**Features Data Documentation (features\_df)**

The **features.csv** file contains important store-specific factors that could influence sales, such as markdowns, weather, and economic indicators. Here's a detailed explanation of each variable:

1. **Store**:
   * **Description**: The unique identifier for each store.
   * **Type**: Integer.
   * **Purpose**: Used to link the store-specific factors to sales data and store attributes (such as type and size). We will later merge this column with other datasets (like the train and stores datasets) for analysis.
2. **Date**:
   * **Description**: The date corresponding to the recorded features.
   * **Type**: Date (currently in string format, will be converted to datetime).
   * **Purpose**: Helps in tracking temporal changes (e.g., weekly or monthly trends). Important for time series analysis and understanding seasonality.
3. **Temperature**:
   * **Description**: The average temperature for the store's location on that date (in Fahrenheit).
   * **Type**: Float.
   * **Purpose**: Weather can significantly influence customer behavior. For example, people may shop more during mild weather and less during extreme conditions. This variable can help model sales fluctuations due to weather.
4. **Fuel\_Price**:
   * **Description**: The price of fuel in the store's region on that date.
   * **Type**: Float.
   * **Purpose**: Fuel price fluctuations can impact transportation costs for customers and suppliers, potentially influencing sales and operational costs. It's important to consider fuel price as a factor affecting demand.
5. **MarkDown1 - MarkDown5**:
   * **Description**: These columns represent various types of markdowns (discounts) offered by the store.
   * **Type**: Float (but with many missing values that likely represent periods with no markdowns).
   * **Purpose**: Discounts can drive sales by encouraging customers to purchase more. Analyzing how different markdown types affect sales can help optimize promotional strategies. We will handle missing values for these columns, likely by filling them with 0 if no markdown was applied during a given period.
6. **CPI**:
   * **Description**: The Consumer Price Index (CPI) for the store's region on that date.
   * **Type**: Float.
   * **Purpose**: CPI is an economic indicator that measures the average change in prices paid by consumers for goods and services over time. It can be used to track inflation and understand how it might affect consumer spending patterns. Missing values in this column will be handled via forward filling.
7. **Unemployment**:
   * **Description**: The unemployment rate in the store's region on that date.
   * **Type**: Float.
   * **Purpose**: Unemployment affects consumer confidence and spending. A higher unemployment rate might lead to lower sales, while lower unemployment can signal higher consumer spending. Like the CPI, missing values will be handled with forward or backward filling.
8. **IsHoliday**:
   * **Description**: Indicates whether the date corresponds to a major holiday.
   * **Type**: Boolean (True for holiday, False for non-holiday).
   * **Purpose**: Sales typically increase around holidays, making this a key variable for identifying spikes in demand. This flag can be used in time series models to capture holiday effects on sales.

**Intuition on features**

Yes, Markdown values can be negative, though it depends on how markdowns are recorded in the data. Here’s why:

Possible Reasons for Negative Markdown Values:

Incorrect Data Entry: A negative markdown could be the result of a data entry error or inconsistency in the way markdowns are logged in the system.

Price Increases After Discount: Some retail systems might record markdowns as negative if prices were raised after being lowered during a promotion. For example, if the store temporarily increased the price after a sale period, the markdown might be recorded as negative to reflect that change.

Refund or Return Adjustments: If a store processes refunds or returns, it may record these transactions as negative markdowns, especially if the refunded amount was more than the discounted price.

Inventory Correction: Sometimes, negative markdowns reflect inventory corrections where the cost of goods was adjusted after being sold at a markdown.

Key Point:

If you encounter negative markdowns, it's essential to investigate the context behind how markdowns are recorded by the company. You might need to treat them as anomalies or adjust them depending on the underlying business logic

**Intuition for Handling Missing Values in CPI and Unemployment:**

1. **CPI (Consumer Price Index)**:
   * **What it represents**: CPI is an economic indicator that reflects the average change in prices paid by consumers over time for a basket of goods and services.
   * **How it affects sales**: High CPI generally indicates inflation, which could reduce consumer purchasing power, while low CPI indicates more stable prices.

