B.Tech AI&DS

Anna University

MIT Campus

Chromepet

AZ5412

Data Visualization Lab

Handcrafted By:

MIRSHA A. K.

Data Analyst | MLOps Engineer | AI Strategist

Linkedin: *www.linkedin.com/in/mirshamorningstar*

GitHub **Email** **Phone** 9940358967

Table of Contents

1. Executive Summary

1.1 Purpose

The purpose of this project is to create an interactive and user-friendly web page to explore and visualize a US accident dataset (2016-2023). The dashboard aims to uncover key insights and patterns related to accidents, including their severity and the influence of various factors such as location, time, and weather conditions.

1.2 Target Users

The primary users of this dashboard include policymakers, researchers, traffic safety analysts, and the general public. These users can leverage the dashboard to understand accident trends, improve road safety measures, and make data-driven decisions.

1.3 Unique Value Proposition

This dashboard stands out by offering a comprehensive and intuitive platform for analyzing US accident data. It combines various features such as data quality visualization, feature engineering, preprocessing, and both general and inferential visualizations. These tools enable users to gain deeper insights into accident data, making it a valuable resource for improving traffic safety and informing policy decisions.

2. Introduction

2.1 Connecting Our Dashboard to Users

Our US Accidents Dashboard is designed to bridge the gap between complex data and user-friendly analysis. By offering an interactive and intuitive platform, we enable users to easily explore and understand accident data. We have deployed our project using Streamlit Cloud, making it easily accessible to users anytime, anywhere, without the need for complex installations or configurations.

2.2 Motivation and Purpose

The motivation behind this project is to leverage data to improve road safety. With thousands of accidents occurring every day, understanding the factors that contribute to these incidents is crucial. The purpose of this dashboard is to provide a tool that helps users identify trends, analyze patterns, and ultimately contribute to reducing accidents on the roads.

2.3 Filling the Market Void

Currently, there is a lack of accessible, comprehensive tools for analyzing US accident data. Many existing solutions are either too complex for the average user or too simplistic to provide meaningful insights. Our dashboard fills this void by combining ease of use with powerful analytical capabilities, offering a unique solution that meets the needs of a wide range of users.

2.4 Main Objectives

The main objectives of our dashboard are:

- To provide an interactive platform for exploring US accident data.
- To visualize data quality, including missing values and outliers.
- Data Preparation for Exploration: Normalize, bin, and sample the data to prepare it for exploration, ensuring that it is in a suitable format for analysis.
- To enable detailed analysis of accident factors such as location, time, and weather.
- To filter the data for analysis
- To support feature engineering and data preprocessing for deeper insights.
- To offer both general and inferential visualizations to help users draw meaningful conclusions from the data.
- To design and Creation of Data Visualizations: Develop visually appealing and informative data visualizations to present insights and patterns discovered during the analysis.
- To design and Evaluation of Color Palettes: Design and evaluate color palettes for visualization based on principles of perception, ensuring that colors are chosen to enhance readability and interpretability of visualizations.

3. Market Analysis

3.1 Market Demand

The demand for data-driven insights in the transportation and safety sector is at an all-time high. As cities grow and the number of vehicles on the road increases, there is a pressing need for tools that can help analyze and mitigate traffic accidents. Governments, city planners, and safety organizations are seeking advanced solutions to improve road safety and reduce accident rates. This demand creates a significant opportunity for our US Accidents Dashboard.Our dashboard stands out in the market as a versatile, user-friendly, and comprehensive tool for traffic accident analysis, meeting the needs of a broad spectrum of users from policymakers to the general public.

4. Unique Selling Points (USPs)

4.1. Streamlined Accessibility

Our project offers easy access to complex data analysis tools through deployment on Streamlit Cloud. Users can access the dashboard from anywhere with an internet connection, eliminating the need for software installations or technical setups.

4.2. Comprehensive Analysis Features

Unlike many existing solutions, our dashboard provides a comprehensive suite of analysis tools. From data quality assessment to feature engineering and inferential visualizations, users can perform a wide range of analyses without switching between multiple platforms.

4.3. Intuitive User Interface

With Streamlit's user-friendly interface, users of all technical backgrounds can navigate the dashboard effortlessly. Clear visualizations and interactive controls make data exploration and interpretation straightforward and engaging.

4.4. Cost-Effective Solution

Built on open-source technology and deployed on Streamlit Cloud, our project offers a cost-effective alternative to expensive proprietary software. Users can access powerful analysis capabilities without the prohibitive costs associated with professional-grade tools.

4.5. Real-Time Data Analysis

By leveraging Streamlit Cloud's capabilities, our dashboard can analyze and visualize real-time data updates. This feature ensures that users always have access to the most up-to-date insights, allowing for timely decision-making and response to changing conditions.

4.6. Customizable Visualizations

Users can tailor the visualizations to their specific needs and preferences. With options to change color palettes, adjust plot sizes, and select specific data subsets, users can create customized views that highlight the insights most relevant to their objectives.

4.7. Educational Value

Beyond its analytical capabilities, our project serves as an educational tool for understanding traffic accident data. Users can learn about data quality assessment,

preprocessing techniques, and inferential analysis methods through interactive exploration of real-world datasets.

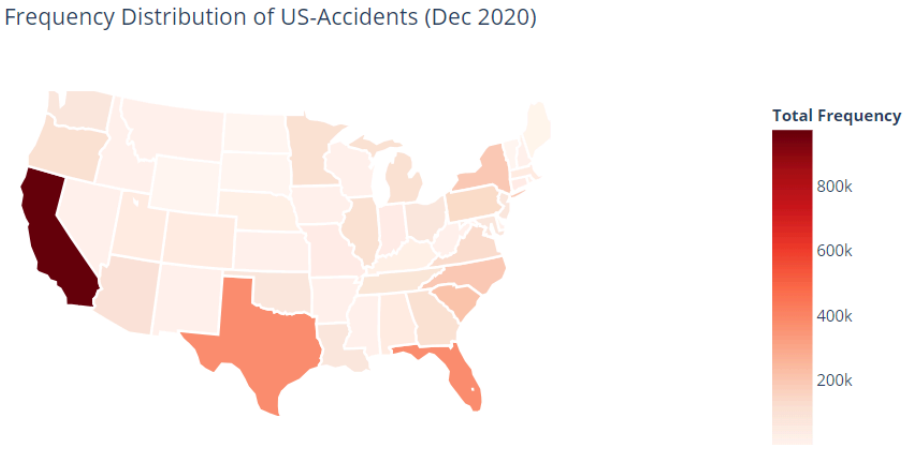
4.8. Community Support and Collaboration

As part of the Streamlit ecosystem, our project benefits from a vibrant community of users and developers. Users can share insights, collaborate on analysis projects, and contribute to the ongoing development and improvement of the dashboard.

5. Dataset

Dataset : US-Accidents: A Countrywide Traffic Accident Dataset

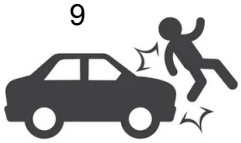
This is a countrywide traffic accident dataset, which covers 49 states of the United States. The data is continuously being collected from February 2016 - 2023, using several data providers, including multiple APIs that provide streaming traffic event data. These APIs broadcast traffic events captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. Currently, there are about 0.5 million accident records in this dataset. Check the below descriptions for more detailed information.



A Detailed Breakthrough

- Dataset Source: The dataset is sourced from [relevant source, e.g., Kaggle].
- Data Structure: The dataset includes 46 columns, each providing different details about the accidents. Key columns include:
    1) **ID** - This is a unique identifier of the accident record.
    2) **Severity** - Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).
    3) **Start_Time** - Shows start time of the accident in the local time zone.
    4) **End_Time** - Shows end time of the accident in the local time zone. End time here refers to when the impact of an accident on traffic flow was dismissed.
    5) **Start_Lat** - Shows latitude in GPS coordinate of the start point.
    6) **Start_Lng** - Shows longitude in GPS coordinate of the start point.
    7) **End_Lat** - Shows latitude in GPS coordinate of the end point.
    8) **End_Lng Shows** - longitude in GPS coordinate of the end point.
    9) **Distance(mi)** - The length of the road extent affected by the accident.
    10) **Description** - Shows natural language description of the accident.
    11) **Number** - Shows the street number in the address field.
    12) **Street** - Shows the street name in the address field.
    13) **City** - Shows the city in the address field.
    14) **County** - Shows the county in the address field.
    15) **State** - Shows the state in the address field.
    16) **ZipCode** - Shows the zip code in the address field.
    17) **Country** - Shows the country in the address field.
    18) **Timezone** - Shows timezone based on the location of the accident (eastern, central, etc.).
    19) **Airport_Code** - Denotes an airport-based weather station which is the closest one to location of the accident.
    20) **Weather_Timestamp** - Shows the time-stamp of a weather observation record (in local time).
    21) **Temperature(F)** - Shows the temperature (in Fahrenheit).
    22) **Wind_Chill(F)** - Shows the wind chill (in Fahrenheit).
    23) **Humidity(%)** - Shows the humidity (in percentage).
    24) **Pressure(in)** - Shows the air pressure (in inches).
    25) **Visibility(mi)** - Shows visibility (in miles).
    26) **Wind_Direction** - Shows wind direction.

27) **Wind_Speed(mph)** - Shows wind speed (in miles per hour).
28) Precipitation(in) - Shows precipitation amount in inches, if there is any.
29) **Weather_Condition** - Shows the weather condition (rain, snow, thunderstorm, fog, etc.)
30) **Amenity** - A POI annotation which indicates presence of amenity in a nearby location.
31) **Bump** - A POI annotation which indicates presence of speed bump or hump in a nearby location.
32) **Crossing** - A POI annotation which indicates presence of crossing in a nearby location.
33) **Give_Way** - A POI annotation which indicates presence of give_way in a nearby location.
34) **Junction** - A POI annotation which indicates the presence of a junction in a nearby location.
35) **No_Exit** - A POI annotation which indicates presence of no_exit in a nearby location.
36) **Railway** - A POI annotation which indicates presence of railway in a nearby location.
37) **Roundabout** - A POI annotation which indicates the presence of a roundabout in a nearby location.
38) **Station** - A POI annotation which indicates presence of a station in a nearby location.
39) **Stop** - A POI annotation which indicates presence of stop in a nearby location.
40) **Traffic_Calming** - A POI annotation which indicates presence of traffic_calming in a nearby location.
41) **Traffic_Signal** - A POI annotation which indicates presence of traffic_signal in a nearby location.
42) **Turning_Loop** - A POI annotation which indicates presence of turning_loop in a nearby location.
43) **Sunrise_Sunset** - Shows the period of day (i.e. day or night) based on sunrise/sunset.
44) **Civil_Twilight** - Shows the period of day (i.e. day or night) based on civil twilight.
45) **Nautical_Twilight** - Shows the period of day (i.e. day or night) based on nautical twilight.
46) **Astronomical_Twilight** - Shows the period of day (i.e. day or night) based on astronomical twilight.
47) **Year:** The year component extracted from the Start_Time attribute.
48) **Month:** The month component extracted from the Start_Time attribute.



49) **Day:** The day component extracted from the Start_Time attribute.
50) **Hour:** The hour component extracted from the Start_Time attribute.
51) **Minute:** The minute component extracted from the Start_Time attribute.
52) **Second:** The second component extracted from the Start_Time attribute.
53) **Hour_Category:** A categorical attribute that categorizes the accidents based on the hour of the day (e.g., morning, afternoon, evening, night).

6. Plots

1. Heatmap for Missing Values

- A heatmap visually represents missing data in a dataset, highlighting the presence and pattern of missing values across different features.

2. Pie Chart

- A pie chart displays data as slices of a circle, showing the relative proportions of different categories within a dataset.

3. Bar Plot

- A bar plot uses rectangular bars to represent and compare the frequency or value of different categories.

4. Violin Plot for Outlier

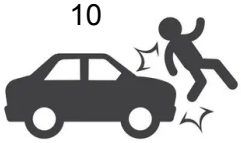
- A violin plot combines a box plot with a kernel density plot, showing the distribution, probability density, and presence of outliers in the data.

5. Box Plot for Outliers

- A box plot summarizes the distribution of a dataset, highlighting the median, quartiles, and potential outliers through the use of "whiskers" and individual points.

6. Histogram for Data Distribution

- A histogram displays the frequency distribution of a dataset by grouping data into bins and showing the count of data points in each bin.



7. Quantile-Quantile Plot for Normalization Visualization

- A quantile-quantile (Q-Q) plot compares the quantiles of a dataset to the quantiles of a theoretical distribution, helping to assess if the data follows the desired distribution, such as normality.

8. Bar Plot for Binning

- A bar plot for binning groups continuous data into discrete bins and represents the count or frequency of data points in each bin.

9. Scatter Plot

- A scatter plot displays individual data points on a two-dimensional plane, showing the relationship between two numerical variables.

10. Line Plot

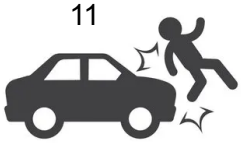
- A line plot connects data points with lines, typically used to show trends over time or the relationship between two continuous variables.

11. Rose Plot

- A rose plot (circular bar plot or wind rose) displays data in circular segments, often used to represent cyclic data like wind direction and speed.

12. Folium Map to Visualize US Accident Places

- A Folium map uses the Folium library to create interactive maps, plotting locations of accidents across the United States for geographical analysis.



7. Description of each page

1. Authentication

- **Feature:** Secure login system requiring users to enter their credentials (username and password) to access the web page.
- **Purpose:** Ensures that only authorized users can access the data and analysis tools, maintaining data privacy and security.

2. Change Theme

- **Feature:** Option to change the dashboard's theme.
- **Purpose:** Allows users to customize the visual appearance of the dashboard to their preference, enhancing the user experience.

3. Color Palette Picker

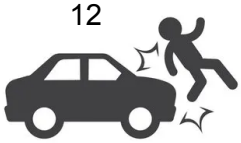
- **Feature:** A tool for users to select different color palettes.
- **Purpose:** Extracting color palettes from images. This app allows users to upload images, apply enhancements, and then extract color palettes using various machine learning clustering algorithms. Users can adjust parameters such as palette size, sample size, and image enhancements. The color palettes can be visualized and adopted for use in matplotlib or plotly plots.

4. Specific Exploration of Data

- **Feature:** Tools for in-depth exploration of specific subsets of the data.
- **Purpose:** Allows users to delve into particular aspects of the dataset, such as filtering accidents by location, time period, or weather conditions.

5. General Exploration of Data

- This Streamlit page is an Exploratory Data Analysis (EDA) tool for analyzing datasets, particularly the US Accidents dataset. Users can upload their CSV files or use a sample dataset provided.
- The page generates a comprehensive profiling report using the pandas-profiling library, including an overview, alerts, missing values, correlation heatmap, and



detailed visualizations for each variable. This enables users to gain insights and understand the dataset's structure and quality effectively

6. Visualization of Data Quality

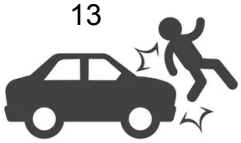
- **Feature:** Visual tools to assess and display the quality of the dataset, including:
  - Missing Values Analysis: Identifies and visualizes missing data in the dataset.
  - Irregular Cardinality: Detects and visualizes unusual or unexpected frequencies in categorical data.
  - Outlier Detection: Identifies and visualizes data points that deviate significantly from the norm.
  - Plots: Various plots to visually represent data quality issues, such as heatmap, pie, bar chart for missing values and violin and box plots for outliers.
- **Purpose:** Helps users understand the completeness and accuracy of the data, highlighting areas with missing or inconsistent entries, and identifying anomalies.

7. Feature Engineering and Data Correlation

- **Feature:** Tools for creating new features and analyzing correlations between different variables.
- **Purpose:** Enhances the data's analytical potential by deriving new insights and understanding the relationships between variables.

8. Data Preprocessing and Preparation

- **Feature:** Modules for data cleaning, normalization, and transformation, including:
  - Normalization: Standardizes data to a common scale, improving the performance of analysis techniques.
  - Binning: Groups continuous data into bins or intervals to simplify analysis.
  - Plots: Visual representations of the preprocessing steps, such as histograms for binning and to understand normalization using Quantile-Quantile plot .



9. Comparison of 25+ Classifier model

- This Streamlit page is designed to compare the performance of various machine learning classification algorithms using the lazypredict library. Users can upload their own CSV files or use an example dataset provided. The page allows users to customize dataset attributes and choose from top-performing machine learning models.
- It generates performance metrics and visualizations, helping users understand and compare different model's accuracy, balanced accuracy, F1 scores, and computation times. Additionally, the page includes functionality for predicting new data based on user-selected attributes and models.

10. Optimization for various parameters

- This Streamlit app provides a user-friendly interface for hyperparameter optimization of a machine learning classification model, specifically using the Decision Tree algorithm. Users can upload their CSV dataset or use the provided example dataset.
- The app allows users to interactively set various hyperparameters such as the number of estimators, max features, and others, and then builds a classification model based on the selected hyperparameters.
- Model performance metrics such as coefficient of determination (R^2) and error (MSE or MAE) are displayed, along with a 3D surface plot showing the relationship between hyperparameters and model performance. Additionally, users can download a CSV file containing detailed grid search results for further analysis.

11. General Visualizations

- **Feature:** Standard charts and graphs to display trends.
- **Purpose:** Provides an overview of the data through common visualizations like bar charts, line graphs, and pie charts, scatter plots, box plots, histogram making it easier to identify patterns and trends.

12. Inferential Visualizations

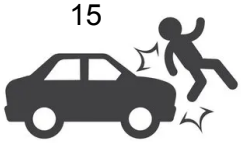
- **Feature:** Advanced visualizations for inferential statistics and deeper analysis, providing a comprehensive inference about the US accident dataset.

8. Hardware Requirements

- ### Development Machine
  - Processor: Intel Core i3 or equivalent
  - RAM: 8 GB or more
  - :Storage: 256 GB SSD or more
  - Operating System: Windows 10, macOS, or Linux
- ### Server (if self-hosting instead of using Streamlit Cloud)
  - Processor: Intel Xeon or equivalent
  - RAM: 16 GB or more
  - Storage: 500 GB SSD or more
  - Operating System: Linux (Ubuntu 20.04 LTS or later recommended)
  - Network: High-speed internet connection with at least 10 Mbps upload/download speed

9. Software Requirements

- ### Programming Language
  - Python: Version 3.8 or higher
- ### Python Libraries
  - **Streamlit:** For building and deploying the dashboard
  - **pandas:** For data manipulation and analysis
  - **Plotly:** For interactive visualizations
  - **seaborn:** For statistical data visualization
  - **numpy:** For numerical operations
  - **scikit-learn:** For machine learning and preprocessing tasks
  - **matplotlib:** For basic plotting
  - **base58:** A library for binary-to-text encoding, commonly used for creating short, human-readable identifiers.
  - **pillow:** A Python Imaging Library (PIL) fork that adds image processing capabilities.



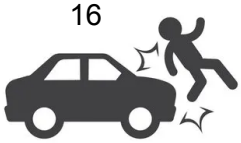
- **lazypredict:** A module that helps in building a lot of basic models without much code and helps understand which models work better without much parameter tuning.
- **xgboost:** An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable.
- **lightgbm:** A highly efficient gradient boosting framework that uses tree-based learning algorithms.
- **pytest:** A testing framework for Python that makes it easy to write simple and scalable test cases.
- **tqdm:** A library providing fast, extensible progress bars for loops and other iterative tasks.
- **ydata-profiling:** A library that generates profile reports from a pandas DataFrame to understand data quickly.
- **streamlit-pandas-profiling:** An integration of Streamlit with pandas-profiling, allowing easy display of profile reports in a Streamlit app.
- **Jinja2:** A templating engine for Python, used for rendering templates to generate HTML or other markup languages.

- **Development Environment**
  - IDE/Code Editor: Visual Studio Code, Jupyter Notebook
- **Deployment Platform**
  - Streamlit Cloud: For hosting the dashboard online

Conclusion

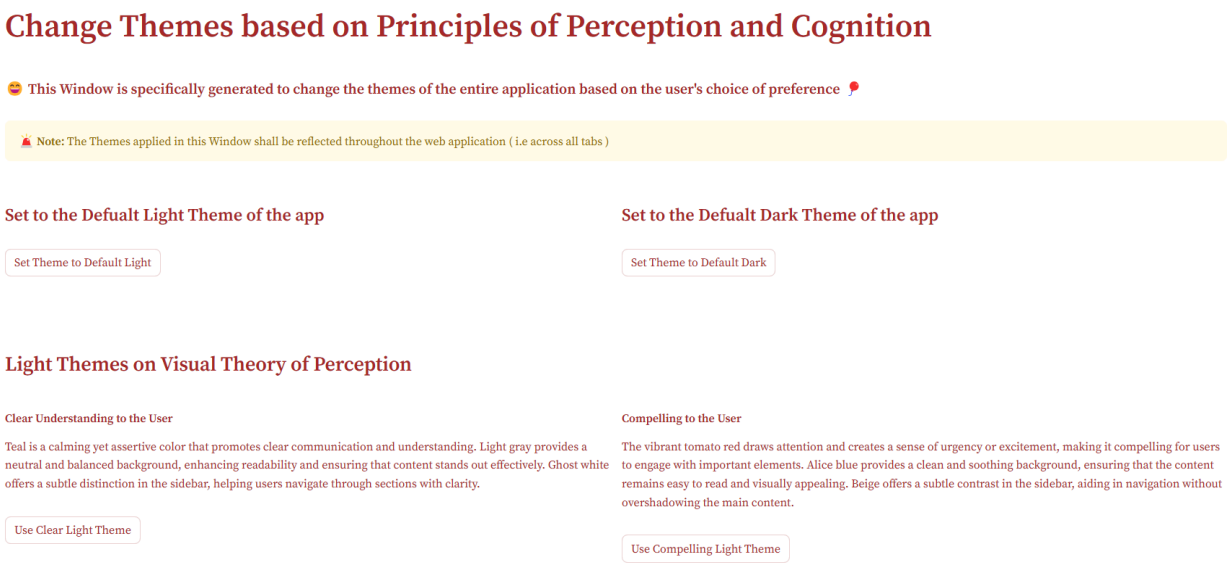
This project has successfully developed an interactive and user-friendly web page for exploring and visualizing a comprehensive US accident dataset. Through the deployment on Streamlit Cloud, we have ensured that our dashboard is accessible from anywhere, without the need for complex installations, making it a versatile tool for a wide range of users, including policymakers, researchers, traffic safety analysts, and the general public.

This project has effectively filled a market void by providing a comprehensive, user-friendly, and cost-effective solution for analyzing US accident data. By transforming complex data into actionable intelligence through thoughtful design and robust analytical capabilities, our dashboard empowers users to make data-driven decisions, ultimately contributing to improved road safety and more informed policy decisions.



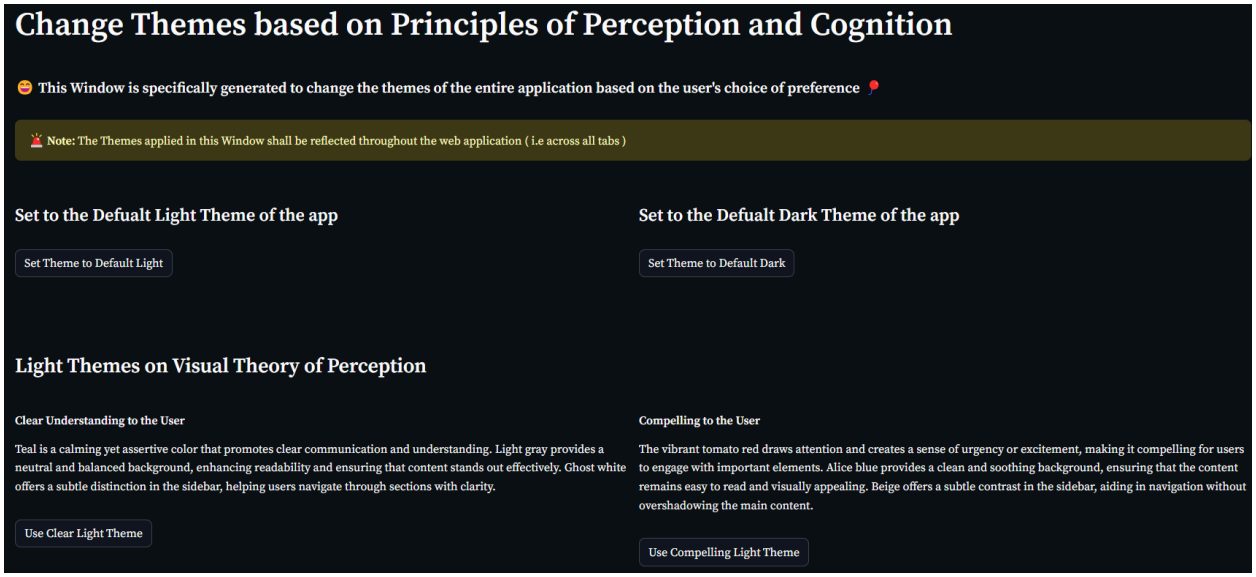
WEB PAGE LAYOUT AND DESIGN:
User Authentication:

Change Theme:

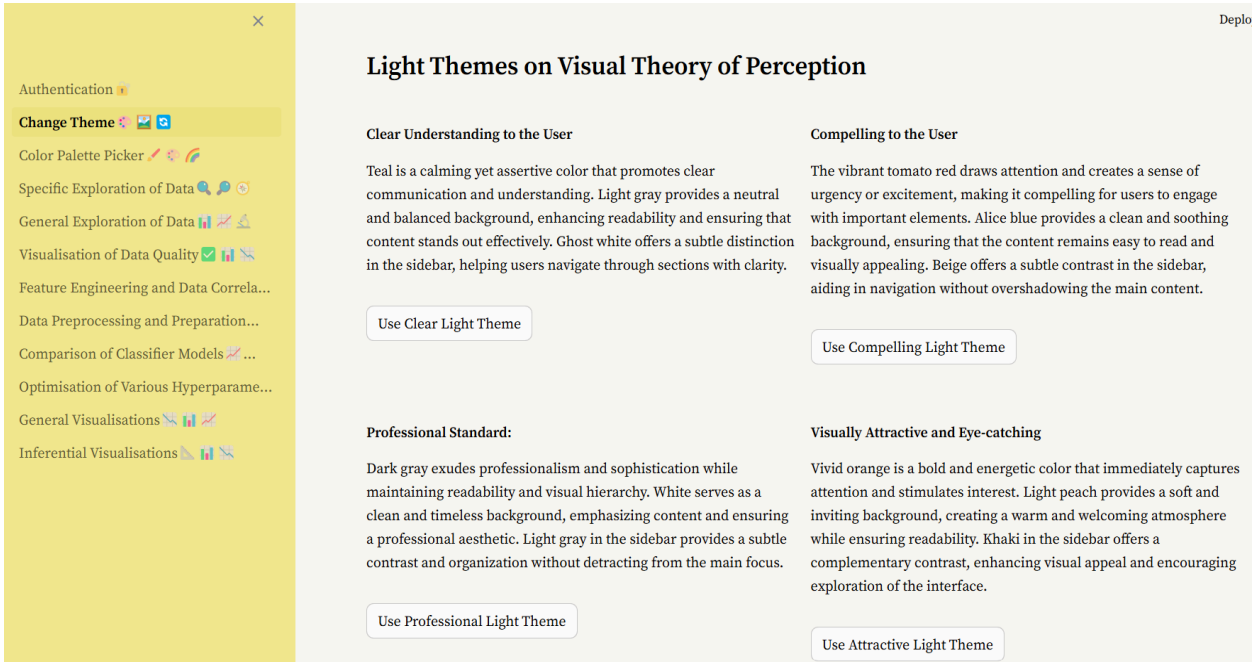


Different available themes:

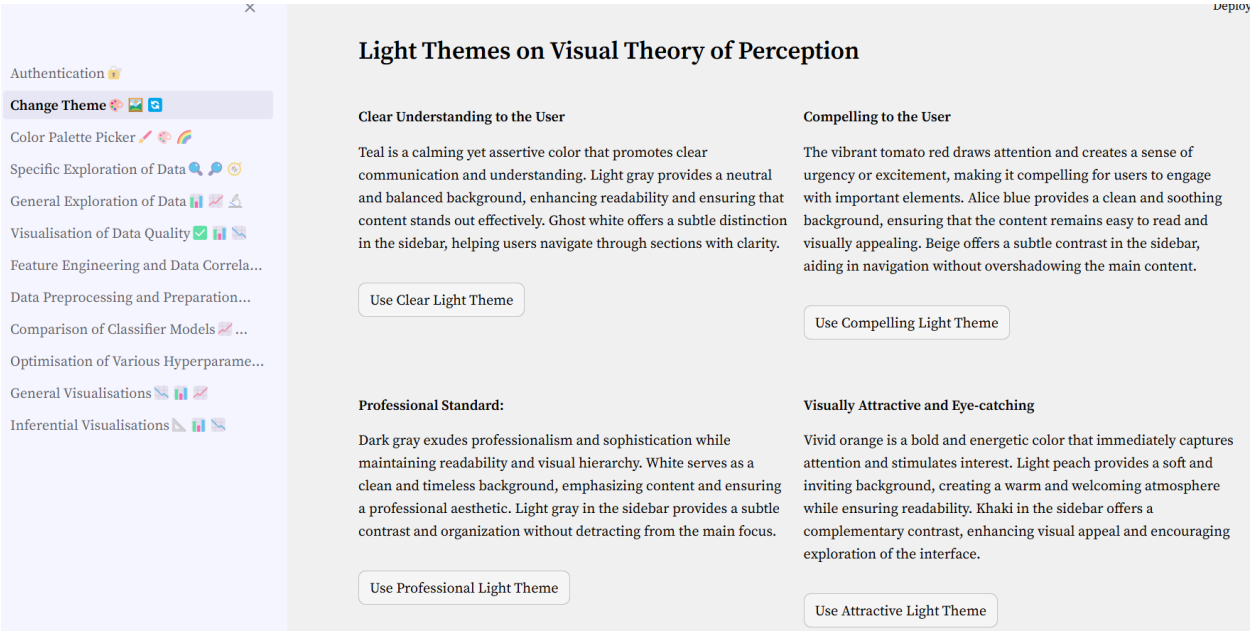
Dark theme:



Visually Attractive and Eye-catching:



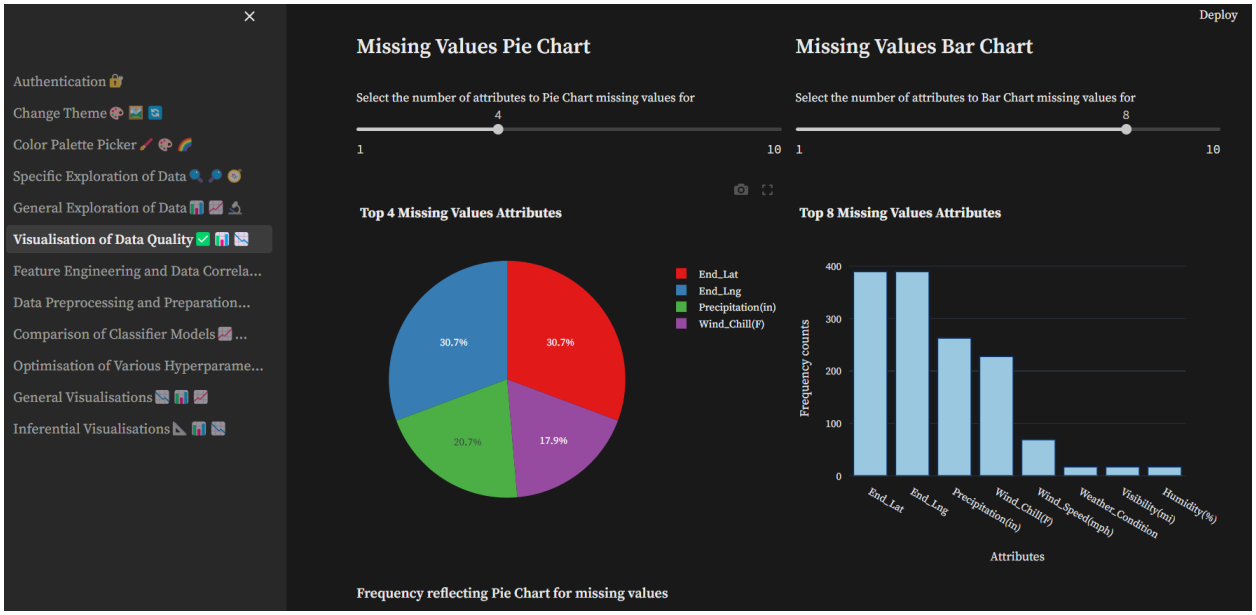
Clear Understanding to the User:



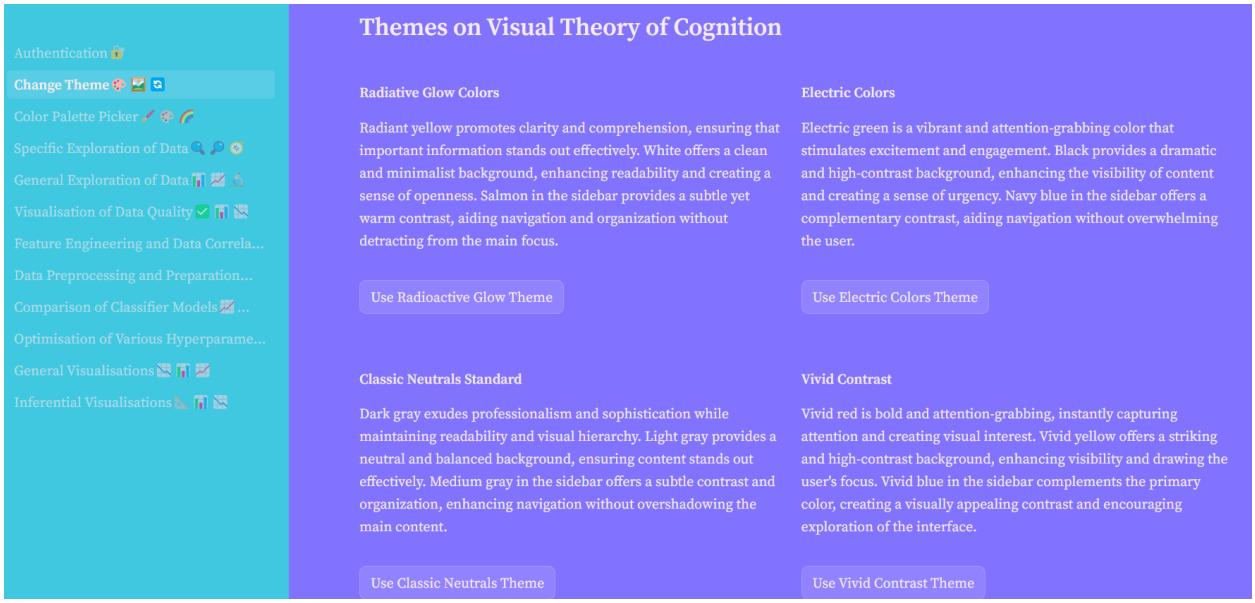
Professional Standard:

People who are color blind to use different themes:

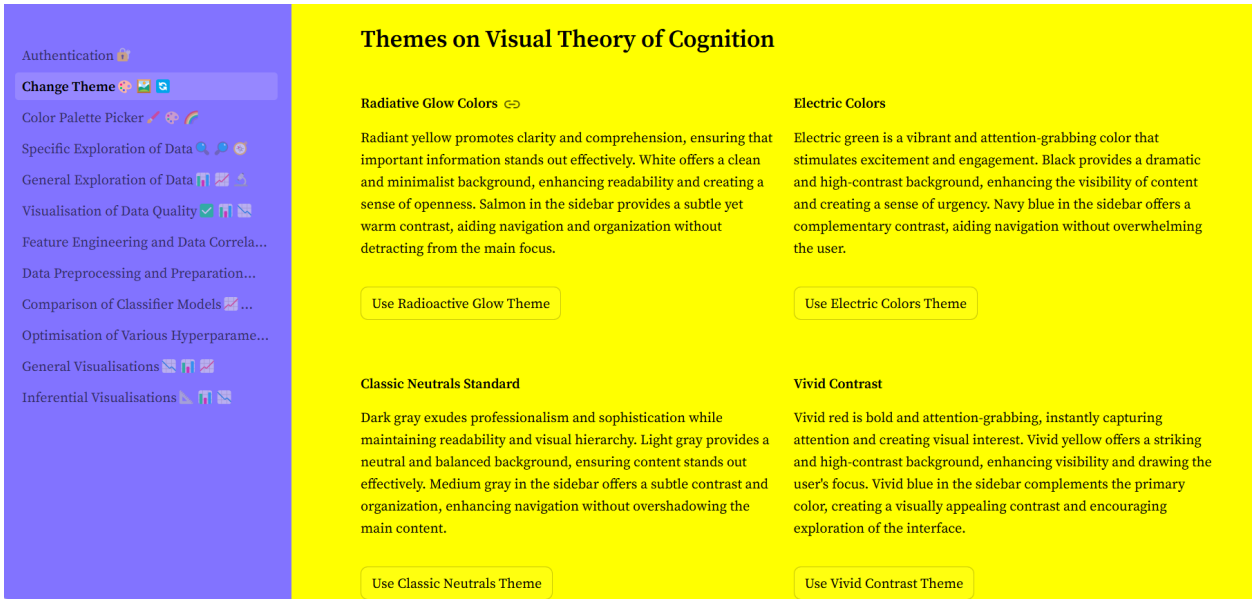
Visually Attractive and Eye-catching:

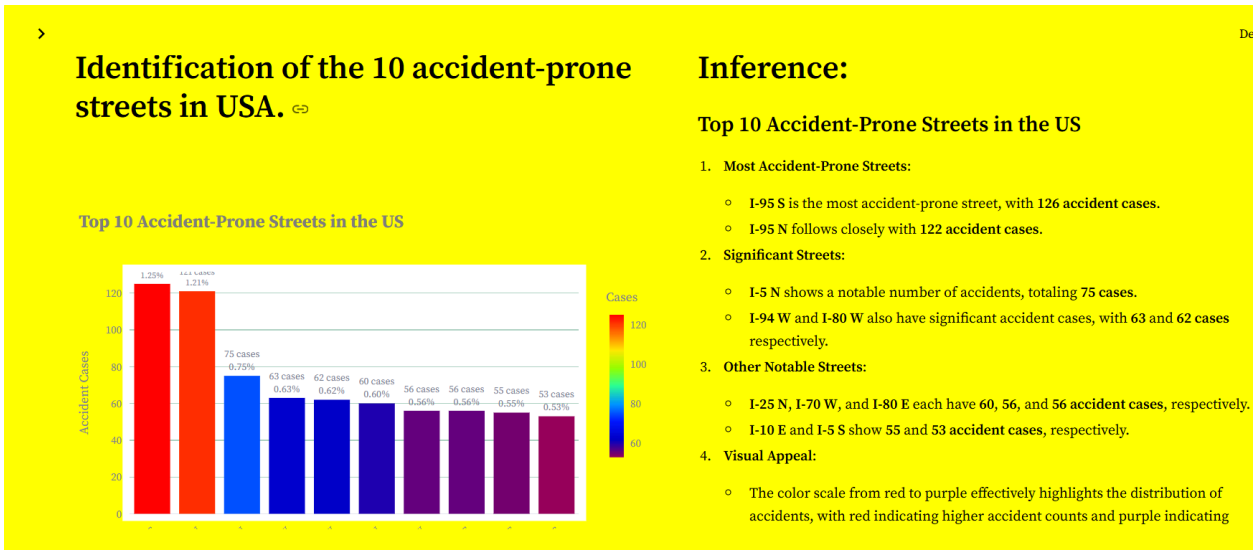


Electric Colors:



Vivid Contrast:

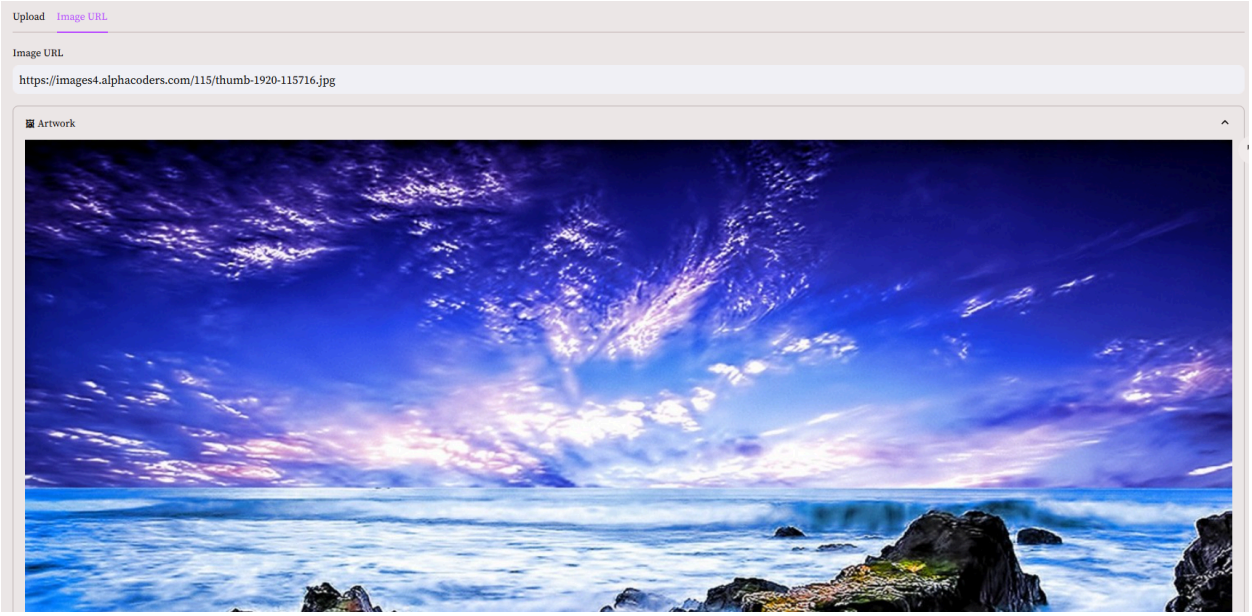




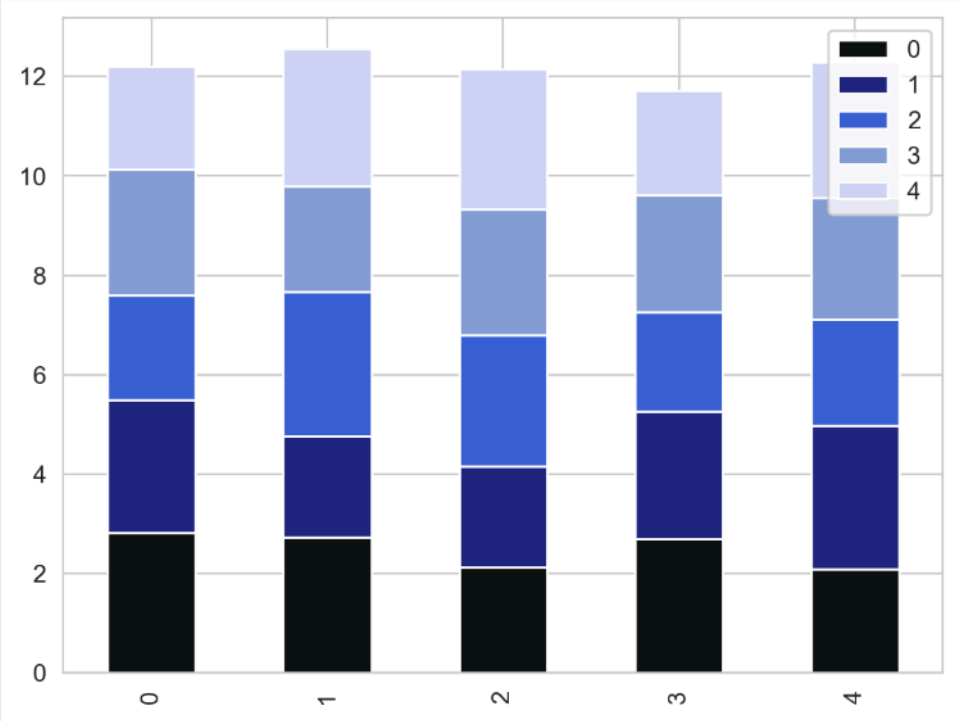
Color palette picker:

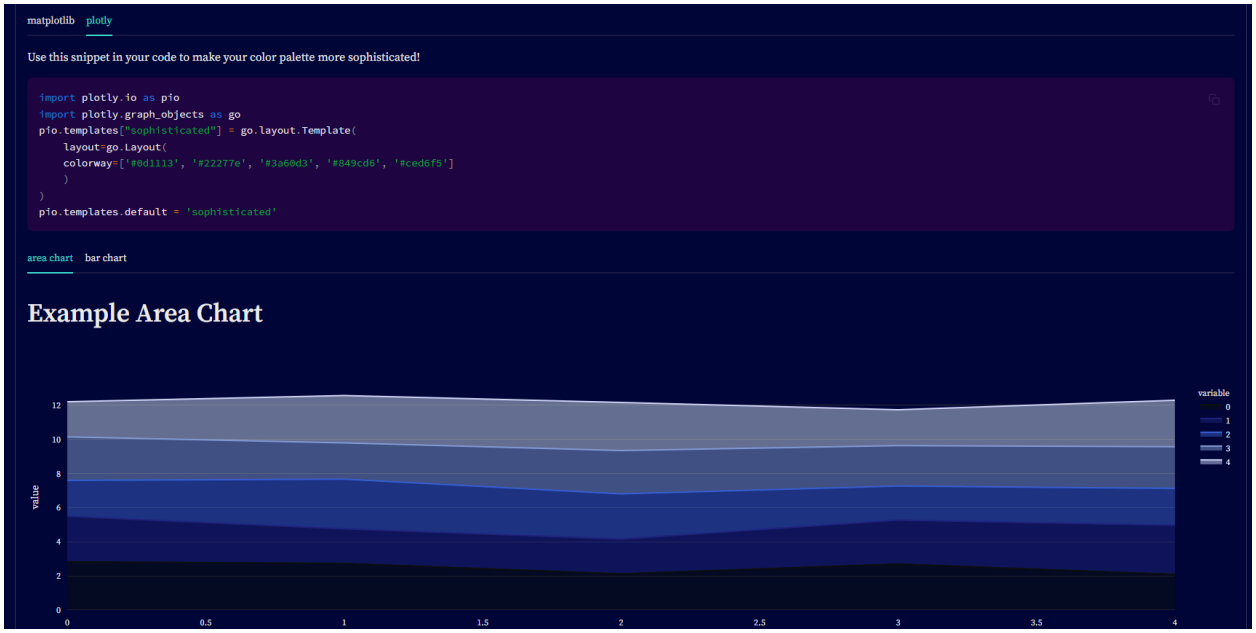
The different colour is picked from a image and these colours are used as colour palette for the plots.

Picking a color palette using image url:

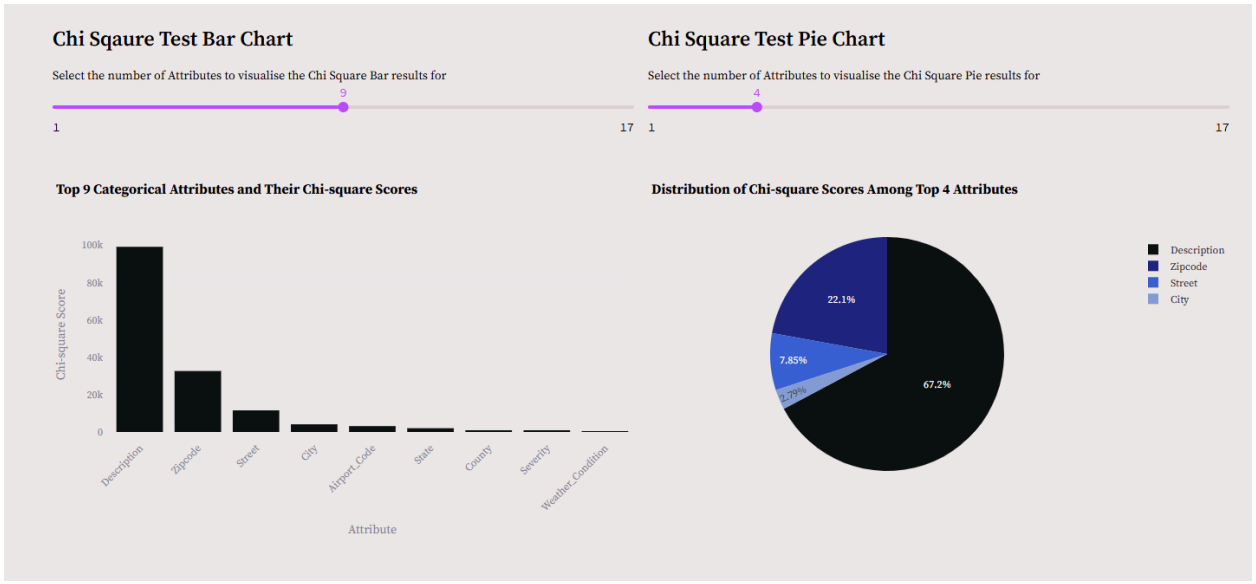


Example Bar plot:

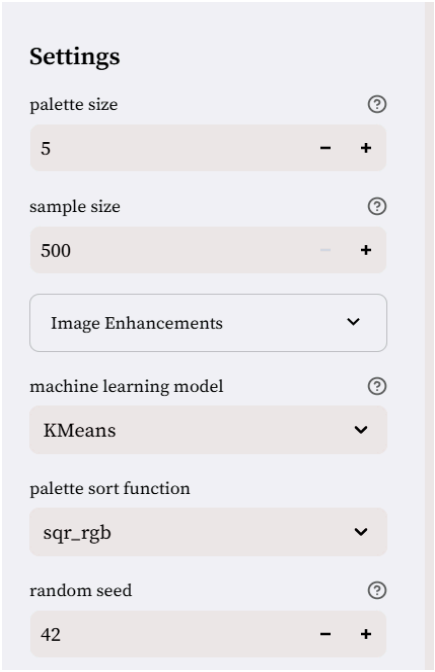


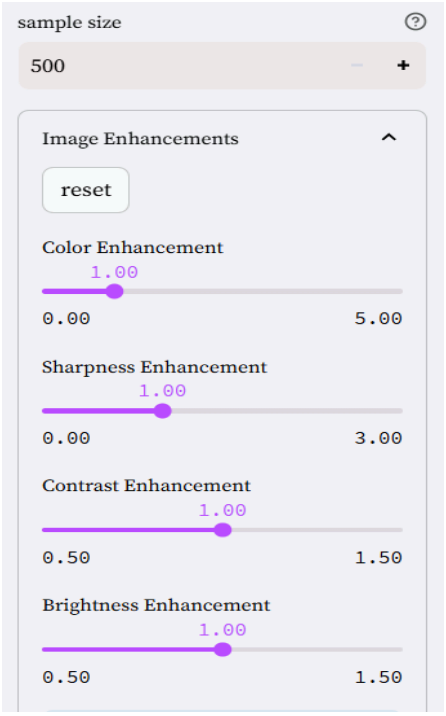


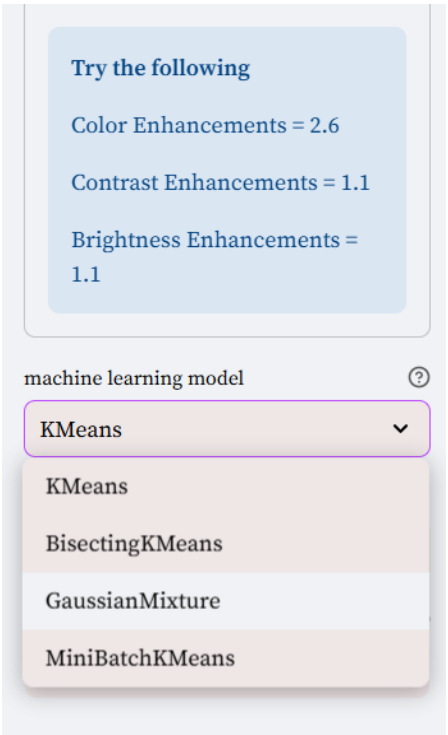
Picked color palette applied in plots:

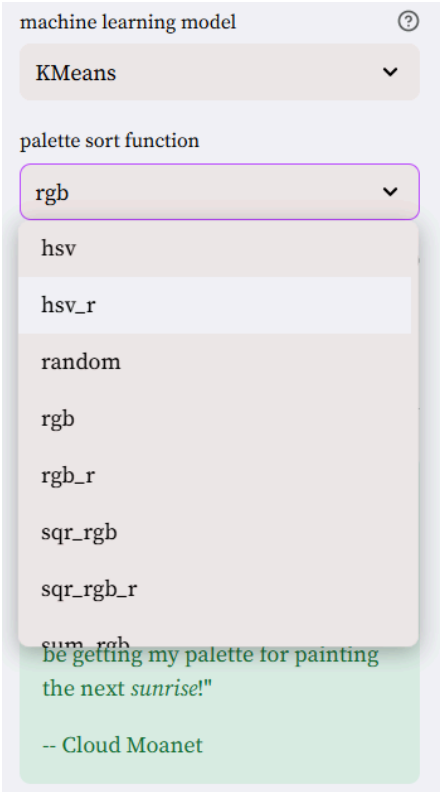


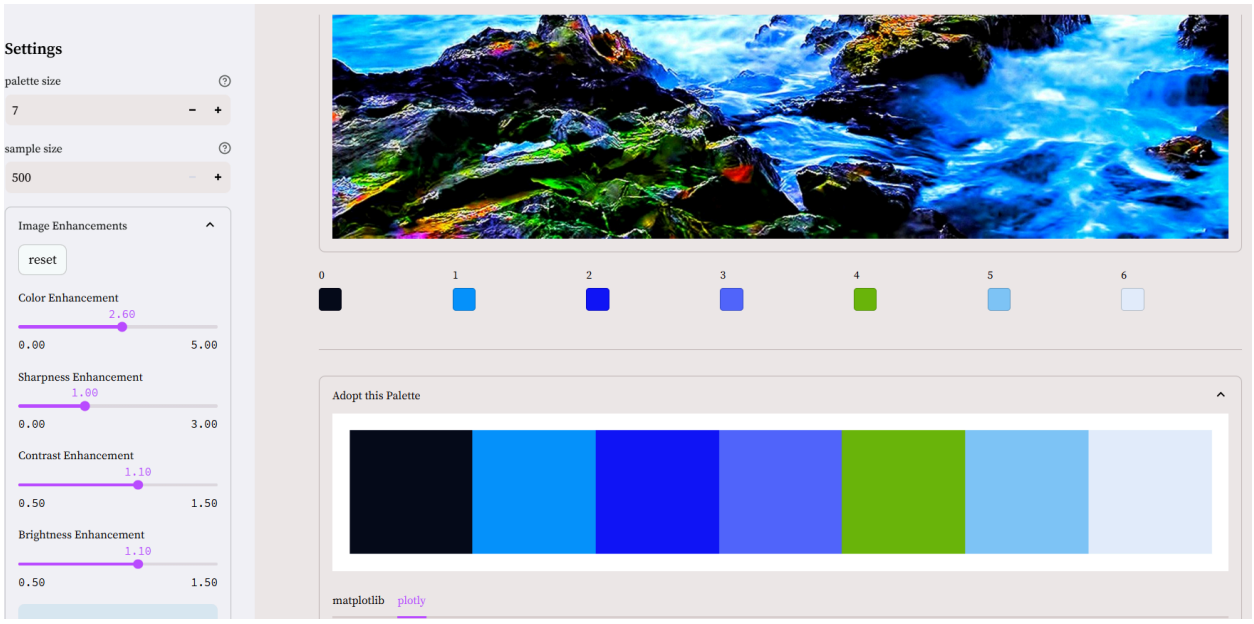
Settings to pick a color palette from a image:





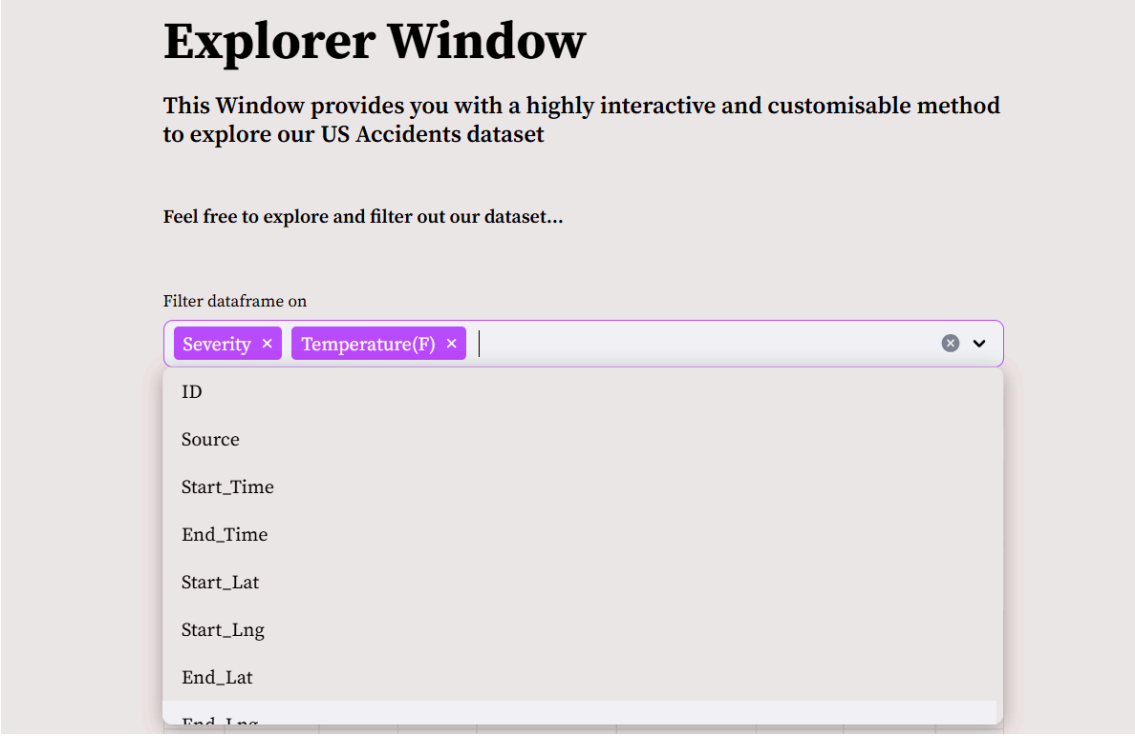


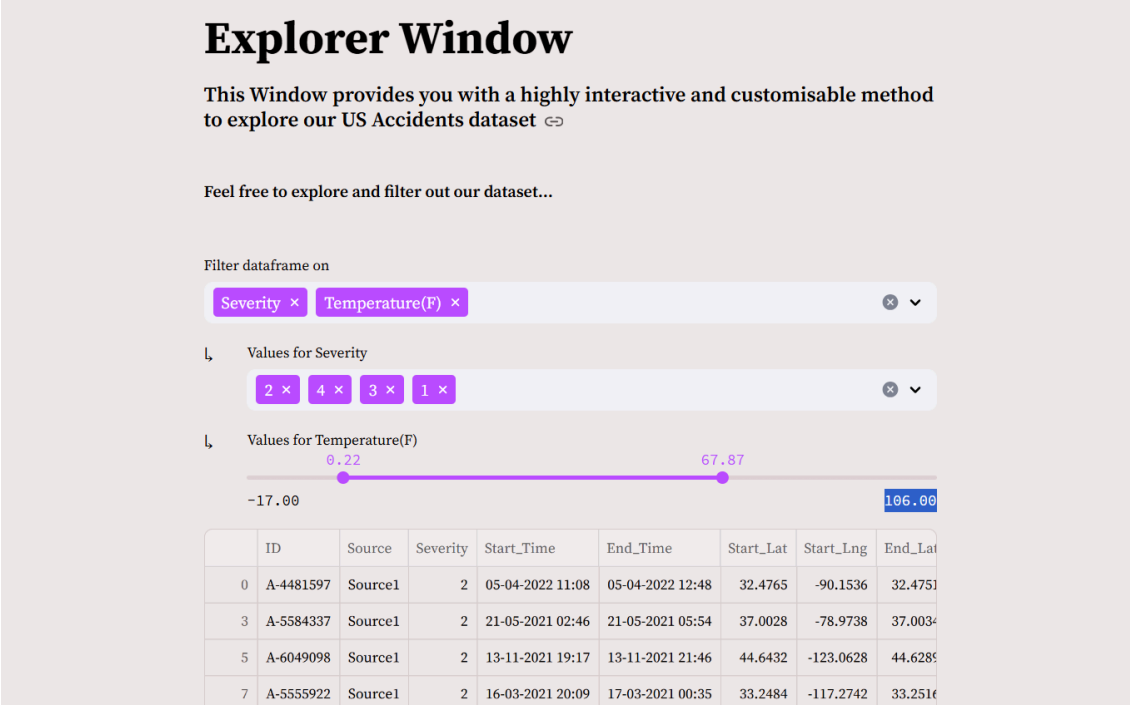




Example Bar plot for picked colour palette:

Data filtering :





Filter dataframe on

Severity × Temperature(F) × Start_Lat ×

Values for Severity

2 × 4 × 3 ×

Values for Temperature(F)

-0.76 85.08

-17.00 99.00

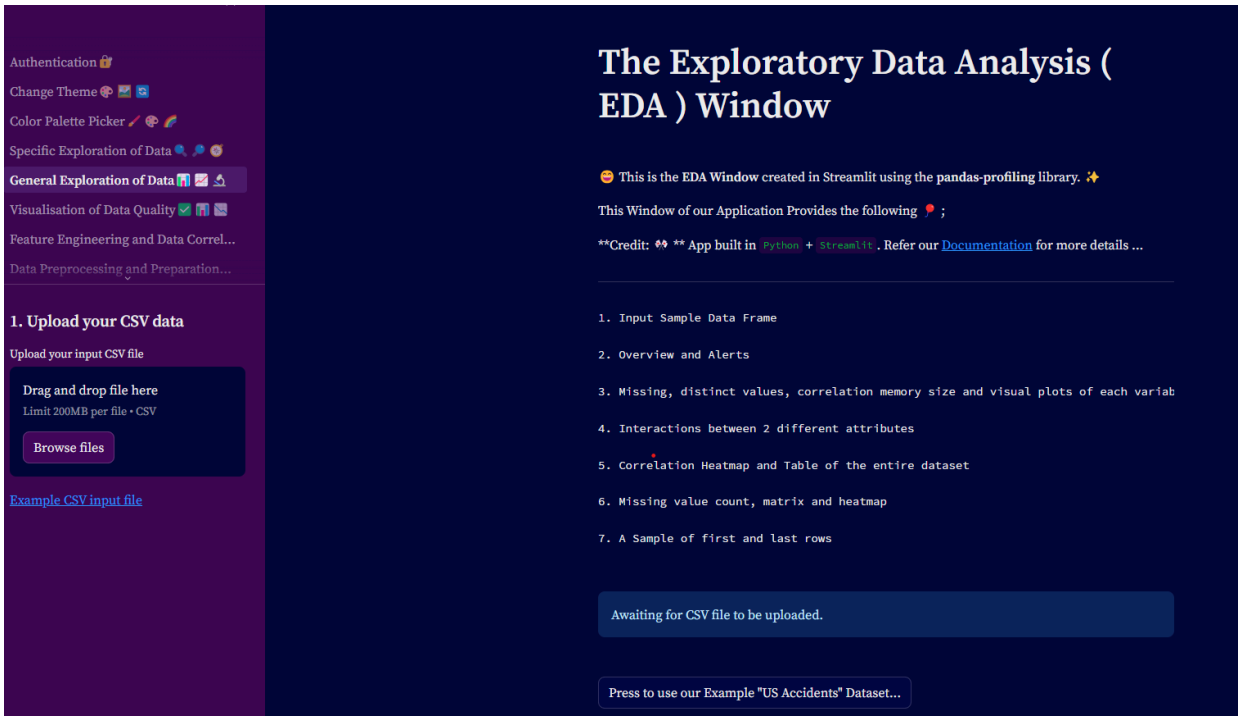
Values for Start_Lat

30.64 41.75

25.53 47.75

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat
0	A-4481597	Source1	2	05-04-2022 11:08	05-04-2022 12:48	32.4765	-90.1536	32.4751
2	A-5124735	Source1	2	21-08-2022 19:52	21-08-2022 21:07	32.7147	-96.7725	32.7138
3	A-5584337	Source1	2	21-05-2021 02:46	21-05-2021 05:54	37.0028	-78.9738	37.0034
4	A-4458809	Source1	2	22-09-2022 07:18	22-09-2022 08:50	32.6452	-96.7896	32.6579
7	A-5555922	Source1	2	16-03-2021 20:09	17-03-2021 00:35	33.2484	-117.2742	33.2516
8	A-6101994	Source1	2	18:00.0	10:00.0	40.6966	-112.0313	40.6966

General Exploration of data:



Input DataFrame

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_La
0	A-7307599	Source1	3	10-12-2019 10:36	10-12-2019 11:05	39.1153	-108.5447	39.115
1	A-1992556	Source2	2	29-07-2019 07:34	29-07-2019 09:05	35.1157	-81.0697	None
2	A-5306504	Source1	4	45:00.0	15:00.0	30.2384	-97.7385	30.226
3	A-2683723	Source2	2	03-07-2018 13:15	03-07-2018 13:45	40.1538	-75.4194	None
4	A-6573560	Source1	2	47:00.0	05:25.0	38.115	-77.5178	38.127
5	A-5006548	Source1	2	30:00.0	33:30.0	33.456	-111.974	33.459
6	A-7015824	Source1	3	08-05-2020 12:44	08-05-2020 12:59	42.38	-71.1445	42.38
7	A-2840915	Source2	2	22-05-2018 11:49	22-05-2018 12:19	36.7408	-119.8146	None
8	A-5750958	Source1	2	06-10-2021 15:16	06-10-2021 16:34	33.9291	-117.1218	33.935
9	A-1273882	Source2	3	10-11-2020 12:07	10-11-2020 14:29	40.5589	-111.8981	None

Overview:

Dataset Profiling Report

Overview

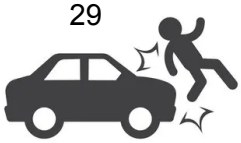
Overview | Alerts 32 | Reproduction

Dataset statistics

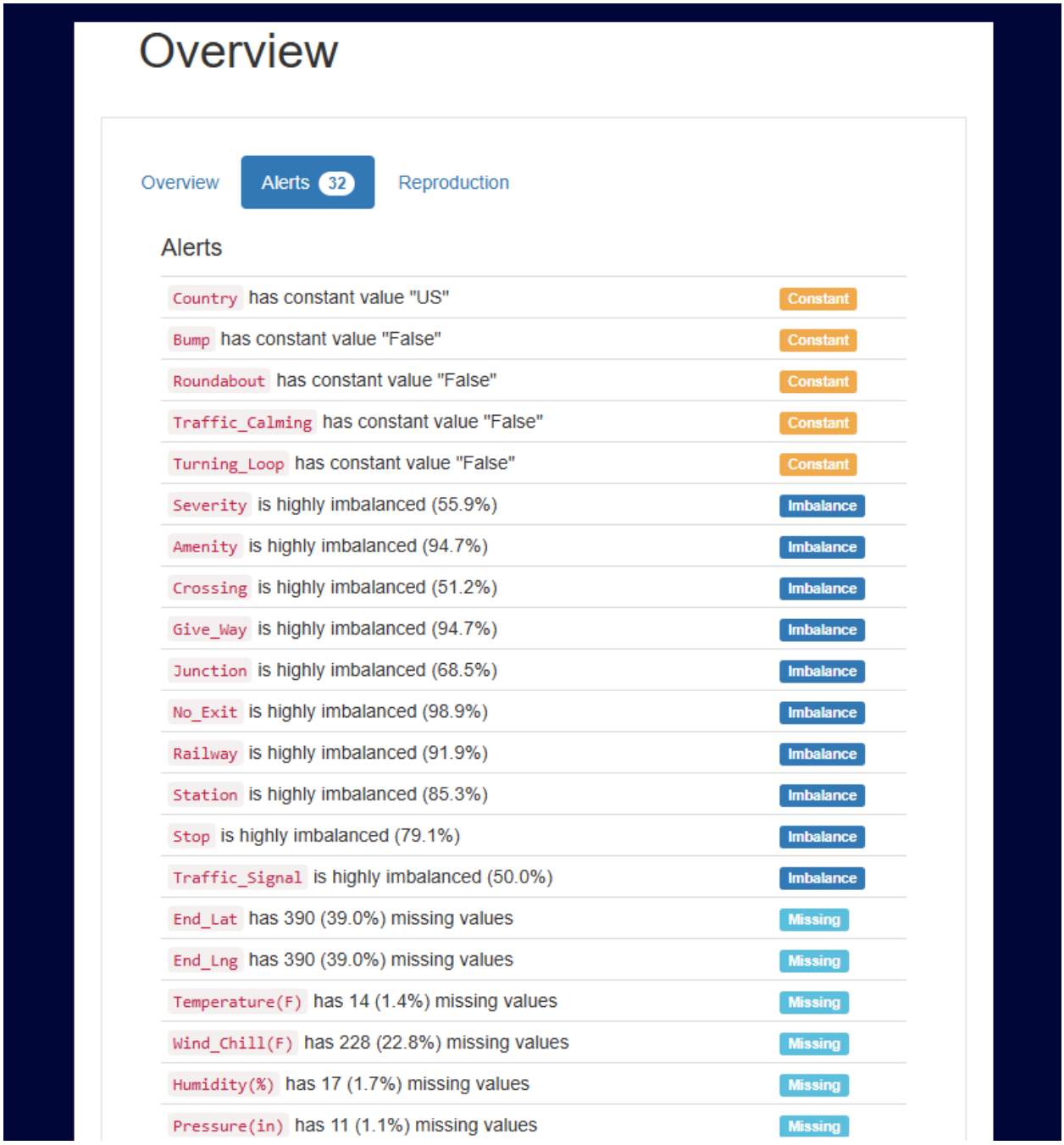
Number of variables	46
Number of observations	1000
Missing cells	1462
Missing cells (%)	3.2%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.2 MiB
Average record size in memory	1.3 KiB

Variable types

Text	9
Categorical	11
Numeric	12
DateTime	1
Boolean	13



Problems in the attribute:



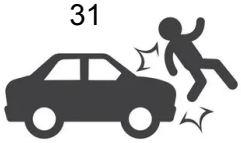
`Visibility(mi)` has 17 (1.7%) missing values	Missing
`Wind_Direction` has 16 (1.6%) missing values	Missing
`Wind_Speed(mph)` has 69 (6.9%) missing values	Missing
`Precipitation(in)` has 263 (26.3%) missing values	Missing
`Weather_Condition` has 17 (1.7%) missing values	Missing
`ID` has unique values	Unique
`Start_Lat` has unique values	Unique
`Start_Lng` has unique values	Unique
`Distance(mi)` has 386 (38.6%) zeros	Zeros
`Wind_Speed(mph)` has 139 (13.9%) zeros	Zeros
`Precipitation(in)` has 688 (68.8%) zeros	Zeros

Overview

Overview | Alerts 32 | Reproduction

Reproduction

Analysis started	2024-05-27 13:39:52.600393
Analysis finished	2024-05-27 13:40:12.974711
Duration	20.37 seconds
Software version	ydata-profiling v4.8.3
Download configuration	config.json



Variable description:

Variables

Description

Description

Text

Distinct	993
Distinct (%)	99.3%
Missing	0
Missing (%)	0.0%
Memory size	114.8 KiB

Information about the attribute:

Overview | Words | Characters

Length

Max length	231
Median length	124
Mean length	68.409
Min length	8

Characters and Unicode

Total characters	68409	
Distinct characters	75	
Distinct categories	8	?
Distinct scripts	2	?
Distinct blocks	1	?

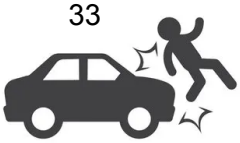
The Unicode Standard assigns character properties to each code point, which can be used to analyse textual variables.

Unique

Unique	987	?
Unique (%)	98.7%	

Sample

1st row	At Horizon Drive/Exit 31 - Accident.
2nd row	Accident on SC-49 both ways at Forest Oaks Dr.
3rd row	Incident on I-35 SB near US-290 Road closed. Take alternate route.
4th row	Accident on Eagleville Rd at Visitation Rd.
5th row	Incident on I-95 NB near MM 118 Expect delays.



