**TSEXPLAIN: Explaining Aggregated Time Series by Surfacing Evolving Contributors**

Yiru Chen  
*Columbia University*  
New York, New York  
yiru.chen@columbia.edu

Silu Huang  
*Microsoft Research*  
Redmond, Washington  
silu.huang@microsoft.com

**Abstract**—Aggregated time series are generated effortlessly everywhere, e.g., “total confirmed covid-19 cases since 2019” and “total liquor sales over time”. Understanding “how” and “why” these key performance indicators (KPI) evolve over time is critical to making data-informed decisions. Existing explanation engines focus on explaining one aggregated value or the difference between two relations. However, this falls short of explaining KPIs’ continuous changes over time. Motivated by this, we propose TSEXPLAIN, a system that explains aggregated time series by surfacing the evolving top contributors. Under the hood, we leverage priworks on two-relations diff as a building block and formulate a \( K \)-Segmentation problem to segment the time series such that each segment after segmentation shares consistent explanations, i.e., contributors. To quantify consistency in each segment, we propose a novel within-segment variance design that is explanation-aware; to derive the optimal \( K \)-Segmentation scheme, we develop an efficient dynamic programming algorithm. Experiments on synthetic and real-world datasets show that our explanation-aware segmentation can effectively identify evolving explanations for aggregated time series and outperform explanation-agnostic segmentation. Further, we proposed an optimal selection strategy of \( K \) and several optimizations to speed up TSEXPLAIN for interactive user experience, achieving up to 13× efficiency improvement.

I. INTRODUCTION

Time series data is prevalent across sectors ranging from finance, retail, IoT to DevOps. Time series analysis is crucial for uncovering insights from time series data and helping business users make data-informed decisions. A business analyst typically focuses on three questions: “what happened” to understand the changes in key performance indicator (KPI), “why happened” to reason why KPI changes, and “now what” [1] to guide what actions should be taken. “What” questions have been extensively studied both academia-wise [2] and industry-wise [3]–[5]. “Why” questions are starting to attract wide attentions [6]–[8]. Existing explanation engines focus on explaining (1) one aggregated value [5], [9] or (2) differences between two given relations: a test relation and a control relation [5]–[8], [10]–[19].

However, KPIs are typically monitored continuously, reporting some aggregated time series. Simply explaining one aggregated value overlooks the trend of time series, e.g. “why up/down”; merely focusing on its two endpoints and explaining their difference overlooks the evolving explanations in between. As evidence, although key influencer feature which explains the difference between two given relations is

---

**Disclaimer.** We remark that identifying the root cause of “why” questions in general is only plausible when combining human interpretations with tools [16]. Following the literature [6]–[8], [12], the explanation in our work does not equate to “cause”, instead it corresponds to the data slice that contributes most to the overall change (see Definition 3.1).

**Motivating Examples.** We now describe two application scenarios for explaining changes in continuously evolving KPIs.

- **COVID-19.** Figure 1a depicts the total number of covid-19 confirmed cases during year 2020. This time series is obtained by performing a groupby-aggregate query over the original table [21], which consists of attributes like state, total-confirmed-cases, and daily-confirmed-cases. By looking at Figure 1a, users can get an understanding of how the overall trend evolves over time. One natural follow-up question is “what makes the increase” – how did different states contribute to the increase as time went along? Manual drill-down and browsing are laborious and overwhelming especially when there is a large number of attributes and each attribute is of high cardinality. Figure 1b illustrates a

---

1Influencer corresponds to explanation in our notion.
sample drill-down view along attribute state: first, looking at these sampled ten states is already distracting, let alone full 50 states in the US; more importantly, it is still unclear what the answer to the above question is, though we do observe that each state contributes differently as time moves along. For instance, state=NY drives the initial outbreak in the US, while state=CA contributes most during the end of 2020.

- **Liquor-sales.** The liquor-sales time series corresponds to query 
  \[
  \text{SELECT date, SUM(Bottles_Sold) FROM Liquor}
  \]
  GROUP BY date, where each row in relation Liquor represents a liquor purchase transaction with attributes including date, Bottle_Volume(ml), Pack, Category_Name, Bottles_Sold and etc. Data analysts wonder why the total number of bottles sold has turned up since mid-January-2020 and how drinking behavior changes during the pandemic. As we will show in our experiment, it turns out that people favor large-pack liquor at the beginning of the pandemic, leading to a sharp sales increase of Pack=12, 24, and 48.

**Problem and Challenges.** These motivating examples can all be abstracted as the same problem: given a relation \( R \), a set of explain-by attributes from \( R \), and a time series aggregated from \( R \), identify the top explanations, i.e., conjunctions of predicates over explain-by attributes, that contribute to the changes in the given aggregated time series. Using this problem formulation, let us illustrate the motivating example of COVID-19. Given a relation COVID-19, where each row records the number of confirmed cases in a city on a particular date, a set of explain-by attributes, e.g., [state], and a time series aggregated from \( R \), e.g., Figure 1a corresponding to query "SELECT date, SUM(total-confirmed-cases) FROM COVID-19 GROUP BY date", our goal is to identify explanations, e.g., state=NY, that explains the continuous surge in Figure 1a. There are two main challenges in solving this problem: (a) explanations evolve over time; (b) interactivity is critical for data exploration and analytics.

**Challenge (a):** explanations tend to change over time. For instance, by looking at Figure 1b, we can observe that the increase in New York (NY) is the main reason of the total increase in Figure 1a during 2020-04 and 2020-05, while California (CA) is the driving force during December 2020. Having observed that explanations evolve over time, our first technical challenge lies in how to identify time periods with consistent explanations and how to derive explanations for each consistent time period.

**Challenge (b):** data analysts typically explore “what” and “why” questions iteratively to understand data and uncover insights. Studies [22] have shown that low latency is critical in fostering user interaction, exploration, and insight extraction. It is desired that each query, including “why” queries, get answered within a second for interactivity. Thus, how to reduce the latency for deriving explanations poses another challenge.

**Prior works.** “Why” questions are gradually gaining attraction both academia-wise and industrial-wise. However, instead of explaining the continuous changes in a time series, existing works focus on explaining either (1) one aggregated value [9], e.g., point \( p_1 \) in Figure 1a; or (2) the difference between two given relations [6], [8], [10], [15]–[17], e.g., a test relation and a control relation corresponding to point \( p_1 \) and \( p_2 \) respectively in Figure 1a. Reiterating the “Disclaimer” above, these tools can recommend and expedite answering “why” questions, but the human-in-the-loop interpretation is still required for true root cause analysis. To summarize, no prior works have studied the problem of explaining the evolving changes in time series as depicted in our motivating examples. Please refer to Section II for detailed comparison.

**Our Solution.** We propose TSEXPLAIN, a system to explain the continuous changes in aggregated time series. In TSEXPLAIN, given a relation, users can freely perform OLAP operations, including drill-down, roll-up, slicing, and dicing, and visualize what has happened to some KPI. To explain, users can then specify the time period they are interested in and a set of explain-by attributes based on their domain knowledge. For the COVID-19 example, TSEXPLAIN returns a trendline visualization in Figure 2, where the whole input time series get partitioned into a few non-overlapping time periods and each time period has its coherent top explanations. We can see that NY and WA are the main contributors to the case increase in the early stage, while CA, TX, IL, and FL become the main contributors in later 2020. Please refer to our Demo paper [23] for more tool interface details. This paper focuses on providing concrete technical details.

Technically, to tackle challenge (a) of deriving evolving explanations, we formulate a K-Segmentation problem, aiming to partition the input time period into \( K \) smaller time periods such that each time period shares consistent top explanations. Our main insight is that explanation-aware segmentation is better than existing explanation-agnostic segmentation [24]–[26] for our task of explaining aggregated time series. Please refer to Section II for more reasoning and Section VII-C for experimental comparison. We then develop a dynamic programming algorithm for solving this K-Segmentation problem. Furthermore, since \( K \) is hard to specify in practice, TSEXPLAIN employs “Elbow method” [27] for identifying the optimal \( K \). We demonstrate the effectiveness of our problem formulation with both synthetic and real-world datasets. As for challenge (b) of interactivity, we first analyze the complexity of each step in TSEXPLAIN, identifying the bottleneck in the whole pipeline. Next, we propose several optimizations for reducing the bottlenecks in TSEXPLAIN. In our experiments, TSEXPLAIN has successfully answered
all queries within one second and performs faster than all baselines.