**Filling Missing Values for CPI**:

* + **Assumption**: CPI usually doesn’t change drastically from one week to the next. Missing values are likely due to gaps in the data and can be filled using the nearest available value.
  + **Action**: We can fill missing CPI values using **forward fill** (propagating the last known value) or **backward fill** (taking the next available value) to maintain consistency in the data.

1. **Unemployment**:
   * **What it represents**: Unemployment represents the percentage of the workforce that is unemployed and actively seeking employment.
   * **How it affects sales**: Higher unemployment might lower consumer spending, while lower unemployment typically boosts consumer confidence and spending.

**Filling Missing Values for Unemployment**:

* + **Assumption**: Like CPI, unemployment doesn’t fluctuate wildly from week to week. Missing values can be filled using **forward fill** to reflect the economic stability.
  + **Action**: We can fill missing values using **forward fill** to maintain a smooth trend in unemployment over time.

**Approach:**

* **Forward Fill** for both CPI and Unemployment: This method ensures that we maintain consistent economic data, without introducing sudden jumps or drops due to missing values.

The **Unemployment** and **CPI** data in this context likely refers to the **regional economic conditions** surrounding the store's location, not the employees or workforce directly within the store itself. Here's the reasoning:

**Unemployment in this Context:**

* **What it represents**: It likely refers to the **regional unemployment rate** where the store operates, affecting the local economy and consumer spending power.
* **Impact**: Higher unemployment in the region might lead to reduced foot traffic and lower sales in stores, as people in the area have less disposable income. Conversely, lower unemployment can boost sales as more people have jobs and the ability to spend.

**CPI in this Context:**

* **What it represents**: It likely refers to the **regional CPI** for the area surrounding the store, reflecting inflation and price levels in the region.

Thus, the **Unemployment** and **CPI** in the dataset are not about the store’s internal operations but reflect the **local economic conditions** of the store’s surrounding area.

Would you like to proceed with this clarification in mind, or would you like to adjust the approach based on this understanding?

**1. Store:**

* The data for the Store column appears to be evenly distributed, with no significant outliers.
* The range is fairly compact, and the boxplot shows that the data is spread between approximately 0 to 50 stores.
* There are no extreme values or significant deviations.

**2. Temperature:**

* The Temperature column also shows a relatively normal distribution without outliers, as most data points fall within the expected range.
* The range of values spans from approximately 20 to 100, with most of the data concentrated between 40 and 80.

**3. Fuel\_Price:**

* The Fuel\_Price boxplot shows that there are no significant outliers, and the data is tightly clustered between 2.75 and 4.25.
* The distribution is compact and seems to follow a normal spread.

**4. MarkDown1, MarkDown2, MarkDown3, MarkDown4, MarkDown5:**

* All the MarkDown columns show heavy presence of outliers.
* There is a clear cluster of data around lower values (close to 0), but many outliers are spread out to significantly larger values.
* For instance, MarkDown1, MarkDown2, MarkDown3, and MarkDown4 exhibit a large number of outliers, suggesting some exceptional markdowns or transactions that deviate significantly from the rest of the data.
* The spread of outliers extends to high values, with a noticeable jump beyond the interquartile range, indicating skewed distributions and potential for extreme values.
* **Recommendation**: These outliers should be investigated further as they could represent special promotions, rare events, or errors in data entry.

**5. CPI (Consumer Price Index):**

* The CPI column does not have significant outliers and shows a regular distribution with a range from approximately 140 to 220.
* The data is spread evenly with a typical bell-shaped distribution.

**6. Unemployment:**

* The Unemployment column seems to have a somewhat larger range of values, but no extreme outliers are visible.
* This suggests that the unemployment data is fairly consistent without many anomalies.

**General Observations:**

* The MarkDown columns show a heavy skew with many outliers, indicating there are several extreme markdown events or prices, which may need further analysis to determine if they are legitimate or should be cleaned.
* Most other numerical columns (like Temperature, Fuel\_Price, CPI, Unemployment) seem to have a normal distribution without extreme deviations.
* Depending on the business case, you might want to treat outliers in the MarkDown columns differently, such as using robust statistical methods or even removing them if they are due to data entry errors or rare events.

