1. Importing Relevant Libraries

- **Libraries used:**
  - pandas, numpy for data handling.
  - datetime for date processing.
  - matplotlib, seaborn for data visualization.
  - scikit-learn (sklearn) modules for:
    - Model building (LinearRegression, RandomForestRegressor, etc.)
    - Data preprocessing (StandardScaler, OneHotEncoder)
    - Model evaluation (mean_squared_error, r2_score, classification_report).

---

2. Reading the Dataset

```
df = pd.read_excel('DTC_IMP.xlsx')
```

- **Action:** Imported the dataset from an Excel file named **DTC_IMP.xlsx** into a DataFrame called df.

---

3. Standardizing the 'Date' Column

- **Problem:** 'Date' entries were in inconsistent formats.
- **Solution:** Defined a function parse_date() that:
  - Tries multiple date formats ("%d.%m.%Y", "%d.%m.%y", etc.).
  - Converts each date string into a standard datetime object.
  - Returns NaT (Not a Time) if parsing fails.

```
df['Date'] = df['Date'].apply(parse_date)
```

- **Result:** All entries in the 'Date' column are now in a **standard datetime format**.

4. Saving the Cleaned Dataset

```
df.to_excel("DATA_IMP_cleaned.xlsx", index=False)
```

- **Action:** Saved the cleaned DataFrame into a new Excel file called **DATA_IMP_cleaned.xlsx** without adding an extra index column.

---

5. Viewing Sample Data

```
df.head()
```

- **Displayed:** First 5 rows of the dataset.
- **Main columns observed:**

    - S.no., Depot Name, Bus No., Route No., Sch.KMs (Scheduled KMs), Act.KMs (Actual KMs), Miss KMs due to BD only (Missed kilometers due to breakdowns).

```
df.tail()
```

- **Displayed:** Last 5 rows of the dataset for completeness check.

6. Cleaning Column Names

```
df.columns = df.columns.str.strip().str.replace('.', ", regex=False)
```

- **Action:**
    - Removed extra spaces and periods (.) from all column names.
- **Purpose:**
    - Made column names clean and consistent for easier analysis.

---

7. Creating a 'MonthYear' Column

```
df['MonthYear'] = df['Date'].dt.to_period('M')
```

- **Action:**
    - Created a new column **MonthYear** showing year and month for each entry (e.g., 2023-04).
- **Purpose:**
    - Useful for **monthly trend analysis** later.

---

8. Checking Columns and Data Types

```
print(df.columns.tolist())
df.dtypes
```

- **Action:**
    - Listed all column names.
    - Verified data types:
        - Sno is integer (int64).
        - Others like Depot Name, Bus No, Route No, etc., are object type (strings/mixed).

9. Checking Data Size

```
len(df.index)
```

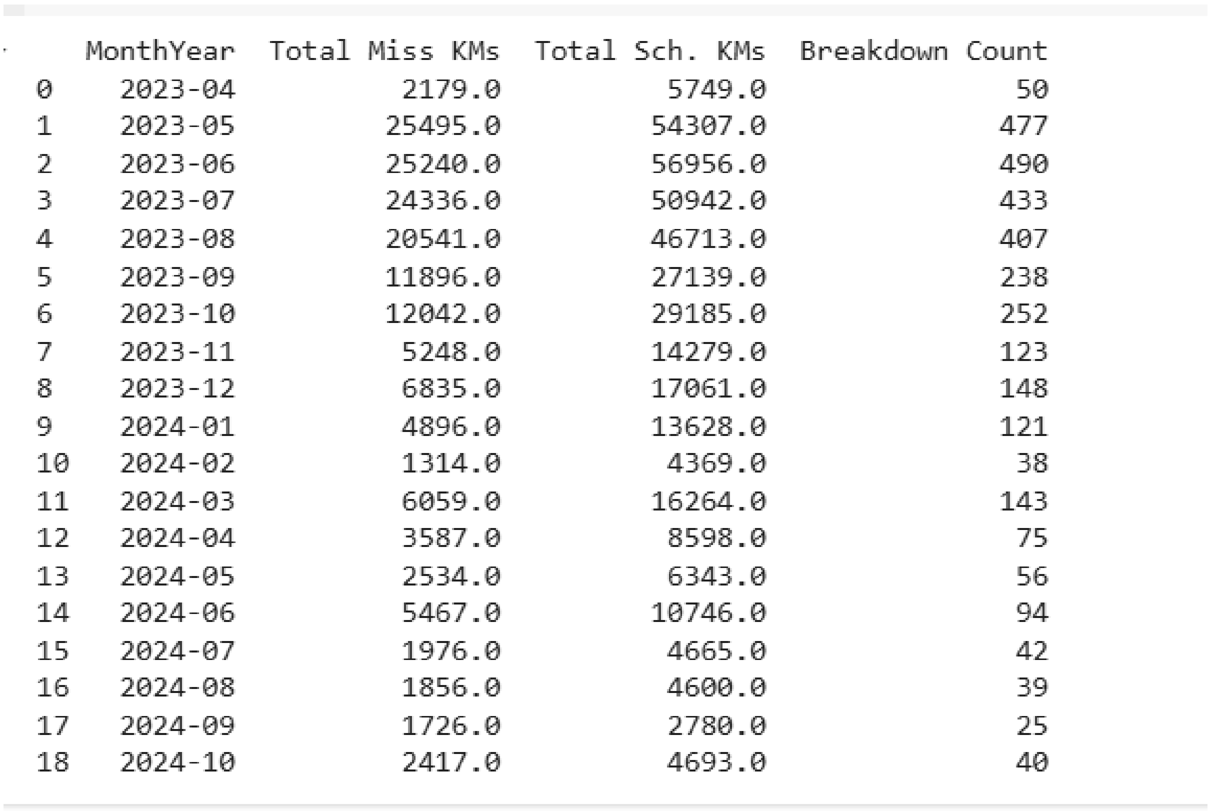
- **Result:**
    - Total **7829 rows** in the dataset.

