

データサイエンス演習 第9回レポート

7月31日

2210593 松浦史明

1. Introduction

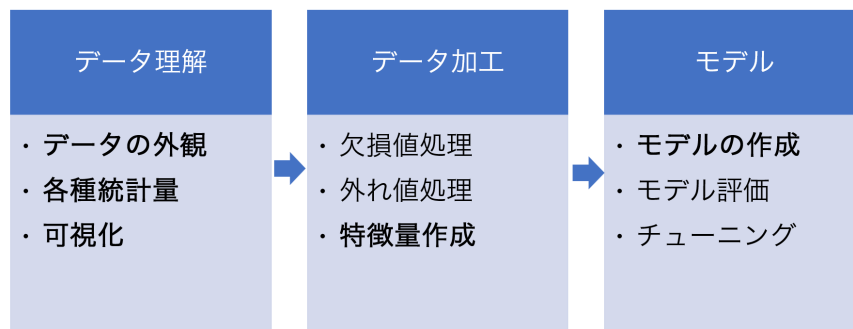
1.1 本レポートの流れ

まず、私が行った分析の流れについて説明する。次に、その過程で気になった点を discussion セクションへ記載する。ソースコードについては、付録へ記載する。

1.2 分析の流れ

本分析では、まずデータセットを理解し、次に特徴量エンジニアリングを行い、機械学習モデルを用いて予測モデルを構築した。主に、以下の3つの点について行った。

- ・ **データの理解**：データをグラフ等を用いて可視化することで探索を行い、主要な統計量やデータの分布を確認した。
- ・ **特徴量エンジニアリング**：重要な特徴量を選定し、不要な特徴量を削除することで、モデルの精度を向上させた。
- ・ **モデルの作成**：LightGBM を用いてモデル作成を行った。クロスバリデーションを行い、モデルの精度を検証した。



第11回講義資料より

図1 データ分析の流れ

2. Methods

2.1 データの理解

データの初期探索（Exploratory Data Analysis, EDA）から始めた。データの構造を把握するため、各列の統計量の確認、欠損値の割合の確認、目的変数と各特微量との関係を計算し、グラフへ起こし可視化するなどして、データへの理解を深めた。

例えば、教育のタイプ（NAME_EDUCATION_TYPE）と、目的変数の関係を可視化した場合、次の図の通りとなった。

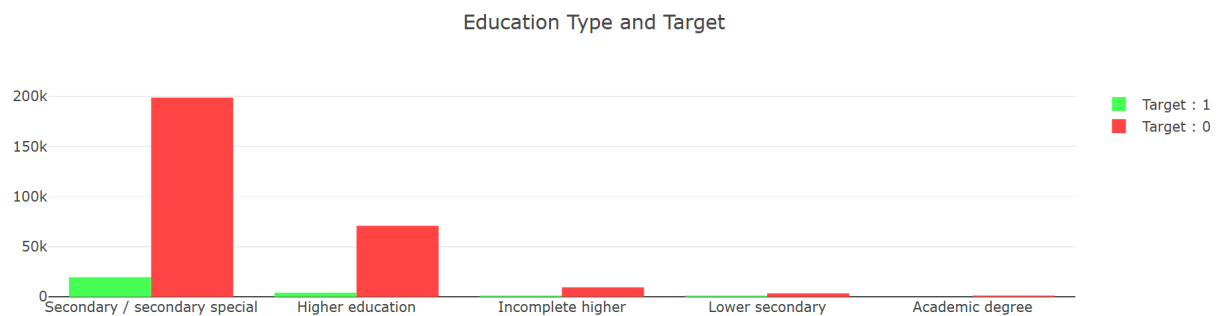


図2 NAME_EDUCATION_TYPE と Target との値の関係

このグラフから、異なる教育レベルが、返済能力にどのように影響しているかをおおまかに見ることができる。この場合、多くのデータセットが中等教育を示している事が分ると同時に、中等教育の母数が多いことから、中等教育のデータの目的変数が1となっていることも多そうであるということが分かる。

2.2 特微量の追加と削除

特微量エンジニアリングの段階では、データセットに含まれる情報を最大限に活用し、予測モデルの精度を高めるために新しい特微量を作成し、不要な特微量を削除することとした。

資料を参考にした特微量の追加・削除

意味のある特微量を作成し、ノイズとなるような特微量を削除する必要があったが、私自身の金利やローンに対する理解は専門家のそれに追いつけることができなかったことから、Kaggler 上位者が作成している特微量 [1] [2] [3]を参考に、主に次のような特微量を作成した。

- ・ DAYS_EMPLOYED_PERC：雇用日数と生まれた日数の比率。
- ・ INCOME_CREDIT_PERC：総収入とクレジット額の比率。借入れに対する収入の比率を示す。
- ・ INCOME_PER_PERSON：1人あたりの収入。
- ・ ANNUITY_INCOME_PERC：年金と総収入の比率。年金支払いが収入に占める割合を示す。

- ・ PAYMENT_RATE：年金とクレジット額の比率、返済負担の重さを示す。

また、特徴量の削除については、データの理解の段階で偏りが大きいデータであると分かっていたり、情報量が少ないと考えられるような特徴量を削除することとした。以下が、主に削除した特徴量である。

- ・ FONDKAPREMONT_MODE：ファンドに関するデータ。
- ・ WALLSMATERIAL_MODE：壁の材料に関するデータ。
- ・ HOUSETYPE_MODE：住宅タイプに関するデータ。
- ・ EMERGENCYSTATE_MODE：緊急状態かどうかに関するデータ。
- ・ FLAG_MOBIL：携帯電話を所有しているかどうか示すフラグ。ほとんどの値が1で偏っている。
- ・ FLAG_EMP_PHONE：職場電話を所有しているかどうか示すフラグ。情報量が少ない。
- ・ FLAG_WORK_PHONE：勤務先電話を所有しているかどうか示すフラグ。
- ・ FLAG_CONT_MOBILE：携帯電話連絡が可能かどうか示すフラグ。
- ・ FLAG_EMAIL：Eメールを所有しているかどうか示すフラグ。
- ・ OBS_30_CNT_SOCIAL_CIRCLE：30日以内に社会的交流があるかどうか。情報量が少ない。
- ・ OBS_60_CNT_SOCIAL_CIRCLE：60日以内に社会的交流があるかどうか。

Importance 値による特徴量の選択

LightGBM のライブラリで提供されている Importance 値を求める関数を用いて、目的変数の説明に特徴量がどれくらい寄与しているかを示す値を算出した。重要度が高い特徴量は選択すべき特徴量であり、重要度が低い特徴量は消去することができ、それによりスコアが上昇する可能性がある。Importance 値を算出し、上から順に抜粋した結果は次の通りである。

