ds_project

April 25, 2024

[ ]:
```
from google.colab import drive
from scipy.stats import zscore, boxcox
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

[ ]:
```
# Define the path to the folder you want to mount
drive_path = "/content/drive"
data_path = os.path.join(drive_path, "MyDrive/dsa_project")

drive.mount(drive_path)
os.listdir(data_path)

TGT = "TARGET"
```

```
Mounted at /content/drive
```

Question 1: SK_ID_CURR Analysis:Get the count of unique values of SK_ID_CURR in file application_train.csv and compare this count to the number of rows in application_train.csv. Compare this with the total row count. Investigate if SK_ID_CURR serves as the table's primary key.?

Analysis:

Since the number of rows is same as number of unique values of SK_ID_CURR i.e. 307511, we can confirm that SK_ID_CURR is the primary id.

[ ]:
```
app_train = pd.read_csv(os.path.join(data_path, "application_train.csv"))

print(f"app_train.shape: {app_train.shape[0]} | len(app_train['SK_ID_CURR'].
 ↪unique()): {len(app_train['SK_ID_CURR'].unique())}")
```

```
app_train.shape: 307511 | len(app_train['SK_ID_CURR'].unique()): 307511
```

Question 2: TARGET Column Analysis: Identify and quantify the unique values within the TARGET column. Assess the dataset's balance by evaluating the proportions of each target value.

Analysis:

There is a clear imbalance in the Target Variables as evident by the percentage of each unique value.

[ ]:
```
print(app_train[TGT].value_counts())
print(app_train[TGT].value_counts(normalize=True))
```

```
TARGET
0    282686
1     24825
Name: count, dtype: int64
TARGET
0    0.919271
1    0.080729
Name: proportion, dtype: float64
```

[ ]:
```
all_num_cols = app_train.select_dtypes(include=['number']).columns.tolist()
```

There is an imbalance in the TARGET values with 0 being 91% of the data.

[ ]:
```
null_df = list()

for c in all_num_cols:
  _miss_count = len(app_train[app_train[c].isnull()])
  _miss_pct = round(_miss_count/len(app_train), 2) * 100

  null_df.append({"col": c, "miss": _miss_count, "miss_pct": _miss_pct})

null_df = pd.DataFrame(null_df).sort_values("miss").reset_index(drop=True)
null_df = null_df[null_df["col"].isin([c for c in null_df["col"].tolist() if␣
 ↪"FLAG" not in c.upper()])]
null_df.head(25)
```

[ ]:

	col	miss	miss_pct
0	SK_ID_CURR	0	0.0
1	REG_CITY_NOT_WORK_CITY	0	0.0
22	REG_CITY_NOT_LIVE_CITY	0	0.0
23	LIVE_REGION_NOT_WORK_REGION	0	0.0
24	LIVE_CITY_NOT_WORK_CITY	0	0.0
25	REG_REGION_NOT_LIVE_REGION	0	0.0
26	AMT_CREDIT	0	0.0
27	REGION_POPULATION_RELATIVE	0	0.0
28	DAYS_BIRTH	0	0.0
29	DAYS_EMPLOYED	0	0.0
30	DAYS_REGISTRATION	0	0.0
31	DAYS_ID_PUBLISH	0	0.0
32	REG_REGION_NOT_WORK_REGION	0	0.0
35	AMT_INCOME_TOTAL	0	0.0
39	REGION_RATING_CLIENT	0	0.0

```
40                     TARGET     0       0.0
41  REGION_RATING_CLIENT_W_CITY     0       0.0
42       HOUR_APPR_PROCESS_START     0       0.0
44                 CNT_CHILDREN     0       0.0
45        DAYS_LAST_PHONE_CHANGE     1       0.0
46               CNT_FAM_MEMBERS     2       0.0
47                  AMT_ANNUITY    12       0.0
48              AMT_GOODS_PRICE   278       0.0
49                   EXT_SOURCE_2   660       0.0
50       DEF_60_CNT_SOCIAL_CIRCLE  1021       0.0
```

[ ]:
```
selected_num_col = list()
selected_num_col.append('TARGET')
selected_num_col.append("CNT_CHILDREN")
selected_num_col.append("AMT_INCOME_TOTAL")
selected_num_col.append("AMT_CREDIT")
selected_num_col.append("AMT_ANNUITY")
selected_num_col.append("REGION_POPULATION_RELATIVE")
selected_num_col.append("DAYS_BIRTH")
selected_num_col.append("DAYS_EMPLOYED")
selected_num_col.append("DAYS_REGISTRATION")
selected_num_col.append("DAYS_ID_PUBLISH")
selected_num_col.append("HOUR_APPR_PROCESS_START")

# Not sure if these columns should be considered numerical or categorical
# selected_num_col.append("REGION_RATING_CLIENT")
# selected_num_col.append("REG_REGION_NOT_WORK_REGION")

# Too many missing values
# selected_num_col.append('YEARS_BUILD_AVG')
# selected_num_col.append('DAYS_LAST_PHONE_CHANGE')
# selected_num_col.append('AMT_REQ_CREDIT_BUREAU_YEAR')
# selected_num_col.append('OWN_CAR_AGE')
# selected_num_col.append('AMT_GOODS_PRICE')

null_df[null_df["col"].isin(set(selected_num_col))]
```

[ ]:
```
                          col  miss  miss_pct
26                 AMT_CREDIT     0       0.0
27  REGION_POPULATION_RELATIVE     0       0.0
28                 DAYS_BIRTH     0       0.0
29              DAYS_EMPLOYED     0       0.0
30          DAYS_REGISTRATION     0       0.0
31            DAYS_ID_PUBLISH     0       0.0
35           AMT_INCOME_TOTAL     0       0.0
40                     TARGET     0       0.0
42     HOUR_APPR_PROCESS_START     0       0.0
```

```
44                  CNT_CHILDREN     0       0.0
47                   AMT_ANNUITY    12       0.0
```

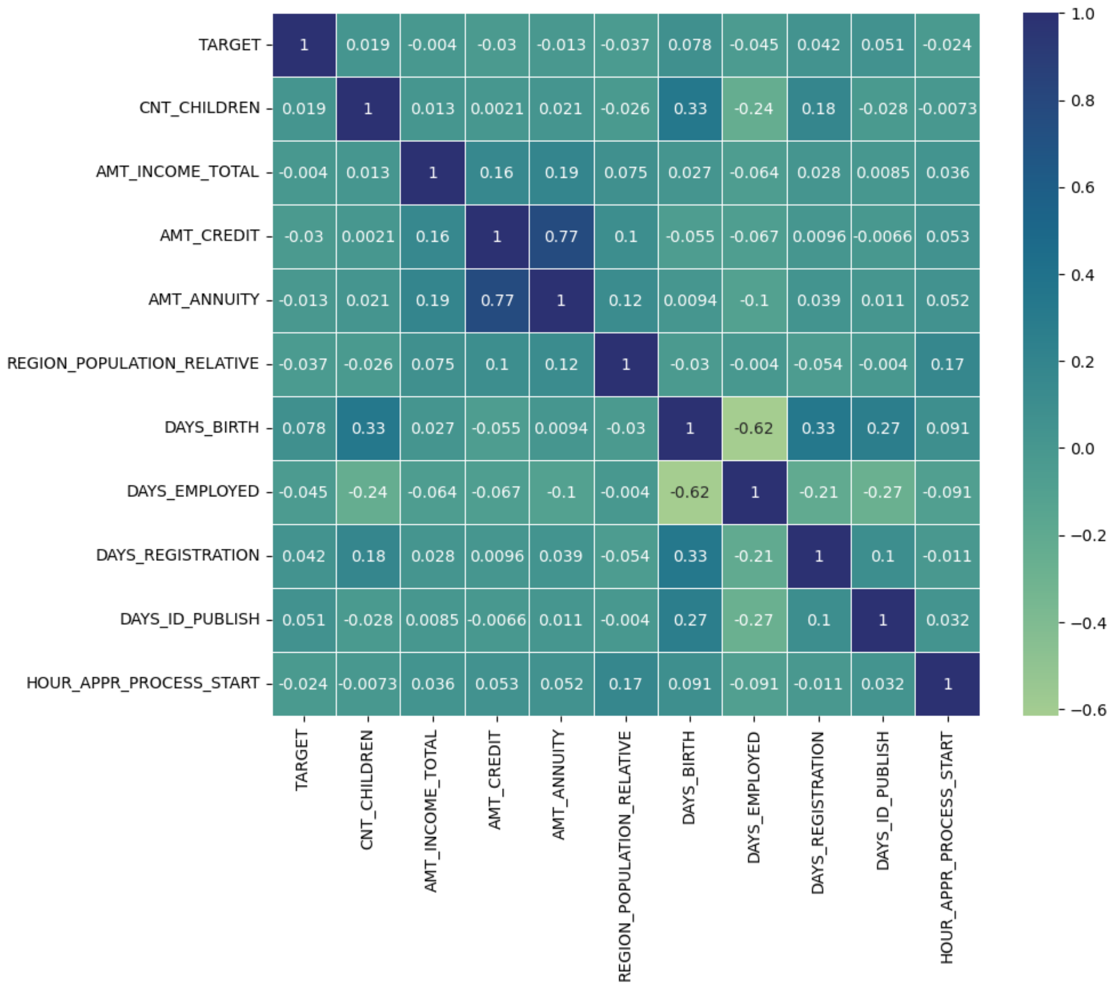
[ ]:
```
app_train[selected_num_col].dtypes
```

[ ]:
```
TARGET                           int64
CNT_CHILDREN                     int64
AMT_INCOME_TOTAL               float64
AMT_CREDIT                     float64
AMT_ANNUITY                    float64
REGION_POPULATION_RELATIVE     float64
DAYS_BIRTH                       int64
DAYS_EMPLOYED                    int64
DAYS_REGISTRATION              float64
DAYS_ID_PUBLISH                  int64
HOUR_APPR_PROCESS_START          int64
dtype: object
```

Question 3: Correlation Analysis: Generate a Pearson correlation matrix and heatmap (for any 10 numeric variables of choice) on application_tain.csv. Write code to list the top 5 features correlated with the TARGET column.

[ ]:
```
corr = app_train[selected_num_col].corr()

