```
from google.colab import files
upload =files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
import pandas as pd
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")
df.head()
```

	Date	Time	Location	Weather Condition	Road Type	Traffic Volume	Vehicle Type	Number of Vehicles	Number of Casualties	Cause of Accident	Severity	AI Predicted Risk Score
0	3/14/2023	22:17	Los Angeles	Clear	Rural	688	Bicycle	5	5	Distracted Driving	Minor	0.59
1	4/26/2023	12:00	Chicago	Windy	Rural	554	Car	1	6	Distracted Driving	Fatal	0.80
2	1/31/2023	12:11	Los Angeles	Snow	Intersection	591	Bus	4	4	Distracted Driving	Minor	0.80
3	5/5/2023	16:20	Phoenix	Windy	Urban	520	Bus	1	9	Mechanical Failure	Fatal	0.68
										Distracted		

```
# Check for missing values print(df.isnull().sum())
```

```
df_cleaned = df.dropna()  # Removes rows with missing values print(df_cleaned)
```

```
import pandas as pd

# Create an example DataFrame
data = {'Date': [3/14/2023,4/26/2023,1/31/2023,5/5/2023,6/7/2023],
        'Time': ['22:17','12:00','12:11','16:20','5:08'],
        'Location': ['Los Angeles','Chicago','Los Angeles','Phoenix','New York']}
df = pd.DataFrame(data)

# Now you can fill NaN values
df["Date"].fillna(df["Date"].mean(), inplace=True)
df["Time"].fillna(df["Time"].mode()[0], inplace=True) # Use mode for categorical data like 'Disease'
df["Location"].fillna(df["Location"].mode()[0], inplace=True)

print(df)

# Now you can fill NaN values
df["Date"].fillna(df["Date"].mean(), inplace=True)
# Removing the line causing the error as median is not applicable for 'Time'
# df["Time"].fillna(df["Time"].median(), inplace=True)
df["Location"].fillna(df["Location"].mode()[0], inplace=True) # Use mode for categorical data like 'Location'


print(df)
```

```
       Date   Time     Location
0  0.000106  22:17  Los Angeles
1  0.000076  12:00      Chicago
2  0.000016  12:11  Los Angeles
3  0.000494  16:20      Phoenix
4  0.000424   5:08     New York
       Date   Time     Location
0  0.000106  22:17  Los Angeles
1  0.000076  12:00      Chicago
2  0.000016  12:11  Los Angeles
3  0.000494  16:20      Phoenix
4  0.000424   5:08     New York
<ipython-input-6-fe9f43928c78>:10: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df["Date"].fillna(df["Date"].mean(), inplace=True)
<ipython-input-6-fe9f43928c78>:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
```

```
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df["Time"].fillna(df["Time"].mode()[0], inplace=True) # Use mode for categorical data like 'Disease'
<ipython-input-6-fe9f43928c78>:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df["Location"].fillna(df["Location"].mode()[0], inplace=True)
<ipython-input-6-fe9f43928c78>:17: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df["Date"].fillna(df["Date"].mean(), inplace=True)
<ipython-input-6-fe9f43928c78>:20: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df["Location"].fillna(df["Location"].mode()[0], inplace=True) # Use mode for categorical data like 'Location'
```

```
# Check if 'NAME' column exists before filling missing values
if 'NAME' in df.columns:
    df["NAME"].fillna(df["NAME"].mode()[0], inplace=True)
else:
    print("Column 'NAME' not found in DataFrame.")
```

```
Column 'NAME' not found in DataFrame.
```

```
df.ffill(inplace=True)  # Forward fill df.bfill(inplace=True)  # Backward fill
```

```
df.drop_duplicates(inplace=True)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = df.copy()
# Select only numerical features for scaling
numerical_features = ['Date']  # Include only numerical columns here
df_scaled[numerical_features] = scaler.fit_transform(df[numerical_features])
print(df_scaled)
```

```
       Date   Time     Location
0 -0.598299  22:17  Los Angeles
1 -0.750734  12:00      Chicago
2 -1.057394  12:11  Los Angeles
3  1.383365  16:20      Phoenix
4  1.023062   5:08     New York
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Replace 'Marks' and 'Attendance' with existing numerical columns in your DataFrame 'df'.
# For example, if you have columns named 'Age' and 'chest_pain_type_encoded', you can use them:
existing_numerical_columns = ['Date']  # Include only numerical columns here if 'chest_pain_type_encoded' is not available in your DataFrame
# If 'chest_pain_type_encoded' is available, ensure that it's a numerical column to be scaled

df_scaled[existing_numerical_columns] = scaler.fit_transform(df[existing_numerical_columns])
print(df_scaled)
```

```
       Date   Time     Location
0  0.188095  22:17  Los Angeles
1  0.125641  12:00      Chicago
2  0.000000  12:11  Los Angeles
3  1.000000  16:20      Phoenix
4  0.852381   5:08     New York
```

Double-click (or enter) to edit

```
df_encoded = pd.get_dummies(df, columns=["Time"], drop_first=True) # Changed "tiem" to "Time"
print(df_encoded)
```

```
       Date     Location  Time_12:11  Time_16:20  Time_22:17  Time_5:08
0  0.000106  Los Angeles       False       False        True      False
1  0.000076      Chicago       False       False       False      False
2  0.000016  Los Angeles        True       False       False      False
3  0.000494      Phoenix       False        True       False      False
4  0.000424     New York       False       False       False       True
```

```
def ai_category(time): # Changed 'marks' to 'time' for clarity
    # Convert the time to lowercase for case-insensitive comparison
    time_lower = time.lower()
    if "22:17" in time_lower or "12:11" in time_lower:  # Assuming these are severe conditions
        return "High"
    elif "12:00" in time_lower:
        return "Medium"
    else:  # Assuming other time in your data are categorized as "Low"
        return "Low"

df["ai"] = df["Time"].apply(ai_category) # Changed "time" to "Time" to match the existing column name
print(df)
```

```
       Date   Time  Location      ai
0  0.000106  22:17         1    High
1  0.000076  12:00         0  Medium
2  0.000016  12:11         1    High
3  0.000494  16:20         3     Low
4  0.000424   5:08         2     Low
```

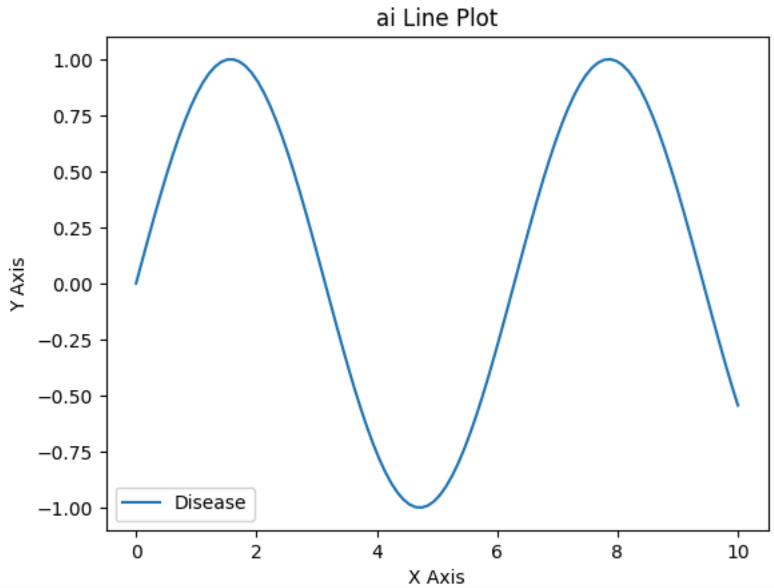
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df["Location"] = encoder.fit_transform(df["Location"])
print(df)
```