10. Checking for Missing Values

df.isna().any()

- **Action:**
  - Checked if any column has missing (NaN) values.
- **Purpose:**
  - Important step before further analysis or model building

ANALYSIS



```
    MonthYear  Total Miss KMs  Total Sch. KMs  Breakdown Count
0     2023-04          2179.0          5749.0               50
1     2023-05         25495.0         54307.0              477
2     2023-06         25240.0         56956.0              490
3     2023-07         24336.0         50942.0              433
4     2023-08         20541.0         46713.0              407
5     2023-09         11896.0         27139.0              238
6     2023-10         12042.0         29185.0              252
7     2023-11          5248.0         14279.0              123
8     2023-12          6835.0         17061.0              148
9     2024-01          4896.0         13628.0              121
10    2024-02          1314.0          4369.0               38
11    2024-03          6059.0         16264.0              143
12    2024-04          3587.0          8598.0               75
13    2024-05          2534.0          6343.0               56
14    2024-06          5467.0         10746.0               94
15    2024-07          1976.0          4665.0               42
16    2024-08          1856.0          4600.0               39
17    2024-09          1726.0          2780.0               25
18    2024-10          2417.0          4693.0               40
```

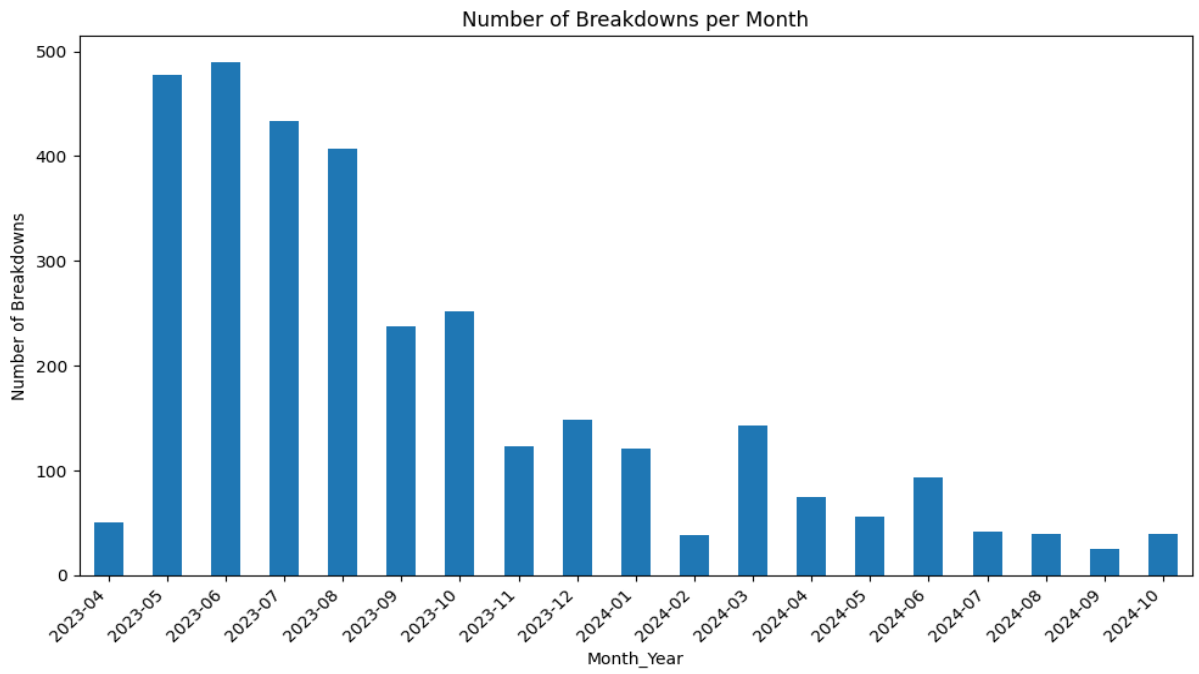
The table provides a monthly summary of bus breakdowns and related data. Here's a breakdown of the columns:

- **MonthYear**: Indicates the month and year for the data.
- **Total Miss KMs**: The total number of kilometers missed due to breakdowns in that month.
- **Total Sch. KMs**: The total number of kilometers scheduled for the buses in that month.
- **Breakdown Count**: The number of breakdown incidents in that month.

Key Observations

- The data spans from April 2023 to October 2024.

- There is a general trend of higher Total Miss KMs and Breakdown Counts in the earlier months of the data (2023-05 to 2023-08) compared to later months.
- The highest values for Total Miss KMs and Breakdown Count occur in May and June 2023.
- The lowest values for Total Miss KMs are in Feb 2024.

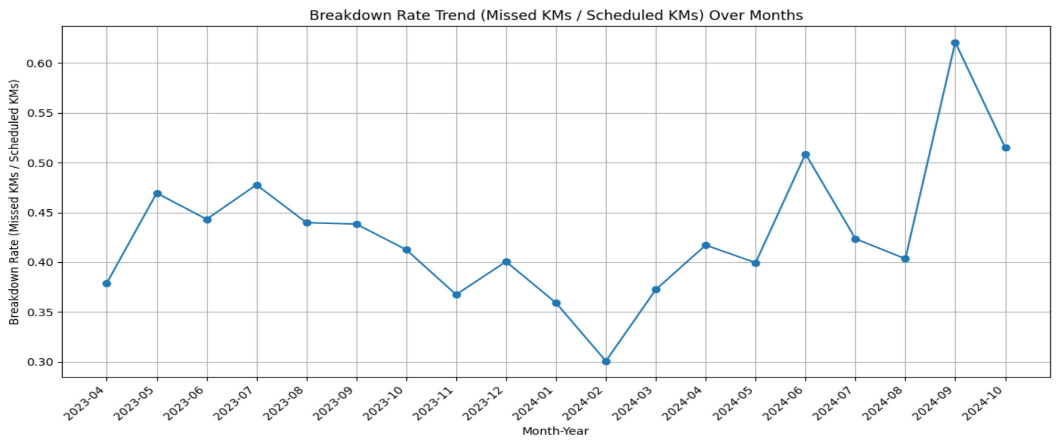


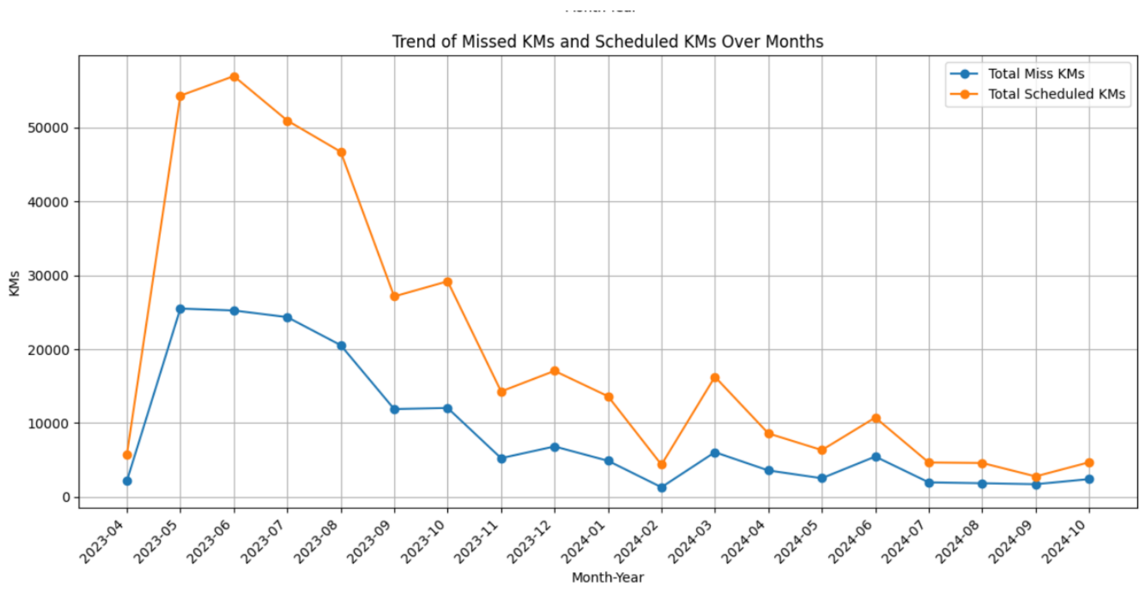
Overall Trend:

- The number of breakdowns is generally higher in the earlier part of the observed period (around mid-2023) and tends to decrease towards the later part (into 2024).

Specific Observations:

- **Peak Breakdowns:** The months with the highest number of breakdowns are May and June 2023.
- **Decline:** There's a noticeable drop in breakdowns from mid-2023 to early 2024.
- **Lowest Breakdowns:** The months with the lowest number of breakdowns are in early 2024, particularly February and September 2024.





```
Breakdown Rate Trend (Missed KMs vs Scheduled KMs over Months):
   MonthYear  Total Miss KMs  Total Sch. KMs  Breakdown Count  Breakdown Rate
0    2023-04          2179.0          5749.0               50        0.379022
1    2023-05         25495.0         54307.0              477        0.469461
2    2023-06         25240.0         56956.0              490        0.443149
3    2023-07         24336.0         50942.0              433        0.477720
4    2023-08         20541.0         46713.0              407        0.439728
5    2023-09         11896.0         27139.0              238        0.438336
6    2023-10         12042.0         29185.0              252        0.412609
7    2023-11          5248.0         14279.0              123        0.367533
8    2023-12          6835.0         17061.0              148        0.400621
9    2024-01          4896.0         13628.0              121        0.359260
10   2024-02          1314.0          4369.0               38        0.300755
11   2024-03          6059.0         16264.0              143        0.372541
12   2024-04          3587.0          8598.0               75        0.417190
13   2024-05          2534.0          6343.0               56        0.399496
14   2024-06          5467.0         10746.0               94        0.508747
15   2024-07          1976.0          4665.0               42        0.423580
16   2024-08          1856.0          4600.0               39        0.403478
17   2024-09          1726.0          2780.0               25        0.620863
18   2024-10          2417.0          4693.0               40        0.515022
```

Observations

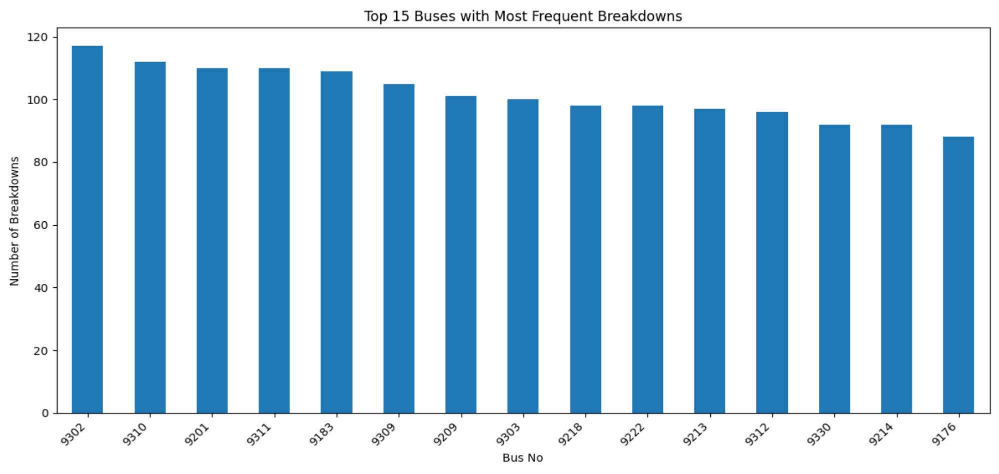
- The data spans from April 2023 to October 2024.
- The breakdown rate varies across the months.
- The highest breakdown rate is in September 2024 (0.620863).
- The lowest breakdown rate is in February 2024 (0.300755).
- Both 'Total Scheduled KMs' and 'Total Miss KMs' show a general decline over the observed period.
- Total Scheduled KMs' is significantly higher than 'Total Miss KMs' throughout the period.
- 'Total Scheduled KMs' starts high in April-May 2023, drops sharply until around October 2023, and then stabilizes at a lower level with some fluctuations.

Interpretation

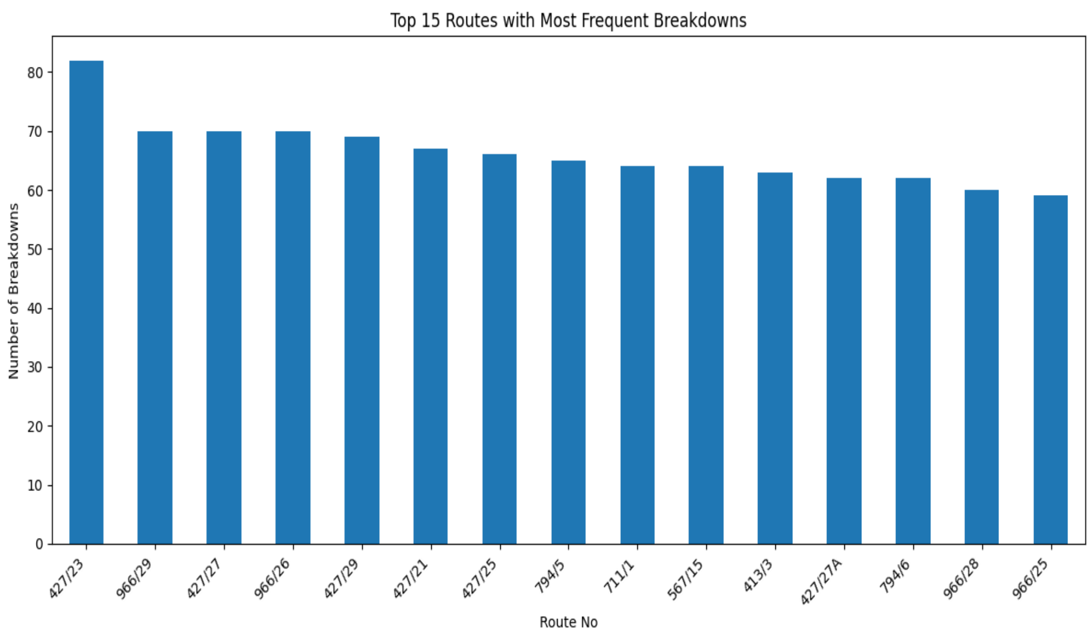
- The breakdown rate represents the proportion of scheduled kilometers that were not completed due to breakdowns.
- A higher breakdown rate indicates a higher impact of breakdowns on the bus service.
- The fluctuations in the breakdown rate suggest that the reliability of the bus service varies from month to month.