**Contributions.** The contributions of this paper are as follows:
- Formulate $K$-Segmentation problem for explaining the continuous changes in an aggregated time series. (Section III)
- Propose a novel within-segment variance design that is explanation-aware to quantify consistency. (Section IV)
- Develop a dynamic program algorithm and several optimizations for improving efficiency. (Section V)
- Conduct experiments on synthetic and real-world datasets, demonstrating the effectiveness and efficiency of TSE-PLAIN. (Section VII)

## II. RELATED WORK

**Data Explanation** Existing data explanation engines mainly focuses on explaining (1) one aggregated value [5], [9] or (2) the difference between two relations [5]–[8], [10]–[19]. In academia, SmartDrillDown [9] explains one aggregated value by identifying explanations (called rules in SmartDrillDown) that have high aggregate value. IDIFF [10] identifies the differences between two instances of an OLAP cube. Scorpion [6] and Microbase [8] aim to find the difference between outlier and inlier data. RSEXplain [11] proposes an intervention-based framework to explain why SQL expression’s result is high (or low). X-Ray [7] tries to reveal the common properties among all incorrect triples versus correct triples. Abuzaid et al. [12] unified various explanation engines and abstracted out a logical operator called diff. The cascading analysts algorithm [13] provides top non-overlapping explanations accounting for the major difference of two specified sets. Li et al. [15] also compares two set differences but with augmented information from other related tables. In industry, Google Trend integrates an explanation component for single value and two relations; Tableau [3] provides Explain data [16] feature; PowerBI [4] supports functionality like Key Influencer [17]; startups like SisuData [18] is built to support “why” questions natively and scalably. However, all prior works fall short of explaining aggregated time series because (1) only explaining one aggregated value overlooks the trend of time series -“why up/down”; (2) only focusing on two set differences (i.e., two endpoints in time series) dismisses the explanation in between. Our TSE-PLAIN aims to identify the evolving explanations for aggregated time series over time.

**Time Series Segmentation** Time series segmentation has been studied for decades. We note that the word “segmentation” is somewhat overloaded in the literature. One line of segmentation works focuses on visual-based piecewise linear approximation. Specifically, [28], [29] use the sliding windows algorithm, which anchors the left point of a potential segment, then attempts to approximate the data to the right with increasing longer segments. Douglas et al., [30] and Ramer et al., [31] designs top-down algorithms to partition the visualization. [32] and [33] have used the bottom-up algorithm to merge from the finest segments. Keogh et al. [26] show that the bottom-up algorithm achieves the best results compared with sliding window and top-down, and further introduced a new online algorithm that combines the sliding window and bottom-up to avoid rescanning of the data when streaming.

Another line of work is semantic segmentation including FLUSS [24], [34], AutoPlait [35], NNSegment proposed in LimeSegment [25], which aims to divide a time series into internally consistent subsequences, e.g., segmenting the heartbeat cycles when a person switches from running to walking. Thus, these algorithms require an extra input called subsequence length, e.g., a heartbeat cycle. However, our task is to explain the trends in the aggregated time series (e.g., the trend of total covid cases) instead of explaining/identifying the periodic difference in one time-series instance. Hence, our segmentation does not rely on the period length.

To conclude, unlike all above, TSE-PLAIN is the first to segment time series based on segments’ explanations.

**Time-series ML Model Explanation** The time-series ML model takes a univariate or multivariate time series as input and outputs a prediction label. Unlike text or image models, there is limited literature on black box Time-series ML Model explainability. FIT [36] is an explainability framework that defines the importance of each observation based on its contribution to the black box model’s distributional shift. Similarly, Rooke et al. [37] extend FIT into WinIT, which measures the effect on the distribution shift of groups of observations. Labaien et al. [38] finds the minimum perturbation required to change a black box model output. Recently, LimeSegment [25] selects representative input time series segments as explanations for time-series classifier’s output. In these works, the “explain target” is the prediction label and the time series serves as the “explain feature”. Contrarily, the “explain target” in TSE-PLAIN is the up/down trend in an aggregated time series and the explain-by attributes are our “explain features”. Unlike explaining instance-level prediction of the black box ML model, TSE-PLAIN is aggregation-level explanation for white-box aggregation.

## III. PROBLEM OVERVIEW

In this section, we start with existing works on the two-relations diff and how they fall short of explaining aggregated time series, followed by our problem formulation: $K$-Segmentation.

### A. Background

1) **Two-Relations Diff:** Diff operator [12] focuses on identifying the difference between two relations and unifies a range of existing explanation engines [6]–[8], [11]. Given a text relation $R_t$ and a control relation $R_c$, the diff operator returns explanations describing how these two relations differ.

**Definition 3.1 (Explanation [12]):** Given a set of explain-by attributes $\mathcal{A}$, an explanation $E$ of order $\beta$ is defined as a conjunction of $\beta$ predicates, denoted as $E = (A_1=a_1\land\ldots\land A_\beta=a_\beta)$ where $A_i \in \mathcal{A}$.

Explain-by attributes $\mathcal{A}$ can be specified by users based on their domain knowledge; otherwise, dimension attributes from $R_t$ are used. Intuitively, an explanation $E$ corresponds to a data
slice satisfying the predicate, and this data slice contributes to the overall change between $R_t$ and $R_e$. To quantify how well an explanation $E$ explains the difference between $R_t$ and $R_e$, [12] proposes a difference metric abstraction, denoted as $\gamma(E)$. It encapsulates the semantics of many prior explanation engines [6]–[8], [11]. Commonly used $\gamma(E)$ include absolute-change, relative-change, risk-ratio. We focus on absolute-change in this paper. Other metrics can be applied similarly.

**Definition 3.2 (Absolute-change):** Given a test relation $R_t$, a control relation $R_e$, an aggregate function $f(M, R)$ on some measure attribute $M$ in relation $R$, and an explanation $E$, the absolute change refers to the absolute difference between $|f(M, R_t) - f(M, R_e)|$ before and after removing records that explanation $E$ corresponds to, i.e., $\gamma(E) = |\sum_{E} |f(M, R_t) - f(M, R_e)| - |\sum_{E} f(M, R_t) - \sum_{E} f(M, R_e)||$, where $\sum_{E} f(M, R_t)$ and $\sum_{E} f(M, R_e)$ denote records satisfying the predicate $E$ in relation $R_t$ and $R_e$ respectively.

As the name indicates, absolute-change only cares about the absolute change of $f(M, R_t) - f(M, R_e)$, no matter the change is an increase or decrease. To distinguish an increase from a decrease, we use $\tau(E)$ to denote the change effect caused by including data that $E$ corresponds to: intuitively, if including $E$ leads to an increase in $|f(M, R_t) - f(M, R_e)|$, $\tau(E) = +$; otherwise, $\tau(E) = -$.

**Definition 3.3 (Change Effect):** Following the setting in Definition 3.2, the change effect of an explanation $E$ is defined as $\tau(E) = sign(\sum_{E} |f(M, R_t) - f(M, R_e)| - |\sum_{E} f(M, R_t) - \sum_{E} f(M, R_e)|)$. Now we have described absolute-change as an example of $\gamma(E)$. With a difference metric $\gamma(E)$, we can then rank each candidate explanation and return top-m explanations with the highest $\gamma(E)$. However, these top-m explanations may contain overlapping records. Consequently, the effects of these records get duplicated, introducing bias in the top-m explanations. Alternatively, we can constrain these $m$ explanations to be non-overlapping and define top-m non-overlapping explanations as in Definition 3.5. Two explanations $E_1$ and $E_2$ are said to be non-overlapping if their correspondent records are non-overlapping in any relation $R$, i.e., $\sigma_{x \in R \cap \sigma_{y \in R}} = 0, \forall R.$

**Definition 3.4 (m Non-Overlapping Explanations):** Given a difference metric $\gamma(E)$ and an explanation order threshold $\beta$, let $E_m$ denote a set of $m$ non-overlapping explanations, i.e., $E_m = \{E_1, ... E_m\}$, where each $E_i$ has its order $\leq \beta$ and is non-overlapping with $E_j, \forall E_i, E_j \in E_m$.

**Definition 3.5 (Top-m Non-Overlapping Explanations):** Top-m non-overlapping explanations are defined as $E_m$ with the highest accumulative diff score, i.e., $E^* = \text{arg max}_{E \in E_m} \sum_{x \in E} \gamma(E)$.

The Cascading Analysts algorithm [13] is designed for returning top-m non-overlapping explanations. We will use the term top-explanation for simplicity whenever there is no ambiguity.

---

Example 1 (Two-Relations Diff): Consider the two points $p_1$ and $p_2$ in Figure 1a — the underlying data corresponds to $p_1$ and $p_2$ constitute a control relation $R_e$ and a test relation $R_t$ respectively. Two-relations diff aims to explain the difference between $R_e$ and $R_t$. First, users can specify a set of explain-by attributes, e.g., $\{\text{state, County}\}$. Take explanation $E = \{\text{state=CA}\}$ as an example. It is of order one, i.e., $b = 1$ and its difference score $\gamma(E)$ can be calculated as $\|p_2.v - p_1.v\| = \|p_1.v - p_1.v\|$ using absolute-change as shown in Figure 1a. We can then obtain Top-3 non-overlapping explanations for differing $R_t$ and $R_e$ using the Cascading Analysts algorithm [13] — $E^*_3 = \{E_1 = (\text{state=CA}), E_2 = (\text{state=TX}), E_3 = (\text{state=FL})\}$.

