TTC Subway Delay Data Analysis API Documentation

Project Information

Course: INF1340H - Programming for Data Science
Institution: University of Toronto, Faculty of Information
Instructor: Dr. Maher Elshakankiri
Group: 24
Students:

- Ahmed Balfaqih - 1012820930
- David Kalliny- 1008558614
- Abdullah Qaisar - 1003065416

##

Table of Contents

##

1. Data Loading Functions

load_ttc_data(file_path: str) -> Optional[pd.DataFrame]

Load TTC subway delay data from CSV files with error handling.

Parameters:

- file_path (str): Path to the CSV file to load

Returns:

- Optional[pd.DataFrame]: Loaded DataFrame if successful, None if error occurs

Example:

```
# Load 2024 dataset

df_2024 = load_ttc_data("ttc-subway-delay-data-2024.csv")

if df_2024 is not None:

    print(f"Loaded {len(df_2024)} rows")
```

##

inspect_dataframe(df: pd.DataFrame, df_name: str = "DataFrame") -> pd.DataFrame

Comprehensive inspection of a DataFrame structure and quality.

Parameters:

- df (pd.DataFrame): DataFrame to inspect
- df_name (str, optional): Name of the dataframe for display purposes. Default: "DataFrame"

Returns:

- pd.DataFrame: DataFrame containing missing values summary (columns with missing data only)

Example:

```
missing_summary = inspect_dataframe(df_2024, "2024 Dataset")

# Prints comprehensive inspection report

# Returns DataFrame with missing value details
```

##

2. Data Cleaning Functions

standardize_column_names(df: pd.DataFrame) -> pd.DataFrame

Standardize column names to lowercase with underscores.

Parameters:

- df (pd.DataFrame): Input DataFrame with original column names

Returns:

- pd.DataFrame: DataFrame with standardized column names

Example:

Before: "Min Delay", "Min Gap"

After: "min_delay", "min_gap"

df_cleaned = standardize_column_names(df)

##

handle_empty_strings(df: pd.DataFrame) -> pd.DataFrame

Replace empty strings and 'None' strings with NaN in object columns.

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame with empty strings replaced by NaN

Example:

df_cleaned = handle_empty_strings(df)

Empty strings in object columns are now NaN

##

convert_data_types(df: pd.DataFrame) -> pd.DataFrame

Convert columns to appropriate data types for analysis.

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame with converted data types

Example:

df_cleaned = convert_data_types(df)

Date column is now datetime

min_delay and min_gap are now numeric

day, station, code are now category type

##

filter_subway_records(df: pd.DataFrame) -> pd.DataFrame

Filter to keep only subway records (exclude bus, streetcar, etc.).

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame containing only subway line records

Example:

df_subway = filter_subway_records(df)

Only contains records for YU, BD, SHP lines

##

standardize_categorical_values(df: pd.DataFrame) -> pd.DataFrame

Standardize categorical values and handle station name variants.

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame with standardized categorical values

Example:

df_cleaned = standardize_categorical_values(df)

"YONGE-UNIVERSITY" becomes "YU"

Station names are uppercase and standardized

##

handle_missing_values(df: pd.DataFrame, strategy: str = 'remove', create_flags: bool = True) -> pd.DataFrame

Handle missing values in the dataframe using specified strategy.

Parameters:

- df (pd.DataFrame): Input DataFrame
- strategy (str, optional): Strategy to use ('remove', 'impute', or 'flag'). Default: 'remove'
- create_flags (bool, optional): Whether to create missing flags for useful columns (bound, line). Default: True

Returns:

- pd.DataFrame: DataFrame with missing values handled according to strategy

Example:

Remove rows with missing critical data

df_cleaned = handle_missing_values(df, strategy='remove', create_flags=True)

Impute missing values

df_imputed = handle_missing_values(df, strategy='impute')

##

handle_data_errors(df: pd.DataFrame, column: str, use_percentile: bool = True, percentile: int = 99) -> pd.DataFrame

Remove obvious data errors (negative values, extreme values) using percentile cap.

