Team 6: Fairness-Aware Instrumentation of ML Pipelines

Biao Huang, Chenqin Yang, Rui Jiang, Zhengyuan Ding
New York University
{bh1918,cy1355,rj1407,zd415}@nyu.edu

ABSTRACT

Fairness is becoming one of the most popular topics in machine learning community. While most of related researches focus on fairness in algorithms and modeling, our paper digs into fairness issues arise during data preparation stage. To enhance fairness-awareness in preprocessing, the paper proposes a Data Preprocessing Function Tracer (DPF Tracer) to help users track changes in user-defined features for protected groups in an automated way. DPF Tracer produces two forms of outputs 1) a DAG visualization of data pipelines and 2) logs on related feature changes after each operation. We also demonstrate a few use cases and results after using DPF Tracer on four fairness-related datasets. In the end, we discuss about the benefits and drawbacks of our tool and list some future works.

1 INTRODUCTION

Software applications using machine-learning based technologies have received growing interests from industry world. While in early years, accuracy and efficiency are more of priority, in recent years however, awareness in responsible use of data science becomes an emerging field of research, especially in issues like fairness.[1,2] Unconscious and implicit biases can be produced during machine learning projects if approached without care, thus causing discrimination against historically disadvantaged groups. Therefore, it is morally and legally motivated to address the potential fairness problems in ML systems.

In an real-world end-to-end machine learning system, the problem can be more vulnerable. Contrary to textbook ML cases, the industrial ML system is often very complicated and largely relies on the data preparation part. As is shown in Figure 1, the data preparation stage for industrial deployment purposes often involves steps like data integration from various sources, data filtering on specific constraints, data cleansing such as missing data imputation and re-scaling, and feature engineering by encoding raw data. In each kind of operation, the data distributions may change and lead to some biases.

- Data integration. When joining data from different sources, number of records may change depending on the function and data properties. For instance, when doing inner join, if there is no shared identifier between two data sources, the number of records will be reduced.
- Data filtering. When doing filtering, in most cases, the filtering function is not uniform and will result to changes in data distributions with regard to protected groups. For example, assuming the distribution of a feature completely depends on time, if we apply the filtering functions to timestamps, the distribution of the target feature will be changed.
- Missing data imputation. Since missing data will not contribute to sample distribution, the sample distribution may be changed due to some imputation strategies. For instance, if the protected groups contain a large number of missing values and we impute them using most frequent values, the outputs will be biased towards values we used for imputation. This may have a negative influence on model performance and validity.

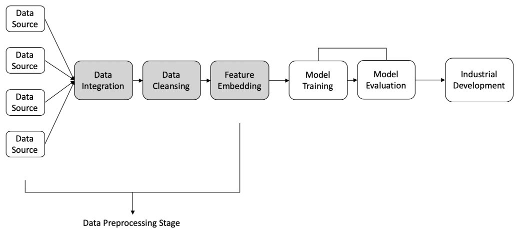


Figure 1: Data Preprocessing Stage in a ML Pipeline

Before feeding datasets into machine learning models, bias may occur in any step and make a big difference to the result. A lot of recent fairness-related researches have been focused on algorithms and modeling,[3] while fairness issues raised in data preparation stage are of lack attention.

Therefore, this paper focuses on fairness awareness in the data preparation pipeline part and proposes a Data Preprocessing Function Tracer (DPF Tracer) in order to help user monitor changes in user-defined features for protected groups. Specifically, DPF Tracer provides two forms of outputs in both visualization of data preparation pipelines using DAG (directed acyclic graph) structure and logs on related changes in sample distribution for each operation. Details will be discussed in Section 2.2.

The main achievements of the DPF Tracer are listed below.

- Generate DAGs and execution logs for intuitive bias visualization.
- Take user-defined features for arguments and features can be highly customized.
- Apply versatile evaluation metrics designed for numerical and categorical features separately.

2 PROBLEM STATEMENT & APPROACH

2.1 Problem Statement

When we implement an end-to-end ML system in the real world, the data preparation stage is crucial to how the final model is produced and performs. However, unintentional bias may occur in any operation within this stage if approached without care.

Thus, we aim to provide users with a tool to track potential fairness problems that can arise in the data preparation stage by tracing changes in proportion of records from user-defined protected groups.

2.2 Approach

We propose two parts to approach fairness-aware instrumentation of ML pipelines.

- End-to-End machine learning pipeline as DAGs.

In order to track changes during the data preparation stage, we first need to understand the pipeline structure and its operational dependencies. For ML data preprocessing pipelines, *Pandas* and *Scikit-Learn* are two most prevalent tools that can produce declarative function abstractions, such as Pandas DataFrame slicing, joining and Sklearn *ColumnTransfomer*. In this paper, the operation unit is *Pandas DataFrame* and operations are limited to Python built-in functions, *Pandas* and *Scikit-Learn* operations. To represent these data abstractions, DAG is a natural choice, where vertices stand for operations and edges indicate dependencies. Users are able to construct an concrete overview of the pipeline structure through DAG visualization and quickly identify sources of a certain data operation.

- Logs of changes in sample distribution.

The DPF Tracer will take in user-defined target columns and monitor line-wise distribution changes in them. When parsing each line, the tracer will generate log files based on *Pandas* or *SkLearn* operations. Different evaluation metrics are introduced to logs to trace targeted numerical and categorical features respectively.

The reason we split the user-input list into two separate lists is that even though most of the bias-related features will be categorical such as Gender, Race and etc, numerical features can still be of importance such as age and income. Thus, some of the metrices can be shared across different types but some will be introduced respectively.

For targeted categorical features, the Tracer will trace: 1) number of missing values, 2) number of classes/levels, 3) number of records for each class/level and 4) proportion for each class/level (since sometimes the minority group can be turned into the majority group after some operations).

For targeted numerical features, the Tracer will trace: 1) number of records, 2) number of missing values, 3) Median and Median Absolute Deviation and 4) Range/Scaling. Inspired and recommended by Professor Hellerstein,[4] we choose median and MAD which is defined below as the statistics instead of mean and standard deviation since they are more robust to outliers and stable with respect to the corruptions.

$$\underset{i}{\mathrm{MAD}}\left\{k_i\right\} = \underset{i}{\mathrm{median}}\left\{\left|k_i - \underset{j}{\mathrm{median}}\left\{k_j\right\}\right|\right\}$$

where k_i is the i^{th} value of a series of numbers

3 IMPLEMENTATION

To effectively trace the target groups, our DPF Tracer takes the original pipeline, the user-defined targeted categorical features and numerical features as inputs. The Tracer starts with "decomposition" of the pipeline so that line-wise operations can be stored into a list. The list of operations can be executed and inspected one by one afterwards. To correctly decompose the pipeline, we utilized the ***Inspect*** module in Python, which is helpful especially when retrieving the source code of a method or extracting and formatting the arguments of a function.[5] After getting the single string that contains the whole source code of the pipeline, we mainly utilize the idea of balanced brackets[6] to break down the pipeline into line-wise operations.