2) Aggregated Time Series: An aggregated time series refers to a special type of time series, where each point $p$ is associated with a timestamp $p.t$ and an aggregated value $p.v$, derived by aggregating all records at timestamp $p.t$. Essentially, an aggregated time series corresponds to the result of a group-by query. Consider a relation $R$ with $\{D_1\}$ dimension attributes and $\{M_j\}$ measure attributes, and a group-by query in the form of $\text{SELECT T, } f(M) \text{ FROM } R \text{ GROUP BY } T$, where $T$ denotes some time-related ordinal dimension ($T \in \{D_1\}$), and $f(M)$ is some aggregate function on measure $M \in \{M_j\}$. The query result can be denoted by an aggregated time series $ts$ with $f(M)$ over time $T$.

**Definition 3.6 (Aggregated Time Series):** An aggregated time series $ts$ over time $[t_1, t_n]$ is a series of points $\{p_1, ..., p_n\}$ ordered by time dimension $T$ and each point’s value $p_i.v$ is an aggregated number from a list of records with the same time $p_i.t$.

Data analysts often visualize aggregated time series to help understand data’s overall trend as time goes along, as shown in Figure 1a. A natural follow-up question is “what makes ups and downs”. Different from the two-relations diff described in Section III-A1, this “explain” question focuses on the whole time horizon and the underlying top-explanation tends to evolve dynamically along the time even when the overall trend looks the same visually.

**Definition 3.7 (Evolving Explanations):** Given $m$ and an aggregated time series $ts$ over time $[t_1, t_n]$, evolving explanations is a sequence of top-explanation at different periods, denoted as $E = [E_m(t_1, t_2), E_m(t_2, t_3), ..., E_m(t_c, t_{c+1})]$, where $c_1 = 1$, $c_{k+1} = n$. $\{c_2, c_3, c_4\}$ denote the $(k-1)$-cutting positions in between, and each $E_m(t_c, t_{c+1})$ denotes top-explanation from $t_c$ to $t_{c+1}$.

**Example 2 (Evolving Explanations):** Figure 1a depicts an aggregated time series that corresponds to a group-by-aggregate query with $f(M) = \text{SUM(total_confirmed_cases)}$ on table Covid-19. Figure 2 illustrates the underlying evolving explanations for the increase in Figure 1a. We have six different time periods, where each period shares the same intrinsic explanations while neighboring periods have different ones. For instance, the top-3 contributors are $E_2(t_3, t_4) = \{E_2 = (\text{state=NY}), E_3 = (\text{state=MA})\}$ during $t_3 = 3/14$ and $t_4 = 5/4$; $E_2(t_4, t_5) = \{E_2 = (\text{state=CA}), E_4 = (\text{state=TX}), E_3 = (\text{state=FL})\}$ from $11/27$ to $12/31$. 

---

Example 3.5 is defined over $E_m$. Alternatively, we can define top-m as at most $m$ explanations, i.e., $E^*_m = \text{arg max}_{E \subseteq E_m} \sum_{E \in E} \gamma(E)$. Our proposed solution in Section IV and V can work with it in a similar way.
B. Problem Definition

Motivated by the observation that the top contributor (i.e., explanation) evolves over time, we study the problem of identifying evolving explanations for the continuous changes happening in an aggregated time series. The overall problem of identifying evolving explanations can be decomposed into two sub-problems: (a) segmentation; (b) find the explanations that contribute most in each segment. These two sub-problems are intertwined with each other: the goodness of a segmentation scheme depends on how cohesive top-explanations are within each segment; meanwhile, top-explanations are derived for each segment after obtaining the optimal segmentation scheme. We remark that performing segmentation without considering explanation information is insufficient, as we will demonstrate experimentally in Section VII.

Segmentation. To explain the continuous change in an aggregated time series \( ts \) over time \([t_1, t_n]\), we need to partition the whole time domain \([t_1, t_n]\) into non-overlapping segments, such that each segment \( P_i = \{p_{c_i}, p_{c_{i+1}}\} \) during time \( t_{c_i} \) and \( t_{c_{i+1}} \), shares the same intrinsic explanations while neighboring segments have different ones. This resembles the well-studied clustering problem, which aims to minimize within-cluster variance and maximize inter-cluster variance. In particular, given a segment number \( K \), we abstract our problem as a K-Segmentation problem, adapting the optimization formula from K-Means [39]. Let \( \mathcal{P}_K \) denote a K-segmentation scheme: \( \mathcal{P}_K = \{P_1 = \{p_{c_1}, p_{c_2}\}, P_2 = \{p_{c_2}, p_{c_3}\}, \ldots, P_K = \{p_{c_K}, p_{c_{K+1}}\}\) where \( c_1 = 1, c_{K+1} = n \), and \( \{c_2, c_3, \ldots, c_K\} \) are \((K-1)\) cutting positions. In Example 2 (Figure 2), \( K = 6 \) and the cutting positions \( \{c_2, \ldots, c_5\} \) correspond to time \( \{3-14, 5-4, 5-29, 9-25, 11-27\} \). Next, we formulate K-Segmentation problem.

Problem 1 (K-Segmentation): Given an aggregated time series \( ts \) and a segmentation number \( K \), identify the optimal segmentation scheme \( \mathcal{P}_K = \arg\min_{\mathcal{P}_K} \sum_{i=1}^{K}|P_i|var(P_i) \) where \( var(P_i) \) denotes the variance in segment \( P_i \).

The design of within-segment variance \( var(P_i) \) is critical to the effectiveness of K-Segmentation. No prior works have studied \( var(P_i) \) with the goal of quantifying explanation consistency. We will dive into it in Section IV.

Explain trend in each segment. Given a fixed segment \( P_i = \{p_{c_i}, p_{c_{i+1}}\} \) from time \( t_{c_i} \) to \( t_{c_{i+1}} \), we will now describe how to explain the trend in \( P_i \) with only one static top-explanation \( \mathcal{E}_m(t_{c_i}, t_{c_{i+1}}) \) — static top-explanation is a special case of evolving explanations \( \mathcal{E} \) with segment number \( K = 1 \). If \( P_i \) is cohesive, meaning that \( P_i \) has consistent top-explanation during \( t_{c_i} \) and \( t_{c_{i+1}} \), we can simply focus on its two endpoints and then employ prior works on two-relations diff (Section III-A). The derived top-explanation \( \mathcal{E}_m^* \) explains the changes from time \( t_{c_i} \) to \( t_{c_{i+1}} \). However, when \( P_i \) is not cohesive, there exists no single static top-explanation \( \mathcal{E}_m^* \) that is capable of explaining the whole trend evolution in \( P_i \). Nevertheless, we can still derive some static top-explanation by looking at its two endpoints and using two-relations diff [13], though the explanation quality might be poor. As we will elaborate in Section IV, each segment in the optimal K-segmentation \( \mathcal{P}_K \) is deemed to be cohesive, and the case of the incohesive segment would only occur during the exploration phase over candidate segmentation schemes. In all, when given a segment \( P_i = \{p_{c_i}, p_{c_{i+1}}\} \), we first obtain a control relation \( R_c = \{\text{SELECT } \ast \text{ FROM } R \text{ WHERE } T = t_{c_i}\} \) and a test relation \( R_t = \{\text{SELECT } \ast \text{ FROM } R \text{ WHERE } T = t_{c_{i+1}}\} \) at two endpoints, and then derive top-explanation with the Cascading Analysts algorithm [13].

Now that we can exploit prior works for deriving static top-explanation within each segment, our problem of identifying evolving explanations boils down to a K-segmentation problem. As \( K \) is not easy to specify in practice, TSEGEXPLAIN identifies the optimal \( K \) for users by default (see Section VI).

IV. K-SEGMENTATION

Designing a good within-segment variance to quantify explanation consistency is the key to our work’s success, and it is non-trivial. In this section, we discuss our design of \( var(P_i) \) in Problem 1 to complete the our problem formulation of K-Segmentation.

Per our definition in Problem 1, K-Segmentation is similar to K-Means clustering. We carefully design the within-segment variance \( var(P_i) \) by making an analogy to the K-Means algorithm. However, different from any existing variance design, \( var(P_i) \) in K-Segmentation should be regarding the variance of top-explanations within each segment \( P_i \).