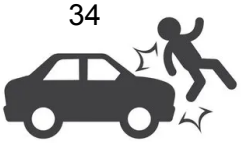
Overview | Words | Characters

Value	Count	Frequency (%)
on	779	6.4%
accident	717	5.9%
to	612	5.0%
at	437	3.6%
due	371	3.1%
rd	350	2.9%
	284	2.3%
near	244	2.0%
from	228	1.9%
blocked	221	1.8%
Other values (2313)	7916	65.1%

Characters | Categories | Scripts | Blocks

Most occurring characters

Value	Count	Frequency (%)
	11159	16.3%
t	4378	6.4%
e	4072	6.0%
n	3852	5.6%
o	3657	5.3%
a	2960	4.3%
d	2928	4.3%
i	2724	4.0%
c	2549	3.7%
r	2108	3.1%
Other values (65)	28022	41.0%



Characters | Categories | Scripts | Blocks

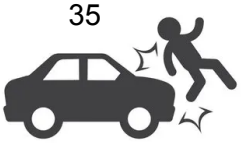
Most occurring categories

Value	Count	Frequency (%)
Lowercase Letter	38531	56.3%
Space Separator	11159	16.3%
Uppercase Letter	10541	15.4%
Decimal Number	4432	6.5%
Dash Punctuation	1546	2.3%
Other Punctuation	1538	2.2%
Open Punctuation	331	0.5%
Close Punctuation	331	0.5%

Most frequent character per category

Lowercase Letter

Value	Count	Frequency (%)
t	4378	11.4%
e	4072	10.6%
n	3852	10.0%
o	3657	9.5%
a	2960	7.7%
d	2928	7.6%
i	2724	7.1%
c	2549	6.6%
r	2108	5.5%
l	1425	3.7%
Other values (16)	7878	20.4%

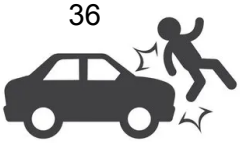


Uppercase Letter

Value	Count	Frequency (%)
A	1262	12.0%
S	1153	10.9%
I	959	9.1%
E	936	8.9%
R	903	8.6%
C	523	5.0%
N	512	4.9%
D	488	4.6%
B	484	4.6%
L	458	4.3%
Other values (16)	2863	27.2%

Decimal Number

Value	Count	Frequency (%)
1	779	17.6%
5	563	12.7%
2	518	11.7%
0	496	11.2%
4	411	9.3%
3	360	8.1%
9	359	8.1%
8	321	7.2%
7	320	7.2%
6	305	6.9%



Other Punctuation

Value	Count	Frequency (%)
.	1100	71.5%
/	389	25.3%
:	19	1.2%
#	17	1.1%
;	6	0.4%
'	6	0.4%
&	1	0.1%

Open Punctuation

Value	Count	Frequency (%)
(	314	94.9%
[	17	5.1%

Close Punctuation

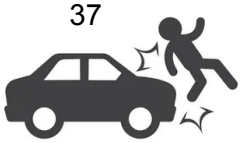
Value	Count	Frequency (%)
)	314	94.9%
]	17	5.1%

Space Separator

Value	Count	Frequency (%)
	11159	100.0%

Dash Punctuation

Value	Count	Frequency (%)
-	1546	100.0%



Characters | Categories | Scripts | Blocks

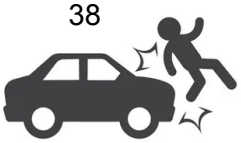
Most occurring scripts

Value	Count	Frequency (%)
Latin	49072	71.7%
Common	19337	28.3%

Most frequent character per script

Latin

Value	Count	Frequency (%)
t	4378	8.9%
e	4072	8.3%
n	3852	7.8%
o	3657	7.5%
a	2960	6.0%
d	2928	6.0%
i	2724	5.6%
c	2549	5.2%
r	2108	4.3%
l	1425	2.9%
Other values (42)	18419	37.5%



Common

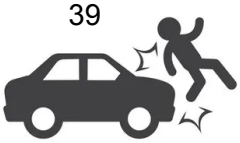
Value	Count	Frequency (%)
	11159	57.7%
-	1546	8.0%
.	1100	5.7%
1	779	4.0%
5	563	2.9%
2	518	2.7%
0	496	2.6%
4	411	2.1%
/	389	2.0%
3	360	1.9%
Other values (13)	2016	10.4%

Overview Words Characters

Characters Categories Scripts Blocks

Most occurring blocks

Value	Count	Frequency (%)
ASCII	68409	100.0%

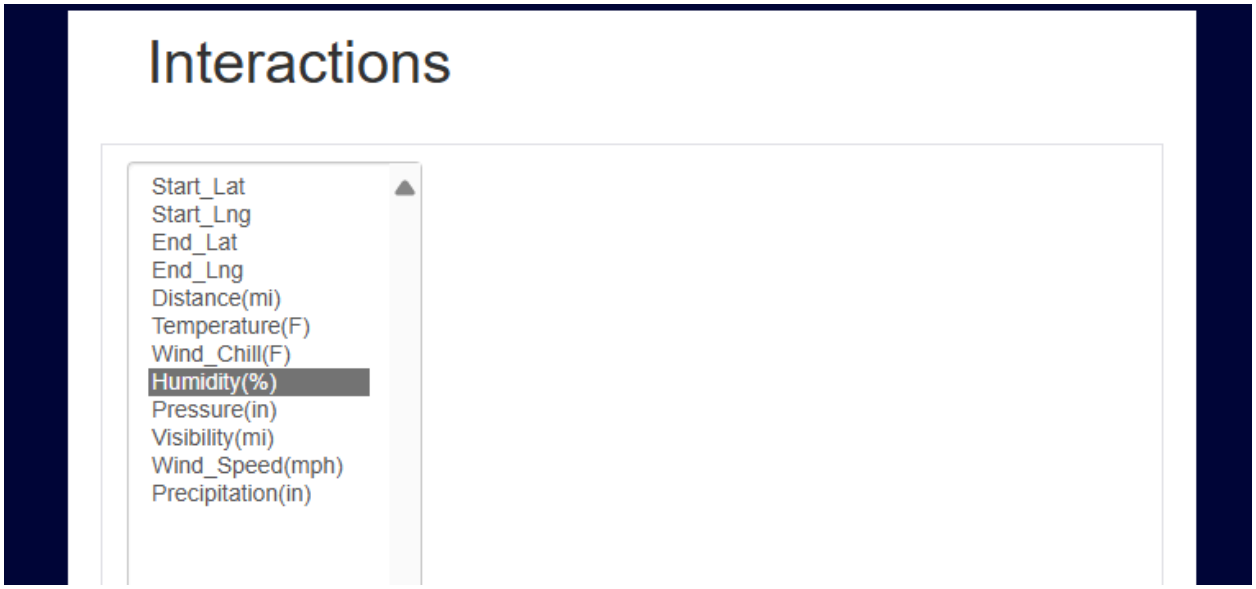


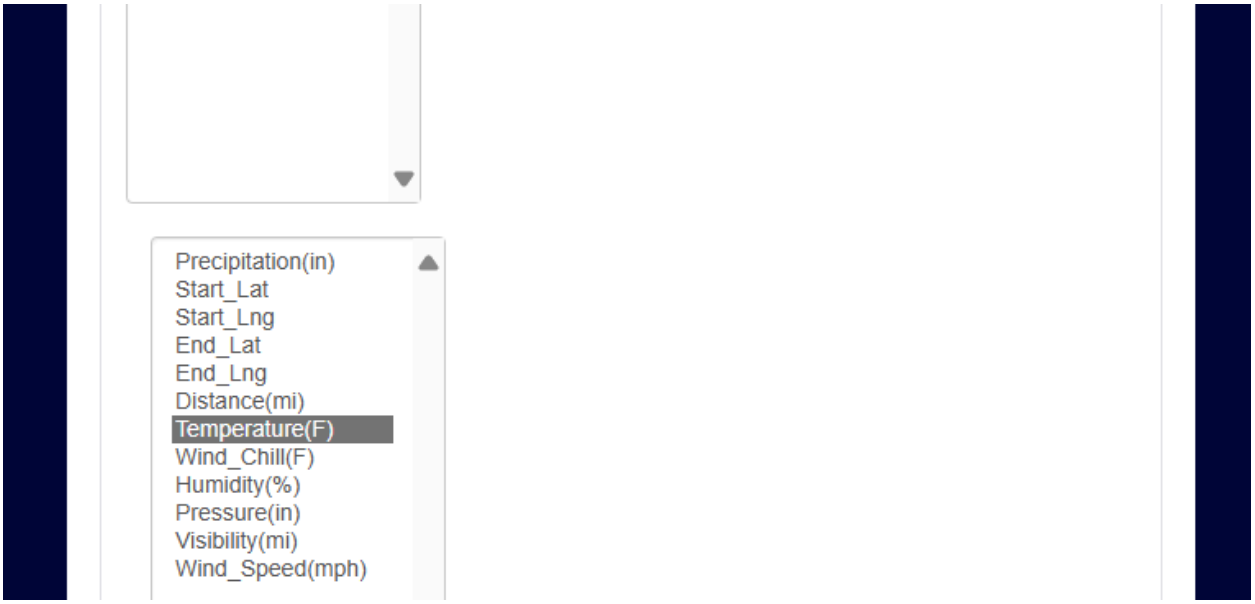
Most frequent character per block

ASCII

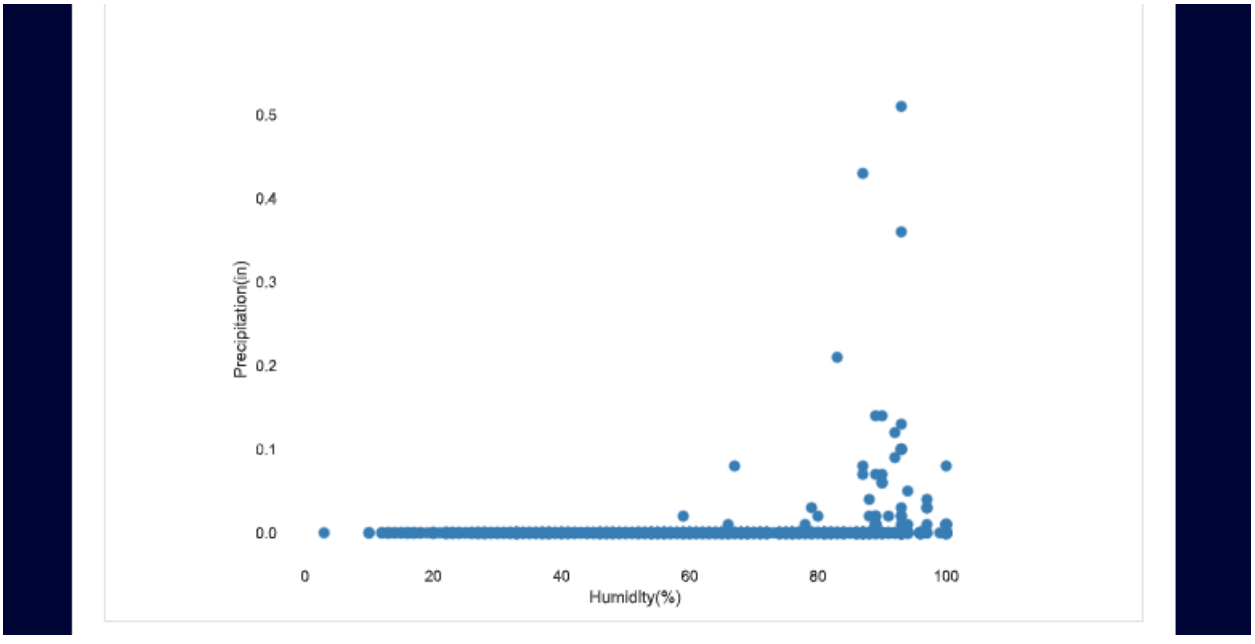
Value	Count	Frequency (%)
	11159	16.3%
t	4378	6.4%
e	4072	6.0%
n	3852	5.6%
o	3657	5.3%
a	2960	4.3%
d	2928	4.3%
i	2724	4.0%
c	2549	3.7%
r	2108	3.1%
Other values (65)	28022	41.0%

Selecting required attributes:

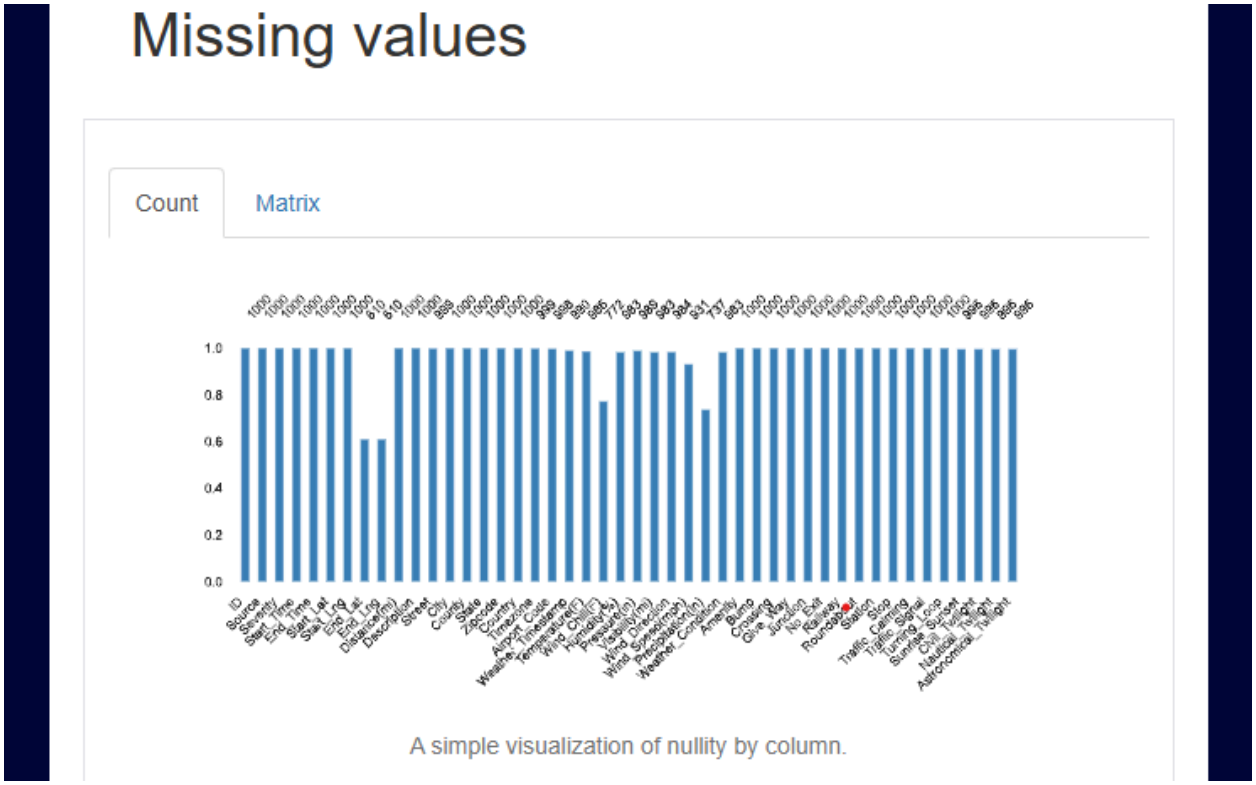




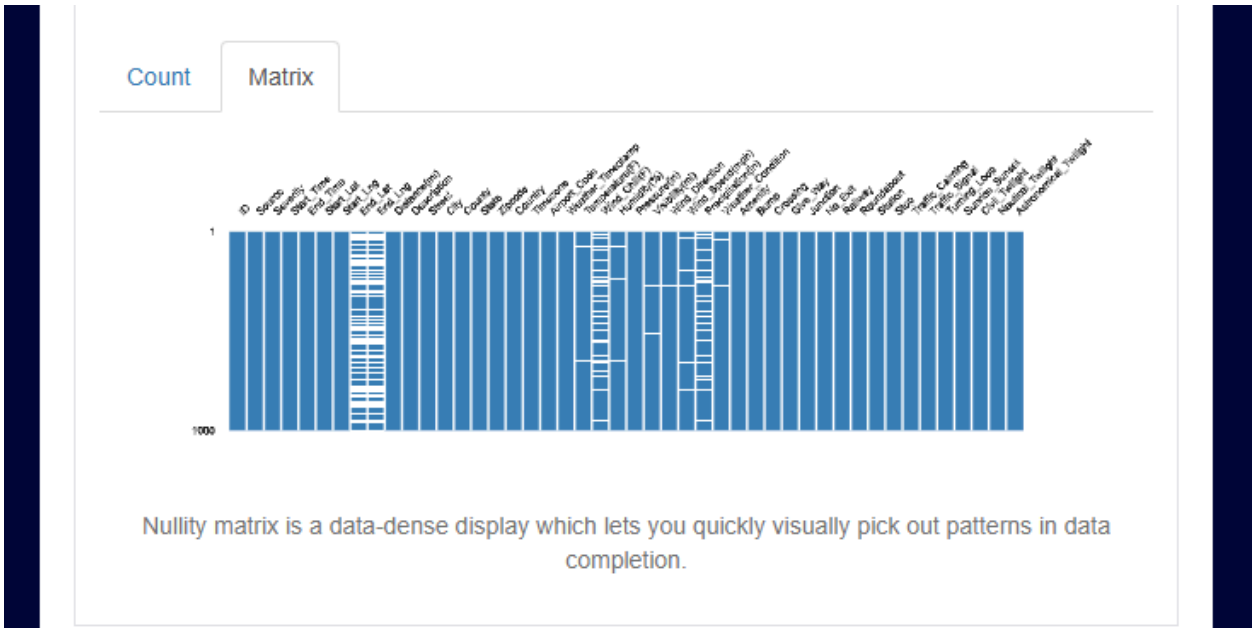
Interactive scatter plot:



Missing values plot:

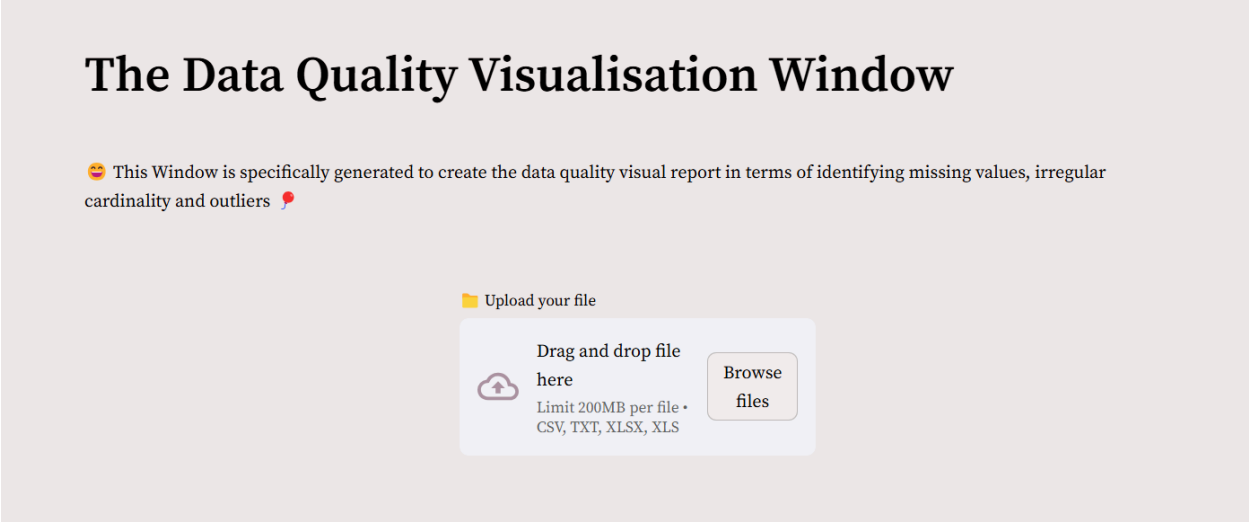


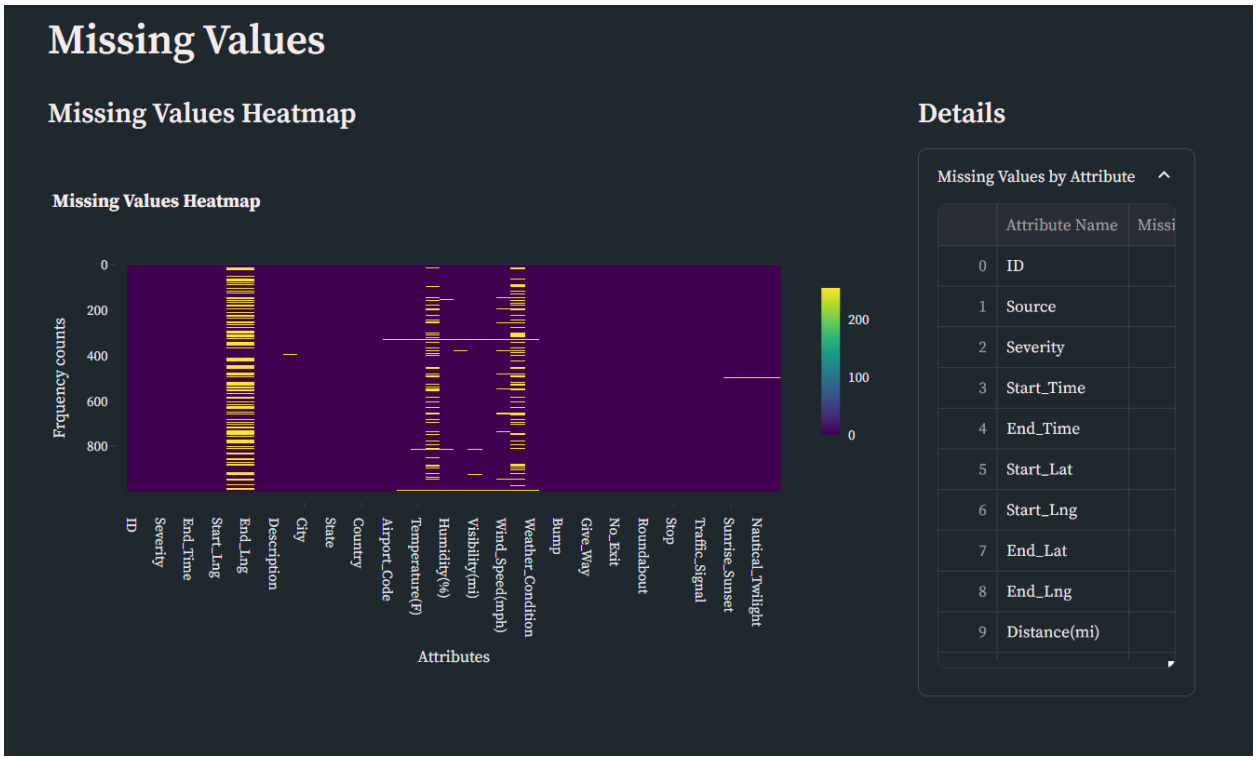
Missing values matrix:



Data quality visualization page:

Drag and drop our own dataset





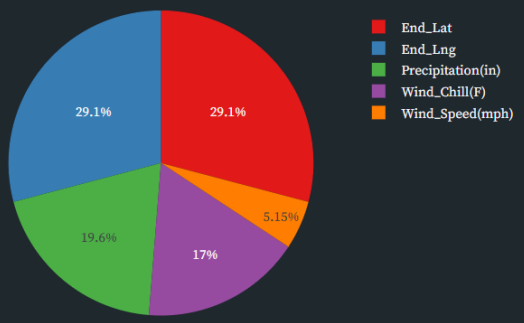
Missing Values Pie Chart

Select the number of attributes to Pie Chart missing values for

5

1 10

Top 5 Missing Values Attributes



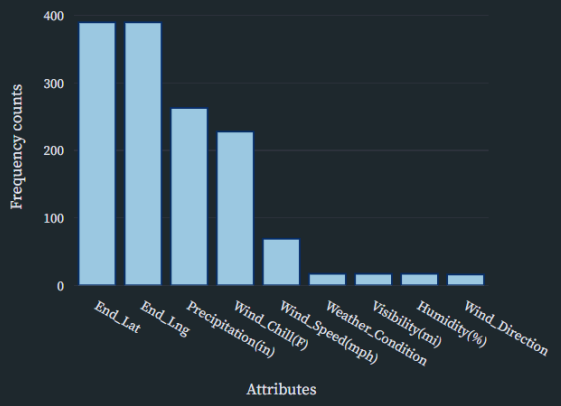
Missing Values Bar Chart

Select the number of attributes to Bar Chart missing values for

9

1 10

Top 9 Missing Values Attributes



Frequency reflecting Pie Chart for missing values

Irregular Cardinality

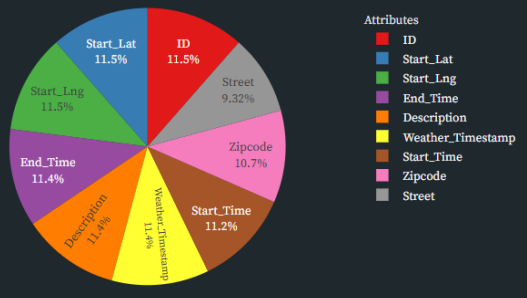
Irregular Cardinality Pie Chart

Select the number of attributes for Pie Chart

9

1 46

Top 9 Irregular Cardinality Attributes



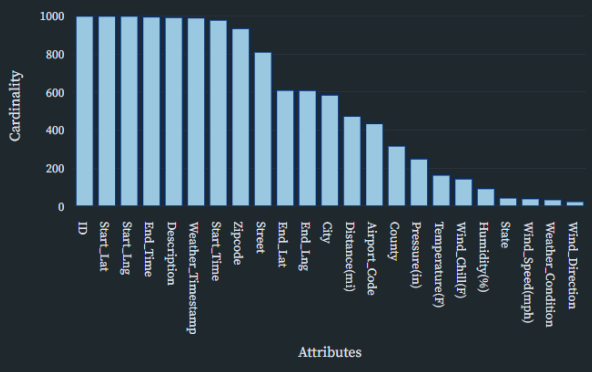
Irregular Cardinality Bar Chart

Select the number of attributes for Bar Chart

23

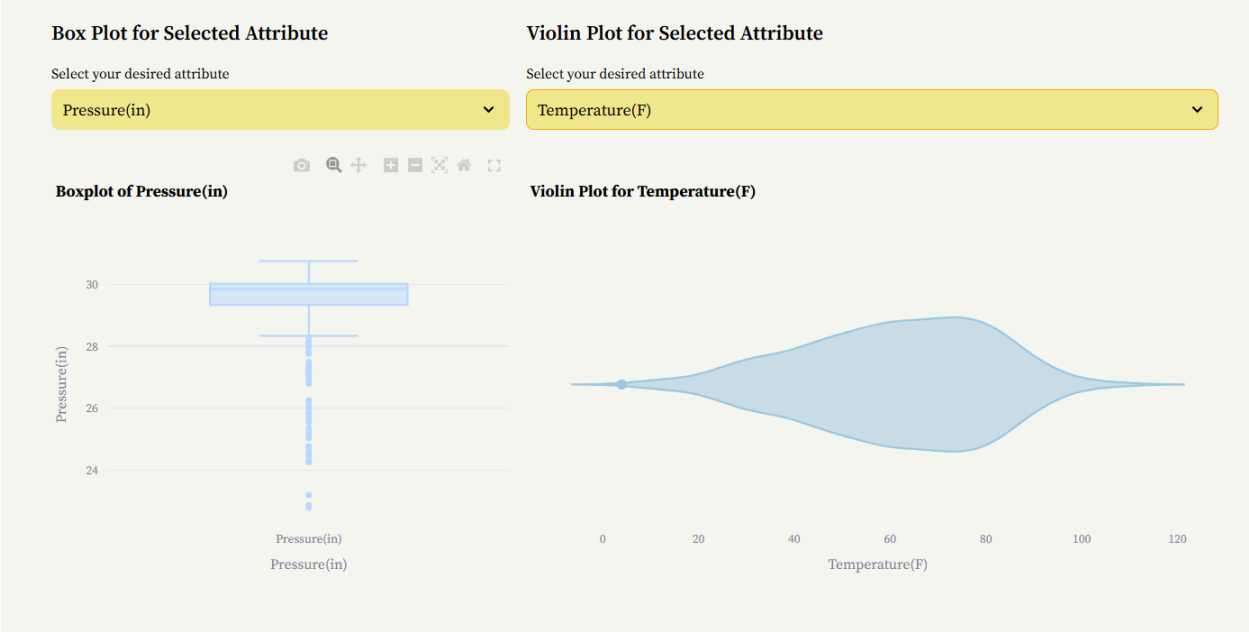
1 46

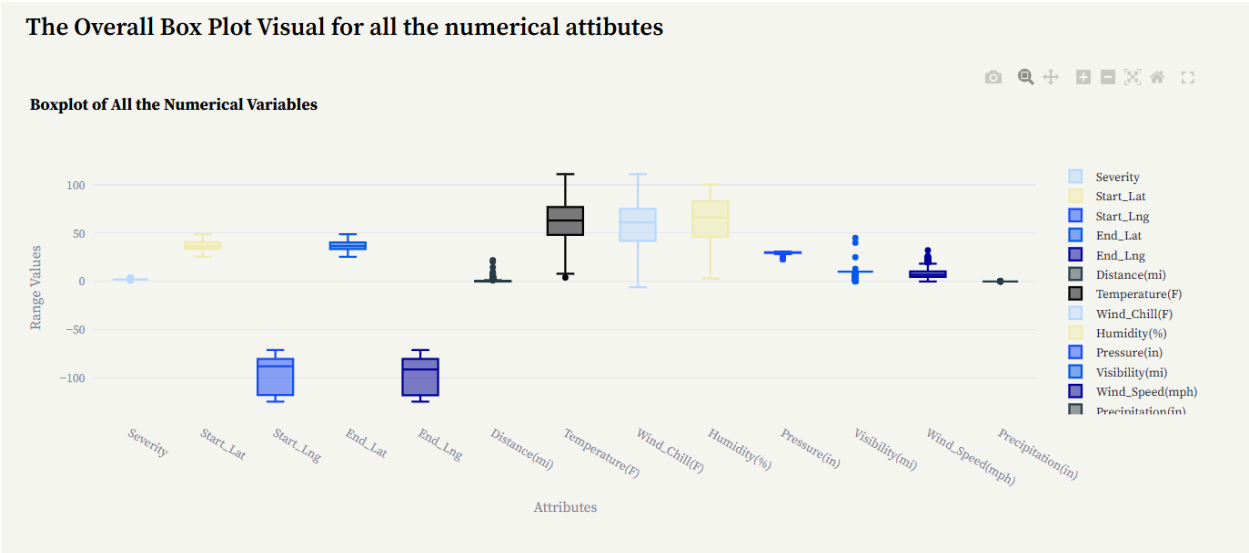
Top 23 Irregular Cardinality Attributes



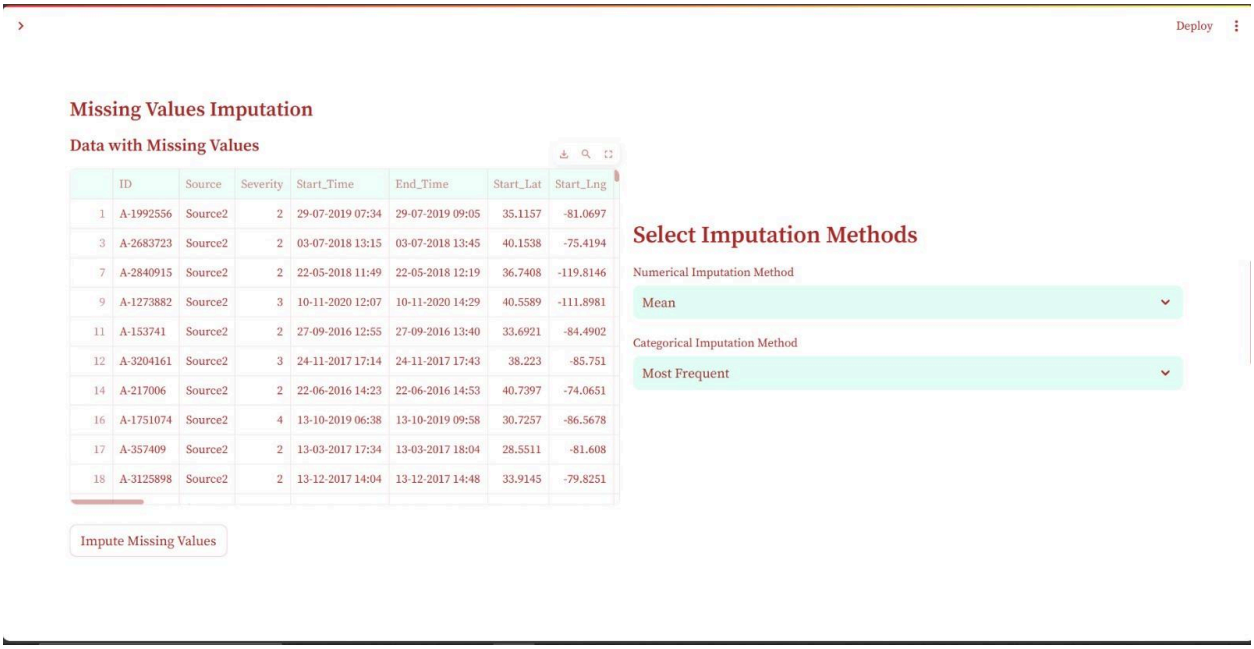
Frequency reflecting Pie Chart for irregular cardinality

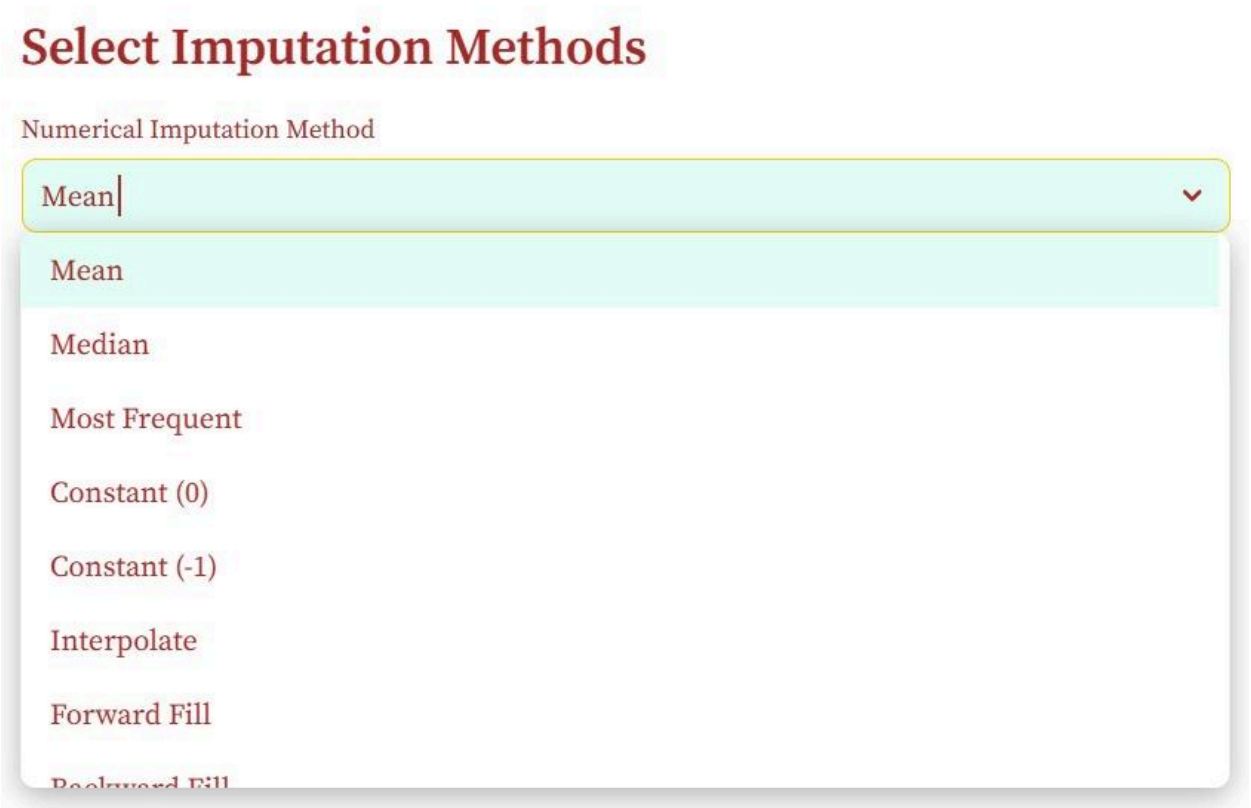
Can select any numerical columns:





Imputing missing values :





Numerical Imputation Method

Mean

Most Frequent

Constant (0)

Constant (-1)

Interpolate

Forward Fill

Backward Fill

KNN

Random

Deploy

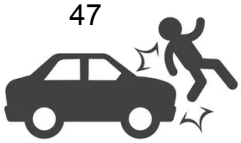
Impute Missing Values

Imputed Data

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Description
0	A-7307599	Source1	3	10-12-2019 10:36	10-12-2019 11:05	39.1153	-108.5447	39.1153	-108.5447	0	At Horizon Drive/Exit 31 - Accident.
1	A-1992556	Source2	2	29-07-2019 07:34	29-07-2019 09:05	35.1157	-81.0697	36.5219	-96.6832	0	Accident on SC-49 both ways at Forest Oaks Dr.
2	A-5306504	Source1	4	45:00.0	15:00.0	30.2384	-97.7385	30.2261	-97.7458	0.953	Incident on I-35 SB near US-290 Road closed. Take alternate route.
3	A-2683723	Source2	2	03-07-2018 13:15	03-07-2018 13:45	40.1538	-75.4194	36.5219	-96.6832	0	Accident on Eagleville Rd at Visitation Rd.
4	A-6573560	Source1	2	47:00.0	05:25.0	38.115	-77.5178	38.1271	-77.5136	0.868	Incident on I-95 NB near MM 118 Expect delays.
5	A-5006548	Source1	2	30:00.0	33:30.0	33.456	-111.974	33.4591	-112.0026	1.661	Stationary traffic on AZ-202 Loop W - Red Mountain Fwy W from Van
6	A-7015824	Source1	3	08-05-2020 12:44	08-05-2020 12:59	42.38	-71.1445	42.38	-71.1445	0	At RT-16/Huron Ave - Accident.
7	A-2840915	Source2	2	22-05-2018 11:49	22-05-2018 12:19	36.7408	-119.8146	36.5219	-96.6832	0	Right hand shoulder blocked due to accident on CA-99 Southbound at
8	A-5750958	Source1	2	06-10-2021 15:16	06-10-2021 16:34	33.9291	-117.1218	33.9355	-117.1256	0.495	Incident on GILMAN SPRINGS RD NB near EUCALYPTUS ST Drive wi
9	A-1273882	Source2	3	10-11-2020 12:07	10-11-2020 14:29	40.5589	-111.8981	36.5219	-96.6832	0	Lane blocked due to accident on I-15 Southbound at Exit 293 10600.