After having the executable list, to help users better understand how the distribution of the targeted features change in the proportion of records from protected groups, we built a DAG visualizer inside Tracer to help users have a more intuitive understanding of each step in the whole pipeline. More importantly, we generated two major *Dataframes* to record categorical and numerical features separately, each containing tracking metrices that can measure changes in distributions as we have discussed in Section 2.

By looping through the executable list, the Tracer keeps records of all changes occurred between each line and output the logs of changes in the targeted features that users care about. Specifically, we used a class called *DataFlowVertex* that treats each step in the pipeline as a vertex which stores the information of its parent vertices, operation name, data source name and affected parameters in the step. Due to the difference between *Pandas* and *Sklearn* operations and the property that *Sklearn Pipeline* can be treated as a whole, we designed separate functions that can tell *Sklearn* from *Pandas* operations. With the *Pandas* part separated from *Sklearn* part, then for *pandas*, we built a function *pd_to_dataflow_graph* that can handle following operators:

- data loading operators: eg. *pd.read_csv()*
- relational operators: joins *pd.Concat(), pd.join(), pd.merge()*; selections *df.loc[some condition]*; projections
- user-defined operators: *df.apply(lambda x:...)*

As for the *Sklearn* part, since *ColumnTransformer* and *Pipeline* have a natural pipeline structure, it is easier for us to formulate. Next, we utilized the *Digraph* from ***graphviz*** module in Python. Due to our well-designed structure of recording each operation as a *DataFlowVertex* class, we could easily use the information such as *parent_vertices*, *name*, *operation* to build each node containing necessary information, and then connected them by edges according to their parent-child relations.

At last, we utilized a design pattern *FunctionWrapper* since the functions in our project are relatively complicated. We wrapped the pipeline-to-graph functions, DAG visualizer and log-metrices function into *Tracer*, so that each time user can give customized categorical and numeric features that they care about and the preprocessing pipeline to *Tracer* and let it do the rest of the work.

4 EVALUATION

4.1 Experimental Setup

Our implementation is conducted under the MacOS Mojave/-Catalina Version with Python 3.6.5. The libraries we used are Pandas==0.23.0, Numpy==1.14.3, Scikit-learn==0.21.2 and Inspect==Python3.6.

We also tested our Tracer on different versions of libraries including Pandas==0.25.1, Numpy==1.17, Scikit-learn==0.21.3, and the Tracer still works compatibly.

4.2 Datasets

To construct the test case of different pipelines with varying complexity, we mainly use four datasets to build pipelines from easy to complex ones.

1) adult-sample.csv,[7] which is retrieved from Assignment3. Two pipelines are built using this dataset — Adult Pipeline Easy & Adult Pipeline Hard.

2) Loan dataset,[8] which is retrieved from a loan eligibility prediction contest held by Analytics Vidhya. The pipeline tested using this dataset — Loan Pipeline.

3) Statlog (German Credit - SCHUFA),[9] which contains several categorical and numerical attributes of 1,000 credit applicants in Germany, including the result of their credit score. Two pipelines are tested using this dataset — German Easy Pipeline & German Normal Pipeline.

4) COMPAS Recidivism Racial Bias,[10] which is widely used in fairness-related researches to study the algorithm bias that occurs in terms of favoring certain racial groups. The pipeline tested using this dataset — Compas Pipeline.

4.3 Results

Here we present several results from the Tracer implemented on the pipelines using COMPAS Recidivism Racial Bias dataset and adult-sample dataset. The pipeline implemented on adult-sample dataset provides a quick idea on how the Tracer functions and its corresponding output logs while the one implemented on COMPAS dataset has high complexity showing the robustness and wide coverage of what the Tracer can do.

- Adult Sample Dataset

The pipeline we used for this dataset mainly comes from the task3 and task4 of assignment3. The pipeline is standard in data processing stage consisting of operations from *Pandas* and *Sklearn*.

The DAG plot is generated in Figure 2 in the Appendix section, where we can observe that the *Sklearn* operations are correctly connected and pointed after the ***dropna*** operation in *Pandas*. The ***label_binarize*** operation is skipped since the idea of our function wrapper is to detect changes over predictors throughout the data processing pipeline.

The logs of changes are shown at the bottom of the figure. All *Pandas* and *Sklearn* operations are inspected. If changes are detected, the corresponding changes in numerical or/and categorical features are displayed. For example, the ***dropna*** operation leads to changes in feature *age* and *hours-per-week* and the distributions of the three categorical features that we are tracing are also changed. From the operations of ***StandardScaler*** on both numerical features, we observe a dramatic change in terms of median, MAD and range, which confirms the operation of scaling. Users are allowed to detect the occurrence of anything unexpected as well.

- COMPAS Recidivism Racial Bias

The pipeline we defined using this dataset includes various types of *Pandas* and *Sklearn* operations. This pipeline includes almost all of the operations data scientists frequently use for data processing including *Pandas* operations such as data loading, data selection (both inplace or non-inplace), data cleaning (NA values, changing data types and etc), merge and concatenation of several dataframes, projection, filtering and *Sklearn* operations such as OneHotEncoder, SimpleImputer, StandardScaler and KBinsDiscretizer.

The DAG plot is generated in Figure 3 in the Appendix section. The dependencies and operations of each node are clearly labeled. Users can quickly grasp the big picture of the entire processing pipeline and the details of each node along with relationships between related nodes.

The log of changes are also shown at the bottom of the figure. Each line of codes are executed and examined. If changes are detected, it will be recorded and outputed. For example, from the line of "`data = data.dropna(subset = ['id', ...])`", we can see that dropping NA values decreases the total number of records of the numerical feature *age* and changes the distribution of the categorical feature *race* where the proportions of Hispanic and African-American are greatly decreased, which may leads to a bias in the modeling stage. From the line of `data = data.loc[(data['days_b_screening_arrest'] >= -30)]`, we may observe a great decrease in the total number of records for African-American. However, from the *class_percent* metric, we observe that the proportion this group covers actually increases for a certain amount. In an extreme case, if approached without care, the original majority group and minority group in a dataset from the beginning can be totally swapped, which may be hard to notice and also have a negative impact on the performance and fairness of the following model. On the other hand, if no change is detected, the Tracer will pass this line and continue the checking.

We also include all the outputs including DAGs and logs of changes generated from the other four pipelines in the Appendix section. (Figure 4, Figure 5, Figure 6, Figure 7)

5 DISCUSSION

This paper mainly presented our Data Preprocessing Function(DPF) Tracer which focuses on fairness in data preprocessing stage of ML system and is designed as a user-friendly tool helping users track changes in both categorical and numeric features they care about after each operation. Our DPF Tracer takes in user-defined target features and generates DAGs and execution logs for tracking how bias may occur during data preprocessing by our versatile evaluation metrics designed for numeric and categorical features respectively. In order to simplify, we also build a wrapper to wrap the whole DAG function and metrices log function together.