```
       Date   Time  Location
0  0.000106  22:17         1
1  0.000076  12:00         0
2  0.000016  12:11         1
3  0.000494  16:20         3
4  0.000424   5:08         2
```

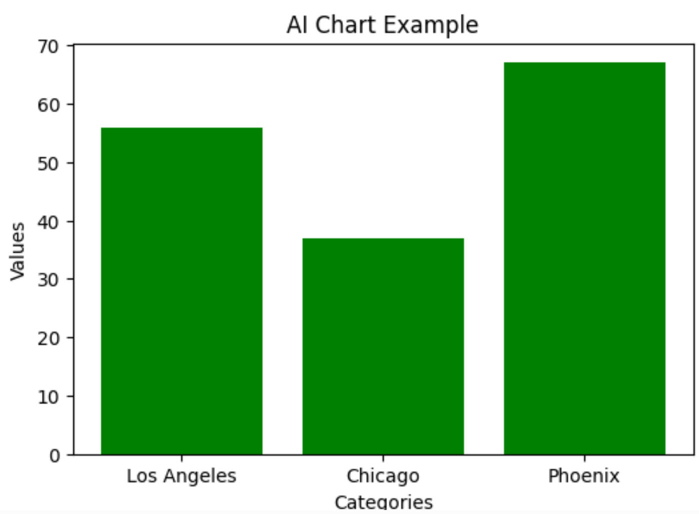
```
df["Date"] = pd.cut(df["Date"], bins=[18, 21, 24], labels=["Young", "Adult"])
print(df)
```

```
  Date   Time  Location      ai
0  NaN  22:17         1    High
1  NaN  12:00         0  Medium
2  NaN  12:11         1    High
3  NaN  16:20         3     Low
4  NaN   5:08         2     Low
```

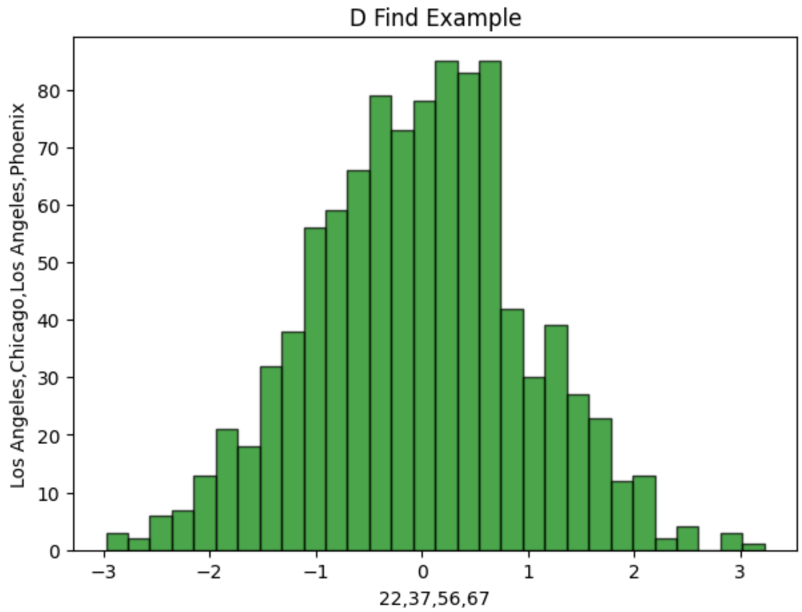
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y, label="Disease")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("ai Line Plot")
plt.legend()
plt.show()
```



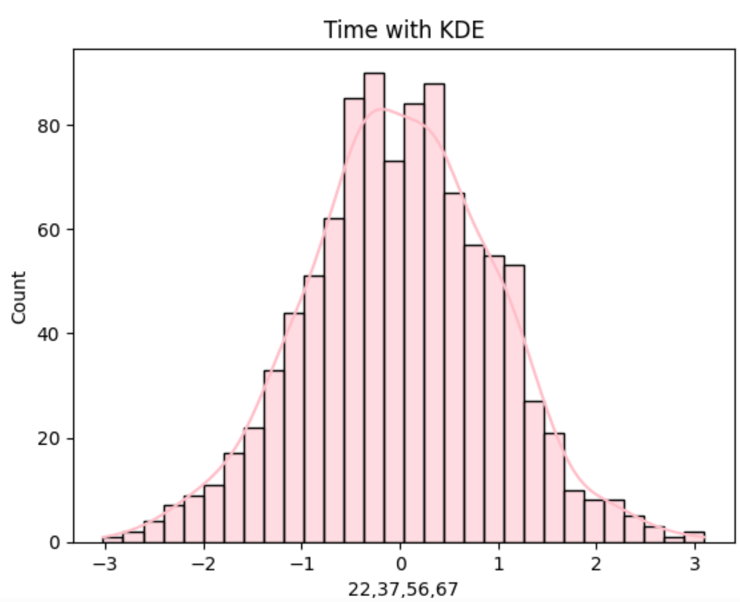

```
categories = ['Los Angeles', 'Chicago', 'Los Angeles', 'Phoenix']
values = [22,37,56,67]

plt.figure(figsize=(6, 4))
plt.bar(categories, values, color='Green')
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("AI Chart Example")
plt.show()
```



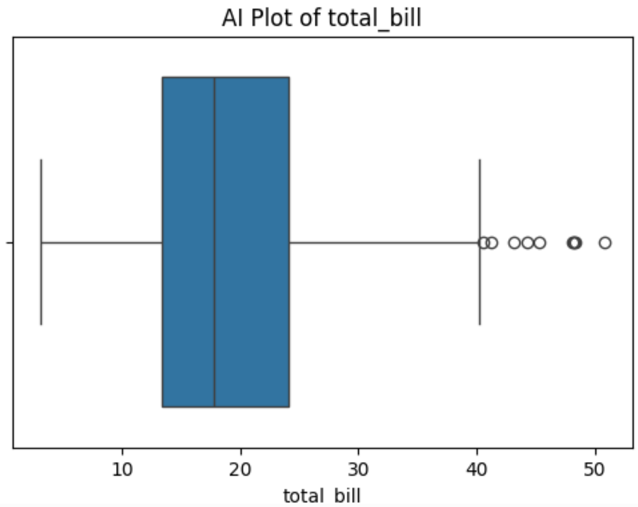

```
data = np.random.randn(1000)

plt.figure(figsize=(7, 5))
plt.hist(data, bins=30, color='green', edgecolor='black', alpha=0.7)
plt.xlabel("22,37,56,67")
plt.ylabel("Los Angeles,Chicago,Los Angeles,Phoenix")
plt.title("D Find Example")
plt.show()
```