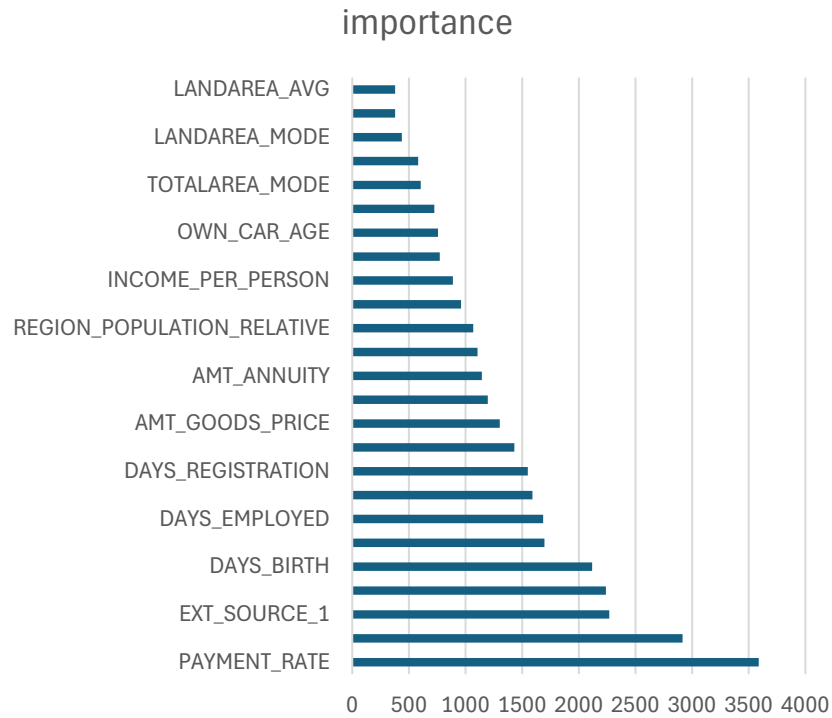


図3 Importance 値

この結果を考慮して，前述の特徴量の追加と削除を行った．

相関の高い変数の削除

一般的に，相関の高い変数はどちらか片方を消すと良い [4]．そのため，相関行列を作成し，指定したしきい値以上の相関を持つ特徴量を削除するようにした．

2.3 モデルの訓練

LightGBM を用いて，Kfold 法を用いたモデルの学習を行った．



図4 Kfold 法 [5]

通常機械学習を行う際、データを訓練データと検証データに分けてモデルをトレーニングし、そのモデルを使用して結果を予測する。一方で、クロスバリデーションでは、この訓練データと検証データを上図のように入れ替えて、それぞれのモデルで予測した値の平均値、もしくは最頻値を最終的な予測値として使用する [図 4]。今回は、検証データはデータの 1/15 を用いることとした。

2.4 スコアの向上

以上の手段を全て活用した上で、スコアの向上を目指し、特徴量の選定を繰り返し行った。重要な特徴量を保持し、情報量の少ない特徴量を削除することで、モデルのノイズを減少させ、予測精度を向上させることとした。また、LightGBM のパラメータの調整を試みることで、最適なモデル構成を探求した。

3. Results

3.1 最終的な結果

分析の結果、LightGBM モデルを用いた際のスコアは 0.79136 となった。なお、このモデルの評価は ROC-AUC スコアで行った。

 20240715_v11_0.792182.csv Complete (after deadline) · 4d ago	0.79136	0.79168	<input type="checkbox"/>
---	---------	---------	--------------------------

図5 最終的なスコア

3.2 試した手法ごとの結果

初期段階では、授業資料に基づいた基本的な決定木モデルを使用した [図 6] が、LightGBM を使用し、特徴量の選定とクロスバリデーションを組み合わせることでスコアが大幅に改善された。特に、application データのみを使用した場合 [図 7] と比べ、bureau and balance データを追加して特徴量エンジニアリングを行った際のスコア [図 8] の向上が顕著であった。これにより、データソースを追加することの重要性と、適切な特徴量選択がモデルの予測精度に大きく影響を与えることが分かった。


 submit_tree_20210829_1.csv Complete (after deadline) · 12d ago · tree_model = DecisionTreeClassifier(criterion="gini", # Entropy基準の場合は"entropy..."	0.66105	0.67113	<input type="checkbox"/>
--	---------	---------	--------------------------

図6 初期段階のスコア



20240709_test.csv

Complete (after deadline) · 10d ago

0.76072

0.76536



図7 データは application のみを使用し, Kfold を使用した場合



20240710_v1.csv

Complete (after deadline) · 10d ago · add bureau and balance to dataframe

0.77535

0.77500



図8 application と bureau and balance を使用し, 特徴量選択を実施した場合

4. Discussion

4.1 成功した点

特徴量の追加や削除により, モデルの予測精度を向上させることに成功した. また, クロスバリデーションを用いることで, モデルの汎用性を確保し, 異なるデータセットに対する適応性を高めることができた.

4.2 改善点

一部の特徴量に関しては, 追加や削除, 統合や相関に基づくフィルタリングを試みたが, スコアを改善させることができなかった. 今後は, 更にデータの深掘りを行い, 隠れたパターンやより影響力のある特徴量を探索する必要がある. また, その他の機械学習アルゴリズムの組み合わせやアンサンブル手法を探索することで, さらなるスコアの向上をさせることができると考える.

4.3 気になった点

相関係数による特徴量選択の問題点

本分析では, 特徴量の削減を目的として相関係数が高い特徴量の削除を試みたが, 予想していたスコアの向上に繋がることは無かった. 具体的には, いくつかの重要と思われる特徴量が相関係数の基準によって除外され, モデルの予測精度が低下するという事態が発生した. これは, 単純な相関係数のみに基づく特徴量の選択が, 必ずしも複雑な構造のデータや予測モデルを構築するための説明変数の選択に適していないことを示していると考えられる.

