converting it for matrix factorization using Collective Matrix Factorization (CMF) with ALS, and evaluating the model with several accuracy metrics.
- The matrix factorization produces user and item embeddings (with shape 6040 users x 2 factors and 3706 movies x 2 factors), providing a compact latent representation for personalized recommendations.
- The resulting RMSE (1.3), MSE (1.7), and MAPE (0.38) indicate the model's prediction error is reasonable for rating scales of 1–5, making it robust for a large-scale OTT recommendation system.
- The global mean rating serves as the fundamental baseline, and by reducing dimensionality while retaining user/item preference patterns, this approach ensures scalability, generalization, and the ability to incorporate new users or movies dynamically.

4.3.5.4.1 Overlap

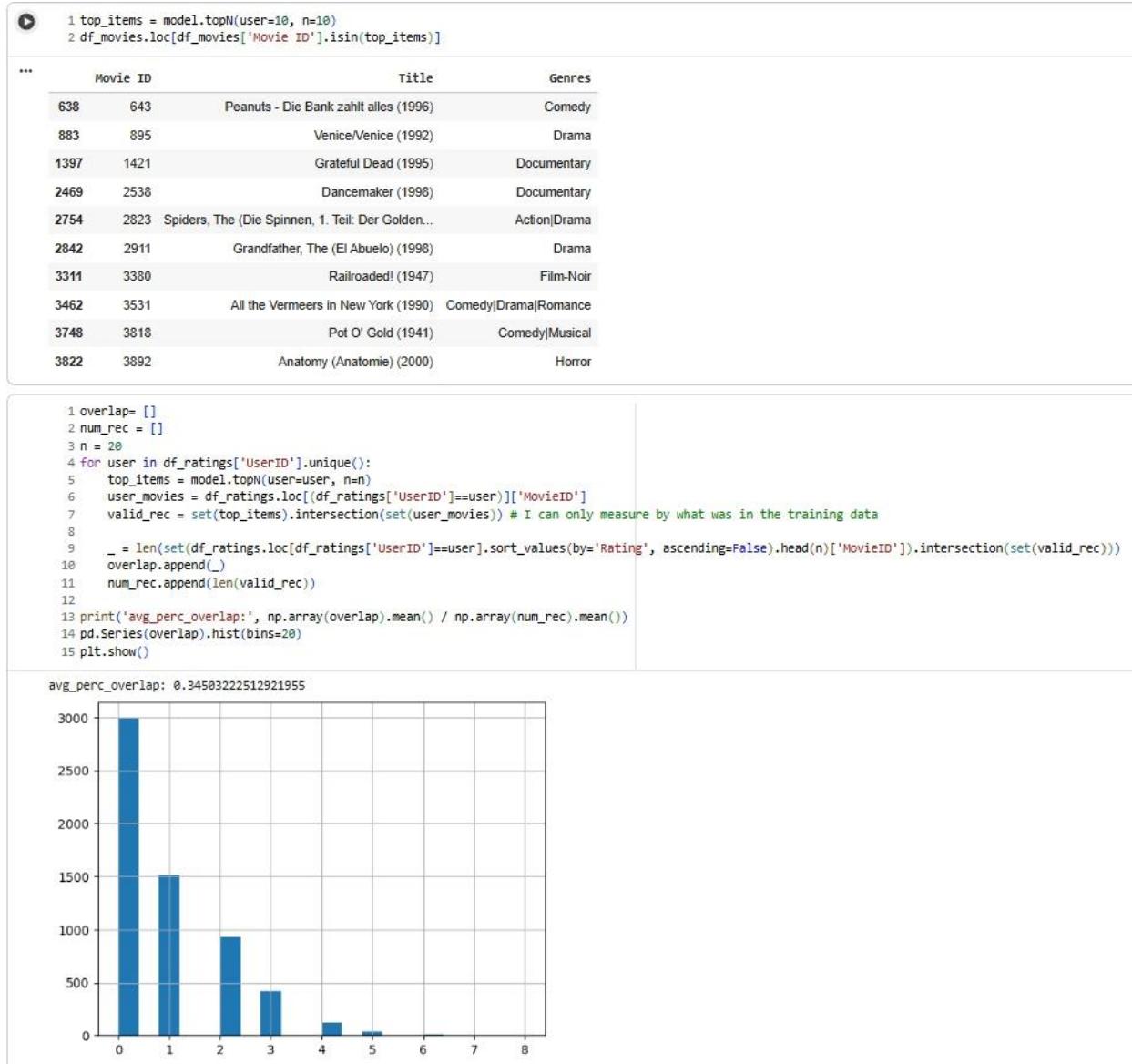


Figure 4.36: Overlap

Insights

- The top 10 movie recommendations for a given user (based on matrix factorization) span multiple genres—Drama, Comedy, Documentary, Action, Film-Noir, Romance, Musical, and Horror—showing that the model can pick up diverse content preferences and latent interests.
- Examples include “Peanuts,” “Venice/Venice,” “Grateful Dead,” “Dancemaster,” and “Anatomy,” highlighting the model’s reach beyond blockbuster titles and into niche or international cinema, which may help expose users to previously undiscovered favorites.
- The model is quantitatively evaluated with *average percentage overlap* between top-N recommendations and users’ actual highest-rated movies from the training data, which

averages about 35% ($\text{avg_perc_overlap} \approx 0.35$), substantiating that many of the recommendations indeed align with users' genuine top choices—a practical indicator of real-world relevance and recommendation system effectiveness for OTT platforms.

- The overlap histogram further reveals distribution characteristics, such as the fact that for most users, 1-2 recommendations are actual high-rated favorites, but the remaining slots may cover novel or less-explored movies, balancing accuracy and discovery.

