

764 lines - 10 Removals

```

1 import inspect
2 import logging
3 import uuid
4 from typing import Iterable, Set, Tuple, Union, List
5
6 import pandas as pd
7
8 from misc.logger import CustomLogger
9 from misc.decorators import timing, suppress_tracking
10 from misc.utils import extract_used_features, keys_mapping
11 from prov_acquisition.repository.neo4j import Neo4jFactory, Neo4jConnector
12
13 from prov_acquisition.prov_libraries.state import GlobalState, DataFrameState
14
15
16 class ProvenanceTracker:
17     """
18     Class that tracks changes in dataframes and traces provenance.
19     """
20
21     def __init__(self) -> None:
22
23         self.logger = CustomLogger('ProvenanceTracker')
24         self.logger.set_level(logging.DEBUG)
25
26         self.__dataframe_tracking = True
27         self.enable_dataframe_warning_msg = True
28
29         self.global_state = GlobalState()
30
31         self.neo4j = Neo4jFactory.create_neo4j_queries(uri="bolt://localhost",
32                                                         user="neo4j",
33                                                         pwd="password")
34         self.neo4j.delete_all()
35
36     @property
37     def dataframe_tracking(self) -> bool:
38         return self.__dataframe_tracking
39
40     @dataframe_tracking.setter
41     def dataframe_tracking(self, value: bool) -> None:
42         if self.enable_dataframe_warning_msg:
43             if value:
44                 self.logger.warning(f' Wrapper dataframe provenance tracker was enable!')
45             else:
46                 self.logger.warning(
47                     f' Wrapper dataframe provenance tracker was disable! Please use track_provenance method for tracking provenance.')
48             self.__dataframe_tracking = value
49
50     @suppress_tracking
51     def _wrapper_track_provenance(self, f, tracker_id: str):
52         """
53         Wrapper function to track provenance.

```

830 lines + 78 Additions

```

1 import inspect
2 import logging
3 import uuid
4 from typing import Iterable, Set, Tuple, Union, List
5
6 import pandas as pd
7 import numpy as np
8 from numbers import Number
9
10 from misc.logger import CustomLogger
11 from misc.decorators import timing, suppress_tracking
12 from misc.utils import extract_used_features, keys_mapping
13 from prov_acquisition.repository.neo4j import Neo4jFactory, Neo4jConnector
14
15 from prov_acquisition.prov_libraries.state import GlobalState, DataFrameState
16
17
18 class ProvenanceTracker:
19     """
20     Class that tracks changes in dataframes and traces provenance.
21     """
22
23     def __init__(self) -> None:
24
25         self.logger = CustomLogger('ProvenanceTracker')
26         self.logger.set_level(logging.DEBUG)
27
28         self.__dataframe_tracking = True
29         self.enable_dataframe_warning_msg = True
30
31         self.global_state = GlobalState()
32
33         self.neo4j = Neo4jFactory.create_neo4j_queries(uri="bolt://localhost",
34                                                         user="neo4j",
35                                                         pwd="password")
36         self.neo4j.delete_all()
37
38     @property
39     def dataframe_tracking(self) -> bool:
40         return self.__dataframe_tracking
41
42     @dataframe_tracking.setter
43     def dataframe_tracking(self, value: bool) -> None:
44         if self.enable_dataframe_warning_msg:
45             if value:
46                 self.logger.warning(f' Wrapper dataframe provenance tracker was enable!')
47             else:
48                 self.logger.warning(
49                     f' Wrapper dataframe provenance tracker was disable! Please use track_provenance method for tracking provenance.')
50             self.__dataframe_tracking = value
51
52     @suppress_tracking
53     def _wrapper_track_provenance(self, f, tracker_id: str):
54         """
55         Wrapper function to track provenance.

```

[illegible][illegible]

```

105         if result is None:
106             return result
107         else:
108             return self.create_tracked_d
ataframe(result, tracker_id=tracker_id)
109
110         # Preparing metadata for provenance
capture.
111         self.global_state.update_basic_prope
rty(description=f.__name__,
112         code=code,
113         code_line=inspect.stack()[1].lineno,
114         function=f.__name__)
115         self.global_state.operation_number +
= 1
116         dataframe_state.update_hash_df_outpu
t()
117         dataframe_state.update_hash_df_outpu
t_common_index()
118
119         # Capture the provenance
120
121         if self.global_state.function == 'me
rge':
122
123             right_df_input = kwargs['right']
124             if 'right' in kwargs else args[1] if len(args) >= 1 else
None
125
126             on = kwargs['on'] if 'on' in kwa
rgs else args[3] if len(args) >= 4 else None
127             left_on = kwargs['left_on'] if
'left_on' in kwargs else args[4] if len(args) >= 5 else
None
128             right_on = kwargs['right_on'] if
'right_on' in kwargs else args[5] if len(args) >= 6 else
None
129             suffixes = kwargs['suffixes'] if
'suffixes' in kwargs else '_x', '_y'
130
131             if on:
132                 left_on = on
133                 right_on = on
134
135             self.global_state.operation_numb
er -= 1
136
137             dataframe_state_right = self.glo
bal_state.dataframes_to_state[right_df_input.tracker_id]
138             dataframe_state.update_hash_row
()
139             dataframe_state_right.update_has
h_row()
140             self.global_state.operation_numb
er += 1
141
142             self.__get_prov_join(dataframe_s
tate_left=dataframe_state,
143                                 dataframe_st
ate_right=dataframe_state_right, left_keys=set(left_on),
144                                 right_keys=s
et(right_on), suffixes=suffixes)
145         else:

```

```

107         if result is None:
108             return result
109         else:
110             return self.create_tracked_d
ataframe(result, tracker_id=tracker_id)
111
112         # Preparing metadata for provenance
capture.
113         self.global_state.update_basic_prope
rty(description=f.__name__,
114         code=code,
115         code_line=inspect.stack()[1].lineno,
116         function=f.__name__)
117         self.global_state.operation_number +
= 1
118         dataframe_state.update_hash_df_outpu
t()
119         dataframe_state.update_hash_df_outpu
t_common_index()
120
121         # Capture the provenance
122
123         if self.global_state.function == 'me
rge':
124
125             right_df_input = kwargs['right']
126             if 'right' in kwargs else args[1] if len(args) >= 1 else
None
127
128             on = kwargs['on'] if 'on' in kwa
rgs else args[3] if len(args) >= 4 else None
129             left_on = kwargs['left_on'] if
'left_on' in kwargs else args[4] if len(args) >= 5 else
None
130             right_on = kwargs['right_on'] if
'right_on' in kwargs else args[5] if len(args) >= 6 else
None
131             suffixes = kwargs['suffixes'] if
'suffixes' in kwargs else '_x', '_y'
132
133             if on:
134                 left_on = on
135                 right_on = on
136
137             self.global_state.operation_numb
er -= 1
138
139             dataframe_state_right = self.glo
bal_state.dataframes_to_state[right_df_input.tracker_id]
140             dataframe_state.update_hash_row
()
141             dataframe_state_right.update_has
h_row()
142             self.global_state.operation_numb
er += 1
143
144             self.__get_prov_join(dataframe_s
tate_left=dataframe_state,
145                                 dataframe_st
ate_right=dataframe_state_right, left_keys=set(left_on),
146                                 right_keys=s
et(right_on), suffixes=suffixes)
147         else:

```

```

146         feature_mapping = self.__get_pro
v_feature_rename(dataframe_state=dataframe_state)
147
148         self.__get_prov_space_transforma
tion(dataframe_state=dataframe_state,
149
feature_mapping=feature_mapping)
150
151         self.__get_prov_value_change(dat
aframe_state=dataframe_state, extra_used_features=used_f
eatures)
152
153         self.global_state.print_current_acti
vities_info()
154
155         self.__prepare_for_next_operation(da
taframe_state=dataframe_state,
156
                                upda
te_df_input=True)
157
158         if result is None:
159             return result
160         else:
161             return self.create_tracked_dataframe
(result, tracker_id=tracker_id)
162
163         return wrap
164
165         @suppress_tracking
166         @timing
167         def __get_prov_join(self, dataframe_state_left: Data
FrameState, dataframe_state_right: DataFrameState,
168
                                left_keys: Set[str], right_keys:
Set[str], suffixes: Tuple[str],
169
                                _merge_feature: bool = False) ->
None:
170
171             """
172             Captures the provenance related to the join oper
ation.
173
Known issues to address: The merge operation can
convert integer columns to float columns if they contain
null values.
174
Changing the type changes the hash of the row, r
esulting in missing corresponding indices.
175
:param dataframe_state_left: The first input dat
aframe state.
176
:param dataframe_state_right: The second input d
ataframe state.
177
:param left_keys: Set of keys used for the left
dataframe.
178
:param right_keys: Set of keys used for the righ
t dataframe.
179
:param suffixes: Suffixes for common keys.
180
:param _merge_feature: Indicates if the merge fe
ature has been previously generated for provenance.
181
182             """
183
184             function_name = "Join"
185
186             left_df_input = dataframe_state_left.df_input_co
py
187
188             right_df_input = dataframe_state_right.df_input_
copy
189
189             df_output = dataframe_state_left.df_output
190

```

```

148         feature_mapping = self.__get_pro
v_feature_rename(dataframe_state=dataframe_state)
149
150         self.__get_prov_space_transforma
tion(dataframe_state=dataframe_state,
151
feature_mapping=feature_mapping)
152
153         self.__get_prov_value_change(dat
aframe_state=dataframe_state, extra_used_features=used_f
eatures)
154
155         self.global_state.print_current_acti
vities_info()
156
157         self.__prepare_for_next_operation(da
taframe_state=dataframe_state,
158
                                upda
te_df_input=True)
159
160         if result is None:
161             return result
162         else:
163             return self.create_tracked_dataframe
(result, tracker_id=tracker_id)
164
165         return wrap
166
167         @suppress_tracking
168         @timing
169         def __get_prov_join(self, dataframe_state_left: Data
FrameState, dataframe_state_right: DataFrameState,
170
                                left_keys: Set[str], right_keys:
Set[str], suffixes: Tuple[str],
171
                                _merge_feature: bool = False) ->
None:
172
173             """
174             Captures the provenance related to the join oper
ation.
175
Known issues to address: The merge operation can
convert integer columns to float columns if they contain
null values.
176
Changing the type changes the hash of the row, r
esulting in missing corresponding indices.
177
:param dataframe_state_left: The first input dat
aframe state.
178
:param dataframe_state_right: The second input d
ataframe state.
179
:param left_keys: Set of keys used for the left
dataframe.
180
:param right_keys: Set of keys used for the righ
t dataframe.
181
:param suffixes: Suffixes for common keys.
182
:param _merge_feature: Indicates if the merge fe
ature has been previously generated for provenance.
183
184             """
185
186             function_name = "Join"
187
188             left_df_input = dataframe_state_left.df_input_co
py
189
190             right_df_input = dataframe_state_right.df_input_
copy
191
191             df_output = dataframe_state_left.df_output
192

```



```

242         if col_name in left_columns or c
ol_name.removesuffix(
243             left_suffix) in common_c
olumns or col_name in common_keys:
244
245             # Get and remove the used en
tity from the left dataframe state
246             used_entity = dataframe_stat
e_left.index_col_to_input_entities.pop(
247                 (left_index, col_name.re
movesuffix(left_suffix)), None)
248
249             if used_entity is None:
250                 continue
251
252             used_features.add(col_name)
253             used_entities.append(used_en
tity['id'])
254
255             # Create derivation relation
between used and generated entities
256             self.global_state.create_der
ivation(used_ent=used_entity['id'],
257
gen_ent=generated_entity['id'])
258
259             # Process right-only or both cases
260             if row['_merge'] == 'right_only' or row
['_merge'] == 'both':
261                 set_of_indexes = dataframe_state_rig
ht.hash_rows_to_indexes.get(right_hash_row, set())
262
263                 for right_index in set_of_indexes:
264                     if col_name in right_columns or
col_name.removesuffix(
265                         right_suffix) in common_
columns or col_name in common_keys:
266
267                         # Get and remove the used en
tity from the right dataframe state
268                         used_entity = dataframe_stat
e_right.index_col_to_input_entities.pop(
269                             (right_index, col_name.r
emovesuffix(right_suffix)), None)
270
271                         if used_entity is None:
272                             continue
273
274                         used_features.add(col_name)
275                         used_entities.append(used_en
tity['id'])
276
277                         # Create derivation relation
between used and generated entities
278                         self.global_state.create_der
ivation(used_ent=used_entity['id'],
279
gen_ent=generated_entity['id'])
280
281             # Collect invalidated entities
282             invalidated = []
283
284             for index in dataframe_state_left.index_col_to_i
nput_entities:
285                 invalidated.append(dataframe_state_left.inde
x_col_to_input_entities[index]['id'])
286

```

```

244         if col_name in left_columns or c
ol_name.removesuffix(
245             left_suffix) in common_c
olumns or col_name in common_keys:
246
247             # Get and remove the used en
tity from the left dataframe state
248             used_entity = dataframe_stat
e_left.index_col_to_input_entities.pop(
249                 (left_index, col_name.re
movesuffix(left_suffix)), None)
250
251             if used_entity is None:
252                 continue
253
254             used_features.add(col_name)
255             used_entities.append(used_en
tity['id'])
256
257             # Create derivation relation
between used and generated entities
258             self.global_state.create_der
ivation(used_ent=used_entity['id'],
259
gen_ent=generated_entity['id'])
260
261             # Process right-only or both cases
262             if row['_merge'] == 'right_only' or row
['_merge'] == 'both':
263                 set_of_indexes = dataframe_state_rig
ht.hash_rows_to_indexes.get(right_hash_row, set())
264
265                 for right_index in set_of_indexes:
266                     if col_name in right_columns or
col_name.removesuffix(
267                         right_suffix) in common_
columns or col_name in common_keys:
268
269                         # Get and remove the used en
tity from the right dataframe state
270                         used_entity = dataframe_stat
e_right.index_col_to_input_entities.pop(
271                             (right_index, col_name.r
emovesuffix(right_suffix)), None)
272
273                         if used_entity is None:
274                             continue
275
276                         used_features.add(col_name)
277                         used_entities.append(used_en
tity['id'])
278
279                         # Create derivation relation
between used and generated entities
280                         self.global_state.create_der
ivation(used_ent=used_entity['id'],
281
gen_ent=generated_entity['id'])
282
283             # Collect invalidated entities
284             invalidated = []
285
286             for index in dataframe_state_left.index_col_to_i
nput_entities:
287                 invalidated.append(dataframe_state_left.inde
x_col_to_input_entities[index]['id'])
288