We experimented our DPF under various setup environments. In order to test the robustness and scalability of our DPF Tracer, We selected 4 independent representative datasets including COMPAS Recidivism Racial Bias, a popular dataset in fairness issues. Then we designed versatile preprocessing pipelines of all levels of complexity to test the robustness of our Tracer and how complex cases it can handle. As a result, our Tracer is robust enough to cover almost all the *Pandas*, *Scikit-Learn* and even some user-defined operations. The both big-pictured and detail-oriented DAGs our DPF Tracer generates also make it easier to trace operations in the meantime.

In the future, we may improve our DPF Tracer in the following directions: 1. introducing a module that can infer data schema before feed into the tracer; 2. customizing start and stop point in the whole pipeline; 3. building interactive GUI and augment visualization work.

6 DETAILED CONTRIBUTIONS

6.1 Biao Huang (bh1918)

Used *inspect* package to trace function codes. Generated basic logs from ML pipelines. Wrote support functions and finalized codes.

6.2 Chenqin Yang (cy1355)

Wrote supporting functions and debugged function tracer. Improved the tracer by defining and adding multiple metrices for log of changes.

6.3 Rui Jiang (rj1407)

Built data preprocessing pipelines; Implemented graph visualization using *Graphviz*; implemented wrapper function and debugged DAG part.

6.4 Zhengyuan Ding (zd415)

Extracted DAG from ML pipeline code (*Pandas* and *Scikit-Learn*) and implemented graph visualization using *Graphviz*.

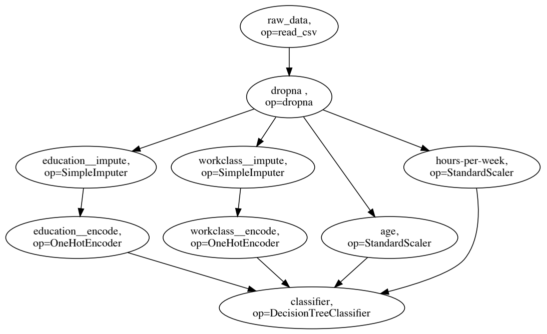
REFERENCES

[1] S. Barocas and A. D. Selbst. Big data's disparate impact. Calif. L. Rev., 104:671, 2016.
[2] C. O'Neil. Weapons of math destruction: How big data increases inequality and threatens democracy. Broadway Books, 2017.
[3] Hajian, Sara, Francesco Bonchi, and Carlos Castillo. "Algorithmic bias: From discrimination discovery to fairness-aware data mining." Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, 2016
[4] Hellerstein, Joseph M.. "Quantitative Data Cleaning for Large Databases." (2008).
[5] https://docs.python.org/3/library/inspect.html
[6] https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-python/
[7] https://github.com/schelterlabs/deml-lab/blob/master/assignment3/adult-sample.csv
[8] https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/
[9] http://www.fairness-measures.org/Pages/Datasets/Schufa.html
[10] https://www.kaggle.com/danofer/compass

Please check our repository for full version of codes and implementation documentations: https://github.com/schelterlabs/deml-project-6

APPENDIX

```python
@tracer(cat_col = ['race', 'occupation', 'education'], numerical_col = ['age', 'hours-per-week'])
def adult_pipeline_normal(f_path = '../pipelines/adult-sample_missing.csv'):
    raw_data = pd.read_csv(f_path, na_values='?')
    data = raw_data.dropna()

    labels = label_binarize(data['income-per-year'], ['>50K', '<=50K'])

    nested_categorical_feature_transformation = Pipeline(steps=[
        ('impute', SimpleImputer(missing_values=np.nan, strategy='most_frequent')),
        ('encode', OneHotEncoder(handle_unknown='ignore'))
    ])

    nested_feature_transformation = ColumnTransformer(transformers=[
        ('categorical', nested_categorical_feature_transformation, ['education', 'workclass']),
        ('numeric', StandardScaler(), ['age', 'hours-per-week'])
    ])

    nested_pipeline = Pipeline([
      ('features', nested_feature_transformation),
      ('classifier', DecisionTreeClassifier())])

    return nested_pipeline
```


raw_data, op=read_csv
dropna , op=dropna
education__impute, op=SimpleImputer
workclass__impute, op=SimpleImputer
hours-per-week, op=StandardScaler
education__encode, op=OneHotEncoder
workclass__encode, op=OneHotEncoder
age, op=StandardScaler
classifier, op=DecisionTreeClassifier

```
###################### Start Pandas Opeation ######################

-------------------------------------------------------
Inpected raw_data = pd.read_csv(f_path, na_values='?')
-------------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-14.0	0.0	0.0	-0.7413	-23.0
hours-per-week	-14.0	0.0	0.0	0.0000	0.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
race	-4.0	0.0	{'White': -6, 'Black': -2, 'Amer-Indian-Eskimo': -2, 'Other': 0, 'Asian-Pac-Islander': 0}	{'White': 0.0271, 'Black': -0.0111, 'Amer-Indian-Eskimo': -0.0184, 'Other': 0.0012, 'Asian-Pac-Islander': 0.0012}
occupation	-8.0	0.0	{'Exec-managerial': 0, 'Adm-clerical': 0, 'Craft-repair': -1, 'Sales': -1, 'Other-service': 0, 'Prof-specialty': -2, 'Transport-moving': -1, 'Machine-op-inspct': 0, 'Farming-fishing': 0, 'Handlers-cleaners': 0, 'Tech-support': 0, 'Protective-serv': -1}	{'Exec-managerial': 0.0106, 'Adm-clerical': 0.0099, 'Craft-repair': -0.0018, 'Sales': -0.0033, 'Other-service': 0.0068, 'Prof-specialty': -0.0157, 'Transport-moving': -0.0056, 'Machine-op-inspct': 0.0046, 'Farming-fishing': 0.0023, 'Handlers-cleaners': 0.0015, 'Tech-support': 0.0008, 'Protective-serv': -0.0101}
education	-2.0	0.0	{'HS-grad': -3, 'Bachelors': -1, 'Some-college': -4, '11th': -2, 'Masters': -2, '7th-8th': 0, '10th': 0, 'Assoc-voc': 0, 'Prof-school': 0, 'Assoc-acdm': 0, '12th': 0, '5th-6th': 0}	{'HS-grad': 0.0078, 'Bachelors': 0.0183, 'Some-college': -0.0138, '11th': -0.0133, 'Masters': -0.0147, '7th-8th': 0.0043, '10th': 0.0028, 'Assoc-voc': 0.0028, 'Prof-school': 0.0014, 'Assoc-acdm': 0.0014, '12th': 0.0014, '5th-6th': 0.0014}

```
**********
-------------------------------------------------------
Inpected data = raw_data.dropna()
-------------------------------------------------------