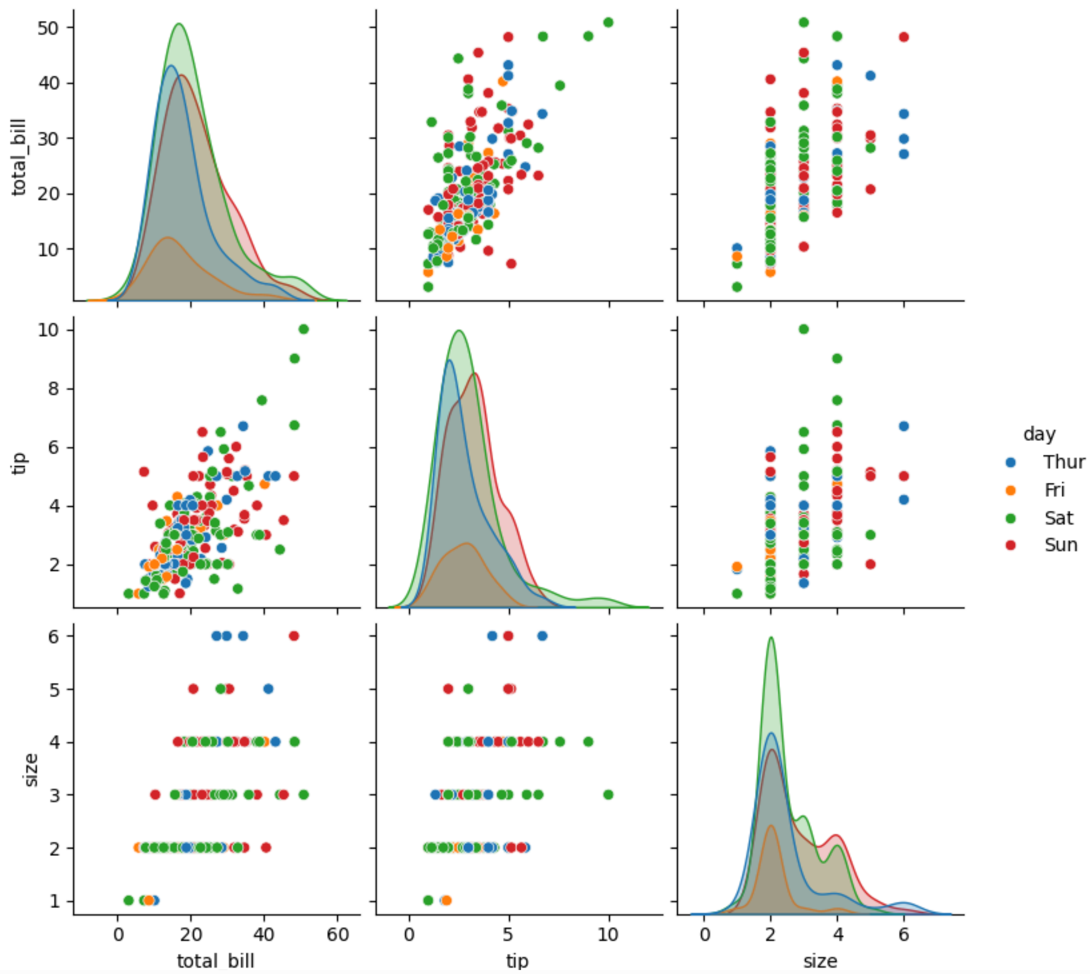

```
import seaborn as sns
import pandas as pd

# Creating sample data
data = np.random.randn(1000)
df = pd.DataFrame(data, columns=['22,37,56,67'])

# Plot
sns.histplot(df['22,37,56,67'], bins=30, kde=True, color='pink')
plt.title("Time with KDE")
plt.show()
```



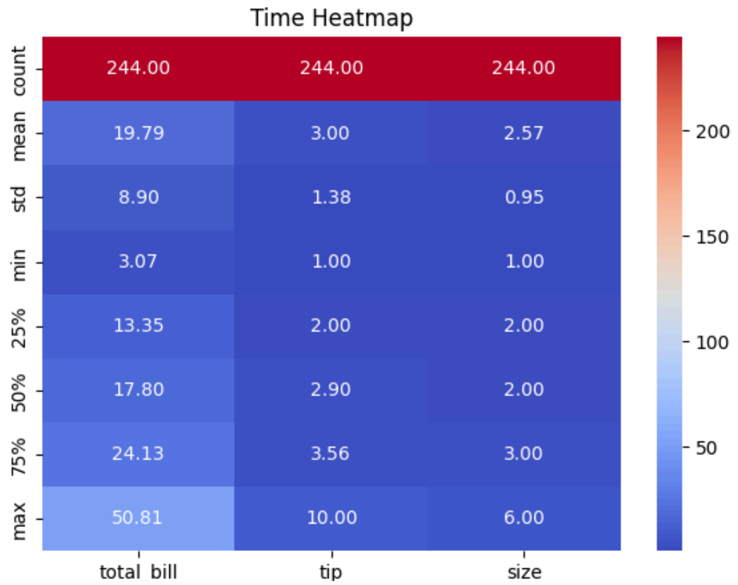

```
tips = sns.load_dataset('tips')

plt.figure(figsize=(6, 4))
# Assuming the column name in your 'tips' dataset is actually 'total_bill' and you want to see the distribution of the total_bill column.
sns.boxplot(x=tips['total_bill'])  # Changed 'Age' to 'total_bill'
plt.title("AI Plot of total_bill") #Update the title
plt.show()
```



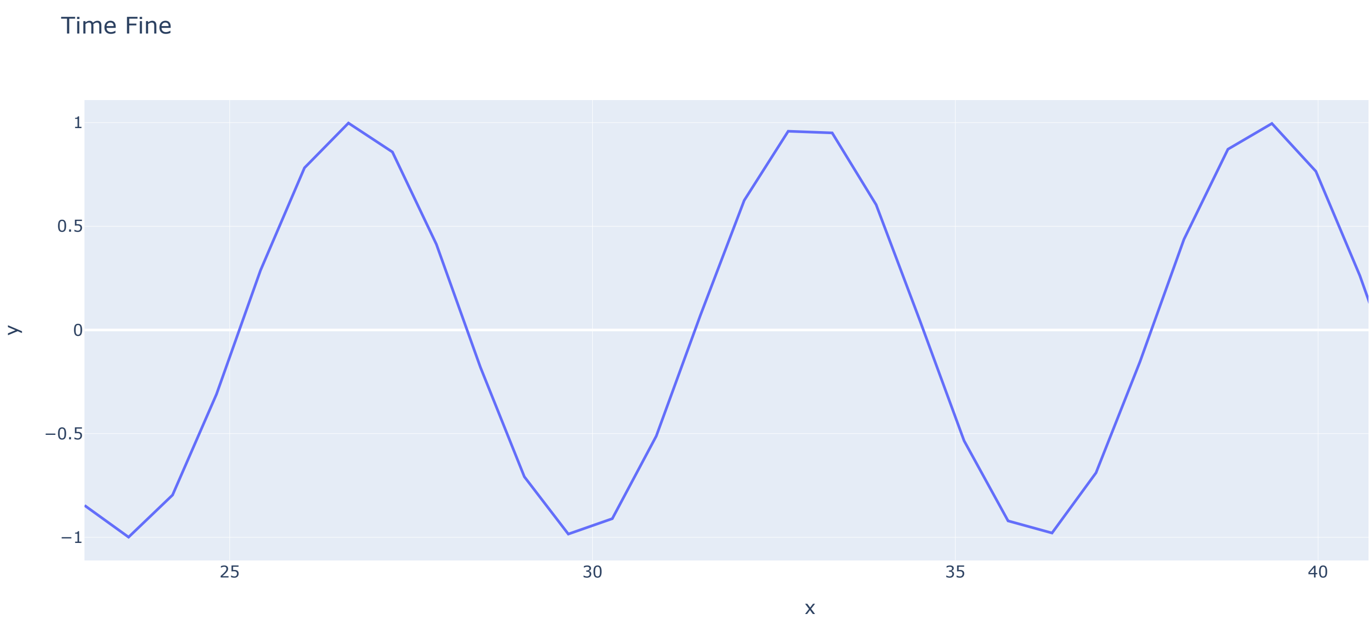

```
sns.pairplot(tips, hue='day') # Changed 'location' to 'day'
plt.show()
```