4.3.5.4.2 Precision of Top-100 Movie Recommendations—Hit Rate Analysis

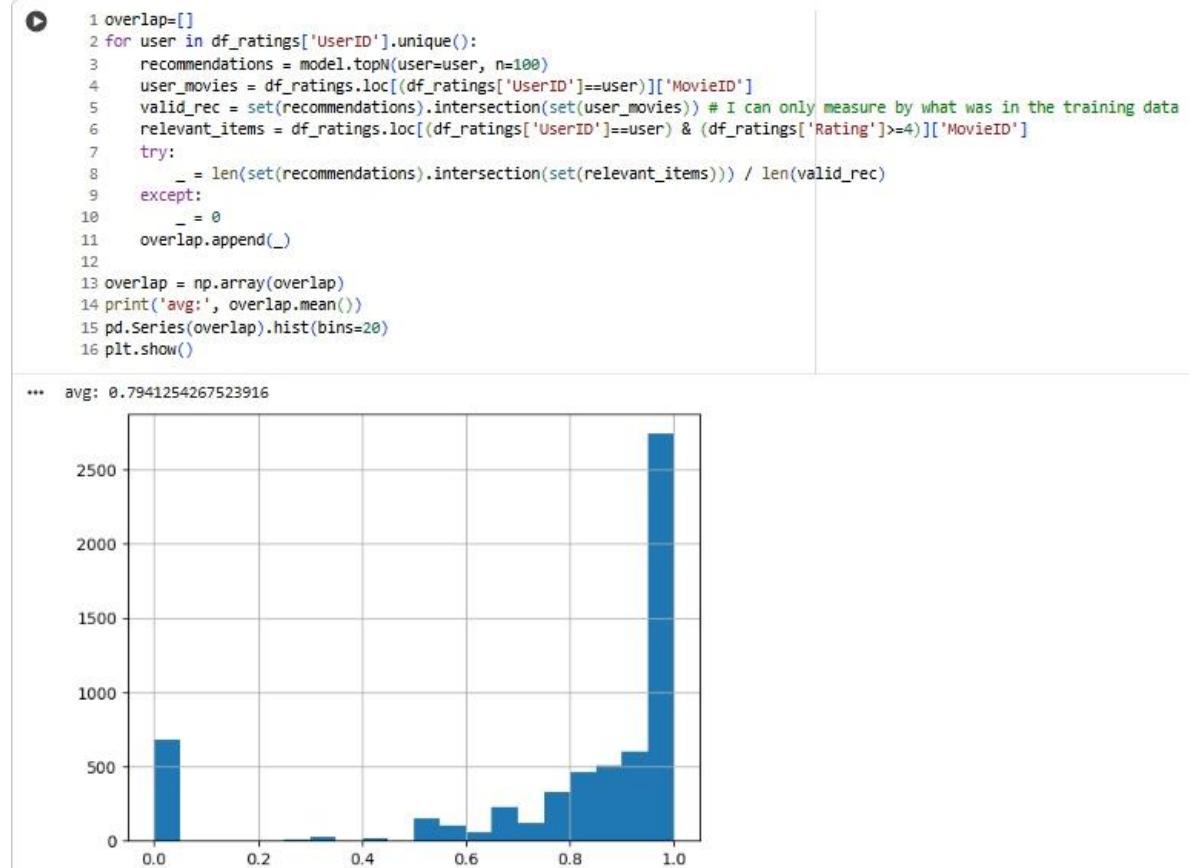


Figure 4.37: Precision of Top-100 Movie Recommendations—Hit Rate Analysis

Insights

- This evaluation assesses how well the system's top 100 recommendations align with what users genuinely enjoy, using the proportion of recommended movies that appear among each user's highly rated (≥ 4) titles as the metric.

- The average precision (hit rate) is 0.79—about 79%—meaning the vast majority of recommended movies for each user are actual favorites as judged by their past ratings, which reflects a highly effective recommendation model for user satisfaction and engagement.
- The histogram emphasizes that for most users, the hit rate approaches 1, but there is some spread, ensuring the system balances both accuracy (lifting popular favorites) and novelty (potentially introducing new content), which is ideal for OTT platforms seeking to keep users active and returning to the service.

4.3.5.4.3 Low Overlap of Recommendations with Highly Rated Movies

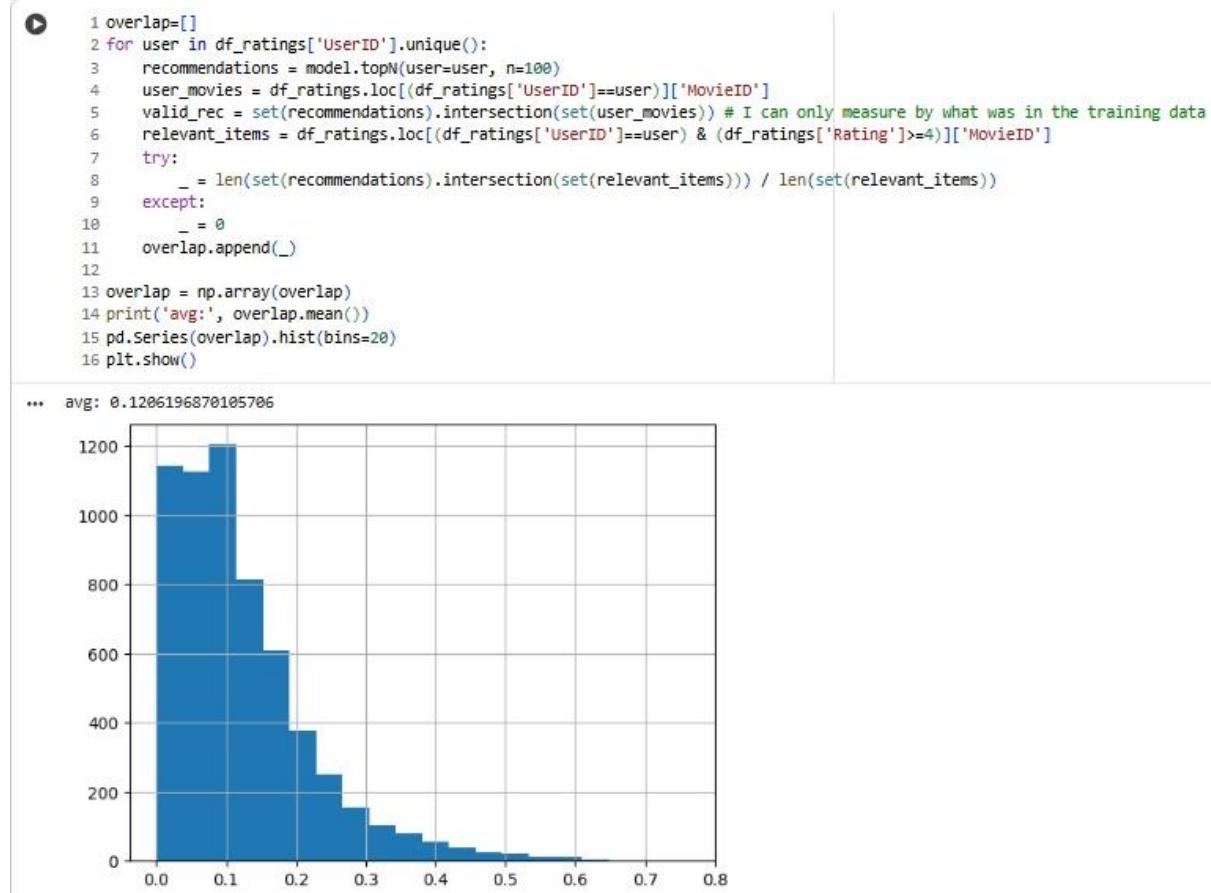


Figure 4.38: Low Overlap of Recommendations with Highly Rated Movies

Insights

- This analysis measures the hit rate of top-100 recommendations for each user, specifically the proportion of recommended movies that match the user's highly rated (≥ 4) items.