###################### Start Sklearn Pipeline ######################

-------------------------------------------------------
Operations SimpleImputer on education
-------------------------------------------------------

-------------------------------------------------------
Operations OneHotEncoder on education
-------------------------------------------------------

**********
Changes in categorical features!
```

	education
missing_count	0
num_class	-10
class_count	{0.0: 84, 1.0: 2}
class_percent	{0.0: 0.9767, 1.0: 0.0233}

```
**********
```

```
-------------------------------------------------------
Operations StandardScaler on age
-------------------------------------------------------

**********
Changes in numerical features!
```

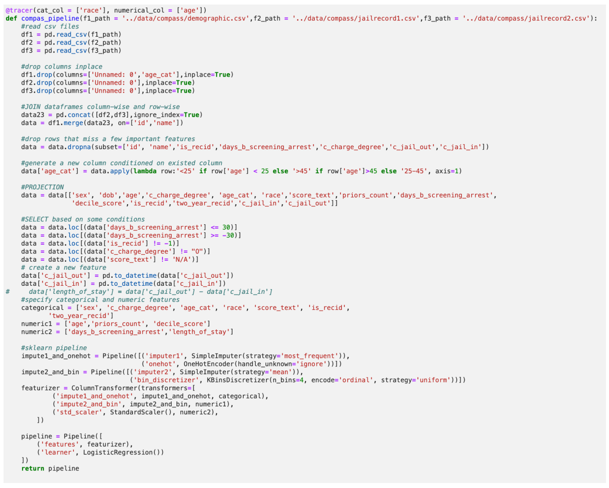
	age
count	0.0000
missing_count	0.0000
median	-36.0972
mad	-12.8320
range	-44.6418

```
**********

-------------------------------------------------------
Operations StandardScaler on hours-per-week
-------------------------------------------------------

**********
Changes in numerical features!
```

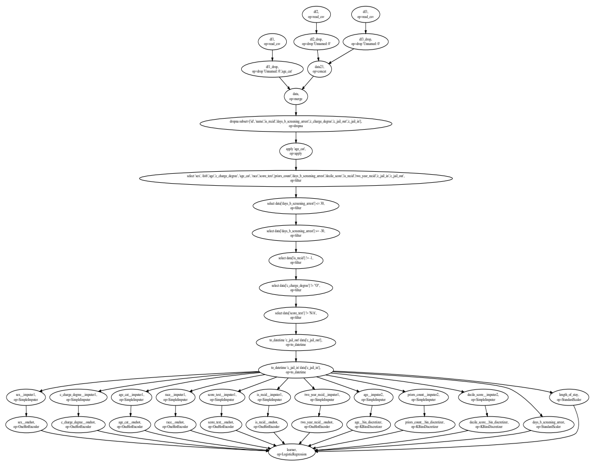
	hours-per-week
count	0.0000
missing_count	0.0000
median	-40.1126
mad	-1.3509
range	-63.7813

```
**********
```

Figure 2: Adult Sample Pipeline - Normal



```python
@tracer(cat_col = ['race'], numerical_col = ['age'])
def compas_pipeline(f1_path = '../data/compass/demographic.csv',f2_path = '../data/compass/jailrecord1.csv',f3_path = '../data/compass/jailrecord2.csv'):
    #read csv files
    df1 = pd.read_csv(f1_path)
    df2 = pd.read_csv(f2_path)
    df3 = pd.read_csv(f3_path)

    #drop columns inplace
    df1.drop(columns=['Unnamed: 0','age_cat'],inplace=True)
    df2.drop(columns=['Unnamed: 0'],inplace=True)
    df3.drop(columns=['Unnamed: 0'],inplace=True)

    #JOIN dataframes column-wise and row-wise
    data23 = pd.concat([df2,df3],ignore_index=True)
    data = df1.merge(data23, on=['id','name'])

    #drop rows that miss a few important features
    data = data.dropna(subset=['id', 'name','is_recid','days_b_screening_arrest','c_charge_degree','c_jail_out','c_jail_in'])

    #generate a new column conditioned on existed column
    data['age_cat'] = data.apply(lambda row:'<25' if row['age'] < 25 else '>45' if row['age']>45 else '25-45', axis=1)

    #PROJECTION
    data = data[['sex', 'dob','age','c_charge_degree', 'age_cat', 'race','score_text','priors_count','days_b_screening_arrest',
                 'decile_score','is_recid','two_year_recid','c_jail_in','c_jail_out']]

    #SELECT based on some conditions
    data = data.loc[(data['days_b_screening_arrest'] <= 30)]
    data = data.loc[(data['days_b_screening_arrest'] >= -30)]
    data = data.loc[(data['is_recid'] != -1)]
    data = data.loc[(data['c_charge_degree'] != "O")]
    data = data.loc[(data['score_text'] != 'N/A')]
    # create a new feature
    data['c_jail_out'] = pd.to_datetime(data['c_jail_out'])
    data['c_jail_in'] = pd.to_datetime(data['c_jail_in'])
#     data['length_of_stay'] = data['c_jail_out'] - data['c_jail_in']
    #specify categorical and numeric features
    categorical = ['sex', 'c_charge_degree', 'age_cat', 'race', 'score_text', 'is_recid',
           'two_year_recid']
    numeric1 = ['age','priors_count', 'decile_score']
    numeric2 = ['days_b_screening_arrest','length_of_stay']

    #sklearn pipeline
    impute1_and_onehot = Pipeline([('imputer1', SimpleImputer(strategy='most_frequent')),
                                   ('onehot', OneHotEncoder(handle_unknown='ignore'))])
    impute2_and_bin = Pipeline([('imputer2', SimpleImputer(strategy='mean')),
                                ('bin_discretizer', KBinsDiscretizer(n_bins=4, encode='ordinal', strategy='uniform'))])
    featurizer = ColumnTransformer(transformers=[
            ('impute1_and_onehot', impute1_and_onehot, categorical),
            ('impute2_and_bin', impute2_and_bin, numeric1),
            ('std_scaler', StandardScaler(), numeric2),
        ])