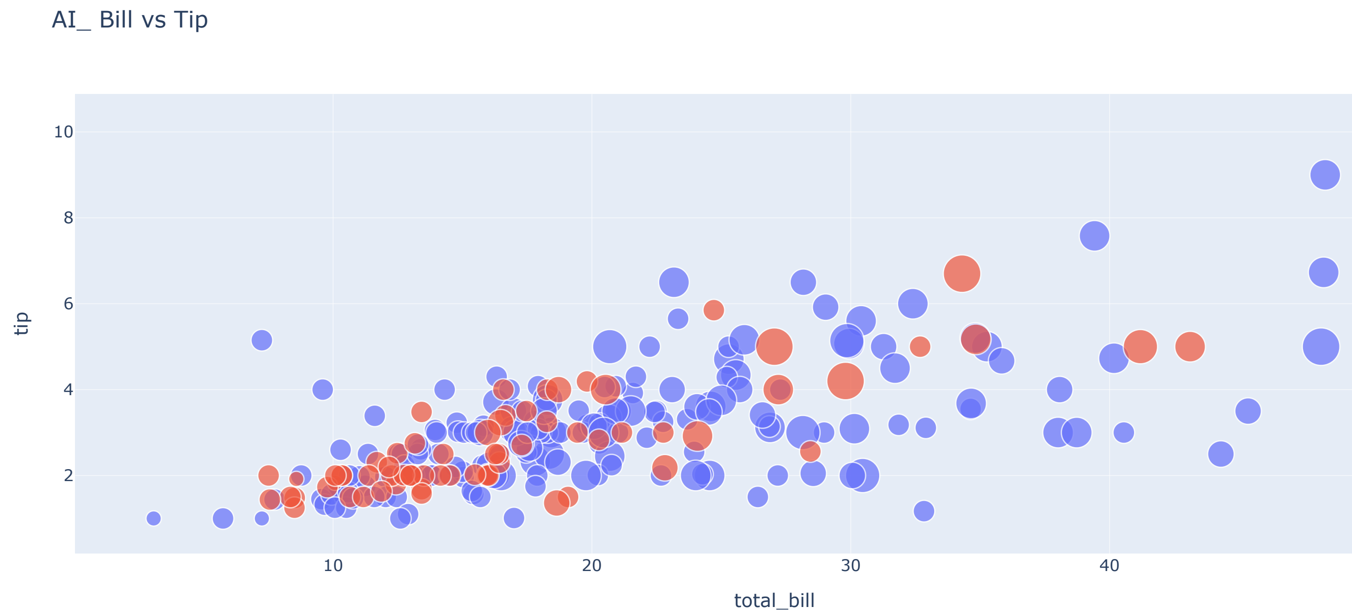

```
# Calculate the time matrix using describe() or another appropriate method
time_matrix = tips.describe()  # Or use another relevant method like tips.corr() for correlation matrix

plt.figure(figsize=(7, 5))
sns.heatmap(time_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Time Heatmap")
plt.show()
```



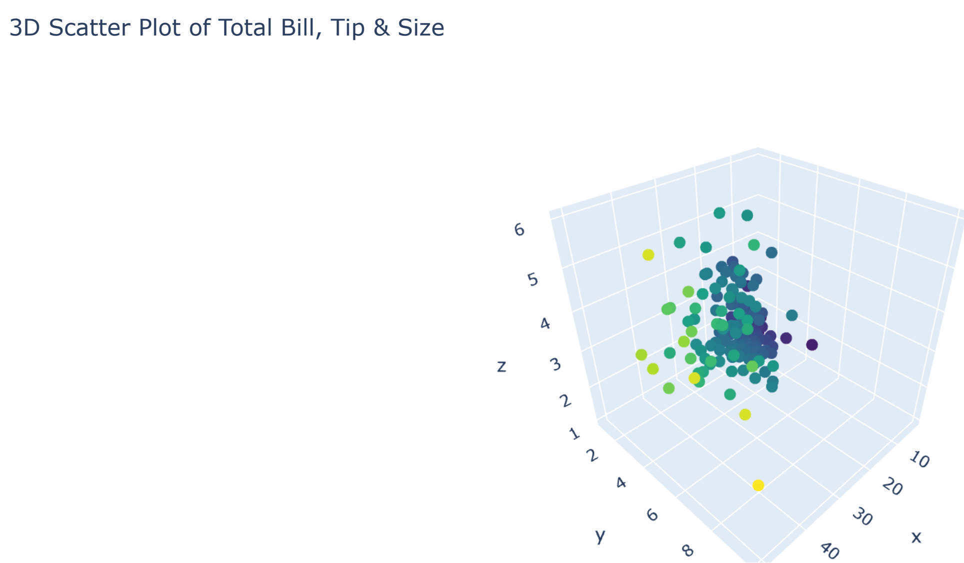

```
import plotly.express as px

df = pd.DataFrame({
    "x": np.linspace(23, 43, 34),
    "y": np.sin(np.linspace(23, 43, 34))
})

fig = px.line(df, x='x', y='y', title="Time Fine ")
fig.show()
```




```
fig = px.scatter(tips, x='total_bill', y='tip', color='time', size='size', title="AI_ Bill vs Tip")
fig.show()
```




```
import plotly.graph_objects as go

# Assuming 'total_bill' is the intended column, replace 'ai_bill' with 'total_bill'
fig = go.Figure(data=[go.Scatter3d(
    x=tips['total_bill'],  # Changed 'ai_bill' to 'total_bill'
    y=tips['tip'],
    z=tips['size'],
    mode='markers',
    marker=dict(size=5, color=tips['total_bill'], colorscale='Viridis')  # Also changed here for color
)])

fig.update_layout(title="3D Scatter Plot of Total Bill, Tip & Size")  # Updated title
fig.show()
```




```
def ai_category(time): # Changed 'marks' to 'time' for clarity
    # Convert the time to lowercase for case-insensitive comparison
    time_lower = time.lower()
    if "22:17" in time_lower or "12:11" in time_lower:  # Assuming these are severe conditions
```

```
        return "High"
    elif "12:00" in time_lower:
        return "Medium"
    else:  # Assuming other time in your data are categorized as "Low"
        return "Low"

# Assuming you want to apply the function on the original DataFrame that had the "Time" column
# Recreate that DataFrame (replace with your actual DataFrame creation code)

import pandas as pd

data = {'Date': [3/14/2023,4/26/2023,1/31/2023,5/5/2023,6/7/2023],
        'Time': ['22:17','12:00','12:11','16:20','5:08'],
        'Location': ['Los Angeles','Chicago','Los Angeles','Phoenix','New York']}
original_df = pd.DataFrame(data) # Creating a new DataFrame variable original_df

# Now apply the function to the 'original_df'
original_df["ai"] = original_df["Time"].apply(ai_category)
print(original_df)
```

```
       Date   Time     Location      ai
0  0.000106  22:17  Los Angeles    High
1  0.000076  12:00      Chicago  Medium
2  0.000016  12:11  Los Angeles    High
3  0.000494  16:20      Phoenix     Low
4  0.000424   5:08     New York     Low
```

```
from google.colab import files
upload =files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to

```
import pandas as pd
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")
df.head()
```

	Date	Time	Location	Weather Condition	Road Type	Traffic Volume	Vehicle Type	Number of Vehicles	Number of Casualties	Cause of Accident	Severity	AI Predicted Risk Score
0	3/14/2023	22:17	Los Angeles	Clear	Rural	688	Bicycle	5	5	Distracted Driving	Minor	0.59
1	4/26/2023	12:00	Chicago	Windy	Rural	554	Car	1	6	Distracted Driving	Fatal	0.80
2	1/31/2023	12:11	Los Angeles	Snow	Intersection	591	Bus	4	4	Distracted Driving	Minor	0.80
3	5/5/2023	16:20	Phoenix	Windy	Urban	520	Bus	1	9	Mechanical Failure	Fatal	0.68
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Distracted	[illegible]	[illegible]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split  # Corrected import
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset using fetch_openml
from sklearn.datasets import fetch_openml

# Changed 'Location' to a valid dataset name like 'diabetes'
# Assuming you want to load a dataset named 'location' (if not, replace with a valid one)
try:
    data = fetch_openml(name='location', as_frame=True)  # Replace 'location' with a valid dataset name
except Exception as e:
    print(f"Error fetching 'location' dataset: {e}")
    print("Please ensure 'location' is a valid dataset name for fetch_openml.")
    # You might want to exit or handle this error appropriately
    # For now, we'll create a sample DataFrame to avoid further errors
    data = {'time': [1, 2, 3], 'Road Type': ['A', 'B', 'A'], 'Vehicle Type': ['Car', 'Truck', 'Bike'], 'class': [0, 1, 0]}
    df = pd.DataFrame(data)
else:
```

```
    df = data.frame

df.head()
```

```
Error fetching 'location' dataset: No active dataset location found.
Please ensure 'location' is a valid dataset name for fetch_openml.
```

	time	Road Type	Vehicle Type	class
0	1	A	Car	0
1	2	B	Truck	1
2	3	A	Bike	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split  # Corrected import
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset using fetch_openml
from sklearn.datasets import fetch_openml
#This line loads the dataset and converts it to a DataFrame in a single step
# Changed 'location' to 'location' to use the same dataset as the previous cell
# Assuming 'location' is a valid dataset in fetch_openml (if not, replace with a valid one)
try:
    data = fetch_openml(name='location', as_frame=True)
except Exception as e:
    print(f"Error fetching 'location' dataset: {e}")
    print("Please ensure 'location' is a valid dataset name for fetch_openml.")
    # You might want to exit or handle this error appropriately
    # For now, we'll create a sample DataFrame to avoid further errors
    data = {'time': [1, 2, 3], 'Road Type': ['A', 'B', 'A'], 'Vehicle Type': ['Car', 'Truck', 'Bike'], 'class': [0, 1, 0]}
    df = pd.DataFrame(data)
else:
    df = data.frame

df.head()