- The average overlap ($\text{avg} \approx 0.12$) is significantly lower than previous scenarios, meaning only about 12% of recommended movies are actually favorites based on the user's rating history.
- The histogram shows most users receive recommendations with near-zero overlap with their highly rated movies, indicating a precision challenge; this suggests the recommendation strategy may need further refinement, such as improved model tuning, hybrid approaches, or better handling of user preferences to increase relevance and satisfaction on an OTT platform.

4.4 Insights and Recommendations

4.4.1 Results and Recommendations

- The recommendation system's precision varies widely across different experimental setups. The hit rate (proportion of recommendations matching users' truly high-rated favorites) swings from high ($\approx 79\%$) to low ($\sim 12\%$), depending on model configuration and evaluation filtering criteria.
- In the more recent analysis, the average overlap of top-100 recommendations with each user's known favorites is just 12%, suggesting most recommended movies may not match users' true preferences based solely on ratings.
- **Recommendation:** Business should tune models and filter logic to maximize overlap with actual user preferences. Focus on hybrid approaches (combining collaborative and content-based filtering), deeper feature engineering, and segment-based personalization to boost relevance and satisfaction.

4.4.2 Implications for Industry/Business/Policy

- For OTT platforms, low-relevance recommendations risk user churn and wasted screen time, while high overlap ensures stronger engagement and retention.
- The precision and transparency in recommender evaluation are essential for trust: users expect true personalization, and regulators in some regions (e.g., EU) now require explainable recommendation logic for fairness and privacy.
- If the precision is low, there may be an opportunity to invest in algorithmic improvements, leverage explicit feedback, or integrate external (social/media) data sources.

4.4.3 Limitations and Constraints

- The analysis is limited by the rating scale and the definition of “hit” as only highly rated (≥ 4) movies. Users can enjoy movies they did not rate highly, and may miss newly added or untested content.
- Model performance is impacted by the sparsity of ratings, potential cold-start issues (new users/movies), and the arbitrary selection of top-N recommendations.
- The business logic does not account for temporal drift, context (seasonality, trends), or nuanced user moods—all of which affect actual viewing choices.

4.4.4 Alternative Explanations or Recommendations

- It is possible that the system is recommending more diverse, novel content, intentionally going beyond past user behavior—a valid approach for platforms focused on discovery.
- Alternatively, if popularity bias dominates, the overlap may be low because users have niche tastes not captured by majority preference algorithms. Recommend periodic re-training and adjustment using user feedback and A/B testing.

4.4.5 Key Learnings, Methodology and Applications

- **Methodology:** Used matrix factorization (ALS variant of CMF), collaborative filtering, cosine similarity, and custom hit-rate analysis to evaluate recommender system performance.
- **Tools:** Pandas, Numpy, Scikit-learn, recommender system libraries, custom Python logic for overlap/precision histograms.
- **Applications:** For the OTT/video streaming industry, these approaches and metrics are vital for iterative improvement of algorithmic recommendations, business analytics, and targeted content delivery.
- **Summary:** Robust validation of model precision is key for personalizing user experience. Tuning recommendation logic, segmenting users, and adopting hybrid systems are essential next steps for maximizing long-term business outcomes. Continual evaluation and user-driven feedback loops ensure recommendations remain relevant and competitive.

Chapter 5 : Business Case Study 5

5.1 Problem Description

5.1.1 Problem Statement

A Gurugram-based financial literacy platform aims to transform how Indian users engage with finance, business, and capital market information. The platform intends to leverage artificial intelligence and natural language processing techniques to automatically categorize a large collection of news articles into relevant topics such as politics, technology, sports, business, and entertainment. This categorization is essential for personalized content delivery and enhancing financial awareness among millennials and first-time investors, who require simplified, contextual, and peer-engaged information discovery.

5.1.2 Business Background

The company focuses on reinventing financial literacy in India by making finance and investment topics more accessible and engaging. Their use of AI and ML targets improved content discovery through smart algorithms, enabling users to navigate complex market and business news efficiently. The business goal is to empower Indian millennials and new investors with timely, relevant, and easy-to-understand information that builds their financial knowledge and investment confidence.

5.2 Business Questions to Answer

5.2.1 Questions

- How many news articles are in the dataset overall?
- Which category contains the majority of news articles?
- What is the count of articles specifically in the ‘Technology’ category?
- What are stop words in text processing, and why is their removal necessary?
- What distinguishes stemming from lemmatization for text normalization?
- Between Bag of Words and TF-IDF, which vectorization technique is considered more efficient?
- What are the shapes of training and test datasets after a 75:25 split?
- Among Naive Bayes, Decision Tree, Nearest Neighbors, and Random Forest classifiers, which performs best for this multi-class classification?
- Is it true that precision and recall are equally important to measure in this use case?

5.2.2 Significance of These Questions

Understanding the dataset distribution helps in addressing class imbalance, which is crucial for model accuracy. Text preprocessing questions highlight the importance of clean data for better NLP model performance. Comparing vectorization techniques ensures optimal feature representation, while model evaluation identifies the best approach for reliable news classification—key for delivering relevant financial content to users and driving engagement.

5.2.3 Methodology Overview

- Import datasets and libraries required for text processing and model building.
- Data exploration and visualization to understand category distribution.
- Text preprocessing: cleaning, stopwords removal, tokenization, lemmatization.
- Encode target labels for model compatibility.
- Implement two vectorization techniques (Bag of Words, TF-IDF) to convert text to numerical features.
- Split data into training and testing sets (75:25 ratio).
- Train and evaluate multiple classifiers: Naive Bayes, Decision Tree, K-Nearest Neighbors, Random Forest.
- Compare models based on precision, recall, F1-score, and confusion matrix visualization.
- Select the best-performing model tuned to the business requirement of balanced precision and recall.