    pipeline = Pipeline([
        ('features', featurizer),
        ('learner', LogisticRegression())
    ])
    return pipeline
```


df2, op=read_csv
df3, op=read_csv
df1, op=read_csv
df2_drop, op=drop 'Unnamed: 0'
df3_drop, op=drop 'Unnamed: 0'
df1_drop, op=drop 'Unnamed: 0','age_cat'
data23, op=concat
data, op=merge
dropna subset=['id', 'name','is_recid','days_b_screening_arrest','c_charge_degree','c_jail_out','c_jail_in'], op=dropna
apply 'age_cat', op=apply
select 'sex', 'dob','age','c_charge_degree', 'age_cat', 'race','score_text','priors_count','days_b_screening_arrest','decile_score','is_recid','two_year_recid','c_jail_in','c_jail_out', op=filter
select data['days_b_screening_arrest'] <= 30, op=filter
select data['days_b_screening_arrest'] >= -30, op=filter
select data['is_recid'] != -1, op=filter
select data['c_charge_degree'] != "O", op=filter
select data['score_text'] != 'N/A', op=filter
to_datetime 'c_jail_out' data['c_jail_out'], op=to_datetime
to_datetime 'c_jail_in' data['c_jail_in'], op=to_datetime
sex__imputer1, op=SimpleImputer
c_charge_degree__imputer1, op=SimpleImputer
age_cat__imputer1, op=SimpleImputer
race__imputer1, op=SimpleImputer
score_text__imputer1, op=SimpleImputer
is_recid__imputer1, op=SimpleImputer
two_year_recid__imputer1, op=SimpleImputer
age__imputer2, op=SimpleImputer
priors_count__imputer2, op=SimpleImputer
decile_score__imputer2, op=SimpleImputer
length_of_stay, op=StandardScaler
sex__onehot, op=OneHotEncoder
c_charge_degree__onehot, op=OneHotEncoder
age_cat__onehot, op=OneHotEncoder
race__onehot, op=OneHotEncoder
score_text__onehot, op=OneHotEncoder
is_recid__onehot, op=OneHotEncoder
two_year_recid__onehot, op=OneHotEncoder
age__bin_discretizer, op=KBinsDiscretizer
priors_count__bin_discretizer, op=KBinsDiscretizer
decile_score__bin_discretizer, op=KBinsDiscretizer
days_b_screening_arrest, op=StandardScaler
learner, op=LogisticRegression

```
###################### Start Pandas Opeation ######################

-----------------------------------------------------
Inpected df1 = pd.read_csv(f1_path)
-----------------------------------------------------

-----------------------------------------------------
Inpected df2 = pd.read_csv(f2_path)
-----------------------------------------------------

-----------------------------------------------------
Inpected df3 = pd.read_csv(f3_path)
-----------------------------------------------------

-----------------------------------------------------
Inpected df1.drop(columns=['Unnamed: 0','age_cat'],inplace=True)
-----------------------------------------------------

-----------------------------------------------------
Inpected df2.drop(columns=['Unnamed: 0'],inplace=True)
-----------------------------------------------------

-----------------------------------------------------
Inpected df3.drop(columns=['Unnamed: 0'],inplace=True)
-----------------------------------------------------

-----------------------------------------------------
Inpected data23 = pd.concat([df2,df3],ignore_index=True)
-----------------------------------------------------

-----------------------------------------------------
Inpected data = df1.merge(data23, on=['id','name'])
-----------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-307.0	0.0	0.0	0.0	0.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
race	0.0	0.0	{'African-American': -159, 'Caucasian': -76, 'Hispanic': -53, 'Other': -17, 'Asian': 0, 'Native American': -2}	{'African-American': -0.0002, 'Caucasian': 0.0041, 'Hispanic': -0.0037, 'Other': -0.0001, 'Asian': 0.0002, 'Native American': -0.0002}

```
**********
-----------------------------------------------------
Inpected data = data.dropna(subset=['id', 'name','is_recid','days_b_screening_arrest','c_charge_degree','c_j
ail_out','c_jail_in'])
-----------------------------------------------------

-----------------------------------------------------
Inpected data = data[['sex', 'dob','age','c_charge_degree', 'age_cat', 'race','score_text','priors_count','d
ays_b_screening_arrest','decile_score','is_recid','two_year_recid','c_jail_in','c_jail_out']]
-----------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-284.0	0.0	0.0	0.0	0.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
race	0.0	0.0	{'African-American': -158, 'Caucasian': -87, 'Hispanic': -27, 'Other': -9, 'Asian': 0, 'Native American': -3}	{'African-American': -0.0019, 'Caucasian': 0.0016, 'Hispanic': -0.0005, 'Other': 0.0009, 'Asian': 0.0002, 'Native American': -0.0004}

```
**********
-----------------------------------------------------
Inpected data = data.loc[(data['days_b_screening_arrest'] <= 30)]
-----------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-451.0	0.0	0.0	0.0	0.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
race	0.0	0.0	{'African-American': -204, 'Caucasian': -188, 'Hispanic': -48, 'Other': -8, 'Asian': -1, 'Native American': -2}	{'African-American': 0.0042, 'Caucasian': -0.0052, 'Hispanic': -0.0016, 'Other': 0.0026, 'Asian': 0.0002, 'Native American': -0.0002}

```
**********
-----------------------------------------------------
Inpected data = data.loc[(data['days_b_screening_arrest'] >= -30)]
-----------------------------------------------------

-----------------------------------------------------
Inpected data = data.loc[(data['is_recid'] != -1)]
-----------------------------------------------------

-----------------------------------------------------
Inpected data = data.loc[(data['c_charge_degree'] != "O")]
-----------------------------------------------------

-----------------------------------------------------
Inpected data = data.loc[(data['score_text'] != 'N/A')]
-----------------------------------------------------


###################### Start Sklearn Pipeline ######################

-----------------------------------------------------
Operations SimpleImputer on race
-----------------------------------------------------

-----------------------------------------------------
Operations OneHotEncoder on race
-----------------------------------------------------

**********
Changes in categorical features!
```

	race
missing_count	0
num_class	-4
class_count	{1.0: 3175, 0.0: 2997}
class_percent	{1.0: 0.5144, 0.0: 0.4856}

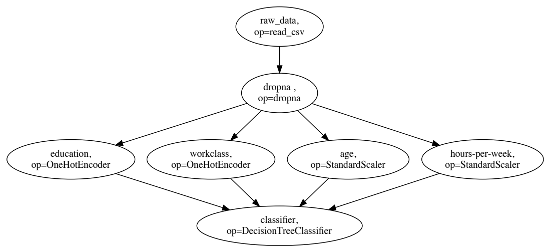
```
**********
-----------------------------------------------------
Operations SimpleImputer on age
-----------------------------------------------------

-----------------------------------------------------
Operations KBinsDiscretizer on age
-----------------------------------------------------

**********
Changes in numerical features!
```

	age
count	0.0000
missing_count	0.0000
median	-31.0000
mad	-10.3782
range	-75.0000

```
**********
```