Parameters:

- df (pd.DataFrame): Input DataFrame
- column (str): Column name to check for errors
- use_percentile (bool, optional): Whether to use percentile capping. Default: True
- percentile (int, optional): Percentile to use for capping. Default: 99

Returns:

- pd.DataFrame: DataFrame with data errors handled

Example:

Handle errors in min_delay column

df_cleaned = handle_data_errors(df, 'min_delay', use_percentile=True, percentile=99)

Negative values removed, extreme values capped at 99th percentile

##

create_datetime_column(df: pd.DataFrame) -> pd.DataFrame

Create a combined datetime column from Date and Time columns.

Parameters:

- df (pd.DataFrame): Input DataFrame with 'date' and 'time' columns

Returns:

- pd.DataFrame: DataFrame with datetime column and extracted time features

Example:

df_with_datetime = create_datetime_column(df)

New columns: datetime, hour, month, year, day_of_week, weekday, is_peak, etc.

##

clean_ttc_data(df: pd.DataFrame, remove_outliers: bool = False, outlier_method: str = 'cap', filter_subway: bool = True) -> pd.DataFrame

Main data cleaning function that applies all cleaning steps in sequence.

Parameters:

- df (pd.DataFrame): Input DataFrame with raw TTC delay data
- remove_outliers (bool, optional): Whether to remove outliers (not currently used). Default: False
- outlier_method (str, optional): Method for handling outliers (not currently used). Default: 'cap'
- filter_subway (bool, optional): Whether to filter to subway records only. Default: True

Returns:

- pd.DataFrame: Cleaned DataFrame ready for analysis

Example:

df_cleaned = clean_ttc_data(df_2024, filter_subway=True)

Applies all 9 cleaning steps

Returns cleaned DataFrame ready for analysis

##

3. Descriptive Analytics Functions

calculate_summary_statistics(df: pd.DataFrame) -> pd.DataFrame

Calculate comprehensive summary statistics for numerical columns.

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame with summary statistics for each numerical column

Example:

```
stats = calculate_summary_statistics(df_cleaned)

print(stats)

# Returns DataFrame with all summary statistics
```

##########

calculate_missing_value_proportions(df: pd.DataFrame) -> pd.DataFrame

Calculate the proportion of missing values for each variable.

Parameters:

- df (pd.DataFrame): Input DataFrame

Returns:

- pd.DataFrame: DataFrame with missing value counts and proportions for each column

Example:

```
missing_df = calculate_missing_value_proportions(df)

# Returns DataFrame sorted by missing count (descending)
```

##########

generate_frequency_distributions(df: pd.DataFrame, categorical_cols: Optional[list] = None) -> Dict[str, pd.Series]

Generate frequency distributions for categorical data.

Parameters:

- df (pd.DataFrame): Input DataFrame
- categorical_cols (Optional[list]): List of categorical column names. If None, auto-detect.

Returns:

- Dict[str, pd.Series]: Dictionary mapping column names to their frequency distributions

Example:

```
freq_dist = generate_frequency_distributions(df, ['line', 'station', 'code'])

# Returns: {'line': Series, 'station': Series, 'code': Series}

print(freq_dist['line'].head(10))  # Top 10 most frequent lines
```

##

segment_data_by_category(df: pd.DataFrame, segment_col: str, numeric_cols: Optional[list] = None) -> pd.DataFrame

Segment the data by relevant categories and calculate statistics for each segment. Statistics Calculated: count, mean, median, std, min, max

Parameters:

- df (pd.DataFrame): Input DataFrame
- segment_col (str): Column name to segment by (e.g., 'line', 'station', 'weekday')
- numeric_cols (Optional[list]): List of numeric columns to analyze. If None, auto-detect.

Returns:

- pd.DataFrame: DataFrame with summary statistics for each segment

Example:

```
# Segment by line

line_stats = segment_data_by_category(df, 'line', ['min_delay', 'min_gap'])

# Returns DataFrame with statistics for each line (YU, BD, SHP)
```

##

run_descriptive_analytics(df_cleaned: pd.DataFrame) -> Dict[str, Any]

Run comprehensive descriptive analytics on cleaned TTC delay data.