plt.figure(figsize = (10,8))
sns.heatmap(corr, annot = True, cmap = 'crest', linewidth = 0.5)
```

[ ]:
```
<Axes: >
```



```
[ ]: corr_vars = corr[TGT].sort_values(ascending = False)
     corr_vars = corr_vars.abs()
     corr_vars = corr_vars.reset_index().rename(columns={"index": "vars"})
     corr_vars = corr_vars[(corr_vars["vars"] != TGT)]

     top5_corr_vars = corr_vars.sort_values(TGT, ascending=False).head(5)["vars"].
      ↪tolist()
     corr_vars.sort_values(TGT, ascending=False).head(5)[["vars", "TARGET"]]
```

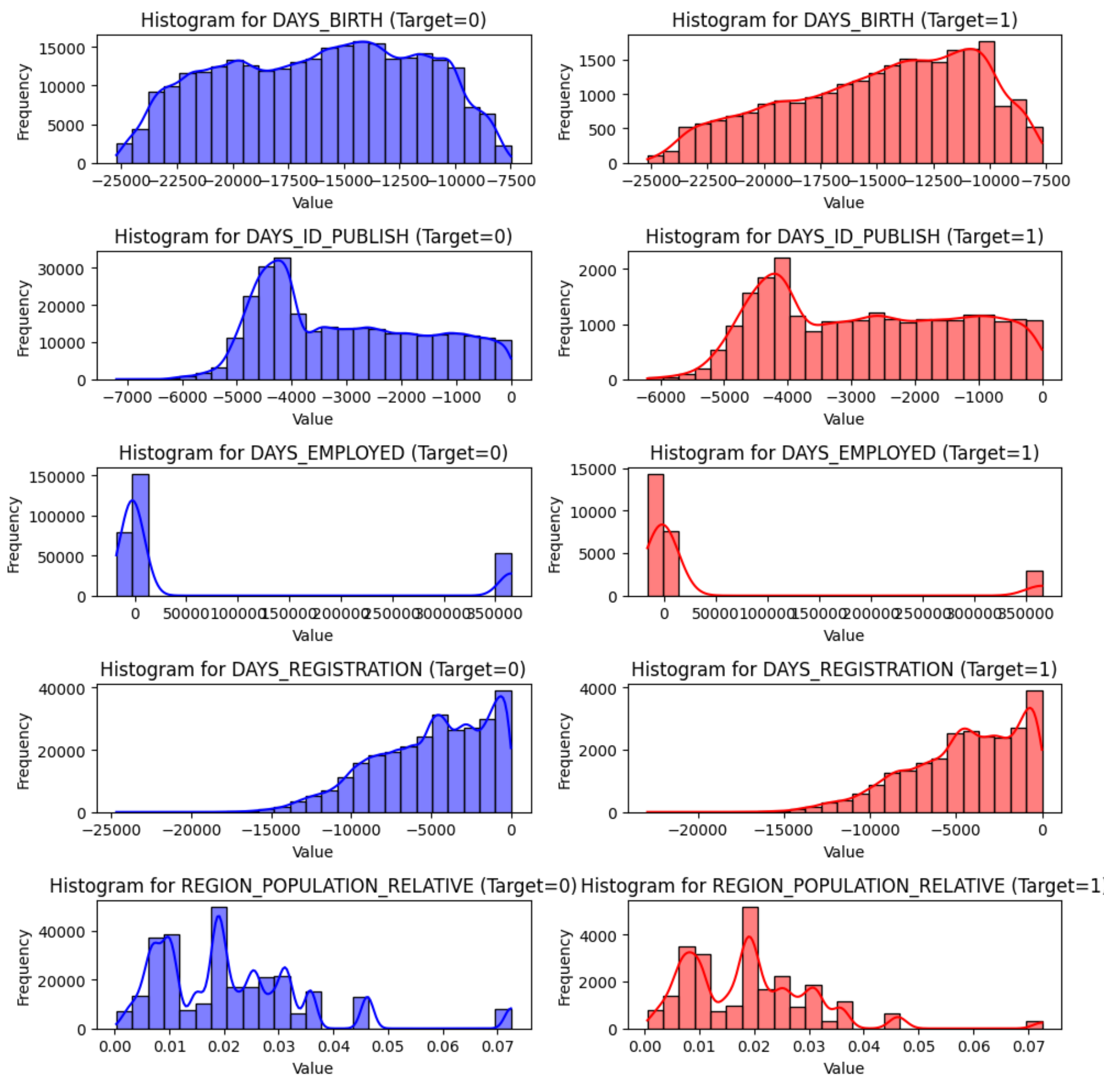
[ ]:

	vars	TARGET
1	DAYS_BIRTH	0.078239
2	DAYS_ID_PUBLISH	0.051457
10	DAYS_EMPLOYED	0.044932
3	DAYS_REGISTRATION	0.041975
9	REGION_POPULATION_RELATIVE	0.037227

Question 4: Histogram: Generate histograms for any five numerical features in applica-

tion_train.csv, and comment on whether they seem Gaussian, or have severe skews. Visualize the relationship between each of these numeric variables and the target variable.

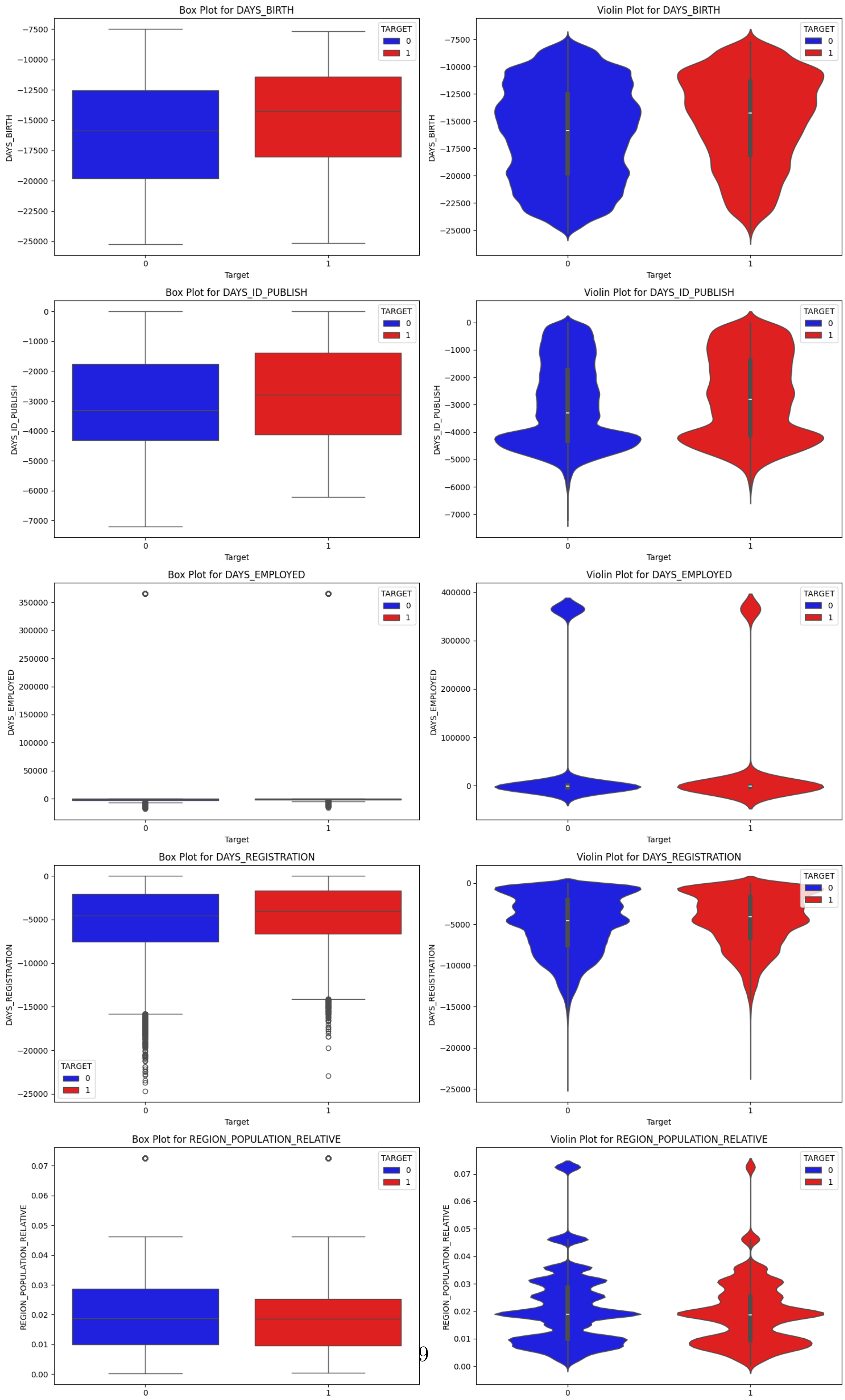
[ ]:
```
plt.figure(figsize = (10,10))