改善策

以上のことから, 特徴量選択の方法として, 相関係数に過度に依存することのリスクが問題として判明した. 今後は, 特徴量の重要性を評価する際に, 相関係数だけでなく, 他の統計的手法を利

用して、よりバランスの取れた方法を取ることが求められる。また、相関係数の高い特徴量を削除する際には、その特徴量が目的変数に与える影響をより慎重に評価し、判断することが重要である。

5. Conclusion

今学期のデータサイエンス演習では、Home Credit Default Risk データセットを用いて、クライアントのローン返済能力を予測するモデルを開発した。データの探索、特徴量の工夫、モデルの訓練と最適化を通じて、信頼性の高い予測が可能なモデルを構築することができた。データを更に分析し、モデルの改善を続けることで、より高い精度の予測モデルの開発ができると考える。

6. References

- [1] S-Analysis, “kaggle1 位の解析手法 「Home Credit Default Risk 債務不履行の予測」 ②特徴量エンジニアリング,” [オンライン]. Available: <https://data-analysis-stats.jp/kaggle/kaggle%E4%BD%8D%E3%81%AE%E8%A7%A3%E6%9E%90%E6%89%8B%E6%B3%95%E3%80%80%E3%80%8CHome-credit-default-risk-%E5%82%B5%E5%8B%99%E4%B8%8D%E5%B1%A5%E8%A1%8C%E3%81%AE%E4%BA%88%E6%B8%AC%E3%80%8D%E2%91%A1/>.
- [2] ねほり.com, “Kaggle の Home Credit Default Risk 体験（他人の Kernel パクリ編）,” [オンライン]. Available: <https://nehoori.com/nikki/2020/01/19/post-14961/>.
- [3] OsciiArt, “HomeCreditRisk : Extensive EDA + Baseline Model JP,” [オンライン]. Available: <https://www.kaggle.com/code/osciiart/homecreditrisk-extensive-eda-baseline-model-jp>.
- [4] 大.長倉, “統計学 最小二乗法 7 多重共線性,” [オンライン]. Available: https://user.keio.ac.jp/~nagakura/stat2017A/stat12_slide_2017_autumn.pdf.
- [5] DS Media, “クロスバリデーションでスコアアップ！交差検証をしてみる,” [オンライン]. Available: <https://www.tech-teacher.jp/blog/cross-validation/>.

7. Appendix

ソースコード

```
1. import numpy as np
2. import pandas as pd
3. import gc
4. import time
5. from contextlib import contextmanager
6. import lightgbm as lgb
7. from lightgbm import LGBMClassifier
8. from sklearn.metrics import roc_auc_score, roc_curve
9. from sklearn.model_selection import KFold, StratifiedKFold
10. import matplotlib.pyplot as plt
11. import seaborn as sns
12. import warnings
13. import shap
14.
15. submission_file_name = "submission_kernel02.csv"
16.
17. # 以下データ整形に関連する関数
18.
19. # 文字列型などのオブジェクト型データを含む列を、ワンホットエンコードする
20. # ワンホットエンコードとは：カテゴリ変数の各カテゴリを新しいダミー変数（特徴量）に変換し、
21. # そのカテゴリに属するデータのみを「1」、それ以外を「0」とするバイナリ形式で表現する手法
22. def one_hot_encoder(df, nan_as_category = True):
23.     original_columns = list(df.columns)
24.     categorical_columns = [col for col in df.columns if df[col].dtype == 'object']
25.     df = pd.get_dummies(df, columns= categorical_columns, dummy_na= nan_as_category)
26.     new_columns = [c for c in df.columns if c not in original_columns]
27.     return df, new_columns
28.
29. def clean_feature_names(df):
30.     # 特徴量名のクリーニングを行う
31.     # JSON の特殊文字をアンダースコアに置換
32.     df.columns = [col.replace('{', '_').replace('}', '_').replace(':', '_').replace(',', '_') for col in df.columns]
33.     return df
34.
```



```

35. def df_init(num_rows = None, nan_as_category = False):
36.     # csv を読み込んで、df にセット
37.     df = pd.read_csv('data/application_train.csv', nrows= num_rows)
38.     test_df = pd.read_csv('data/application_test.csv', nrows= num_rows)
39.     print("Train samples: {}, test samples: {}".format(len(df), len(test_df)))
40.
41.     # Train データと Test データを縦に結合し、インデックスをリセット
42.     df = pd.concat([df, test_df]).reset_index(drop=True)
43.
44.     # CODE_GENDER が'XNA'であるデータを除外
45.     df = df[df['CODE_GENDER'] != 'XNA']
46.     # 授業資料では、性別（CODE_GENDER）欠損値（XNA）であった場合、最頻値（F）として
    いた
47.     # df.loc[df['CODE_GENDER'] == 'XNA', 'CODE_GENDER'] = 'F'
48.
49.     # 資料の特徴量
50.     # 収入区分（NAME_INCOME_TYPE）は全部で 8 区分あるが、うち 4 区分は件数が小さい（30
    件以下）ため、
51.     # 近い意味合いのクラスに併合してしまうこととする。
52.     # df.loc[df['NAME_INCOME_TYPE'] == 'Businessman', 'NAME_INCOME_TYPE'] = 'Commercial
    associate'
53.     # df.loc[df['NAME_INCOME_TYPE'] == 'Maternity leave', 'NAME_INCOME_TYPE'] = 'Pensioner'
54.     # df.loc[df['NAME_INCOME_TYPE'] == 'Student', 'NAME_INCOME_TYPE'] = 'State servant'
55.     # df.loc[df['NAME_INCOME_TYPE'] == 'Unemployed', 'NAME_INCOME_TYPE'] = 'Pensioner'
56.
57.     # 組織区分（ORGANIZATION_TYPE）は区分が多すぎるため、
58.     # 頻度の小さいものは併合して整理することとする。
59.     # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].str.contains("Business
    Entity"),
60.     #                                     "Business_Entity", df["ORGANIZATION_TYPE"])
61.     # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].str.contains("Industry"),
62.     #                                     "Industry", df["ORGANIZATION_TYPE"])
63.     # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].str.contains("Trade"),
64.     #                                     "Trade", df["ORGANIZATION_TYPE"])
65.     # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].str.contains("Transport"),
66.     #                                     "Transport", df["ORGANIZATION_TYPE"])
67.     # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].isin(["School",
    "Kindergarten", "University"]),

