PD-Explain: A Unified Python-native Framework for Query Explanations Over DataFrames

Itay Elyashiv
Bar-Ilan University
elyashi1@biu.ac.il

Amir Gilad
Hebrew University
amirg@cs.huji.ac.il

Edna Isakov
Bar-Ilan University
edna.isakov@live.biu.ac.il

Tal Tikochinsky
Bar-Ilan University
tal.tikochinsky@live.biu.ac.il

Amit Somech
Bar-Ilan University
somecha@cs.biu.ac.il

ABSTRACT

Interfaces that rely on the Python programming language have become a popular tool for data analysis and exploration. In particular, the Pandas library allows users to query, manipulate, and visualize data in an easy and intuitive manner. However, users who perform such manipulations over the data in the exploratory process may struggle to justify their results, or understand which part (if any) of the obtained results is interesting and why. To handle such scenarios we developed PD-Explain, a Python library that adapts multiple prevalent query explanation approaches from the literature, and makes them accessible to Pandas users. PD-Explain is seamlessly integrated with Pandas and contains explanation functions that users can employ to choose the explanation approach they wish to use along with the necessary parameters in order to get the explanation in the suitable form. PD-Explain further allows users to automatically detect the interesting parts of a query result and get a visualization of the explanation accompanied by a Natural Language description. Our demonstration will include four different types of query result explanations and three real-world datasets with appropriate analysis tasks that will highlight the intuitive nature and usefulness of PD-Explain in data exploration tasks.

PVLDB Reference Format:
Itay Elyashiv, Amir Gilad, Edna Isakov, Tal Tikochinsky, and Amit Somech. PD-Explain: A Unified Python-native Framework for Query Explanations Over DataFrames. PVLDB, 17(12): 4473 - 4476, 2024.
doi:10.14778/3685800.3685903

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/analysis-bots/pd-explain.

1 INTRODUCTION

Exploratory Data Analysis (EDA) is an essential process performed by data scientists and analysts in order to examine a new dataset up-close, better understand its nature and characteristics, and extract insights from it. In recent years, many data analysts and scientists choose to perform EDA processes using programmatic manipulation of Dataframe objects (essentially, database tables and views), with libraries such as `pandas` [6]. Pandas offers a flexible, programmatic interface in Python, used for querying and transforming tabular data. When used in an interactive Python interface, such as VSCode or a Jupyter notebook, users can execute analytical code, generate new Dataframes, and add any form of text they wish.

However, one of the major challenges when analyzing a new dataset is drawing conclusions from the results of analytical operations (e.g., filter, group-by-aggregate). This is often done manually, by employing data visualization, adding a textual explanation, or by subsequent operations employed in attempt to interpret the results implicitly. All of these approaches depend heavily on the understanding and skill level of the analyst performing the data analysis, which can vary widely.

Numerous solutions for explaining query results have been suggested in previous work, each addressing a different aspect of the problem, such as outliers in aggregated queries [11], query results interestingness [2], and quantifying the contribution of tuples to the result [1]. Many of these solutions require a dedicated interface or use an SQL engine to query the data. Hence, using them may involve the process of exporting the data, and learning a specific UI, tailored to the scenario covered by the solution.