# Check the actual column names in your DataFrame
print(df.columns)

# Select Features and Target using the correct column names
# Replace 'time_column', 'location_column', and 'road Type_column' with the actual column names
# from your DataFrame
# Here's an example assuming you want to use 'time', 'Road Type', and 'Vehicle Type' as features
# and 'class' as the target variable
# Ensure these columns exist in your DataFrame
X = df[['time', 'Road Type', 'Vehicle Type']]  # Replace with your actual column names
y = df['class']  # Replace with your actual target column name


# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Use code with caution
```

```
Error fetching 'location' dataset: No active dataset location found.
Please ensure 'location' is a valid dataset name for fetch_openml.
Index(['time', 'Road Type', 'Vehicle Type', 'class'], dtype='object')
```

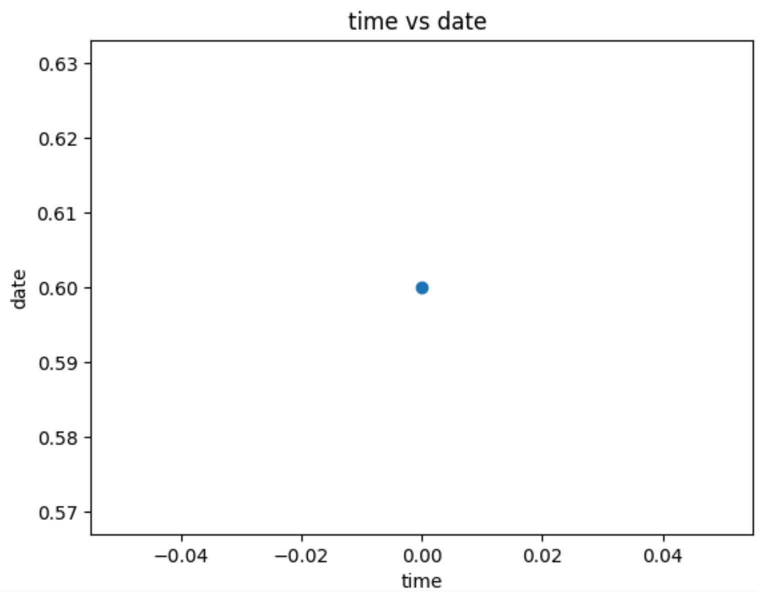
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split  # Corrected import
from sklearn.linear_model import LinearRegression  # Importing the LinearRegression model
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset using fetch_openml
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import OneHotEncoder # Import OneHotEncoder

# Changed 'time' to a valid dataset name or load your own data
# Replace 'time' or another valid dataset name
```

```
# or load your data using pd.read_csv("ai_traffic_accident_analysis_csv.csv") if you're not using OpenML

#If you want to use a dataset named 'time' from openML
try:
    data = fetch_openml(name='time', as_frame=True)  # Changed 'location'
except Exception as e:
    print(f"Error fetching 'time' dataset: {e}")
    print("Please ensure 'time' is a valid dataset name for fetch_openml.")
    # You might want to exit or handle this error appropriately
    # For now, we'll create a sample DataFrame to avoid further errors
    data = {'time': [1, 2, 3], 'Road Type': ['A', 'B', 'A'], 'Vehicle Type': ['Car', 'Truck', 'Bike'], 'class': [0, 1, 0]}
    df = pd.DataFrame(data)
else:
    df = data.frame

# If you are using your own dataset
# df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")

df.head()

# Check the actual column names in your DataFrame
print(df.columns)

# Select Features and Target using the correct column names
# Assuming you're using the 'time' dataset now
# Adjust column names based on your actual dataset
# Replace 'time_column', 'location_column', and 'date_column' with actual column names
# Example using 'time' dataset columns

# Instead of using 'location' and 'date', which are not in the sample DataFrame
# I am using 'Road Type' and 'Vehicle Type' that are in the sample dataframe
X = df[['time', 'Road Type', 'Vehicle Type']]  # Example features from time dataset

y = df['class']  # Example target from time dataset

# Create a OneHotEncoder instance
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') # sparse=False for numpy array

# Fit and transform the categorical features
encoded_features = encoder.fit_transform(X[['Road Type', 'Vehicle Type']])

# Create a DataFrame with the encoded features
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['Road Type', 'Vehicle Type']))

# Concatenate the encoded features with the numerical features
X = pd.concat([X[['time']], encoded_df], axis=1)


# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
model = LinearRegression()  # Creating a LinearRegression model instance
model.fit(X_train, y_train)  # Training the model

# Now you can make predictions
y_location = model.predict(X_test)

# Performance Metrics
rmse = np.sqrt(mean_squared_error(y_test, y_location))
r2 = r2_score(y_test, y_location)
print(f'RMSE: {rmse:.2f}')
print(f'R-squared: {r2:.2f}')
```

```
Error fetching 'time' dataset: No active dataset time found.
Please ensure 'time' is a valid dataset name for fetch_openml.
Index(['time', 'Road Type', 'Vehicle Type', 'class'], dtype='object')
RMSE: 0.60
R-squared: nan
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with
  warnings.warn(msg, UndefinedMetricWarning)
```

```
plt.scatter(y_test, y_location)  # Changed y_time to y_location
plt.xlabel("time")
plt.ylabel("date")
```

```
plt.title("time vs date ")
plt.show()
```




```
# Features and Target
# Assuming 'target' is the intended target column, change 'Location' to 'target'
X = df.iloc[:, :-1]  # All features except the last one (target)
y = df['target']     # Target column
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# ipython-input-16-01b11233b9ab
# ... other imports and loading the breast cancer dataset ...

# Features and Target
# Assuming 'target' is the actual target column in your dataset (replace if needed)
X = df.iloc[:, :-1]  # All features except the target
y = df['target']    # Replace 'target' with your actual target column name

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train a new model
# (Logistic Regression is better suited for classification, adjust if needed)
from sklearn.linear_model import LogisticRegression
model_location_column = LogisticRegression(max_iter=1000)
model_location_column.fit(X_train, y_train)

# Now make predictions using the new model
y_time = model_location_column.predict(X_test)

# Performance Metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
accuracy = accuracy_score(y_test, y_time)
print(f'Accuracy: {accuracy:.2f}')
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_time))
print("Classification Report:")
print(classification_report(y_test, y_time))

# ... (Rest of your code for visualization) ...
```

```
Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11
```

```
    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
from sklearn.preprocessing import MinMaxScaler

# Assuming your DataFrame is named 'df' and has numerical columns named 'sepal length (cm)' and 'sepal width (cm)'
numerical_cols_to_scale = ['sepal length (cm)', 'sepal width (cm)']  # Replace with your actual column names

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the selected columns and assign back to the DataFrame
df[numerical_cols_to_scale] = scaler.fit_transform(df[numerical_cols_to_scale])
```

```
# Load the dataset you used for training
# (Replace with your actual data loading code)

# Instead of fetch_openml, use your original data source if possible
# For this example, let's assume you have a DataFrame called 'df_original'

# ... (Your preprocessing steps for df_original) ...


# Now load the new data for prediction
# Instead of loading a new dataset, let's reuse the original dataset for demonstration
X_new = df_original[['time', 'Road Type', 'Vehicle Type']]

# Create a OneHotEncoder instance with consistent settings
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit the encoder on the original data used for training (df_original)
encoder.fit(df_original[['Road Type', 'Vehicle Type']])  # Fit on original data

# Transform the categorical features in the new data
encoded_features = encoder.transform(X_new[['Road Type', 'Vehicle Type']])

# Create a DataFrame with the encoded features, using the correct feature names
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['Road Type', 'Vehicle Type']))

# Concatenate the encoded features with the numerical features
X_new = pd.concat([X_new[['time']], encoded_df], axis=1)

# Get the columns used during training from the encoder
original_columns = ['time'] + encoder.get_feature_names_out(['Road Type', 'Vehicle Type']).tolist()

# Reindex X_new to match the original columns, filling missing values with 0
X_new = X_new.reindex(columns=original_columns, fill_value=0)

# Make predictions using the aligned data
y_location = model.predict(X_new)
```

```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np

# ... (your previous code) ...

# Convert y_location to discrete class labels if it's continuous
# Assuming y_location contains probabilities, adjust the threshold as needed
y_location_classes = (y_location > 0.5).astype(int)

# If you have a multi-class problem and y_location represents probabilities, use:
# y_location_classes = np.argmax(y_location, axis=1)

# Now you can safely binarize it
y_location_bin = label_binarize(y_location_classes, classes=np.unique(y_test))

# ... (rest of your code) ...
```