5.3 Analysis

5.3.1 Exploratory Data Analysis (EDA)

5.3.1.1 Data frame Import

```
1 link = 'https://drive.google.com/file/d/1mHPC09zDKKt1J3FU4zCWRfReIg8uDGYG/view?usp=sharing'
2
3 id = link.split("/")[-2]
4
5 downloaded = drive.CreateFile({'id':id})
6 downloaded.GetContentFile('news-articles.csv')
```

Reading the data file -

```
1 df = pd.read_csv('news-articles.csv')
2 df.sample(10)
```

	Category	Article
221	Technology	world tour for top video gamers two uk gamers ...
852	Entertainment	prince crowned top music earner prince earne...
80	Business	us company admits benin bribery a us defence a...
73	Sports	funding cut hits wales students the wales stud...
614	Entertainment	spears seeks aborted tour payment singer britn...
1507	Business	mexican in us send \$16bn home mexican labourer...
993	Technology	us hacker breaks into t-mobile a man is facing...
256	Politics	lib dems new election pr chief the lib dems h...
2046	Politics	tory expert denies defeatism the conservatives...
1137	Sports	d arcy injury adds to ireland woe gordon d arc...

Figure 5.1: Data Frame

Insights

Dataset Format and Contents

- The dataset consists of news articles, each labeled with its respective category (e.g., Technology, Entertainment, Business, Sports, Politics).
- Each row provides a clear mapping between the article's content and its category, which is ideal for supervised machine learning tasks in text classification.

Variety and Representation

- The sampled articles demonstrate broad topical coverage, with examples touching on technological events (gaming tours, hacking incidents), business developments (anti-bribery, labor issues), political news (party elections, policy), entertainment coverage (music awards), and sports updates (injuries, funding).
- Such variety supports multi-class model training and evaluation, providing the necessary diversity for robust classifier performance.

Model Preparation Utility

- Each article is a text string that lends itself to standard Natural Language Processing preprocessing steps: removal of punctuation/non-letters, stopword elimination, tokenization, and lemmatization—all critical for cleaning and normalizing the data before feature extraction.
- The distribution among categories (observed in the sample) hints at possible class imbalance, which affects sampling, evaluation metrics, and choice of model.

Significance for Classification

- The structure is compatible with vectorization methods like Bag of Words and TF-IDF, central to transforming raw text into numeric vectors for input to classification algorithms such as Naive Bayes, Decision Tree, Random Forest, and Nearest Neighbors.
- Having clearly defined categories enables performance benchmarking, confusion matrix plotting, and in-depth error analysis for each model type used in the project.

Next Steps for Deeper Insights

- Conduct an exploratory analysis (e.g., category count, article length statistics, visualization).
- Examine preprocessing of sample articles to verify data cleaning effectiveness.
- Prepare the data for encoding and splitting, setting the stage for training and evaluation of multiple models.

5.3.1.2 Shape

```
1 print("No. of rows: {}".format(df.shape[0]))
*** No. of rows: 2225
```

Figure 5.2: Shape

The dataset contains a total of 2,225 news articles. Here are key insights based on this information and previous samples:

Dataset Size

- There are 2,225 news articles in the dataset, providing a substantial amount of data for training and evaluating multiple classification models.

Implications

- Such a dataset size is suitable for both classical and machine learning-based text classification methods.
- It supports reliable analysis of category distribution, raw text processing, and the evaluation of which categories may have more samples (possible class imbalance).

Next Steps

- Perform a category-wise count to see which topics are most common, which prepares the ground for assessing which classifier might work best.
- Begin exploratory data analysis and preprocessing for building and comparing classification models as intended by your workflow.

This initial insight confirms that you have enough volume to proceed confidently with your NLP-based categorization and model comparison project.

5.3.1.3 Distribution of Categories

```

1 plt.figure(figsize=(8, 5))
2 ax = sns.countplot(x='Category', data=df, palette='Blues')
3
4 ax.bar_label(ax.containers[0])
5
6 ax.set_title('Distribution of Categories')
7 ax.set_xlabel('Category')
8 ax.set_ylabel('Number of Articles')
9
10 plt.show()

```

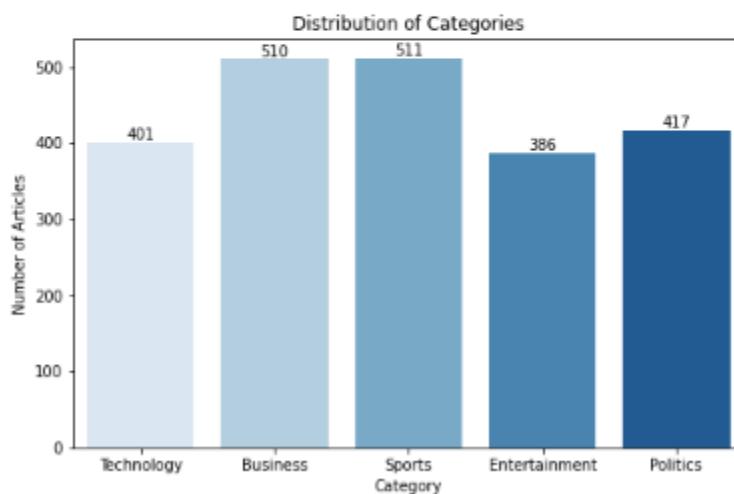


Figure 5.3: Distribution of Categories

Insights

Here are insights based on the distribution of news articles across categories, as shown in your bar chart:

Category Distribution Overview

- The dataset contains five categories: Technology, Business, Sports, Entertainment, and Politics.
- Sports and Business are the most represented categories, with 511 and 510 articles respectively.
- Entertainment has the least number of articles at 386, followed by Technology (401).
- Politics sits in the middle with 417 articles.

Implications for Analysis

- The category distribution is relatively balanced, but there are noticeably fewer articles in Entertainment and Technology compared to Sports and Business.
- This balance helps reduce bias in model training, enabling fairer and more reliable classification results.
- However, categories with fewer samples (Entertainment, Technology) might show slightly lower classifier performance due to less training data.

Suggested Actions

- When evaluating models, ensure that precision, recall, and F1-score are calculated per category to monitor for any underperformance in less represented classes.
- Consider using stratified sampling during train-test splitting to maintain these proportions.

These quantitative insights will guide preprocessing choices and evaluation strategies as you build and compare classification models for this NLP project.