Previously Missing Values Filled

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Description	Street	City	County	State	Zipcode	Country	Timezone	Air
1	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
3	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
7	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
9	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
11	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
12	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
14	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
16	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
17	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No
18	None	None	None	None	None	None	None	36.5219	-96.6832	None	None	None	None	None	None	None	None	None	No



Feature Engineering and Data Correlation page:

Drag and drop your own dataset:

The Feature Engineering and Data Correlation Analysis Window

😍 This Window is specifically generated to carry out Feature Engineering and Feature Identification for predicting a target feature by visualizing hidden relationships and underlying complicated patterns. 📍

Upload your file

Drag and drop file here
Limit 200MB per file • CSV, TXT, XLSX, XLS

Browse files

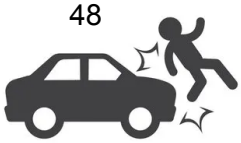
Correlation Tests Analysis

Correlation with target (numerical features):

	Severity	State	End_Lng	Start_Lng	End_Lat	Start_Lat	Humidity(%)	Pressure(in)	Precipitation(in)	Precipitation(in)_positive	End_Time	Weather_Timestamp	Description	Year	Di
Severity	1	0.4147	0.4109	0.4108	0.3549	0.3544	0.323	0.1939	0.1754	0.1754	0.1735	0.1731	0.1685	0.1353	

Correlation with target (numerical features):

Severity 1.00 ↑ 0.00	End_Time 0.17 ↑ 0.02	Minute 0.01 ↑ 0.00	County -0.09 ↓ -0.01	Timezone -0.34 ↓ -0.04
State 0.41 ↑ 0.05	Weather_Timestamp 0.17 ↑ 0.02	Sunrise_Sunset -0.00 ↓ -0.00	Hour -0.10 ↓ -0.01	Zipcode -0.40 ↓ -0.05
End_Lng 0.41 ↑ 0.05	Description 0.17 ↑ 0.02	Nautical_Twilight -0.01 ↓ -0.00	Day -0.10 ↓ -0.01	Source nan ↑ nan
Start_Lng 0.41 ↑ 0.05	Year 0.14 ↑ 0.02	Wind_Direction -0.01 ↓ -0.00	Weekday -0.13 ↓ -0.02	Country nan ↑ nan



Chi Sqaure Tests for Categorical Data

	Description	Zipcode	Street	City	Airport_Code	State	County	Severity	Weather_Condition	Timezone	Nautical_Twilight	Civil_Twilight	Astronomical_Twilight	Wind
0	99,031.5026	32,606.2649	11,557.1625	4,106.2716	3,149.9611	2,016.0152	713.2998	500	164.4823	85.4841	31.9326	28.3799	24.1152	

Chi Sqaure Test Bar Chart

Select the number of Attributes to visualise the Chi Square Bar results for

9

1 17

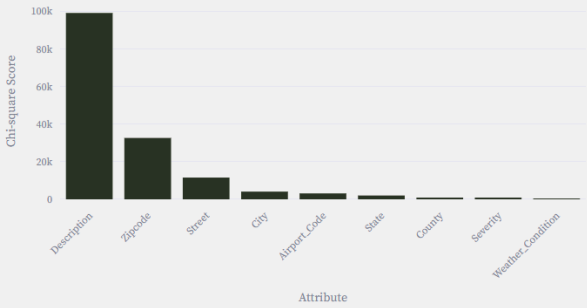
Chi Square Test Pie Chart

Select the number of Attributes to visualise the Chi Square Pie results for

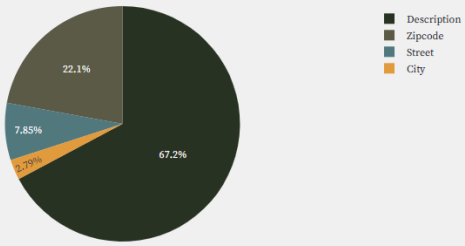
4

1 17

Top 9 Categorical Attributes and Their Chi-square Scores



Distribution of Chi-square Scores Among Top 4 Attributes



Select K Best Test for Numerical Attributes

Select the 'K' Value

5 − +

	Severity	Description	Unnamed: 0	ID	Weather_Timestamp	End_Time	End_Lng	Start_Lng	Zipcode	Start_Lat	End_Lat	Street	Airport_Code	City	County	Pressure(in)_positive
0	1.3878	1.2604	1.2206	1.2202	1.2163	1.2155	1.1799	1.1794	1.1599	1.1414	1.136	1.115	1.001	0.9932	0.9386	0.919

Select K Best Bar Chart

Select the number of attributes to Bar Chart

23

1 44

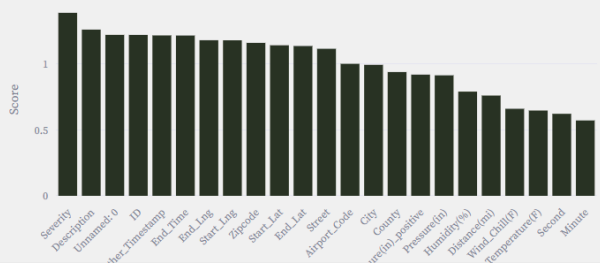
Select K BestPie Chart

Select the number of attributes to Pie Chart

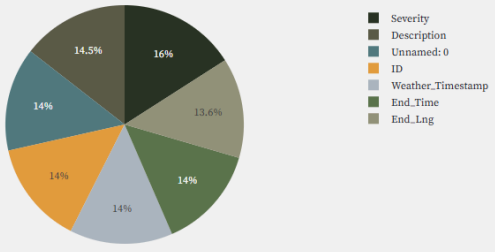
7

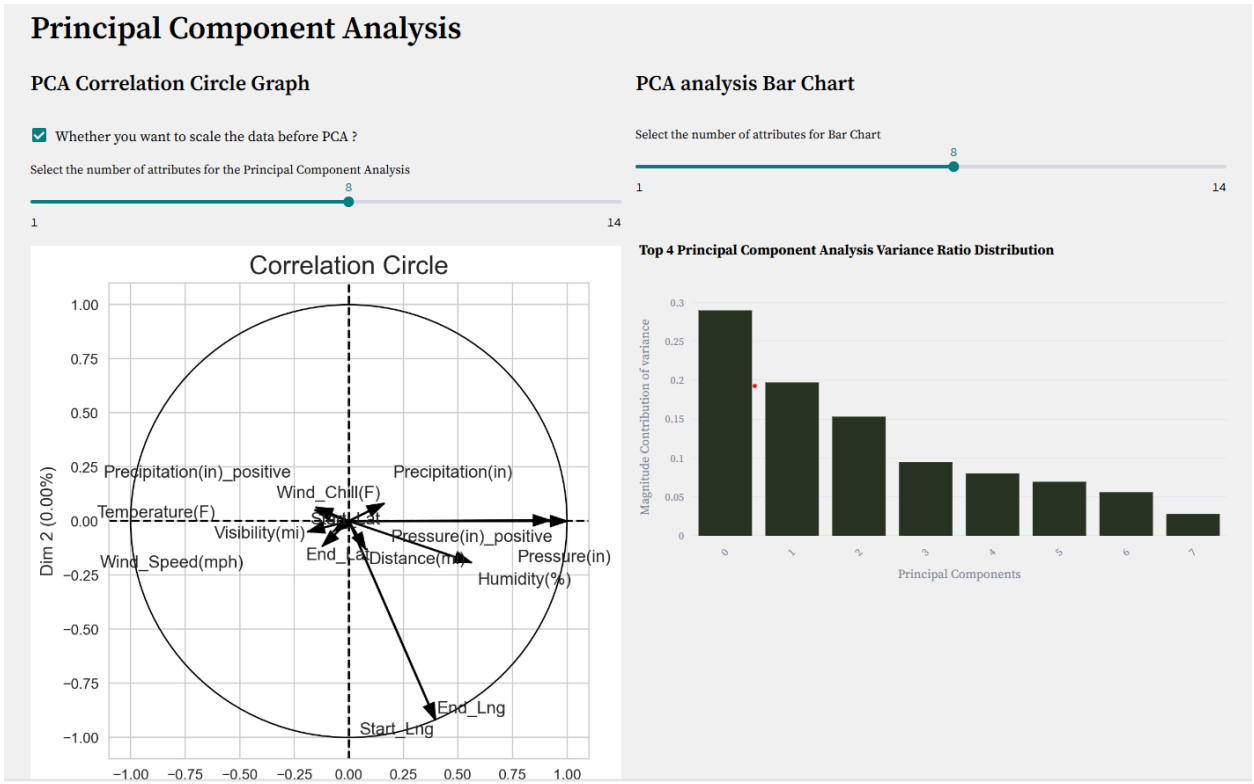
1 44

Top 23 Numerical Attributes and Their SelectKBest Scores

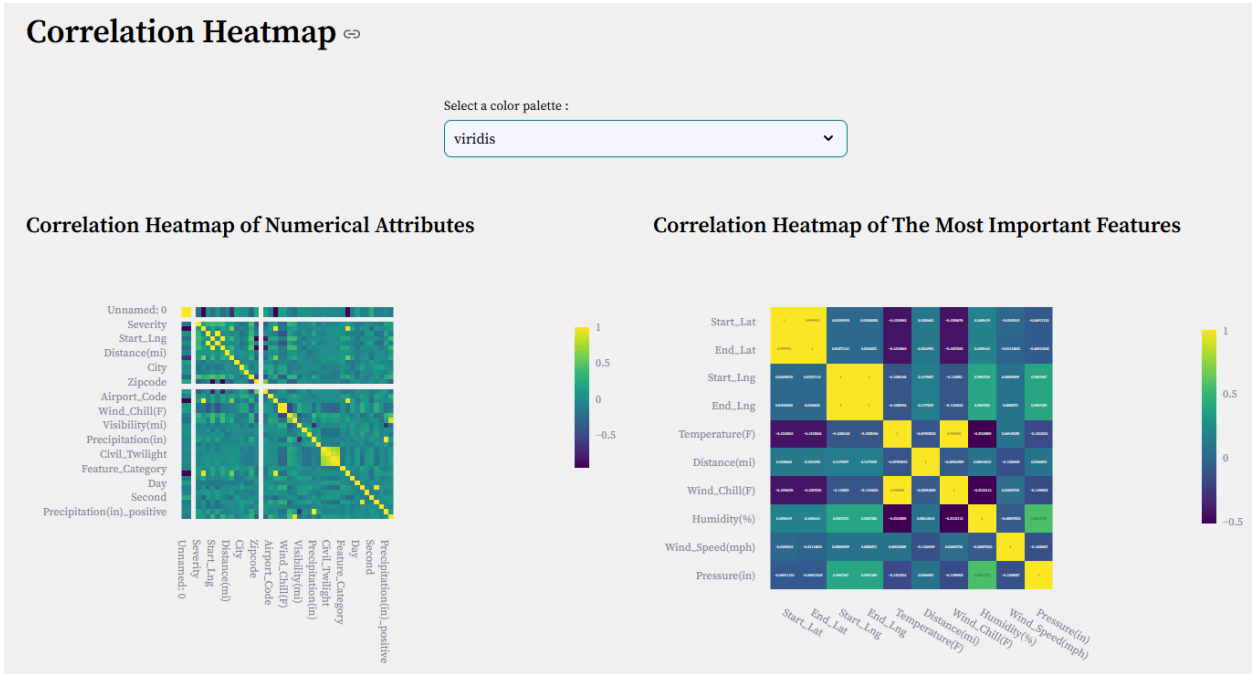


Distribution of SelectKBest Scores Among Top 7 Numerical Attributes

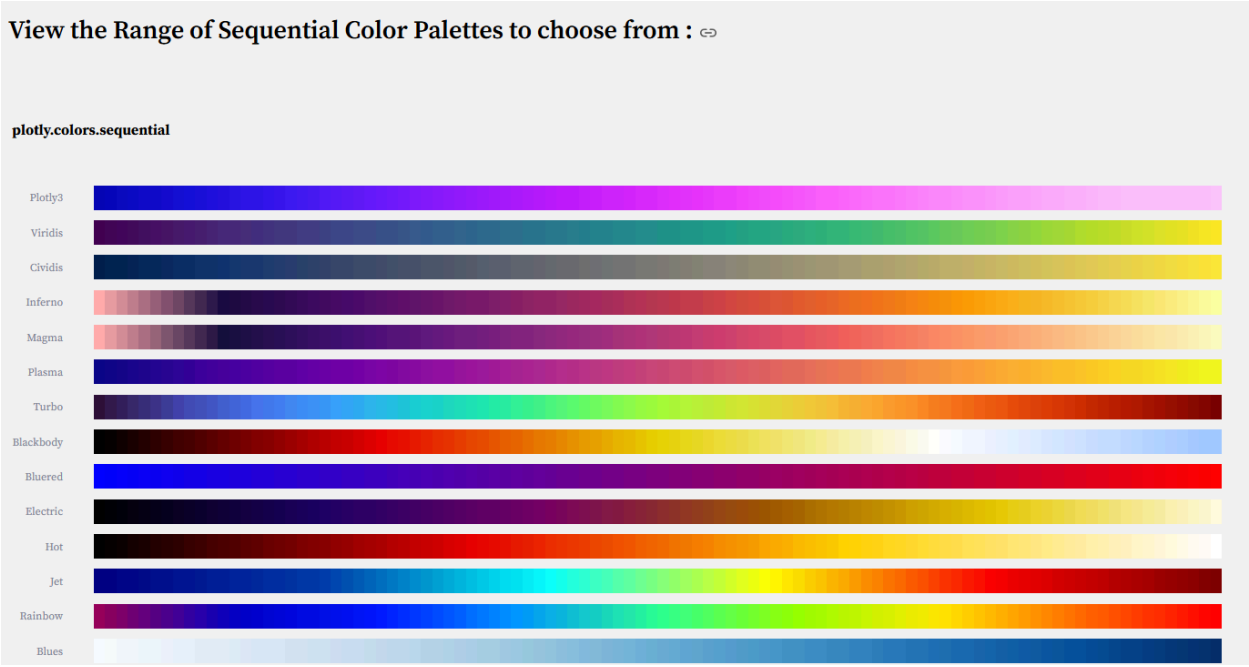




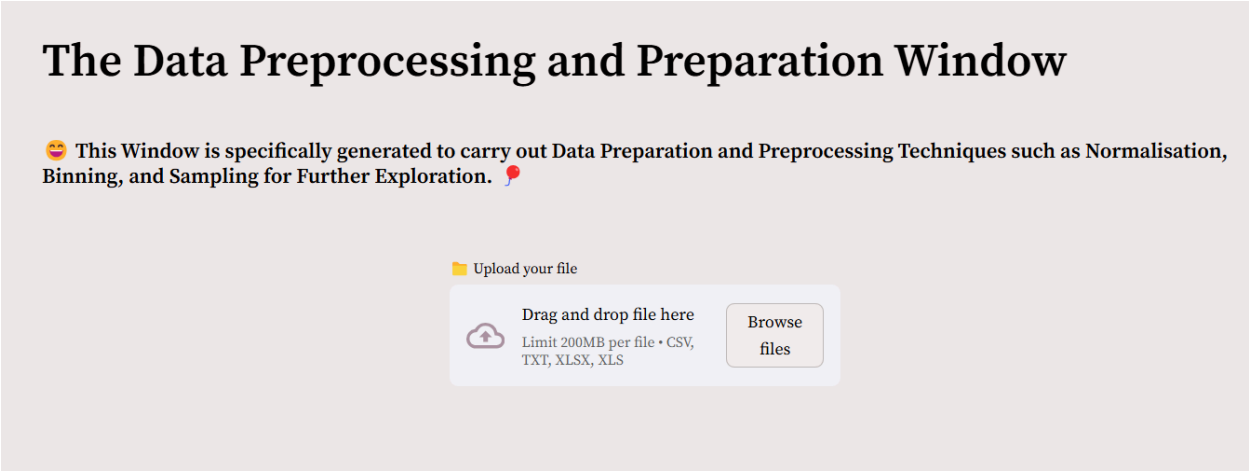
We can apply different types of color palette:



Can visualize different color palettes:



Data preprocessing:



Can select different attribute to normalize:

Deploy

Data Normalisation Process

Feature Scaling is an essential step in the data analysis and preparation of data for modeling. Wherein, we make the data scale-free for easy analysis.

Normalization is one of the feature scaling techniques. We particularly apply normalization when the data is skewed on either axis i.e. when the data does not follow the Gaussian distribution.

Min-Max Normalisation

The Normalised Data strictly lies between 0 to 1

Select the Attribute to visualise the Min-Max Normalisation for

Temperature(F)

Standard Normalisation

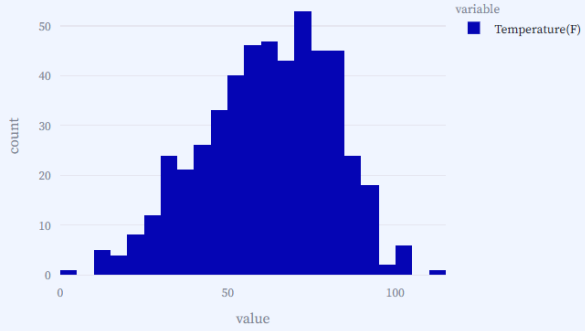
The Normalised Data strictly has the Mean of 0 and Standard Deviation of approximately 1

Select the Attribute to visualise the Standardisation for

Temperature(F)

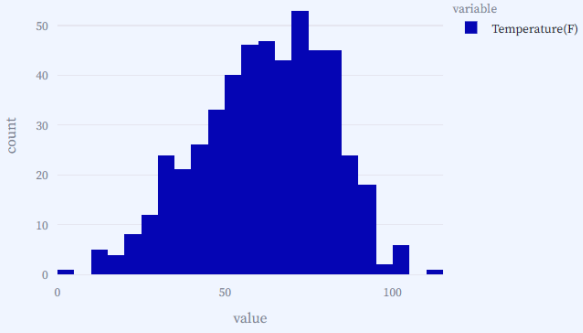
Visual inspection

Original Data

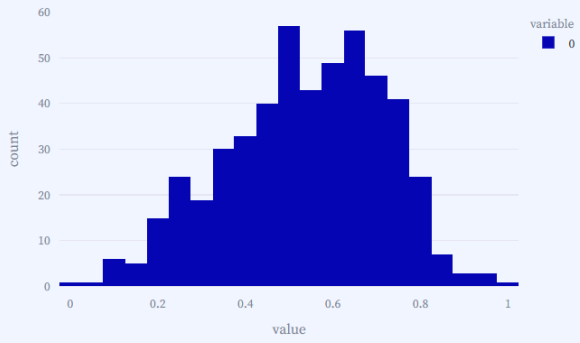


Visual inspection

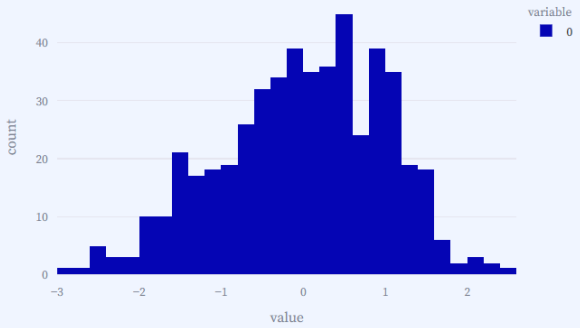
Original Data



Normalised Data



Normalised Data



Statistical test for normality


```
{
  "statistic" : 0.9871549068642782
  "pvalue" : 0.0002027261496641954
}
```


Statistical test for normality


```
{
  "statistic" : 0.9871549068642782
  "pvalue" : 0.0002027261496641954
}
```


Evaluate model performance

Original Data Scores: `0.9444444444444444`

Normalized Data Scores: `0.9444444444444444`

Evaluate model performance

Original Data Scores: `0.9444752475247524`

Normalized Data Scores: `0.9385346534653465`

Statistical Method Normalisation Verification

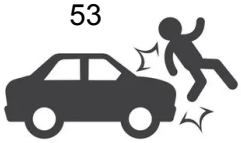
Normal Data - W statistic: `0.9871549068642782` P-value: `0.0002027261496641954`

Non-Normal Data - W statistic: `0.9871549068642782` P-value: `0.0002027261496641954`

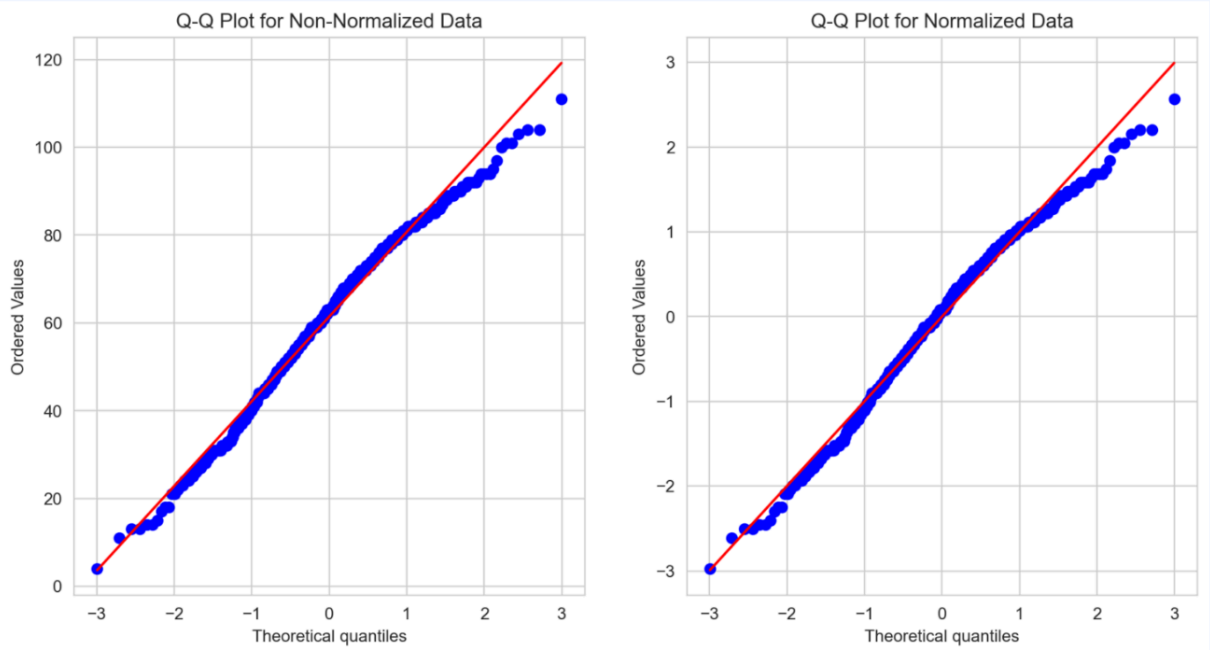
Interpreting the Results

• W statistic: Closer to 1 indicates data is more likely to be normal.

• P-value: A p-value greater than the chosen alpha level (commonly set at 0.05) suggests that the null hypothesis of normality cannot be rejected. A small p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, so you reject the null hypothesis of normality



Quantile-Quantile Plot



Binning:

Interactive Binning

Binning Options

Select the Numerical Attribute to visualise the Binning for

Temperature(F)

Enter bin labels (comma-separated):

very cold,cold,moderate,warm,hot

Enter value ranges for bins (comma-separated):

-100,32,50,70,90,200

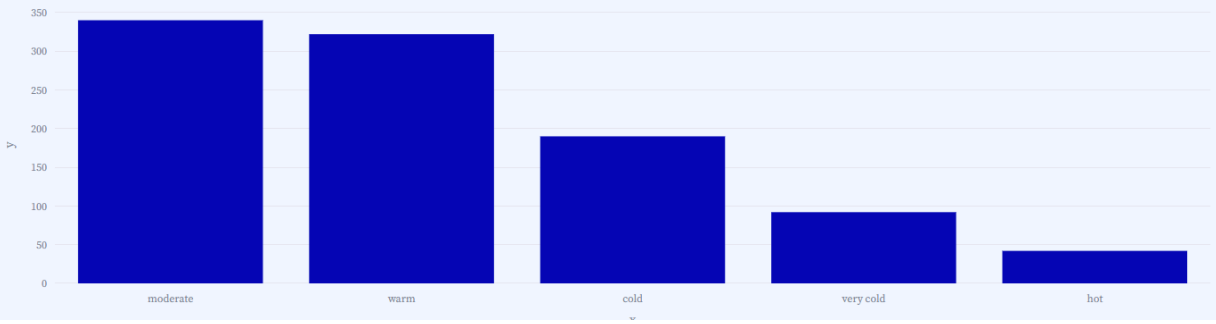
Perform Binning

Binning visualization:

Data after binning:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Description	Street
0	A-7307599	Source1	3	10-12-2019 10:36	10-12-2019 11:05	39.1153	-108.5447	39.1153	-108.5447	0	At Horizon Drive/Exit 31 - Accident.	I-70 E
1	A-1992556	Source2	2	29-07-2019 07:34	29-07-2019 09:05	35.1157	-81.0697	None	None	0	Accident on SC-49 both ways at Forest Oaks Dr.	Charl
2	A-5306504	Source1	4	45:00.0	15:00.0	30.2384	-97.7385	30.2261	-97.7458	0.953	Incident on I-35 SB near US-290 Road closed. Take alternate route.	I-35 S
3	A-2683723	Source2	2	03-07-2018 13:15	03-07-2018 13:45	40.1538	-75.4194	None	None	0	Accident on Eagleville Rd at Visitation Rd.	Eagle
4	A-6573560	Source1	2	47:00.0	05:25.0	38.115	-77.5178	38.1271	-77.5136	0.868	Incident on I-95 NB near MM 118 Expect delays.	I-95 N

Distribution of Bins:



Comparison of 25+ classifier model:

The Machine Learning Algorithm Comparison Window

In this Page Window, the **lazypredict** library is used for building and comparing several machine learning models at once.

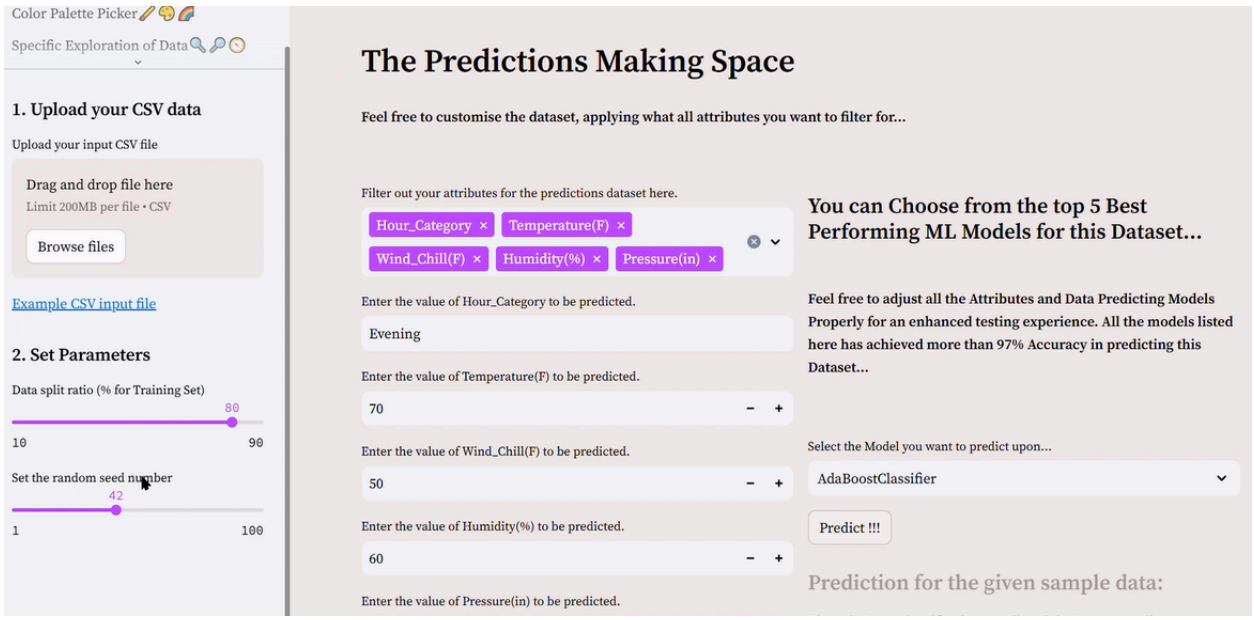
1. Dataset

Awaiting for CSV file to be uploaded.

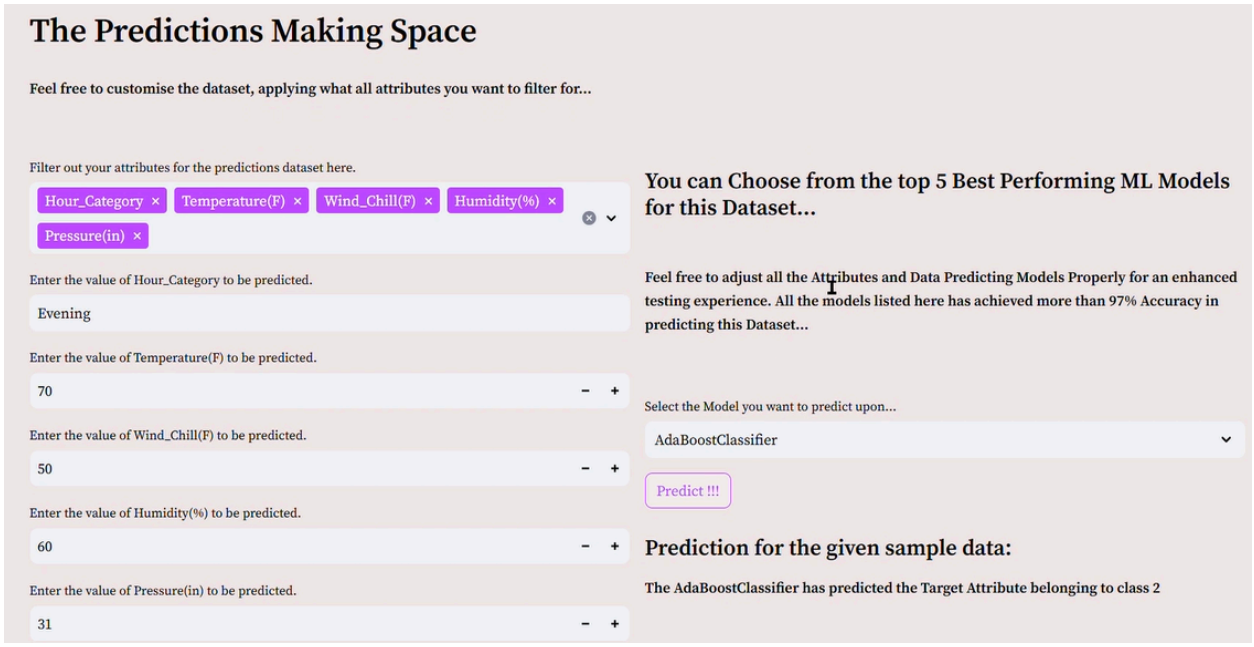
Press to use our Example "US Accidents Dataset"...

The Predictions Making Space

Feel free to customise the dataset, applying what all attributes you want to filter for...



Trained using AdaBoostClassifier:



Now Click Here to view The Overall Comparsion Report of 25 + Machine Learning Models...

SHOW MODELS' COMPARISON REPORT

1.2. Dataset dimension

X

```
(1000, 38)
```

Y

```
(1000,)
```

1.3. Variable details:

Predictor variables

2. Table of Model Performance

Training set

Model	Accuracy	Balanced Accuracy	F1 Score	Time Taken
LGBMClassifier	1	1	1	2.4185
BaggingClassifier	1	1	1	0.7213
RandomForestClassifier	1	1	1	1.0954
DecisionTreeClassifier	1	1	1	0.1085
ExtraTreeClassifier	1	1	1	0.0722
ExtraTreesClassifier	1	1	1	1.1554
LabelPropagation	1	1	1	0.4546
LabelSpreading	1	1	1	0.4074
SVC	0.9913	0.9914	0.9913	0.2436
QuadraticDiscriminantAr	0.9902	0.9901	0.9903	0.0619

Download training.csv File

We can download the files and plots:

Test set

Model	Accuracy	Balanced Accuracy	F1 Score	Time Taken
LGBMClassifier	1	1	1	7.1366
SVC	1	1	1	0.2397
RandomForestClassifier	1	1	1	2.2567
DecisionTreeClassifier	1	1	1	0.0796
ExtraTreesClassifier	1	1	1	0.7462
QuadraticDiscriminantAnalysis	0.9875	0.9904	0.9875	0.2115
BaggingClassifier	0.975	0.9722	0.9749	0.3384
ExtraTreeClassifier	0.9625	0.9583	0.9616	0.053
LabelPropagation	0.925	0.9167	0.9206	0.1963
LabelSpreading	0.925	0.9167	0.9206	0.2289

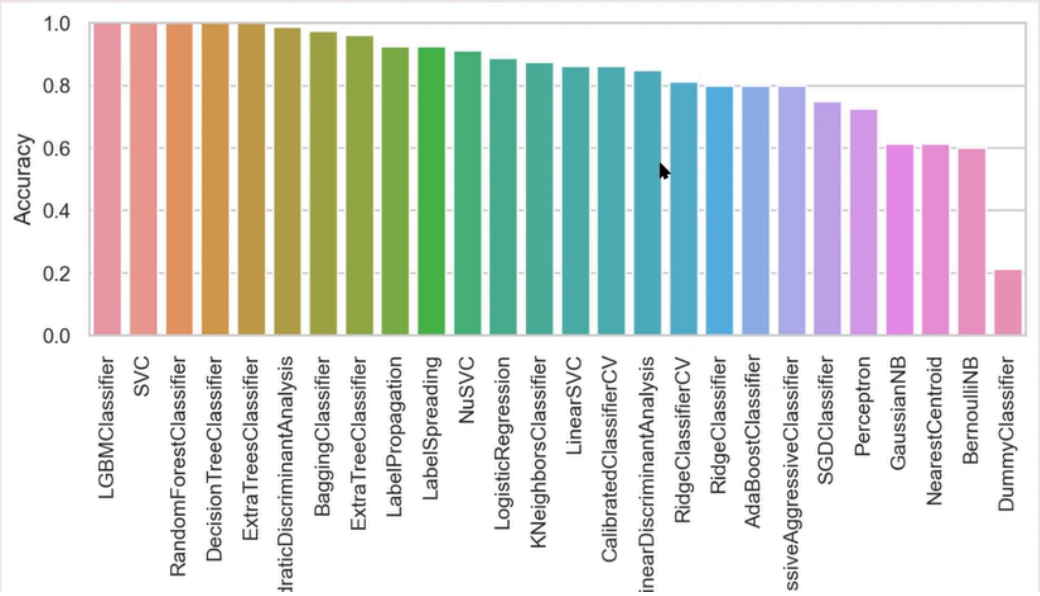
Download test.csv File

Accuracy vs model plot:

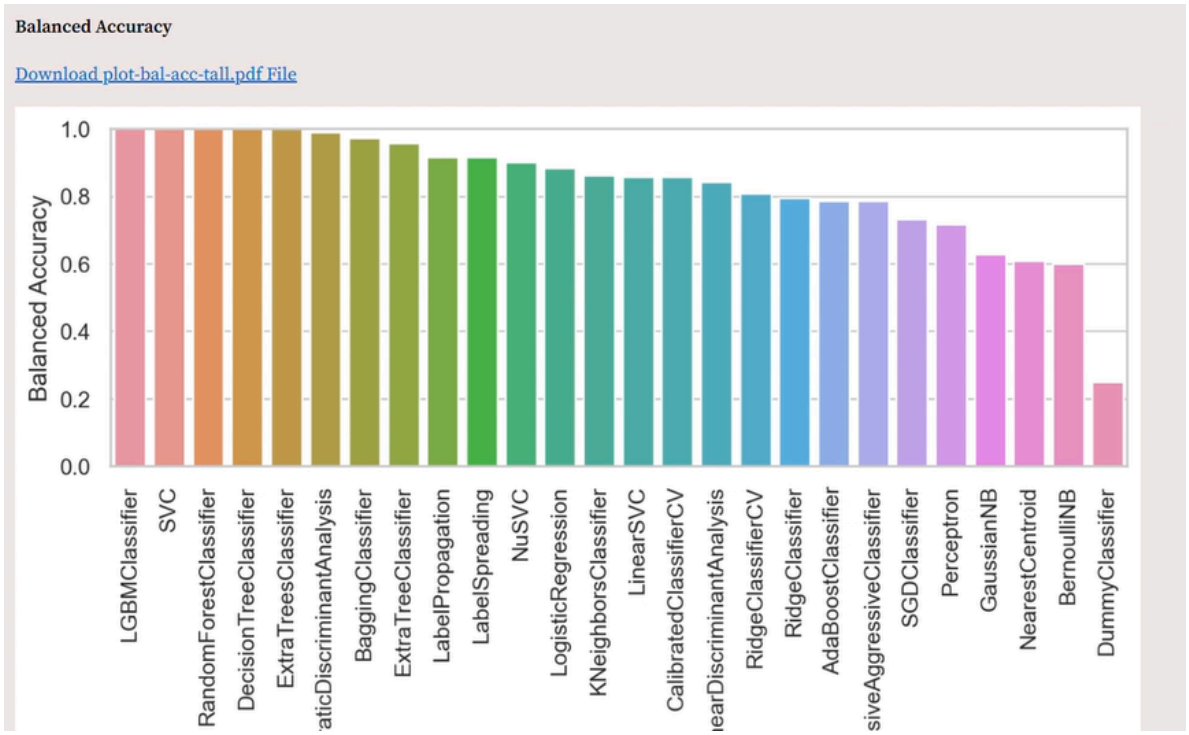
3. Plot of Model Performance (Test set)