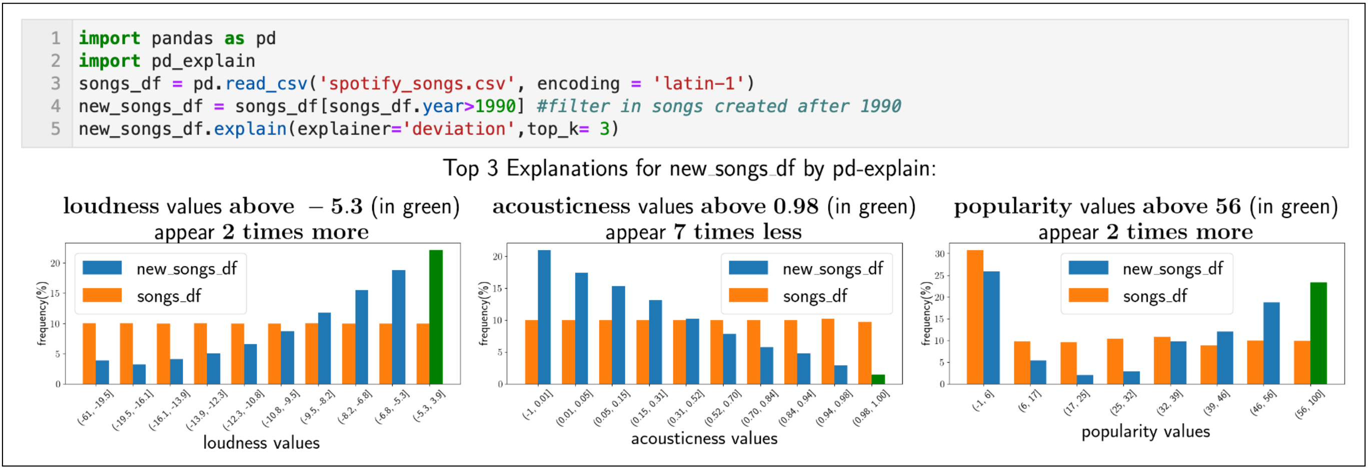
We propose a demonstration of PD-Explain[1]*, a unified framework for in-situ Dataframe explanations.* PD-Explain is a wrapper for Python's Pandas[2], allowing users to obtain explanations for queries over Dataframes without deferring to external tools. PD-Explain is implemented as a Python package and can be easily called and used by analysts to obtain different forms of explanations for their query results. Our current implementation supports three approaches of explanations inspired by prior work. In particular, the package allows users to get explanations inspired by a recent work about EDA explanations called FEDEX [2], a prominent work that suggests explanations for outliers, called SCORPION [11], and a novel work on query result explanations through Shapley values by Deutch et. al. [1]. PD-Explain automatically presents the explanation according to the appropriate format, in the form of a custom visualization accompanied by a textual description.

EXAMPLE 1. *Figures 1 and 2 depict an example analytical workflow using the Spotify dataset*[3]*, assisted by PD-Explain.*

The user first loads the dataset using the Pandas `read_csv()` *command and stores it in a DataFrame (*`songs_df`*) (Lines 1–3). To focus on newer songs, a filter operation on 'Year' > 1990 is applied, resulting*

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685903

[1]https://github.com/analysis-bots/pd-explain.
[2]https://github.com/pandas-dev/pandas.
[3]https://www.kaggle.com/mrmorj/dataset-of-songs-in-spotify.



```
1 import pandas as pd
2 import pd_explain
3 songs_df = pd.read_csv('spotify_songs.csv', encoding = 'latin-1')
4 new_songs_df = songs_df[songs_df.year>1990] #filter in songs created after 1990
5 new_songs_df.explain(explainer='deviation',top_k= 3)
```