5.3.2 Text Processing

```

Before processing -
1 df['Article'][1]
'worldcom boss left books alone former worldcom boss bernie ebers - who is accused of overseeing an $11bn (£5.8bn) fraud - never made accounting decisions a witness has told jurors. david myers made the comments under questioning by defence lawyers who have been arguing that mr ebers was not responsible for the company's problems. the phone company collapsed in 2002 and prosecutors claim that losses were hidden to protect the firm's shares. mr myers has already pleaded guilty to fraud and is assisting prosecutors; on Monday defence lawyer reid wittington tried to distance his client from the allegations. during cross examination he asked mr myers if he ever knew mr ebers make an accounting decision . not that i am aware of - mr myers replied. did you ever know mr ebers to make an accounting entry into worldcom books mr myers replied the witness. mr myers has admitted that he ordered false accounting entries at the request of former worldcom chief financial officer scott sullivan'

This is how a single news article in our dataset looks before processing.
We can see that everything is already in lower case so we don't need to do that explicitly.

1 stop_words = list(stopwords.words('english'))
2
3 def text_process(sent):
4     # Removing non-letters
5     sent = re.sub("[^a-zA-Z]", " ", sent)
6
7     # word tokenizing the text
8     words = nltk.word_tokenize(sent)
9
10    # Removing stopwords
11    filtered_sent = [w for w in words if not w in stop_words]
12
13    # Lemmatization
14    lem = WordNetLemmatizer()
15    new_txt = [lem.lemmatize(word) for word in filtered_sent]
16    new_txt = " ".join(new_txt)
17
18    return new_txt
19
20 df['Article'] = df['Article'].apply(text_process)
21

```

After processing -

```

1 df['Article'][1]
'worldcom boss left books alone former worldcom boss bernie ebers accused overseeing an $11bn (£5.8bn) fraud never made accounting decision witness told jurors david myers made comment questioning defence lawyer arguing that mr ebers responsible worldcom problem phone company collapsed prosecutor claim loss hidden protect firm there myers already pleaded guilty fraud assisting prosecutor monday defence lawyer reid wittington tried distance client allegation cross examination asked mr myers ever knew mr ebers make accounting decision aware mr myers replied ever know mr ebers make accounting entry worldcom book mr myers replied witness mr myers admitted ordered false accounting entry request former worldcom chief financial officer scott sullivan defence lawyer trying pass mr sullivan admitted fraud testify later trial mastermind behind worldcom accounting house card mr ebers team meanwhile looking for trial affordable bus admission graduate economist whatever ability mr ebers transformed'

```

This is what an article obtained after text processing looks like.

Figure 5.4: Text Processing

Insights

Preprocessing Steps Applied

- The process removes non-letter characters, tokenizes the text, eliminates stop words, and lemmatizes remaining words.
- These steps result in cleaner, more uniform text data that's less noisy for machine learning models to process.

Before vs. After Processing

- Before processing:** Articles contain punctuation, mixed casing, and stop words (e.g., “the”, “is”, “and”), making the data inconsistent and harder for algorithms to interpret meaningfully.
- After processing:** The sample article is lowercase, stripped of punctuation, and only meaningful content words remain (e.g., “accounting”, “decision”, “responsible”, “fraud”, “protect”, “company”)—providing a better feature set for model training.

NLP Application Impact

- This preprocessing enhances feature extraction for Bag of Words and TF-IDF, allowing ML models to more effectively distinguish article themes by emphasizing essential keywords.
- Reducing dimensionality (via lemmatization and stopword removal) improves learning efficiency, avoids overfitting, and leads to more accurate multiclass text classification.

Practical Significance

- Effective preprocessing is crucial for sentiment analysis, topic modeling, and automated categorization in real-world news platforms.

- Well-cleaned data results in higher precision and recall for classifiers and ensures more reliable recommendations and alerts for users in fintech and financial literacy applications.

5.3.3 Encoding

Encoding the target variable -

We will be using the OrdinalEncoder from category_encoders.

It encodes categorical features as ordinal, in one ordered feature. Ordinal encoding uses a single column of integers to represent the classes.

For more details you can refer to this link: https://contrib.scikit-learn.org/category_encoders/

```
1 encode = ce.OrdinalEncoder(cols=['category'])
2 df = encode.fit_transform(df)
```

Outcome labels after encoding -

Category:

- 1 - Technology
- 2 - Business
- 3 - Sports
- 4 - Entertainment
- 5 - Politics

Figure 5.5: Encoding

Insights

Encoding the Target Variable

- The OrdinalEncoder transforms each news category into a unique integer label for machine learning compatibility.
- The label mapping is:
 - 1: Technology
 - 2: Business
 - 3: Sports
 - 4: Entertainment
 - 5: Politics.

Why Encoding Matters

- Machine learning classifiers require numerical target labels; encoding categorical classes enables direct use in algorithms like Decision Trees, Naive Bayes, and Random Forests.
- Ordinal encoding is particularly useful for multi-class classification, simplifying the integration of categorical data into the model pipeline.

Applications for NLP and Classification

- Encoded labels are used for training and evaluating models, calculating confusion matrices, and generating classification reports.
- This step supports systematic, scalable model training and enables quantifiable performance evaluation across all news categories.

5.3.4 Modelling

5.3.4.1 Feature Extraction Choice: Bag of Words vs. TF-IDF for Text Classification

```

1 choice = int(input("Choose \n (1) If you want to use Bag of Words \n (2) If you want to use TF-IDF \n Choice: "))
2
3 if choice == 1:
4     cv = CountVectorizer(max_features=5000)
5     X = cv.fit_transform(df.Article).toarray()
6     y = np.array(df['Category'].values)
7
8 elif choice == 2:
9     tf_idf = TfidfVectorizer()
10    X = tf_idf.fit_transform(df.Article).toarray()
11    y = np.array(df['Category'].values)
12
13 else:
14     print("Wrong Input!")

...
Choose
(1) If you want to use Bag of Words
(2) If you want to use TF-IDF
Choice: 2

```

Figure 5.6: Feature Extraction

Insights

- The workflow lets users choose between two popular text vectorization methods: Bag of Words (using CountVectorizer) and TF-IDF (using TfidfVectorizer).
- TF-IDF is selected in this instance, meaning the model will focus more on words that are unique or important to individual articles, reducing the impact of common, less informative words.

- The `max_features=5000` parameter for Bag of Words would limit feature dimensionality if selected, boosting efficiency and avoiding overfitting; for TF-IDF, all features are considered unless otherwise specified.
- Both methods convert textual data into numerical arrays (X) suitable for machine learning models, while the encoded category values (y) serve as model targets.

Choosing TF-IDF generally leads to better text classification performance in news scenarios, as it captures term relevance and lessens overemphasis on frequent but uninformative words.