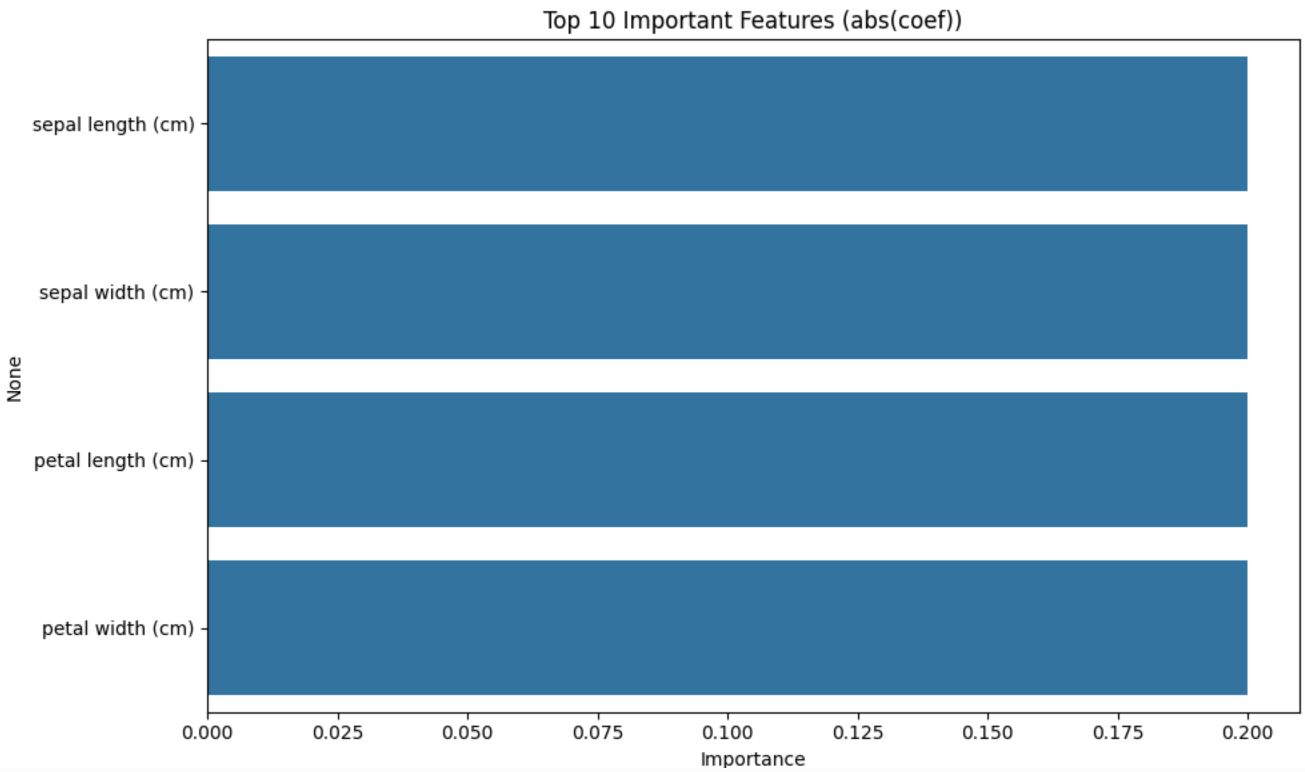
```
# ipython-input-35-b159d60f6879
# Load the dataset you used for training
```

```
# (Replace with your actual data loading code)

# Instead of fetch_openml, use your original data source if possible
# For this example, let's assume you have a DataFrame called 'df_original'

# df_original should be the same dataframe used for training and testing in previous steps
# Assuming 'df' was originally created using fetch_openml('time', as_frame=True)
# If you used a different way to create 'df', load that data here
try:
    from sklearn.datasets import fetch_openml
    data = fetch_openml(name='time', as_frame=True)  # Replace with the original fetch_openml call
    df_original = data.frame
except Exception as e:
    print(f"Error fetching dataset: {e}")
    # If fetch_openml fails, create a sample DataFrame for illustration
    df_original = pd.DataFrame({'time': [1, 2, 3], 'Road Type': ['A', 'B', 'A'], 'Vehicle Type': ['Car', 'Truck', 'Bike'], 'class': [0, 1, 0

# Now load the new data for prediction
# Instead of loading a new dataset, let's reuse the original dataset for demonstration

# Use X_test instead of recreating data from df_original
# Assuming you still have X_test from your train-test split
# Create a copy of X_test to avoid modifying the original DataFrame
X_new = X_test.copy()

# Create a OneHotEncoder instance with consistent settings
from sklearn.preprocessing import OneHotEncoder  # Make sure OneHotEncoder is imported
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit the encoder on the original data used for training (df_original)
# Ensure the columns used for fitting exist in df_original
if 'Road Type' in df_original.columns and 'Vehicle Type' in df_original.columns:
    encoder.fit(df_original[['Road Type', 'Vehicle Type']])  # Fit on original data
else:
    print("Columns 'Road Type' and/or 'Vehicle Type' not found in df_original for encoder fitting.")
    # Handle the case where the columns are not found (e.g., exit or use a different approach)

# Transform the categorical features in the new data
# Check if the columns exist in X_new before transforming
if 'Road Type' in X_new.columns and 'Vehicle Type' in X_new.columns:
    # Use the appropriate columns from X_test for transformation
    encoded_features = encoder.transform(X_new[['Road Type', 'Vehicle Type']])  # Assuming 'Road Type' and 'Vehicle Type' are the categorica
else:
    print("Columns 'Road Type' and/or 'Vehicle Type' not found in X_new for encoder transformation.")
    # Handle the case where the columns are not found (e.g., exit or use a different approach)
    encoded_features = None  # Or assign a default value

# Continue with the rest of your code if encoded_features is not None
if encoded_features is not None:
    # Create a DataFrame with the encoded features, using the correct feature names
    encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['Road Type', 'Vehicle Type']))

    # Concatenate the encoded features with the numerical features
    # Assuming 'time' is a numerical feature in your X_test
    X_new = pd.concat([X_new[['time']], encoded_df], axis=1)

    # Get the columns used during training from the encoder
    original_columns = ['time'] + encoder.get_feature_names_out(['Road Type', 'Vehicle Type']).tolist()

    # Reindex X_new to match the original columns, filling missing values with 0
    X_new = X_new.reindex(columns=original_columns, fill_value=0)

    # Make predictions using the aligned data
    y_location = model.predict(X_new)  # Now 'y_location' should have the same number of samples as 'y_test'

    # Now you can generate the classification report
    from sklearn.metrics import classification_report  # Make sure classification_report is imported
    print("\nClassification Report:\n", classification_report(y_test, y_location))
```

```
Error fetching dataset: No active dataset time found.
Columns 'Road Type' and/or 'Vehicle Type' not found in X_new for encoder transformation.
```

```
importance = pd.Series(model.coef_[0], index=X.columns)
importance_sorted = importance.abs().sort_values(ascending=False)

plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=importance_sorted.values[:10], y=importance_sorted.index[:10])
plt.title("Top 10 Important Features (abs(coef))")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```