```

```

68. # "Education", df["ORGANIZATION_TYPE"])
69. # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].isin(["Emergency", "Police",
    "Medicine", "Government", "Postal", "Military", "Security Ministries", "Legal Services"]),
70. # "Official", df["ORGANIZATION_TYPE"])
71. # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].isin(["Bank", "Insurance"]),
72. # "Finance", df["ORGANIZATION_TYPE"])
73. # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].str.contains("Government"),
74. # "Government", df["ORGANIZATION_TYPE"])
75. # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].isin(["Realtor",
    "Housing"]),
76. # "Realty", df["ORGANIZATION_TYPE"])
77. # df["ORGANIZATION_TYPE"] = np.where(df["ORGANIZATION_TYPE"].isin(["Hotel",
    "Restaurant", "Services"]),
78. # "TourismFoodSector", df["ORGANIZATION_TYPE"])
79. # df["ORGANIZATION_TYPE"] =
    np.where(df["ORGANIZATION_TYPE"].isin(["Cleaning", "Electricity", "Telecom", "Mobile", "Advertising",
    "Religion", "Culture"]),
80. # "Other", df["ORGANIZATION_TYPE"])
81.
82. # 職業区分 (OCCUPATION_TYPE) は区分が多すぎるため、
83. # 頻度の小さいものは併合して整理することとする。
84. # df["OCCUPATION_TYPE"] = np.where(df["OCCUPATION_TYPE"].isin(["Low-skill Laborers",
    "Cooking staff", "Security staff", "Private service staff", "Cleaning staff", "Waiters/barmen staff"]),
85. # "Low_skill_staff", df["OCCUPATION_TYPE"])
86. # df["OCCUPATION_TYPE"] = np.where(df["OCCUPATION_TYPE"].isin(["IT staff", "High skill tech
    staff"]),
87. # "High_skill_staff", df["OCCUPATION_TYPE"])
88. # df["OCCUPATION_TYPE"] = np.where(df["OCCUPATION_TYPE"].isin(["Secretaries", "HR
    staff", "Realty agents"]),
89. # "Others", df["OCCUPATION_TYPE"])
90.
91. # 申込様式 (NAME_TYPE_SUITE) は細かい区分が存在するが、構成比 1%を切るものは全て
    Rare としてまとめる。
92. # また、Spouse, partner はスペースが入っていて使いづらいので改名しておく。
93. # tmp = df["NAME_TYPE_SUITE"].value_counts() / len(df)
94. # rare_labels = tmp[tmp < 0.01].index
95. # df["NAME_TYPE_SUITE"] = np.where(df["NAME_TYPE_SUITE"].isin(rare_labels), 'Rare',
    df["NAME_TYPE_SUITE"])

```

```

96. # del tmp
97. # gc.collect()
98. # df["NAME_TYPE_SUITE"] = np.where(df["NAME_TYPE_SUITE"].str.contains("Spouse, partner"),
99. #                                   "Spouse_partner", df["NAME_TYPE_SUITE"])
100. # 学歴区分 (NAME_EDUCATION_TYPE) のうち「Academic degree」は頻度が低いので、
101. # Higher education に併合して整理することとする。
102. # また、Secondary / secondary special はスペースが入っていて使いづらいので改名しておく。
103. # df["NAME_EDUCATION_TYPE"] = np.where(df["NAME_EDUCATION_TYPE"] == "Academic
    degree",
104. #                                       "Higher education", df["NAME_EDUCATION_TYPE"])
105. # df["NAME_EDUCATION_TYPE"] =
    np.where(df["NAME_EDUCATION_TYPE"].str.contains("Secondary / secondary special"),
106. #        "Secondary_secondary_special", df["ORGANIZATION_TYPE"])
107. #
108. # # NAME_FAMILY_STATUS の Single / not married はスペースが入っていて使いづらいので改名
    しておく。
109. # df["NAME_FAMILY_STATUS"] = np.where(df["NAME_FAMILY_STATUS"].str.contains("Single / not
    married"),
110. #                                       "Single_not_married", df["NAME_FAMILY_STATUS"])
111. #
112. # # NAME_HOUSING_TYPE の House / apartment はスペースが入っていて使いづらいので改名
    しておく。
113. # df["NAME_HOUSING_TYPE"] = np.where(df["NAME_HOUSING_TYPE"].str.contains("House /
    apartment"),
114. #                                       "House_apartment", df["NAME_HOUSING_TYPE"])
115.
116. # 変数の削除
117. # 以下にあげるカテゴリ変数は、欠損率が高いため今回は使わない
118. drop_cols = ["FONDKAPREMONT_MODE", "WALLSMATERIAL_MODE",
    "HOUSETYPE_MODE", "EMERGENCYSTATE_MODE"]
119. df.drop(drop_cols, axis = 1, inplace = True)
120.
121. # 以下にあげるフラグは偏りが大きく（ほとんど 0 または 1）、情報量が少ないと考えられるた
    め、今回は使わない
122. drop_cols = ["FLAG_MOBIL", "FLAG_EMP_PHONE", "FLAG_WORK_PHONE",
    "FLAG_CONT_MOBILE", "FLAG_EMAIL",
123.               'OBS_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE']
124. df.drop(drop_cols, axis = 1, inplace = True)

```

```

125.
126.
127. # 以下のカテゴリ変数を数値に変換する。変換された数値は 0 から始まる。
128. # 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY'の各列に対して実行
129. for bin_feature in ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY']:
130.     df[bin_feature], uniques = pd.factorize(df[bin_feature])
131.
132. # カテゴリカルな特徴量をワンホットエンコーディングします。NaN がある場合の扱いは引数
    に依存
133. df, cat_cols = one_hot_encoder(df, nan_as_category)
134.
135. # 特徴量名のクリーニング
136. df = clean_feature_names(df)
137.
138. # 'DAYS_EMPLOYED'の特定の値（365243）を NaN に置き換え
139. # 就労日数（DAYS_EMPLOYED）が 365243 人という方がいるが、異常値なので一旦、欠損値
    として扱う
140. # df['DAYS_EMPLOYED'].replace(365243, np.nan, inplace=True)
141. df['DAYS_EMPLOYED'] = df['DAYS_EMPLOYED'].replace(365243, np.nan)
142.
143. # 新しい特徴量の生成
144. df['DAYS_EMPLOYED_PERC'] = df['DAYS_EMPLOYED'] / df['DAYS_BIRTH'] # 雇用日数と生まれ
    た日数の比率
145. df['INCOME_CREDIT_PERC'] = df['AMT_INCOME_TOTAL'] / df['AMT_CREDIT'] # 総収入とク
    レジット額の比率
146. df['INCOME_PER_PERSON'] = df['AMT_INCOME_TOTAL'] / df['CNT_FAM_MEMBERS'] # 1 人
    あたりの収入
147. df['ANNUITY_INCOME_PERC'] = df['AMT_ANNUITY'] / df['AMT_INCOME_TOTAL'] # 年金と
    総収入の比率
148. df['PAYMENT_RATE'] = df['AMT_ANNUITY'] / df['AMT_CREDIT'] # 年金とクレジット額の比率
149.
150.
151. # 以下自作特徴量エンジニアリング
152. # df.drop(columns = ["ENTRANCES_AVG", "ELEVATORS_AVG", "YEARS_BUILD_MEDI"]) # これは
    やっても特に変化なし
153.
154.
155. ## 相関の高いものを削除