First, let us review the problem formulation of K-Means. Given a set of \( n \) objects \( \{o_1, o_2, \ldots, o_n\} \) as inputs, where each object is a \( d \)-dimensional vector, K-Means aims to partition these \( n \) objects into \( K \) partitions \( \mathcal{P}_K = \{P_1, P_2, \ldots, P_K\} \) minimizing the within-cluster variance:

\[
\arg\min_{\mathcal{P}_K} \sum_{i=1}^{K} |P_i|var(P_i) \tag{1}
\]

\[
var(P_i) = \frac{1}{|P_i|} \sum_{o \in P_i} dist(o, \mu_i) \tag{2}
\]

where \( \mu_i \) is the centroid of partition \( P_i \) and \( dist(o, \mu_i) \) denotes the distance between an object \( o \) and the centroid \( \mu_i \) in \( P_i \), e.g., \( L2 \) distance. Comparing Problem 1 with Eq. 1, we can see K-Segmentation employs the same optimization objective as K-Means but with a customized \( var(P_i) \). To develop a good \( var(P_i) \) in K-Segmentation, careful thoughts are required around: (1) what is an “object”; (2) what is the centroid of a segment; and (3) how to measure the distance between object and centroid based on explanations.
1) Object in K-Segmentation: Given an aggregated time series \( ts = \{p_1, p_2, ..., p_n\} \), K-Segmentation aims to segment \( ts \) into \( K \) partitions such that explanations are shared within each partition. Since every single point itself cannot reveal any time series trend, the atomic unit for partitioning is a segment of size two, i.e., \([p_i, p_{i+1}]\). That is, an object in K-Segmentation refers to a segment of size two as shown in Figure 3 and there is in total \((n-1)\) objects, i.e., \( \{o_1 = [p_1, p_2], o_2 = [p_2, p_3], ..., o_{n-1} = [p_{n-1}, p_n]\} \).

2) Centroid of a partition: Different from the general K-Means, objects in K-Segmentation follow a time ordering where \( o_1 < o_2 < ... < o_{n-1} \), and a partitioning scheme in K-Segmentation is only valid when objects in each partition form a segment. Hence, a partition in K-Segmentation can be denoted as \( P_i = [p_{ci}, p_{ci+1}] \) with objects \( \{o_{ci} = [p_{ci}, p_{ci+1}], o_{ci+1}, ..., o_{ci+1} = [p_{ci+1}, p_{ci+1}]\} \). Naturally, we can use segment \([p_{ci}, p_{ci+1}]\) as the centroid of partition \( P_i \).

3) Distance between object and centroid: Essentially, both object and centroid are segments of the input time series \( ts \). Thus, we focus on the design of distance between segments. Distance between an object and a centroid follows naturally.

High-Level Idea. As our goal is to group objects with the same top-explanations into one partition, the distance between two segments shall be based on their top-explanations. Given two segments \( P_i \) and \( P_j \), their top-explanations \( E^*_m(P_i) \) and \( E^*_m(P_j) \) can be derived based on Section III-A. Each is a ranked list of explanations, i.e., \( E^*_m(P_i) = [E^*_1, E^*_2, ..., E^*_m] \) and \( E^*_m(P_j) = [E^*_1, E^*_2, ..., E^*_j] \) as shown in Figure 3. Strawman approaches like measuring the Jaccard distance between \( E^*_m(P_i) \) and \( E^*_m(P_j) \) fail short when there are multiple explain-by attributes. For instance, it is unclear how to quantify the partial overlap between \( E^*_1 = \{\text{state} = \text{WA}\} \) and \( E^*_2 = \{\text{state} = \text{WA} \text{ and } \text{age} > 50\} \). Alternatively, we can measure the distance between \( P_i \) and \( P_j \) by how well \( E^*_m(P_i) \) explains \( P_j \) and how well \( E^*_m(P_j) \) explains \( P_i \).

How well \( E^*_m(P_j) \) explains \( P_i \). We draw inspiration from the information retrieval community. Normalized discounted cumulative gain (NDCG) is a widely used measure for ranking quality in information retrieval and we adapt NDCG to quantify how well \( E^*_m(P_j) \) explains \( P_i \). To model our scenario after the web search, we can treat each segment \( P_i \) as a query and each explanation \( E \) as a document. \( E^*_m(P_j) \) corresponds to a ranked list of retrieved documents returned by the search engine, while \( E^*_m(P_i) \) is the ideal retrieved document list. The relevance between a segment \( P_i \) and an explanation \( E \) is quantified by the difference metric \( \gamma(E, P_i) \). As illustrated in Table I (row in blue), given an explanation \( E^*_j \in E^*_m(P_j) \) with rank \( r \), the relevance of \( E^*_j \) towards \( P_i \) can be calculated as \( \gamma(E^*_j, P_i) \). However, explanation \( E^*_j \) might make KPI increase in segment \( P_i \), but lead to a decrease in segment \( P_j \) (see Definition 3.3). When \( E^*_j \) has opposite effects on \( P_i \) and \( P_j \), we shall rectify the relevance to zero as our ultimate goal is to measure the distance between \( P_i \) with \( P_j \). Formally, we denote the rectified relevance as \( \tilde{\gamma}(E^*_j, P_i) \) and we have \( \tilde{\gamma}(E^*_j, P_i) = \gamma(E^*_j, P_i) \times \mathbb{1}_{\gamma(E^*_j, P_i) = \gamma(E^*_j, P_j)} \). Take \( E^*_3 \) in Table I as an example, \( E^*_3 \) contributes the increase in segment \( P_j \) but the decrease in \( P_i \), thus the relevance is rectified to 0.

| Expl | Effect (+/-) | Relevance (\( +\)) | Relevance (\( -\)) | Relevance \( \tilde{\gamma} \) |
|------|-------------|---------------------|---------------------|---------------------|
| \( E^*_j \) | + \( \gamma(E^*_j, P_i) \) | \( \gamma(E^*_j, P_j) \times \mathbb{1}_{\gamma(E^*_j, P_i) = \gamma(E^*_j, P_j)} \) | 0 |
| \( E^*_j \) | - \( \gamma(E^*_j, P_j) \) | \( \gamma(E^*_j, P_i) \times \mathbb{1}_{\gamma(E^*_j, P_j) = \gamma(E^*_j, P_i)} \) | 0 |

TABLE I: Example of DCG Between \( E^*_m(P_j) \) and \( P_i \).

Now we have mapped our scenario to query-document retrieval setting, i.e., query \( P_i \), retrieved document list \( E^*_m(P_j) \), and the rectified relevance formula \( \bar{\gamma} \). NDCG\((P_i, E^*_m(P_j))\) can be calculated using Eq. 3, 4, and 5. It quantifies how well \( E^*_m(P_j) \) explains \( P_i \), with ranges from 0 to 1. At an extreme, when \( E^*_m(P_j) \) is exactly the same as \( E^*_m(P_i) \) and with the same effect on \( P_i \) and \( P_j \), NDCG\((P_i, E^*_m(P_j))\) = 1, meaning \( E^*_m(P_j) \) explains \( P_i \) perfectly.

\[
\text{DCG}(P_i, E^*_m(P_j)) = \sum_{r=1}^{m} \frac{\bar{\gamma}(E^*_r, P_i)}{\log_2(r+1)} \quad (3)
\]

\[
\text{NDCG}(P_i, E^*_m(P_j)) = \frac{\text{DCG}(P_i, E^*_m(P_j))}{\text{DCG}(P_i, E^*_m(P_j))} \quad (5)
\]

Calculating Distance. We then define the distance between \( P_i \) and \( P_j \) as in Eq. 6, where NDCG\((P_i, E^*_m(P_j))\) quantifies how well \( E^*_m(P_j) \) explains \( P_i \) and NDCG\((P_j, E^*_m(P_i))\) quantifies how well \( E^*_m(P_i) \) explains \( P_j \).

\[
\text{dist}(P_i, P_j) = 1 - \frac{\text{NDCG}(P_i, E^*_m(P_j)) + \text{NDCG}(P_j, E^*_m(P_i))}{2} \quad (6)
\]

Eq. 6 averages \( \text{NDCG}(P_i, E^*_m(P_j)) \) and \( \text{NDCG}(P_j, E^*_m(P_i)) \) to obtain the similarity between \( P_i \) and \( P_j \), followed by a complement to get the distance. \( \text{dist}(P_i, P_j) \) is symmetric with ranges \([0, 1]\).

4) Putting all together: Given a partition \( P_i = [p_{ci}, p_{ci+1}] \), it contains a continuous list of objects \( \{P_e = [p_x, p_{x+1}]\} \) where \( c_i \leq x < c_{i+1} \) and its centroid is \( P_i = [p_{ci}, p_{ci+1}] \). Using Eq. 6 and 2, we derive our variance of \( P_i \) as in Eq. 7.

\[
\text{var}(P_i) = \frac{1}{c_{i+1} - c_i} \sum_{x=c_i}^{c_{i+1}-1} \text{dist}(P_e, P_i), \quad \text{where} \quad P_e = [p_x, p_{x+1}] \quad (7)
\]

Our problem formulation of K-Segmentation is now complete, by instantiating \( \text{var}(P_k) \) in Problem 1 with Eq. 7.