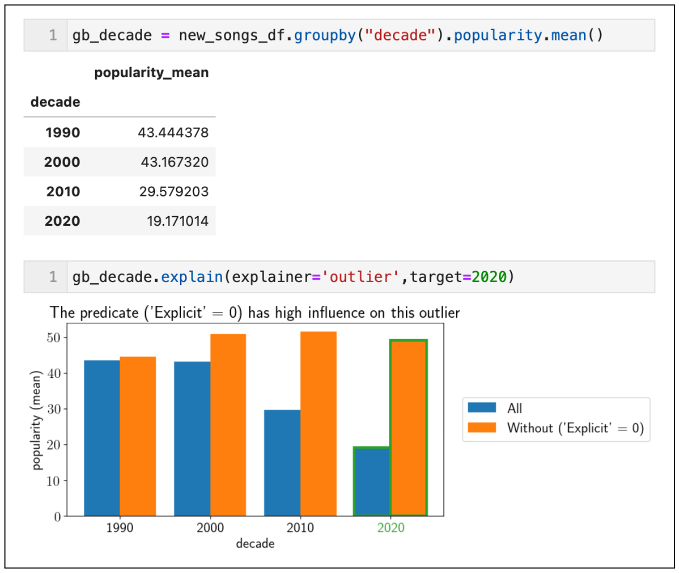
Figure 1: Example Exploration Workflow with PD-Explain The user loads the Spotify Songs dataset to a Dataframe, then filters it according to 'Year' > 1990. The user then asks PD-Explain to display the top-3 'deviation' explanations for the resulting Dataframe, demonstrating interesting aspects in which the resulted Dataframe significantly *deviates* from the original one.

in a new DataFrame called `new_songs_df` *(Line 4). The user then asks PD-Explain to explain "what is interesting" in the resulting DataFrame by using the FEDEX [2] explainer (Line 5). PD-Explain detects three useful explanations, showing that newer songs (in* `new_songs_df`*) are louder, less acoustic, and more popular compared to all songs (*`songs_df`*). Each plot is accompanied by a caption that highlights the interesting phenomenon.*

Next, as depicted in Figure 2, the user employs a group-by operation on the new songs to examine the mean popularity by decade. Learning that newer songs tend to be more popular, the user is surprised to see that the average popularity of songs produced in the current decade is significantly lower than that of songs from 1990 to 2020. To explain this outlier, PD-Explain is used again, now asking for an outlier explanation (generated based on [11]). PD-Explain detects that the predicate 'Explicit' = 0 (i.e., songs that do not contain explicit lyrics) can explain the outlier, since when omitted, the mean popularity of 2020s songs (and even more so, 2010s songs) is on par with the other decades. The underlying conclusion here is that non-explicit songs are significantly less popular than explicit ones in recent decades, thus pulling down the popularity mean.

PD-Explain features two more explanation types, for high-variance group-by-and-aggregate queries and for Boolean queries. We further plan to add more explanation approaches to PD-Explain, making it a universal platform for Dataframe query explanations.

PD-Explain "wraps" the pandas Dataframe objects in order to record the queries performed by the user, as well as to allow the user to obtain explanations from within the Dataframes – without interfering with Pandas functionality. Specifically, PD-Explain has two main components: the Dataframe Wrapper and the State Manager. The former component carefully integrates with Pandas by creating a new class called 'ExpDataframe'. The new class overloads all necessary data operations in the original 'Dataframe' class, and effectively adapts to its various data load and create functions. Our Wrapper also successfully handles group-by operations, in which Pandas creates an additional, intermediate object before returning the aggregated results. Second, the State Manager component tracks the user's queries in the Dataframes. Rather than naively



```
1 gb_decade = new_songs_df.groupby("decade").popularity.mean()
```

decade	popularity_mean
1990	43.444378
2000	43.167320
2010	29.579203
2020	19.171014

```
1 gb_decade.explain(explainer='outlier',target=2020)
```

Figure 2: Example Outlier Explanation in PD-Explain

keeping all previous Dataframes and queries, the State Manager efficiently stores only the required data to generate explanations: the query details, attributes, and pointers to the input Dataframes.

Related Work. A plethora of explanation tools are suggested in previous work, e.g., [1, 2, 7–9, 11]. However, most assume an SQL interface, rather than interactive Dataframe exploration in Python. LUX [5] is a prominent, recent system for Dataframe visualization. PD-Explain complements LUX by focusing on query explanations, which require additional constructs for tracing the origin of resulted Dataframes. Closer to this demonstration our previous works [2, 3] focus on explanations in EDA notebooks, yet not particularly suited for pandas Dataframes. PD-Explain generalizes these works by introducing a unified, seamless framework for multiple query explanation types, adapted to the use-case of Dataframe exploration in one interface.