```

```

156.  ## 相関行列を計算する
157.  # corr_matrix = df.corr().abs()
158.  ## 相関行列の上三角行列を取得
159.  # upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool_))
160.  ## 相関係数が 0.8 以上の特徴量の一覧を取得
161.  # to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
162.  ## 相関係数が高い特徴量を削除
163.  # df.drop(to_drop, axis=1, inplace=True)
164.  #
165.  # print("dropped columns: {}".format(to_drop))
166.
167.  # テストデータ用の DataFrame を削除し、ガベージコレクションを実行してメモリを解放
168.  del test_df
169.  gc.collect()
170.
171.  # データフレームの上から数行を csv として出力
172.  df_origin_head = df.head(5)
173.  df_origin_head.to_csv('df_origin.csv', index=False)
174.
175.  # 加工後のデータフレームを返す
176.  return df
177.
178. def df_adjust_bureau_and_balance(num_rows = None, nan_as_category = False):
179.     bureau = pd.read_csv('data/bureau.csv', nrows = num_rows)
180.     bb = pd.read_csv('data/bureau_balance.csv', nrows = num_rows)
181.
182.     # 'bureau_balance.csv' から読み込んだデータにワンホットエンコーディングを適用し、
183.     # カテゴリ変数をダミー変数に変換します。nan_as_category が True の場合、NaN もカテゴリと
        して扱う
184.     bb, bb_cat = one_hot_encoder(bb, nan_as_category)
185.     # 同様に 'bureau.csv' データに対してもワンホットエンコーディングを行う
186.     bureau, bureau_cat = one_hot_encoder(bureau, nan_as_category)
187.
188.     # 特徴量名のクリーニング
189.     bb = clean_feature_names(bb)
190.     bureau = clean_feature_names(bureau)
191.
192.     # Bureau Balance の集計操作を行い、SK_ID_BUREAU に基づいてグループ化し、

```

```

193. # 最小値、最大値、頻度を計算します。また、ワンホットエンコーディングで追加されたカテ
    # ゴリごとに平均を取る
194. bb_aggregations = {'MONTHS_BALANCE': ['min', 'max', 'size']}
195. for col in bb_cat:
196.     bb_aggregations[col] = ['mean']
197. bb_agg = bb.groupby('SK_ID_BUREAU').agg(bb_aggregations)
198. bb_agg.columns = pd.Index([e[0] + "_" + e[1].upper() for e in bb_agg.columns.tolist()])
199.
200. # 集計結果を元の bureau データフレームに結合
201. bureau = bureau.join(bb_agg, how='left', on='SK_ID_BUREAU')
202. # SK_ID_BUREAU カラムはもはや不要なので削除
203. bureau.drop(['SK_ID_BUREAU'], axis=1, inplace= True)
204.
205. # 不要なオブジェクトを削除してメモリを解放
206. del bb, bb_agg
207. gc.collect()
208.
209. # 以下資料の参考箇所
210. # 負債比（＝負債総額÷ローン総額）NEW_DEPT_RATIO を作成する
211. # bureau['NEW_DEPT_RATIO'] = bureau['AMT_CREDIT_SUM_DEBT'] /
    (bureau['AMT_CREDIT_SUM']+1)
212.
213. # ローンの新旧（90 日以前だったら旧：old、そうでなければ新：new）
    NEWS_DAYS_CREDIT_UPDATE を作成する
214. bureau['NEWS_DAYS_CREDIT_UPDATE'] = bureau['DAYS_CREDIT_UPDATE'].apply(lambda x :
    'old' if x < -90 else 'new')
215.
216.
217. # Bureau データに対する数値的特徴量の集計を定義し、
218. # それぞれの特徴量について最小値、最大値、平均、分散などを計算
219. num_aggregations = {
220.     'DAYS_CREDIT': ['min', 'max', 'mean', 'var'],
221.     'DAYS_CREDIT_ENDDATE': ['min', 'max', 'mean'],
222.     'DAYS_CREDIT_UPDATE': ['mean'],
223.     'CREDIT_DAY_OVERDUE': ['max', 'mean'],
224.     'AMT_CREDIT_MAX_OVERDUE': ['mean'],
225.     'AMT_CREDIT_SUM': ['max', 'mean', 'sum'],
226.     'AMT_CREDIT_SUM_DEBT': ['max', 'mean', 'sum'],

```

```

227.     'AMT_CREDIT_SUM_OVERDUE': ['mean'],
228.     'AMT_CREDIT_SUM_LIMIT': ['mean', 'sum'],
229.     'AMT_ANNUITY': ['max', 'mean'],
230.     'CNT_CREDIT_PROLONG': ['sum'],
231.     'MONTHS_BALANCE_MIN': ['min'],
232.     'MONTHS_BALANCE_MAX': ['max'],
233.     'MONTHS_BALANCE_SIZE': ['mean', 'sum']
234. }
235.
236. # カテゴリ変数についても平均を集計
237. cat_aggregations = {}
238. for cat in bureau_cat: cat_aggregations[cat] = ['mean']
239. for cat in bb_cat: cat_aggregations[cat + "_MEAN"] = ['mean']
240.
241. # 全集計を行い、集約されたデータフレームを作成
242. bureau_agg = bureau.groupby('SK_ID_CURR').agg(**num_aggregations, **cat_aggregations)
243. bureau_agg.columns = pd.Index(['BURO_' + e[0] + "_" + e[1].upper() for e in bureau_agg.columns.tolist()])
244.
245. # アクティブなクレジットのみを抽出し、集計
246. active = bureau[bureau['CREDIT_ACTIVE_Active'] == 1]
247. active_agg = active.groupby('SK_ID_CURR').agg(num_aggregations)
248. active_agg.columns = pd.Index(['ACTIVE_' + e[0] + "_" + e[1].upper() for e in active_agg.columns.tolist()])
249. bureau_agg = bureau_agg.join(active_agg, how='left', on='SK_ID_CURR')
250.
251. # 同様にクローズドなクレジットも集計
252. closed = bureau[bureau['CREDIT_ACTIVE_Closed'] == 1]
253. closed_agg = closed.groupby('SK_ID_CURR').agg(num_aggregations)
254. closed_agg.columns = pd.Index(['CLOSED_' + e[0] + "_" + e[1].upper() for e in
    closed_agg.columns.tolist()])
255. bureau_agg = bureau_agg.join(closed_agg, how='left', on='SK_ID_CURR')
256.
257. # 再び不要なオブジェクトを削除してメモリを解放
258. del closed, closed_agg, bureau
259. gc.collect()
260.
261. # 最終的に集計されたデータフレームを返す
262. return bureau_agg
263.