To further investigate the **distribution of outliers** in the MarkDown columns, we can use **histograms**. Histograms provide a detailed view of the frequency distribution of values in a dataset, helping us visualize:

* **How often the extreme values occur** (whether outliers are rare or common).
* **The shape of the distribution** (whether it's skewed or normal).

**Why Use a Histogram:**

* **Understand the spread**: A histogram helps to see the spread of values, including how frequently extreme values (outliers) appear.
* **Detect skewness**: It allows us to see if the data is heavily **skewed** (i.e., long tail to the right or left). If the data is right-skewed (with a long tail), this confirms that extreme values are pulling the distribution toward higher values, which could be addressed with transformations like **log transformation** or **capping**.
* **Guide outlier treatment**: Seeing the full distribution can help you decide whether to cap values, apply transformations, or handle outliers in another way.

In histogram what to look for

* **Peaks**: Look for where the majority of data points cluster (usually around 0 for markdowns).
* **Tails**: Look at the long tail in the distribution, which will show how many extreme values exist.
* **Skewness**: Determine whether the data is **right-skewed** (most values are small but a few are large) or left-skewed.

By doing this, you can better understand:

* Whether most of the markdown values are clustered around zero (as expected), and how frequently high markdowns occur.
* The overall distribution to decide on further actions like **capping**, **scaling**, or **transforming**.

A screenshot of a computer

Description automatically generated

Looking at the histograms, the **MarkDown** columns exhibit a clear **right-skewed distribution** with a long tail. Most of the values are concentrated near 0, but there are some extreme values (outliers) extending far beyond the majority of the data points. Here’s a breakdown:

**Observations:**

1. **Concentration of Low Values**: The majority of the markdown values are small, clustered around 0, indicating that markdowns are often low or absent for most entries.
2. **Long Tail**: There is a long tail extending to high values, which are the extreme outliers. This confirms that the dataset has a heavy right skew, with a few very large markdown values.
3. **Skewness**: The distributions suggest that the data is **heavily skewed**, which could impact the performance of certain models (e.g., linear regression).

**Next Steps: Handling Outliers and Skewness**

1. **Capping or Removing Outliers**:
   * We can cap the extreme values at a certain percentile (e.g., 95th or 99th percentile) to limit the impact of outliers.
2. **Log Transformation**:
   * Apply a **log transformation** to reduce the effect of skewness. Since the log transformation can handle skewed distributions well, it would bring extreme values closer to the bulk of the data.
3. **Normalize the Features**:
   * After outlier treatment, we can **normalize** the markdown features to ensure that they are on a similar scale, which can help during modeling.

**Suggested Action Plan:**

1. **Cap outliers** at the 95th percentile to reduce the impact of extreme values.
2. Optionally, apply a **log transformation** to handle skewness.
3. **Normalize** the markdown features for better model performance.

you can absolutely apply **all three** techniques (capping, log transformation, and normalization), but it depends on the specific needs of your analysis and model. Let me explain:

**1. Capping Outliers:**

* **Why**: Capping is useful to prevent extreme values from disproportionately influencing your model, while still preserving most of the data's structure.
* **When to Apply**: Capping is generally applied **first** before any transformations to remove extreme values.

**2. Log Transformation:**

* **Why**: Log transformation helps reduce skewness and brings extreme values closer to the central bulk of the distribution. This is especially useful for highly skewed data like the MarkDown columns.
* **When to Apply**: After capping, apply log transformation to further reduce skewness and improve data normalization.

**3. Normalization:**

* **Why**: Normalization brings features to the same scale (usually between 0 and 1), which is important for models that are sensitive to feature scaling (e.g., gradient descent-based models).
* **When to Apply**: After capping and log transformation, you normalize the features to standardize the data range.

**Order of Application:**

1. **Cap the outliers** to limit the extreme values.
2. **Apply log transformation** to handle skewness.
3. **Normalize** the data to bring the features onto a similar scale.

Issues with log transformations

**Explanation of Changes:**

* **Log Transformation**: We handle 0 values by replacing them with NaN before applying the log transformation (log1p), then fill the NaN values back with 0.
* **MinMaxScaler**: After transforming the data, we apply the scaler to normalize the features.

This should prevent inf or NaN issues during the scaling step.

Would you like to proceed with this solution?