2 THE PD-EXPLAIN FRAMEWORK

We next describe our general approach and the manner in which it is adapted to different explanation approaches.

2.1 Dataframe Explanation Framework

We begin by describing our model for Dataframe exploration, then the abstract components required to generate query explanations using various *Explainers* implemented in PD-Explain.

A model for Dataframe Exploration workflows. An interactive EDA process using a Python Dataframe interface is modelled as follows. First, the user loads one or more data tables from external sources (i.e., a CSV file, a spreadsheet, or an SQL query from a database server) into an interactive Python interface such as VSCode or a Jupyter notebook. Each loaded table is then represented as a *Dataframe object*, denoted D, a two-dimensional tabular structure comprising of a multi-set of rows $\mathcal{R}(D)$ over a schema $\mathcal{A}(D)$.

Then, in each step in the Dataframe exploration process, the user performs a data manipulation step $Q = (\mathcal{D}_{in}, q, D_{out})$: Employing a data manipulation procedure q, on one or more Dataframes, $\mathcal{D}_{in} = \{D^1_{in}, D^2_{in}, \ldots, D^k_{in}\}$, which generates an output Dataframe D_{out}. Importantly, an operation q can be employed on any set of output Dataframes $\{D^j_{out} \mid j < i\}$ generated in previous steps. For example, in Figure 1 (Line 4), the user employs a Pandas `filter` operation on the Dataframe `songs_df`, which generates the output Dataframe `new_songs_df`, containing songs produced after 1990.

PD-Explain supports various native Pandas operations including filter, join (merge), union, and group-by-and-aggregate. However, each explainer instance supports a different set of DataFrame operations, as detailed in Section 3. Explainers require arguments—some global, like k, and some unique, such as the outlier's deviation direction. Users may select a specific explainer and provide some, all, or none of its arguments; if not provided, PD-Explain will automatically select the appropriate explainer and infer parameters based on the operation type, results DataFrame, and additional metadata.

Problem definition (Intuitive). In essence, when a user employs a Dataframe operation $Q = (\mathcal{D}_{in}, q, D_{out})$, an *Explainer* detects rows in the input Dataframe(s) $\mathcal{D}_{in}$ that are highly influential with regard to an *interesting phenomenon* in the output Dataframe D_{out}. These Explainers can identify influential rows that cause, e.g., a significant deviation between an input and output Dataframe, presence of outliers, high variance, and suggest reasons for why a certain tuple of interest appears in D_{out}. PD-Explain currently implements explanation approaches that are inspired by a subset of the wide variety of existing approaches (e.g., [1, 2, 7–9, 11]) and we plan to implement more approaches in the future.

Each Explainer in PD-Explain implements a specific (1) notion for determining the *interestingness* of Q, and (2) notion for measuring the contribution of rows (and sets of rows) to the interestingness. PD-Explain then provides a unified manner for obtaining semantically-connected sets of rows, used with all Explainers, as described below. For a compelling presentation, each explainer also implements a visualization and description templates (See Section 2.3).

Interestingness measurement for D_{out}. Each Explainer in PD-Explain implements a function that quantifies an interesting phenomenon in the results data, used both for detecting such phenomena and for assessing the contribution of rows to it, as described next. PD-Explain supports both an overall assessment of interest $I(Q)$ as well as a column-level assessment $I_A(Q)$, since not all columns in the resulting Dataframe may showcase interesting phenomena. A plethora of interestingness measures exist in the literature (see [4] for a survey). For example, our *Outliers* Explainer uses the maximal standardized distance in a column, and our *Deviation* Explainer assesses the interestingness by measuring the change in a column's value distribution after employing a filter operation (see Section 2.2).