```

```

264. def df_adjust_previous_applications(num_rows = None, nan_as_category = False):
265.     prev = pd.read_csv('data/previous_application.csv', nrows = num_rows)
266.     prev, cat_cols = one_hot_encoder(prev, nan_as_category= True)
267.
268.     # 特徴量名のクリーニング
269.     # prev = clean_feature_names(prev)
270.
271.     # Days 365.243 values -> nan
272.     prev['DAYS_FIRST_DRAWING'] = prev['DAYS_FIRST_DRAWING'].replace(365243, np.nan)
273.     prev['DAYS_FIRST_DUE'] = prev['DAYS_FIRST_DUE'].replace(365243, np.nan)
274.     prev['DAYS_LAST_DUE_1ST_VERSION'] =
        prev['DAYS_LAST_DUE_1ST_VERSION'].replace(365243, np.nan)
275.     prev['DAYS_LAST_DUE'] = prev['DAYS_LAST_DUE'].replace(365243, np.nan)
276.     prev['DAYS_TERMINATION'] = prev['DAYS_TERMINATION'].replace(365243, np.nan)
277.     # prev['DAYS_FIRST_DRAWING'].replace(365243, np.nan, inplace= True)
278.     # prev['DAYS_FIRST_DUE'].replace(365243, np.nan, inplace= True)
279.     # prev['DAYS_LAST_DUE_1ST_VERSION'].replace(365243, np.nan, inplace= True)
280.     # prev['DAYS_LAST_DUE'].replace(365243, np.nan, inplace= True)
281.     # prev['DAYS_TERMINATION'].replace(365243, np.nan, inplace= True)
282.
283.
284.     # Add feature: value ask / value received percentage
285.     prev['APP_CREDIT_PERC'] = prev['AMT_APPLICATION'] / prev['AMT_CREDIT']
286.     # Previous applications numeric features
287.     num_aggregations = {
288.         'AMT_ANNUITY': ['min', 'max', 'mean'],
289.         'AMT_APPLICATION': ['min', 'max', 'mean'],
290.         'AMT_CREDIT': ['min', 'max', 'mean'],
291.         'APP_CREDIT_PERC': ['min', 'max', 'mean', 'var'],
292.         'AMT_DOWN_PAYMENT': ['min', 'max', 'mean'],
293.         'AMT_GOODS_PRICE': ['min', 'max', 'mean'],
294.         'HOUR_APPR_PROCESS_START': ['min', 'max', 'mean'],
295.         'RATE_DOWN_PAYMENT': ['min', 'max', 'mean'],
296.         'DAYS_DECISION': ['min', 'max', 'mean'],
297.         'CNT_PAYMENT': ['mean', 'sum'],
298.     }
299.     # Previous applications categorical features
300.     cat_aggregations = {}

```



```

301. for cat in cat_cols:
302.     cat_aggregations[cat] = ['mean']
303.
304. # cat_aggregations = {}
305. # for cat in cat_cols:
306. #     if cat in prev.columns: # データに列が存在することを確認
307. #         if "PRODUCT_COMBINATION" in cat or "NAME_TYPE_SUITE" in cat:
308. #             # 接頭辞を除去して正しいカテゴリ名のみを使用
309. #             corrected_cat = cat.split("_")[-1] # これで、例えば 'Cash Street: high' のような形式になる
310. #             cat_aggregations[corrected_cat] = ['mean']
311. #         else:
312. #             cat_aggregations[cat] = ['mean']
313. #     else:
314. #         cat_aggregations[cat] = ['mean']
315.
316.
317. # print(num_aggregations)
318. # print(cat_aggregations)
319.
320. prev_agg = prev.groupby('SK_ID_CURR').agg(**num_aggregations, **cat_aggregations)
321. prev_agg.columns = pd.Index(['PREV_' + e[0] + "_" + e[1].upper() for e in prev_agg.columns.tolist()])
322. # Previous Applications: Approved Applications - only numerical features
323. approved = prev[prev['NAME_CONTRACT_STATUS_Approved'] == 1]
324. approved_agg = approved.groupby('SK_ID_CURR').agg(num_aggregations)
325. approved_agg.columns = pd.Index(['APPROVED_' + e[0] + "_" + e[1].upper() for e in
approved_agg.columns.tolist()])
326. prev_agg = prev_agg.join(approved_agg, how='left', on='SK_ID_CURR')
327. # Previous Applications: Refused Applications - only numerical features
328. refused = prev[prev['NAME_CONTRACT_STATUS_Refused'] == 1]
329. refused_agg = refused.groupby('SK_ID_CURR').agg(num_aggregations)
330. refused_agg.columns = pd.Index(['REFUSED_' + e[0] + "_" + e[1].upper() for e in
refused_agg.columns.tolist()])
331. prev_agg = prev_agg.join(refused_agg, how='left', on='SK_ID_CURR')
332. del refused, refused_agg, approved, approved_agg, prev
333. gc.collect()
334.
335. # 特徴量名のクリーニング
336. prev_agg = clean_feature_names(prev_agg)

```

```

337.
338.  ## 以下自作特徴量エンジニアリング
339.  # df.drop(columns = ["ENTRANCES_AVG", "ELEVATORS_AVG", "EARS_BUILD_MEDI"])
340.
341.  return prev_agg
342.
343. def df_adjust_pos_cash(num_rows = None, nan_as_category = False):
344.     pos = pd.read_csv('data/POS_CASH_balance.csv', nrows = num_rows)
345.     pos, cat_cols = one_hot_encoder(pos, nan_as_category= True)
346.
347.     # 特徴量名のクリーニング
348.     pos = clean_feature_names(pos)
349.
350.     # Features
351.     aggregations = {
352.         'MONTHS_BALANCE': ['max', 'mean', 'size'],
353.         'CNT_INSTALLMENT': ['max', 'mean', 'std', 'min', 'median'], #ここを追加
354.         'CNT_INSTALLMENT_FUTURE': ['max', 'mean', 'sum', 'min', 'median', 'std'], #ここを追加
355.         'SK_DPD': ['max', 'mean'],
356.         'SK_DPD_DEF': ['max', 'mean']
357.     }
358.     for cat in cat_cols:
359.         aggregations[cat] = ['mean']
360.
361.     # ここで SK_ID_CURR を基準に、データを集約
362.     # 特徴量を集約している
363.     pos_agg = pos.groupby('SK_ID_CURR').agg(aggregations)
364.     pos_agg.columns = pd.Index(['POS_' + e[0] + "_" + e[1].upper() for e in pos_agg.columns.tolist()])
365.     # Count pos cash accounts
366.     pos_agg['POS_COUNT'] = pos.groupby('SK_ID_CURR').size()
367.     del pos
368.     gc.collect()
369.     return pos_agg
370.
371. def df_adjust_installments_payments(num_rows = None, nan_as_category = False):
372.     ins = pd.read_csv('data/installments_payments.csv', nrows = num_rows)
373.     ins, cat_cols = one_hot_encoder(ins, nan_as_category= True)
374.