app_train0 = app_train[(app_train[TGT] == 0)]
app_train1 = app_train[(app_train[TGT] == 1)]

for i, _v in enumerate(top5_corr_vars):

    plt.subplot(len(top5_corr_vars), 2, 2*i+1)
    sns.histplot(x=app_train0[_v], bins=25, kde=True, color="blue")
    plt.title(f'Histogram for {_v} (Target=0)')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

    plt.subplot(len(top5_corr_vars), 2, 2*i+2)
    sns.histplot(x=app_train1[_v], bins=25, kde=True, color="red")
    plt.title(f'Histogram for {_v} (Target=1)')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

# Adjust layout to prevent overlapping titles
plt.tight_layout()

# Show the plots
plt.show()
```



[ ]:

```
plt.figure(figsize=(15, 5 * len(top5_corr_vars)))

for i, var in enumerate(top5_corr_vars):
    # Box plot
    plt.subplot(len(top5_corr_vars), 2, 2 * i + 1)
    sns.boxplot(x=app_train[TGT], y=app_train[var], hue=app_train[TGT], palette={0: 'blue', 1: 'red'})
    plt.title(f'Box Plot for {var}')
    plt.xlabel('Target')
    plt.ylabel(var)

    # Violin plot
    plt.subplot(len(top5_corr_vars), 2, 2 * i + 2)
```

```
    sns.violinplot(x=app_train[TGT], y=app_train[var], hue=app_train[TGT],␣
↪palette={0: 'blue', 1: 'red'})
    plt.title(f'Violin Plot for {var}')
    plt.xlabel('Target')
    plt.ylabel(var)

# Adjust layout to prevent overlapping titles
plt.tight_layout()

# Show the plots
plt.show()
```


Box Plot for DAYS_BIRTH
Violin Plot for DAYS_BIRTH
DAYS_BIRTH
Box Plot for DAYS_ID_PUBLISH
Violin Plot for DAYS_ID_PUBLISH
DAYS_ID_PUBLISH
Box Plot for DAYS_EMPLOYED
Violin Plot for DAYS_EMPLOYED
DAYS_EMPLOYED
Box Plot for DAYS_REGISTRATION
Violin Plot for DAYS_REGISTRATION
DAYS_REGISTRATION
Box Plot for REGION_POPULATION_RELATIVE
Violin Plot for REGION_POPULATION_RELATIVE
REGION_POPULATION_RELATIVE
Target
TARGET
0
1

[ ]:
```
for c in top5_corr_vars:
  _miss_count = len(app_train[app_train[c].isnull()])
  print(f"Null values in {c}: {_miss_count}, {round(_miss_count/len(app_train),␣
↪2) * 100} %")
```

```
Null values in DAYS_BIRTH: 0, 0.0 %
Null values in DAYS_ID_PUBLISH: 0, 0.0 %
Null values in DAYS_EMPLOYED: 0, 0.0 %
Null values in DAYS_REGISTRATION: 0, 0.0 %
Null values in REGION_POPULATION_RELATIVE: 0, 0.0 %
```

[ ]:
```
app_train["DAYS_EMPLOYED"].describe()
```

[ ]:
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

Question 5: Outlier Analysis: Perform outlier analysis on the chosen variables.

[ ]:
```
# Record count of rows where Z score of target column in higher than 2 and 3␣
↪separately
suff = "z_score"
z_score_cols = list()
outlier_df = list()

for c in top5_corr_vars:
  app_train[f"{c}_{suff}"] = zscore(app_train[c])
  over_2 = len(app_train[(app_train[f"{c}_{suff}"].abs() > 2)])
  over_3 = len(app_train[(app_train[f"{c}_{suff}"].abs() > 3)])
  outlier_df.append({"col": c, "over_2": over_2, "over_3": over_3})