Measuring contribution of sets of rows. Each Explainer contains an additional method for assessing the contribution of rows R in an input Dataframe D_{in} to the interestingness score $I_A(Q)$. Various existing methods can be employed [1, 2, 7–9, 11] such as causality-based influence and intervention notions, which examine the result that would have been obtained had R not been in the data, as well as other approaches such as Shapley values [10].

Semantic partitioning of the input data. While any row or a combination of rows can be used in producing query explanation, PD-Explain focuses on detecting meaningful sets of rows that are *semantically related*, as we have detailed in [2]. This allows users to grasp high-level insights that characterize the examined operation, as well as to significantly reduce the number of candidate sets of rows when producing the explanations.

Given an input Dataframe $D_{in} \in \mathcal{D}_{in}$, a row partition divides D_{in} into p disjoint sets of rows D_{in}: $\mathcal{R}(D_{in}) = \{R_1, R_2, \ldots, R_p\}$ such that for all $R_i, R_j \in \mathcal{R}$, $\bigcup_{R_i \in \mathcal{R}} R_i = D_{in}$. PD-Explain utilizes attribute-level partition schemes $\mathcal{R}_1, \mathcal{R}_2, \ldots$ in which the rows are divided according to an attribute $A \in D_{in}$ (for all $D_{in} \in \mathcal{D}_{in}$) as follows: PD-Explain first detects the data type of A, and then employs all supported partitions. For instance, numeric attributes are used to partition the rows using binning methods such as equi-width, equi-height, and 1-d clustering. Date/Time columns divide the input rows according to several temporal hierarchies (i.e., by hour, week, month, etc.). PD-Explain features additional partition schemes, and also supports custom ones, defined by the user.

Generating explanations. When an Explainer is called w.r.t. an operation $Q = (\mathcal{D}_{in}, q, D_{out})$, PD-Explain first provides it with all sets of rows from its partitioning schemes on $\mathcal{D}_{in}$. It then calculates the *contribution* of each set of rows to the interestingness score of D_{out}, and returns the top-k most influential sets of rows (k can be set by the user). PD-Explain then generates a coherent, captioned visualization that summarizes the explanation, as described below.

2.2 Supported Explainers

PD-Explain currently supports four types of query explanations. **(1) Outlier Explainer [11]**. This Explainer supports group-by-and-aggregate Dataframe operations. The user first specifies an outlier tuple in D_{out}, as depicted in Figure 2, then our Outlier Explainer detects sets of rows in the input Dataframe, that when omitted – the outlier's deviation drops.The user is notified in case no such sets were found. The interestingness $I_A(Q)$ is calculated as the maximal standardized value, and rows contribution is measured using an *influence* assessment (See [11] for more details). **(2) Deviation Explainer [2]** This explainer supports filter (selection), join, and set (e.g., union, difference) Dataframe operations. It then detects sets of rows that highly contribute to deviation in value distributions resulted from the applied operation (See Figure 1 for an example). The column-level interestingness I_A is quantified by the distributional distance between $D_{out}[A]$ and its counterpart in an input Dataframe $D_{in}[A]$ (where $A \in \mathcal{A}(D_{in})$), using the two-sample Kolmogorov–Smirnov (KS) test. The contribution is measured via *standardized influence*, where the influence score of a given set of rows is standardized w.r.t. other sets in the same partition. **(3) Boolean Query Explainer [1]** This Explainer supports Boolean filter, join, and union operations.

In a similar manner, we intend to incorporate more Explainers in PD-Explain to further diversify its explanation abilities.

2.3 Library Implementation Details

PD-Explain is a Python library implemented as a transparent wrapper for `pandas`, allowing users to obtain query explanations using an `explain` procedure that can be simply called from any output Dataframe. To allow this seamless interface, PD-Explain introduces the following components:

Dataframe Wrapper. PD-Explain extends the `pandas` Dataframe object, overloading all of its existing methods. This is done using our `ExpDataFrame` class that contains modified versions of the Dataframe internal methods (e.g., filter, join). The `ExpDataFrame` object is initialized via any `pandas` 'read' function (e.g., `read_csv`), and is automatically created for every `pandas` operation employed by the user (thus, extending `GroupByDataFrame` and `Series` objects as well). The wrapper allows for executing the `explain` command from each Dataframe object, as well as for updating the State Manager, as described next.