Accuracy

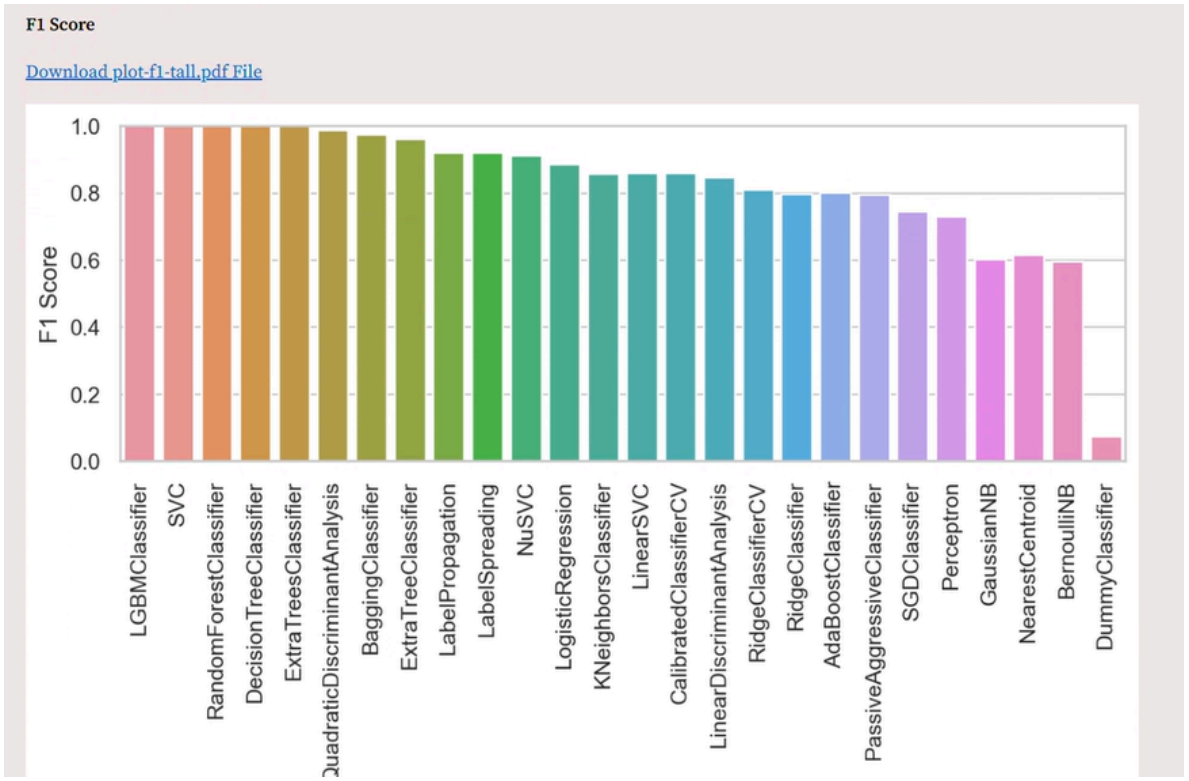
Download plot-acc-tall.pdf File



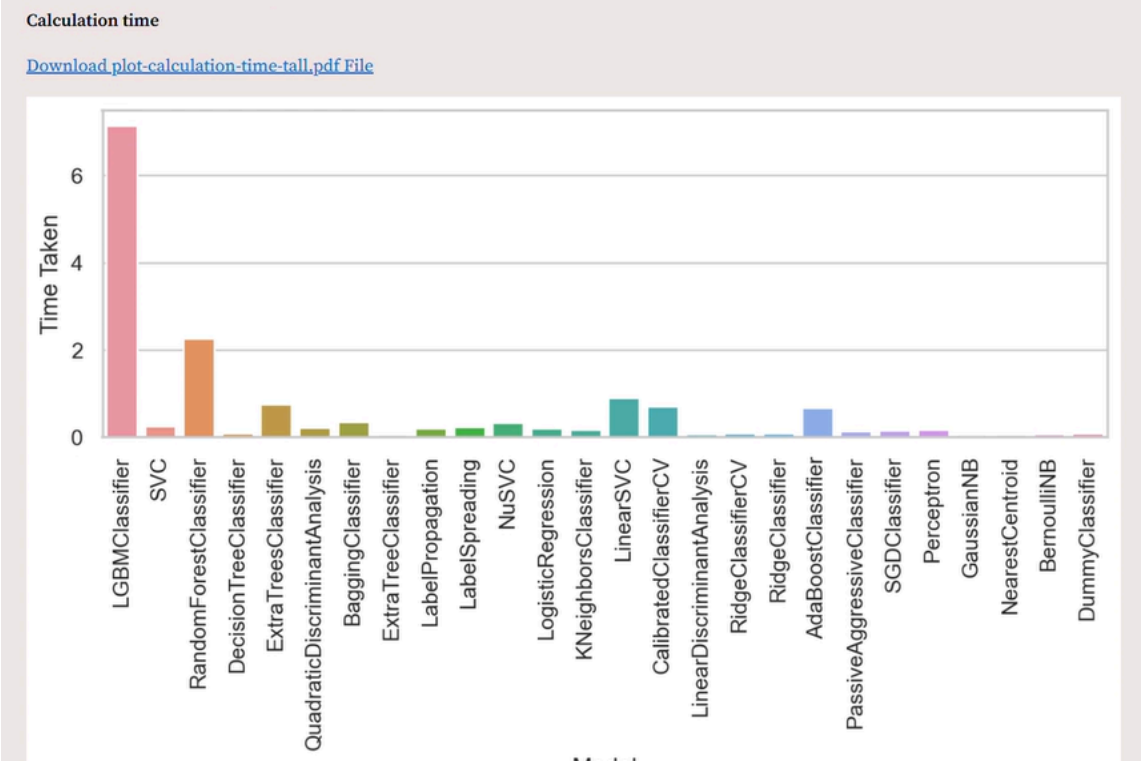
Balanced accuracy vs model plot:



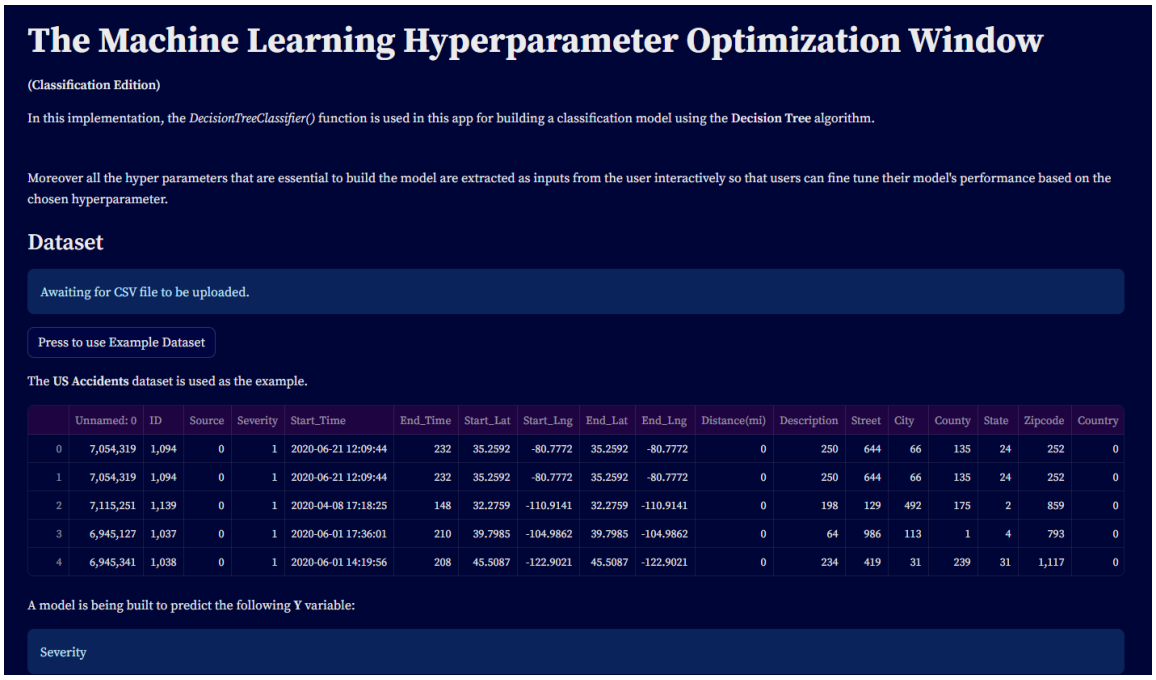
F1 score vs model plot:



Calculation time vs model plot:



Optimization for various parameters:



The Machine Learning Hyperparameter Optimization Window

(Classification Edition)

In this implementation, the *DecisionTreeClassifier()* function is used in this app for building a classification model using the **Decision Tree** algorithm.

Moreover all the hyper parameters that are essential to build the model are extracted as inputs from the user interactively so that users can fine tune their model's performance based on the chosen hyperparameter.

Dataset

Awaiting for CSV file to be uploaded.

Press to use Example Dataset

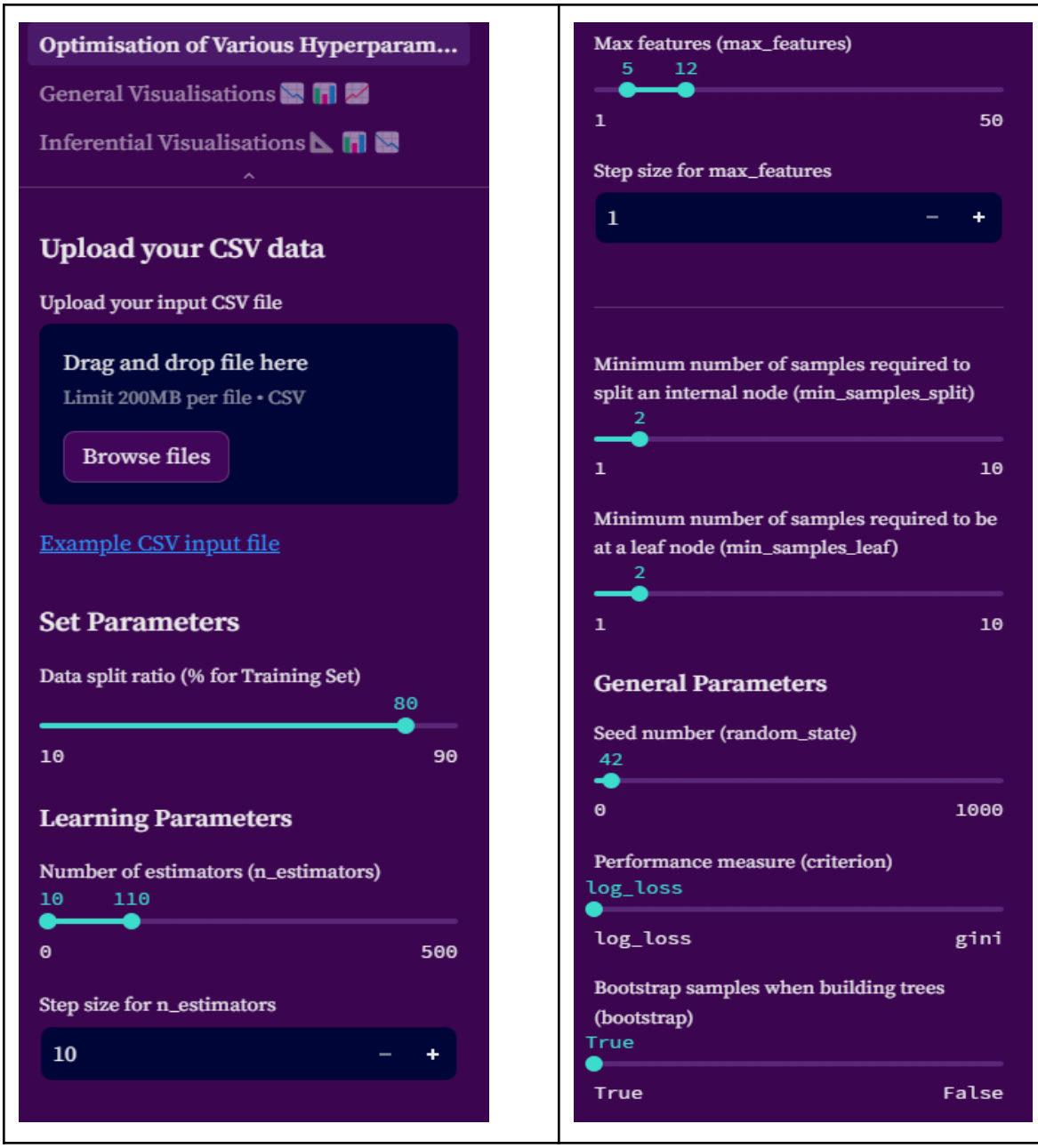
The **US Accidents** dataset is used as the example.

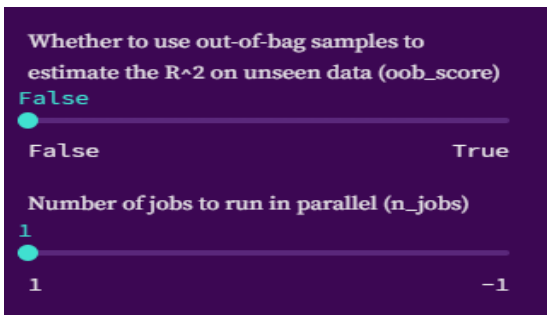
	Unnamed: 0	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Description	Street	City	County	State	Zipcode	Country
0	7,054,319	1,094	0	1	2020-06-21 12:09:44	232	35.2592	-80.7772	35.2592	-80.7772	0	250	644	66	135	24	252	0
1	7,054,319	1,094	0	1	2020-06-21 12:09:44	232	35.2592	-80.7772	35.2592	-80.7772	0	250	644	66	135	24	252	0
2	7,115,251	1,139	0	1	2020-04-08 17:18:25	148	32.2759	-110.9141	32.2759	-110.9141	0	198	129	492	175	2	859	0
3	6,945,127	1,037	0	1	2020-06-01 17:36:01	210	39.7985	-104.9862	39.7985	-104.9862	0	64	986	113	1	4	793	0
4	6,945,341	1,038	0	1	2020-06-01 14:19:56	208	45.5087	-122.9021	45.5087	-122.9021	0	234	419	31	239	31	1,117	0

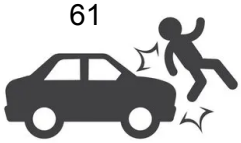
A model is being built to predict the following Y variable:

Severity

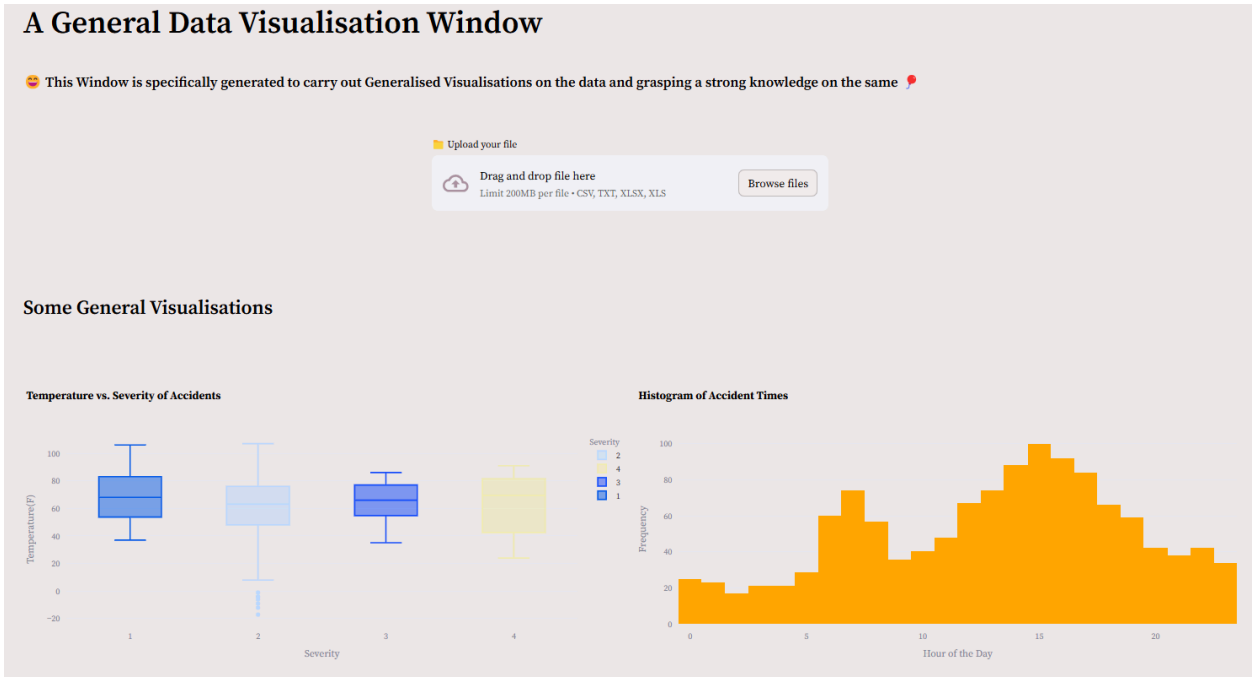
Different settings:

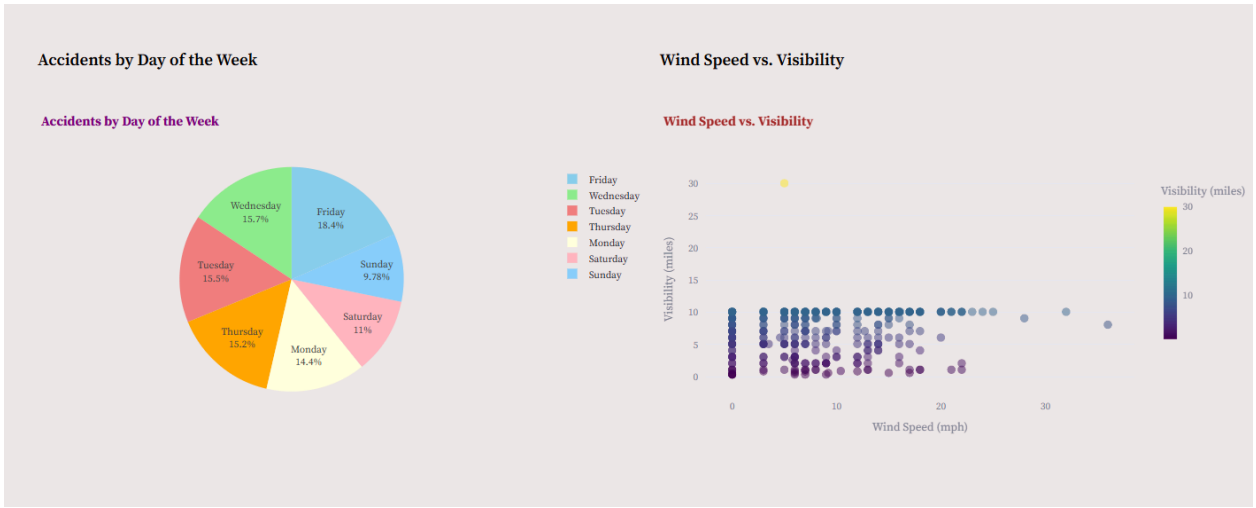




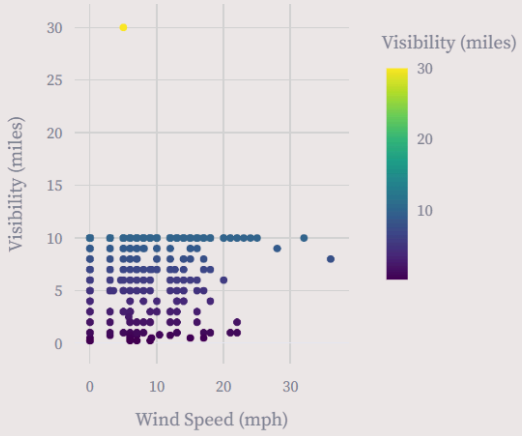


General data visualization:

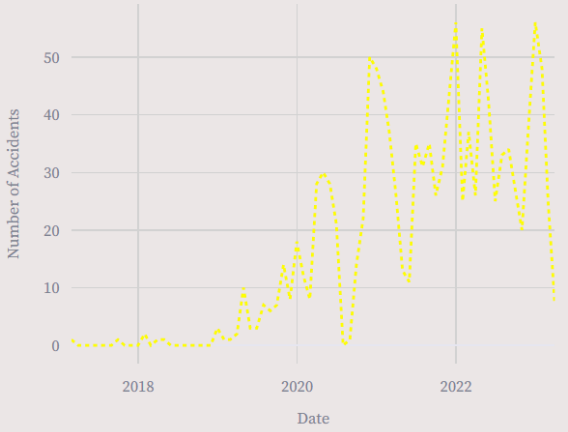




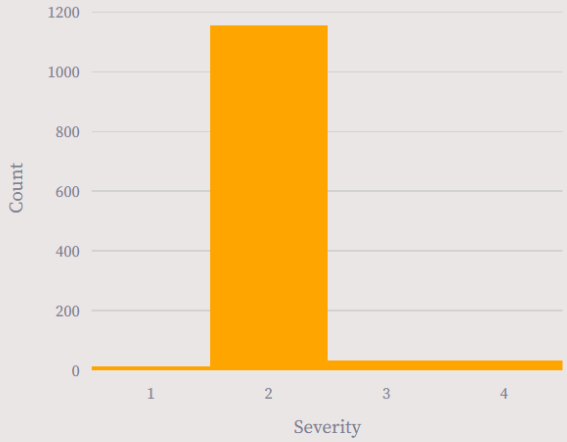
Scatter Plot of Wind Speed Vs. Visibility



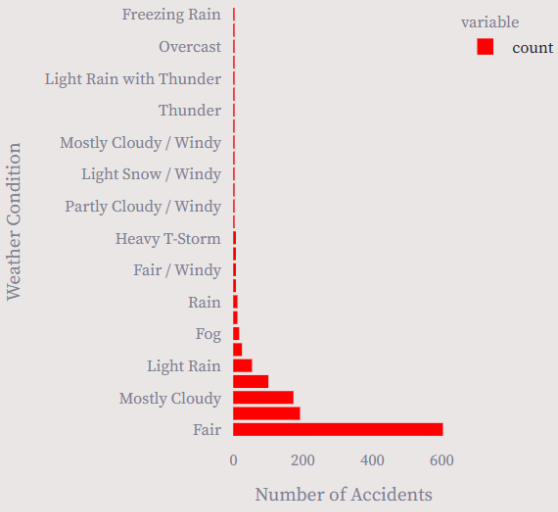
Trends Line Chart



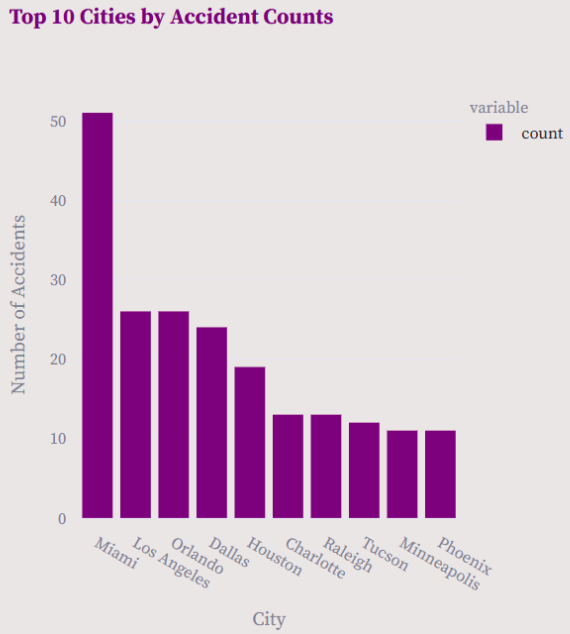
Count Plot of Accident Severity



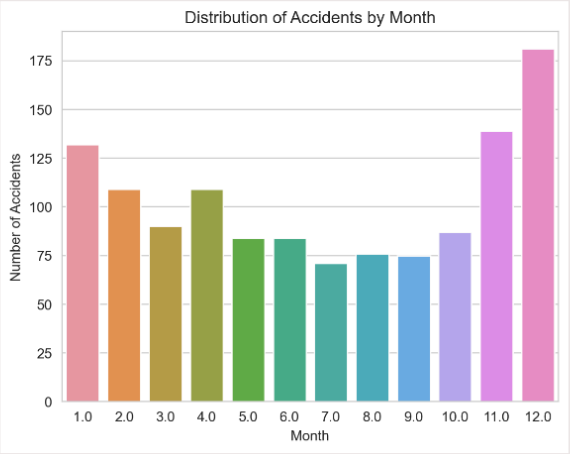
Accidents by Weather Condition



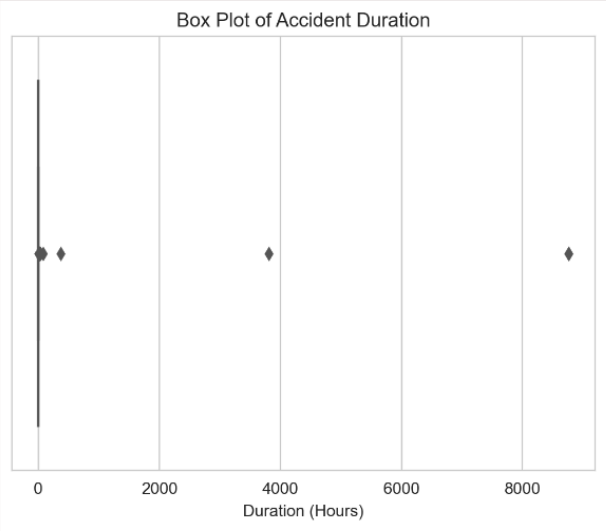
Top 10 Cities by Accident Counts



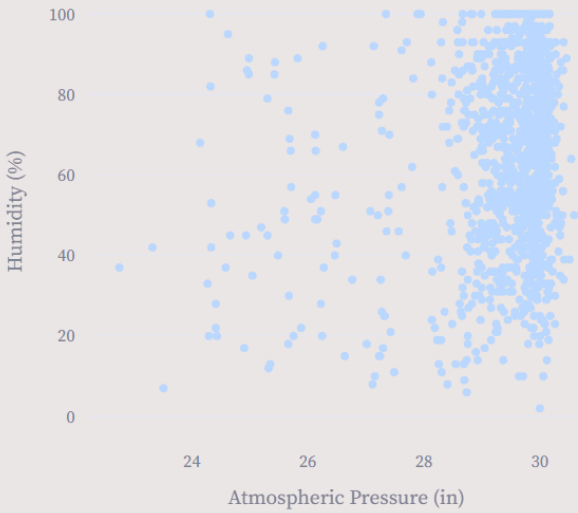
Distribution of Accidents by Month



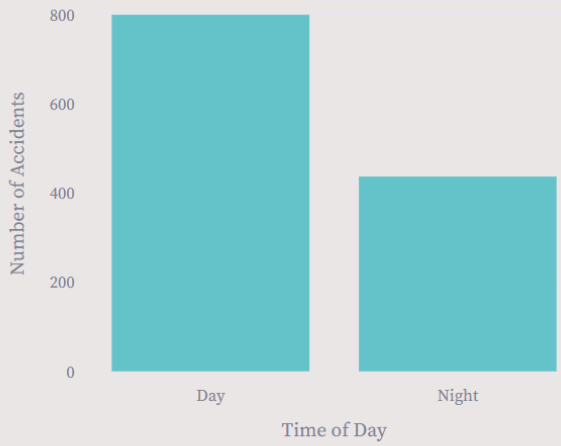
Box Plot of Accident Duration



Pressure vs. Humidity

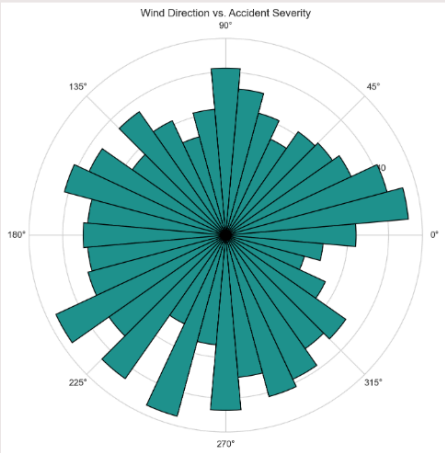


Day vs. Night Accidents

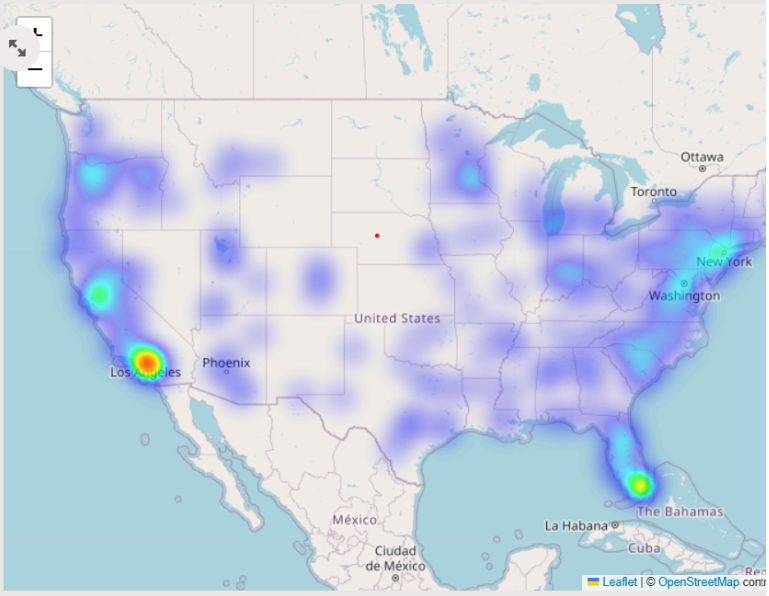


Visibility vs. Severity

Rose Plot for Accidents and Wind Direction



Kernel Density Accidents in Folium Map



Inferential Page:

The Inferential Data Visualisation Window

😄 **This Window is specifically generated to carry out Highly Meaningful and Inferential Visualisations on the data and arriving at Conclusions on the same.** 🏓

Upload your file

Drag and drop file here
Limit 200MB per file • CSV, TXT, XLSX, XLS

Browse files

Some Inferences:

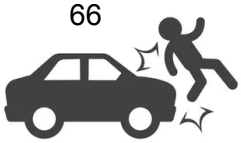
Identification of the 10 accident-prone streets in USA.

Inference:

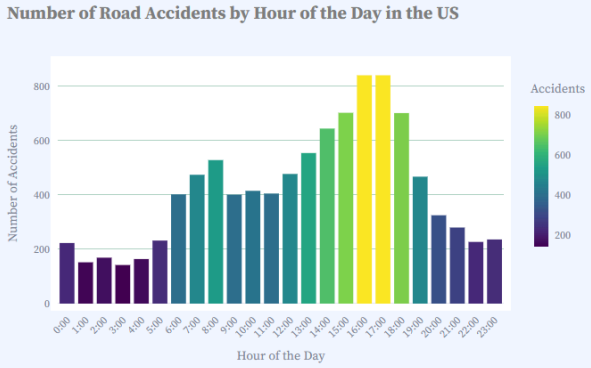
Top 10 Accident-Prone Streets in the US

1. **Most Accident-Prone Streets:**
   - **I-95 S** is the most accident-prone street, with **126 accident cases**.
   - **I-95 N** follows closely with **122 accident cases**.
2. **Significant Streets:**
   - **I-5 N** shows a notable number of accidents, totaling **75 cases**.
   - **I-94 W** and **I-80 W** also have significant accident cases, with **63** and **62 cases** respectively.
3. **Other Notable Streets:**
   - **I-25 N**, **I-70 W**, and **I-80 E** each have **60**, **56**, and **56 accident cases**, respectively.
   - **I-10 E** and **I-5 S** show **55** and **53 accident cases**, respectively.
4. **Visual Appeal:**
   - The color scale from red to purple effectively highlights the distribution of accidents, with red indicating higher accident counts and purple indicating lower counts, enhancing visual understanding of the most accident-prone streets.

Overall, the plot identifies that **I-95 S** and **I-95 N** are the streets with the highest number of road accidents, suggesting a need for targeted safety measures on these routes. This insight can inform traffic safety initiatives and preventive measures to reduce accidents on these critical streets.



Number of Road Accidents by Hour of the Day in the US



The hour with the most road accidents is 16.0:00 with 840 accidents.

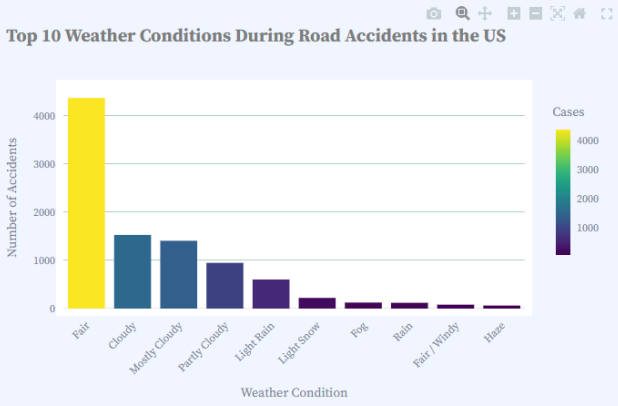
Inference:

Number of Road Accidents by Hour of the Day in the US

1. **Peak Accident Hour**:
   - **16:00** is the hour with the most road accidents, totaling **840 accidents**.
2. **High Accident Period**:
   - There is a significant increase in accidents between **15:00 and 18:00**, indicating a peak period likely corresponding to afternoon and evening rush hours.
3. **Morning Increase**:
   - A noticeable increase in accidents starts around **6:00**, with a steady rise through the morning hours, peaking again around **8:00**.
4. **Lower Accident Periods**:
   - Early morning hours (**0:00 to 5:00**) and late evening hours (**20:00 to 23:00**) have relatively fewer accidents.
5. **Visual Appeal**:
   - The 'viridis' color scale effectively highlights the distribution of accidents, with brighter colors indicating higher accident counts, enhancing visual understanding of critical times.

Overall, the plot clearly identifies the times of day with the highest accident rates, particularly focusing on afternoon rush hours, suggesting the need for targeted traffic management during these peak periods.

Top 10 Weather Conditions During Road Accidents in the US



The weather condition with the most road accidents is Fair with 4367 accidents.

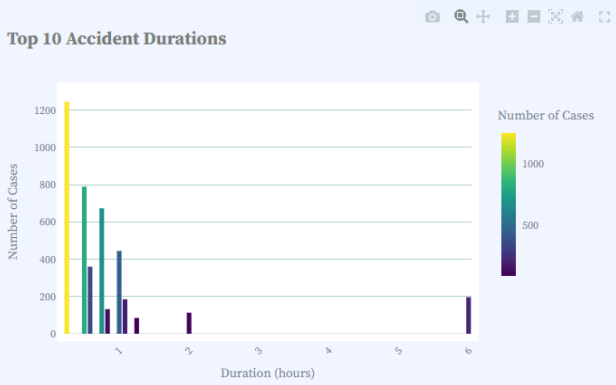
Inference:

Top 10 Weather Conditions During Road Accidents in the US

1. **Most Accident-Prone Weather Condition**:
   - **Fair** is the weather condition with the most road accidents, totaling **4367 accidents**.
2. **Significant Weather Conditions**:
   - **Cloudy** and **Mostly Cloudy** conditions also show a considerable number of accidents, though significantly fewer than Fair weather.
   - **Partly Cloudy** and **Light Rain** conditions follow, contributing to the overall accident count but at a lower frequency.
3. **Less Frequent Conditions**:
   - **Light Snow**, **Fog**, **Rain**, **Fair/Windy**, and **Haze** have relatively fewer accidents compared to the top conditions.
4. **Visual Appeal**:
   - The 'viridis' color scale effectively highlights the distribution of accidents, with brighter colors indicating higher accident counts, enhancing visual understanding of critical weather conditions.

Overall, the plot helps identify that most road accidents occur under **Fair** weather conditions, suggesting that even in seemingly safe conditions, vigilance is necessary. This insight can inform safety campaigns and preventive measures across varying weather conditions.

Top 10 Accident Durations



The accident duration with the most cases is 0.25 hours with 1244.0 cases.

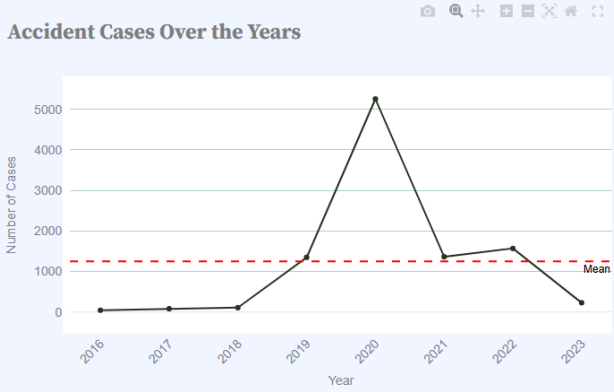
Inference:

Top 10 Accident Durations

1. **Most Frequent Accident Duration:**
   - The duration with the most cases is **0.25 hours**, totaling **1244 cases**.
2. **Significant Durations:**
   - Durations around **0.5 hours** and **1 hour** also show a considerable number of accidents, indicating a high frequency of shorter duration accidents.
3. **Less Frequent Durations:**
   - Durations extending up to **6 hours** are observed but with significantly fewer cases compared to shorter durations.
4. **Visual Appeal:**
   - The 'viridis' color scale effectively highlights the distribution of accidents, with brighter colors indicating higher accident counts, enhancing visual understanding of critical accident durations.