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Purpose of Visualization:**

* **Boxplots** will show if outliers have been capped and whether the data is more centered.
* **Histograms** will show if the **skewness** has been reduced after the log transformation.

**Observations:**

1. **Capping Effect**: The values are now contained within the 0–1 range (due to normalization), and extreme outliers have been reduced.
2. **Log Transformation Effect**: Skewness in some columns (e.g., MarkDown1 and MarkDown4) has been reduced, although a few outliers remain.
3. **Normalization Effect**: The columns are now scaled between 0 and 1, making the features ready for modeling, especially for algorithms sensitive to feature scaling.

**intuition behind choosing Min-Max Normalization for normalization:**

1. **Preserves Data Distribution**:  
   Min-Max normalization scales the data into a defined range, typically [0, 1]. This preserves the relationships and distribution of the original data, which is useful when you want to keep the proportionate differences between data points intact. Unlike other methods (e.g., standardization), Min-Max does not shift or distort the shape of the distribution.
2. **Handles Data with Known Boundaries**:  
   If your data is bounded or constrained (e.g., prices, percentages, or scaled values like MarkDown), Min-Max normalization ensures all values fall within the specified range, making it easier to compare features.
3. **For Models Sensitive to Feature Scaling**:  
   Algorithms such as **k-NN**, **neural networks**, and **gradient descent-based methods** (e.g., linear regression) are sensitive to the scale of features. Min-Max normalization helps ensure that features contribute equally to the model training process, preventing large-scale features from dominating smaller-scale ones.
4. **Simplicity and Interpretability**:  
   Min-Max normalization is intuitive since the data is scaled within a fixed range, making it easier to interpret. For example, after normalization, the largest value in the dataset is 1, and the smallest is 0, which simplifies understanding and visualizing the data.
5. **Works Well with Non-Normally Distributed Data**:  
   Min-Max normalization is particularly useful when the data is not normally distributed (which is often the case with skewed data like MarkDown columns). It allows us to handle distributions that may not be centered around the mean, unlike standardization, which centers data around 0.

**In summary:**

We chose **Min-Max Normalization** because it effectively scales all features within the same range while maintaining the relative differences between data points. It is particularly beneficial for algorithms sensitive to feature scaling, and it works well with data that has boundaries or is not normally distributed.

**Stores df**

**Summary Statistics:**

* The **Size** of the stores ranges from **34,875** square feet to **219,622** square feet, with a mean size of approximately **130,287** square feet.
* The **Store** column simply identifies the stores with values ranging from 1 to 45.
* The **Type** column represents the store type (A, B, or C), but it doesn't provide numerical statistics since it's categorical.

**Missing Values:**

* There are **no missing values** in this dataset, so no additional imputation is necessary.

**Next Steps:**

1. **Visualize Store Types**: We can create a bar plot to see the distribution of store types (A, B, C).
2. **Visualize Store Size**: We can create a histogram or boxplot to visualize the distribution of store sizes and check for any outliers.

Data leakage occurs when information from outside the training dataset, particularly from the test or validation data, is used to train the model. This usually happens inadvertently and gives the model access to data it wouldn’t normally have during real-world predictions, leading to overly optimistic performance during training but poor generalization on new data.

**Examples of Data Leakage**

1. **Future Information in Time Series**:
   * In time series forecasting, using future sales data in a rolling average calculation for current sales would introduce data from the future, resulting in leakage.
   * **Solution**: When creating features, shift or lag data appropriately so only past information is used for predictions.
2. **Feature Engineering on Entire Dataset Before Splitting**:
   * If you calculate aggregated statistics (e.g., mean, standard deviation) over the entire dataset before splitting into training and test sets, the test set information might "leak" into the training set.
   * **Solution**: Perform feature engineering after splitting, or calculate statistics only on the training set and apply them to both training and test sets.
3. **Target Leakage**:
   * When a feature is created from or directly correlated with the target variable, it introduces leakage because it gives the model direct hints to predict the target.
   * **Solution**: Avoid creating features that rely on or are derived from the target variable itself.

**Why It’s Problematic**

Data leakage artificially inflates model performance during training. This often leads to models that fail in production because they rely on information unavailable in a real-world scenario, resulting in poor generalization.

Would you like to examine potential areas of data leakage in this project?