State Management. Since `pandas` Dataframes only store the result data (rows and columns), PD-Explain needs to track the origin of the Dataframes, in order to provide the Explainers with the required meta-data needed to produce explanations. Therefore, our overloaded `pandas` methods update the State Manager with the operation type and parameters used to generate it, as well as pointers to the input Dataframes. This enables PD-Explain to calculate the contribution of sets of rows by simulating the Dataframe operation again after varying the input data.

Explanation Visualization Generator. The results of each Explainer contain sets of rows alongside their contribution scores. PD-Explain takes these results and generates a compelling data visualization alongside a Natural Language (NL) caption. Each Explainer implements visualization and NL caption templates, which are instantiated with concrete information at run time.

3 DEMO SCENARIO

VLDB participants are invited to employ PD-Explain on multiple available datasets, queries, and explanations approaches.

A walk-through of PD-Explain. We will start the demonstration by showing a pre-made Jupyter notebook containing an exploratory data analysis session of the Spotify dataset (shown in Figures 1 and 2), which will include the necessary code lines for importing the PD-Explain package, loading the dataset, performing queries, and obtaining explanations. In particular, we will emphasize that, because PD-Explain wraps the `pandas_read` function, we can simply use the pandas `read_csv` function to obtain an `ExpDataframe` of the Spotify dataset. We will guide users through these new abilities and show the different forms of supported explanations. We will then show the obtained explanations for all the currently supported approaches [1, 2, 11] and stress the differences between them.

An interactive exploration task. In the next stage of the demonstration, users will be invited to load one of the three datasets (or use their own). We have prepared different exploration questions for each dataset. An example question for the Spotify dataset can be: "How are popular songs different from other songs in the dataset?" Or for the Adults dataset: "What are the main characteristics of high-income individuals?" We will allow users to explore the data on their own and experiment with the different functions of PD-Explain to answer the question while providing hints and guidance as they go. We will encourage users to use the different explanation functions to obtain necessary insights in order to make headway in answering the questions.

Under the hood. We will demonstrate how PD-Explain adapts prominent query result explanation approaches to the dataframe setting. Additionally, we will show how dataframe queries are simulated and how PD-Explain detects and utilizes interesting parts of their result sets in the explanation generation process.

REFERENCES

[1] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. 2022. Computing the Shapley Value of Facts in Query Answering. In *SIGMOD*. 1570–1583.

[2] Daniel Deutch, Amir Gilad, Tova Milo, Amit Mualem, and Amit Somech. 2022. FEDEX: An Explainability Framework for Data Exploration Steps. *PVLDB* 15, 13 (2022), 3854–3868.

[3] Daniel Deutch, Amir Gilad, Tova Milo, and Amit Somech. 2020. ExplainED: explanations for EDA notebooks. *PVLDB* 13, 12 (2020), 2917–2920.

[4] Liqiang Geng and Howard J Hamilton. 2006. Interestingness measures for data mining: A survey. *CSUR* (2006).

[5] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *PVLDB* 15, 3 (2021), 727–738.

[6] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *SciPyi Proceedings*. 51 – 56.

[7] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB* 4, 1 (2010), 34–45.

[8] Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. 2011. Tracing data errors with view-conditioned causality. In *SIGMOD*. 505–516.

[9] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *SIGMOD*. 1579–1590.

[10] LS SHAPLEY. 1953. A value for n-person games. *Contributions to the Theory of Games* 28 (1953), 307–317.

[11] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. *PVLDB* 6, 8 (2013), 553–564.