Possible Implications

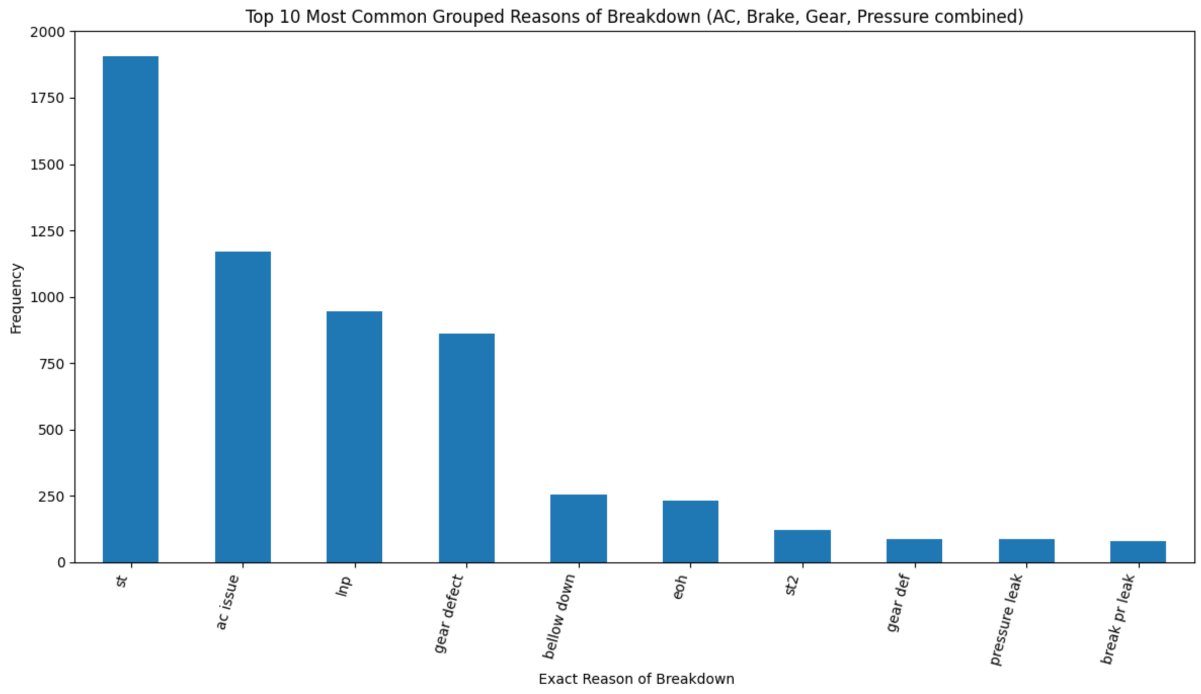
- The high breakdown rate in September 2024 indicates that the service was least reliable in that month.
- The low breakdown rate in February 2024 indicates that the service was most reliable in that month.
- The gap between 'Total Scheduled KMs' and 'Total Miss KMs' widens over time, indicating that the proportion of scheduled kilometers that were actually run increases
- The decline in 'Total Miss KMs' suggests an improvement in service reliability, with fewer kilometers being missed due to breakdowns.
- The increasing gap between scheduled and missed kilometers indicates that the bus service became more effective in fulfilling its planned schedule over the period.



- The chart reveals a substantial difference in the number of breakdowns across the top 15 buses. Some buses experience considerably more breakdowns than others.
- Bus number 9302 has the highest breakdown count, followed closely by 9310, 9201, 9311 and 9183. These buses appear to be disproportionately affected by breakdowns.



- The chart displays the 15 bus routes with the highest number of breakdowns.
- Route "427/23" has the highest number of breakdowns.

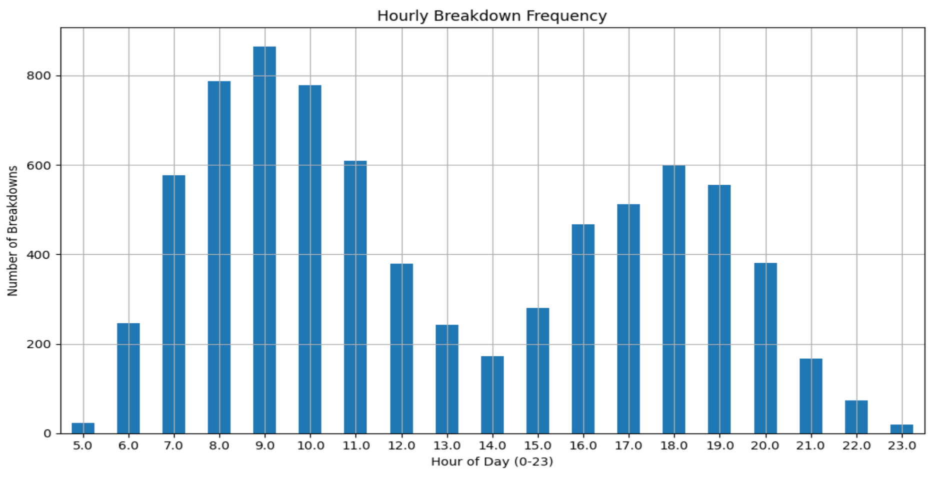


<u>INTERPRETATION:</u>

- The chart shows the frequency of the top 10 grouped reasons for breakdowns.
- "st" is the most frequent reason for breakdowns, followed by "ac issue".
- "break pr leak" has the lowest frequency among the top 10 reasons.
- st" is a major cause of breakdowns.
- AC issues are also a significant contributor to breakdowns.