outlier_df = pd.DataFrame(outlier_df)
outlier_df
```

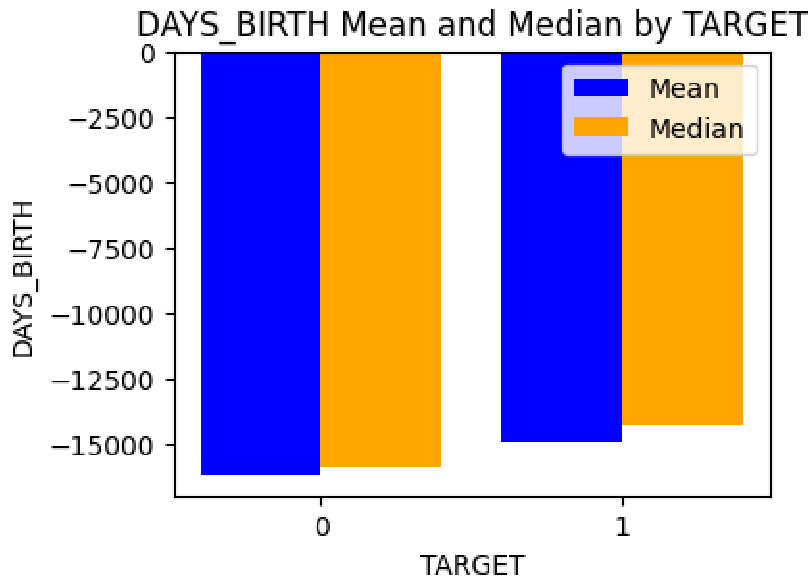
[ ]:

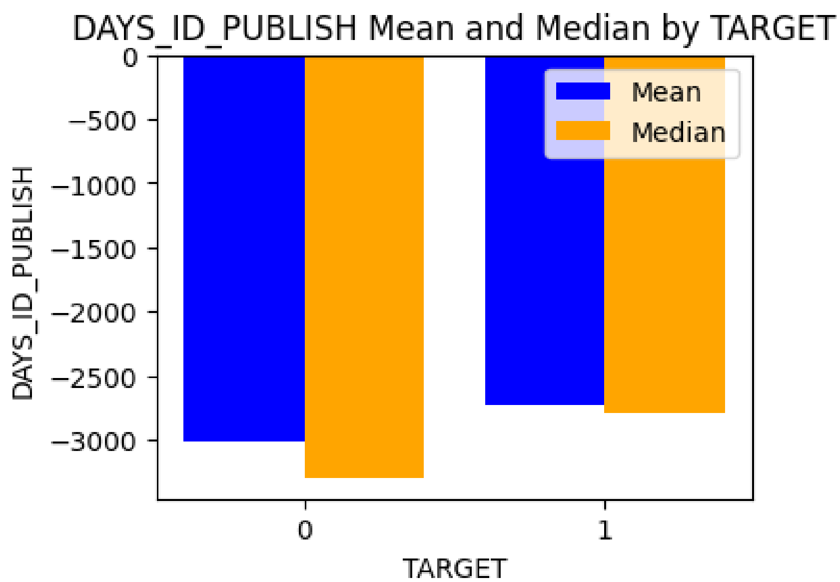
	col	over_2	over_3
0	DAYS_BIRTH	1210	0
1	DAYS_ID_PUBLISH	390	0
2	DAYS_EMPLOYED	55374	0
3	DAYS_REGISTRATION	11330	749

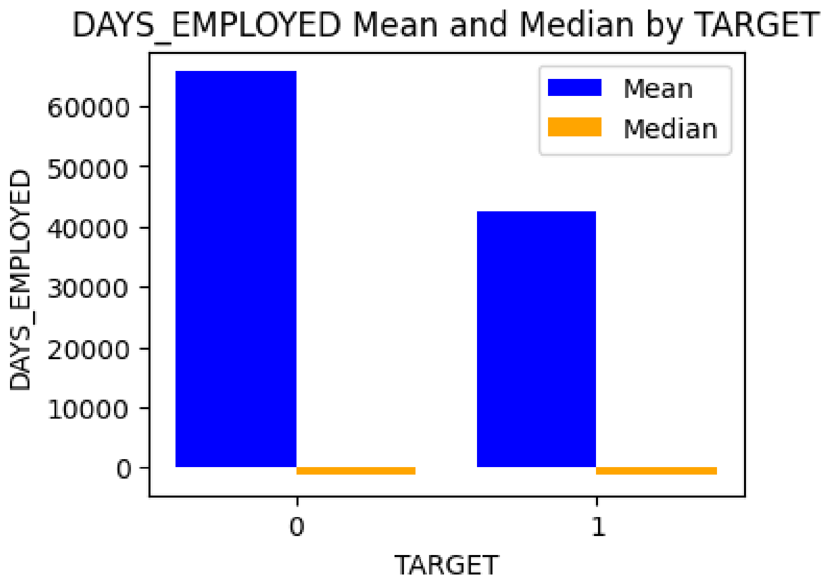
```
4  REGION_POPULATION_RELATIVE    8412    8412
```

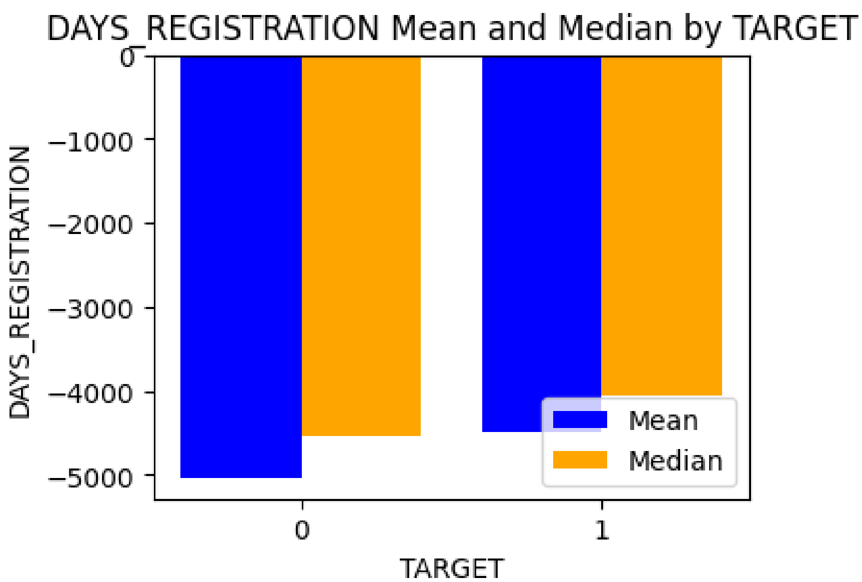
Question 6: Transformation of Nuemric Variables: If skewed, perform suitable transformations on these five numerical variables. Check the relationship of each of these numeric variables with the target variable using bar charts. Visualize the relationship between each of these numeric variables and the target variable. Perform outlier analysis on the transformed variables and report any differences before and after transformation.

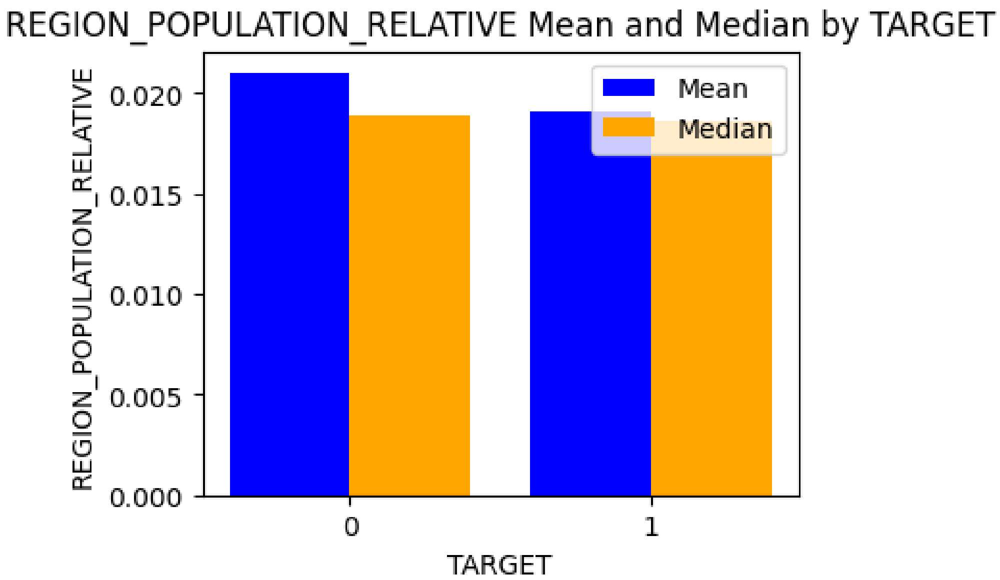
[ ]:
```
# Calculate the selected metrics for each numeric variable grouped by the␣
↪target variable
metrics = ['mean', 'median']
central_tend = {}
for metric in metrics:
    if metric == 'mean':
        t = app_train.groupby(TGT)[top5_corr_vars].mean()
    elif metric == 'median':
        t = app_train.groupby(TGT)[top5_corr_vars].median()
    elif metric == 'std':
        t = app_train.groupby(TGT)[top5_corr_vars].std()
    elif metric == 'count':
        t = app_train.groupby(TGT)[top5_corr_vars].count()
    central_tend[metric] = t

# Merge the DataFrames on 'TARGET' column
merged_df = pd.merge(central_tend["mean"],
                     central_tend["median"],\
                     left_index=True, right_index=True,
                     suffixes=('_mean', '_median'))

merged_df = merged_df.reset_index()

# Columns to plot
columns_to_plot = list(merged_df.columns)[1:]

# Plot bar graphs
for column in top5_corr_vars:
    plt.figure(figsize=(4, 3))
    plt.bar(merged_df['TARGET'], merged_df[column + '_mean'], label='Mean',␣
↪width=0.4, color='blue')
    plt.bar(merged_df['TARGET'] + 0.4, merged_df[column + '_median'],␣
↪label='Median', width=0.4, color='orange')
    plt.xlabel('TARGET')
    plt.ylabel(column)
    plt.title(f'{column} Mean and Median by TARGET')
    plt.xticks(merged_df['TARGET'] + 0.2, merged_df['TARGET'])
    plt.legend()
    plt.show()
```


DAYS_BIRTH Mean and Median by TARGET
0
−2500
−5000
−7500
−10000
−12500
−15000
DAYS_BIRTH
Mean
Median
0
1
TARGET


DAYS_ID_PUBLISH Mean and Median by TARGET
0
−500
−1000
−1500
−2000
−2500
−3000
DAYS_ID_PUBLISH
Mean
Median
0
1
TARGET


DAYS_EMPLOYED Mean and Median by TARGET
Mean
Median
60000
50000
40000
30000
20000
10000
0
DAYS_EMPLOYED
0
1
TARGET

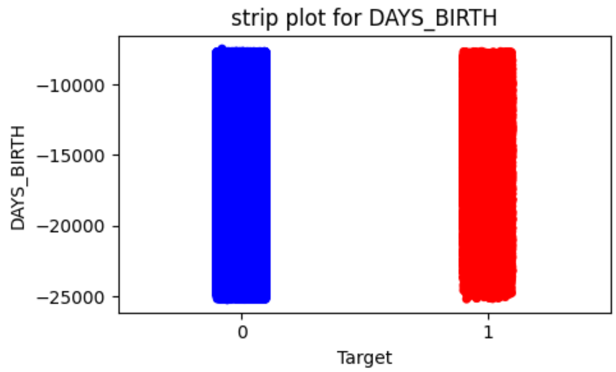

DAYS_REGISTRATION Mean and Median by TARGET
0
−1000
−2000
−3000
−4000
−5000
DAYS_REGISTRATION
Mean
Median
0
1
TARGET

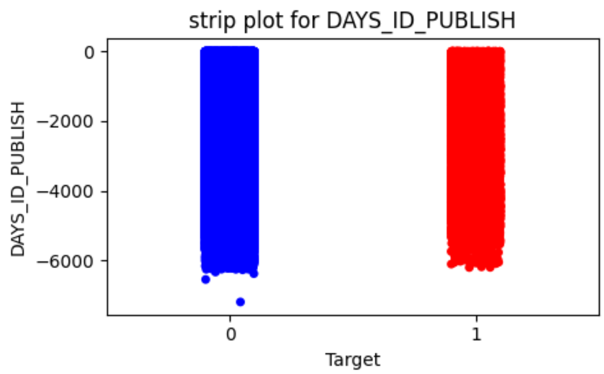