Parameters:

- df_cleaned (pd.DataFrame): Cleaned DataFrame from the data cleaning pipeline

Returns:

- Dict[str, Any]: Dictionary containing:
  - summary_stats: Summary statistics DataFrame
  - missing_proportions: Missing value proportions DataFrame
  - frequency_distributions: Dictionary of frequency distributions
  - segment_stats: Dictionary of segment statistics by category

Example:

```
results_desc = run_descriptive_analytics(df_cleaned)

print(results_desc['summary_stats'])

print(results_desc['frequency_distributions']['line'])
```

##

4. Diagnostic Analytics Functions

display_code_categories() -> None

Display a table explaining TTC delay code categories.

Returns:

- None: Prints table to console

Shows:

- Prefix codes (MU, PU, EU, TU) and their meanings
- Examples of codes in each category

Example:

display_code_categories()

Prints table showing:

MU: Train mechanical issues (MUI, MUIS, MUPAA)

PU: Passenger-related delays (PUTDN)

EU: Signal/electrical failures (EUCD, EUBK, EUBO)

TU: Tunnel/track issues (TUNOA, TUO, TUOPO, TUOS)

##

run_diagnostic_analytics(df_cleaned: pd.DataFrame) -> Dict[str, Any]

Perform diagnostic analytics on cleaned TTC delay data.

Parameters:

- df_cleaned (pd.DataFrame): Cleaned DataFrame from the data cleaning pipeline. Must contain columns: min_delay, min_gap, line, station, code, is_peak

Returns:

- Dict[str, Any]: Dictionary containing:
  - corr_numeric: Correlation matrix for numeric columns
  - xtab_line_cause: Cross-tabulation of Line x Code
  - xtab_station_peak: Cross-tabulation of Station x is_peak
  - regression_gap_delay: Regression results (slope, intercept, r-value, p-value, etc.)
  - ttest_peak_offpeak: T-test results (mean_peak, mean_offpeak, t_stat, pvalue)

Example:

results_diag = run_diagnostic_analytics(df_cleaned)

print(results_diag['corr_numeric']) # Correlation matrix

print(results_diag['regression_gap_delay']['rvalue']) # Correlation coefficient

##

5. Predictive Analytics Functions

prepare_features_for_prediction(df: pd.DataFrame) -> Tuple[pd.DataFrame, pd.Series]

Prepare features and target variables for delay classification.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- Tuple[pd.DataFrame, pd.Series]: (X, y) where X is features DataFrame and y is target Series

Example:

X, y = prepare_features_for_prediction(df_cleaned)

print(f"Features: {X.shape[1]}, Samples: {len(X)}")

print(f"Long delays: {y.sum()}, Short delays: {(y==0).sum()}")

##

train_and_evaluate_models(X_train: pd.DataFrame, X_test: pd.DataFrame, y_train: pd.Series, y_test: pd.Series) -> Dict[str, Any]

Train and evaluate three classification models: Logistic Regression, Decision Tree, and Random Forest.

Parameters:

- X_train (pd.DataFrame): Training features
- X_test (pd.DataFrame): Testing features
- y_train (pd.Series): Training target
- y_test (pd.Series): Testing target

Returns:

- Dict[str, Any]: Dictionary containing:

- models: Dictionary with model objects ('Logistic Regression', 'Decision Tree', 'Random Forest')
- predictions: Dictionary with test predictions for each model
- metrics: Dictionary with accuracy and f1 scores for each model
- feature_importance: Dictionary with feature importance for tree-based models ('Decision Tree', 'Random Forest')

Example:

```
results = train_and_evaluate_models(X_train, X_test, y_train, y_test)

print(f"Logistic Regression Accuracy: {results['accuracy_lr']:.4f}")

print(f"Random Forest Accuracy: {results['accuracy_rf']:.4f}")
```

##

run_predictive_analytics(df_cleaned: pd.DataFrame) -> Dict[str, Any]

Run complete predictive analytics pipeline.