V. OUR SOLUTION: TSEXPLAIN

Now we have formulated our K-Segmentation problem (Problem 1), together with the designed within-segment variance in Eq. 7. In this section, we will present our solution TSEXPLAIN: we will start with a dynamic programming (DP) algorithm for solving K-Segmentation, assuming \( \text{var}(P_k) \) is available for each partition \( P_k \); next, we will describe our solution pipeline (Figure 4) – steps for preparing \( \text{var}(P_k) \).
and DP – together with complexity analysis; last, we propose several optimizations for speeding up TSEXPLAIN.

A. DP for K-Segmentation

Different from K-Means, which is computationally intractable (NP-Hard), K-Segmentation is polynomially solvable. At a high level, the search space in K-Means is \(K^n\), while it is \(\binom{n-2}{K-1}\) in K-Segmentation as the task is essentially to identify \((K-1)\) cutting positions among the \((n-2)\) non-endpoints of a given time series \(ts\).

Intuitively, K-segmentation exhibits an optimal substructure – the optimal solution of K-segmentation can be constructed from the optimal solutions of its subproblems. Thus, we can use dynamic programming to solve K-Segmentation. Let \(D(j, k)\) denote the minimal total within-segment variance of \(k\)-segments over time range \([t_j, t_k]\), i.e.,

\[
D(j, k) = \min_{p_k=[p_j, p_k]} \sum_{i=j}^{k-1} |P_i|\sigma^2(P_i),
\]

where \(\sigma^2(P_i)\) is the variance of \(i\)-th segment during \([t_j, t_k]\), i.e., \(D(j, k) = |P_k|\sigma^2(P_k)\). The DP recursion function is expressed in Eq. 8.

\[
D(j, k) = \arg \min_{1 \leq i \leq j} [D(j', k-1) + |P_k|\sigma^2(P_k)], \quad P_k = [p_{j'}, p_j]
\]  

(8)

This recursive computation can be constructed from the optimal solutions of its subproblems. Thus, we can use dynamic programming to solve K-Segmentation. Let \(D(j, k)\) denote the minimal total within-segment variance of \(k\)-segments over time range \([t_j, t_k]\), i.e., \(D(j, k) = \min_{p_k=[p_j, p_k]} \sum_{i=j}^{k-1} |P_i|\sigma^2(P_i), \)

where \(\sigma^2(P_i)\) is the variance of \(i\)-th segment during \([t_j, t_k]\), i.e., \(D(j, k) = |P_k|\sigma^2(P_k)\). The DP recursion function is expressed in Eq. 8.

\[
D(j, k) = \arg \min_{1 \leq i \leq j} [D(j', k-1) + |P_k|\sigma^2(P_k)], \quad P_k = [p_{j'}, p_j]
\]  

(8)

B. Solution Pipeline and Complexity Analysis

The overall solution pipeline is depicted in Figure 4 with modules (a) Preprocessing, (b) Cascading Analysts, and (c) K-Segmentation. We will dive into each module and analyze its computational complexity in the following. The optimization modules with blue background in the following. The optimization modules with blue background in the following.

Precomputation. Module (a) is responsible for computing the diff score \(\gamma(E)\) for each candidate explanation \(E\) over every segment \([p_{j'}, p_j]\) where \(1 \leq j' < j \leq n\). Given an explanation order threshold \(\beta\), we can enumerate all candidate explanations — each is in the form of \(E = [A_1=a_1, ..., A_\beta=a_\beta]\) where \(1 \leq \beta \leq \beta\). Let \(\epsilon\) be the total number of candidate explanations. By default, \(\beta\) is set as 3.

As illustrated in Figure 4, given an explanation \(E\) and a segment \([p_{j'}, p_j]\), the difference score \(\gamma(E)\), i.e., absolute-change in Definition 3.2, can be calculated by looking at the points at time \(t_j\) and \(t_{j'}\) of the aggregated time-series \(ts(R)\) and \(ts(R-\sigma_{x,R})\), i.e., the aggregated time-series from relation \(R\) when excluding data with predicate \(E\). As most aggregate function \(f(M)\) is decomposable, e.g., SUM, AVG, Variance, we can derive \(ts(R-\sigma_{x,R})\) by using \(ts(R)\) and \(ts(\sigma_{x,R})\). Since TSEXPLAIN is designed to integrate with existing interactive data analysis tools like PowerBI, data cube is typically maintained in memory, and thus we can easily access \(ts(R)\) and \(ts(\sigma_{x,R})\) from the data cube.

Time complexity: For each segment \([p_{j'}, p_j]\) and explanation \(E\), it takes \(O(1)\) for computing the difference score \(\gamma(E)\). There are in total \(n^2\) segments and \(\epsilon\) explanations. Thus, the total time complexity for module (a) is \(O(\epsilon \cdot n^2)\).

The Cascading Analysts (CA) Algorithm. Module (b) is responsible for calculating top-explanation \(E^*_m\) for each segment \([p_{j'}, p_j]\). TSEXPLAIN employs [13] to identify top-explanation (Definition 3.5). In a nutshell, the CA algorithm recursively performs drill-down operations in dimensions and selects data slices summarized by some conjunction predicate, i.e., an explanation in our context. Here the drill-down dimensions and data slices are selected via dynamic programming to maximize the total diff score \(\gamma(E)\) under the constraint of the number of data slices \(m\).

Given three explain-by attributes \(A = \{A_i, A_j, A_r\}\), Figure 5 illustrates the CA algorithm for identifying top-5 explanations. Each node in Figure 5 corresponds to an explanation \(E\) and is associated with its diff score \(\gamma(E)\) obtained from module (a). For instance, the left-most node at level two denotes explanation \(E = \{A_i=a_i^2 \land A_j=a_j^2\}\). To identify top-m non-overlapping explanations, the algorithm starts from the root node with \(m\) quotas. It enumerates the first drill-down

---

Fig. 4: Solution Pipeline in TSEXPLAIN.

Fig. 5: Illustration of Cascading Analysts.
dimension (i.e. $A_i$ in Figure 5), and the quota assigned to each drill-down sub-tree, e.g., two out of five is assigned to the subtree rooted at node ($A_1=a_1^2$) and ($A_2=a_2^2$) respectively and another one is assigned to the subtree rooted at node ($A_1=a_1^1$). Again, for the sub-tree rooted at node ($A=a_2^2$) with two quotas, the algorithm enumerates next dimension to drill down (i.e., $A_1$), and assigns quota to each drill-down sub-tree (i.e., one to ($A_1=a_1^2$&$A_2=a_1^1$) and one to ($A_1=a_1^2$&$A_2=a_1^1$)).

The enumeration of drill-down dimension and quota assignment are performed via dynamic programming to maximize the total score $\sumEA$ with the total quota $\leq m$. The algorithm returns top-5 explanations (blue nodes) with maximum $\sumEA=3+4+4+4+3=17$. Please refer to [13] for details.

Time complexity: The CA algorithm [13] takes $O(\cdot |A| \cdot m^2)$ per segment, where $\epsilon$ is the total number of candidate explanations, $|A|$ is the number of explain-by attributes, and $m$ is a user-specified explanation number. By default, $m$ is set as 3. Since there are in total $n^2$ segments, the total time complexity is $O(\cdot |A| \cdot m^2 \cdot n^2)$.

K-Segmentation. Module (c) is responsible for identifying the best $K$-segmentation scheme $P_K^*$ that minimizes the total variance. As described in Section IV (Eq. 6), we first compute $dist(P_k, P_s)$ based on the top-explanations $E_m$ on centroid segment $P_k=[p_{ij}, p_{ji}]$ and object $P_s=[p_x, p_{x+1}]$ obtained from module (b); next, $\var(P_k)$ is calculated for every segment based on Eq. 7; lastly, DP is utilized for deriving $D(n, K)$ and the optimal segmentation scheme $P_K^*$.

Time complexity: Calculating $dist(P_k, P_s)$ takes $O(m)$ in looking at top-$m$ explanations. There are $(n-l)$ centroid segments $P_k$ of length $l$ where $1 \leq l < n$ and each $P_k$ of length $l$ contains $l$ objects $P_s$. In total, we have $\sum_{l=1}^{n-1} (n-l) = O(n^3)$ pairs of $(P_k, P_s)$ and thus the time complexity for calculating all distance is $O(m \cdot n^3)$. Similarly, the total time complexity for calculating variance $\var(P_k)$ of all $P_k$ is $O(n^3)$. With $\var(P_k)$ available, DP takes $O(n \cdot nK) = O(n^2K^2)$, as each step in Eq. 8 involves $O(n)$ for enumerating the last cut’s position and there is in total $nK$ steps. Thus, the complexity of module (c) is $O(m \cdot n^3 + K \cdot n^2)$.