```

```

287         for index in dataframe_state_right.index_col_to_
input_entities:
288             invalidated.append(dataframe_state_right.ind
ex_col_to_input_entities[index]['id'])
289
290         invalidated.extend(used_entities)
291
292         # Create activity and relation in the global sta
te
293         act_id = self.global_state.create_activity(func
tion_name=function_name, used_features=list(used_feature
s),
294
                                     descr
ption=self.global_state.description,
295
                                     code=
self.global_state.code, code_line=self.global_state.code
_line,
296
                                     track
er_id=dataframe_state_left.tracker_id)
297
298         self.global_state.create_relation(act_id=act_id,
generated=generated_entities, used=used_entities,
299
                                     invalidated=in
validated,
300
                                     same=False)
301
302         # Update the index_col_to_input_entities for the
left and right dataframe states
303         dataframe_state_left.index_col_to_input_entities
= index_col_to_input_entities
304         dataframe_state_right.index_col_to_input_entitie
s = {}
305
306         # Remove the '_merge' column from the output dat
aframe if the merge feature is not required
307         if not _merge_feature:
308             del df_output['_merge']
309
310         @suppress_tracking
311         @timing
312         def __get_prov_space_transformation(self, dataframe_
state: DataFrameState,
313
                                     feature_mapping:
dict = {}) -> None:
314             """
315             Captures the provenance related to a change in t
he dataframe's dimensionality.
316             This function uses indexes and can only be used
if the operation for capturing provenance does not invo
lve reindexing.
317
318             Types of operations captured by this method:
319             - Feature Selection: One or more features ar
e removed.
320             - Feature Augmentation: One or more features
are added.
321             - Instance Drop: One or more records are rem
oved.
322             - Instance Generation: One or more records a
re added.
323             - Dimensionality Reduction: Features and rec
ords are added/removed. The overall number of removed fe
atures and records is greater than those added.
324             - Space Augmentation: Features and records a
re added/removed. The overall number of added features a
nd records is greater than those removed.

```

```

289         for index in dataframe_state_right.index_col_to_
input_entities:
290             invalidated.append(dataframe_state_right.ind
ex_col_to_input_entities[index]['id'])
291
292         invalidated.extend(used_entities)
293
294         # Create activity and relation in the global sta
te
295         act_id = self.global_state.create_activity(func
tion_name=function_name, used_features=list(used_feature
s),
296
                                     descr
ption=self.global_state.description,
297
                                     code=
self.global_state.code, code_line=self.global_state.code
_line,
298
                                     track
er_id=dataframe_state_left.tracker_id)
299
300         self.global_state.create_relation(act_id=act_id,
generated=generated_entities, used=used_entities,
301
                                     invalidated=in
validated,
302
                                     same=False)
303
304         # Update the index_col_to_input_entities for the
left and right dataframe states
305         dataframe_state_left.index_col_to_input_entities
= index_col_to_input_entities
306         dataframe_state_right.index_col_to_input_entitie
s = {}
307
308         # Remove the '_merge' column from the output dat
aframe if the merge feature is not required
309         if not _merge_feature:
310             del df_output['_merge']
311
312         @suppress_tracking
313         @timing
314         def __get_prov_space_transformation(self, dataframe_
state: DataFrameState,
315
                                     feature_mapping:
dict = {}) -> None:
316             """
317             Captures the provenance related to a change in t
he dataframe's dimensionality.
318             This function uses indexes and can only be used
if the operation for capturing provenance does not invo
lve reindexing.
319
320             Types of operations captured by this method:
321             - Feature Selection: One or more features ar
e removed.
322             - Feature Augmentation: One or more features
are added.
323             - Instance Drop: One or more records are rem
oved.
324             - Instance Generation: One or more records a
re added.
325             - Dimensionality Reduction: Features and rec
ords are added/removed. The overall number of removed fe
atures and records is greater than those added.
326             - Space Augmentation: Features and records a
re added/removed. The overall number of added features a
nd records is greater than those removed.

```

```

325         - Space Transformation: Features and records
are added/removed. In this case, there can be a reduction
in dimensionality for one axis and a space augmentation
for the other.

326         :param dataframe_state: Input and output DataFrame
state.
327         :param feature_mapping: Mapping between the features
of the output DataFrame and the input DataFrame.
328         :return: None
329         """
330
331         function_name1 = "Feature Selection"
332         function_name2 = "Feature Augmentation"
333
334         function_name3 = "Instance Drop"
335         function_name4 = "Instance Generation"
336
337         function_name5 = "Dimensionality Reduction"
338         function_name6 = "Space Augmentation"
339         function_name7 = "Space Transformation"
340
341         df_input = dataframe_state.df_input_copy
342         df_output = dataframe_state.df_output
343
344         dropped_rows = df_input.index.difference(df_output.index)
345         dropped_cols = df_input.columns.difference(df_output.columns)
346         augs_rows = df_output.index.difference(df_input.index)
347         augs_cols = df_output.columns.difference(df_input.columns)
348         int_rows = df_output.index.intersection(df_input.index)
349
350         used_entities = []
351         generated_entities = []
352         used_cols = set()
353
354         self.logger.info(f' Dropped cols: {dropped_cols}')
355         self.logger.info(f' Dropped rows: {dropped_rows}')
356         self.logger.info(f' Generated rows: {augs_rows}')
357         self.logger.info(f' Generated cols: {augs_cols}')
358
359         output_values = df_output.to_numpy()
360
361         # Determine the type of function
362         function_name = function_name7
363
364         if len(dropped_cols) > 0 and len(augs_cols) == 0
and len(dropped_rows) == 0 and len(augs_rows) == 0:
365             function_name = function_name1
366         if len(dropped_cols) == 0 and len(augs_cols) > 0
and len(dropped_rows) == 0 and len(augs_rows) == 0:
367             function_name = function_name2

```

```

327         - Space Transformation: Features and records
are added/removed. In this case, there can be a reduction
in dimensionality for one axis and a space augmentation
for the other.

328         - One-Hot Encoding: Multiple binary features
added representing categorical values of another feature.
329
330         :param dataframe_state: Input and output DataFrame
state.
331         :param feature_mapping: Mapping between the features
of the output DataFrame and the input DataFrame.
332         :return: None
333         """
334
335         function_name1 = "Feature Selection"
336         function_name2 = "Feature Augmentation"
337
338         function_name3 = "Instance Drop"
339         function_name4 = "Instance Generation"
340
341         function_name5 = "Dimensionality Reduction"
342         function_name6 = "Space Augmentation"
343         function_name7 = "Space Transformation"
344
345         function_name8 = "One-Hot Encoding"
346
347         df_input = dataframe_state.df_input_copy
348         df_output = dataframe_state.df_output
349
350         dropped_rows = df_input.index.difference(df_output.index)
351         dropped_cols = df_input.columns.difference(df_output.columns)
352         augs_rows = df_output.index.difference(df_input.index)
353         augs_cols = df_output.columns.difference(df_input.columns)
354         int_rows = df_output.index.intersection(df_input.index)
355
356         used_entities = []
357         generated_entities = []
358         used_cols = set()
359
360         self.logger.info(f' Dropped cols: {dropped_cols}')
361         self.logger.info(f' Dropped rows: {dropped_rows}')
362         self.logger.info(f' Generated rows: {augs_rows}')
363         self.logger.info(f' Generated cols: {augs_cols}')
364
365         output_values = df_output.to_numpy()
366
367         # Determine the type of function
368         function_name = function_name7
369
370         if len(dropped_cols) > 0 and len(augs_cols) == 0
and len(dropped_rows) == 0 and len(augs_rows) == 0:
371             function_name = function_name1
372         if len(dropped_cols) == 0 and len(augs_cols) > 0
and len(dropped_rows) == 0 and len(augs_rows) == 0:
373             if set(pd.unique(df_output[augs_cols].values.ravel())) == set([0, 1]) and len(augs_cols) > 1:
374                 function_name = function_name8

