Code for Loans Dataset

Note: due to problems arising when trying to use pandas-profiling, Visual and Statistical evaluation of dataset were done manually.

```
import os

HOME = os.path.expanduser('~')
PROJECT_DIR = os.path.join(HOME, 'Desktop', 'NayaOneProject')
os.chdir(PROJECT_DIR)
print(os.getcwd())
```

```
/Users/apple/Desktop/NayaOneProject
```

```
#Import Necessary Libraries
```

```
#Libraries for EDA
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#Libraries for CTGAN
```

```
from sdv.tabular import CTGAN
from sdv.constraints import Inequality
```

```
#Libraries necessary for evaluation
```

```
from sdv.evaluation import evaluate
from table-evaluator import TableEvaluator
```

```
#Import Dataset
```

```
Loans = pd.read_csv("lc_loan.csv")
```

```
/Users/apple/opt/anaconda3/lib/python3.8/site-packages/IPython/core/
interactiveshell.py:3165: DtypeWarning: Columns (19,55) have mixed
types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

```
#Inspect Dataset
```

```
Loans.info()
```

```
#Drop Irrelevant Columns and Columns Missing a large portion of their
values.
#Or that were observed to decrease accuracy of CTGAN in early
fitments.
```

```
Loans =
Loans.drop(columns=['url','id','member_id','zip_code','grade','emp_tit
le','title','issue_d','annual_inc_joint','dti_joint','verification_sta
tus_joint','last_pymnt_d','next_pymnt_d','last_credit_pull_d','earlies
t_cr_line','desc','policy_code'])

Loans.info()

#Drop columns with large number of missing values

Loans.drop(Loans.iloc[:,42:53],axis=1,inplace=True)

Loans.drop(Loans.iloc[:,43:],axis=1,inplace=True)

#Missing Value imputation for Month Variables:

Loans.mths_since_last_major_derog.describe()

#Assign value outside range of values in above output, i.e. a value
less than
#the minimum to all missing values. This will result in only the null
entries
#of the column to be filled with a made up number -1.

Loans = Loans.fillna(value={'mths_since_last_major_derog':-1})

#Assign ranges to number of months and No Major Derog to -1, resulting
in the null values
#to be imputed correctly as No Major derog, since initially # of
months is not present
#in the column

Loans['Last_Major_Derog_Length']=pd.cut(Loans.mths_since_last_major_de
rog,bins=[-1.5,-0.5,6,12,24,48,96,200],labels=['No Major
Derog','<6months','<12months','1-2 years','2-4 years','4-8years','8+
years'])

#Check if imputation worked as intendeded by referencing summary
statistics above.

Loans.Last_Major_Derog_Length.value_counts()

Loans.mths_since_last_delinq.describe()

#Assign value outside range of values in above output, i.e. a value
less than
#the minimum to all missing values. This will result in only the null
entries
#of the column to be filled with a made up number -1.

Loans = Loans.fillna(value={'mths_since_last_delinq':-1})

#Assign ranges to number of monthsand No Delinq to -1, resulting in
the null values to
```

```
#be imputed correctly as No Delinq, since initially # of months is not
present
#in the column

Loans['Last_Delinq_Length']=pd.cut(Loans.mths_since_last_delinq ,bins=
[-1.5,-0.5,6,12,24,48,96,200],labels=['No
Delinq','<6months','<12months','1-2 years','2-4 years','4-8years','8+
years'])

#Check if imputation worked as intendeded by referencing summary
statistics above.

Loans.Last_Delinq_Length.value_counts()

Loans.mths_since_last_record.describe()

#Assign value outside range of values in above output, i.e. a value
less than
#the minimum to all missing values. This will result in only the null
entries
#of the column to be filled with a made up number -1.

Loans = Loans.fillna(value={'mths_since_last_record':-1})

#Assign ranges to number of months and No Previous Record to -1,
#resulting in the null values to
#be imputed correctly as No Previous Record, since initially # of
months is not present
#in the column

Loans['Last_Record_Length']=pd.cut(Loans.mths_since_last_record ,bins=
[-1.5,-0.5,6,12,24,48,96,200],labels=['No Previous
Record','<6months','<12months','1-2 years','2-4 years','4-8years','8+
years'])

#Check if imputation worked as intendeded by referencing summary
statistics above.