Overall, the plot helps identify that most accidents are resolved within a short period, typically **0.25 hours**. This insight can inform traffic management and emergency response strategies to improve road safety and reduce accident durations.

Accident Cases Over the Years



The year with the most accident cases is 2020 with 5255 cases.

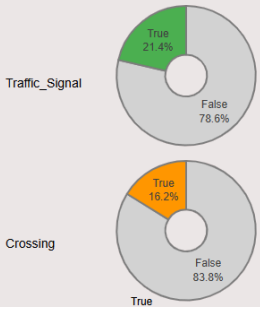
Inference:

Accident Cases Over the Years

1. **Year with the Most Accident Cases:**
   - **2020** had the most accident cases, totaling **5255 cases**.
2. **Trend Over the Years:**
   - There is a significant increase in the number of cases leading up to 2020, followed by a sharp decline in 2021.
   - A slight increase is observed in 2022, but the number of cases drops again in 2023.
3. **Mean Line:**
   - The red dashed line represents the mean number of cases over the years, providing a benchmark to compare yearly data.
4. **Visual Appeal:**
   - The line chart effectively shows the trend of accident cases over the years, with clear peaks and troughs that highlight significant changes in accident frequencies.

Overall, the plot indicates that 2020 was an outlier year with an exceptionally high number of accidents. Understanding the factors contributing to this spike can help in formulating strategies to prevent similar surges in the future.

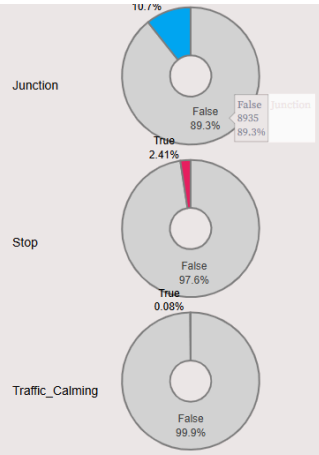
Top 5 Useful Amenities in Road Conditions



Inference:

Top 5 Useful Amenities in Road Conditions

1. **Traffic Signal**:
   - **Traffic signals** are present in **21.4%** of the road conditions, making them the most common useful amenity. This indicates a significant presence of controlled intersections to manage traffic flow and enhance safety.
2. **Crossing**:
   - **Crossings** account for **16.2%** of the road conditions. This suggests that pedestrian crossings are a crucial feature in urban planning to ensure pedestrian safety and reduce accidents.
3. **Junction**:
   - **Junctions** are found in **10.7%** of the cases. The presence of junctions highlights the importance of road intersections and their potential role in accident occurrences due to the convergence of multiple roads.
4. **Stop**:
   - **Stop signs** are present in **2.41%** of the road conditions, indicating a lesser but still important role in traffic control, particularly in residential or low-traffic areas.



5. **Traffic Calming**:
   - **Traffic calming measures** (e.g., speed bumps) are seen in only **0.08%** of the road conditions. This low percentage points to a minimal implementation of such measures, which are typically used to slow down traffic and enhance safety in specific areas.
6. **Visual Appeal**:
   - The use of donut charts effectively displays the proportion of True and False values for each amenity, with distinct colors aiding visual interpretation and understanding of the distribution.

Overall, the plot highlights that traffic signals and crossings are the most prevalent amenities in road conditions, underscoring their critical role in managing traffic and ensuring safety. The lower presence of stop signs and traffic calming measures suggests areas for potential improvement in road safety measures.

Congratulations, you have reached the last page of this Web Application !!!

Thank You So Much for Allocating Time for getting yourself immersed in this Data Exploration and Visualisation Dashboard....

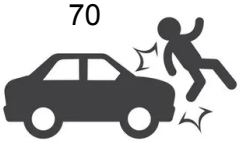
Our web page url:

https://us-accidents-dashboard.streamlit.app/

Source code:

User Authentication:

```
import pickle
from pathlib import Path
import streamlit as st
import streamlit_authenticator as stauth
from streamlit.components.v1 import html
from st_pages import hide_pages

st.set_page_config(initial_sidebar_state="collapsed")

# User authentication
names = ["Mirsha Morningstar", "Rameez Akther", "Chandru", "Mekesh"]
usernames = ["Mirsha Morningstar","Rameez", "Chandru","Mekesh"]

# Load hashed passwords
file_path = Path(__file__).parent/"hashed_pw.pkl"
with file_path.open("rb") as file:
    hashed_passwords = pickle.load(file)

hide_pages(["streamlit_app","user_auth","re_auth"])

placeholder1 = st.empty()
placeholder2 = st.empty()
placeholder1.title("💥🚗💣US Accidents Dashboard")
placeholder2.markdown(
    """Hello There !!!  A Warm Welcome to our Application Dashboard. This highly interactive and sophisticated dashboard provides insights into intricate patterns, relationships and brings forth the overall knowledge of the " US accidents data ". This Web- Application Dashboard is specially built using Streamlit and Plotly. Feel free to Access it by logging in below."""
)


authenticator = stauth.Authenticate(names,usernames,hashed_passwords,"US Accidents Dashboard","abcdef",cookie_expiry_days=0)
```

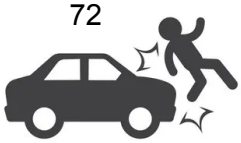



```
name,authentication_status,username = authenticator.login("Login your credentials",
"main")



if authentication_status == False :
    hide_pages(["streamlit_app","user_auth","re_auth","EDA_window"])
    st.error("The provided Username or Password is incorrect !!!")

if authentication_status == None:
    hide_pages(["streamlit_app","user_auth","re_auth","EDA_window"])
    st.warning("Kindly enter your Username and Password")

if authentication_status:
    st.session_state["logged_in"] = True

    print("Success")
    placeholder1.empty()
    placeholder2.empty()
    hide_pages(["streamlit_app","user_auth","re_auth","EDA_window"])
    st.switch_page(r"pages\2_Change_Theme🎨🖼️🔄.py")
```

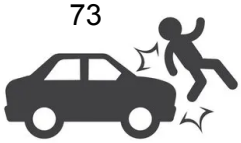
Change theme:

```
import toml
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
from mlxtend.plotting import plot_pca_correlation_graph
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
```

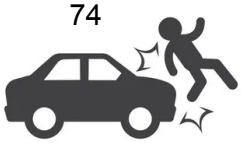

```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff

from scipy import stats
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

from streamlit_option_menu import option_menu

warnings.filterwarnings('ignore')

def
modify(primary="#bd4bff",back="#efe6e6",secondary="#f0f2f5",font="serif",text="blac
k",dark = False):
    # Path to your config.toml file
    config_path =
r"C:\Users\mekes\Downloads\STREAMLIT-20240526T095929Z-001\STREAMLIT\.strea
mlit\config.toml"

    # Read the contents of config.toml
    with open(config_path, "r") as f:
        config_data = toml.load(f)


    # Modify the theme section as per your requirements
    config_data["theme"]["primaryColor"] = primary  # Sample: Purple
```

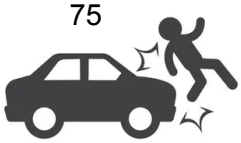
```
    config_data["theme"]["backgroundColor"] = back  # Sample: Light Gray
    config_data["theme"]["secondaryBackgroundColor"] = secondary  # Sample: Light
Blueish Gray
    config_data["theme"]["textColor"] = text  # Sample: Blue
    config_data["theme"]["font"] = font  # Sample: Serif Font

    # Write back the modified contents to config.toml
    with open(config_path, "w") as f:
        toml.dump(config_data, f)


st.set_page_config(initial_sidebar_state="collapsed",page_title= " Change Application
Themes ",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png",layout="wide")



st.markdown('''# **Change Themes based on Principles of Perception and Cognition**''')
add_vertical_space(2)
st.markdown('''##### :smile: This Window is specifically generated to change the
themes of the entire application based on the user's choice of preference :balloon: ''' )

add_vertical_space(1)

st.warning("**Note:** The Themes applied in this Window shall be reflected throughout
the web application ( i.e across all tabs ) ",icon = "🚨")

add_vertical_space(2)
c1,c2 = st.columns(2)
with c1:
    st.markdown("#### Set to the Defualt Light Theme of the app")
    add_vertical_space(1)
    if st.button("Set Theme to Default Light"):
        modify()
```

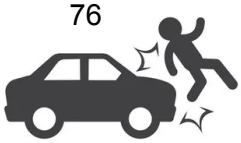
```
with c2:
   st.markdown("#### Set to the Defualt Dark Theme of the app")
   add_vertical_space(1)
   if st.button("Set Theme to Default Dark"):
      modify("#FF4B4B","#0e1117","#262730",text="#FAFAFA")


add_vertical_space(5)


st.markdown("### Light Themes on Visual Theory of Perception")
add_vertical_space(2)
c1,c2 = st.columns(2)

with c1:
   st.markdown("###### Clear Understanding to the User ")
   st.write("Teal is a calming yet assertive color that promotes clear communication and understanding. Light gray provides a neutral and balanced background, enhancing readability and ensuring that content stands out effectively. Ghost white offers a subtle distinction in the sidebar, helping users navigate through sections with clarity.")
   add_vertical_space(1)
   if st.button("Use Clear Light Theme"):
      modify("#008080",'#f0f0f0','#f8f8ff')


with c2:
   st.markdown("###### Compelling to the User ")
   st.write(" The vibrant tomato red draws attention and creates a sense of urgency or excitement, making it compelling for users to engage with important elements. Alice blue provides a clean and soothing background, ensuring that the content remains easy to read and visually appealing. Beige offers a subtle contrast in the sidebar, aiding in navigation without overshadowing the main content.")
   add_vertical_space(1)
   if st.button("Use Compelling Light Theme"):
      modify("#ff6347",'#f0f8ff','#f5f5dc')


add_vertical_space(3)

c1,c2 = st.columns(2)
```

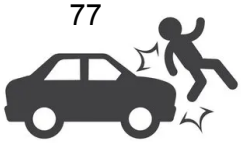
```
with c1:
    st.markdown("###### Professional Standard: ")
    st.write("Dark gray exudes professionalism and sophistication while maintaining readability and visual hierarchy. White serves as a clean and timeless background, emphasizing content and ensuring a professional aesthetic. Light gray in the sidebar provides a subtle contrast and organization without detracting from the main focus.")
    add_vertical_space(1)
    if st.button("Use Professional Light Theme"):
        modify("#333333",'#ffffff','#f0f0f0')

    add_vertical_space(5)

with c2:
    st.markdown("###### Visually Attractive and Eye-catching ")
    st.write("Vivid orange is a bold and energetic color that immediately captures attention and stimulates interest. Light peach provides a soft and inviting background, creating a warm and welcoming atmosphere while ensuring readability. Khaki in the sidebar offers a complementary contrast, enhancing visual appeal and encouraging exploration of the interface.")
    add_vertical_space(1)
    if st.button("Use Attractive Light Theme"):
        modify("#ffaa00",'#f9f6f2','#f0e68c')

add_vertical_space(5)



st.markdown("### Dark Themes on Visual Theory of Perception")
add_vertical_space(2)
c1, c2 = st.columns(2)

with c2:
    st.markdown("###### Compelling to the User")
    st.write("The vibrant orange-red stands out against the dark background, immediately drawing the user's attention. The dark gray background provides a sleek and modern appearance while enhancing readability. The charcoal sidebar offers a subtle contrast, making it easy to navigate through sections without distracting from the main content.")
    add_vertical_space(1)
    if st.button("Use Compelling Dark Theme"):
        modify("#FF204E", "#222831", "#31363F",text="#F5EDED")
```

```
with c1:
    st.markdown("###### Clear Understanding to the User")
    st.write("Turquoise promotes clear communication and understanding, ensuring that important information stands out effectively. Dark slate gray creates a visually appealing contrast while maintaining a professional appearance. Gunmetal in the sidebar offers a subtle distinction, aiding in navigation and organization without overwhelming the user.")
    add_vertical_space(1)
    if st.button("Use Clear Dark Theme"):
        modify("#40E0D0", "#030637", "#3C0753",text="#EEEEEE")

add_vertical_space(3)

c1, c2 = st.columns(2)

with c1:
    st.markdown("###### Professional Standard")
    st.write("Light gray exudes professionalism and sophistication while ensuring readability and visual hierarchy. Black provides a classic and timeless background, emphasizing content and maintaining a professional aesthetic. Dark slate gray in the sidebar offers a subtle contrast and organization without detracting from the main focus.")
    add_vertical_space(1)
    if st.button("Use Professional Dark Theme"):
        modify("#CCCCCC", "#1a1a1a", "#2a2a2a",text="#EEEEEE")

    add_vertical_space(5)

with c2:
    st.markdown("###### Visually Attractive and Eye-catching")
    st.write("Vivid yellow is bold and eye-catching, instantly capturing attention and creating visual interest. Midnight black offers a dramatic and striking background, providing a sleek and modern appearance. Dark charcoal in the sidebar complements the primary color, enhancing visual appeal and encouraging exploration of the interface.")
    add_vertical_space(1)
    if st.button("Use Attractive Dark Theme"):
        modify("#FFD700", "#121212", "#222222",text="#F5EDED")

add_vertical_space(5)
```

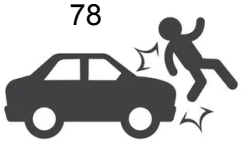
```
st.markdown("### Themes on Visual Theory of Cognition")
add_vertical_space(2)
c1, c2 = st.columns(2)

with c2:
    st.markdown("###### Electric Colors")
    st.write("Electric green is a vibrant and attention-grabbing color that stimulates excitement and engagement. Black provides a dramatic and high-contrast background, enhancing the visibility of content and creating a sense of urgency. Navy blue in the sidebar offers a complementary contrast, aiding navigation without overwhelming the user.")
    add_vertical_space(1)
    if st.button("Use Electric Colors Theme"):
        modify("FF3EA5",  "#8576FF", "#41C9E2",text="#F5EDED")

with c1:
    st.markdown("###### Radiative Glow Colors")
    st.write("Radiant yellow promotes clarity and comprehension, ensuring that important information stands out effectively. White offers a clean and minimalist background, enhancing readability and creating a sense of openness. Salmon in the sidebar provides a subtle yet warm contrast, aiding navigation and organization without detracting from the main focus.")
    add_vertical_space(1)
    if st.button("Use Radioactive Glow Theme"):
        modify("#ffcc00", "#ffffff", "#E3FEF7",text="brown")

add_vertical_space(3)
c1, c2 = st.columns(2)

with c1:
    st.markdown("###### Classic Neutrals Standard")
    st.write("Dark gray exudes professionalism and sophistication while maintaining readability and visual hierarchy. Light gray provides a neutral and balanced background, ensuring content stands out effectively. Medium gray in the sidebar offers a subtle contrast and organization, enhancing navigation without overshadowing the main content.")
    add_vertical_space(1)
    if st.button("Use Classic Neutrals Theme"):
        modify("#333333", "#f5f5f5", "#808080")
```

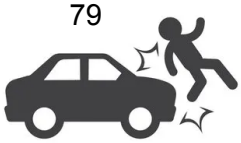
```
    add_vertical_space(5)

with c2:
    st.markdown("###### Vivid Contrast")
    st.write("Vivid red is bold and attention-grabbing, instantly capturing attention and creating visual interest. Vivid yellow offers a striking and high-contrast background, enhancing visibility and drawing the user's focus. Vivid blue in the sidebar complements the primary color, creating a visually appealing contrast and encouraging exploration of the interface.")
    add_vertical_space(1)
    if st.button("Use Vivid Contrast Theme"):
        modify("#ff0000", "#ffff00", "#8576FF")
```

Color palette:

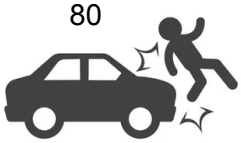
```
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff
```

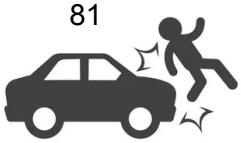
```
from scipy import stats
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier


warnings.filterwarnings('ignore')



import streamlit as st

sys.path.insert(0, ".")
from utils import show_palette, model_dict, get_palette, \
    sort_func_dict, store_palette, display_matplotlib_code, display_plotly_code,\
     get_df_rgb, enhancement_range, plot_rgb_3d, plot_hsv_3d, print_praise


gallery_files = glob(os.path.join(".", "images", "*"))
gallery_dict = {image_path.split("/")[-1].split(".")[-2].replace("-", " "): image_path
    for image_path in gallery_files}

st.set_page_config(initial_sidebar_state="collapsed",page_title= " Change Application
Themes ",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png",layout="wide")


st.markdown("""# **Extract Synthetic Custom Color Palettes from your own images and
images across online**""")
add_vertical_space(2)
st.markdown("""##### :smile: This Window is specifically Synthesized for Extracting
Color Palettes from User's Choice of Images :balloon: """ )

add_vertical_space(1)
```

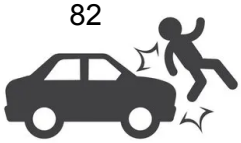
```
st.sidebar.markdown("---")

toggle = st.sidebar.checkbox("Toggle Update", value=True, help="Continuously update
the pallete with every change in the app.")
click = st.sidebar.button("Find Palette", disabled=bool(toggle))

st.sidebar.markdown("---")
st.sidebar.header("Settings")
palette_size = int(st.sidebar.number_input("palette size", min_value=1, max_value=20,
value=5, step=1, help="Number of colors to infer from the image."))
sample_size = int(st.sidebar.number_input("sample size", min_value=5,
max_value=3000, value=500, step=500, help="Number of sample pixels to pick from the
image."))

# Image Enhancement
enhancement_categories = enhancement_range.keys()
enh_expander = st.sidebar.expander("Image Enhancements", expanded=False)
with enh_expander:

    if st.button("reset"):
        for cat in enhancement_categories:
            if f"{cat}_enhancement" in st.session_state:
                st.session_state[f"{cat}_enhancement"] = 1.0
enhancement_factor_dict = {
    cat: enh_expander.slider(f"{cat} Enhancement",
                             value=1.,
                             min_value=enhancement_range[cat][0],
                             max_value=enhancement_range[cat][1],
                             step=enhancement_range[cat][2],
                             key=f"{cat}_enhancement")
    for cat in enhancement_categories
}
enh_expander.info("**Try the following**\n\nColor Enhancements = 2.6\n\nContrast
Enhancements = 1.1\n\nBrightness Enhancements = 1.1")

# Clustering Model
model_name = st.sidebar.selectbox("machine learning model", model_dict.keys(),
help="Machine Learning model to use for clustering pixels and colors together.")
sklearn_info = st.sidebar.empty()
```

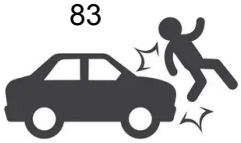
```
sort_options = sorted(list(sort_func_dict.keys()) + [key + "_r" for key in
sort_func_dict.keys() if key!="random"])
sort_func = st.sidebar.selectbox("palette sort function", options=sort_options, index=5)

# Random Number Seed
seed = int(st.sidebar.number_input("random seed", value=42, help="Seed used for all
random samplings."))
np.random.seed(seed)
st.sidebar.markdown("---")


# =======
#   App
# =======
img =
Image.open(r"C:\Users\mekes\Downloads\STREAMLIT-20240526T095929Z-001\STREA
MLIT\sample_nature.jpg")
# provide options to either select an image form the gallery, upload one, or fetch from
URL
upload_tab, url_tab = st.tabs(["Upload", "Image URL"])

with upload_tab:
    file = st.file_uploader("Upload Art", key="file_uploader")
    if file is not None:
        try:
            img = Image.open(file)
        except:
            st.error("The file you uploaded does not seem to be a valid image. Try uploading
a png or jpg file.")
    if st.session_state.get("image_url") not in ["", None]:
        st.warning("To use the file uploader, remove the image URL first.")

with url_tab:
    url_text = st.empty()

    # FIXME: the button is a bit buggy, but it's worth fixing this later

    # url_reset = st.button("Clear URL", key="url_reset")
    # if url_reset and "image_url" in st.session_state:
    #     st.session_state["image_url"] = ""
    #     st.write(st.session_state["image_url"])
```

```
    url = url_text.text_input("Image URL", key="image_url")

    if url!="":
        try:
            response = requests.get(url)
            img = Image.open(BytesIO(response.content))
        except:
            st.error("The URL does not seem to be valid.")

# convert RGBA to RGB if necessary
n_dims = np.array(img).shape[-1]
if n_dims == 4:
    background = Image.new("RGB", img.size, (255, 255, 255))
    background.paste(img, mask=img.split()[3]) # 3 is the alpha channel
    img = background

# apply image enhancements
for cat in enhancement_categories:
    img = getattr(ImageEnhance, cat)(img)
    img = img.enhance(enhancement_factor_dict[cat])

# show the image
with st.expander("🖼️  Artwork", expanded=True):
    st.image(img, use_column_width=True)


if click or toggle:

    df_rgb = get_df_rgb(img, sample_size)

    # (optional for later)
    # plot_rgb_3d(df_rgb)
    # plot_hsv_3d(df_rgb)

    # calculate the RGB palette and cache it to session_state
    st.session_state["palette_rgb"] = get_palette(df_rgb, model_name, palette_size,
sort_func=sort_func)

    if "palette_rgb" in st.session_state:
```

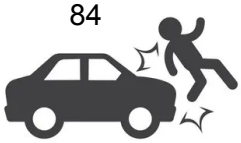
```
    # store individual colors in session state
    store_palette(st.session_state["palette_rgb"])

    st.write("---")

    # sort the colors based on the selected option
    colors = {k: v for k, v in st.session_state.items() if k.startswith("col_")}
    sorted_colors = {k: colors[k] for k in sorted(colors, key=lambda k:
int(k.split("_")[-1]))}

    # find the hex representation for matplotlib and plotly settings
    palette_hex = [color for color in sorted_colors.values()][:palette_size]
    with st.expander("Adopt this Palette", expanded=False):
      st.pyplot(show_palette(palette_hex))

      matplotlib_tab, plotly_tab = st.tabs(["matplotlib", "plotly"])

      with matplotlib_tab:
        display_matplotlib_code(palette_hex)

        import matplotlib as mpl
        from cycler import cycler

        mpl.rcParams["axes.prop_cycle"] = cycler(color=palette_hex)
        import matplotlib.pyplot as plt

        x = np.arange(5)
        y_list = np.random.random((len(palette_hex), 5))+2
        df = pd.DataFrame(y_list).T

        area_tab, bar_tab = st.tabs(["area chart", "bar chart"])

        with area_tab:
          fig_area , ax_area = plt.subplots()
          df.plot(kind="area", ax=ax_area, backend="matplotlib", )
          st.header("Example Area Chart")
          st.pyplot(fig_area)

        with bar_tab:
          fig_bar , ax_bar = plt.subplots()
          df.plot(kind="bar", ax=ax_bar, stacked=True, backend="matplotlib", )
```

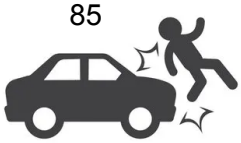
```
            st.header("Example Bar Chart")
            st.pyplot(fig_bar)


        with plotly_tab:
            display_plotly_code(palette_hex)

            import plotly.io as pio
            import plotly.graph_objects as go
            pio.templates["sophisticated"] = go.layout.Template(
                layout=go.Layout(
                colorway=palette_hex
                )
            )
            pio.templates.default = 'sophisticated'

            area_tab, bar_tab = st.tabs(["area chart", "bar chart"])

            with area_tab:
                fig_area = df.plot(kind="area", backend="plotly", )
                st.header("Example Area Chart")
                st.plotly_chart(fig_area, use_container_width=True)

            with bar_tab:
                fig_bar = df.plot(kind="bar", backend="plotly", barmode="stack")
                st.header("Example Bar Chart")
                st.plotly_chart(fig_bar, use_container_width=True)


else:
    st.info("👈 Click on 'Find Palette' ot turn on 'Toggle Update' to see the color palette.")

st.sidebar.success(print_praise())
st.sidebar.write("---\n")
```

Specific Exploration of a data:

```
import numpy as np
import pandas as pd
import sklearn.datasets
```

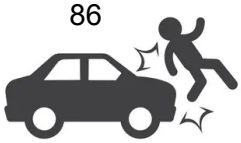



```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
from sklearn import datasets

import streamlit.components.v1 as components
from pandas.api.types import (
    is_categorical_dtype,
    is_datetime64_any_dtype,
    is_numeric_dtype,
    is_object_dtype,
)

st.set_page_config(initial_sidebar_state="collapsed", page_title="The Data Frames
Explorer Window",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
     },
     page_icon="exploratory-analysis (1).png")

st.title("Welcome to the Data Frame Explorer Window")

st.markdown("##### This Window provides you with a highly interactive and
customisable method to explore our US Accidents dataset")

add_vertical_space(2)

st.markdown("###### Feel free to explore and filter out our dataset...")
add_vertical_space(2)


df = pd.read_csv("US_Accident23_1000.csv")
filtered_df = dataframe_explorer(df, case=False)
st.dataframe(filtered_df, use_container_width=True)
```

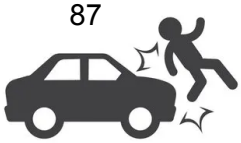



General exploration of data:

```
import numpy as np
import pandas as pd
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer

#from pandas_profiling import ProfileReport
from ydata_profiling import ProfileReport
from streamlit_pandas_profiling import st_profile_report

st.set_page_config(initial_sidebar_state="collapsed",page_title= " The Exploratory
Data Analysis Window",
       menu_items={
        'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
        'Report a bug': "mailto:a.k.mirsha9@gmail.com",
        'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png")

# Web App Title
st.markdown('''
# **The Exploratory Data Analysis ( EDA ) Window**''')
add_vertical_space(2)
st.markdown('''
:smile: This is the **EDA Window** created in Streamlit using the
**pandas-profiling** library. :sparkles: \n
This Window of our Application Provides the following :balloon: ;

**Credit: :crossed_flags: ** App built in `Python` + `Streamlit`. Refer our
[Documentation](https://drive.google.com/drive/folders/1Amtvj8MfXswe0AVTreA4IjS
G2UeOl4Lt?usp=sharing) for more details ...

---
''')

st.text('''
```

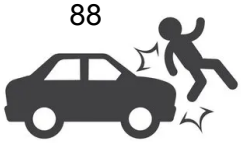
```
            \t\t1. Input Sample Data Frame\n
            \t\t2. Overview and Alerts\n
            \t\t3. Missing, distinct values, correlation memory size and visual plots of each
variable\n
            \t\t4. Interactions between 2 different attributes\n
            \t\t5. Correlation Heatmap and Table of the entire dataset\n
            \t\t6. Missing value count, matrix and heatmap\n
            \t\t7. A Sample of first and last rows\n''' )
add_vertical_space(2)

# Upload CSV data
with st.sidebar.header('1. Upload your CSV data'):
    uploaded_file = st.sidebar.file_uploader("Upload your input CSV file", type=["csv"])
    st.sidebar.markdown("""
[Example CSV input
file](https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubilit
y_with_descriptors.csv)
""")

# Pandas Profiling Report
if uploaded_file is not None:
    @st.cache_data
    def load_csv():
        csv = pd.read_csv(uploaded_file)
        return csv
    df = load_csv()
    pr = ProfileReport(df, explorative=True)
    st.header('**Input DataFrame**')
    st.write(df)
    st.write('---')
    st.header('**Dataset Profiling Report**')
    st_profile_report(pr)
else:
    st.info('Awaiting for CSV file to be uploaded.')
    add_vertical_space(2)
    if st.button('Press to use our Example "US Accidents" Dataset...'):
        # Example data
        @st.cache_data
        def load_data():
            a = pd.read_csv("US_Accidents_1000.csv")
            return a
```

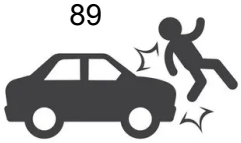
```
    df = load_data()
    pr = ProfileReport(df, explorative=True)
    st.header('**Input DataFrame**')
    st.write(df)
    st.write('---')
    st.header('**Dataset Profiling Report**')
    st_profile_report(pr)
```

Visualization of Data quality:

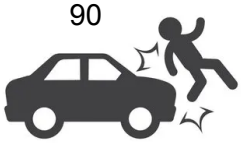
```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np

warnings.filterwarnings('ignore')


st.set_page_config(initial_sidebar_state="collapsed",page_title= " Data Quality
Visualisation ",
    menu_items={
     'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
     'Report a bug': "mailto:a.k.mirsha9@gmail.com",
     'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
  },page_icon="analysis.png",layout="wide")
```

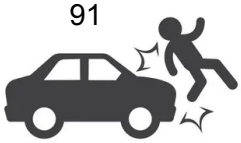
```
st.markdown('''
# **The Data Quality Visualisation Window**''')
add_vertical_space(2)
st.markdown(''':smile: This Window is specifically generated to create the data quality
visual report in terms of identifying missing values, irregular cardinality and outliers
:balloon: ''' )

add_vertical_space(3)

c1,c2,c3 = st.columns(3)
with c2:
    fl = st.file_uploader(":file_folder: Upload your file",type=(["csv","txt","xlsx","xls"]))
if fl is not None:
    filename = fl.name
    st.write(filename)
    df = pd.read_csv(filename, encoding = "ISO-8859-1")
else:
    df = pd.read_csv("US_Accidents_1000.csv", encoding = "ISO-8859-1")

add_vertical_space(3)

st.markdown("## Missing Values")

col1,col2 = st.columns([0.75,0.25])



with col1:
    st.markdown('#### Missing Values Heatmap')
    miss_heatmap = px.imshow(df.isnull(),title="Missing Values
Heatmap",labels={'x':'Attributes','y':'Frquency counts'})

    st.plotly_chart(miss_heatmap,use_container_width=True)

with col2:
    st.markdown("#### Details")
    with st.expander("Missing Values by Attribute",expanded=True):
        st.write(pd.DataFrame({
            'Attribute Name': [x for x in df.columns],
            'Missing Values': [df[x].isnull().sum() for x in df.columns]
        }))
```

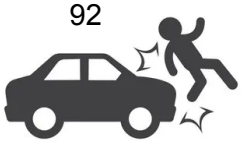
```
add_vertical_space(5)

miss1,miss2 = st.columns([0.5,0.5])

with miss1:
    st.markdown("#### Missing Values Pie Chart")
    add_vertical_space(1)
    num = st.select_slider("Select the number of attributes to Pie Chart missing values
for", options=[1,2,3,4,5,6,7,8,9,10],value=4)
    missing_values_counts = df.isnull().sum().sort_values(ascending=False)
    top_missing_values = missing_values_counts.head(num)
    misspie = px.pie(values=top_missing_values, names=top_missing_values.index,
title='Top {} Missing Values Attributes'.format(num))
    misspie.update_traces(marker=dict(colors=px.colors.qualitative.Set1))
    st.plotly_chart(misspie,use_container_width=True)
    st.markdown(" **Frequency reflecting Pie Chart for missing values** ")

with miss2:
    st.markdown("#### Missing Values Bar Chart")
    add_vertical_space(1)
    num = st.select_slider("Select the number of attributes to Bar Chart missing values
for", options=[1,2,3,4,5,6,7,8,9,10],value=8)
    missing_values_counts = df.isnull().sum().sort_values(ascending=False)
    top_missing_values = missing_values_counts.head(num)
    missbar = px.bar(x=top_missing_values.index, y=top_missing_values.values,
title='Top {} Missing Values
Attributes'.format(num),labels={'x':"Attributes","y":"Frequency counts"})
    missbar.update_traces(marker=dict(color='rgb(158,202,225)',
line=dict(color='rgb(8,48,107)', width=1.5)), selector=dict(type='bar'))
    st.plotly_chart(missbar,use_container_width=True)



add_vertical_space(5)
st.markdown("## Irregular Cardinality")

irr1, irr2 = st.columns([0.5, 0.5])

with irr1:
```

```
    st.markdown("#### Irregular Cardinality Pie Chart")
    add_vertical_space(1)
    num_pie = st.select_slider("Select the number of attributes for Pie Chart",
options=list(range(1, len(df.columns) + 1)), value=8)
    cardinality = df.apply(lambda x: len(x.unique()), axis=0).sort_values(ascending=False)
    top_cardinality_pie = cardinality.head(num_pie)
    cardpie = px.pie(values=top_cardinality_pie, names=top_cardinality_pie.index,
title=f'Top {num_pie} Irregular Cardinality Attributes')
    cardpie.update_traces(marker=dict(colors=px.colors.qualitative.Set1),
textinfo='label+percent', textposition='inside')
    cardpie.update_layout(
        legend_title="Attributes")
    st.plotly_chart(cardpie, use_container_width=True)
    st.markdown("**Frequency reflecting Pie Chart for irregular cardinality**")

with irr2:
    st.markdown("#### Irregular Cardinality Bar Chart")
    add_vertical_space(1)
    num_bar = st.select_slider("Select the number of attributes for Bar Chart",
options=list(range(1, len(df.columns) + 1)), value=23)
    cardinality = df.apply(lambda x: len(x.unique()), axis=0).sort_values(ascending=False)
    top_cardinality_bar = cardinality.head(num_bar)
    cardbar = px.bar(x=top_cardinality_bar.index, y=top_cardinality_bar.values,
title=f'Top {num_bar} Irregular Cardinality Attributes', labels={'x': "Attributes", "y":
"Cardinality"})
    cardbar.update_traces(marker=dict(color='rgb(158,202,225)',
line=dict(color='rgb(8,48,107)', width=1.5)), selector=dict(type='bar'))
    st.plotly_chart(cardbar, use_container_width=True)


add_vertical_space(5)

st.markdown("## Outliers Detection and Visualisation")
add_vertical_space(2)

out1,out2 = st.columns([0.4,0.6])

with out1:
    st.markdown("##### Box Plot for Selected Attribute")
    attribute= st.selectbox("Select your desired attribute" , options=
df.select_dtypes(include=['int', 'float']).columns,index=6)
```

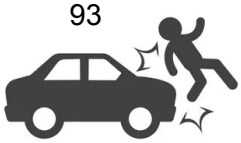
```
    # Select numerical columns

    trace = go.Box(y=df[attribute], name=attribute)
    layout = go.Layout(title="Boxplot of {}".format(attribute),
                       xaxis=dict(title="{}".format(attribute)),
                       yaxis=dict(title="{}".format(attribute)))

    # Create figure
    fig = go.Figure(data=trace, layout=layout)
    st.plotly_chart(fig,use_container_width=True)


with out2:
    st.markdown("##### Violin Plot for Selected Attribute")
    attribute2= st.selectbox("Select your desired attribute" , options=
df.select_dtypes(include=['int', 'float']).columns,index=6,key=2)
    # Select numerical columns

    fig = px.violin(df, x=attribute2,
                    title=f'Violin Plot for {attribute2}',
                    labels={'X': attribute2})

    # Update scatter plot style
    fig.update_traces(marker=dict(color='rgb(158, 202, 225)', size=10))
    st.plotly_chart(fig,use_container_width=True)


add_vertical_space(5)
st.markdown("#### The Overall Box Plot Visual for all the numerical attibutes")

numerical_columns = df.select_dtypes(include=['int', 'float']).columns

# Create traces for each numerical column
traces = []
for column in numerical_columns:
    trace = go.Box(y=df[column], name=column)
    traces.append(trace)

# Create layout
layout = go.Layout(title="Boxplot of All the Numerical Variables",
                   xaxis=dict(title="Attributes"),
```

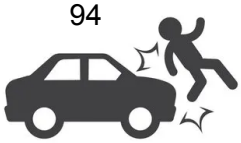
```
        yaxis=dict(title="Range Values"))

# Create figure
fig = go.Figure(data=traces, layout=layout)
st.plotly_chart(fig,use_container_width=True)
```

Feature engineering and correlation:

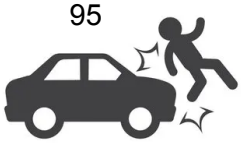
```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
from mlxtend.plotting import plot_pca_correlation_graph
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff
```

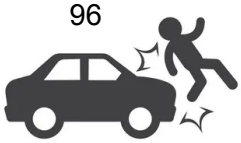
```
warnings.filterwarnings('ignore')



st.set_page_config(initial_sidebar_state="collapsed",page_title= " Feature
Engineering and Data Correlation Analysis ",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png",layout="wide")



st.markdown("""# **The Feature Engineering and Data Correlation Analysis Window**""")
add_vertical_space(2)
st.markdown(""":smile: This Window is specifically generated to carry out Feature
Engineering and Feature Identification for predicting a target feature by visualizing
hidden relationships and underlying complicated patterns. :balloon: """ )

add_vertical_space(3)

c1,c2,c3 = st.columns(3)
with c2:
   fl = st.file_uploader(":file_folder: Upload your file",type=(["csv","txt","xlsx","xls"]))
if fl is not None:
   filename = fl.name
   st.write(filename)
   df = pd.read_csv(filename, encoding = "ISO-8859-1")
else:
   df = pd.read_csv("US_Norm.csv", encoding = "ISO-8859-1")

add_vertical_space(3)

st.markdown("## Correlation Tests Analysis")

numerical_features = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
```