```


		375	else:
		376	function_name = function_name2
368		377	
369	if len(dropped_cols) == 0 and len(augs_cols) ==	378	if len(dropped_cols) == 0 and len(augs_cols) ==
	0 and len(dropped_rows) > 0 and len(augs_rows) == 0:		0 and len(dropped_rows) > 0 and len(augs_rows) == 0:
370	function_name = function_name3	379	function_name = function_name3
371	if len(dropped_cols) == 0 and len(augs_cols) ==	380	if len(dropped_cols) == 0 and len(augs_cols) ==
	0 and len(dropped_rows) == 0 and len(augs_rows) > 0:		0 and len(dropped_rows) == 0 and len(augs_rows) > 0:
372	function_name = function_name4	381	function_name = function_name4
373		382	
374	if (len(dropped_cols) > len(augs_cols)) and len	383	if (len(dropped_cols) > len(augs_cols)) and len
	(dropped_rows) > len(augs_cols):		(dropped_rows) > len(augs_rows):
375	function_name = function_name5	384	function_name = function_name5
376	if (len(dropped_cols) < len(augs_cols)) and len	385	if (len(dropped_cols) < len(augs_cols)) and len
	(dropped_rows) < len(augs_cols):		(dropped_rows) < len(augs_rows):
377	function_name = function_name6	386	function_name = function_name6
378		387	
379	# Iterate over the removed rows to find values d	388	# Iterate over the removed rows to find values d
	eleted due to an Instance Drop operation		eleted due to an Instance Drop operation
380	for index in dropped_rows:	389	for index in dropped_rows:
381	for _, col_name in feature_mapping.items():	390	for col_name in df_input.columns:
382	used_entity = dataframe_state.index_col_	391	used_entity = dataframe_state.index_col_
	to_input_entities.pop((index, col_name), None)		to_input_entities.pop((index, col_name), None)
383	if used_entity:	392	if used_entity:
384	used_cols.add(col_name)	393	used_cols.add(col_name)
385	used_entities.append(used_entity['i	394	used_entities.append(used_entity['i
	d'])		d'])
386		395	
387	# Iterate over the added rows to find values add	396	# Iterate over the added rows to find values add
	ed due to an Instance Generation operation		ed due to an Instance Generation operation
388	for index in augs_rows:	397	for index in augs_rows:
389	i = df_output.index.get_loc(index)	398	i = df_output.index.get_loc(index)
390	for col_name, _ in feature_mapping.items():	399	for col_name, _ in feature_mapping.items():
391	col = df_output.columns.get_loc(col_nam	400	col = df_output.columns.get_loc(col_nam
	e)		e)
392	output_value = output_values[i, col]	401	output_value = output_values[i, col]
393	generated_entity = self.global_state.cre	402	generated_entity = self.global_state.cre
	ate_entity(value=output_value, feature_name=col_name,		ate_entity(value=output_value, feature_name=col_name,
394		403	
	index=index,		index=index,
395		404	
	instance=self.global_state.operation_number)		instance=self.global_state.operation_number)
396	dataframe_state.index_col_to_input_entit	405	dataframe_state.index_col_to_input_entit
	ies[(index, col_name)] = generated_entity		ies[(index, col_name)] = generated_entity
397	generated_entities.append(generated_entit	406	generated_entities.append(generated_entit
	ty['id'])		ty['id'])
398		407	
399	# Iterate over the remaining rows to find values	408	# Iterate over the remaining rows to find values
	added/removed due to feature removal/addition		added/removed due to feature removal/addition
400	for index in int_rows:	409	for index in int_rows:
401		410	
402	i = df_output.index.get_loc(index)	411	i = df_output.index.get_loc(index)
403		412	
404	for col_name in dropped_cols:	413	for col_name in dropped_cols:
405		414	
406	if col_name in set(feature_mapping.value	415	if col_name in set(feature_mapping.value
	s()):		s()):
407	continue	416	continue
408		417	
409	used_entity = dataframe_state.index_col_	418	used_entity = dataframe_state.index_col_
	to_input_entities.pop((index, col_name), None)		to_input_entities.pop((index, col_name), None)
410	if used_entity:	419	if used_entity:
411	used_cols.add(col_name)	420	used_cols.add(col_name)
412	used_entities.append(used_entity['i	421	used_entities.append(used_entity['i
	d'])		d'])
413		422	
414	for col_name in augs_cols:	423	for col_name in augs_cols:
415		424	

```

416         if col_name in feature_mapping:
417             continue
418         col = df_output.columns.get_loc(col_name)
419         output_value = output_values[i, col]
420         generated_entity = self.global_state.create_entity(
421             value=output_value, feature_name=col_name,
422             index=index,
423             instance=self.global_state.operation_number)
424         dataframe_state.index_col_to_input_entities[
425             (index, col_name)] = generated_entity
426         generated_entities.append(generated_entity['id'])

```

```

425
426         if len(generated_entities) > 0 or len(used_entities) > 0:
427             act_id = self.global_state.create_activity(
428                 function_name, used_features=list(used_cols),
429                 description=self.global_state.description,
430                 code=self.global_state.code,
431                 code_line=self.global_state.code_line,
432                 generated_records=len(augs_rows) > 0,
433                 generated_features=list(augs_cols),
434                 deleted_records=len(dropped_rows) > 0,
435                 deleted_used_features=list(dropped_cols),
436                 tracker_id=dataframe_state.tracker_id)
437         self.global_state.create_relation(
438             act_id=act_id, generated=generated_entities,
439             used=used_entities, invalidate_d=None, same=True)

```

```

425         if col_name in feature_mapping:
426             continue
427
428         col = df_output.columns.get_loc(col_name)
429         output_value = output_values[i, col]
430         generated_entity = self.global_state.create_entity(
431             value=output_value, feature_name=col_name,
432             index=index,
433             instance=self.global_state.operation_number)
434         dataframe_state.index_col_to_input_entities[
435             (index, col_name)] = generated_entity
436         generated_entities.append(generated_entity['id'])