Loans.Last_Record_Length.value_counts()

#Drop original Months Columns

Loans =
Loans.drop(columns=['mths_since_last_major_derog','mths_since_last_rec
ord','mths_since_last_delinq'])

#Encode Address state into larger Regions in the US:

Loans.addr_state.value_counts()

#changing from states to regions
#West Region: Pacific & Mountain
Loans=Loans.replace(to_replace=['CA','OR','WA','HI','AK'],
```

```python
value='Pacific')
Loans=Loans.replace(to_replace=['NV','ID','MT','WY','UT','CO','AZ','NM
'], value='Mountain')
#MidWest Region: West NorthCentral & East NorthCentral
Loans=Loans.replace(to_replace=['WI','IL','MI','IN','OH'], value='East
North Central')
Loans=Loans.replace(to_replace=['ND','SD','NE','KS','MN','IA','MO'],
value='West North Cetral')
#North-East Region:Middle Atlantic & New England
Loans=Loans.replace(to_replace=['NY','ME','CT','RI','NJ'],
value='Middle Atlantic')
Loans=Loans.replace(to_replace=['PA','VT','ME','NH','MA'], value='New
England')
#South Region: West South Central, East South Central, South Atlantic
Loans=Loans.replace(to_replace=['TX','OK','AR','LA'], value='West
South Central')
Loans=Loans.replace(to_replace=['KY','TN','MS','AL'], value='East
South Central')
Loans=Loans.replace(to_replace=['DE','MD','DC','WV','VA','NC','SC','GA
','FL'], value='South Atlantic')

#Drop Remaining Null Values:

Loans = Loans.dropna()

#Turn Object Columns to Categorical:

Loans[Loans.select_dtypes(['object']).columns] =
Loans.select_dtypes(['object']).apply(lambda x: x.astype('category'))

#Key Summary Statistics:

#Numerical Variable Statistics:

Loans[['loan_amnt','installment','int_rate','annual_inc','total_pymnt'
]].describe().transpose()

#Categorical Variable Statistics:

Loans[['emp_length','verification_status','loan_status','addr_state','
home_ownership']].describe(include=object).transpose()

#Correlation Matrix Of Numerical Variables

corrmatrix = Loans.corr()
plt.figure(figsize=(15, 12))
sns.heatmap(corrmatrix, annot=False)
plt.savefig('Correlation Matrix.png')

#Graphs Of distribution Plots

sns.displot(Loans, x="loan_amnt", kind="kde").set(title='Original
Distribution of Loan Amount',xlabel='Loan Amount')
```

```
sns.displot(Loans, x="total_pymnt", kind="kde").set(title='Original
Distribution of Total Payment',xlabel='Total Payment')

sns.displot(Loans, x="installment", kind="kde").set(title='Original
Distribution of Installment',xlabel='installment')

LoanStatusDist = sns.catplot(y="loan_status",kind="count",
data=Loans).set(title='Original Distribution of Loan
Status',ylabel='Status Of Loan')

LoanStatusDist = sns.catplot(y="home_ownership",kind="count",
data=Loans).set(title='Original Distribution of Home
Ownership',ylabel='Home Ownership')

LoanStatusDist = sns.catplot(y="addr_state",kind="count",
data=Loans).set(title='Original Distribution of Areas of Loans
Issued',ylabel='Areas in the US')

ppdata = Loans[['loan_amnt', 'installment',
'int_rate','loan_status','total_pymnt']].copy()

sns.pairplot(ppdata)

plt.scatter(Loans.loan_amnt,Loans.installment)
plt.title("Loan Amount VS Installment")
plt.xlabel("Loan Amount")
plt.ylabel("Installment")
plt.savefig('Corr1.png')
plt.show()

plt.scatter(Loans.loan_amnt,Loans.funded_amnt)
plt.title("Loan Amount VS Funded Amount")
plt.xlabel("Loan Amount")
plt.ylabel("Funded Amount")
plt.savefig('Corr2.png')
plt.show()

samplecorrmatrix = Loans.corr()
plt.figure(figsize=(15, 12))
sns.heatmap(samplecorrmatrix, annot=False)
plt.savefig('Original Correlation Matrix.png')

#Create Samples out of Processed Dataset Using simple random sample
and two coefficients,
#0.1 and 0.01

SampleLoans = Loans.sample(frac = 0.1)

SmallSample = Loans.sample(frac=0.01)

SampleLoans.info()
```

```
#Corr Matrix of Variables to see if original Correlations hold, and
sampling hasn't
#deteriorated originality of data.

samplecorrmatrix = SampleLoans.corr()
plt.figure(figsize=(15, 12))
sns.heatmap(samplecorrmatrix, annot=False)
plt.savefig('Sample Correlation Matrix.png')

#Introduce Constrains needed after original correlation inspection:

equality_constraint1 = Inequality(
    low_column_name='funded_amnt',
    high_column_name='loan_amnt'
)

equality_constraint2 = Inequality(
    low_column_name='funded_amnt_inv',
    high_column_name='loan_amnt'
)

inequality_constraint1 = Inequality(
    low_column_name='funded_amnt_inv',
    high_column_name='funded_amnt'
)

constraints = [equality_constraint1, equality_constraint2]

constraints2 = [inequality_constraint1, equality_constraint1,
equality_constraint2]