Implications

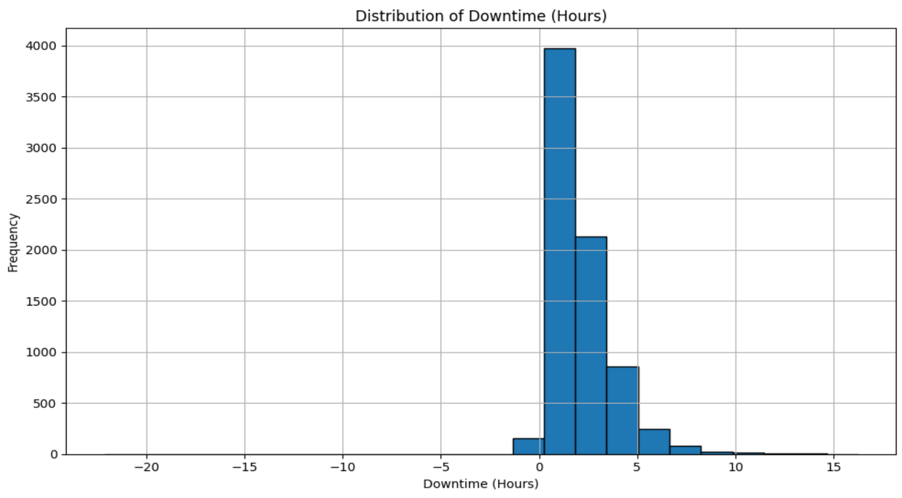
- Maintenance efforts should prioritize addressing the issues causing "st" breakdowns.
- Preventive maintenance or design improvements for AC systems could help reduce breakdowns.



Analysis of breakdown incidents reveals distinct temporal patterns. The highest frequency of breakdowns is observed during the late evening hours (21:00 - 23:00), registering between 21.0 and 23.0 incidents. A secondary, yet notable, peak occurs in the late afternoon (17:00 - 19:00), with breakdown figures ranging from 17.0 to 19.0. Conversely, the early morning period (05:00 - 06:00) exhibits the lowest incidence of breakdowns, recording only 5.0 to 6.0 events.

These temporal variations suggest potential underlying factors:

- The pronounced late evening peak may be attributed to the cumulative effects of operational wear and tear experienced throughout the day, potentially compounded by reduced maintenance activities during these hours.
- The afternoon peak could correlate with periods of heightened operational demand, such as peak traffic or rush hour, leading to increased mechanical strain and a higher likelihood of failures.
- The minimal breakdown occurrences in the early morning likely reflect reduced operational activity and the potential benefit of overnight maintenance or vehicle inactivity.



Data Inconsistencies:

- **Negative Downtime:** Illogical values indicate buses supposedly repaired *before* breakdown. Likely due to data entry errors, timezone issues, or incorrect manual logging.

Zero Downtime Spike:

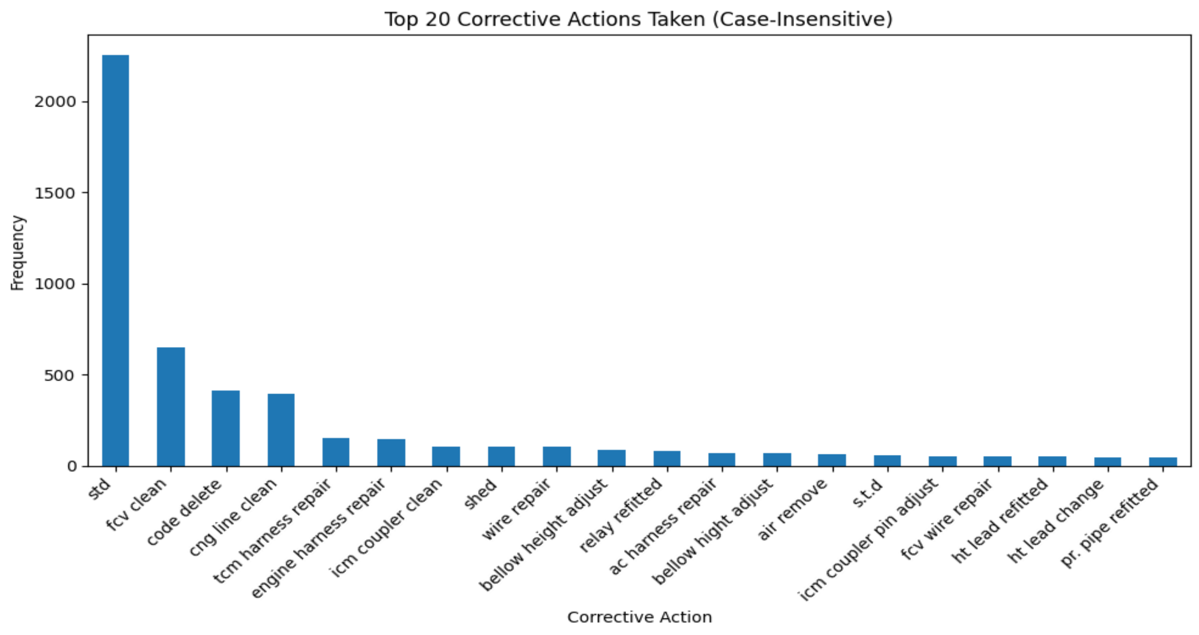
- **High Frequency:** A large number of incidents with zero downtime.
- **Potential Causes:** Immediate repairs *or* data inaccuracies (missing "Put-on road" time), possibly placeholder entries.

Positive Downtime Trend:

- **Right-Skewed Distribution:** Most breakdowns resolved quickly (0-5 hours).
- **Longer Downtimes:** Less frequent, but indicate severe issues requiring more than 10 hours for repair.

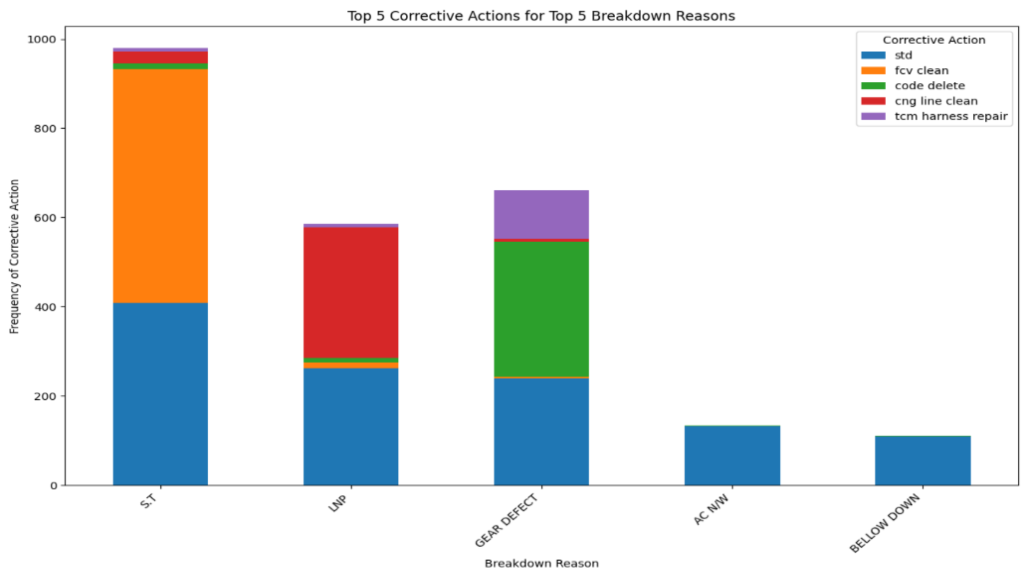
Overall Interpretation:

- Majority of breakdowns are resolved relatively
- Significant data quality issues exist (negative and potentially zero downtime).
- Longer downtimes point to more complex mechanical problems.
- Addressing data inaccuracies is crucial for reliable downtime analysis.