```
[ ]: import matplotlib.pyplot as plt
import seaborn as sns
unique_tgt_values = app_train[TGT].unique()
palette = {value: 'blue' if value == 0 else 'red' for value in␣
 ↪unique_tgt_values}
plt.figure(figsize=(5, 15))

for i, var in enumerate(top5_corr_vars):
    plt.subplot(len(top5_corr_vars), 1, i + 1)
    sns.stripplot(x=app_train[TGT], y=app_train[var], palette = {'0':'blue','1':
 ↪'red'}, jitter = True)
    plt.title(f'strip plot for {var}')
    plt.xlabel('Target')
    plt.ylabel(var)

# Adjust layout to prevent overlapping titles
plt.tight_layout()

# Show the plots
plt.show()
```

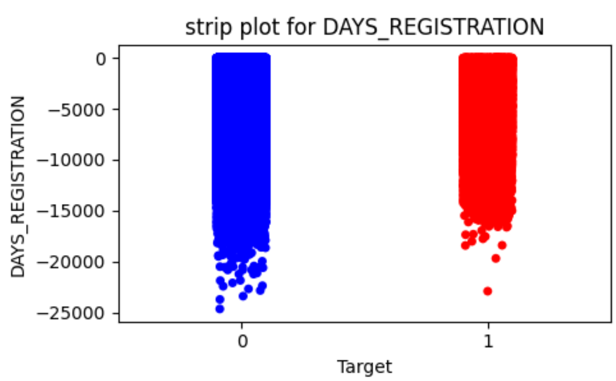
```
<ipython-input-29-23f86dad648a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

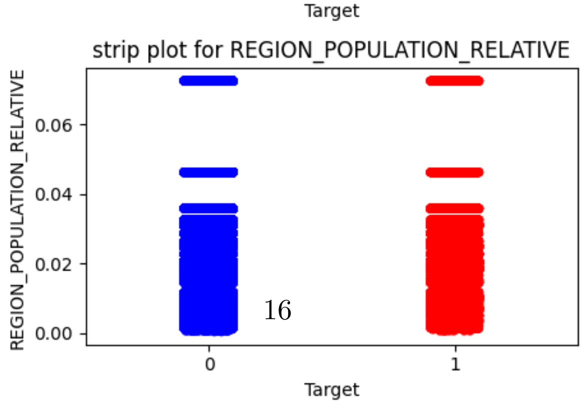
```
  sns.stripplot(x=app_train[TGT], y=app_train[var], palette =
{'0':'blue','1':'red'}, jitter = True)
<ipython-input-29-23f86dad648a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.stripplot(x=app_train[TGT], y=app_train[var], palette =
{'0':'blue','1':'red'}, jitter = True)
<ipython-input-29-23f86dad648a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.stripplot(x=app_train[TGT], y=app_train[var], palette =
{'0':'blue','1':'red'}, jitter = True)
<ipython-input-29-23f86dad648a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.stripplot(x=app_train[TGT], y=app_train[var], palette =
{'0':'blue','1':'red'}, jitter = True)
<ipython-input-29-23f86dad648a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.stripplot(x=app_train[TGT], y=app_train[var], palette =
{'0':'blue','1':'red'}, jitter = True)
```


strip plot for DAYS_BIRTH
DAYS_BIRTH
−10000
−15000
−20000
−25000
0
1
Target


strip plot for DAYS_ID_PUBLISH
DAYS_ID_PUBLISH
0
−2000
−4000
−6000
0
1
Target


strip plot for DAYS_EMPLOYED
DAYS_EMPLOYED
300000
200000
100000
0
0
1
Target


strip plot for DAYS_REGISTRATION
DAYS_REGISTRATION
0
−5000
−10000
−15000
−20000
−25000
0
1
Target


strip plot for REGION_POPULATION_RELATIVE
REGION_POPULATION_RELATIVE
0.06
0.04
0.02
0.00
0
1
Target

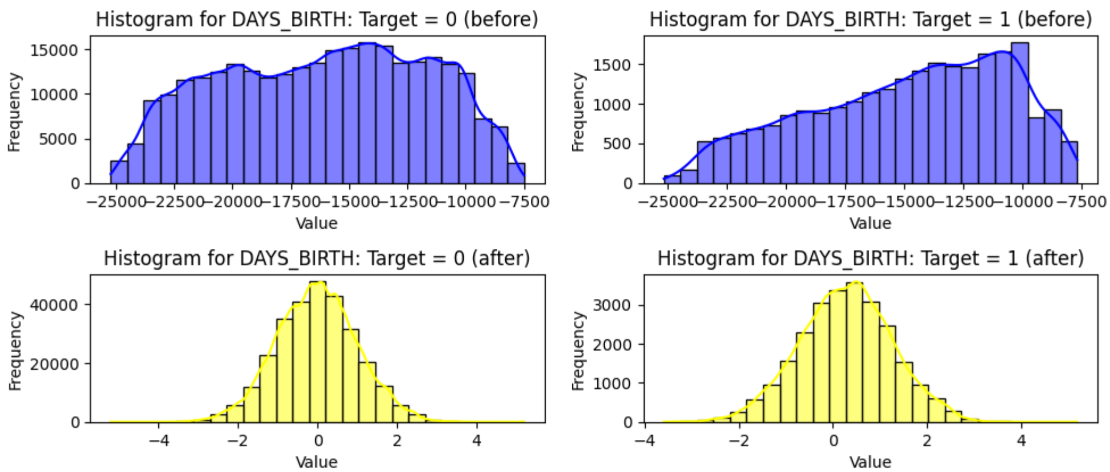
[ ]:

```python
from sklearn.preprocessing import QuantileTransformer

# Transformation for DAYS_REGISTRATION
# Increasing - Positively skewed
t_map = dict()
t_map["DAYS_ID_PUBLISH"] = "quantile"
t_map["DAYS_REGISTRATION"] = "quantile"
t_map["DAYS_BIRTH"] = "quantile"
t_map["DAYS_EMPLOYED"] = "quantile"
t_map["REGION_POPULATION_RELATIVE"] = "quantile"

offset_map = dict()
offset_map["DAYS_REGISTRATION"] = 50000
offset_map["DAYS_ID_PUBLISH"] = "min"
offset_map["DAYS_BIRTH"] = "min"
offset_map["DAYS_EMPLOYED"] = "min"

def transform_var(df, var, t_type):
  if t_type == "log":
    df[var] = df[var].apply(lambda x: np.log(x + 1))
  elif t_type == "sqrt":
    df[var] = df[var].apply(lambda x: np.sqrt(x + 1))
  elif t_type == "cube_root":
    df[var] = df[var].apply(lambda x: x**(1/3))
  elif t_type == "exp":
    df[var] = df[var].apply(lambda x: np.exp(x))
  elif t_type == "inv":
    df[var] = df[var].apply(lambda x: 1/x)
  elif t_type == "logit":
    df[var] = df[var].apply(lambda x: np.log(x/(1+x)))
  elif t_type == "boxcox":
    df[var], _ = boxcox(df[var])
  elif t_type == "quantile":
    df[var] = QuantileTransformer(output_distribution='normal').
↪fit_transform(np.array(df[var]).reshape(-1, 1))
  return df

def plot_histogram(df, col, col_i, all_cols, sp_i, sp_j, tgt_val, suffix,␣
↪color="blue"):
  plt.subplot(len(all_cols), sp_i, sp_j)
  sns.histplot(x=df[col], bins=25, kde=True, color=color)
  plt.title(f'Histogram for {col}: Target = {tgt_val} ({suffix})')
  plt.xlabel('Value')
  plt.ylabel('Frequency')
```