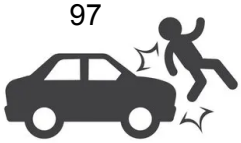
```
# Calculate correlation coefficients for numerical features
correlation_matrix = df[numerical_features].corr()
correlation_with_target = correlation_matrix['Severity'].sort_values(ascending=False)

# Transpose the Series to display horizontally
correlation_with_target_transposed = correlation_with_target.to_frame().T

st.write("Correlation with target (numerical features):")
st.write(correlation_with_target_transposed)



metric_cards_per_column = 10  # Number of metric cards to display per column

# Display metric cards in side-by-side columns
st.write("Correlation with target (numerical features):")

num_features = len(correlation_with_target_transposed.columns)
num_columns = num_features // metric_cards_per_column + (num_features %
metric_cards_per_column > 0)

columns = st.columns(num_columns)

card_counter = 0
summ = 0
for item in correlation_with_target:
    if not pd.isnull(item) and item != 0:
        print("Correlation score:", item)
        summ += abs(item)



for i in range(num_columns):
    with columns[i]:
        for j in range(metric_cards_per_column):
            if card_counter >= num_features:
                break
            feature = correlation_with_target_transposed.columns[card_counter]
            correlation_score = correlation_with_target_transposed.iloc[0, card_counter]
            delta = correlation_score - 1 if feature == "Severity" else correlation_score
```

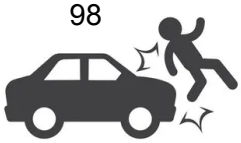
```
            st.metric(label=feature, value="{:.2f}".format(correlation_score),
delta="{:.2f}".format(delta/summ),)
            card_counter += 1


add_vertical_space(5)
st.markdown("### Chi Sqaure Tests for Categorical Data")
add_vertical_space(1)
# Perform Chi-square test for categorical features
categorical_features = ['Source',
 'Description',
 'Street',
 'City',
 'County',
 'State',
 'Zipcode',
 'Country',
 'Timezone',
 'Airport_Code',
 'Wind_Direction',
 'Weather_Condition',
 'Sunrise_Sunset',
 'Civil_Twilight',
 'Nautical_Twilight',
 'Astronomical_Twilight',
 'Severity']

chi2_results = chi2(df[categorical_features], df['Severity'])
categorical_scores = pd.Series(chi2_results[0], index=categorical_features)


x=categorical_scores.sort_values(ascending=False)

st.write(x.to_frame().T)

col1,col2 = st.columns([0.5,0.5])


with col1:
   st.markdown('#### Chi Sqaure Test Bar Chart')
   sorted_categorical_scores = categorical_scores.sort_values(ascending=False)
```

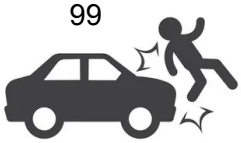
```
    # Define the top n attributes
    top_n = st.select_slider("Select the number of Attributes to visualise the Chi Square Bar results for", list(range(1, len(sorted_categorical_scores) + 1)),value = 9)  # You can change this value as needed

    # Select the top n attributes
    top_n_attributes = sorted_categorical_scores.head(top_n)

    # Create a bar chart for the top n attributes
    bar_chart = px.bar(x=top_n_attributes.index, y=top_n_attributes.values, labels={'x': 'Attribute', 'y': 'Chi-square Score'},
                       title=f'Top {top_n} Categorical Attributes and Their Chi-square Scores')
    bar_chart.update_layout(xaxis_tickangle=-45)
    st.plotly_chart(bar_chart,use_container_width=True)


with col2:
    st.markdown("#### Chi Square Test Pie Chart")

    # Define the top n attributes
    top_n = st.select_slider("Select the number of Attributes to visualise the Chi Square Pie results for", list(range(1, len(sorted_categorical_scores) + 1)),value = 4,key=3)  # You can change this value as needed

    # Select the top n attributes
    top_n_attributes = sorted_categorical_scores.head(top_n)
    # Create a pie chart to visualize the distribution of chi-square scores among the top n attributes
    pie_chart = px.pie(values=top_n_attributes.values, names=top_n_attributes.index,
                       title=f'Distribution of Chi-square Scores Among Top {top_n} Attributes')
    st.plotly_chart(pie_chart,use_container_width=True)

add_vertical_space(5)

st.markdown("### Select K Best Test for Numerical Attributes")
add_vertical_space(3)

c1,c2,c3 = st.columns(3)
with c2:
    k = st.number_input(label="Select the 'K' Value",min_value=3,max_value=10,value=5)
```

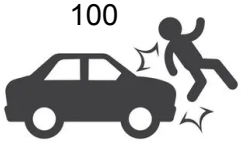
```
add_vertical_space(2)
# Perform SelectKBest test for numerical attributes using mutual information
select_k_best = SelectKBest(score_func=mutual_info_classif, k=k)  # You can change
the value of k as needed
select_k_best.fit(df[numerical_features], df['Severity'])

# Get the scores and corresponding feature names
numerical_scores = pd.Series(select_k_best.scores_, index=numerical_features)

# Sort the numerical scores in descending order
sorted_numerical_scores = numerical_scores.sort_values(ascending=False)

st.write(sorted_numerical_scores.to_frame().T)

miss1,miss2 = st.columns([0.5,0.5])

with miss1:
   st.markdown("#### Select K Best Bar Chart")
   add_vertical_space(1)
   num = st.select_slider("Select the number of attributes to Bar Chart",
options=list(range(1, len(sorted_numerical_scores) + 1)),value=23)
   bar_chart_numerical = px.bar(x=sorted_numerical_scores.index[:num],
y=sorted_numerical_scores.values[:num],
                        labels={'x': 'Attribute', 'y': 'Score'},
                        title='Top {} Numerical Attributes and Their SelectKBest
Scores'.format(num))
   bar_chart_numerical.update_layout(xaxis_tickangle=-45)
   st.plotly_chart(bar_chart_numerical, use_container_width=True)

with miss2:
   st.markdown("#### Select K BestPie Chart")
   add_vertical_space(1)
   num = st.select_slider("Select the number of attributes to Pie Chart ",
options=list(range(1, len(sorted_numerical_scores) + 1)),value=7)
   pie_chart_numerical = px.pie(values=sorted_numerical_scores.values[:num],
names=sorted_numerical_scores.index[:num],
                         title='Distribution of SelectKBest Scores Among Top {} Numerical
Attributes'.format(num))
   st.plotly_chart(pie_chart_numerical, use_container_width=True)
```

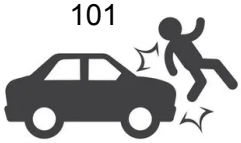
```
add_vertical_space(5)
st.markdown("## Principal Component Analysis")

irr1, irr2 = st.columns([0.5, 0.5])

with irr1:
    st.markdown("#### PCA Correlation Circle Graph")
    add_vertical_space(1)

    float_columns = df.select_dtypes(include=['float']).columns

    # Select only columns with floating-point data
    float_columns = df.select_dtypes(include=['float']).columns
    # Extract features and target variable
    X = df[float_columns] # Features
    y = df['Severity'] # Target
    # Standardize the features

    scaling = st.checkbox("Whether you want to scale the data before PCA ?",value =
True)
    if scaling:
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
    else:
        X_scaled = X

    num1_bar = st.select_slider("Select the number of attributes for the Principal
Component Analysis", options=list(range(1, len(X.columns) + 1)), value=8)

    # Perform PCA
    pca = PCA(n_components=num1_bar) # Specify the number of components to keep
    X_pca = pca.fit_transform(X_scaled)


    # Get the explained variance ratio
    explained_variance_ratio = pca.explained_variance_ratio_

    # Calculate the correlation between original features and principal components
    correlation_matrix = pd.DataFrame(pca.components_, columns=float_columns)
```

```
    correlation_matrix.index = ['PC' + str(i) for i in range(1, len(correlation_matrix) + 1)]



    fig, ax = plot_pca_correlation_graph(X, float_columns,
                                         dimensions=(1, 2))

    # Display the plot using Streamlit
    st.pyplot(fig,use_container_width=True)

import plotly.graph_objects as go

with irr2:
    st.markdown("#### PCA analysis Bar Chart")
    add_vertical_space(1)
    num_bar = st.select_slider("Select the number of attributes for Bar Chart",
options=list(range(1, len(X.columns) + 1)), value=8)

    #pl=plt.bar(range(len(pca.explained_variance_ratio_)),
pca.explained_variance_ratio_, color='skyblue')
    fig = px.bar(x=range(len(pca.explained_variance_ratio_[:num_bar])),
y=pca.explained_variance_ratio_[:num_bar], labels={'x': 'Principal Components', 'y':
'Magnitude Contribution of variance'},
                 title=f'Top {top_n} Principal Component Analysis Variance Ratio
Distribution')
    fig.update_layout(xaxis_tickangle=-45)
    st.plotly_chart(fig, use_container_width=True)

add_vertical_space(5)

st.markdown("## Correlation Heatmap")
add_vertical_space(2)


numerical = df.select_dtypes(include=['float','int']).columns
important =
['Start_Lat','End_Lat','Start_Lng','End_Lng','Temperature(F)','Distance(mi)','Wind_Chill(
F)', 'Humidity(%)', 'Wind_Speed(mph)', 'Pressure(in)']


df = pd.read_csv('US_Norm.csv')
```

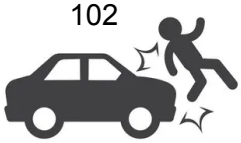



```
c1,c2,c3 = st.columns(3)
with c2:
    colorscales = px.colors.named_colorscales()
    palette = st.selectbox("Select a color palette : ",options= colorscales,index=31)
add_vertical_space(2)


c1,c2 = st.columns(2)

# Adding a color palette selection box
st.markdown("### View the Range of Sequential Color Palettes to choose from :")
add_vertical_space(2)

fig = px.colors.sequential.swatches_continuous()
st.plotly_chart(fig, use_container_width=True)

add_vertical_space(3)
color_palettes = [
    "Plotly3", "Viridis", "Cividis", "Inferno", "Magma", "Plasma", "Turbo", "Blackbody",
"Bluered",
    "Electric", "Hot", "Jet", "Rainbow", "Blues", "BuGn", "GnBu", "Greens", "Greys", "OrRd",
    "Oranges", "PuBu", "PuBuGn", "PuRd", "Purples", "RdBu", "RdPu", "Reds", "YlGn",
"YlGnBu",
    "YlOrBr", "YlOrRd", "algae", "amp", "deep", "dense", "gray", "haline", "ice", "matter",
    "solar", "speed", "tempo", "thermal", "turbid", "Burg", "Burgyl", "Redor", "Oryel",
"Peach",
    "Pinkyl", "Mint", "Bluyl", "Darkmint", "Emrld", "aggrnyl", "BluGrn", "Teal", "Tealgrn",
"Purp",
    "Purpor", "Sunset", "Magenta", "Sunsetdark", "Agsunset", "Brwnyl"
]

with c1:
    st.markdown('#### Correlation Heatmap of Numerical Attributes')
    fig = px.imshow(df[numerical].corr(),color_continuous_scale=palette)
    st.plotly_chart(fig, use_container_width=True)

with c2:
    st.markdown('#### Correlation Heatmap of The Most Important Features')
    fig = px.imshow(df[important].corr(),text_auto=True,color_continuous_scale=palette)
    st.plotly_chart(fig, use_container_width=True)
```

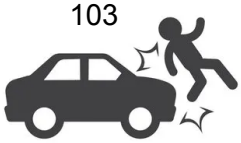



Data preprocessing and preparation:

```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
from mlxtend.plotting import plot_pca_correlation_graph
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff
from scipy import stats
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

warnings.filterwarnings('ignore')

st.set_page_config(
```

```
    initial_sidebar_state="collapsed",
    page_title="Data Pre-Processing and Preparation for Exploration",
    menu_items={
        'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
        'Report a bug': "mailto:a.k.mirsha9@gmail.com",
        'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the 'US Accidents Dataset'\n"
    },
    page_icon="analysis.png",
    layout="wide"
)

st.markdown('# **The Data Preprocessing and Preparation Window**')
add_vertical_space(2)
st.markdown('##### :smile: This Window is specifically generated to carry out Data
Preparation and Preprocessing Techniques such as Normalisation, Binning, and
Sampling for Further Exploration. :balloon:')

add_vertical_space(3)

c1, c2, c3 = st.columns(3)
with c2:
    fl = st.file_uploader(":file_folder: Upload your file", type=(["csv", "txt", "xlsx", "xls"]))
if fl is not None:
    filename = fl.name
    st.write(filename)
    df = pd.read_csv(filename, encoding="ISO-8859-1")
else:
    df = pd.read_csv("US_Accidents_1000.csv", encoding="ISO-8859-1")

add_vertical_space(3)

st.markdown('## Data Normalisation Process')
add_vertical_space(2)
st.markdown("""**Feature Scaling is an essential step in the data analysis and
preparation of data for modeling. Wherein, we make the data scale-free for easy
analysis.**
```

Normalization is one of the feature scaling techniques. We particularly apply normalization when the data is skewed on either axis i.e. when the data does not follow the Gaussian distribution."")

add_vertical_space(5)

numerical_columns = df.select_dtypes(include=['int', 'float']).columns
print(numerical_columns)

data = df.dropna()

c1, c2 = st.columns(2)

with c1:
    st.markdown('##### Min-Max Normalisation')
    st.write("**The Normalised Data strictly lies between 0 to 1**")
    add_vertical_space(2)

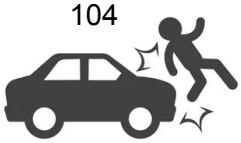
    attribute = st.selectbox("Select the Attribute to visualise the Min-Max Normalisation for", list(numerical_columns), index=6)

    # Normalize data
    scaler = MinMaxScaler()
    data_normalized = scaler.fit_transform(data[[attribute]])
    add_vertical_space(3)

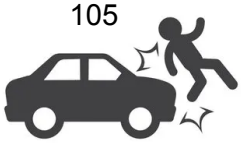
    st.markdown("##### Visual inspection")
    add_vertical_space(1)

Displaying the plots in Streamlit
    fig = px.histogram(data[attribute], title='Original Data', color_discrete_sequence=px.colors.sequential.Plotly3)
    wig = px.histogram(data_normalized.flatten(), title='Normalised Data', color_discrete_sequence=px.colors.sequential.Plotly3)
    st.plotly_chart(fig, use_container_width=True)
    st.plotly_chart(wig, use_container_width=True)

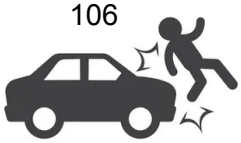
    add_vertical_space(3)



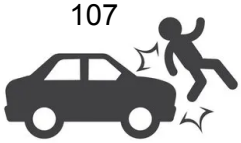
```
    st.markdown("##### Statistical test for normality")
    add_vertical_space(1)
    st.write(stats.shapiro(data_normalized))

    add_vertical_space(3)

    st.markdown("##### Evaluate model performance")
    model = RandomForestClassifier()
    scores_original = cross_val_score(model, data[[attribute]], data['Severity'], cv=2)
    scores_normalized = cross_val_score(model, data_normalized, data['Severity'],
cv=2)

    st.write("Original Data Scores:", scores_original.mean())
    st.write("Normalized Data Scores:", scores_normalized.mean())

with c2:
    st.markdown('##### Standard Normalisation')
    st.write("**The Normalised Data strictly has the Mean of 0 and Standard Deviation of
approximately 1**")
    add_vertical_space(2)

    attribute = st.selectbox("Select the Attribute to visualise the Standardisation for",
list(numerical_columns), index=6)

    # Normalize data
    scaler = StandardScaler()
    data_normalized = scaler.fit_transform(data[[attribute]])
    add_vertical_space(3)

    st.markdown("##### Visual inspection")
    add_vertical_space(1)

    # Displaying the plots in Streamlit
    fig = px.histogram(data[attribute], title='Original Data',
color_discrete_sequence=px.colors.sequential.Plotly3)
    wig = px.histogram(data_normalized.flatten(), title='Normalised Data',
color_discrete_sequence=px.colors.sequential.Plotly3)
    st.plotly_chart(fig, use_container_width=True)
    st.plotly_chart(wig, use_container_width=True)

    add_vertical_space(3)
```

```
    st.markdown("##### Statistical test for normality")
    add_vertical_space(1)
    st.write(stats.shapiro(data_normalized))

    add_vertical_space(3)

    st.markdown("##### Evaluate model performance")
    model = RandomForestClassifier()
    scores_original = cross_val_score(model, data[[attribute]], data['Severity'], cv=5)
    scores_normalized = cross_val_score(model, data_normalized, data['Severity'],
cv=5)

    st.write("Original Data Scores:", scores_original.mean())
    st.write("Normalized Data Scores:", scores_normalized.mean())

add_vertical_space(5)
st.markdown('### Statistical Method Normalisation Verification')
w_statistic, p_value = stats.shapiro(data_normalized)
st.write("Normal Data - W statistic:", w_statistic, "P-value:", p_value)
# Test for non-normal data
w_statistic_non_normal, p_value_non_normal = stats.shapiro(data['Temperature(F)'])
st.write("Non-Normal Data - W statistic:", w_statistic_non_normal, "P-value:",
p_value_non_normal)

add_vertical_space(3)

st.write(""**Interpreting the Results**

**● W statistic: Closer to 1 indicates data is more likely to be normal.**

**● P-value: A p-value greater than the chosen alpha level (commonly set at 0.05)
suggests that the null hypothesis of normality cannot be rejected. A small p-value
(typically ≤ 0.05) indicates strong evidence against the null hypothesis, so you reject the
null hypothesis of normality**"")

add_vertical_space(5)
st.markdown("### Quantile-Quantile Plot")

add_vertical_space(3)
```

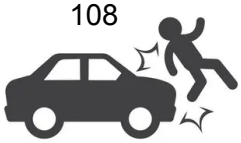
```
# Create Q-Q plots
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Q-Q plot for non-normalized data
stats.probplot(data['Temperature(F)'], dist="norm", plot=axes[0])
axes[0].set_title('Q-Q Plot for Non-Normalized Data')

# Q-Q plot for normalized data
stats.probplot(data_normalized.flatten(), dist="norm", plot=axes[1])
axes[1].set_title('Q-Q Plot for Normalized Data')

# Displaying the plots in Streamlit
st.pyplot(fig)
add_vertical_space(5)

st.markdown("### Interactive Binning")
add_vertical_space(3)

# Function to perform binning
def perform_binning(data, column_name, labels, value_ranges):
    bins = pd.cut(data[column_name], bins=value_ranges, labels=labels)
    data['{}_Bin'.format(column_name)] = bins
    return data

def plot_binned_column(data, column_name, title):
    bar = px.bar(x=data[column_name].value_counts().index,
y=data[column_name].value_counts().values,
                 color_discrete_sequence=px.colors.sequential.Plotly3)
    st.plotly_chart(bar)

st.subheader('Binning Options')
add_vertical_space(2)
bin_col_name = st.selectbox("Select the Numerical Attribute to visualise the Binning
for", list(numerical_columns), index=6)
add_vertical_space(1)
if bin_col_name in df.columns:
    labels_input = st.text_input("Enter bin labels (comma-separated):", value='very
cold,cold,moderate,warm,hot')
    add_vertical_space(1)
    labels = [label.strip() for label in labels_input.split(',')]
```

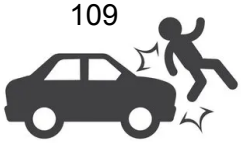
```
    value_ranges_input = st.text_input("Enter value ranges for bins
(comma-separated):", value='-100,32,50,70,90,200')
    value_ranges = [float(val.strip()) for val in value_ranges_input.split(',')]
    add_vertical_space(2)
    if st.button('Perform Binning'):
        data = perform_binning(df, bin_col_name, labels, value_ranges)
        add_vertical_space(1)
        st.write('Data after binning:')
        add_vertical_space(1)
        st.write(data.head())
        add_vertical_space(2)
        st.write('Distribution of Bins:')
        add_vertical_space(1)
        plot_binned_column(data, '{}_Bin'.format(bin_col_name), 'Distribution of {}
Bins'.format(bin_col_name))
```

Comparison of classifier model:

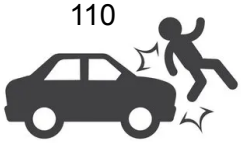
```
# App created by Data Professor http://youtube.com/dataprofessor
# GitHub repo of this app https://github.com/dataprofessor/ml-auto-app
# Demo of this app https://share.streamlit.io/dataprofessor/ml-auto-app/main/app.py

import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
import pandas as pd
from lazypredict.Supervised import LazyRegressor
from lazypredict.Supervised import LazyClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_diabetes #load_boston
import matplotlib.pyplot as plt
import seaborn as sns
import base64
import io

#---------------------------------#
```

```
# Page layout
## Page expands to full width
st.set_page_config(initial_sidebar_state="collapsed",page_title='The Machine
Learning Classification Algorithms Comparison Window',
    layout='wide',
    menu_items={
        'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
        'Report a bug': "mailto:a.k.mirsha9@gmail.com",
        'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },
    page_icon="classification.png")
add_vertical_space(2)
#---------------------------------#
# Model building
def build_model(df):

    X = data.drop(columns=['Severity','Start_Time'])  # Features
    Y = data['Severity']  # Target variable

    st.markdown('**1.2. Dataset dimension**')
    st.write('X')
    st.info(X.shape)
    st.write('Y')
    st.info(Y.shape)

    st.markdown('**1.3. Variable details**:')
    st.write('Predictor variables')
    st.info(list(X.columns))
    st.write('Target variable')
    st.info(Y.name)

    # Build lazy model
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size =
split_size,random_state = seed_number)
    clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
    models_train,predictions_train = clf.fit(X_train, X_train, Y_train, Y_train)
    models_test,predictions_test = clf.fit(X_train, X_test, Y_train, Y_test)
```

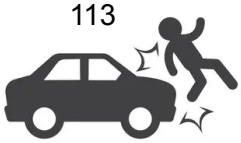
```
    st.subheader('2. Table of Model Performance')

    st.write('Training set')
    st.write(predictions_train[['Accuracy', 'Balanced Accuracy', 'F1 Score', 'Time Taken']])
    st.markdown(filedownload(predictions_train,'training.csv'), unsafe_allow_html=True)

    st.write('Test set')
    st.write(predictions_test[['Accuracy', 'Balanced Accuracy', 'F1 Score', 'Time Taken']])
    st.markdown(filedownload(predictions_test,'test.csv'), unsafe_allow_html=True)

    st.subheader('3. Plot of Model Performance (Test set)')


    with st.markdown('**Accuracy**'):
        # Tall
        predictions_test["Accuracy"] = [0 if i < 0 else i for i in predictions_test["Accuracy"]
]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="whitegrid")
        ax1 = sns.barplot(y=predictions_test.index, x="Accuracy", data=predictions_test)
        ax1.set(xlim=(0, 1))
    st.markdown(imagedownload(plt,'plot-acc-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(9, 3))
    sns.set_theme(style="whitegrid")
    ax1 = sns.barplot(x=predictions_test.index, y="Accuracy", data=predictions_test)
    ax1.set(ylim=(0, 1))
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-acc-wide.pdf'), unsafe_allow_html=True)

    with st.markdown('**Balanced Accuracy**'):
        # Tall
        predictions_test["Balanced Accuracy"] = [0 if i < 0 else i for i in
predictions_test["Balanced Accuracy"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="whitegrid")
        ax1 = sns.barplot(y=predictions_test.index, x="Balanced Accuracy",
data=predictions_test)
        ax1.set(xlim=(0, 1))
```

```
    st.markdown(imagedownload(plt,'plot-bal-acc-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(9, 3))
    sns.set_theme(style="whitegrid")
    ax1 = sns.barplot(x=predictions_test.index, y="Balanced Accuracy",
data=predictions_test)
    ax1.set(ylim=(0, 1))
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-bal-acc-wide.pdf'), unsafe_allow_html=True)


    with st.markdown('**F1 Score**'):
        # Tall
        predictions_test["F1 Score"] = [0 if i < 0 else i for i in predictions_test["F1 Score"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="whitegrid")
        ax1 = sns.barplot(y=predictions_test.index, x="F1 Score", data=predictions_test)
        ax1.set(xlim=(0, 1))
    st.markdown(imagedownload(plt,'plot-f1-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(9, 3))
    sns.set_theme(style="whitegrid")
    ax1 = sns.barplot(x=predictions_test.index, y="F1 Score", data=predictions_test)
    ax1.set(ylim=(0, 1))
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-f1-wide.pdf'), unsafe_allow_html=True)



    with st.markdown('**Calculation time**'):
        # Tall
        predictions_test["Time Taken"] = [0 if i < 0 else i for i in predictions_test["Time
Taken"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="whitegrid")
        ax3 = sns.barplot(y=predictions_test.index, x="Time Taken", data=predictions_test)
    st.markdown(imagedownload(plt,'plot-calculation-time-tall.pdf'),
unsafe_allow_html=True)
        # Wide
```

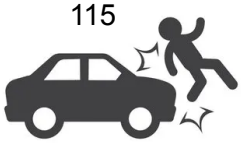

```
    plt.figure(figsize=(9, 3))
    sns.set_theme(style="whitegrid")
    ax3 = sns.barplot(x=predictions_test.index, y="Time Taken", data=predictions_test)
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-calculation-time-wide.pdf'),
unsafe_allow_html=True)

# Download CSV data
# https://discuss.streamlit.io/t/how-to-download-file-in-streamlit/1806
def filedownload(df, filename):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()  # strings <-> bytes conversions
    href = f'<a href="data:file/csv;base64,{b64}" download={filename}>Download
{filename} File</a>'
    return href

def imagedownload(plt, filename):
    s = io.BytesIO()
    plt.savefig(s, format='pdf', bbox_inches='tight')
    plt.close()
    b64 = base64.b64encode(s.getvalue()).decode()  # strings <-> bytes conversions
    href = f'<a href="{b64}" download={filename}>Download
{filename} File</a>'
    return href

#---------------------------------#
st.write("""
# The Machine Learning Algorithm Comparison Window""")

add_vertical_space(1)
st.write("""


In this Page Window, the **lazypredict** library is used for building and comparing
several machine learning models at once.


""")

add_vertical_space(2)
```


```
#---------------------------------#
# Sidebar - Collects user input features into dataframe
with st.sidebar.header('1. Upload your CSV data'):
    uploaded_file = st.sidebar.file_uploader("Upload your input CSV file", type=["csv"])
    st.sidebar.markdown("""
[Example CSV input
file](https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubilit
y_with_descriptors.csv)
""")

# Sidebar - Specify parameter settings
with st.sidebar.header('2. Set Parameters'):
    split_size = st.sidebar.slider('Data split ratio (% for Training Set)', 10, 90, 80, 5)
    seed_number = st.sidebar.slider('Set the random seed number', 1, 100, 42, 1)


#---------------------------------#
# Main panel

# Displays the dataset
st.subheader('1. Dataset')
add_vertical_space(1)

if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)
    st.markdown('**1.1. Glimpse of dataset**')
    st.write(df)
    x = True
else:
    st.info('Awaiting for CSV file to be uploaded.')
    add_vertical_space(1)
    if st.button('Press to use our Example "US Accidents Dataset"...'):
        data = pd.read_csv("US_Norm.csv")

        x = True


import pandas as pd
from sklearn.model_selection import train_test_split
```

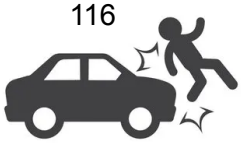



```
from sklearn.ensemble import (RandomForestClassifier, ExtraTreesClassifier,
BaggingClassifier,
                              AdaBoostClassifier)
from sklearn.svm import SVC, NuSVC, LinearSVC
from lightgbm import LGBMClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.discriminant_analysis import (QuadraticDiscriminantAnalysis,
LinearDiscriminantAnalysis)
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from sklearn.linear_model import (LogisticRegression, RidgeClassifierCV,
RidgeClassifier,
                                  PassiveAggressiveClassifier, SGDClassifier, Perceptron)
from sklearn.calibration import CalibratedClassifierCV
from sklearn.semi_supervised import LabelPropagation, LabelSpreading
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report


xdata = pd.read_csv(r"C:\Users\DELL\Desktop\STREAMLIT\US.csv")
xdata = xdata.sample(n=1000,random_state=42)
# Load your dataset
data = xdata.drop(["ID","Source"],axis=1)

# Assume the last column is the target variable
X =
data.drop(columns=['Severity','Start_Time","End_Time","Description","Street","Zipcode"
,"Country","Timezone","Airport_Code","Weather_Timestamp","Amenity","Bump","Crossi
ng","Give_Way","Junction","No_Exit","Railway","Roundabout","Station","Stop","Traffic_
Calming","Traffic_Signal","Turning_Loop","Civil_Twilight","Nautical_Twilight","Astrono
mical_Twilight"])  # Features
y = data['Severity']  # Target variable

### Time for user prediction
add_vertical_space(2)
st.markdown("## The Predictions Making Space")
add_vertical_space(1)
```


```
st.markdown("###### Feel free to customise the dataset, applying what all attributes you want to filter for...")
add_vertical_space(3)

c1,c2 = st.columns(2)
with c1:
    attributes = X.columns
    default_attributes = ["Hour_Category","Temperature(F)","Wind_Chill(F)","Humidity(%)","Pressure(in)"]
    attributes = st.multiselect("Filter out your attributes for the predictions dataset here.",attributes,default_attributes)

with c2:
    classifier_names = [
    'RandomForestClassifier',
    'LGBMClassifier',
    'ExtraTreesClassifier',
    'SVC',
    'DecisionTreeClassifier',
    'QuadraticDiscriminantAnalysis',
    'BaggingClassifier',
    'ExtraTreeClassifier',
    'LabelPropagation',
    'LabelSpreading',
    'NuSVC',
    'LogisticRegression',
    'KNeighborsClassifier',
    'LinearSVC',
    'CalibratedClassifierCV',
    'LinearDiscriminantAnalysis',
    'RidgeClassifierCV',
    'RidgeClassifier',
    'AdaBoostClassifier',
    'PassiveAggressiveClassifier',
    'SGDClassifier',
    'Perceptron',
    'GaussianNB',
    'NearestCentroid',
    'BernoulliNB',
    'DummyClassifier']
```

```
    st.markdown("#### You can Choose from the top 5 Best Performing ML Models for this Dataset...")
    add_vertical_space(2)
    st.markdown("**Feel free to adjust all the Attributes and Data Predicting Models Properly for an enhanced testing experience. All the models listed here has achieved more than 97% Accuracy in predicting this Dataset...**")
    add_vertical_space(3)
    model_name = st.selectbox("Select the Model you want to predict upon...",classifier_names,0)


# Split the dataset into training and testing sets
X = data[attributes]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define preprocessing steps
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore', sparse=True)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create a dictionary of classifiers
classifiers = {
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100, random_state=42),
    'LGBMClassifier': LGBMClassifier(random_state=42),
    'ExtraTreesClassifier': ExtraTreesClassifier(n_estimators=100, random_state=42),
    'SVC': SVC(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'QuadraticDiscriminantAnalysis': QuadraticDiscriminantAnalysis(),
    'BaggingClassifier': BaggingClassifier(n_estimators=100, random_state=42),
    'ExtraTreeClassifier': ExtraTreeClassifier(random_state=42),
    'LabelPropagation': LabelPropagation(),
    'LabelSpreading': LabelSpreading(),
```

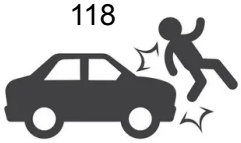



```
    'NuSVC': NuSVC(random_state=42),
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
    'KNeighborsClassifier': KNeighborsClassifier(),
    'LinearSVC': LinearSVC(random_state=42, max_iter=10000),
    'CalibratedClassifierCV': CalibratedClassifierCV(),
    'LinearDiscriminantAnalysis': LinearDiscriminantAnalysis(),
    'RidgeClassifierCV': RidgeClassifierCV(),
    'RidgeClassifier': RidgeClassifier(random_state=42),
    'AdaBoostClassifier': AdaBoostClassifier(n_estimators=100, random_state=42),
    'PassiveAggressiveClassifier': PassiveAggressiveClassifier(max_iter=1000,
random_state=42),
    'SGDClassifier': SGDClassifier(max_iter=1000, random_state=42),
    'Perceptron': Perceptron(max_iter=1000, random_state=42),
    'GaussianNB': GaussianNB(),
    'NearestCentroid': NearestCentroid(),
    'BernoulliNB': BernoulliNB(),
    'DummyClassifier': DummyClassifier(strategy='most_frequent')
}

if model_name in classifiers:
    classifier = classifiers[model_name]
else:
    raise ValueError(f"Unsupported model: {model_name}")

# Create and train the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', classifier)
])


# Train the model
pipeline.fit(X_train, y_train)

new_data = {}

with c1:
    for item in attributes:
        if data[item].dtype in ['int64','float64']:
            if item=='Pressure(in)':
```

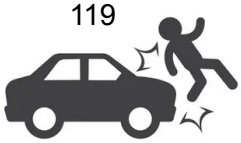

```
            new_data[item]=[st.number_input(f"Enter the value of {item} to be
predicted.",value=31)]
        else:
            new_data[item]=[st.number_input(f"Enter the value of {item} to be
predicted.",value=69)]
    elif data[item].dtype in ['bool','object']:
        new_data[item]=[st.text_input(f"Enter the value of {item} to be
predicted.",value="Evening")]

with c2:
  if st.button("Predict !!!"):
    new_data = pd.DataFrame(new_data)
    # Ensure the new data has the same preprocessing as the training data
    prediction = pipeline.predict(new_data)

    st.markdown('#### Prediction for the given sample data:')
    st.markdown(f"**The {model_name} has predicted the Target Attribute belonging
to class {prediction[0]}**")

add_vertical_space(3)
st.markdown("### Now Click Here to view The Overall Comparsion Report of **25 +**
Machine Learning Models...")
add_vertical_space(3)

if st.button("SHOW MODELS' COMPARISON REPORT"):
  add_vertical_space(3)
  data = pd.read_csv(r"C:\Users\DELL\Desktop\STREAMLIT\US_Norm.csv")
  data = data.drop(["Unnamed: 0","ID","Source","Description","Street"],axis=1)
  build_model(data)
```

Optimization for various parameters:

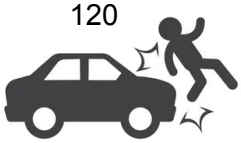
```
import streamlit as st
import pandas as pd
import numpy as np
import base64
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
```

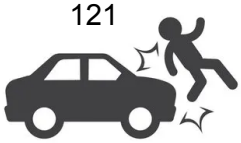
```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_diabetes
from streamlit_extras.add_vertical_space import add_vertical_space

#---------------------------------#
# Page layout
## Page expands to full width
st.set_page_config(initial_sidebar_state="collapsed",page_title='The Machine Learning Hyperparameter Optimization Window',
         menu_items={
        'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=drive_link',
        'Report a bug': "mailto:a.k.mirsha9@gmail.com",
        'About': "### This is an extremely cool web application built as a part of my Data Science Mini Project on my ' US Accidents Dataset '\n"
    },
   page_icon="image.png",
   layout='wide')

#---------------------------------#
#---------------------------------#
st.write("""
# The Machine Learning Hyperparameter Optimization Window
**(Classification Edition)**

In this implementation, the *DecisionTreeClassifier()* function is used in this app for building a classification model using the **Decision Tree** algorithm.""")

add_vertical_space(2)

st.write("""
Moreover all the hyper parameters that are essential to build the model are extracted as inputs from the user interactively so that users can fine tune their model's performance based on the chosen hyperparameter.
""")

#---------------------------------#
# Sidebar - Collects user input features into dataframe
st.sidebar.header('Upload your CSV data')
```

```
uploaded_file = st.sidebar.file_uploader("Upload your input CSV file", type=["csv"])
st.sidebar.markdown("""
[Example CSV input
file](https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubility_with_descriptors.csv)
""")

# Sidebar - Specify parameter settings
st.sidebar.header('Set Parameters')
split_size = st.sidebar.slider('Data split ratio (% for Training Set)', 10, 90, 80, 5)

st.sidebar.subheader('Learning Parameters')
parameter_n_estimators = st.sidebar.slider('Number of estimators (n_estimators)', 0, 500, (10,110), 10)
parameter_n_estimators_step = st.sidebar.number_input('Step size for n_estimators', 10)
st.sidebar.write('---')
parameter_max_features = st.sidebar.slider('Max features (max_features)', 1, 50, (5,12), 1)
st.sidebar.number_input('Step size for max_features', 1)
st.sidebar.write('---')
parameter_min_samples_split = st.sidebar.slider('Minimum number of samples required to split an internal node (min_samples_split)', 1, 10, 2, 1)
parameter_min_samples_leaf = st.sidebar.slider('Minimum number of samples required to be at a leaf node (min_samples_leaf)', 1, 10, 2, 1)

st.sidebar.subheader('General Parameters')
parameter_random_state = st.sidebar.slider('Seed number (random_state)', 0, 1000, 42, 1)
parameter_criterion = st.sidebar.select_slider('Performance measure (criterion)', options=['log_loss','entropy','gini'])
parameter_bootstrap = st.sidebar.select_slider('Bootstrap samples when building trees (bootstrap)', options=[True, False])
parameter_oob_score = st.sidebar.select_slider('Whether to use out-of-bag samples to estimate the R^2 on unseen data (oob_score)', options=[False, True])
parameter_n_jobs = st.sidebar.select_slider('Number of jobs to run in parallel (n_jobs)', options=[1, -1])
```