```

```

435         # if function_name == function_name8:
436         #     col_val = pd.Series({col:df_input[col].unique()
437             #                     for col in df_input})
438
439         #     for col_name2 in col_val.index:
440         #         used_entity = dataframe_state.index_col_to_input_entities[
441             #             (index, col_name2)]
442
443         #         if len(set(augs_cols)) == len(np.unique(
444             #             col_val[col_name2])):
445             #             if set(augs_cols) == set(col_val[
446                 #                 col_name2]):
447                 #                     used_cols.add(col_name2)
448                 #                     used_entities.append(used_entity['id'])
449                 #                     self.global_state.create_derivation(
450                     #                         used_ent=used_entity['id'],
451                     #                         gen_ent=generated_entity['id'])

```

```

448
449         if len(generated_entities) > 0 or len(used_entities) > 0:
450             act_id = self.global_state.create_activity(
451                 function_name, used_features=list(used_cols),
452                 description=self.global_state.description,
453                 code=self.global_state.code,
454                 code_line=self.global_state.code_line,
455                 generated_records=len(augs_rows) > 0,
456                 generated_features=list(augs_cols),
457                 deleted_records=len(dropped_rows) > 0,
458                 deleted_used_features=list(dropped_cols),
459                 tracker_id=dataframe_state.tracker_id)

```

```

459
460         if function_name == function_name8:
461             self.global_state.create_relation(
462                 act_id=act_id, generated=generated_entities,
463                 used=used_entities, invalidate_d=None, same=True)

```

```

439
440     @suppress_tracking
441     @timing
442     def __get_prov_feature_rename(self, dataframe_state:
DataFrameState) -> dict:
443         """
444         Captures the provenance related to the renaming
of one or more features.
445
446         :param dataframe_state: Input and output DataFra
me state.
447         :return: dict - Mapping between the features of
the output DataFrame and the input DataFrame.
448         """
449
450         function_name = "Feature Rename"
451
452         df_input = dataframe_state.df_input_copy
453         df_output = dataframe_state.df_output
454
455         int_rows = df_output.index.intersection(df_inpu
t.index)
456
457         used_entities = []
458         generated_entities = []
459         used_features = set()
460         generated_features = set()
461
462         output_values = df_output.to_numpy()
463
464         hash_df_output_common_index = dataframe_state.ha
sh_df_output_common_index.to_dict()
465         hash_df_input = dataframe_state.hash_df_input.to
_dict()
466
467         feature_mapping = keys_mapping(hash_df_output_co
mmon_index, hash_df_input)
468
469         # Iterate over the intersecting rows to find fea
ture rename operations
470         for index in int_rows:
471
472             i = df_output.index.get_loc(index)
473
474             for col_name1, col_name2 in feature_mapping.
items():
475
476                 if col_name1 == col_name2:
477                     continue
478
479                 col = df_output.columns.get_loc(col_name
1)
480
481                 output_value = output_values[i, col]
482
483                 used_entity = dataframe_state.index_col_
to_input_entities.pop((index, col_name2), None)
484                 if used_entity:
485                     generated_entity = self.global_stat
e.create_entity(value=output_value, feature_name=col_nam
e1,

```

```

462                                     invali
dated=None, same=False)
463         else:
464             self.global_state.create_relation(act_id
=act_id, generated=generated_entities, used=used_entitie
s,
465                                     invali
dated=None, same=True)
466
467     @suppress_tracking
468     @timing
469     def __get_prov_feature_rename(self, dataframe_state:
DataFrameState) -> dict:
470         """
471         Captures the provenance related to the renaming
of one or more features.
472
473         :param dataframe_state: Input and output DataFra
me state.
474         :return: dict - Mapping between the features of
the output DataFrame and the input DataFrame.
475         """
476
477         function_name = "Feature Rename"
478
479         df_input = dataframe_state.df_input_copy
480         df_output = dataframe_state.df_output
481
482         int_rows = df_output.index.intersection(df_inpu
t.index)
483
484         used_entities = []
485         generated_entities = []
486         used_features = set()
487         generated_features = set()
488
489         output_values = df_output.to_numpy()
490
491         hash_df_output_common_index = dataframe_state.ha
sh_df_output_common_index.to_dict()
492         hash_df_input = dataframe_state.hash_df_input.to
_dict()
493
494         feature_mapping = keys_mapping(hash_df_output_co
mmon_index, hash_df_input)
495
496         # Iterate over the intersecting rows to find fea
ture rename operations
497         for index in int_rows:
498
499             i = df_output.index.get_loc(index)
500
501             for col_name1, col_name2 in feature_mapping.
items():
502
503                 if col_name1 == col_name2:
504                     continue
505
506                 col = df_output.columns.get_loc(col_name
1)
507
508                 output_value = output_values[i, col]
509
510                 used_entity = dataframe_state.index_col_
to_input_entities.pop((index, col_name2), None)
511                 if used_entity:
512                     generated_entity = self.global_stat
e.create_entity(value=output_value, feature_name=col_nam
e1,

```

```

485         index=index,
486         instance=self.global_state.operation_number)
487         generated_entities.append(generated_
entity['id'])
488         used_entities.append(used_entity['i
d'])
489         generated_features.add(col_name1)
490         used_features.add(col_name2)
491
492         dataframe_state.index_col_to_input_e
ntities[(index, col_name1)] = generated_entity
493         self.global_state.create_derivation
(used_ent=used_entity['id'], gen_ent=generated_entity['i
d'])
494
495         if len(generated_features) > 0:
496             self.logger.info(f' Feature rename detect:
{feature_mapping}')
497             act_id = self.global_state.create_activity(f
unction_name, used_features=list(used_features),
498             escription=self.global_state.description,
499             ode=self.global_state.code,
500             ode_line=self.global_state.code_line,
501             enerated_features=list(generated_features),
502             racker_id=dataframe_state.tracker_id)
503
504             self.global_state.create_relation(act_id=act
_id, generated=generated_entities, used=used_entities,
505             d=[],
506             same=True)
507
508             return feature_mapping
509
510             @suppress_tracking
511             @timing
512             def __get_prov_value_change(self, dataframe_state, e
xtra_used_features: set = None) -> None:
513
514                 """
515                 Captures the provenance related to a change in t
he values of the DataFrame.
516                 The type of generated activity can be of two typ
es:
517                 Value Transformation: generic case.
518                 Imputation: the DataFrame column has undergone a
replacement of null values.
519
520                 :param dataframe_state: Input and output DataFra
me state.
521                 :param extra_used_features: Extra features used
in the input. Indicates additional features that contri
bute to the generation of new entities.
522                 :return: None
523
524                 """
525
526                 self.logger.info(f' Check for value change...')
512         index=index,
513         instance=self.global_state.operation_number)
514         generated_entities.append(generated_
entity['id'])
515         used_entities.append(used_entity['i
d'])
516         generated_features.add(col_name1)
517         used_features.add(col_name2)
518
519         dataframe_state.index_col_to_input_e
ntities[(index, col_name1)] = generated_entity
520         self.global_state.create_derivation
(used_ent=used_entity['id'], gen_ent=generated_entity['i
d'])
521
522         if len(generated_features) > 0:
523             self.logger.info(f' Feature rename detect:
{feature_mapping}')
524             act_id = self.global_state.create_activity(f
unction_name, used_features=list(used_features),
525             escription=self.global_state.description,
526             ode=self.global_state.code,
527             ode_line=self.global_state.code_line,
528             enerated_features=list(generated_features),
529             racker_id=dataframe_state.tracker_id)
530
531             self.global_state.create_relation(act_id=act
_id, generated=generated_entities, used=used_entities,
532             d=[],
533             same=True)
534
535             return feature_mapping
536
537             @suppress_tracking
538             @timing
539             def __get_prov_value_change(self, dataframe_state, e
xtra_used_features: set = None) -> None:
540
541                 """
542                 Captures the provenance related to a change in t
he values of the DataFrame.
543                 The type of generated activity can be of two typ
es:
544                 Value Transformation: generic case.
545                 Imputation: the DataFrame column has undergone a
replacement of null values.
546                 Ordinal Encoding: the DataFrame column has been
encoded.
547
548                 :param dataframe_state: Input and output DataFra
me state.
549                 :param extra_used_features: Extra features used
in the input. Indicates additional features that contri
bute to the generation of new entities.
550                 :return: None
551
552                 """
553
554                 self.logger.info(f' Check for value change...')