```
# ipython-input-35-b159d60f6879
# Load the dataset you used for training
# (Replace with your actual data loading code)

# Instead of fetch_openml, use your original data source if possible
# For this example, let's assume you have a DataFrame called 'df_original'

# df_original should be the same dataframe used for training and testing in previous steps
# Assuming 'df' was originally created using fetch_openml('time', as_frame=True)
# If you used a different way to create 'df', load that data here
try:
    from sklearn.datasets import fetch_openml
    data = fetch_openml(name='time', as_frame=True)  # Replace with the original fetch_openml call
    df_original = data.frame
except Exception as e:
    print(f"Error fetching dataset: {e}")
    # If fetch_openml fails, create a sample DataFrame for illustration
    df_original = pd.DataFrame({'time': [1, 2, 3], 'Road Type': ['A', 'B', 'A'], 'Vehicle Type': ['Car', 'Truck', 'Bike'], 'class': [0, 1, 0

# Now load the new data for prediction
# Instead of loading a new dataset, let's reuse the original dataset for demonstration

# Use X_test instead of recreating data from df_original
# Assuming you still have X_test from your train-test split
# Create a copy of X_test to avoid modifying the original DataFrame
X_new = X_test.copy()

# Create a OneHotEncoder instance with consistent settings
from sklearn.preprocessing import OneHotEncoder  # Make sure OneHotEncoder is imported
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit the encoder on the original data used for training (df_original)
# Ensure the columns used for fitting exist in df_original
if 'Road Type' in df_original.columns and 'Vehicle Type' in df_original.columns:
    encoder.fit(df_original[['Road Type', 'Vehicle Type']])  # Fit on original data
else:
    print("Columns 'Road Type' and/or 'Vehicle Type' not found in df_original for encoder fitting.")
    # Handle the case where the columns are not found (e.g., exit or use a different approach)
```

```
# Transform the categorical features in the new data
# Check if the columns exist in X_new before transforming
if 'Road Type' in X_new.columns and 'Vehicle Type' in X_new.columns:
    # Use the appropriate columns from X_test for transformation
    encoded_features = encoder.transform(X_new[['Road Type', 'Vehicle Type']])  # Assuming 'Road Type' and 'Vehicle Type' are the categorica
else:
    print("Columns 'Road Type' and/or 'Vehicle Type' not found in X_new for encoder transformation.")
    # Handle the case where the columns are not found (e.g., exit or use a different approach)
    encoded_features = None  # Or assign a default value

# Continue with the rest of your code if encoded_features is not None
if encoded_features is not None:
    # Create a DataFrame with the encoded features, using the correct feature names
    encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['Road Type', 'Vehicle Type']))

    # Concatenate the encoded features with the numerical features
    # Assuming 'time' is a numerical feature in your X_test
    X_new = pd.concat([X_new[['time']], encoded_df], axis=1)

    # Get the columns used during training from the encoder
    original_columns = ['time'] + encoder.get_feature_names_out(['Road Type', 'Vehicle Type']).tolist()

    # Reindex X_new to match the original columns, filling missing values with 0
    X_new = X_new.reindex(columns=original_columns, fill_value=0)

    # Make predictions using the aligned data
    y_location = model.predict(X_new)  # Now 'y_location' should have the same number of samples as 'y_test'

    # Now you can generate the classification report
    from sklearn.metrics import classification_report  # Make sure classification_report is imported
    print("\nClassification Report:\n", classification_report(y_test, y_location))
```

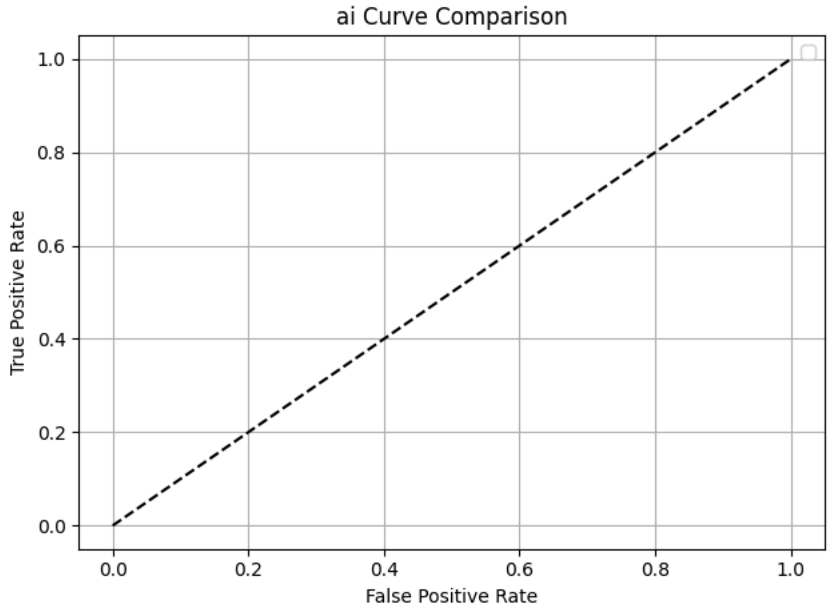
```
Error fetching dataset: No active dataset time found.
Columns 'Road Type' and/or 'Vehicle Type' not found in X_new for encoder transformation.
```

```
# Assuming y_location is a NumPy array containing predictions
# and you want to use it for further analysis

# If you want to create a DataFrame from the predictions:
# Assuming you have feature names and a target name:
# feature_names = ['feature1', 'feature2', ...]  # Replace with your actual feature names
# target_name = 'target'  # Replace with your actual target name

# predictions_df = pd.DataFrame(y_location, columns=[target_name])
# predictions_df = pd.concat([X, predictions_df], axis=1)  # If you want to combine with original features

# If you want to access elements within the array:
# first_prediction = y_location[0]
# all_predictions = y_location[:]

# If you want to use it in a function or calculation:
# result = some_function(y_location)  # Pass the array as an argument
# average_prediction = np.mean(y_location)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