- The chart displays the frequency of the top 20 corrective actions performed.
- **"Std"** has by far the highest frequency, significantly exceeding all other corrective actions. This likely represents a standard procedure or a very common type of fix.
- **"Fcv clean"** is the second most frequent action, but its frequency is considerably lower than "std".

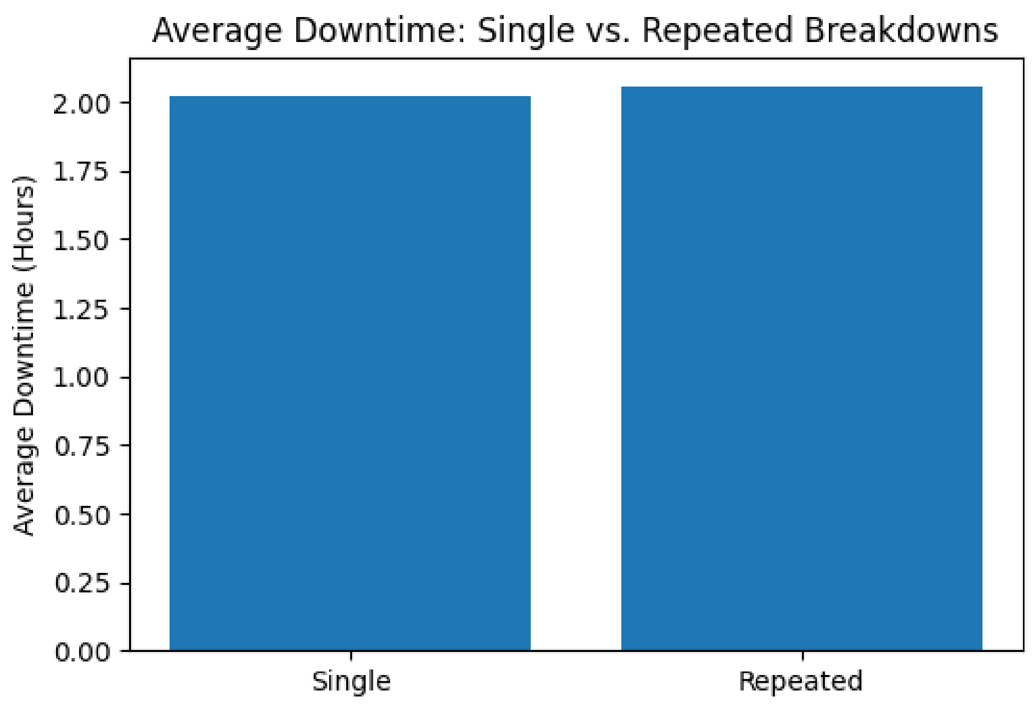
- **"Code delete"** and **"Cng line clean"** have similar frequencies and are the next most common actions after "fcv clean".
- There's a noticeable drop in frequency after "cng line clean".
- The remaining corrective actions all have relatively low and similar frequencies



The chart illustrates which corrective actions are most frequently used to address the top 5 reasons for breakdowns. It helps in understanding the relationship between specific breakdown causes and their typical solutions.

Key Interpretations

- **"std" is the most common corrective action:** Across all top 5 breakdown reasons, "std" is used most frequently. This suggests it's a standard procedure, a common fix, or perhaps the initial step taken in most cases.
- **Breakdown reason "S.T." heavily relies on "std":** For "S.T." breakdowns, "std" is overwhelmingly the dominant corrective action. This indicates a strong link between this particular breakdown reason and the "std" procedure.
- **Other corrective actions are more specialized:** While "std" is prevalent, other corrective actions like "fcv clean," "code delete," "cng line clean," and "tcm harness repair" are used more selectively depending on the breakdown reason.
- **"code delete" is prominent for "GEAR DEFECT":** Compared to other breakdown reasons, "code delete" is used more often in cases of "GEAR DEFECT." This suggests that software or electronic control issues are more likely to be involved in gear-related problems.
- **"cng line clean" is notable for "LNP":** The corrective action "cng line clean" appears more frequently for "LNP" breakdowns, indicating a potential issue with the CNG system in these cases.