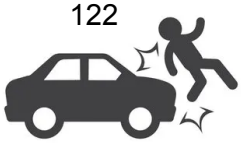



```
n_estimators_range = np.arange(parameter_n_estimators[0],
parameter_n_estimators[1]+parameter_n_estimators_step,
parameter_n_estimators_step)
max_features_range = np.arange(parameter_max_features[0],
parameter_max_features[1]+1, 1)
param_grid = dict(max_features=max_features_range,
n_estimators=n_estimators_range)

#---------------------------------#
# Main panel

# Displays the dataset
st.subheader('Dataset')



#---------------------------------#
# Model building

def filedownload(df):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()  # strings <-> bytes conversions
    href = f'<a href="data:file/csv;base64,{b64}"
download="model_performance.csv">Download CSV File</a>'
    return href

def build_model(df):
    X = df.drop(columns=['Severity','Start_Time'])  # Features
    Y = df['Severity']  # Target variable

    st.markdown('A model is being built to predict the following **Y** variable:')
    st.info(Y.name)

    # Data splitting
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=split_size)
    #X_train.shape, Y_train.shape
    #X_test.shape, Y_test.shape
    #RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0,
bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0,
```

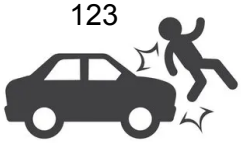



warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)[source]

```
    rf = RandomForestClassifier(n_estimators=parameter_n_estimators,
        random_state=parameter_random_state,
        max_features=parameter_max_features,
        criterion=parameter_criterion,
        min_samples_split=parameter_min_samples_split,
        min_samples_leaf=parameter_min_samples_leaf,
        bootstrap=parameter_bootstrap,
        oob_score=parameter_oob_score,
        n_jobs=parameter_n_jobs)

    grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)
    grid.fit(X_train, Y_train)

    st.subheader('Model Performance')

    Y_pred_test = grid.predict(X_test)
    st.write('Coefficient of determination ($R^2$):')
    st.info( r2_score(Y_test, Y_pred_test) )

    st.write('Error (MSE or MAE):')
    st.info( mean_squared_error(Y_test, Y_pred_test) )

    st.write("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

    st.subheader('Model Parameters')
    st.write(grid.get_params())

    #-----Process grid data-----#
    grid_results =
pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.cv_results_[
"mean_test_score"], columns=["R2"])],axis=1)
    # Segment data into groups based on the 2 hyperparameters
    grid_contour = grid_results.groupby(['max_features','n_estimators']).mean()
    # Pivoting the data
    grid_reset = grid_contour.reset_index()
    grid_reset.columns = ['max_features', 'n_estimators', 'R2']
    grid_pivot = grid_reset.pivot('max_features', 'n_estimators')
```

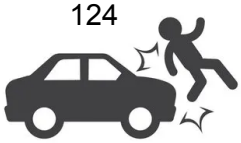
```
    x = grid_pivot.columns.levels[1].values
    y = grid_pivot.index.values
    z = grid_pivot.values

    #-----Plot-----#
    layout = go.Layout(
            xaxis=go.layout.XAxis(
              title=go.layout.xaxis.Title(
              text='n_estimators')
             ),
             yaxis=go.layout.YAxis(
              title=go.layout.yaxis.Title(
              text='max_features')
            ) )
    fig = go.Figure(data= [go.Surface(z=z, y=y, x=x)], layout=layout )
    fig.update_layout(title='Hyperparameter tuning',
                      scene = dict(
                        xaxis_title='n_estimators',
                        yaxis_title='max_features',
                        zaxis_title='R2'),
                      autosize=False,
                      width=800, height=800,
                      margin=dict(l=65, r=50, b=65, t=90))
    st.plotly_chart(fig)

    #-----Save grid data-----#
    x = pd.DataFrame(x)
    y = pd.DataFrame(y)
    z = pd.DataFrame(z)
    df = pd.concat([x,y,z], axis=1)
    st.markdown(filedownload(grid_results), unsafe_allow_html=True)

#---------------------------------#
if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)
    st.write(df)
    build_model(df)
else:
    st.info('Awaiting for CSV file to be uploaded.')
    if st.button('Press to use Example Dataset'):
```

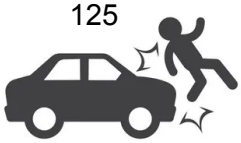
```
data = pd.read_csv("US_Norm.csv")


st.markdown('The **US Accidents** dataset is used as the example.')
st.write(data.head(5))
build_model(data)
```

General visualizations:

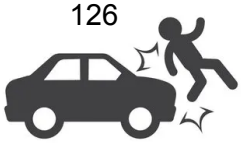
```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
from mlxtend.plotting import plot_pca_correlation_graph
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff
```

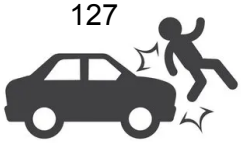
```
from scipy import stats
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import folium
from streamlit_folium import folium_static
from folium.plugins import HeatMap

warnings.filterwarnings('ignore')



st.set_page_config(initial_sidebar_state="collapsed",page_title= " General Data
Visualisation Window ",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png",layout="wide")


st.markdown("""# **A General Data Visualisation Window**""")
add_vertical_space(2)
st.markdown("""##### :smile: This Window is specifically generated to carry out
Generalised Visualisations on the data and grasping a strong knowledge on the same
:balloon: """ )

add_vertical_space(3)

c1,c2,c3 = st.columns(3)
with c2:
    fl = st.file_uploader(":file_folder: Upload your file",type=(["csv","txt","xlsx","xls"]))
if fl is not None:
    filename = fl.name
    st.write(filename)
    df = pd.read_csv(filename, encoding = "ISO-8859-1")
else:
```

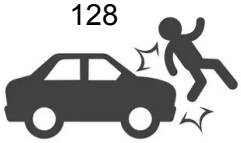
```
    df = pd.read_csv("US_Accidents53.csv", encoding = "ISO-8859-1")

add_vertical_space(3)

add_vertical_space(3)

st.markdown("### Some General Visualisations")
add_vertical_space(3)

c1,c2 = st.columns(2)
with c1:
    fig = px.box(df, x='Severity', y='Temperature(F)', color='Severity', title='Temperature
vs. Severity of Accidents')
    st.plotly_chart(fig,use_container_width=True)

with c2:
    fig = px.histogram(df, x='Hour', nbins=24, color_discrete_sequence=['orange'],
labels={'Hour': 'Hour of the Day', 'count': 'Frequency'}, title='Histogram of Accident
Times')
    fig.update_layout(xaxis_title='Hour of the Day', yaxis_title='Frequency',
title_font_size=16, showlegend=False)
    st.plotly_chart(fig,use_container_width=True)

add_vertical_space(5)

df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['Weekday'] = df['Start_Time'].dt.day_name()
c1,c2 = st.columns(2)
with c1:
    # Pie chart using Plotly Express
    st.markdown('##### Accidents by Day of the Week')
    week_day_counts = df['Weekday'].value_counts()
    print(df['Weekday'])

    fig = px.pie(df, names=week_day_counts.index, values=week_day_counts.values,
            title='Accidents by Day of the Week', color_discrete_sequence=['skyblue',
'lightgreen', 'lightcoral', 'orange', 'lightyellow', 'lightpink', 'lightskyblue'])
    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.update_layout(title_font_size=16, title_font_color='purple')
```

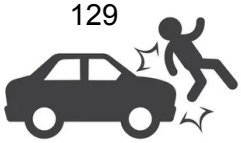
```
    st.plotly_chart(fig,use_container_width=True)

with c2:
    # Scatter plot using Plotly Express
    st.markdown('##### Wind Speed vs. Visibility')
    fig = px.scatter(df, x='Wind_Speed(mph)', y='Visibility(mi)', color='Visibility(mi)',
                     title='Wind Speed vs. Visibility', labels={'Wind_Speed(mph)': 'Wind Speed
(mph)', 'Visibility(mi)': 'Visibility (miles)'})
    fig.update_traces(marker=dict(size=10, opacity=0.5), selector=dict(mode='markers'))
    fig.update_layout(title_font_size=16, title_font_color='brown', showlegend=False)
    # Update layout to display labels around the pie chart

    st.plotly_chart(fig,use_container_width=True)


add_vertical_space(5)

c1,c2 = st.columns(2)
with c1:
    st.markdown("##### Scatter Plot of Wind Speed Vs. Visibility")
    # Scatter plot using Plotly Express
    fig = px.scatter(df, x='Wind_Speed(mph)', y='Visibility(mi)', color='Visibility(mi)',
                     title='Wind Speed vs. Visibility', labels={'Wind_Speed(mph)': 'Wind Speed
(mph)', 'Visibility(mi)': 'Visibility (miles)'})

    # Update layout
    fig.update_layout(title_font_size=16, title_font_color='blue',
                      xaxis_title='Wind Speed (mph)', yaxis_title='Visibility (miles)',
                      xaxis=dict(showgrid=True, gridwidth=1, gridcolor='lightgrey'),
                      yaxis=dict(showgrid=True, gridwidth=1, gridcolor='lightgrey'))

    st.plotly_chart(fig, use_container_width=True)

with c2:
    # Convert 'Start_Time' column to datetime and set it as index
    st.markdown("##### Trends Line Chart")
    df['Start_Time'] = pd.to_datetime(df['Start_Time'])
    df.set_index('Start_Time', inplace=True)

    # Resample the data by month and plot
    monthly_accidents = df.resample('M').size()
```

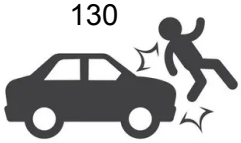
```
    fig = px.line(x=monthly_accidents.index, y=monthly_accidents.values,
                  title='Accidents Over Time', labels={'x': 'Date', 'y': 'Number of Accidents'})
    fig.update_traces(line=dict(color='yellow', dash='dot', width=2))
    fig.update_layout(title_font_size=16, title_font_color='yellow',
                      xaxis_title_font_size=12, yaxis_title_font_size=12,
                      xaxis=dict(showgrid=True, gridwidth=1, gridcolor='lightgrey'),
                      yaxis=dict(showgrid=True, gridwidth=1, gridcolor='lightgrey'))

    st.plotly_chart(fig, use_container_width=True)

add_vertical_space(5)

c1,c2 = st.columns(2)
with c1:
    st.markdown("##### Count Plot of Accident Severity")
    # Count plot using Plotly Express

    fig = px.histogram(df, x='Severity', title='Distribution of Accident Severity',
color_discrete_sequence=['orange'],
                       labels={'Severity': 'Severity'})
    fig.update_layout(title_font_size=16, title_font_color='orange',
                      xaxis_title='Severity', yaxis_title='Count',
                      xaxis=dict(showgrid=False), yaxis=dict(showgrid=True, gridwidth=1,
gridcolor='lightgrey'))

    st.plotly_chart(fig, use_container_width=True)

with c2:
    # Convert 'Start_Time' column to datetime and set it as index
    # Weather Condition Impact
    st.markdown('##### Accidents by Weather Condition')
    fig_weather = px.bar(df['Weather_Condition'].value_counts(),
                         orientation='h',
                         title='Accidents by Weather Condition',
                         labels={'value': 'Number of Accidents', 'index': 'Weather Condition'},
                         color_discrete_sequence=['red'],
                         height=500)
    fig_weather.update_layout(title_font_size=16, title_font_color='red',
                              yaxis_title='Weather Condition', xaxis_title='Number of Accidents')

    st.plotly_chart(fig_weather, use_container_width=True)
```

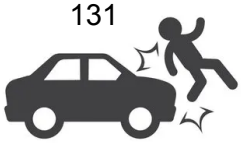
```
c1,c2 = st.columns(2)
with c1:
    # Accident Hotspots by City
    city_counts = df['City'].value_counts().head(10)
    st.markdown('##### Top 10 Cities by Accident Counts')
    fig_city = px.bar(city_counts,
                title='Top 10 Cities by Accident Counts',
                labels={'value': 'Number of Accidents', 'index': 'City'},
                color_discrete_sequence=['purple'],
                height=500)
    fig_city.update_layout(title_font_size=16, title_font_color='purple',
                    yaxis_title='Number of Accidents', xaxis_title='City')

    st.plotly_chart(fig_city, use_container_width=True)

df = pd.read_csv('US_Accidents53.csv')

with c2:
        # Set a custom color palette
    custom_palette = sns.color_palette("Set2")
    sns.set_palette(custom_palette)

    # Set the seaborn style
    sns.set_style("whitegrid")

    # 1. Distribution of Accidents by Month
    st.markdown('##### Distribution of Accidents by Month')
    fig1, ax1 = plt.subplots()
    sns.countplot(x='Month', data=df, ax=ax1)
    ax1.set_title('Distribution of Accidents by Month')
    ax1.set_xlabel('Month')
    ax1.set_ylabel('Number of Accidents')
    st.pyplot(fig1,use_container_width=True)

add_vertical_space(5)
c1,c2 = st.columns(2)

add_vertical_space(5)
with c1:
```

```
    # 2. Accident Duration Analysis
    st.markdown('##### Box Plot of Accident Duration')
    fig2, ax2 = plt.subplots()
    df['Duration'] = (pd.to_datetime(df['End_Time']) -
pd.to_datetime(df['Start_Time'])).dt.total_seconds() / 3600
    sns.boxplot(x='Duration', data=df, ax=ax2)
    ax2.set_title('Box Plot of Accident Duration')
    ax2.set_xlabel('Duration (Hours)')
    st.pyplot(fig2,use_container_width=True)

with c2:
    # 3. Scatter Plot of Pressure vs. Humidity
    st.markdown('##### Pressure vs. Humidity')
    fig1 = px.scatter(df, x='Pressure(in)', y='Humidity(%)')
    fig1.update_layout(xaxis_title='Atmospheric Pressure (in)', yaxis_title='Humidity (%)')
    fig.update_traces(marker=dict(color='rgba(0,0,0,0)'))  # Make the scatter markers
transparent
    fig.update_layout(barmode='overlay')

    # Marginal X bar plot
    fig.add_trace(px.bar(df, x='Pressure(in)').data[0])

    # Marginal Y bar plot
    fig.add_trace(px.bar(df, y='Humidity(%)').data[0])
    st.plotly_chart(fig1,use_container_width=True)


add_vertical_space(5)
# Set different color palettes
palette1 = px.colors.qualitative.Pastel
palette2 = px.colors.qualitative.Set3
palette3 = px.colors.sequential.Viridis
c1,c2 = st.columns(2)


with c1:
    # 6. Day-Night Accident Comparison
    st.markdown('##### Day vs. Night Accidents')
    fig2 = px.bar(df['Sunrise_Sunset'].value_counts(),
x=df['Sunrise_Sunset'].value_counts().index,
y=df['Sunrise_Sunset'].value_counts().values,
```

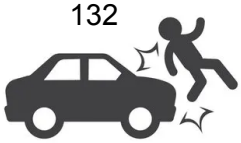
```
            title='Day vs. Night Accidents', color_discrete_sequence=palette1)
    fig2.update_layout(xaxis_title='Time of Day', yaxis_title='Number of Accidents')
    st.plotly_chart(fig2)

with c2:
    wata = pd.read_csv("US_Accident23_1000.csv")
    # 7. Impact of Visibility on Accident Severity
    st.markdown('##### Visibility vs. Severity')
    fig3 = px.violin(wata, x='Severity', y='Visibility(mi)', title='Visibility vs. Severity',
color='Severity', color_discrete_sequence=palette1)
    fig3.update_layout(xaxis_title='Severity', yaxis_title='Visibility (miles)')
    st.plotly_chart(fig3)


add_vertical_space(5)

c1,c2 = st.columns([0.4,0.6])
with c1:
    st.markdown("##### Rose Plot for Accidents and Wind Direction")
    add_vertical_space(2)
    # Generate sample data for demonstration (replace with your actual data)
    wind_directions = np.random.randint(0, 360, size=len(df))  # Wind directions in
degrees
    accident_severity = df['Severity']  # Accident severity levels

    # Create a rose plot
    num_bins = 36  # Number of bins (36 for each 10 degrees)
    theta = np.linspace(0.0, 2 * np.pi, num_bins, endpoint=False)
    radii, _ = np.histogram(wind_directions, bins=num_bins)
    colors = plt.cm.viridis(accident_severity / max(accident_severity))  # Color mapping
based on severity

    fig, ax = plt.subplots(subplot_kw={'projection': 'polar'}, figsize=(8, 8))
    bars = ax.bar(theta, radii, width=2 * np.pi / num_bins, color=colors, edgecolor='black')

    plt.title('Wind Direction vs. Accident Severity')
    st.pyplot(fig,use_container_width=True)

with c2 :
    st.markdown("##### Kernel Density Accidents in Folium Map")
    add_vertical_space(2)
```

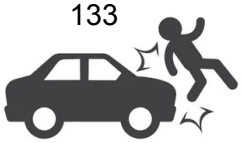
```
    from streamlit_folium import folium_static
    import folium
    # Create a map centered around an average location
    map_center = [df['Start_Lat'].mean(), df['Start_Lng'].mean()]
    map = folium.Map(location=map_center, zoom_start=5)

    # Add a heatmap layer
    heatmap = HeatMap(list(zip(df['Start_Lat'], df['Start_Lng'])), min_opacity=0.2,
radius=15, blur=15)
    map.add_child(heatmap)

    # Save the map as an HTML file
    map.save('accident_heatmap.html')
    print('Heatmap generated and saved as accident_heatmap.html.')
    folium_static(map)
```

Inferential visualizations:

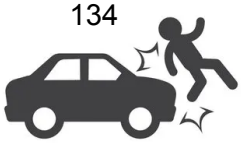
```
import streamlit as st
import streamlit_extras as stex
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_extras.dataframe_explorer import dataframe_explorer
import plotly.express as px
import warnings
import pandas as pd
from mlxtend.plotting import plot_pca_correlation_graph
import os
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import numpy as np
import math
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.preprocessing import LabelEncoder
```

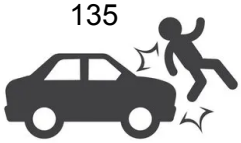
```
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import plotly.figure_factory as ff

from scipy import stats
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import folium
from streamlit_folium import folium_static
from folium.plugins import HeatMap

warnings.filterwarnings('ignore')



st.set_page_config(initial_sidebar_state="collapsed",page_title= " Inferential Data
Visualisation Window ",
        menu_items={
         'Get Help':
'https://drive.google.com/drive/folders/1gosDbNFWAlPriVNjC8_PnytQv7YimI1V?usp=
drive_link',
         'Report a bug': "mailto:a.k.mirsha9@gmail.com",
         'About': "### This is an extremely cool web application built as a part of my Data
Science Mini Project on the ' US Accidents Dataset '\n"
    },page_icon="analysis.png",layout="wide")


st.title(''' **The Inferential Data Visualisation Window**''')
add_vertical_space(2)
st.markdown('''##### :smile: This Window is specifically generated to carry out Highly
Meaningful and Inferential Visualisations on the data and arriving at Conclusions on
the same. :balloon: ''' )

add_vertical_space(3)
```

```
c1,c2,c3 = st.columns(3)
with c2:
    fl = st.file_uploader(":file_folder: Upload your file",type=(["csv","txt","xlsx","xls"]))
if fl is not None:
    filename = fl.name
    st.write(filename)
    df = pd.read_csv(filename, encoding = "ISO-8859-1")
else:
    df =
pd.read_csv(r"C:\Users\mekes\Downloads\STREAMLIT-20240526T095929Z-001\STREA
MLIT\US.csv", encoding = "ISO-8859-1")
    df = df.sample(n=10000,random_state = 42)

add_vertical_space(3)

add_vertical_space(3)

st.markdown("## Some Inferential Visualisations")
add_vertical_space(3)


c2,c3 = st.columns([5,4.75],gap='medium')
with c2:
    st.header("Identification of the 10 accident-prone streets in USA.")
    add_vertical_space(3)

    # Calculate the top 10 streets with the most number of accidents
    top_10_streets = df['Street'].value_counts().head(10).reset_index()
    top_10_streets.columns = ['Street', 'Cases']

    # Calculate total cases for annotation purposes
    total = sum(df['Street'].value_counts())

    # Streamlit app

    # Create a bar plot using Plotly Express
    fig = px.bar(
        top_10_streets,
        x='Street',
        y='Cases',
```

```
        color='Cases',
        color_continuous_scale='rainbow',
        title='Top 10 Accident-Prone Streets in the US'
    )

    # Annotate each bar with percentage
    fig.update_traces(texttemplate='%{y} cases<br>%{text:.2f}%', text=[(value / total) *
100 for value in top_10_streets['Cases']], textposition='outside')

    # Update layout for better aesthetics
    fig.update_layout(
        xaxis_title='Streets',
        yaxis_title='Accident Cases',
        title_font_size=20,
        title_font_color='grey',
        xaxis_tickangle=-45,
        xaxis_tickfont_size=12,
        yaxis_tickfont_size=12,
        plot_bgcolor='white'
    )

    # Show grid lines for the y-axis
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#b2d6c7')

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)
    add_vertical_space(5)

with c3:


    st.write('''## Inference:

#### Top 10 Accident-Prone Streets in the US

1. **Most Accident-Prone Streets**:
   - **I-95 S** is the most accident-prone street, with **126 accident cases**.
   - **I-95 N** follows closely with **122 accident cases**.

2. **Significant Streets**:
   - **I-5 N** shows a notable number of accidents, totaling **75 cases**.
```

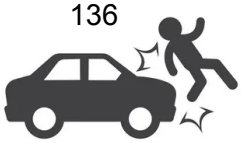
- **I-94 W** and **I-80 W** also have significant accident cases, with **63** and **62 cases** respectively.

3. **Other Notable Streets**:
   - **I-25 N**, **I-70 W**, and **I-80 E** each have **60**, **56**, and **56 accident cases**, respectively.
   - **I-10 E** and **I-5 S** show **55** and **53 accident cases**, respectively.

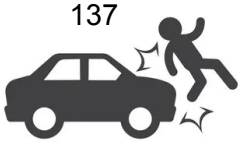
4. **Visual Appeal**:
   - The color scale from red to purple effectively highlights the distribution of accidents, with red indicating higher accident counts and purple indicating lower counts, enhancing visual understanding of the most accident-prone streets.

Overall, the plot identifies that **I-95 S** and **I-95 N** are the streets with the highest number of road accidents, suggesting a need for targeted safety measures on these routes. This insight can inform traffic safety initiatives and preventive measures to reduce accidents on these critical streets.''')

```
with c2:
    add_vertical_space(4)
    # Calculate the number of accidents per hour
    accidents_per_hour = df['Hour'].value_counts().sort_index().reset_index()
    accidents_per_hour.columns = ['Hour', 'Accidents']

    # Streamlit app
    st.header('Number of Road Accidents by Hour of the Day in the US')

    # Create a bar plot using Plotly Express
    fig = px.bar(
        accidents_per_hour,
        x='Hour',
        y='Accidents',
        color='Accidents',
        color_continuous_scale='viridis',
        title='Number of Road Accidents by Hour of the Day in the US'
    )

    # Update layout for better aesthetics
    fig.update_layout(
        xaxis_title='Hour of the Day',
        yaxis_title='Number of Accidents',
```

```
        title_font_size=20,
        title_font_color='grey',
        xaxis_tickmode='array',
        xaxis_tickvals=list(range(24)),
        xaxis_ticktext=[f'{i}:00' for i in range(24)],
        xaxis_tickangle=-45,
        xaxis_tickfont_size=12,
        yaxis_tickfont_size=12,
        plot_bgcolor='white'
    )

    # Show grid lines for the y-axis
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#b2d6c7')

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)

    # Print the hour with the most accidents
    peak_hour = accidents_per_hour.loc[accidents_per_hour['Accidents'].idxmax(),
'Hour']
    peak_accidents = accidents_per_hour.loc[accidents_per_hour['Accidents'].idxmax(),
'Accidents']
    st.write(f"The hour with the most road accidents is {peak_hour}:00 with
{peak_accidents} accidents.")


with c3:
    add_vertical_space(3)

    st.write('''## Inference:

#### Number of Road Accidents by Hour of the Day in the US

1. **Peak Accident Hour**:
   - **16:00** is the hour with the most road accidents, totaling **840 accidents**.

2. **High Accident Period**:
   - There is a significant increase in accidents between **15:00 and 18:00**, indicating a
peak period likely corresponding to afternoon and evening rush hours.

3. **Morning Increase**:
```

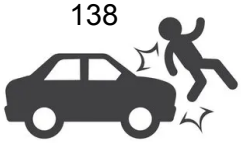
- A noticeable increase in accidents starts around **6:00**, with a steady rise through the morning hours, peaking again around **8:00**.

4. **Lower Accident Periods**:
   - Early morning hours (**0:00 to 5:00**) and late evening hours (**20:00 to 23:00**) have relatively fewer accidents.

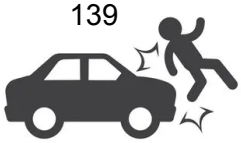
5. **Visual Appeal**:
   - The 'viridis' color scale effectively highlights the distribution of accidents, with brighter colors indicating higher accident counts, enhancing visual understanding of critical times.

Overall, the plot clearly identifies the times of day with the highest accident rates, particularly focusing on afternoon rush hours, suggesting the need for targeted traffic management during these peak periods.''')

```
with c2:
    add_vertical_space(8)
        # Count the number of accidents for each weather condition
    weather_counts = df['Weather_Condition'].value_counts().head(10).reset_index()
    weather_counts.columns = ['Weather_Condition', 'Cases']

    # Streamlit app
    st.header('Top 10 Weather Conditions During Road Accidents in the US')

    # Create a bar plot using Plotly Express
    fig = px.bar(
        weather_counts,
        x='Weather_Condition',
        y='Cases',
        color='Cases',
        color_continuous_scale='viridis',
        title='Top 10 Weather Conditions During Road Accidents in the US'
    )

    # Update layout for better aesthetics
    fig.update_layout(
        xaxis_title='Weather Condition',
        yaxis_title='Number of Accidents',
        title_font_size=20,
        title_font_color='grey',
```

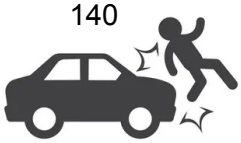
```
        xaxis_tickangle=-45,
        xaxis_tickfont_size=12,
        yaxis_tickfont_size=12,
        plot_bgcolor='white'
    )

    # Show grid lines for the y-axis
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#b2d6c7')

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)

    # Print the weather condition with the most accidents
    top_weather_condition = weather_counts.iloc[0]
    top_weather = top_weather_condition['Weather_Condition']
    top_cases = top_weather_condition['Cases']
    st.write(f"The weather condition with the most road accidents is {top_weather} with
{top_cases} accidents.")

with c3:
    add_vertical_space(3)
    st.write('''## Inference:

#### Top 10 Weather Conditions During Road Accidents in the US

1. **Most Accident-Prone Weather Condition**:
   - **Fair** is the weather condition with the most road accidents, totaling **4367
accidents**.

2. **Significant Weather Conditions**:
   - **Cloudy** and **Mostly Cloudy** conditions also show a considerable number of
accidents, though significantly fewer than Fair weather.
   - **Partly Cloudy** and **Light Rain** conditions follow, contributing to the overall
accident count but at a lower frequency.

3. **Less Frequent Conditions**:
   - **Light Snow**, **Fog**, **Rain**, **Fair/Windy**, and **Haze** have relatively fewer
accidents compared to the top conditions.

4. **Visual Appeal**:
```

- The 'viridis' color scale effectively highlights the distribution of accidents, with brighter colors indicating higher accident counts, enhancing visual understanding of critical weather conditions.

Overall, the plot helps identify that most road accidents occur under **Fair** weather conditions, suggesting that even in seemingly safe conditions, vigilance is necessary. This insight can inform safety campaigns and preventive measures across varying weather conditions.''')

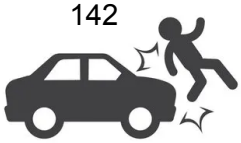
```
with c2:
    add_vertical_space(9)
    # Convert 'Start_Time' and 'End_Time' columns to datetime objects
    df['Start_Time'] = pd.to_datetime(df['Start_Time'])
    df['End_Time'] = pd.to_datetime(df['End_Time'])
        # Calculate accident duration
    df['Accident_Duration'] = (df['End_Time'] - df['Start_Time']).dt.total_seconds() /
3600  # Convert duration to hours

    # Get the top 10 accident durations
    top_10_accident_duration =
df['Accident_Duration'].value_counts().head(10).reset_index()
    top_10_accident_duration.columns = ['Duration (hours)', 'Number of Cases']

    # Streamlit app
    st.header('Top 10 Accident Durations')

    # Create a bar plot using Plotly Express
    fig = px.bar(
        top_10_accident_duration,
        x='Duration (hours)',
        y='Number of Cases',
        color='Number of Cases',
        color_continuous_scale='viridis',
        title='Top 10 Accident Durations'
    )

    # Update layout for better aesthetics
    fig.update_layout(
        xaxis_title='Duration (hours)',
        yaxis_title='Number of Cases',
        title_font_size=20,
```

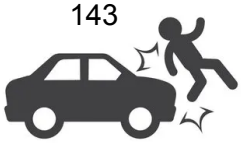
```
        title_font_color='grey',
        xaxis_tickangle=-45,
        xaxis_tickfont_size=12,
        yaxis_tickfont_size=12,
        plot_bgcolor='white'
    )

    # Show grid lines for the y-axis
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#b2d6c7')

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)

    # Print the duration with the most accidents
    top_duration_condition = top_10_accident_duration.iloc[0]
    top_duration = top_duration_condition['Duration (hours)']
    top_cases = top_duration_condition['Number of Cases']
    st.write(f"The accident duration with the most cases is {top_duration} hours with
{top_cases} cases.")

with c3:
    add_vertical_space(1)
    st.write('''## Inference:

#### Top 10 Accident Durations

1. **Most Frequent Accident Duration**:
   - The duration with the most cases is **0.25 hours**, totaling **1244 cases**.

2. **Significant Durations**:
   - Durations around **0.5 hours** and **1 hour** also show a considerable number of
accidents, indicating a high frequency of shorter duration accidents.

3. **Less Frequent Durations**:
   - Durations extending up to **6 hours** are observed but with significantly fewer
cases compared to shorter durations.

4. **Visual Appeal**:
   - The 'viridis' color scale effectively highlights the distribution of accidents, with
brighter colors indicating higher accident counts, enhancing visual understanding of
critical accident durations.
```


Overall, the plot helps identify that most accidents are resolved within a short period, typically **0.25 hours**. This insight can inform traffic management and emergency response strategies to improve road safety and reduce accident durations.''')

```
with c2:
    add_vertical_space(5)
    # Convert 'Start_Time' column to datetime object
    df['Start_Time'] = pd.to_datetime(df['Start_Time'])

    # Extract year from 'Start_Time'
    df['Year'] = df['Start_Time'].dt.year

    # Count the number of accident cases for each year
    accident_cases_by_year = df['Year'].value_counts().sort_index().reset_index()
    accident_cases_by_year.columns = ['Year', 'Number of Cases']

    # Streamlit app
    st.header('Accident Cases Over the Years')

    # Create a line plot using Plotly Express
    fig = px.line(
        accident_cases_by_year,
        x='Year',
        y='Number of Cases',
        markers=True,
        title='Accident Cases Over the Years'
    )

    # Update layout for better aesthetics
    fig.update_layout(
        xaxis_title='Year',
        yaxis_title='Number of Cases',
        title_font_size=24,
        title_font_color='grey',

        xaxis_tickangle=-45,
        xaxis_tickfont_size=14,
        yaxis_tickfont_size=14,
        plot_bgcolor='white',
```

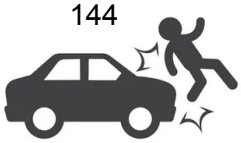
```
        font=dict(family='Arial', size=12, color='black')
    )

    # Show grid lines
    fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#b2d6c7')

    # Add a horizontal line to show the mean number of cases
    mean_cases = accident_cases_by_year['Number of Cases'].mean()
    fig.add_hline(y=mean_cases, line_dash='dash', line_color='red',
annotation_text='Mean',
                  annotation_position='bottom right')

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)

    # Print the year with the most accidents
    top_year_condition =
accident_cases_by_year.iloc[accident_cases_by_year['Number of Cases'].idxmax()]
    top_year = top_year_condition['Year']
    top_cases = top_year_condition['Number of Cases']
    st.write(f"The year with the most accident cases is {top_year} with {top_cases}
cases.")


with c3:
    add_vertical_space(2)
    st.write('''## Inference:

#### Accident Cases Over the Years

1. **Year with the Most Accident Cases**:
   - **2020** had the most accident cases, totaling **5255 cases**.

2. **Trend Over the Years**:
   - There is a significant increase in the number of cases leading up to 2020, followed by
a sharp decline in 2021.
   - A slight increase is observed in 2022, but the number of cases drops again in 2023.

3. **Mean Line**:
   - The red dashed line represents the mean number of cases over the years, providing
a benchmark to compare yearly data.
```

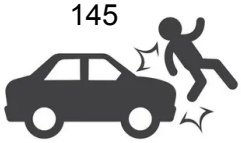
4. **Visual Appeal**:
   - The line chart effectively shows the trend of accident cases over the years, with clear peaks and troughs that highlight significant changes in accident frequencies.

Overall, the plot indicates that 2020 was an outlier year with an exceptionally high number of accidents. Understanding the factors contributing to this spike can help in formulating strategies to prevent similar surges in the future.''')

```
with c2:
    add_vertical_space(7)
    # List of road conditions
    # List of road conditions
    road_conditions = ['Crossing' ,'Junction', 'Roundabout', 'Stop', 'Traffic_Calming',
                       'Traffic_Signal']

    # Calculate the sum of each road condition to find the top 5 useful amenities
    condition_sums = df[road_conditions].sum().sort_values(ascending=False).head(5)
    top_6_conditions = condition_sums.index.tolist()

    # Colors for the pie chart slices
    colors = ['#4CAF50', '#FF9800', '#03A9F4', '#E91E63', '#9C27B0','#9C26B0']

    # Streamlit app
    st.header('Top 5 Useful Amenities in Road Conditions')

    # Create a pie chart using Plotly Express
    fig = go.Figure()

    # Function to annotate each slice with percentage
    def func(pct, allvals):
        absolute = int(pct / 100. * sum(allvals))
        return "{:.1f}%\n({:d} Cases)".format(pct, absolute)

    # Plot pie chart for each top 5 road condition
    for idx, condition in enumerate(top_6_conditions):
        values = df[condition].value_counts().tolist()
        labels = ['False', 'True']
        colors_condition = ['lightgrey', colors[idx]]
```

```
        fig.add_trace(go.Pie(
            labels=labels,
            values=values,
            name=condition,
            hole=0.3,
            textinfo='percent+label',
            insidetextorientation='horizontal',

            marker=dict(colors=colors_condition, line=dict(color='grey', width=2)),
            domain=dict(row=idx, column=0)
        ))

    # Update layout for better aesthetics
    fig.update_layout(
        title_text='Top 5 Useful Amenities in Road Conditions',
        annotations=[dict(text=condition, x=0.2, y=0.925 - 0.175 * idx, font_size=14,
showarrow=False) for idx, condition in enumerate(top_6_conditions)],
        grid=dict(rows=6, columns=1),
        showlegend=False,
        title_font_size=24,
        title_font_color='grey',

        font=dict(family='Arial', size=12, color='black'),
        height=1200  # Set the height to avoid overlapping
    )

    # Display the plot in the Streamlit app
    st.plotly_chart(fig)

with c3:
    add_vertical_space(2)
    st.write('''## Inference:

#### Top 5 Useful Amenities in Road Conditions

1. **Traffic Signal**:
   - **Traffic signals** are present in **21.4%** of the road conditions, making them the
most common useful amenity. This indicates a significant presence of controlled
intersections to manage traffic flow and enhance safety.

2. **Crossing**:
```

- **Crossings** account for **16.2%** of the road conditions. This suggests that pedestrian crossings are a crucial feature in urban planning to ensure pedestrian safety and reduce accidents.

3. **Junction**:
   - **Junctions** are found in **10.7%** of the cases. The presence of junctions highlights the importance of road intersections and their potential role in accident occurrences due to the convergence of multiple roads.

4. **Stop**:
   - **Stop signs** are present in **2.41%** of the road conditions, indicating a lesser but still important role in traffic control, particularly in residential or low-traffic areas.

5. **Traffic Calming**:
   - **Traffic calming measures** (e.g., speed bumps) are seen in only **0.08%** of the road conditions. This low percentage points to a minimal implementation of such measures, which are typically used to slow down traffic and enhance safety in specific areas.

4. **Visual Appeal**:
   - The use of donut charts effectively displays the proportion of True and False values for each amenity, with distinct colors aiding visual interpretation and understanding of the distribution.

Overall, the plot highlights that traffic signals and crossings are the most prevalent amenities in road conditions, underscoring their critical role in managing traffic and ensuring safety. The lower presence of stop signs and traffic calming measures suggests areas for potential improvement in road safety measures.''')

```
from streamlit_extras.bottom_container import bottom
with bottom():
    st.markdown("#### Congratulations, you have reached the last page of this Web Application !!!")
    st.write("**Thank You So Much** for Allocating Time for getting yourself immersed in this Data Exploration and Visualisation Dashboard....")
```