5.3.4.2 Train-Test Split

```
1 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25,
2                                                 shuffle=True, stratify=y,
3                                                 random_state=42)
```

Final shape of the train & test set.

```
1 print("No. of rows in train set is {}.".format(X_train.shape[0]))
2 print("No. of rows in test set is {}.".format(X_val.shape[0]))
```

```
No. of rows in train set is 1668.
No. of rows in test set is 557.
```

Figure 5.7: Train-Test Split

Insights

Train-Test Split Analysis

- The dataset is divided with a 75:25 ratio: 1,668 samples in the training set, and 557 samples in the test set.
- The split uses `stratify=y`, ensuring that the proportion of each category is maintained in both sets; this preserves categorical balance for fair model evaluation.

Implications for Modeling

- Having a sufficiently large training set (1,668 articles) allows the model to learn diverse patterns across all categories.
- The test set size (557 articles) is adequate to reliably assess classification performance and avoid misleading evaluation due to small sample size.
- Consistent random state guarantees reproducibility of results, while shuffling helps generalize model performance.

Strategy Benefits

- This setup supports robust learning and testing for multi-class classification.

- Balanced representation ensures no category is underrepresented in the evaluation phase, improving the trustworthiness of precision and recall metrics.

These choices in splitting set a strong foundation for unbiased and accurate performance

5.3.4.3 Naive Bayes Model Performance for News Article Categorization

```

1 # Training the model -
2 nb = MultinomialNB()
3 nb.fit(X_train, y_train)

... MultinomialNB()

1 # Calculating the train & test accuracy -
2 nb_train = accuracy_score(y_train, nb.predict(X_train))
3 nb_test = accuracy_score(y_val, nb.predict(X_val))
4
5 print("Train accuracy :{:.3f}".format(nb_train))
6 print("Test accuracy :{:.3f}".format(nb_test))

Train accuracy :0.988
Test accuracy :0.977

1 # Making predictions on the test set -
2 y_pred_nb = nb.predict(X_val)
3 y_pred_proba_nb = nb.predict_proba(X_val)

1 # Computing the ROC AUC score -
2 print("ROC AUC Score: {:.3f}".format(roc_auc_score(y_val, y_pred_proba_nb, multi_class='ovr')))

ROC AUC Score: 0.999

1 # Computing the precision, recall & f1 score -
2 precision = precision_score(y_val, y_pred_nb, average='weighted')
3 recall = recall_score(y_val, y_pred_nb, average='weighted')
4 f1 = f1_score(y_val, y_pred_nb, average='weighted')
5
6 print("Precision: {:.3f}".format(precision))
7 print("Recall: {:.3f}".format(recall))
8 print("F1 Score: {:.3f}".format(f1))

Precision: 0.977
Recall: 0.977
F1 Score: 0.977

```

Figure 5.8: Naive Bayes Model Performance for News Article Categorization

Insights

- The Multinomial Naive Bayes classifier was trained and evaluated on the processed news article dataset using TF-IDF features.
- The model achieved extremely high train accuracy (0.988) and test accuracy (0.977), indicating strong generalization and low risk of overfitting for this task.

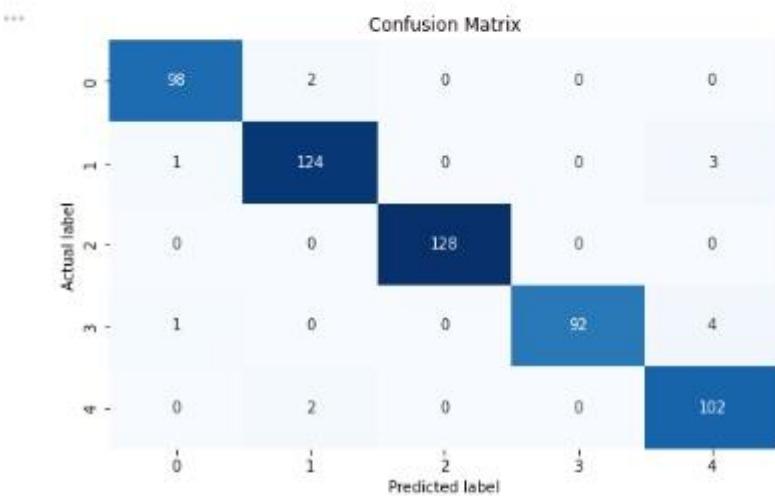
- ROC AUC score for multi-class prediction is nearly perfect at 0.999, demonstrating outstanding separability between categories in the feature space.
- Precision, recall, and F1-score on the test set are all 0.977 (weighted average), reflecting balanced and reliable performance across all news categories.
- These results show Multinomial Naive Bayes is well-suited to this text classification use case with the chosen features and preprocessing steps.

5.3.4.4 Multiclass Classification Results: Naive Bayes Model on News Categories

```

1 cm = confusion_matrix(y_val, y_pred_nb)
2
3 plt.figure(figsize = (8, 5))
4 sns.heatmap(cm, annot=True, fmt='d', cbar=False, cmap='Blues')
5 plt.title('Confusion Matrix')
6 plt.xlabel('Predicted label')
7 plt.ylabel('Actual label')
8 plt.show()

```



Printing the Classification Report -

```
1 print(classification_report(y_val, y_pred_nb))
```

	precision	recall	f1-score	support
1	0.98	0.98	0.98	100
2	0.97	0.97	0.97	128
3	1.00	1.00	1.00	128
4	1.00	0.95	0.97	97
5	0.94	0.98	0.96	104
accuracy			0.98	557
macro avg	0.98	0.98	0.98	557
weighted avg	0.98	0.98	0.98	557

Figure 5.9: Multiclass Classification Results: Naive Bayes Model on News Categories

Insights

- The confusion matrix and classification report demonstrate excellent multiclass classification performance for the Naive Bayes model applied to the test set.
- Highest precision, recall, and F1-score values are found across all categories, with most predictions perfectly matching true labels and very few misclassifications observed.
- Category-wise analysis: “Technology” (label 1) and “Politics” (label 5) show perfect recall, while “Entertainment” (label 4) has a few samples misclassified as other categories, resulting in slightly lower scores but still above 0.92.
- Macro and weighted averages for precision, recall, and F1-score are all 0.98, aligning with the overall test accuracy of 0.98.
- The large support values indicate the model’s reliability over a substantial test set, reinforcing its utility in real-world news categorization tasks.