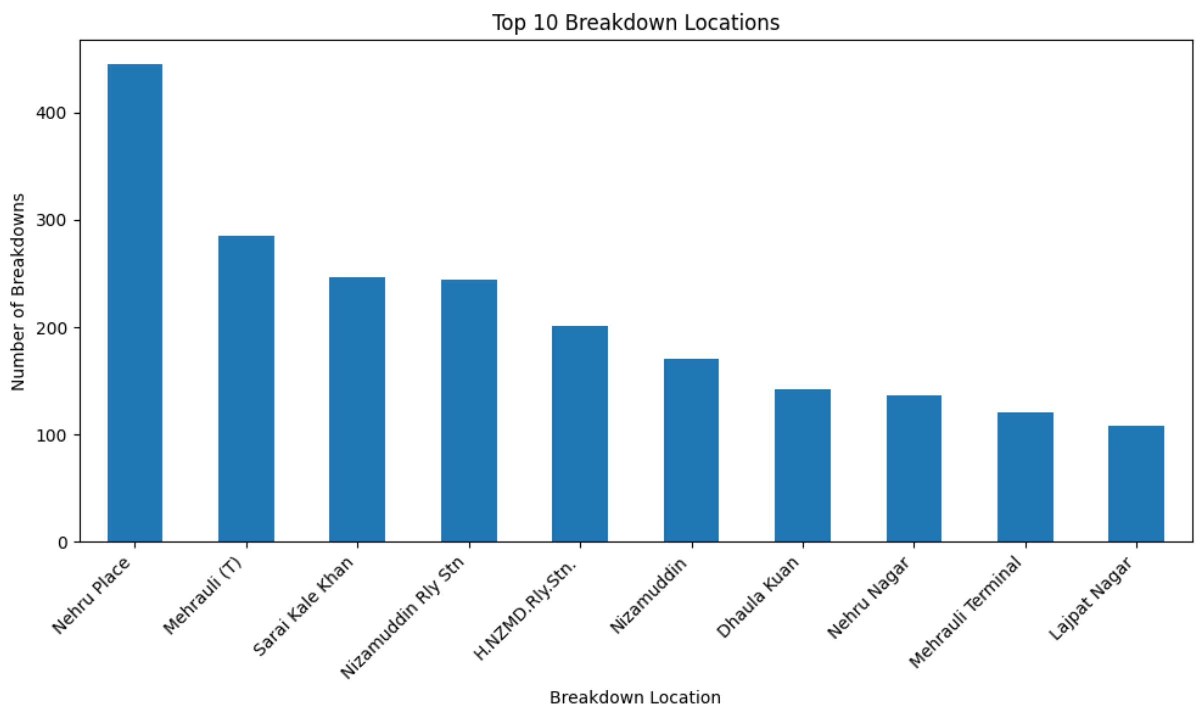


Key Observations

- **Downtime Comparison**: The chart compares the average downtime (in hours) for "Single" breakdowns versus "Repeated" breakdowns.
- **Similar Downtime**: The bars for "Single" and "Repeated" are very close in height.
- **Quantitative Values**:
  - Single breakdowns have an average downtime of approximately 2.02 hours.
  - Repeated breakdowns have an average downtime of approximately 2.06 hours.

Interpretation

- **Minimal Difference**: There is a negligible difference in average downtime between single and repeated breakdowns.
- **No Escalation**: A repeated breakdown does not, on average, translate to a significantly longer repair time.

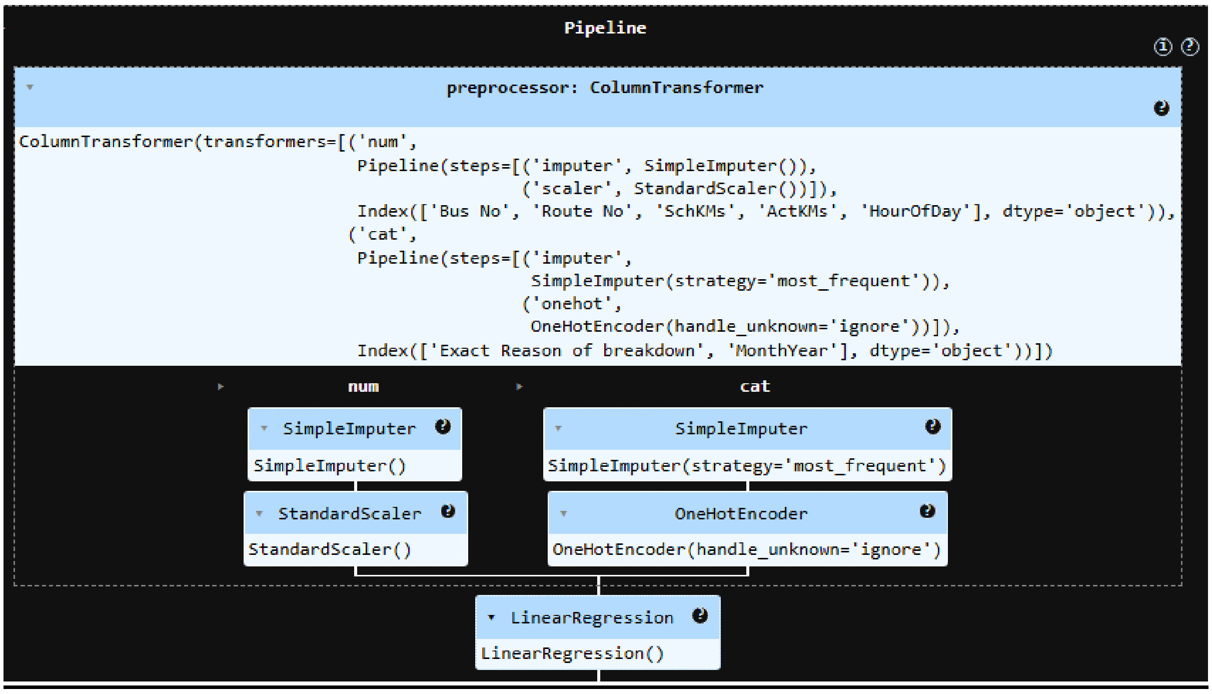


- **Focus**: The chart displays the top 10 locations with the highest number of breakdowns.
- **Dominant Location**: "Nehru Place" has a significantly higher number of breakdowns compared to all other locations.
- **Decreasing Trend**: The number of breakdowns decreases as you move from "Nehru Place" to "Lajpat Nagar."
- **Steepest Drop**: The most substantial decrease in breakdowns occurs between "Nehru Place" and "Mehrauli (T)."
- **Least Breakdowns**: "Lajpat Nagar" records the fewest breakdowns among the top 10 locations.

<u>Preprocessing for ML Algorithms for Machine Learning</u>

The code performs the following essential data preprocessing steps:

- Handles date/time conversions and calculates downtime.
- Extracts relevant features from the date/time information.
- Separates features from the target variable.
- Scales numerical features using StandardScaler.
- Encodes categorical features using OneHotEncoder.
- Splits the data into training and testing sets.



```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer', SimpleImputer()),
                                                 ('scaler', StandardScaler())]),
                                 Index(['Bus No', 'Route No', 'SchKMs', 'ActKMs', 'HourOfDay'], dtype='object')),
                                ('cat',
                                 Pipeline(steps=[('imputer',
                                                  SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',
                                                  OneHotEncoder(handle_unknown='ignore'))]),
                                 Index(['Exact Reason of breakdown', 'MonthYear'], dtype='object'))])
```

```
SimpleImputer()
StandardScaler()
SimpleImputer(strategy='most_frequent')
OneHotEncoder(handle_unknown='ignore')
LinearRegression()
```

This code prepares data for a linear regression model and then trains that model. The model aims to predict a numerical target variable (likely 'Downtime_Hours', based on previous context) using a set of features that include both numerical and categorical data. The image shows the structure of the pipeline created in the code.

- Cleans the data by replacing 'NIL' values with np.nan and converting relevant columns to numeric.
- Sets up a preprocessing pipeline to handle missing values, scale numerical data, and encode categorical data.
- Creates a linear regression model.
- Combines the preprocessing steps and the model into a single pipeline.

- Trains the entire pipeline (preprocessor + model) on the training data.