Takeaway. To sum up, the total time complexity is $O(\epsilon \cdot n^2 + \epsilon \cdot |A| \cdot m^2 \cdot n^2 + m \cdot n^3 + K \cdot n^2)$ — it grows linearly to $\epsilon$ and to $|A|$, quadratic to $m$, and cubed to $n$. In general, $m$, $K$, and $|A|$ are small due to the user’s limited perception. Thus, the runtime depends mainly on the number of candidate explanations $\epsilon$ and the size of the aggregated time series $n$. Next, we focus on reducing $\epsilon$ and $n$ for speedup.

C. Optimizations

As described above, the runtime largely depends on the number of candidate explanations $\epsilon$ and the time series length $n$. For Liquor dataset used in Section VII, $\epsilon$ is around 5000 even when only two explain-by attributes are considered, i.e., $|A|=2$. Next, we introduce two optimizations for speedup: (I.) guess-and-verify to reduce $\epsilon$ and (II.) sketching to reduce $n$. Due to space limits, we omit detailed algorithms and complexity analysis. Please find them in our technical report [40].

1) Guess-and-verify: As discussed in Section V-B, the CA algorithm is one of the bottlenecks in our pipeline. Guess-and-verify is designed to reduce the input explanation number $(\epsilon)$ in CA.

High-level Idea. To ensure non-overlapping, the CA algorithm drills down dimensions and selects explanations as shown in Figure 5. The intuition behind guess-and-verify is that explanation $E$ with higher diff score $\gamma(E)$ is more likely to be in the top-$m$ non-overlapping explanations $E_m$. Thus, instead of using all candidate explanations as the input of CA algorithm, we can take a guess and limit the input to only include top explanations with the highest diff score. A smaller input size can dramatically reduce the runtime of the CA algorithm, but the returned results $E_m$ may not be the optimal top-$m$ non-overlapping explanations $E_m$. To mitigate this, we verify whether the returned result is optimal. This process is repeated until the result is verified to be optimal.

2) Sketching: As discussed in Section V-B, the time complexity in each module is at least quadratic to the time series size $n$. This is because each point is treated as a candidate cutting position in K-Segmentation, and thus, the total number of segments involved in each module is $O(n^2)$. Hence, reducing the number of candidate cutting positions can dramatically improve efficiency. Sketching is designed for this purpose, as depicted in Figure 4.

High-level Idea. Given a time series with $n$ points, K-Segmentation aims to select $(K-1)$ cutting points out of $(n-2)$ non-endpoints. Our intuition is that some points are worse-suited to be used as the cutting points and can be eliminated more cost-effectively; next, since the remaining points are of a much smaller size, it is affordable to input them in our solution pipeline (Section V-B). We call the remaining points sketch, as its role is to represent the original $n$ points in the given time series. In particular, Sketching consists of two phases: (I.) sketch selection and (II.) pipeline with sketch. We propose to select sketch using our proposed pipeline in Section V-B, but with the constraint that each segment’s length should be within the length of $L$, where $L << N$. Afterward, we conduct the whole pipeline in Section V-B over the sketch. Please find details in technical report [40].

VI. OPTIMAL SELECTION OF $K$

In real-world datasets, it is hard for users to specify the number of segments $K$ in advance. By varying segment number $K$, TSEXPLAIN outputs segmentation schemes with different variance scores, generating a $K$-Variance curve. The left-hand side of Figure 7 illustrates an example. Intuitively, $K$-Variance curves decrease monotonically as the increase of $K$. At an extreme, when $K=n-1$, the total variance reaches a minimum score of zero. Furthermore, the total variance score drops quickly when $K$ is small and slows down when $K$ grows larger. This indicates that the marginal improvement of increasing $K$ becomes smaller when $K$ is large. Also, a larger
K brings about too many segments, which would exceed user perception limitations. Thus, our goal is to identify the optimal K with relatively low variance and keep the segmentation scheme concise.

Such a task is well-studied in the machine learning community [41]. We borrow the idea of a well-known method named the “Elbow method” [41, 42] which picks the “elbow point” of the K-Variance curve as the optimal K. We use a task-agnostic algorithm [27] to automatically determine the “elbow point” of our K-Variance curve. This algorithm first normalizes the curve to be from (0, 0) to (1, 1). Then, it picks \( K^* = \arg \max_K \{ \text{total var}(K) - K \} \) as the “elbow point” where \( \text{total var}(K) \) denotes the normalized total variance when the segment number is K.

In our implementation, we collect the dynamic programming results \( D(n, K) \) varying K from 1 to 20, plot the K-Variance curve, and then choose the elbow point. We note that compared to calculating \( D(n, K) \) with varying K from 1 to 20 does not add extra cost, since \( D(n, K) \) gets generated for \( 1 \leq K < 20 \) during the dynamic programming process of \( D(n, K = 20) \). We constrain K to be at most 20 due to user perception limitation: when K is too large, e.g., \( K \geq 20 \), it would be hard for users to interpret the explanation results. We admit that when the time series is long, i.e., \( n \) is large, restricting K under 20 might not return the explanations at the finest granularity, but we argue that explanations at coarse grain with \( K \leq 20 \) is a better choice considering user perception limit. Empirically, we observe that TEXPLAIN chooses 6 or 7 segments in most cases in our real-world experiments (Section VII-D).

VII. EXPERIMENTS

We study two questions: (1) how effective is TEXPLAIN in identifying the evolving explanations; (Section VII-C and VII-D) (2) how fast is TEXPLAIN (Section VII-E).

TEXPLAIN is implemented in C++. All experiments are run single-threaded on Macbook Air 2020 with Apple M1 chip 8-core CPU and 16GB memory. More experiments on variance metric design, case studies and comparisons with baselines, and scalability can be found in the technical report [40].

A. Datasets

We introduce the datasets used in the experiments.

1) Synthetic datasets: We synthesize datasets and generate their ground truth K-Segmentation \( P^*_K \). Each dataset is one relation \( R \) with schema: \( T, \text{sales}, \text{category} \). The aggregated time series represents how the total sales changes along the time \( T \) — SELECT \( T, \text{count(sales)} \) FROM \( R \) GROUP BY \( T \). We set the explain-by attributes \( \mathcal{A} = \{ \text{category} \} \) and there are three categories: \( a_1, a_2, a_3 \). Each predicate e.g., \( \{ \text{category}=a_1 \} \) denotes an explanation \( E_i \).

Synthesize Procedure: The aggregated sales time series can be viewed as a summation of each category’s time series. We start by synthesizing each category’s time series. In detail, we first randomly pick cutting points \( \{ c_{1,j}, ..., c_{j} \} \) for each category \( E_i \)’s time series — SELECT \( T, \text{count(sales)} \) FROM \( R \) WHERE category=\( a_i \), GROUP BY \( T \). For each segment \( \{ p_{i,1}, p_{i,j+1} \}, 1 \leq k < j_i \) defined by these cutting points, we synthesize either an upward or downward trend in linear shape. We restrict the adjacent segments to have different up or down trends. We can then derive the ground truth segmentation’s cutting points of the aggregated time series as the union of each category’s cutting points, i.e., \( \bigcup_{i=1}^{3} \{ c_{i,1}, ..., c_{i,j} \} \). We treat \( \bigcup_{i=1}^{3} \{ c_{i,1}, ..., c_{i,j} \} \) as our ground truth because (1) each predicate has a consistent up or down trend in each segment; (2) our restriction that adjacent segments have different trend direction guarantees that every cutting point is necessary and \( \bigcup_{i=1}^{3} \{ c_{i,1}, ..., c_{i,j} \} \) is the minimal coherent segmentation method. Then, we add Gaussian Noise \( N(0, \sigma^2) \) to each predicate’s time series to simulate noisy time series. We quantify the noise level using signal-to-noise level, namely \( SNR_{dB} \) [43]. We set the time series’ length at 100 and synthesize 20 datasets with seven different levels of \( SNR_{dB} \). The lower the \( SNR_{dB} \) is, the noisier the time series is. We remark that the number K and the length of each segment are diverse in our synthetic datasets, with segment number K varying from 2 to 10 and segment length varying from 6 to 84. Please find the distribution in the technical report [40].

Example 3 (Synthetic Dataset and Ground Truth Segmentation): In a synthetic dataset, the predicate category = \( a_1 \) has its cutting points at 52, 76, the predicate category = \( a_2 \) has its cutting points at 70, 90, and the predicate category = \( a_3 \) has its cutting point at 31. Thus, we can derive the aggregated time series ground truth segmentation as the union of three predicates’ – \{ 31, 52, 70, 76, 90 \}.