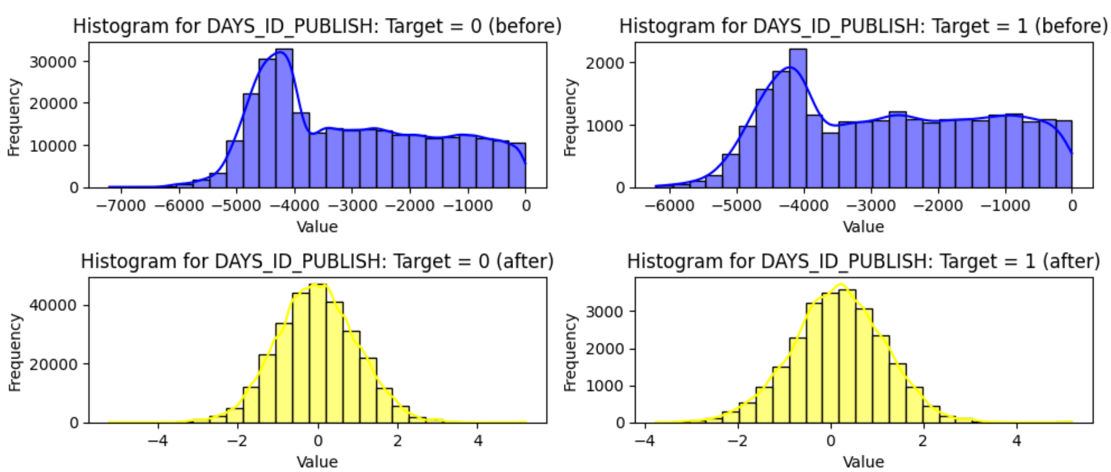
```python
def transform_var_and_plot(var, df, all_vars):

    i = 1
    plt.figure(figsize = (10,10))

    print(f"Plotting: {var}")

    if var in t_map:

      t_type = t_map[var]

      # ================================
      # Plot before transformation
      # ================================
      plot_histogram(df=df[(df[TGT] == 0)], col_i=i, col=var, suffix="before",
                     sp_i=2, sp_j=2*i+1, all_cols=all_vars, tgt_val="0")
      plot_histogram(df=app_train[(app_train[TGT] == 1)], col_i=i, col=var,
                     sp_i=2, sp_j=2*i+2, all_cols=all_vars, tgt_val="1",␣
↪suffix="before")

      # ================================
      # Transform data
      # ================================
      if var in offset_map:
        if str(offset_map[var]) == "min":
          df[var] = df[var] + abs(df[var].min())
        else:
          df[var] = df[var] + offset_map[var]

      df = transform_var(df=df, var=var, t_type=t_type)

      # Increment the value of i
      i += 1

    else:
      print(f"No Transformation required")

    # ================================
    # Plot after transformation
    # ================================
    plot_histogram(df=df[(df[TGT] == 0)], col_i=i, col=var,
                   sp_i=2, sp_j=2*i+1, all_cols=all_vars, tgt_val="0",
                   color="yellow", suffix="after")
    plot_histogram(df=df[(df[TGT] == 1)], col_i=i, col=var,
                   sp_i=2, sp_j=2*i+2, all_cols=all_vars, tgt_val="1",
```

```
                  color="yellow", suffix="after")

    # Increment the value of i
    i += 1

    # Adjust layout to prevent overlapping titles
    plt.tight_layout()

    # Show the plots
    plt.show()

for var in top5_corr_vars:
  transform_var_and_plot(var=var, all_vars=top5_corr_vars, df=app_train)
```

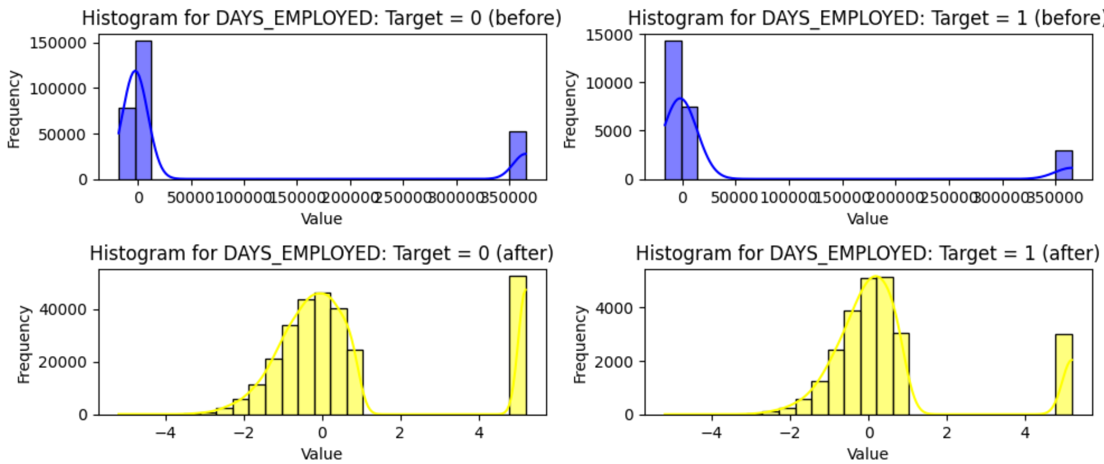
```
Plotting: DAYS_BIRTH
```



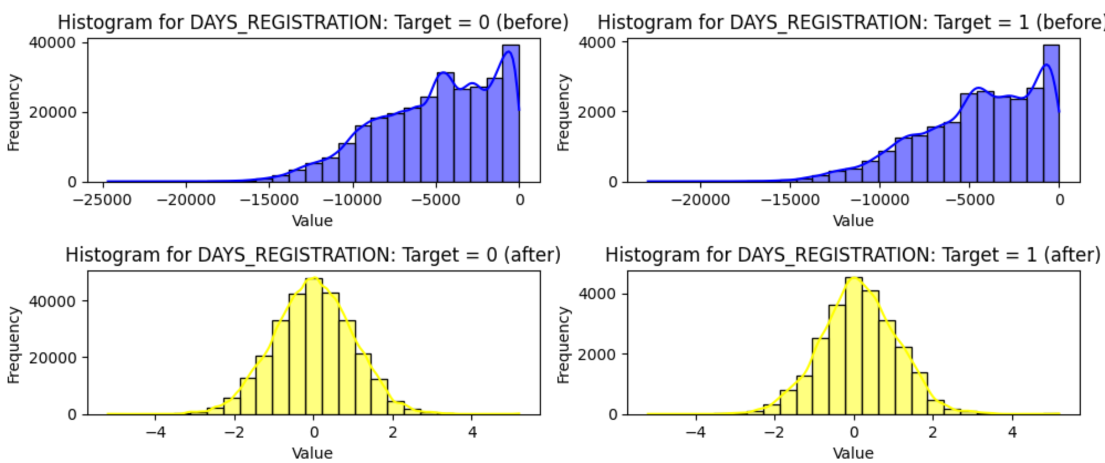

```
Plotting: DAYS_ID_PUBLISH
```



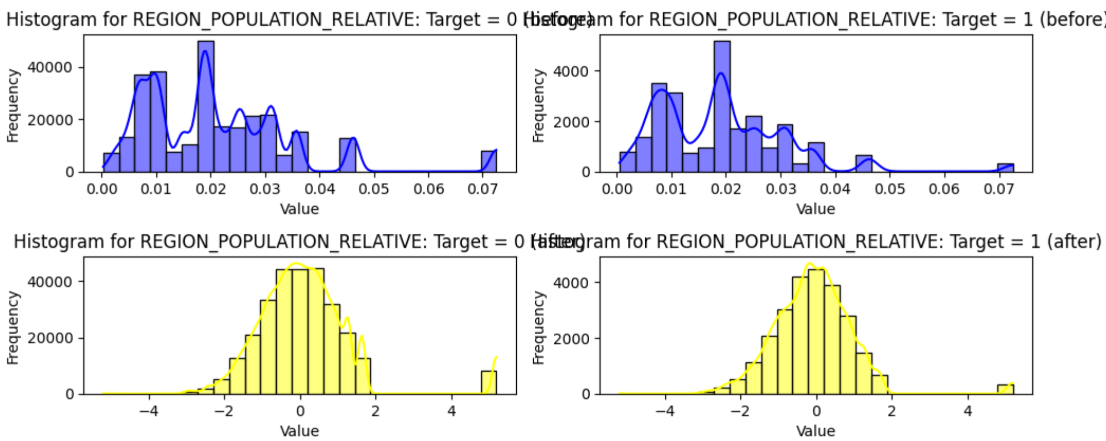
`Plotting: DAYS_EMPLOYED`



`Plotting: DAYS_REGISTRATION`



`Plotting: REGION_POPULATION_RELATIVE`



[ ]:
```
# Running outlier analysis post transformation again
suff = "z_score"
z_score_cols = list()
outlier_df = list()

for c in top5_corr_vars:
  app_train[f"{c}_{suff}"] = zscore(app_train[c])
  over_2 = len(app_train[(app_train[f"{c}_{suff}"].abs() > 2)])
  over_3 = len(app_train[(app_train[f"{c}_{suff}"].abs() > 3)])
  outlier_df.append({"col": c, "over_2": over_2, "over_3": over_3})

outlier_df = pd.DataFrame(outlier_df)
outlier_df
```

[ ]:

	col	over_2	over_3
0	DAYS_BIRTH	13708	770
1	DAYS_ID_PUBLISH	14253	801
2	DAYS_EMPLOYED	55386	0
3	DAYS_REGISTRATION	13929	995
4	REGION_POPULATION_RELATIVE	11383	8453

[ ]:
```
app_train.select_dtypes(include=['object']).columns
```

[ ]:
```
Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE',
       'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE',
       'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE'],
      dtype='object')
```

Question 7: Categorical Features: Check cardinality and rare values of at least five categorical features. Discuss whether each of them is ordinal or nominal. Discuss the suitable methods for encoding each of them.