```

```

375. # 特徴量名のクリーニング
376. ins = clean_feature_names(ins)
377.
378. ## 自作
379. ## 日付関連のデータ型を日付型に変換
380. # ins['PAYMENT_DATE'] = pd.to_datetime(ins['PAYMENT_DATE'])
381. #
382. ## "TODAY" として分析日を設定 (例: データの最新日)
383. # TODAY = ['PAYMENT_DATE'].max()
384. #
385. ## 過去 30 日間で過去 90 日間の支払いを集計
386. # past_30 = ins[ins['PAYMENT_DATE'] >= TODAY - pd.Timedelta(days=30)]
387. # past_90 = ins[ins['PAYMENT_DATE'] >= TODAY - pd.Timedelta(days=90)]
388. #
389. ## 各期間の支払い総額を計算
390. # sum_past_30 = past_30.groupby('SK_ID_CURR')['AMT_PAYMENT'].sum().rename('SUM_30')
391. # sum_past_90 = past_90.groupby('SK_ID_CURR')['AMT_PAYMENT'].sum().rename('SUM_90')
392. #
393. ## データフレームを結合
394. # summary = pd.concat([sum_past_30, sum_past_90], axis=1)
395. ## 総額の比を計算
396. # summary['RATIO_30_90'] = summary['SUM_30'] / summary['SUM_90']
397. ## NaN の場合は適切に処理 (0 とする、あるいは特定の値で補完するなど)
398. # summary['RATIO_30_90'].fillna(0, inplace=True)
399.
400.
401. # Percentage and difference paid in each installment (amount paid and installment value)
402. ins['PAYMENT_PERC'] = ins['AMT_PAYMENT'] / ins['AMT_INSTALLMENT']
403. ins['PAYMENT_DIFF'] = ins['AMT_INSTALLMENT'] - ins['AMT_PAYMENT']
404. # Days past due and days before due (no negative values)
405. ins['DPD'] = ins['DAYS_ENTRY_PAYMENT'] - ins['DAYS_INSTALLMENT']
406. ins['DBD'] = ins['DAYS_INSTALLMENT'] - ins['DAYS_ENTRY_PAYMENT']
407. ins['DPD'] = ins['DPD'].apply(lambda x: x if x > 0 else 0)
408. ins['DBD'] = ins['DBD'].apply(lambda x: x if x > 0 else 0)
409. # Features: Perform aggregations
410. aggregations = {
411.     'NUM_INSTALLMENT_VERSION': ['nunique'],
412.     'DPD': ['max', 'mean', 'sum'],

```

```

413.     'DBD': ['max', 'mean', 'sum'],
414.     'PAYMENT_PERC': ['max', 'mean', 'sum', 'var'],
415.     'PAYMENT_DIFF': ['max', 'mean', 'sum', 'var'],
416.     'AMT_INSTALMENT': ['max', 'mean', 'sum'],
417.     'AMT_PAYMENT': ['min', 'max', 'mean', 'sum'],
418.     'DAYS_ENTRY_PAYMENT': ['max', 'mean', 'sum']
419. }
420. for cat in cat_cols:
421.     aggregations[cat] = ['mean']
422. ins_agg = ins.groupby('SK_ID_CURR').agg(aggregations)
423. ins_agg.columns = pd.Index(['INSTAL_' + e[0] + "_" + e[1].upper() for e in ins_agg.columns.tolist()])
424. # Count installments accounts
425. ins_agg['INSTAL_COUNT'] = ins.groupby('SK_ID_CURR').size()
426.
427. ## 新しい特徴量を既存の集計データフレームに結合
428. # ins_agg = ins_agg.join(summary, how='left')
429.
430. ## メモリ解放
431. # del ins, summary
432. # gc.collect()
433.
434. del ins
435. gc.collect()
436. return ins_agg
437.
438.
439. def df_adjust_credit_card_balance(num_rows = None, nan_as_category = False):
440.     cc = pd.read_csv('data/credit_card_balance.csv', nrows = num_rows)
441.     cc, cat_cols = one_hot_encoder(cc, nan_as_category= True)
442.
443.     # 特徴量名のクリーニング
444.     cc = clean_feature_names(cc)
445.
446.     # General aggregations
447.     cc.drop(['SK_ID_PREV'], axis= 1, inplace = True)
448.
449.     # cc_agg = cc.groupby('SK_ID_CURR').agg(['min', 'max', 'mean', 'sum', 'var'])
450.     # cc_agg.columns = pd.Index(['CC_' + e[0] + "_" + e[1].upper() for e in cc_agg.columns.tolist()])

```

```

451.
452.  ## 修正
453.  ## 文字列やカテゴリデータには mean を用いる（あるいは他の適切な集計方法を選択）
454.  # aggregations = {}
455.  # for col in cc.columns:
456.  #     if cc[col].dtype == 'object':
457.  #         aggregations[col] = ['mean'] # カテゴリカルデータは mean など適切な集計を選択
458.  #     else:
459.  #         aggregations[col] = ['min', 'max', 'mean', 'sum', 'var'] # 数値データの場合はこれらの集計を行
460.  #
461.  # cc_agg = cc.groupby('SK_ID_CURR').agg(aggregations)
462.  # cc_agg.columns = pd.Index(['CC_' + '_' + e.upper() for e in cc_agg.columns.tolist()])
463.  # 集計関数をカラムのデータ型に基づいて選択
464.  aggregations = {}
465.  for col in cc.columns:
466.      if cc[col].dtype == bool:
467.          # Boolean 型データに対しては平均値（発生頻度）を計算
468.          aggregations[col] = ['mean']
469.      elif cc[col].dtype == 'object':
470.          # カテゴリカルデータ（文字列）には平均を適用する（ワンホットエンコードされた後の
471.          # 処理）
472.          aggregations[col] = ['mean'] # あるいは他のカテゴリカルデータ集計方法を選択
473.          # 数値データに対しては様々な集計を行う
474.          aggregations[col] = ['min', 'max', 'mean', 'sum', 'var']
475.
476.  # 集計実行
477.  cc_agg = cc.groupby('SK_ID_CURR').agg(aggregations)
478.  # 列名のフォーマット
479.  cc_agg.columns = pd.Index(['CC_' + '_' + e.upper() for e in cc_agg.columns.tolist()])
480.
481.
482.  # Count credit card lines
483.  cc_agg['CC_COUNT'] = cc.groupby('SK_ID_CURR').size()
484.  del cc
485.  gc.collect()
486.  return cc_agg