Parameters:

- df_cleaned (pd.DataFrame): Cleaned DataFrame from the data cleaning pipeline

Returns:

- Dict[str, Any]: Dictionary containing all model results, metrics, and feature importance

Example:

```
results_predictive = run_predictive_analytics(df_cleaned)

# Trains models, evaluates performance, creates visualizations

# Returns comprehensive results dictionary
```

##

6. Data Visualization

visualize_data_quality_before_after(df_before: pd.DataFrame, df_after: pd.DataFrame) -> None

Visualize data quality improvements from the cleaning process.

Parameters:

- df_before (pd.DataFrame): Original DataFrame before cleaning
- df_after (pd.DataFrame): Cleaned DataFrame after cleaning

Returns:

- None: Displays visualization

Example:

visualize_data_quality_before_after(df_original, df_cleaned)

Shows before/after comparison charts

##

visualize_delay_distributions(df: pd.DataFrame) -> None

Create histograms and box plots showing delay distributions.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- None: Displays visualization

Example:

visualize_delay_distributions(df_cleaned)

Displays 2x2 grid of distribution visualizations

##

visualize_correlation_heatmap(df: pd.DataFrame) -> None

Create correlation heatmap for numeric variables.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- None: Displays visualization

Example:

visualize_correlation_heatmap(df_cleaned)

Displays correlation matrix heatmap

##

visualize_trends_over_time(df: pd.DataFrame) -> None

Create time series visualizations showing delay trends.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- None: Displays visualization

Example:

visualize_trends_over_time(df_cleaned)

Displays 2x2 grid of time series visualizations

##

visualize_comparative_analysis(df: pd.DataFrame) -> None

Create bar charts and pie charts comparing delays across categories.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- None: Displays visualization

Example:

visualize_comparative_analysis(df_cleaned)

Displays 2x2 grid of comparative visualizations

##

visualize_predictive_model_results(results: Dict[str, Any], X_test: pd.DataFrame, y_test: pd.Series) -> None

Visualize predictive model performance and insights.

Parameters:

- results (Dict[str, Any]): Results dictionary from train_and_evaluate_models()
- X_test (pd.DataFrame): Test features
- y_test (pd.Series): Test target

Returns:

- None: Displays visualization

Example:

visualize_predictive_model_results(results, X_test, y_test)

Displays model performance visualizations

##

visualize_multivariate_relationships(df: pd.DataFrame) -> None

Create a scatter plot matrix showing relationships between key numeric variables.

Parameters:

- df (pd.DataFrame): Cleaned DataFrame

Returns:

- None: Displays visualization

Example:

visualize_multivariate_relationships(df_cleaned)

Displays pair plot matrix

##

run_comprehensive_visualizations(df_cleaned: pd.DataFrame, df_before: Optional[pd.DataFrame] = None, results_predictive: Optional[Dict[str, Any]] = None) -> None

Optional function to run additional visualizations.

Parameters:

- df_cleaned (pd.DataFrame): Cleaned DataFrame
- df_before (Optional[pd.DataFrame]): Optional raw DataFrame for before/after comparison
- results_predictive (Optional[Dict[str, Any]]): Optional predictive analytics results

Returns:

- None: Displays visualizations if applicable

Example:

run_comprehensive_visualizations(df_cleaned, df_original, results_predictive)

##

7. Main Execution Functions

main() -> Tuple[Optional[pd.DataFrame], Optional[pd.DataFrame], Optional[Dict[str, Any]], Optional[Dict[str, Any]]]

Main execution function that orchestrates the entire analysis pipeline.

Returns:

- Tuple[Optional[pd.DataFrame], Optional[pd.DataFrame], Optional[Dict[str, Any]], Optional[Dict[str, Any]]]:
  - df_final: Cleaned and combined DataFrame
  - df_original: Original combined DataFrame (before cleaning)
  - results_desc: Descriptive analytics results
  - results_diag: Diagnostic analytics results

Example:

```
df_final, df_original, results_desc, results_diag = main()

# Executes complete analysis pipeline

# Returns cleaned data and analytics results
```