2) Real-world datasets: We use the two real-word datasets in the motivating examples of Section I. Here, we briefly go through the aggregated time series and explain-by attributes. In practice, we expect users to provide explain-by attributes based on their domain knowledge.

a) COVID-19: [44] records the daily/total confirmed cases of 58 states. Naturally, there are two aggregated time series: \( \ominus \) the covid total confirmed cases trend, SELECT date, SUM(total-confirmed-cases) FROM Covid GROUP BY date; \( \ominus \) the covid daily confirmed cases trend, SELECT date, SUM(daily-confirmed-cases) FROM Covid GROUP BY date. We choose state as our explain-by attribute to answer “which states are the main contributors to the rises or drops?”.

b) Liquor-sales: contains liquor purchase transactions in Iowa from 2020-1-2 to 2020-6-30. The time series is SELECT date, SUM(Bottles_Sold) FROM Liquor GROUP BY date. We pick four attributes out of 24 attributes as our explain-by features: Bottle_Volume (ml) – the size of each bottle in a purchase (e.g., 750ml); Pack – the number of bottles per pack (e.g., 6); Category_Name – the category of the purchased liquor (e.g., American Flavored Vodka); Vendor_Name – the vendor of the purchased liquor (e.g., Phillips Beverage). Below, we use BV, P, CN, and VN to represent them respectively for short.
B. Baselines

TSEXPLAIN is the first explanation-aware segmentation to surface the evolving explanations for time series. The closest segmentation works are the bottom up algorithm (Bottom-Up) which performs best overall in piecewise linear approximation [26] and recent semantic segmentation algorithm - FLUSS [24] and NNSegment [25]. All these methods are explanation-agnostic, partition time series solely based on the visual shapes, and require segment number as input. For a fair comparison, the $K$ for baselines is either given or borrowed from TSEXPLAIN’s results. Implementation wise, we reproduce Bottom-Up based on the pseudo-code in Keogh et al [26]. FLUSS is implemented using Stump library [45] and NNSegment is implemented using the authors’ code [25].

C. Explanations of Synthetic Datasets

We evaluate TSEXPLAIN’s effectiveness on synthetic datasets and perform quantitative comparisons between TSEXPLAIN and the three baselines. For a fair comparison, we adopt the oracle segment number $K$ of the ground truth, and we run TSEXPLAIN and baselines with known $K$.

Metric We propose a metric to compare the effectiveness of these methods by quantifying the distance between these methods’ output and ground truth. We calculate the edit distance between outputs and ground truth. Since different datasets have different segment number $K$ and time series lengths $n$, we normalize our edit distance by $K$ and $n$. The lower the metric is, the more effective the method is. We term this metric distance percent($\%$).

Figure 6 shows the comparison between TSEXPLAIN and baselines. As FLUSS and NNSegment both involve parameters, i.e., period and window size, we try multiple parameters and report the best overall results. The x-axis is SNR, and the y-axis is the distance percent. We report the average distance percent for each SNR level. TSEXPLAIN always has the best performance and Bottom-Up is the most comparable baseline among all three. When SNR > 35, the distance percent(%) of TSEXPLAIN is close to 0, indicating for cleaner datasets, TSEXPLAIN’s output is almost the same as ground truth. However, the baselines are incapable of detecting ground truth.

![Fig. 6: Distance percentage of TSEXPLAIN and baselines.](image)

Takeaway. For synthetic datasets, TSEXPLAIN performs more effectively than all baselines. Bottom-Up is the most comparable baseline. For less noisy datasets, TSEXPLAIN can detect the ground truth accurately, while baselines can not.

D. Case Study on Real-World Datasets

This subsection demonstrates TSEXPLAIN’s effectiveness on two different real-world datasets: Covid-19 and Liquor.

![Fig. 7: Segmentation of total-confirmed-cases. Left is the K-Variance curve. Right shows aggregated time series(top), Bottom-Up results(middle), Bottom-Up results(bottom).](image)

day gives an illustrative comparison between TSEXPLAIN and baselines. Due to space constraints, we only show the overall best baseline - Bottom-Up’s result. More use cases and comparisons with all three baselines can be found in the technical report [40]. TSEXPLAIN chooses the optimal $K$ as discussed in Section VI. For fair comparison, Bottom-Up uses the same segment number $K$ as TSEXPLAIN selects. We focus on the top three explanations in experiments and set each explanation’s order as 3. Experimentally, different top explanation numbers and different orders have similar results. For very fuzzy datasets, we apply a moving average to smooth it. The moving average window, explanation number, and order can be customized via the demo interface [23].

1) COVID-19: We explain two time series in COVID-19 dataset, total-confirmed-cases and daily-confirmed-cases separately.

a) Total-confirmed-cases. Figure 7(left) shows the $K$ – Variance curve of different segment numbers $K$ and TSEXPLAIN identifies that the optimal $K$ equals 6. Figure 7(right) shows the overall trend, TSEXPLAIN’s output segmentation scheme with the top three explanations’ trend, and baseline’s output. Please refer to the legend of Figure 2 for the top-3 explanations of TSEXPLAIN. The first segment is from 1/22 to 3/14, where the increase of total-confirmed-cases is due to WA, NY, CA’s increase. From 3/14 to 5/4, NY increases the most, followed by NJ and MA. Then, between 5/4 and 5/29, IL, CA, NY slowly increase. Later on, the confirmed cases surge mainly because of the sharp increase in CA, TX, FL, and IL, especially IL increases quickly between 9/25 to 11/27. Contrarily, as shown in the bottom chart of Figure 7(right), the Bottom-Up baseline’s output shows the phenomenon that the neighboring segments’ explanations are exactly the same, i.e., 6/16–7/31 and 7/31–11/2. Also, TSEXPLAIN detects the IL’s increase earlier while baseline discovers it much later.

b) Daily-confirmed-cases. In Figure 8 and table II, TSEXPLAIN segments it into seven periods. Specifically, from 3/7 to 4/6, NY, NJ, and MA’s rises contribute to the overall rise in the US. From 4/7 to 5/25, NY and NJ decline dramatically, and TSEXPLAIN captures an interesting pattern that CA starts to rise. During the holiday season in 2020, namely the last segment, CA and NY surge again while IL declines. Comparing TSEXPLAIN and Bottom-Up in Figure 8, we can clearly see the drawbacks of the baseline that it can not detect the NY’s increase and drop during 3/7–5/25.
We can see TSE recognizes that from 4/21 to 5/8, BV=1750&P=12(-) increases a lot and from 3/6 to 6/10, BV=1000 - BV=750&P=12 + BV=375&P=24(-). TSE explains this in such a way that ① BV=1000 decreases sharply, otherwise the overall bottles sold can increase much more; ② BV=1750&P=6(+)+ and BV=750&P=12(+) directly contribute to the increase. Moreover, with some background knowledge, we find that liquor with BV=1000 is mainly sold in independent stores. In March, Iowa’s close down proclamation [46] requires restaurants and bars to shut down, and most independent retailers rely on selling liquor to bars and restaurants. As a result, their business significantly declined. In late April, Iowa Governor issued a proclamation reopening restaurants, and the business of independent liquor stores gradually recovered. We can see TSE recognizes that from 4/21 to 5/8, BV=1000ml&Pack=12 increases a lot and from 5/8 to 6/10, BV=1000 becomes the top increasing explanation indicating that independent stores benefited from the reopening policy. In comparison, the explanations of the Bottom-Up results look flat, as shown at the bottom of Figure 9(right), indicating the detected explanations change subtly. Also, we remark that although we specify four explain-by attributes, the results are only about BV and P. This indicates that TSE is able to identify interesting attributes and ignore the less interesting ones, i.e. VN and CN.

**Takeaway.** TSE effectively explains real-world datasets while the baselines defect in ① Less explanation diversity: the neighboring segments have the same explanations; ② Less effectiveness: the underlying key explanations can not be detected, and explanations are detected relatively late; ③ Less significance: The explanations detected change subtly along the time. Also, TSE is able to identify interesting attributes out of all candidate attributes.

### E. Efficiency Evaluation

We conduct two experiments to evaluate the efficiency. In the first experiment, we report the breakdown latency and study the impact of our optimization strategies. In the second experiment, we compare the end-to-end runtime between TSE and three baselines. More experiments regarding scalability and effectiveness can be found in the technical report [40].

#### 1) Latency Breakdown and Quality:

**a) Methods:** Besides the two optimizations in Section V-C, we introduce another straight-forward optimization – filter – given an explanation $E$, if each point in its aggregated time series has a smaller value than a $ratio$ of the overall aggregated time series, we filter out $E$ as its support is low. We set $ratio$ as 0.001 by default.