[ ]:
```
cat_var = list()
cat_var.append("NAME_CONTRACT_TYPE")
cat_var.append("CODE_GENDER")
cat_var.append("ORGANIZATION_TYPE")
cat_var.append("WEEKDAY_APPR_PROCESS_START")
cat_var.append("EMERGENCYSTATE_MODE")

card = list()
card_pct = list()

for v in cat_var:
    u_count = len(app_train[v].unique())
    cat_pct = app_train[column].value_counts(normalize=True) * 100

    cat_pct = pd.DataFrame(app_train[v].value_counts(normalize=True) * 100)
    cat_pct = cat_pct.reset_index().rename(columns={"index": "val", v: "pct"})
    cat_pct["var"] = v

    card_pct.append(cat_pct)
    card.append({"var": v, "cardinality": u_count})

card = pd.DataFrame(card)
card_pct = pd.concat(card_pct, ignore_index=True).reset_index(drop=True)
card_pct.head()
```

[ ]:

	val	pct	var
0	Cash loans	90.478715	NAME_CONTRACT_TYPE
1	Revolving loans	9.521285	NAME_CONTRACT_TYPE
2	F	65.834393	CODE_GENDER
3	M	34.164306	CODE_GENDER
4	XNA	0.001301	CODE_GENDER

[ ]:
```
card
```

[ ]:

	var	cardinality
0	NAME_CONTRACT_TYPE	2
1	CODE_GENDER	3
2	ORGANIZATION_TYPE	58
3	WEEKDAY_APPR_PROCESS_START	7
4	EMERGENCYSTATE_MODE	3

[ ]:
```
# All values which have percentages less than 5
# Our threshold for categorization intorarity is 5 percent
rare_df = card_pct[card_pct["pct"] < 5]
rare_df.head()
```

[ ]:

	val	pct	var
4	XNA	0.001301	CODE_GENDER
9	Medicine	3.639870	ORGANIZATION_TYPE
10	Business Entity Type 2	3.431747	ORGANIZATION_TYPE
11	Government	3.383294	ORGANIZATION_TYPE
12	School	2.891929	ORGANIZATION_TYPE
13	Trade: type 7	2.546576	ORGANIZATION_TYPE
14	Kindergarten	2.237318	ORGANIZATION_TYPE
15	Construction	2.185613	ORGANIZATION_TYPE
16	Business Entity Type 1	1.945947	ORGANIZATION_TYPE
17	Transport: type 4	1.755384	ORGANIZATION_TYPE
18	Trade: type 3	1.135569	ORGANIZATION_TYPE
19	Industry: type 9	1.095245	ORGANIZATION_TYPE
20	Industry: type 3	1.065978	ORGANIZATION_TYPE
21	Security	1.055897	ORGANIZATION_TYPE
22	Housing	0.961917	ORGANIZATION_TYPE
23	Industry: type 11	0.879318	ORGANIZATION_TYPE
24	Military	0.856555	ORGANIZATION_TYPE
25	Bank	0.815255	ORGANIZATION_TYPE
26	Agriculture	0.798020	ORGANIZATION_TYPE
27	Police	0.761274	ORGANIZATION_TYPE
28	Transport: type 2	0.716722	ORGANIZATION_TYPE
29	Postal	0.701438	ORGANIZATION_TYPE
30	Security Ministries	0.641928	ORGANIZATION_TYPE
31	Trade: type 2	0.617864	ORGANIZATION_TYPE
32	Restaurant	0.588922	ORGANIZATION_TYPE
33	Services	0.512177	ORGANIZATION_TYPE
34	University	0.431529	ORGANIZATION_TYPE
35	Industry: type 7	0.425025	ORGANIZATION_TYPE
36	Transport: type 3	0.386002	ORGANIZATION_TYPE
37	Industry: type 1	0.337874	ORGANIZATION_TYPE
38	Hotel	0.314135	ORGANIZATION_TYPE
39	Electricity	0.308932	ORGANIZATION_TYPE
40	Industry: type 4	0.285193	ORGANIZATION_TYPE
41	Trade: type 6	0.205196	ORGANIZATION_TYPE
42	Industry: type 5	0.194790	ORGANIZATION_TYPE
43	Insurance	0.194139	ORGANIZATION_TYPE
44	Telecom	0.187636	ORGANIZATION_TYPE
45	Emergency	0.182107	ORGANIZATION_TYPE
46	Industry: type 2	0.148938	ORGANIZATION_TYPE
47	Advertising	0.139507	ORGANIZATION_TYPE
48	Realtor	0.128776	ORGANIZATION_TYPE
49	Culture	0.123248	ORGANIZATION_TYPE
50	Industry: type 12	0.119996	ORGANIZATION_TYPE
51	Trade: type 1	0.113167	ORGANIZATION_TYPE
52	Mobile	0.103086	ORGANIZATION_TYPE
53	Legal Services	0.099183	ORGANIZATION_TYPE

```
54              Cleaning  0.084550     ORGANIZATION_TYPE
55       Transport: type 1  0.065364     ORGANIZATION_TYPE
56        Industry: type 6  0.036421     ORGANIZATION_TYPE
57       Industry: type 10  0.035446     ORGANIZATION_TYPE
58              Religion  0.027641     ORGANIZATION_TYPE
59       Industry: type 13  0.021788     ORGANIZATION_TYPE
60           Trade: type 4  0.020812     ORGANIZATION_TYPE
61           Trade: type 5  0.015934     ORGANIZATION_TYPE
62        Industry: type 8  0.007805     ORGANIZATION_TYPE
71                   Yes  1.439205  EMERGENCYSTATE_MODE
```

Categorical Variables: 1. NAME_CONTRACT_TYPE: Nominal, One-Hot Encoding 2. CODE_GENDER: Nominal, One-Hot Encoding 3. ORGANIZATION_TYPE: Nominal, Target Encoding 4. WEEKDAY_APPR_PROCESS_START: Nominal, One-Hot Encoding 5. EMERGENCYSTATE_MODE: Nominal, Binary Encoding

Question 8: Feature Engineering: Utilize previous_application.csv to compute and integrate the count of previous applications per SK_ID_CURR into application_train.csv. Further, create at least five new features from additional files, justifying their selection and aggregation method.

[ ]:
```
prev_app = pd.read_csv(os.path.join(data_path, "previous_application.csv"))
bureau = pd.read_csv(os.path.join(data_path, "bureau.csv"))
bureau_bal = pd.read_csv(os.path.join(data_path, "bureau_balance.csv"))
cc_bal = pd.read_csv(os.path.join(data_path, "credit_card_balance.csv"))
```

[ ]:
```
# Integrate count of previous applications
app_train = app_train.merge(prev_app.groupby('SK_ID_CURR').size().
 ↪reset_index(name='PREV_APP_COUNT'), on='SK_ID_CURR', how='left')
```