Figure 3: Compas Pipeline - Normal

```python
@tracer(cat_col = ['race', 'occupation', 'education'], numerical_col = ['age', 'hours-per-week'])
def adult_pipeline_easy(f_path = '../pipelines/adult-sample.csv'):

    raw_data = pd.read_csv(f_path, na_values='?')
    data = raw_data.dropna()

    labels = label_binarize(data['income-per-year'], ['>50K', '<=50K'])

    feature_transformation = ColumnTransformer(transformers=[
        ('categorical', OneHotEncoder(handle_unknown='ignore'), ['education', 'workclass']),
        ('numeric', StandardScaler(), ['age', 'hours-per-week'])
    ])


    income_pipeline = Pipeline([
      ('features', feature_transformation),
      ('classifier', DecisionTreeClassifier())])

    return income_pipeline
```



raw_data, op=read_csv
dropna , op=dropna
education, op=OneHotEncoder
workclass, op=OneHotEncoder
age, op=StandardScaler
hours-per-week, op=StandardScaler
classifier, op=DecisionTreeClassifier

```
###################### Start Pandas Opeation ######################

-------------------------------------------------------
Inpected raw_data = pd.read_csv(f_path, na_values='?')
-------------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-8.0	0.0	0.0	-0.7413	-19.0
hours-per-week	-8.0	0.0	0.0	-1.4826	0.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
race	0.0	0.0	{'White': -6, 'Black': -1, 'Amer-Indian-Eskimo': -1, 'Asian-Pac-Islander': 0, 'Other': 0}	{'White': 0.007, 'Black': -0.0013, 'Amer-Indian-Eskimo': -0.0074, 'Asian-Pac-Islander': 0.0009, 'Other': 0.0009}
occupation	-6.0	0.0	{'Exec-managerial': 0, 'Adm-clerical': 0, 'Craft-repair': -1, 'Sales': 0, 'Prof-specialty': -1, 'Other-service': 0, 'Transport-moving': 0, 'Machine-op-inspct': 0, 'Farming-fishing': 0, 'Protective-serv': 0, 'Handlers-cleaners': 0, 'Tech-support': 0}	{'Exec-managerial': 0.0035, 'Adm-clerical': 0.003, 'Craft-repair': -0.0079, 'Sales': 0.0025, 'Prof-specialty': -0.0083, 'Other-service': 0.0021, 'Transport-moving': 0.0019, 'Machine-op-inspct': 0.0014, 'Farming-fishing': 0.0007, 'Protective-serv': 0.0005, 'Handlers-cleaners': 0.0005, 'Tech-support': 0.0002}
education	0.0	0.0	{'HS-grad': -1, 'Bachelors': 0, 'Some-college': -4, 'Masters': -1, '11th': -2, '7th-8th': 0, 'Assoc-voc': 0, '10th': 0, 'Prof-school': 0, 'Assoc-acdm': 0, '12th': 0, '5th-6th': 0}	{'HS-grad': 0.0152, 'Bachelors': 0.0191, 'Some-college': -0.0235, 'Masters': -0.0057, '11th': -0.0157, '7th-8th': 0.0026, 'Assoc-voc': 0.0026, '10th': 0.0017, 'Prof-school': 0.0009, 'Assoc-acdm': 0.0009, '12th': 0.0009, '5th-6th': 0.0009}

```
**********
-------------------------------------------------------
Inpected data = raw_data.dropna()
-------------------------------------------------------


###################### Start Sklearn Pipeline ######################

-------------------------------------------------------
Operations OneHotEncoder on education
-------------------------------------------------------

**********
Changes in categorical features!
```

	education
missing_count	0
num_class	-10
class_count	{0.0: 90, 1.0: 2}
class_percent	{0.0: 0.9783, 1.0: 0.0217}

```
**********
```

```
-------------------------------------------------------
Operations StandardScaler on age
-------------------------------------------------------

**********
Changes in numerical features!
```

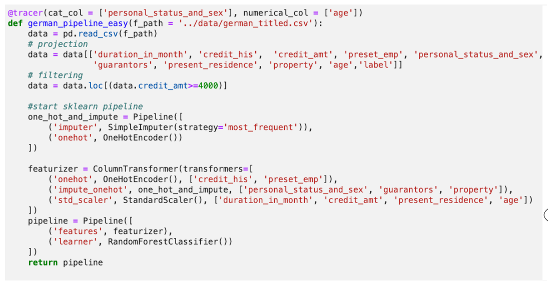
	age
count	0.0000
missing_count	0.0000
median	-36.1059
mad	-12.8706
range	-48.4315

```
**********

-------------------------------------------------------
Operations StandardScaler on hours-per-week
-------------------------------------------------------

**********
Changes in numerical features!
```

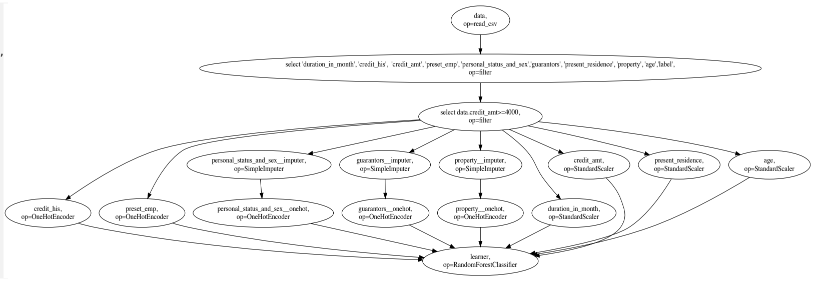
	hours-per-week
count	0.0000
missing_count	0.0000
median	-40.0814
mad	0.0000
range	-63.7616

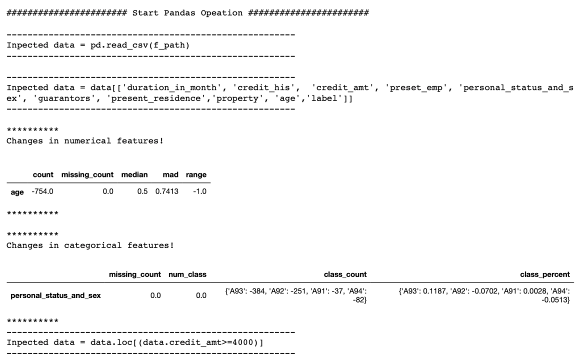
```
**********
```

Figure 4: Adult Sample Pipeline - Easy



```
@tracer(cat_col = ['personal_status_and_sex'], numerical_col = ['age'])
def german_pipeline_easy(f_path = '../data/german_titled.csv'):
    data = pd.read_csv(f_path)
    # projection
    data = data[['duration_in_month', 'credit_his',  'credit_amt', 'preset_emp', 'personal_status_and_sex',
                 'guarantors', 'present_residence', 'property', 'age','label']]
    # filtering
    data = data.loc[(data.credit_amt>=4000)]

    #start sklearn pipeline
    one_hot_and_impute = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder())
    ])

    featurizer = ColumnTransformer(transformers=[
        ('onehot', OneHotEncoder(), ['credit_his', 'preset_emp']),
        ('impute_onehot', one_hot_and_impute, ['personal_status_and_sex', 'guarantors', 'property']),
        ('std_scaler', StandardScaler(), ['duration_in_month', 'credit_amt', 'present_residence', 'age'])
    ])
    pipeline = Pipeline([
        ('features', featurizer),
        ('learner', RandomForestClassifier())
    ])
    return pipeline
```





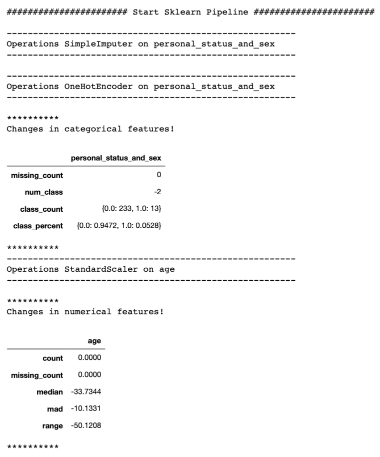
```
###################### Start Pandas Opeation ######################