We study TSE with different optimizations: **VANIL-LAT EXPLAIN** (short as **VANILLA**) is the plain version without any optimization; **w filter** filters out the explanations below the default $ratio$; **O1** represents the algorithm which applies **guess-and-verify** after filtering; likewise, **O2** applies **sketching** and **O1+O2** applies both optimizations. Table IV summarizes the statistics of the datasets. $\epsilon$ is the total candidate explanation number, $n$ is the time series length, and #record is the dataset record number. We also report the filtered $\epsilon$. We remark that in this experiment, $K$ is unspecified, and the time of selecting the optimal $K$ via the Elbow Method (Section VI) is included in the latency (Figure 10).

| dataset                  | $\epsilon$ | filtered $\epsilon$ | n  | #records |
|--------------------------|------------|---------------------|----|----------|
| total-confirmed-cases    | 58         | 54                  | 345| 1152300  |
| daily-confirmed-cases    | 58         | 55                  | 345| 1152300  |
| Liquor-sales             | 8197       | 1812                | 128| 1161436  |

**TABLE IV: Real-world Dataset Statistics.**

**b) Results:** Figure 10 illustrates the breakdown latency, which consists of three parts: precomputation(blue), the CA algorithm(orange), K-segmentation(green) corresponding to
three modules in section V-B. For COVID-19 total and daily-confirmed-cases dataset, the filter strategy only slightly improves since the predicate number $\epsilon$ is similar before and after filtering. In contrast, the sketching in O2 significantly reduces the latency of the CA algorithm and K-segmentation. Overall, O1+O2 reduces the latency of total-confirmed-cases from 175 ms to 33ms, and the latency of daily-confirmed-cases from 217ms to 43ms. For the Liquor dataset, the predicate number $\epsilon$ is very large even after filtering. The CA algorithm is the bottleneck. VANILLATSEXPLAIN takes 9.8888s. After filtering, it still takes 2.59s. O1 takes big effect in this case. Each optimization O1 or O2 alone can significantly shrink the runtime to around 1.1s. The two optimizations together reduce the runtime to 756ms.

| Dataset          | Variance(VANILLA) | Variance(O1+O2) |
|------------------|-------------------|-----------------|
| Total-confirmed-cases | 22.702            | 22.744          |
| Daily-confirmed-cases | 91.619            | 91.994          |
| Liquor-sales     | 33.653            | 33.653          |

TABLE V: Quality of optimization strategies.

2) End-to-end Efficiency Comparison with Baselines:

a) Methods: The three baselines in Section VII-B solely focus on visual shape based segmentation without providing any explanations and require the segmentation number as an input. To make them comparable, first, after segmenting using each baseline, we add the explanation module using the CA algorithm in Section V-B; second, we reuse the optimal $K$ TSEXPLAIN finds in Section VII-D, and then run all baselines and TSEXPLAIN with this given optimal $K$. We also remark that the latency of determining the optimal $K$ is very low, around 2ms in our experiments.

b) Results: Figure 11 reports the end-to-end efficiency comparison. For each baseline, we report the segmentation and explanation time separately, while for TSEXPLAIN, we show the overall time since our segmentation module interleaves with the explanation module. To illustrate the effectiveness of our proposed optimizations, we also report VANILLATSEXPLAIN (VANILLA). We can tell that for different datasets, FLUSS is always the slowest, NNSegment and Bottom-Up rank in the middle. VANILLATSEXPLAIN is similar to the Bottom-Up on the COVID-19 datasets and becomes slow when the predicate number goes up in the Liquor-sales dataset. Yet, combined with all the proposed optimizations, TSEXPLAIN is the fastest compared to all baselines on all datasets.

![Fig. 10: Latency of TSEXPLAIN.](image1)

![Fig. 11: Efficiency comparison with baselines.](image2)

Takeaway. TSEXPLAIN can explain these three time series within 800ms, and our optimizations have accelerated the running time up to $13 \times$ with neglectable effects on quality. What’s more, TSEXPLAIN is faster than all the baselines.

VIII. DISCUSSION

Time-varying Attribute. Different from non-temporal attributes, time-varying attributes are those whose values may change over time [47]. Please refer to [40] for discussion.

Seasonal Datasets For seasonality datasets, TSEXPLAIN can explain the seasonality dataset directly and detect the repeated pattern of evolving explanations which indicates the periodicity property. Users can also first decompose the seasonal datasets [48] and explain the seasonality and trend separately.

Real-time Time Series. We briefly discuss how TSEXPLAIN can be extended to support real-time time series explanation. TSEXPLAIN first gives users the segmentation results of existing time series. When new data arrives, it incrementally computes the top explanations for the new time series, runs the segmentation algorithm based on the existing time series’ cutting point and newly arrived data points, and updates the segmentation results.

IX. CONCLUSION

This work introduces TSEXPLAIN, the first explanation engine that identifies the evolving explanations for aggregated time series. We formulate the problem for deriving evolving explanations as a K-Segmentation problem, aiming to partition the input time series into $K$ smaller segments such that each period has consistent top explanations. We propose a novel explanation-aware variance metric to quantify the consistency in each segment and develop a dynamic programming algorithm for identifying the optimal K-Segmentation scheme. TSEXPLAIN automatically determines the optimal $K$ using the “elbow method”. In our experiments, TSEXPLAIN effectively discovers the evolving explanation of synthetic and real-world datasets. We propose optimizations that enable TSEXPLAIN to answer all our queries interactively within one second. Several future work directions include extending the difference metric library, recommending explain-by attributes, and adding hints for segments with higher variance.
REFERENCES

[1] Berit Hoffmann, “Unveiling Sisu’s vision for Decision Intelligence,” https://sisudata.com/blog/unveiling-vision-decision-intelligence, 2021.

[2] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals,” Data mining and knowledge discovery, vol. 1, no. 1, pp. 29–53, 1997.

[3] Tableau, “BI and Analytics Software,” https://www.tableau.com/, 2021.

[4] Microsoft, “PowerBI,” https://powerbi.microsoft.com/en-us/, 2021.

[5] Google Trend, “Taylor swift compared with kim kardashian,” https://trends.google.com/trends/explore?q=%2Fm%2F0dl567,%2Fm%2F0drlz1&date=now%207-d&geo=US, 2020.

[6] E. Wu and S. Madden, “Scorpion: Explaining away outliers in aggregate queries,” Proc. VLDB Endow., vol. 6, pp. 553–564, 2013.

[7] Z. Liu and J. Heer, “The effects of interactive latency on exploratory visual analysis,” IEEE Transactions on Visualization and Computer Graphics, vol. 20, no. 12, pp. 2122-2131, 2014. [Online]. Available: https://doi.org/10.1109/TVCG.2014.2346452

[8] Y. Chen and S. Huang, “Tsexplain: Surfacing evolving explanations for time-series black-box models,” in “Data mining in time series databases,” in Proceedings of the 2021 International Conference on Data Mining, 2021, pp. 533–534, 2020.

[9] S. Gharghabi, C.-M. Yeh, Y. Ding, W. Ding, P. Hibbing, S. LaMunion, A. Kaplan, S. E. Crouter, and E. Keogh, “Domain agnostic online semantic segmentation for multi-dimensional time series,” Data mining and knowledge discovery, vol. 33, no. 1, pp. 96–130, 2019.

[10] Y. Matsubar, C. Faloutsos, and C. Faloutsos, “Autoplait: Automatic mining of co-evolving time sequences,” in Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014, pp. 193–204.

[11] S. Tonekaboni, S. Joshi, K. Campbell, D. K. Duvenaud, and A. Goldemberg, “What went wrong and when? instance-wise feature importance for time-series black-box models,” in “Data mining,” 2019, pp. 799–809, 2020.

[12] C. Rokie, J. Smith, K. K. Leung, M. Volkovs, and S. Zuberi, “Temporal dependencies in feature importance for time series predictions,” arXiv preprint arXiv:2107.14317, 2021.

[13] J. Labaen, E. Augusti, and D. X. Carlos, “Contrastive explanations for a deep learning model on time-series data,” in "International Conference on Big Data Analytics and Knowledge Discovery." Springer, 2020, pp. 235–244.

[14] J. A. Hartigan and M. A. Wong, “Algorithm as 136: A k-means clustering algorithm,” Journal of the royal statistical society. series c (applied statistics), vol. 28, no. 1, pp. 100–108, 1979.

[15] Y. Chen and S. Huang, “Tsexplain: Explaining aggregated time series by surfacing evolving contributors,” arXiv preprint arXiv:2211.10909, 2022.

[16] T. M. Kodinariya and P. R. Makwana, “Review on determination of cluster in k-means clustering,” International Journal, vol. 1, no. 6, pp. 90–95, 2013.

[17] A. Ng, “Clustering: With the k-means algorithm,” Machine Learning, vol. 4, no. 39, p. 271–280, 2011.

[18] Wikipedia, the free encyclopedia, “Signal-to-noise ratio,” https://otexts.com/,

[19] T. M. Kodinariya and P. R. Makwana, “Review on determining number of contributing subsets for aggregate query answers,” Proceedings of the VLDB Endowment, vol. 12, no. 12, pp. 1899–1901, 2019.

[20] C. Rooke, J. Smith, K. K. Leung, M. Volkovs, and S. Zuberi, “Temporal dependencies in feature importance for time series predictions,” arXiv preprint arXiv:2107.14317, 2021.