[ ]:
```
# Feature 1: AVG_AMT_BALANCE
# Represents the average balance maintained by the client across all credit␣
 ↪cards. It's a direct indicator of the client's financial health and their␣
 ↪ability to maintain a balance.
# Aggregation Method: Calculate the mean of AMT_BALANCE for each SK_ID_CURR␣
 ↪across all records.
f1 = cc_bal.groupby('SK_ID_CURR')['AMT_BALANCE'].mean().
 ↪reset_index(name='AVG_AMT_BALANCE')

# Feature 2: MAX_AMT_CREDIT_LIMIT_ACTUAL
# Shows the maximum credit limit that has been granted to the client on any of␣
 ↪their credit cards, indicating the maximum level of trust a credit␣
 ↪institution has in them.
# Aggregation Method: Find the maximum of AMT_CREDIT_LIMIT_ACTUAL for each␣
 ↪SK_ID_CURR.
f2 = cc_bal.groupby('SK_ID_CURR')['AMT_CREDIT_LIMIT_ACTUAL'].max().
 ↪reset_index(name='MAX_AMT_CREDIT_LIMIT_ACTUAL')
```

```
# Feature 3: TOTAL_ACTIVE_CREDITS
# The total number of active credits as reported by the bureau can indicate the
↪current credit commitments of the client, providing insights into their debt
↪levels.
# Aggregation Method: Count the number of rows where CREDIT_ACTIVE equals
↪"Active" for each SK_ID_CURR.
f3 = bureau[bureau['CREDIT_ACTIVE'] == 'Active'].groupby('SK_ID_CURR').size().
↪reset_index(name='TOTAL_ACTIVE_CREDITS')

# Feature 4: AVG_DAYS_CREDIT
# Reflects the average number of days since each credit was reported to the
↪bureau, indicating the age of the client's credit history.
# Aggregation Method: Calculate the mean of DAYS_CREDIT (considering the
↪absolute value to reflect the age) for each SK_ID_CURR.
f4 = bureau.groupby('SK_ID_CURR')['DAYS_CREDIT'].mean().
↪reset_index(name='AVG_DAYS_CREDIT').abs()

# Feature 5: AVG_CREDIT_UTILIZATION
# Credit utilization ratio is a crucial factor in credit scoring models,
↪reflecting the amount of credit the client uses relative to their credit
↪limit. Lower utilization rates are generally seen as indicators of good
↪credit management and financial health, as they suggest the client is not
↪overly reliant on credit. This feature calculates the average credit
↪utilization ratio across all the client's credit card records.
# Calculate the credit utilization ratio for each record in cc_bal by dividing
↪AMT_BALANCE by AMT_CREDIT_LIMIT_ACTUAL (taking care to handle division by
↪zero).
# Compute the average of these ratios for each SK_ID_CURR to get their average
↪credit utilization.
cc_bal['CREDIT_UTILIZATION'] = cc_bal.apply(lambda x: x['AMT_BALANCE'] /
↪x['AMT_CREDIT_LIMIT_ACTUAL'] if x['AMT_CREDIT_LIMIT_ACTUAL'] > 0 else 0,
↪axis=1)
f5 = cc_bal.groupby('SK_ID_CURR')['CREDIT_UTILIZATION'].mean().
↪reset_index(name='AVG_CREDIT_UTILIZATION')
```

[ ]:
```
# Merge all features
for f in [f1, f2, f3, f4, f5]:
  app_train = app_train.merge(f, on='SK_ID_CURR', how='left')
app_train.head()
```

[ ]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0      100002       1         Cash loans           M            N
1      100003       0         Cash loans           F            N
2      100004       0    Revolving loans           M            Y
3      100006       0         Cash loans           F            N
4      100007       0         Cash loans           M            N
```

```
  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
0               Y             0          202500.0    406597.5      24700.5
1               N             0          270000.0   1293502.5      35698.5
2               Y             0           67500.0    135000.0       6750.0
3               Y             0          135000.0    312682.5      29686.5
4               Y             0          121500.0    513000.0      21865.5

   …  DAYS_ID_PUBLISH_z_score DAYS_EMPLOYED_z_score  \
0  …                 0.487781             -0.123150
1  …                 1.754771             -0.292100
2  …                 0.316508              0.030834
3  …                 0.349662             -0.646246
4  …                -0.094446             -0.646097

  DAYS_REGISTRATION_z_score REGION_POPULATION_RELATIVE_z_score PREV_APP_COUNT  \
0                  0.227714                          -0.111551            1.0
1                  0.957197                          -1.595084            3.0
2                  0.065299                          -0.574065            1.0
3                 -1.237752                          -0.832670            9.0
4                  0.051314                           0.503268            6.0

  AVG_AMT_BALANCE  MAX_AMT_CREDIT_LIMIT_ACTUAL  TOTAL_ACTIVE_CREDITS  \
0             NaN                          NaN                   2.0
1             NaN                          NaN                   1.0
2             NaN                          NaN                   NaN
3             0.0                     270000.0                   NaN
4             NaN                          NaN                   NaN

   AVG_DAYS_CREDIT  AVG_CREDIT_UTILIZATION
0           874.00                     NaN
1          1400.75                     NaN
2           867.00                     NaN
3              NaN                     0.0
4          1149.00                     NaN

[5 rows x 133 columns]
```

Question 9: NaN Handling. Document your strategy for managing NaN values, providing rationale for your chosen approach.

[ ]:
```
# We will only consider numerical, categorical and new features for checking␣
↪null values and deciding the strategy
# to handle null appropriately. Same strategies can be extended to handle␣
↪similar columns.

num_vars = list()
```

```
num_vars.append("DAYS_ID_PUBLISH")
num_vars.append("DAYS_REGISTRATION")
num_vars.append("DAYS_BIRTH")
num_vars.append("DAYS_EMPLOYED")
num_vars.append("REGION_POPULATION_RELATIVE")

extra_features = list()
extra_features.append("AVG_AMT_BALANCE")
extra_features.append("MAX_AMT_CREDIT_LIMIT_ACTUAL")
extra_features.append("TOTAL_ACTIVE_CREDITS")
extra_features.append("AVG_DAYS_CREDIT")
extra_features.append("AVG_CREDIT_UTILIZATION")

all_req_cols = cat_var + num_vars + extra_features
all_req_cols
```

[ ]:
```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'ORGANIZATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'EMERGENCYSTATE_MODE',
 'DAYS_ID_PUBLISH',
 'DAYS_REGISTRATION',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'REGION_POPULATION_RELATIVE',
 'AVG_AMT_BALANCE',
 'MAX_AMT_CREDIT_LIMIT_ACTUAL',
 'TOTAL_ACTIVE_CREDITS',
 'AVG_DAYS_CREDIT',
 'AVG_CREDIT_UTILIZATION']
```

[ ]:
```
# Percentage of Missing Values
miss_df = app_train[all_req_cols].isnull().sum().sort_values(ascending=False)/
 ↪len(app_train)
miss_df = miss_df[miss_df > 0]
miss_df
```

[ ]:
```
AVG_AMT_BALANCE                0.717392
MAX_AMT_CREDIT_LIMIT_ACTUAL    0.717392
AVG_CREDIT_UTILIZATION         0.717392
EMERGENCYSTATE_MODE            0.473983
TOTAL_ACTIVE_CREDITS           0.293846
AVG_DAYS_CREDIT                0.143149
dtype: float64
```

```
[ ]: # Checking unique values in columns to decide how to handle NULLS
     # This is the only categorical column to be considered
     app_train["EMERGENCYSTATE_MODE"].unique()
```

```
[ ]: array(['No', nan, 'Yes'], dtype=object)
```

```
[ ]: # AVG_AMT_BALANCE
     # MAX_AMT_CREDIT_LIMIT_ACTUAL
     # AVG_DAYS_CREDIT
     for c in ["AVG_AMT_BALANCE", "MAX_AMT_CREDIT_LIMIT_ACTUAL", "AVG_DAYS_CREDIT"]:
       num_rows = len(cc_bal[cc_bal["SK_ID_CURR"].isin(list(app_train[app_train[c].
      ↪isnull()]["SK_ID_CURR"]))])
       print(f"Number of rows {c}: {str(num_rows)}")
```

```
Number of rows AVG_AMT_BALANCE: 0
Number of rows MAX_AMT_CREDIT_LIMIT_ACTUAL: 0
Number of rows AVG_DAYS_CREDIT: 375463
```

```
[ ]: # AVG_CREDIT_UTILIZATION
     # TOTAL_ACTIVE_CREDITS
     for c in ["AVG_CREDIT_UTILIZATION", "TOTAL_ACTIVE_CREDITS"]:
       num_rows = len(bureau[bureau["SK_ID_CURR"].isin(list(app_train[app_train[c].
      ↪isnull()]["SK_ID_CURR"]))])
       print(f"Number of rows {c}: {str(num_rows)}")
```

```
Number of rows AVG_CREDIT_UTILIZATION: 1014675
Number of rows TOTAL_ACTIVE_CREDITS: 119568
```

Handling Null Values

Categorical variables

- EMERGENCYSTATE_MODE: This is a categorical variable with only 2 unique values "Yes" and "No". For the rows with missing values, we can fill the column with an "Unknown" string.

Numerical variables

All the numerical fields are derived fields (type numerical) and the missing values are indicative of this data not being available (i.e. no history available for those customers). We can fill it with zeroes since it would indicate zero values for maintained by the client:

- AVG_CREDIT_UTILIZATION - Indicate 0 credit utilization.
- TOTAL_ACTIVE_CREDITS - Total active credits 0.
- AVG_AMT_BALANCE - 0 average amount balance.
- MAX_AMT_CREDIT_LIMIT_ACTUAL - 0 credit limit.
- AVG_DAYS_CREDIT - 0 days of credit.

NOTE: For all the numerical variables, we will also add a flag which will be indicative of whether the value is missing or not. In total we will need 5 flag columns, one for each of the above columns.

```
[ ]: app_train.to_csv("app_train_modified.csv")
```

```
[ ]: # from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/MyDrive/Colab Notebooks" # @param {type:
↪"string"}
NOTEBOOK_NAME = "ds_project.ipynb" # @param {type:"string"}
#------------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic >␣
↪/dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

```
Mounted at /content/drive/
```