-----------------------------------------------------
Inpected data = pd.read_csv(f_path)
-----------------------------------------------------

-----------------------------------------------------
Inpected data = data[['duration_in_month', 'credit_his',  'credit_amt', 'preset_emp', 'personal_status_and_s
ex', 'guarantors', 'present_residence','property', 'age','label']]
-----------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-754.0	0.0	0.5	0.7413	-1.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
personal_status_and_sex	0.0	0.0	{'A93': -384, 'A92': -251, 'A91': -37, 'A94': -82}	{'A93': 0.1187, 'A92': -0.0702, 'A91': 0.0028, 'A94': -0.0513}

```
**********
-----------------------------------------------------
Inpected data = data.loc[(data.credit_amt>=4000)]
-----------------------------------------------------
```



```
###################### Start Sklearn Pipeline ######################

-----------------------------------------------------
Operations SimpleImputer on personal_status_and_sex
-----------------------------------------------------

-----------------------------------------------------
Operations OneHotEncoder on personal_status_and_sex
-----------------------------------------------------

**********
Changes in categorical features!
```

	personal_status_and_sex
missing_count	0
num_class	-2
class_count	{0.0: 233, 1.0: 13}
class_percent	{0.0: 0.9472, 1.0: 0.0528}

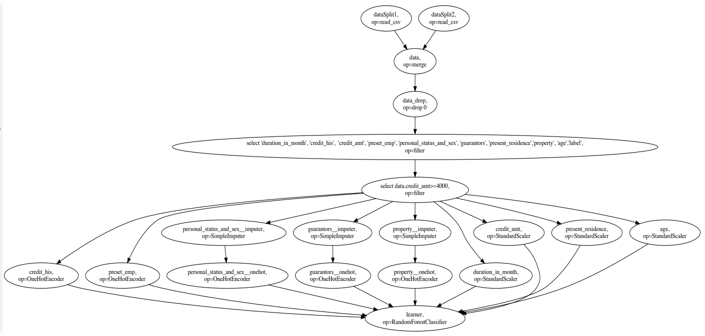
```
**********
-----------------------------------------------------
Operations StandardScaler on age
-----------------------------------------------------

**********
Changes in numerical features!
```

	age
count	0.0000
missing_count	0.0000
median	-33.7344
mad	-10.1331
range	-50.1208

```
**********
```

Figure 5: German Pipeline - Easy

```python
@tracer(cat_col = ['personal_status_and_sex'], numerical_col = ['age'])
def german_pipeline_normal(f_path_1='../data/german_titled_split_1.csv', f_path_2='../data/german_titled_split_2.csv'):
    # load data
    dataSplit1 = pd.read_csv(f_path_1, index_col = 0)
    dataSplit2 = pd.read_csv(f_path_2, index_col = 0)

    # join
    data = dataSplit1.merge(dataSplit2, on='identifier')

    # drop first col
    data.drop(data.columns[0], axis=1, inplace = True)

    # projection
    data = data[['duration_in_month', 'credit_his',  'credit_amt', 'preset_emp', 'personal_status_and_sex', 'guarantors', 'present_residence'
                 'property', 'age','label']]
    # filtering
    data = data.loc[(data.credit_amt>=4000)]

    #start sklearn pipeline
    one_hot_and_impute = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder())
    ])

    featurizer = ColumnTransformer(transformers=[
        ('onehot', OneHotEncoder(), ['credit_his', 'preset_emp']),
        ('impute_onehot', one_hot_and_impute, ['personal_status_and_sex', 'guarantors', 'property']),
        ('std_scaler', StandardScaler(), ['duration_in_month', 'credit_amt', 'present_residence', 'age'])
    ])
    pipeline = Pipeline([
        ('features', featurizer),
        ('learner', RandomForestClassifier())
    ])
    return pipeline
```


dataSplit1, op=read_csv
dataSplit2, op=read_csv
data, op=merge
data_drop, op=drop 0
select 'duration_in_month', 'credit_his', 'credit_amt', 'preset_emp', 'personal_status_and_sex', 'guarantors', 'present_residence','property', 'age','label', op=filter
select data.credit_amt>=4000, op=filter
personal_status_and_sex__imputer, op=SimpleImputer
guarantors__imputer, op=SimpleImputer
property__imputer, op=SimpleImputer
credit_amt, op=StandardScaler
present_residence, op=StandardScaler
age, op=StandardScaler
credit_his, op=OneHotEncoder
preset_emp, op=OneHotEncoder
personal_status_and_sex__onehot, op=OneHotEncoder
guarantors__onehot, op=OneHotEncoder
property__onehot, op=OneHotEncoder
duration_in_month, op=StandardScaler
learner, op=RandomForestClassifier

```
###################### Start Pandas Opeation ######################

-------------------------------------------------------
Inpected dataSplit1 = pd.read_csv(f_path_1, index_col = 0)
-------------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-inf	-inf	-inf	-inf	-inf

```
**********

-------------------------------------------------------
Inpected dataSplit2 = pd.read_csv(f_path_2, index_col = 0)
-------------------------------------------------------

-------------------------------------------------------
Inpected data = dataSplit1.merge(dataSplit2, on='identifier')
-------------------------------------------------------

-------------------------------------------------------
Inpected data.drop(data.columns[0], axis=1, inplace = True)
-------------------------------------------------------

-------------------------------------------------------
Inpected data = data[['duration_in_month', 'credit_his',  'credit_amt', 'preset_emp', 'personal_status_and_s
ex', 'guarantors', 'present_residence','property', 'age','label']]
-------------------------------------------------------

**********
Changes in numerical features!
```

	count	missing_count	median	mad	range
age	-754.0	0.0	0.5	0.7413	-1.0

```
**********

**********
Changes in categorical features!
```

	missing_count	num_class	class_count	class_percent
personal_status_and_sex	0.0	0.0	{'A93': -384, 'A92': -251, 'A91': -37, 'A94': -82}	{'A93': 0.1187, 'A92': -0.0702, 'A91': 0.0028, 'A94': -0.0513}