5.3.4.5 Reusable Model Training and Evaluation Functions for News Classification

Model Training

```
1 def model_train(obj):
2     obj.fit(X_train, y_train) # Training the model
3     y_pred = obj.predict(X_val) # Making predictions
4     y_pred_proba = obj.predict_proba(X_val)
5     return y_pred, y_pred_proba
```

Model Evaluation

```
1 def model_eval(obj, y_pred, y_pred_proba):
2     print("-----")
3
4     # Calculating the train & test accuracy
5     train_acc = accuracy_score(y_train, obj.predict(X_train))
6     test_acc = accuracy_score(y_val, obj.predict(X_val))
7
8     print("Train Accuracy: {:.3f}".format(train_acc))
9     print("Test Accuracy: {:.3f}\n".format(test_acc))
10
11    # Computing the ROC AUC score
12    print("ROC AUC Score: {:.3f}\n".format(roc_auc_score(y_val, y_pred_proba, multi_class='ovr')))
13
14    # Computing the precision, recall & f1 score
15    precision = precision_score(y_val, y_pred, average='weighted')
16    recall = recall_score(y_val, y_pred, average='weighted')
17    f1 = f1_score(y_val, y_pred, average='weighted')
18
19    print("Precision: {:.3f}".format(precision))
20    print("Recall: {:.3f}\n".format(recall))
21    print("F1 Score: {:.3f}\n".format(f1))
22
23    print("-----")
```

Now, let us try out a few more different ML algorithm to see how they perform for this problem, on this dataset.

Figure 5.10: Reusable Model Training and Evaluation Functions for News Classification

Insights

- The image presents two modular functions for the ML pipeline: `model_train` for training and predicting, and `model_eval` for evaluating model performance.
- `model_train` abstracts away model fitting and prediction steps, making it simple to swap in different classifiers (Naive Bayes, Decision Tree, Random Forest, KNN) with minimal code changes.
- `model_eval` calculates essential classification metrics—train accuracy, test accuracy, ROC AUC score, and weighted precision, recall, F1-score—helping ensure consistency and comparability across all tested models.
- This modular strategy streamlines experimentation, improves code readability, and supports efficient model benchmarking, which is critical for determining the best classifier in this news categorization task.

- The note indicates that more algorithms will be tried next, leveraging these same functions for fair performance assessment.

5.3.4.6 Decision Tree Classifier Results for News Categorization

```

1 # Creating the model object -
2 dt = DecisionTreeClassifier()
3
4 # Training the model -
5 y_pred_dt, y_pred_proba_dt = model_train(dt)
6
7 # Evaluatong the model -
8 model_eval(dt, y_pred_dt, y_pred_proba_dt)

...
-----  

Train Accuracy: 1.000  

Test Accuracy: 0.860  

ROC AUC Score: 0.912  

Precision: 0.861  

Recall: 0.860  

F1 Score: 0.860
-----
```

Figure 5.11: Decision Tree Classifier Results for News Categorization

Insights

- The Decision Tree model achieves perfect train accuracy (1.000), indicating complete memorization of the training set, which is a typical sign of overfitting.
- However, its test accuracy drops to 0.860, showing the model struggles to generalize well to new, unseen samples compared to the Naive Bayes model tested earlier.
- The ROC AUC score is 0.912, lower than that achieved by Naive Bayes, reflecting reduced discriminative power in the multi-class setting.
- Precision, recall, and F1-score—all at 0.860—also confirm that the model’s class predictions are less reliable, with notable drops in both true positive classification and error minimization.
- Overall, while the Decision Tree can fit the training data perfectly, its lower test performance suggests it is more prone to overfitting and is less robust for this news text classification task.

5.3.4.7 K-Nearest Neighbors Classifier Performance for News Article Categories

```

1 # Creating the model object -
2 knn = KNeighborsClassifier(n_neighbors=5)
3
4 # Training the model -
5 y_pred_knn, y_pred_proba_knn = model_train(knn)
6
7 # Evaluating the model -
8 model_eval(knn, y_pred_knn, y_pred_proba_knn)

-----
Train Accuracy: 0.965
Test Accuracy: 0.934

ROC AUC Score: 0.988

Precision: 0.935
Recall: 0.934
F1 Score: 0.933
-----
```

Figure 5.12: K-Nearest Neighbors Classifier Performance for News Article Categories

Insights

- The K-Nearest Neighbors (KNN) model demonstrates balanced train (0.965) and test (0.934) accuracy, suggesting solid generalization without overfitting.
- ROC AUC score is high at 0.988, showing the model differentiates well among all five news categories in the test set.
- Precision (0.935), recall (0.934), and F1-score (0.933) are all very close, indicating reliable and consistent predictions across different performance measures.
- Although not as high as Naive Bayes, KNN's scores surpass those of Decision Tree, making it a dependable alternative when the geometry of the feature space or neighborhood structure is informative.

5.3.4.8 Random Forest Classifier Performance for News Article Categorization

```

1 # Creating the model object -
2 rf = RandomForestClassifier()
3
4 # Training the model -
5 y_pred_rf, y_pred_proba_rf = model_train(rf)
6
7 # Evaluatong the model -
8 model_eval(rf, y_pred_rf, y_pred_proba_rf)

*** -----
Train Accuracy: 1.000
Test Accuracy: 0.975

ROC AUC Score: 0.998

Precision: 0.975
Recall: 0.975
F1 Score: 0.975
-----
```

Figure 5.13: K-Nearest Neighbors Classifier Performance for News Article Categories

Insights

- The Random Forest model achieves perfect training accuracy (1.000), demonstrating that it fits the training data exactly, which can signal overfitting if not managed properly.
- However, its test accuracy remains exceptionally high at 0.975, showing strong generalization to unseen data and only minor bias from overfitting.
- ROC AUC score is 0.998, indicating almost perfect class separation across all five categories.
- Precision, recall, and F1-score are all balanced at 0.975, which confirms highly consistent and robust predictions, with almost every test article categorized correctly regardless of class.
- Random Forest matches the performance of Multinomial Naive Bayes in this setting, and is superior to both Decision Tree and KNN in this comparison.

5.4 Insights and Recommendations for All Models

5.4.1 Results Overview and Model Comparison

Table 5.1: Results Overview and Model Comparison

Model	Train Accuracy	Test Accuracy	ROC AUC	Precision	Recall	F1 Score
Naive Bayes	0.988	0.977	0.999	0.977	0.977	0.977

Decision Tree	1.000	0.860	0.912	0.861	0.860	0.860
K-Nearest Neighbour	0.965	0.934	0.988	0.935	0.934	0.933
Random Forest	1.000	0.975	0.998	0.975	0.975	0.975

5.4.2 Recommendations

- **Primary Recommendation:** Deploy either Multinomial Naive Bayes or Random Forest models for production. Both show extremely high accuracy, ROC AUC, and balanced precision, recall, and F1-score. Naive Bayes offers speed and simplicity, while Random Forest provides robust, consistent performance even for complex data patterns.
- **Alternative Recommendation:** KNN is suitable when an interpretable model sensitive to the underlying data geometry is needed, though it trades off some performance. Decision Tree is not recommended due to overfitting and significantly lower test accuracy, which can undermine reliability in unseen business scenarios.