```

```

527
528     function_name1 = "Value Transformation"
529     function_name2 = "Imputation"
530
531     df_input = dataframe_state.df_input_copy
532     df_output = dataframe_state.df_output
533
534     int_columns = df_output.columns.intersection(df_
input.columns)
535
536     generated_entities = []
537     used_entities = []
538     extra_used_entities = []
539     imp_cols = set()
540     trans_cols = set()
541
542     values_output = df_output[int_columns].to_numpy
()
543
544     if extra_used_features is None:
545         used_features = set()
546
547     for i in df_output.index:
548         index = df_output.index.get_loc(i)
549
550         for col_name in int_columns:
551             col = int_columns.get_loc(col_name)
552             new_value = values_output[index][col]
553
554             used_entity = dataframe_state.index_col_
to_input_entities.get((index, col_name), None)
555
556             if used_entity is None:
557                 continue
558
559             old_value = used_entity['value']
560
561             if new_value != old_value:
562                 if pd.isnull(old_value) and pd.isnul
l(new_value):
563                     continue
564                 elif pd.isnull(old_value):
565                     imp_cols.add(col_name)

```

```

555
556     function_name1 = "Value Transformation"
557     function_name2 = "Imputation"
558     function_name3 = "Ordinal Encoding"
559
560     df_input = dataframe_state.df_input_copy
561     df_output = dataframe_state.df_output
562
563     int_columns = df_output.columns.intersection(df_
input.columns)
564
565     generated_entities = []
566     used_entities = []
567     extra_used_entities = []
568     imp_cols = set()
569     trans_cols = set()
570     enc_cols = set()
571     values_output = df_output[int_columns].to_numpy
()
572
573     dropped_rows = df_input.index.difference(df_outp
ut.index)
574     drop_index = set()
575     for j in range(0, len(dropped_rows)):
576         drop_index.add(df_input.index.get_loc(droppe
d_rows[j]))
577     drop_mod = sum(j < df_input.index for j in drop_
index)
578     drop_mod = np.delete(drop_mod, list(drop_index))
579
580     if extra_used_features is None:
581         used_features = set()
582
583     for i in df_output.index:
584         index = df_output.index.get_loc(i)
585
586         if len(drop_index) > 0:
587             index_in = index + drop_mod[index]
588         else:
589             index_in = index
590
591         for col_name in int_columns:
592             col = int_columns.get_loc(col_name)
593             new_value = values_output[index][col]
594
595             used_entity = dataframe_state.index_col_
to_input_entities.get((index_in, col_name), None)
596
597             if used_entity is None:
598                 continue
599
600             old_value = used_entity['value']
601
602             if new_value != old_value:
603                 if pd.isnull(old_value) and pd.isnul
l(new_value):
604                     continue
605                 elif pd.isnull(old_value):
606                     imp_cols.add(col_name)
607                 elif type(old_value) == str and isin
stance(new_value, Number):
608                     if new_value.is_integer():
609                         if old_value.isdigit():
610                             if int(old_value) == new
_value:
611                                 continue
612
613

```

```

566         else:
567             trans_cols.add(col_name)
568
569             entity = self.global_state.create_entity(
570                 value=new_value, feature_name=col_name, index=index,
571                 instance=self.global_state.operation_number)
572             dataframe_state.index_col_to_input_entities[
573                 (index, col_name)] = entity
574             generated_entities.append(entity['id'])
575             used_entities.append(used_entity['id'])
576
577             self.global_state.create_derivation(
578                 (used_ent=used_entity['id'], gen_ent=entity['id']))
579             # Extra features to add
580             for used_feature in used_features:
581                 used_entity = dataframe_state.index_col_to_input_entities.get(
582                     (index, used_feature), None)
583                 if used_entity is None:
584                     continue
585                 extra_used_entities.append(used_entity)
586
587             self.global_state.create_derivation(
588                 (used_ent=used_entity['id'], gen_ent=entity['id']))
589             imp_cols = imp_cols.difference(trans_cols)
590

```

```

591         if len(imp_cols) > 0:

```

```

614         else:
615             enc_cols.add(col_name)
616
617         else:
618             trans_cols.add(col_name)
619
620             entity = self.global_state.create_entity(
621                 value=new_value, feature_name=col_name, index=index,
622                 instance=self.global_state.operation_number)
623             dataframe_state.index_col_to_input_entities[
624                 (index, col_name)] = entity
625             generated_entities.append(entity['id'])
626             used_entities.append(used_entity['id'])
627
628             self.global_state.create_derivation(
629                 (used_ent=used_entity['id'], gen_ent=entity['id']))
630             # Extra features to add
631             for used_feature in used_features:
632                 used_entity = dataframe_state.index_col_to_input_entities.get(
633                     (index, used_feature), None)
634                 if used_entity is None:
635                     continue
636                 extra_used_entities.append(used_entity)
637
638             self.global_state.create_derivation(
639                 (used_ent=used_entity['id'], gen_ent=entity['id']))
640             imp_cols = imp_cols.difference(trans_cols)
641
642             if len(enc_cols) > 0:
643                 self.logger.info(f' Encoding detect on {enc_cols} columns')
644                 enc_cols = enc_cols.union(used_features)
645                 extra_used_entities.extend(used_entities)
646                 act_id = self.global_state.create_activity(
647                     function_name=function_name3, used_features=list(enc_cols),
648                     description=self.global_state.description,
649                     code=self.global_state.code,
650                     code_line=self.global_state.code_line,
651                     tracker_id=dataframe_state.tracker_id)
652                 self.global_state.create_relation(
653                     act_id=act_id, generated=generated_entities,
654                     used=used_entities if len(extra_used_entities) == 0 else
655                     extra_used_entities, invalidate_d=None, same=len(
656                     extra_used_entities) == len(used_entities))

```

```

656         if len(imp_cols) > 0:

```

```

592         self.logger.info(f' Imputation detect on {im
p_cols} columns')
593         act_id = self.global_state.create_activity(f
unction_name=function_name2, used_features=list(imp_col
s),
594                                                     d
escription=self.global_state.description,
595                                                     c
ode=self.global_state.code,
596                                                     c
ode_line=self.global_state.code_line,
597                                                     t
racker_id=dataframe_state.tracker_id)
598         self.global_state.create_relation(act_id=act
_id, generated=generated_entities, used=None,
599                                           invalidate
d=None, same=True)
600
601         if len(trans_cols) > 0:
602             self.logger.info(f' Value transformation det
ect on {trans_cols} columns')
603             trans_cols = trans_cols.union(used_features)
604             extra_used_entities.extend(used_entities)
605             act_id = self.global_state.create_activity(f
unction_name=function_name1, used_features=list(trans_co
ls),
606                                                     d
escription=self.global_state.description,
607                                                     c
ode=self.global_state.code,
608                                                     c
ode_line=self.global_state.code_line,
609                                                     t
racker_id=dataframe_state.tracker_id)
610             self.global_state.create_relation(act_id=act
_id, generated=generated_entities,
611                                               used=used_
entities if len(
612                                               extra_
used_entities) == 0 else extra_used_entities,
613                                               invalidate
d=None, same=len(extra_used_entities) == len(used_entiti
es))
614         else:
615             self.logger.info(f' Value transformation and
Imputation not detected')
616
617         @suppress_tracking
618         @timing
619         def check_equals_dataframe(self, feature_mapping: di
ct)-> bool:
620             """
621             Check if the df_input and df_output dataframes a
re equal.
622             """
623
624             function_name = "Check Equals Dataframe"
625
626             result = False
627             hash_df_output = self.hash_df_output.copy()
628             hash_df_output.rename(feature_mapping)
629             if self._df_output is not None and self.hash_df_
output is not None:

```

```

657         self.logger.info(f' Imputation detect on {im
p_cols} columns')
658         act_id = self.global_state.create_activity(f
unction_name=function_name2, used_features=list(imp_col
s),
659                                                     d
escription=self.global_state.description,
660                                                     c
ode=self.global_state.code,
661                                                     c
ode_line=self.global_state.code_line,
662                                                     t
racker_id=dataframe_state.tracker_id)
663         self.global_state.create_relation(act_id=act
_id, generated=generated_entities, used=None,
664                                           invalidate
d=None, same=True)
665
666         if len(trans_cols) > 0:
667             self.logger.info(f' Value transformation det
ect on {trans_cols} columns')
668             trans_cols = trans_cols.union(used_features)
669             extra_used_entities.extend(used_entities)
670             act_id = self.global_state.create_activity(f
unction_name=function_name1, used_features=list(trans_co
ls),
671                                                     d
escription=self.global_state.description,
672                                                     c
ode=self.global_state.code,
673                                                     c
ode_line=self.global_state.code_line,
674                                                     t
racker_id=dataframe_state.tracker_id)
675             self.global_state.create_relation(act_id=act
_id, generated=generated_entities,
676                                               used=used_
entities if len(
677                                               extra_
used_entities) == 0 else extra_used_entities,
678                                               invalidate
d=None, same=len(extra_used_entities) == len(used_entiti
es))
679         if len([enc_cols, imp_cols, trans_cols]) == 0:
680
681             self.logger.info(f' Value transformation, En
coding and Imputation not detected')
682
683         @suppress_tracking
684         @timing
685         def check_equals_dataframe(self, feature_mapping: di
ct)-> bool:
686             """
687             Check if the df_input and df_output dataframes a
re equal.
688             """
689
690             function_name = "Check Equals Dataframe"
691
692             result = False
693             hash_df_output = self.hash_df_output.copy()
694             hash_df_output.rename(feature_mapping)
695             if self._df_output is not None and self.hash_df_
output is not None:

```

```

631         result = self.hash_df_input.equals(hash_df_o
        utput)
632
633         if result:
634             self.logger.info(f' {function_name}: datafra
me are equals!')
635
636         return result
637
638     def __prepare_for_next_operation(self, dataframe_sta
te: DataFrameState, update_df_input: bool = True,
639                                     save: bool = True) ->
        None:
640         """
641         Prepare for the next operation.
642
643         """
644         self.global_state.description = None
645
646         dataframe_state.df_input = dataframe_state.df_ou
tput
647
648         if update_df_input:
649             dataframe_state.df_input_copy = dataframe_st
ate.df_input.copy()
650             dataframe_state.hash_df_input = dataframe_st
ate.hash_df_output
651
652             dataframe_state.hash_df_output = None
653
654         if save:
655             # Save to neo4j
656
657             session = Neo4jConnector().create_session()
658
659             self.neo4j.create_constraint(session=sessio
n)
660
661             self.neo4j.add_activities(self.global_state.
current_activities, session)
662             self.neo4j.add_entities(self.global_state.cu
rrent_entities)
663             self.neo4j.add_derivations(self.global_stat
e.current_derivations)
664             self.neo4j.add_relations(self.global_state.c
urrent_relations)
665
666             if self.global_state.last_activities:
667                 next_operations = [{'act_in_id': a_in['i
d'], 'act_out_id': a_out['id']}
668                                     for a_out in self.glo
bal_state.current_activities
669                                     for a_in in self.glob
al_state.last_activities
670                                     if a_out['tracker_i
d'] == a_in['tracker_id']]
671                 self.neo4j.add_next_operations(next_oper
ations, session)
672
673                 self.global_state.last_activities = self.glo
bal_state.current_activities.copy()
674
675             # Free memory
676             del self.global_state.current_activities[:]
677             del self.global_state.current_entities[:]
678             del self.global_state.current_derivations[:]
679             del self.global_state.current_relations[:]
680
681

```

```

697         result = self.hash_df_input.equals(hash_df_o
        utput)
698
699         if result:
700             self.logger.info(f' {function_name}: datafra
me are equals!')
701
702         return result
703
704     def __prepare_for_next_operation(self, dataframe_sta
te: DataFrameState, update_df_input: bool = True,
705                                     save: bool = True) ->
        None:
706         """
707         Prepare for the next operation.
708
709         """
710         self.global_state.description = None
711
712         dataframe_state.df_input = dataframe_state.df_ou
tput
713
714         if update_df_input:
715             dataframe_state.df_input_copy = dataframe_st
ate.df_input.copy()
716             dataframe_state.hash_df_input = dataframe_st
ate.hash_df_output
717
718             dataframe_state.hash_df_output = None
719
720         if save:
721             # Save to neo4j
722
723             session = Neo4jConnector().create_session()
724
725             self.neo4j.create_constraint(session=sessio
n)
726
727             self.neo4j.add_activities(self.global_state.
current_activities, session)
728             self.neo4j.add_entities(self.global_state.cu
rrent_entities)
729             self.neo4j.add_derivations(self.global_stat
e.current_derivations)
730             self.neo4j.add_relations(self.global_state.c
urrent_relations)
731
732             if self.global_state.last_activities:
733                 next_operations = [{'act_in_id': a_in['i
d'], 'act_out_id': a_out['id']}
734                                     for a_out in self.glo
bal_state.current_activities
735                                     for a_in in self.glob
al_state.last_activities
736                                     if a_out['tracker_i
d'] == a_in['tracker_id']]
737                 self.neo4j.add_next_operations(next_oper
ations, session)
738
739                 self.global_state.last_activities = self.glo
bal_state.current_activities.copy()
740
741             # Free memory
742             del self.global_state.current_activities[:]
743             del self.global_state.current_entities[:]
744             del self.global_state.current_derivations[:]
745             del self.global_state.current_relations[:]
746
747