```
**********
-------------------------------------------------------
Inpected data = data.loc[(data.credit_amt>=4000)]
-------------------------------------------------------
```

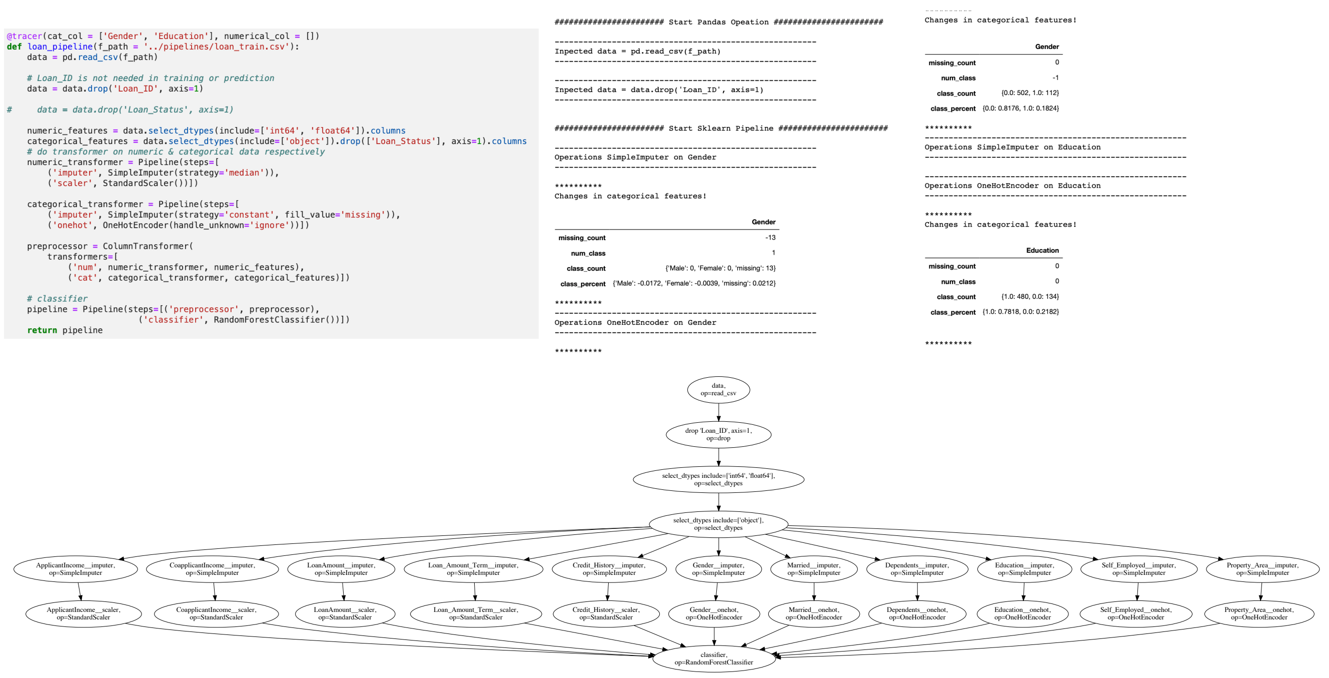
```
###################### Start Sklearn Pipeline ######################

-------------------------------------------------------
Operations SimpleImputer on personal_status_and_sex
-------------------------------------------------------

-------------------------------------------------------
Operations OneHotEncoder on personal_status_and_sex
-------------------------------------------------------

**********
Changes in categorical features!
```

	personal_status_and_sex
missing_count	0
num_class	-2
class_count	{0.0: 233, 1.0: 13}
class_percent	{0.0: 0.9472, 1.0: 0.0528}

```
**********
-------------------------------------------------------
Operations StandardScaler on age
-------------------------------------------------------

**********
Changes in numerical features!
```

	age
count	0.0000
missing_count	0.0000
median	-33.7344
mad	-10.1331
range	-50.1208

Figure 6: German Pipeline - Normal



```python
@tracer(cat_col = ['Gender', 'Education'], numerical_col = [])
def loan_pipeline(f_path = '../pipelines/loan_train.csv'):
    data = pd.read_csv(f_path)

    # Loan_ID is not needed in training or prediction
    data = data.drop('Loan_ID', axis=1)

#     data = data.drop('Loan_Status', axis=1)

    numeric_features = data.select_dtypes(include=['int64', 'float64']).columns
    categorical_features = data.select_dtypes(include=['object']).drop(['Loan_Status'], axis=1).columns
    # do transformer on numeric & categorical data respectively
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_features),
            ('cat', categorical_transformer, categorical_features)])

    # classifier
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', RandomForestClassifier())])
    return pipeline
```

```
###################### Start Pandas Opeation ######################

------------------------------------------------------
Inpected data = pd.read_csv(f_path)
------------------------------------------------------

------------------------------------------------------
Inpected data = data.drop('Loan_ID', axis=1)
------------------------------------------------------


###################### Start Sklearn Pipeline ######################

------------------------------------------------------
Operations SimpleImputer on Gender
------------------------------------------------------

**********
Changes in categorical features!

                                                                 Gender
missing_count                                                       -13
    num_class                                                         1
  class_count                {'Male': 0, 'Female': 0, 'missing': 13}
class_percent  {'Male': -0.0172, 'Female': -0.0039, 'missing': 0.0212}

**********
------------------------------------------------------
Operations OneHotEncoder on Gender
------------------------------------------------------

**********
```

```
**********
Changes in categorical features!

                                      Gender
missing_count                              0
    num_class                             -1
  class_count           {0.0: 502, 1.0: 112}
class_percent   {0.0: 0.8176, 1.0: 0.1824}

**********
------------------------------------------------------
Operations SimpleImputer on Education
------------------------------------------------------

------------------------------------------------------
Operations OneHotEncoder on Education
------------------------------------------------------

**********
Changes in categorical features!

                                   Education
missing_count                              0
    num_class                              0
  class_count           {1.0: 480, 0.0: 134}
class_percent   {1.0: 0.7818, 0.0: 0.2182}


**********
```

data, op=read_csv
drop 'Loan_ID', axis=1, op=drop
select_dtypes include=['int64', 'float64'], op=select_dtypes
select_dtypes include=['object'], op=select_dtypes
ApplicantIncome__imputer, op=SimpleImputer
CoapplicantIncome__imputer, op=SimpleImputer
LoanAmount__imputer, op=SimpleImputer
Loan_Amount_Term__imputer, op=SimpleImputer
Credit_History__imputer, op=SimpleImputer
Gender__imputer, op=SimpleImputer
Married__imputer, op=SimpleImputer
Dependents__imputer, op=SimpleImputer
Education__imputer, op=SimpleImputer
Self_Employed__imputer, op=SimpleImputer
Property_Area__imputer, op=SimpleImputer
ApplicantIncome__scaler, op=StandardScaler
CoapplicantIncome__scaler, op=StandardScaler
LoanAmount__scaler, op=StandardScaler
Loan_Amount_Term__scaler, op=StandardScaler
Credit_History__scaler, op=StandardScaler
Gender__onehot, op=OneHotEncoder
Married__onehot, op=OneHotEncoder
Dependents__onehot, op=OneHotEncoder
Education__onehot, op=OneHotEncoder
Self_Employed__onehot, op=OneHotEncoder
Property_Area__onehot, op=OneHotEncoder
classifier, op=RandomForestClassifier

Figure 7: Loan Pipeline