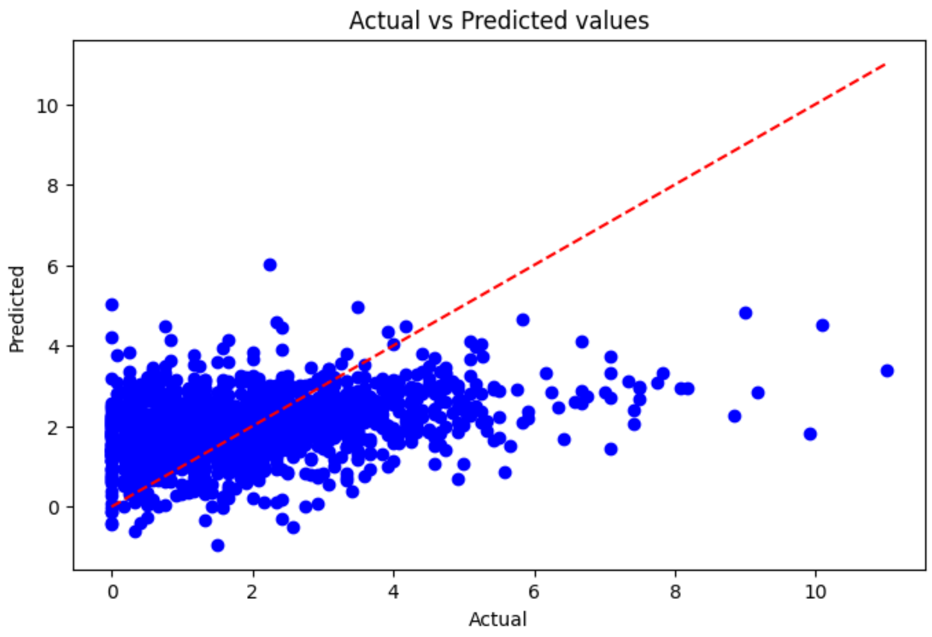
```
Feature Importances:
                                               Feature  Importance
810  cat__Exact Reason of breakdown_S.T,GEAR DEFECT...   10.207122
861       cat__Exact Reason of breakdown_STREEING FAIL    7.110457
247   cat__Exact Reason of breakdown_E.O.H,GATE DEFECT    6.514840
817        cat__Exact Reason of breakdown_S.T- 2 + LNP    5.535273
653    cat__Exact Reason of breakdown_R/O/TYRE PUNCHAR    5.315439
..                                                 ...         ...
265  cat__Exact Reason of breakdown_ENGINE FAN BELT...   -2.934931
556  cat__Exact Reason of breakdown_LNP,WATER PUMP ...   -3.009943
323     cat__Exact Reason of breakdown_F/R/S WHEEL JAM   -3.093197
10      cat__Exact Reason of breakdown_A/C NOT WORKING   -3.218661
636         cat__Exact Reason of breakdown_R/BUS SOUND   -3.689211

[1141 rows x 2 columns]
```

The code calculates and displays the importance of each feature in a linear regression model. The output shows which features are most strongly related to the target variable, helping to understand the factors that influence downtime.

OUTPUT OF LINEAR REGRESSION MODEL

R2 Score: 0.10430851221090898
Mean Absolute Error: 1.1436147047183989

Root Mean Squared Error: 1.4767095441402214



- **R-squared:** The R-squared score is 0.104. This indicates that the model explains only about 10.4% of the variability in the target variable. This is a very low R-squared, suggesting that the model does not fit the data well.
- **Mean Absolute Error:** The MAE is 1.143. This means that, on average, the model's predictions are off by approximately 1.143 units of the target variable.
- **Root Mean Squared Error:** The RMSE is 1.477. RMSE gives a general idea of the magnitude of the errors. Since RMSE is larger than MAE, it indicates that there are some large errors in the predictions.
- **Actual vs. Predicted Plot:**
  - The scatter plot shows that the predicted values do not closely align with the actual values. The points are scattered widely around the dashed line, indicating a weak relationship between predictions and actuals.
  - Most of the predicted values seem to cluster in a narrow range, regardless of the actual values.

Overall Assessment

- The model's performance is poor.
- The low R-squared indicates that the model has very limited predictive power.
- Both MAE and RMSE suggest that the model makes predictions with a non-negligible error.

- The scatter plot visually confirms this, showing a significant discrepancy between predicted and actual values.

```
Random Forest Regressor Model Evaluation Metrics:
  Mean Squared Error (MSE): 5.96
  R-squared (R2): 0.00
  Root Mean Squared Error (RMSE): 2.44

Random Forest Regressor Predictions:
   Actual_Missed_KMs  Random_Forest_Predictions
8                 5                        4.4

1                 5                        1.6
```

The model helps identify buses likely to miss their scheduled kilometers due to breakdowns, aiding in maintenance planning and resource allocation.

 The key takeaway is that the Random Forest Regressor model is performing very poorly. The R2 of 0.00 indicates that the model has no predictive power.

 Errors in Predictions: Both MSE and RMSE suggest that the model's predictions have a significant average error. Looking at the sample predictions, we can see that the model is quite far off from the actual values.

```
K-Nearest Neighbors Classifier Model Evaluation Metrics:
  Accuracy: 1.00
  Precision: 1.00
  Recall: 1.00
  F1-score: 1.00

K-Nearest Neighbors Classifier Predictions:
  Actual_Maintenance_Status KNN_Predictions
5                        OK              OK

1                   Overdue         Overdue
```

This code predicts the maintenance status of buses (e.g., "Needs Repair", "Functional") using a K-Nearest Neighbors (KNN) Classifier. It analyzes historical breakdown data to classify whether a bus requires maintenance based on features like kilometers driven, breakdown reasons, and timestamps.

Model Evaluation Metrics

- **Accuracy: 1.00**

    - Accuracy measures the proportion of correctly classified instances out of the total instances. An accuracy of 1.00 indicates that the model correctly classified all instances in the dataset.
- **Precision: 1.00**
    - Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. A precision of 1.00 means that when the model predicts an instance as positive, it is always correct.
- **Recall: 1.00**
    - Recall measures the proportion of correctly predicted positive instances out of all actual positive instances. A recall of 1.00 means that the model correctly identifies all positive instances.
- **F1-score: 1.00**
    - The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance. An F1-score of 1.00 indicates perfect precision and recall.

```
Support Vector Machine Classifier Model Evaluation Metrics:
  Accuracy: 1.00
  Precision: 1.00
  Recall: 1.00
  F1-score: 1.00

Support Vector Machine Classifier Predictions:
  Actual_Maintenance_Status SVM_Predictions
5                        OK              OK

1                   Overdue         Overdue
```

Model Evaluation Metrics

- **Accuracy: 1.00**
    - The model correctly classified 100% of the instances.
- **Precision: 1.00**
    - When the model predicted a class, it was always correct.
- **Recall: 1.00**
    - The model correctly identified all instances of each actual class.
- **F1-score: 1.00**
    - The model has perfect precision and recall.

Support Vector Machine Classifier Predictions

- The model's predictions match the actual maintenance status for both provided examples:
    - Actual "OK" was predicted as "OK".
    - Actual "Overdue" was predicted as "Overdue".