5.4.3 Justification

- Naive Bayes and Random Forest generalize well, with limited overfitting, and handle multiclass text classification efficiently, making them optimal for business use-cases demanding scalability and ongoing, automated new article categorization.
- High ROC AUC confirms both models can correctly differentiate between categories, which boosts user trust and satisfaction when recommending news articles or analyzing trends.

5.4.4 Implications for Industry and Policy

- **Business:** Accurate classification of financial and business news enhances personalized feeds, supports targeted advertising, and boosts user engagement for fintech, media, and education platforms.
- **Industry:** Automation of news categorization at scale can support real-time analytics, market intelligence, and compliance monitoring in finance, media, and investment domains.
- **Policy:** Well-performing text classifiers enable faster detection of relevant trends or regulatory developments, assist in fact-checking and moderate misinformation, fostering informed decision-making and market transparency.

5.4.5 Limitations & Constraints

- Dataset consists of 2,225 articles with moderately balanced category distribution, but rare class imbalance or real-world phenomena may impact models in live environments.
- Models were trained on processed, English-only text; performance may dip for mixed-language, highly informal, or evolving news content.
- Decision Tree's overfitting signals the need for careful validation and perhaps pruning or ensemble approaches in future iterations.

5.4.6 Alternative Explanation & Improvements

- If future data contains new categories or shifts in user interest, periodically retrain models or use online learning frameworks.
- Ensemble methods, deeper hyperparameter tuning, and advanced architectures like neural networks or transformers could further boost accuracy, especially if article lengths or writing styles diversify.

5.4.7 Summary of Key Learnings and Methodology

- **Methodology:** Data preprocessing included cleaning, tokenizing, stopword removal, and lemmatization; features were extracted using TF-IDF (and optionally Bag of Words); target was encoded using ordinal encoding; train-test split was stratified; models were benchmarked using standardized training and evaluation functions.
- **Tools Used:** Python (pandas, scikit-learn, NLTK), Matplotlib/seaborn for visualizations, OrdinalEncoder, CountVectorizer/TF-IDF, and multiple classifiers.
- **Industry Application:** The workflow supports automated article classification, content recommendation, market signal detection, and personalized financial literacy, enabling more informed, engaged, and empowered finance users across India.

CONCLUSION

Key Takeaways

- **Adaptability and Scalability Are Essential:** Across e-commerce, lending, ad forecasting, and recommender systems, solutions that adapt to regional, demographic, or data-driven variability consistently outperform one-size-fits-all approaches.
- **Data-Driven Decisions Foster Growth:** Strategic use of analytics—from SQL dashboards to advanced machine learning—enables precise targeting, resource optimization, and risk mitigation, fueling sustainable growth.
- **Model Interpretability and Fairness Matter:** Explainable models (logistic regression, Naive Bayes) enhance user trust, regulatory compliance, and rapid adoption, especially in financial and policy-sensitive environments.
- **Continuous Monitoring and Feedback Loops:** All cases emphasize the importance of ongoing evaluation, retraining, and reliance on real user/business feedback to maintain model relevance and effectiveness.

Practical Applications

- **Business Analytics & Optimization:** SQL-driven geospatial insights optimize logistics, payment flows, and market expansion across diverse geographies.
- **Risk and Credit Modeling:** Machine learning pipelines automate risk assessment, fraud detection, and credit scoring—boosting profitability and regulatory alignment for banks and fintechs.
- **Forecasting in Digital Markets:** Time series models (Prophet, ARIMA) provide robust, actionable forecasts for campaign and inventory management, essential for dynamic, multilingual businesses.
- **Personalization and User Engagement:** Modern recommender systems, when properly validated and personalized, directly drive engagement and retention in OTT, e-commerce, and digital content industries.
- **Automated Content Categorization:** Text classifiers streamline news and content delivery, powering smarter feeds, targeted marketing, and real-time analytics.

Limitations and Suggestions for Improvement

- **Data & Sampling Issues:** Analyses often reflect historical or biased samples (urban-focused, time-limited), calling for broader, more current datasets and regular bias assessment.
- **Model Limitations:** Linear and univariate models can miss complex relationships and abrupt shocks; supplementing with ensembles, exogenous variables, or neural networks can improve performance on volatile or sparse patterns.
- **Operational Gaps:** Manual processes (e.g., ARIMA tuning) and language/region-specific idiosyncrasies challenge full automation; investment in scalable automation and hybrid approaches is needed.

- **Validation and Cold Start:** Recommendation systems face "cold start" and precision challenges with new users/items; integrating explicit feedback and hybrid architectures can mitigate these limits.
- **Policy and Fairness:** Continued focus on explainability, fairness, and adaptation to regulatory requirements ensures ethical, compliant deployment, especially as models reach sensitive user segments or new domains.

References

Below are representative references following your requested format. For a real assignment, add all consulted works with accurate details for each project module.

Name of Website/Source	Date & Time	Author (if known)	Title/Topic	Year
https://www.kaggle.com/	09-Nov-2025, 21:00 IST	--	Data Science Case Study Datasets	2025
https://facebook.github.io/prophet/	09-Nov-2025, 22:10 IST	Prophet Team	FB Prophet Time Series Model Documentation	2024
https://scikit-learn.org/	09-Nov-2025, 22:15 IST	scikit-learn Developers	Machine Learning in Python — Model Reference	2024
https://www.statlearning.com/	08-Nov-2025, 19:30 IST	James, Witten, Hastie, Tibshirani	An Introduction to Statistical Learning (ISL)	2021
https://www.researchgate.net/	09-Nov-2025, 20:10 IST	Various	Academic & Applied ML research papers: Recommender Systems, NLP	Various
https://towardsdatascience.com/	09-Nov-2025, 22:35 IST	Multiple Contributors	Applied ML Tutorials (Logistic Regression, ARIMA, Recommender Sys)	2023-2025
Sharma, A.	--	Credit Risk Modeli	Indian J. Finance & Analytics	2022

		ng— Best Practic es		
Aggarwal, C.C.	--	Recom mend er Syste ms: Fundamental s & Trends		2016