# Define Different models to work with (please note numbers of models
in the code do not align with numbers of models in the report. In
particular, Model 1 in the report is model3 below, model 3 in the
report is model1 below)

model = CTGAN(constraints=constraints)

model1 = CTGAN(constraints=constraints2, epochs=200, verbose = 'TRUE')

model2 = CTGAN(constraints=constraints2, epochs=100,
              batch_size=100, verbose='TRUE')

model3 = CTGAN(verbose='TRUE')

model4 = CTGAN(constraints=constraints)

model5 = CTGAN(constraints=constraints2, epochs=200, verbose='TRUE')

model6 = CTGAN(constraints=constraints2, epochs=200, batch_size = 100,
verbose='TRUE')
```

```
#Fit models to original sample as fitting to original data was
impossible given the processing power we had, due to the enormous size
of the dataset.

#Started of with fitment of the smaller dataset to observe which model
performs better.

model.fit(SmallSample)

model1.fit(SmallSample)

model2.fit(SmallSample)

model3.fit(SmallSample)

model4.fit(SmallSample)

model5.fit(SmallSample)

model6.fit(SmallSample)

# Sample Synthetic Data generated

new_data = model.sample(num_rows=1000)

new_data1 = model1.sample(num_rows=1000)

new_data2 = model2.sample(num_rows=1000)

new_data3 = model3.sample(num_rows=1000)

new_data4 = model4.sample(num_rows=1000)

new_data5 = model5.sample(num_rows=1000)

new_data6 = model6.sample(num_rows=1000)

#Inspect Synthetic Data

SampleLoans.head()

new_data.head()

new_data1.head()

new_data2.head()

new_data3.head()

new_data4.head()

new_data5.head()

new_data6.head()

# Evaluate Synthetic Data
```

```
evaluate(new_data, SampleLoans)

evaluate(new_data1, SampleLoans)

evaluate(new_data2, SampleLoans)

evaluate(new_data3, SampleLoans)

evaluate(new_data4, SampleLoans)

evaluate(new_data5, SampleLoans)

evaluate(new_data6, SampleLoans)

#Observe Correlations:

print(SampleLoans.corr())

print(new_data.corr())

print(new_data1.corr())

print(new_data2.corr())

print(new_data3.corr())

print(new_data4.corr())

print(new_data5.corr())

print(new_data6.corr())

#It was evident that Model 3 (i.e. model1) performs better so we
proceeded on with fitting the slightly bigger sample 'Loans Sample' to
model1.

model1b = CTGAN(constraints=constraints2)

model1b.fit(SampleLoans)

#Due to the increased size of the dataset we sample a slightly larger
sample from it. 4000rows>1000rows.

new_data1b = model1b.sample(num_rows=1000)

new_data1bb = model1b.sample(num_rows=4000)

#Evaluation of results.

evaluate(newdata1b,SampleLoans)

evaluate(newdata1bb,SampleLoans)

#Visual and Statistical Evaluation

#Numerical variables statistics of original sample and synthetic data:
```

```
SampleLoans[['loan_amnt','installment','int_rate','annual_inc','total_
pymnt']].describe().transpose()

new_data1bb[['loan_amnt','installment','int_rate','annual_inc','total_
pymnt']].describe().transpose()

#Categorical variables statistics of original sample and synthetic
data:

SampleLoans[['emp_length','verification_status','loan_status','addr_st
ate','home_ownership']].describe(include=object).transpose()

new_data1bb[['emp_length','verification_status','loan_status','addr_st
ate','home_ownership']].describe(include=object).transpose()

#Defining variables to include in comparative correlation matrix.

OrigCorrMatrix =
SampleLoans[['loan_amnt','funded_amnt','int_rate','installment',
'annual_inc','total_pymnt']].copy()

SynthCorrMatrix =
new_data1bb[['loan_amnt','funded_amnt','int_rate','installment',
'annual_inc','total_pymnt']].copy()

#Correlation matrix of chosen variables from synthetic and original
distribution

corrmatrix = OrigCorrMatrix.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corrmatrix, annot=True)
plt.savefig('Original Sample Correlation Matrix.png')

corrmatrix = SynthCorrMatrix.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corrmatrix, annot=True)
plt.savefig('Synthetic Sample Correlation Matrix.png')

#Employ Table Evaluator.

#Need to define categorical columns to feed in the table evaluator

CatColumns=['term','sub_grade','emp_length','home_ownership','loan_sta
tus','verification_status','pymnt_plan','purpose','addr_state','initia
l_list_status','application_type','Last_Major_Derog_Length','Last_Deli
nq_Length','Last_Record_Length']

table_evaluator = TableEvaluator(SampleLoans, new_data1bb,
cat_cols=CatColumns)

table_evaluator.visual_evaluation()

#Test ability to classify Loan status
```

```
table_evaluator.evaluate(target_col='loan_status')
```