```

```

487.
488.
489. def kfold_lightgbm(df, num_folds):
490.     # Divide in training/validation and test data
491.     # TARGET 列が null でないものを訓練データとして、null のものをテストデータとして分割
492.     train_df = df[df['TARGET'].notnull()]
493.     test_df = df[df['TARGET'].isnull()]
494.
495.     # 訓練データとテストデータの形状を出力
496.     print("Starting LightGBM. Train shape: {}, test shape: {}".format(train_df.shape, test_df.shape))
497.
498.     # 元のデータフレームを削除してメモリを解放
499.     del df
500.     gc.collect()
501.
502.     folds = KFold(n_splits= num_folds, shuffle=True, random_state=1001)
503.
504.     # Out-of-Fold 予測とサブミッション用の予測を格納する配列を初期化
505.     # Create arrays and dataframes to store results
506.     oof_preds = np.zeros(train_df.shape[0])
507.     sub_preds = np.zeros(test_df.shape[0])
508.
509.     # モデル訓練に使用する特徴量を選択
510.     feats = [f for f in train_df.columns if f not in
               ['TARGET', 'SK_ID_CURR', 'SK_ID_BUREAU', 'SK_ID_PREV', 'index']]
511.
512.     # 各フォールドでモデルを訓練
513.     for n_fold, (train_idx, valid_idx) in enumerate(folds.split(train_df[feats], train_df['TARGET'])):
514.         train_x, train_y = train_df[feats].iloc[train_idx], train_df['TARGET'].iloc[train_idx]
515.         valid_x, valid_y = train_df[feats].iloc[valid_idx], train_df['TARGET'].iloc[valid_idx]
516.
517.         # LightGBM parameters found by Bayesian optimization
518.         # LightGBM モデルのパラメータを設定し、モデルを初期化
519.         clf = LGBMClassifier(
520.             nthread=4,
521.             n_estimators=10000,
522.             learning_rate=0.02,
523.             num_leaves=34,

```

```

524.     colsample_bytree=0.9497036,
525.     subsample=0.8715623,
526.     max_depth=8,
527.     # もしや 6 のほうが良い？
528.     # max_depth=6,
529.     reg_alpha=0.041545473,
530.     reg_lambda=0.0735294,
531.     min_split_gain=0.0222415,
532.     min_child_weight=39.3259775,
533.     silent=-1,
534.     verbose=-1, )
535.
536.
537. # モデルを構築
538. clf.fit(train_x, train_y,
539.         eval_set=[(valid_x, valid_y)],
540.         eval_metric='auc',
541.         callbacks=[lgb.early_stopping(stopping_rounds=200,
542.                                       verbose=True)]
543.         )
544.
545. # 予測値を Out-of-Fold 配列とサブミッション用配列に格納
546. oof_preds[valid_idx] = clf.predict_proba(valid_x, num_iteration=clf.best_iteration_)[:, 1]
547. sub_preds += clf.predict_proba(test_df[feats], num_iteration=clf.best_iteration_)[:, 1] / folds.n_splits
548.
549. # 各フォールドの AUC スコアを出力
550. print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_preds[valid_idx])))
551.
552. # LightGBD 組み込みの、importance 関数
553. # 特徴量名と重要度をデータフレームに格納
554. feature_importances = pd.DataFrame({
555.     'feature': clf.feature_name_,
556.     'importance': clf.feature_importances_
557. })
558. # 重要度に基づいて降順にソート
559. feature_importances = feature_importances.sort_values(by='importance', ascending=False)
560. # CSV ファイルに出力
561. feature_importances.to_csv('feature_importances.csv', index=False)

```

```

562.     print("特微量の重要度が 'feature_importances.csv' に保存されました。")
563.
564.     # plt.figure(figsize=(20, 100)) # 幅 20 インチ、高さ 10 インチに設定
565.     # # 特微量の重要度をプロット
566.     # ax = lgb.plot_importance(clf, max_num_features=50, importance_type='gain')
567.     # ax.set_title('Feature Importance')
568.     # ax.set_ylabel('Importance')
569.     # ax.set_xlabel('Features')
570.     # # グラフを画像ファイルとして保存
571.     # plt.savefig('feature_importance.png', dpi=300) # dpi は解像度を指定
572.     #
573.     # plt.show()
574.     # # SHAP 値の計算
575.     # #TreeExplainer は、決定木系のモデルの SHAP 値を取得するもの。
576.     # explainer = shap.TreeExplainer(model=clf)
577.     # shap_values = explainer(valid_x)
578.     #
579.     # # Waterfall Plot の表示
580.     # shap.plots._waterfall.waterfall_legacy(explainer.expected_value[0], shap_values[0],
feature_names=valid_x.columns)
581.     # plt.show()
582.
583.     # メモリ解放
584.     del clf, train_x, train_y, valid_x, valid_y
585.     gc.collect()
586.
587.     # AUC 計算をしている
588.     # 全体の AUC スコアを出力
589.     print('Full AUC score %.6f' % roc_auc_score(train_df['TARGET'], oof_preds))
590.
591.     test_df.loc[:, 'TARGET'] = sub_preds
592.     test_df[['SK_ID_CURR', 'TARGET']].to_csv(submission_file_name, index= False)
593.     print(test_df[['SK_ID_CURR', 'TARGET']].head())
594.
595.
596.
597.
598. if __name__ == "__main__":

```



```

599. df = df_init(nan_as_category = True)
600.
601. # bureau の結合
602. df_bureau = df_adjust_bureau_and_balance()
603. print("Bureau df shape:", df_bureau.shape)
604. df = df.join(df_bureau, how='left', on='SK_ID_CURR')
605. del df_bureau
606. gc.collect()
607.
608. # previous_applications の結合
609. prev = df_adjust_previous_applications()
610. print("Previous applications df shape:", prev.shape)
611. df = df.join(prev, how='left', on='SK_ID_CURR')
612. del prev
613. gc.collect()
614.
615. # POS-CASH balance の結合
616. pos = df_adjust_pos_cash()
617. print("Pos-cash balance df shape:", pos.shape)
618. df = df.join(pos, how='left', on='SK_ID_CURR')
619. del pos
620. gc.collect()
621.
622. # installments payments の結合
623. ins = df_adjust_installments_payments()
624. print("Installments payments df shape:", ins.shape)
625. df = df.join(ins, how='left', on='SK_ID_CURR')
626. del ins
627. gc.collect()
628.
629. # credit card balance の結合
630. cc = df_adjust_credit_card_balance()
631. print("Credit card balance df shape:", cc.shape)
632. df = df.join(cc, how='left', on='SK_ID_CURR')
633. del cc
634. gc.collect()
635.
636.

```

```
637. # print(df.sample(5))
638. # データフレームの上から数行を csv として出力
639. df_head = df.head(5)
640. df_head.to_csv('df_output.csv', index=False)
641. print("csv outputted")
642.
643.
644. # 15 とか 20 のほうが良いかも
645. kfold_lightgbm(df, num_folds = 15)
646.
647.
```