```



```

682     def create_tracked_dataframe(self, df: pd.DataFrame,
683     tracker_id: str):
684         """
685         Create a tracked dataframe.
686         """
687
688         class DataFrameTracked(pd.DataFrame, metaclass=T
689         rackedDataFrameMeta, tracker=self, tracker_id=tracker_i
690         d):
691             pass
692
693             return DataFrameTracked(df)
694
695         def subscribe(self, df: Union[pd.DataFrame, List[pd.
696         DataFrame]]) -> List[pd.DataFrame]:
697
698             if isinstance(df, pd.DataFrame):
699                 # Case where only one dataframe is passed
700                 tracker_id = str(uuid.uuid4())
701                 dataframe_state = DataFrameState(tracker_id=
702                 tracker_id)
703                 df_tracked = self.create_tracked_dataframe(d
704                 f=df, tracker_id=tracker_id)
705                 dataframe_state.df_input = df_tracked
706                 dataframe_state.df_input_copy = df_tracked.c
707                 opy()
708                 self.global_state.add_dataframes(df_tracked.
709                 tracker_id, dataframe_state)
710                 self.global_state.init_prov_entities(tracker
711                 _id=df_tracked.tracker_id)
712                 return [df_tracked]
713
714             elif isinstance(df, list):
715                 # Case where a list of dataframes is passed
716                 tracked_dfs = []
717                 for single_df in df:
718                     tracker_id = str(uuid.uuid4())
719                     dataframe_state = DataFrameState(tracker
720                     _id=tracker_id)
721                     df_tracked = self.create_tracked_datafra
722                     me(df=single_df, tracker_id=tracker_id)
723                     dataframe_state.df_input = df_tracked
724                     dataframe_state.df_input_copy = df_track
725                     ed.copy()
726                     self.global_state.add_dataframes(df_trac
727                     ked.tracker_id, dataframe_state)
728                     self.global_state.init_prov_entities(trac
729                     ker_id=df_tracked.tracker_id)
730                     tracked_dfs.append(df_tracked)
731                 return tracked_dfs
732
733             else:
734                 raise ValueError("Invalid input format. Expe
735                 cted a single DataFrame or a list of DataFrames.")
736
737         class TrackedDataFrameMeta(type):
738             """
739             Defines the metaclass for the DataFrameTracked

```

```

748     def create_tracked_dataframe(self, df: pd.DataFrame,
749     tracker_id: str):
750         """
751         Create a tracked dataframe.
752         """
753
754         class DataFrameTracked(pd.DataFrame, metaclass=T
755         rackedDataFrameMeta, tracker=self, tracker_id=tracker_i
756         d):
757             pass
758
759             return DataFrameTracked(df)
760
761         def subscribe(self, df: Union[pd.DataFrame, List[pd.
762         DataFrame]]) -> List[pd.DataFrame]:
763
764             if isinstance(df, pd.DataFrame):
765                 # Case where only one dataframe is passed
766                 tracker_id = str(uuid.uuid4())
767                 dataframe_state = DataFrameState(tracker_id=
768                 tracker_id)
769                 df_tracked = self.create_tracked_dataframe(d
770                 f=df, tracker_id=tracker_id)
771                 dataframe_state.df_input = df_tracked
772                 dataframe_state.df_input_copy = df_tracked.c
773                 opy()
774                 self.global_state.add_dataframes(df_tracked.
775                 tracker_id, dataframe_state)
776                 self.global_state.init_prov_entities(tracker
777                 _id=df_tracked.tracker_id)
778                 return [df_tracked]
779
780             elif isinstance(df, list):
781                 # Case where a list of dataframes is passed
782                 tracked_dfs = []
783                 for single_df in df:
784                     tracker_id = str(uuid.uuid4())
785                     dataframe_state = DataFrameState(tracker
786                     _id=tracker_id)
787                     df_tracked = self.create_tracked_datafra
788                     me(df=single_df, tracker_id=tracker_id)
789                     dataframe_state.df_input = df_tracked
790                     dataframe_state.df_input_copy = df_track
791                     ed.copy()
792                     self.global_state.add_dataframes(df_trac
793                     ked.tracker_id, dataframe_state)
794                     self.global_state.init_prov_entities(trac
795                     ker_id=df_tracked.tracker_id)
796                     tracked_dfs.append(df_tracked)
797                 return tracked_dfs
798
799             else:
800                 raise ValueError("Invalid input format. Expe
801                 cted a single DataFrame or a list of DataFrames.")
802
803         class TrackedDataFrameMeta(type):
804             """
805             Defines the metaclass for the DataFrameTracked

```

```

736
737     """
738
739     def __new__(cls, name, bases, dct, tracker, tracker_
id: str):
740         """
741         Every method (except exceptions) will be encapsu
lated.
742         The wrapper function will take care of tracking
the provenance
743
744         """
745
746         child = super().__new__(cls, name, bases, dct)
747
748         setattr(child, 'tracker_id', tracker_id)
749
750         exceptions = ['__init__', '_constructor_sliced',
'_get_item_cache', '_clear_item_cache', '_ixs',
751                     '_box_col_values', 'iterrows', '__
repr__', '_info_repr', 'to_string', '__len__', 'itertupl
es',
752                     'to_dict', '__getitem__', '_maybe_
cache_changed', '_append', '_set_item', '_sanitize_colum
n',
753                     '_ensure_valid_index', '_set_item_
mgr', '_iset_item_mgr', '_cmp_method', '_dispatch_frame_
op',
754                     '_construct_result', '_setitem_fra
me', 'isna', 'to_numpy', 'values', 'corr', 'isnull', 'nu
nique',
755                     'select_dtypes', 'items']
756
757         for base in bases:
758             for field_name, field in base.__dict__.items
():
759                 if callable(field):
760                     if field_name not in exceptions and
not isinstance(field, Iterable):
761                         setattr(child, field_name, track
er._wrapper_track_provenance(field, tracker_id))
762
763         return child
764

```

```

802
803     """
804
805     def __new__(cls, name, bases, dct, tracker, tracker_
id: str):
806         """
807         Every method (except exceptions) will be encapsu
lated.
808         The wrapper function will take care of tracking
the provenance
809
810         """
811
812         child = super().__new__(cls, name, bases, dct)
813
814         setattr(child, 'tracker_id', tracker_id)
815
816         exceptions = ['__init__', '_constructor_sliced',
'_get_item_cache', '_clear_item_cache', '_ixs',
817                     '_box_col_values', 'iterrows', '__
repr__', '_info_repr', 'to_string', '__len__', 'itertupl
es',
818                     'to_dict', '__getitem__', '_maybe_
cache_changed', '_append', '_set_item', '_sanitize_colum
n',
819                     '_ensure_valid_index', '_set_item_
mgr', '_iset_item_mgr', '_cmp_method', '_dispatch_frame_
op',
820                     '_construct_result', '_setitem_fra
me', 'isna', 'to_numpy', 'values', 'corr', 'isnull', 'nu
nique',
821                     'select_dtypes', 'items']
822
823         for base in bases:
824             for field_name, field in base.__dict__.items
():
825                 if callable(field):
826                     if field_name not in exceptions and
not isinstance(field, Iterable):
827                         setattr(child, field_name, track
er._wrapper_track_provenance(field, tracker_id))
828
829         return child
830

```