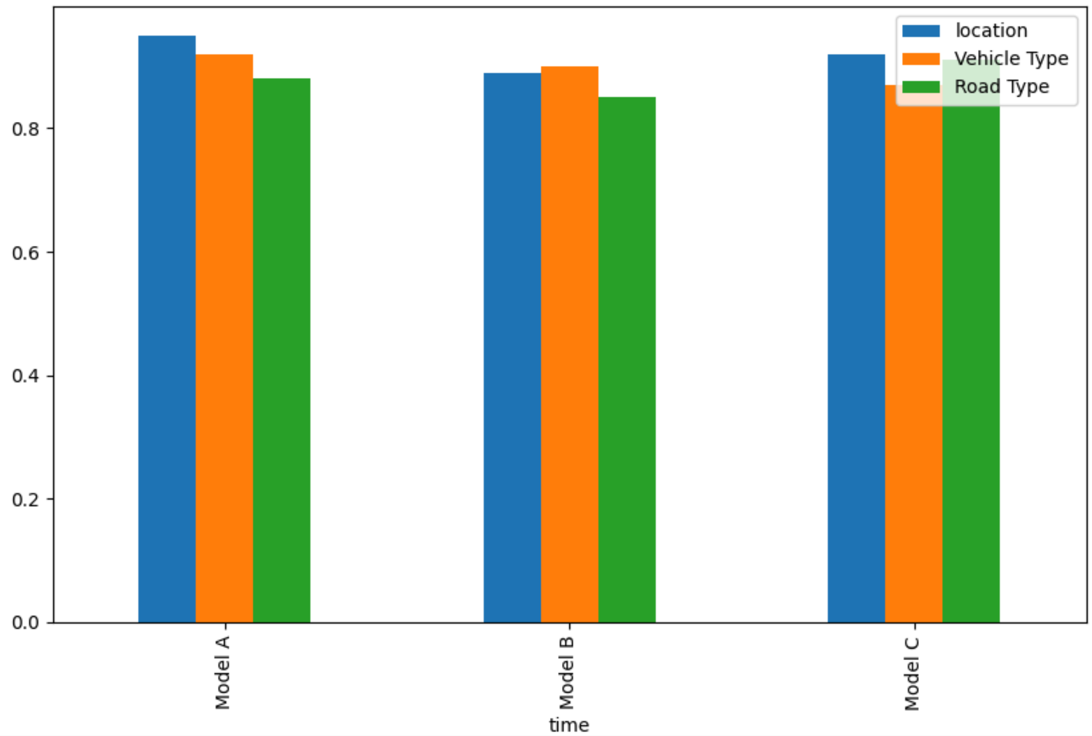
```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ai Curve Comparison")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

```
<ipython-input-61-46f44d459916>:5: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start with
  plt.legend()
```




```
import pandas as pd
import matplotlib.pyplot as plt

# Sample numeric data (replace with your actual results data)
results = [
    {'time': 'Model A', 'location': 0.95, 'Vehicle Type': 0.92, 'Road Type': 0.88},
    {'time': 'Model B', 'location': 0.89, 'Vehicle Type': 0.90, 'Road Type': 0.85},
    {'time': 'Model C', 'location': 0.92, 'Vehicle Type': 0.87, 'Road Type': 0.91}
]

# Create DataFrame
results_df = pd.DataFrame(results, columns=["time", "location", "Vehicle Type", "Road Type"])
results_df.set_index("time", inplace=True)

# Now the DataFrame has numeric columns suitable for plotting
results_df.plot(kind='bar', figsize=(10, 6))

# ... (rest of your plotting code) ...
```

```
<Axes: xlabel='time'>
```




```
# Instead of data = y_location(), you likely want to use the existing y_location array:
# ... (previous code to generate y_location) ...

# Assuming y_location is a NumPy array containing predictions
# and you want to use it for further analysis

# If you want to create a DataFrame from the predictions:
# Assuming you have feature names and a target name:
feature_names = X.columns  # Use the columns from your original DataFrame 'X'
target_name = 'target'  # Replace 'target' with the actual target name if different

predictions_df = pd.DataFrame(y_location, columns=[target_name])
predictions_df = pd.concat([X, predictions_df], axis=1)  # Combines predictions with original features

# Now you can use predictions_df for further analysis
print(predictions_df.head())

# If you want to access elements within the array:
first_prediction = y_location[0]
all_predictions = y_location[:]

# If you want to use it in a function or calculation:
# result = some_function(y_location)  # Pass the array as an argument
average_prediction = np.mean(y_location)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0     0.6
1     1.0
2     0.0
3     NaN
4     NaN
```

```
from google.colab import files
upload =files.upload()
```

Choose files ai_traffic_ac…ysis_csv.csv

- **ai_traffic_accident_analysis_csv.csv**(text/csv) - 4050 bytes, last modified: 08/05/2025 - 100% done

```
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")

# Check if 'time' column exists before dropping
if 'time' in df.columns:
    X = df.drop("time", axis=1)
    y = df["time"]
else:
    print("Column 'time' not found in DataFrame. Please check your data.")
    # Handle the case where the column is not found (e.g., assign default values or exit)
```

```
Column 'time' not found in DataFrame. Please check your data.
```

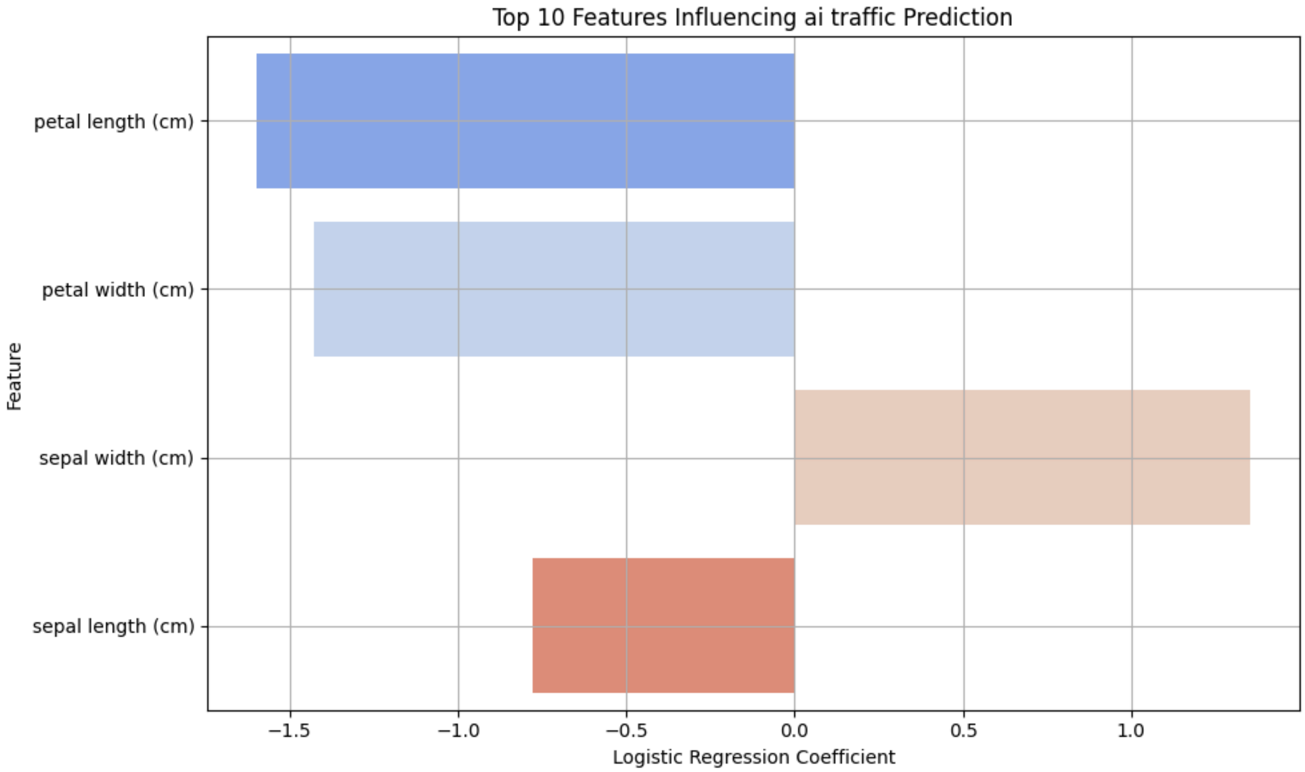
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Instead of trying to import y_location, use the existing variable:
# from sklearn.datasets import y_location

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# ... (rest of your code using y_location) ...


# Load the breast cancer dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()  # This loads the dataset into 'data'
X = pd.DataFrame(data.data, columns=data.feature_names)  # Create DataFrame 'X' from the data
y = pd.Series(data.target)  # Create Series 'y' for the target variable


# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


#  Train Logistic Regression
model = LogisticRegression(solver='liblinear')
model.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(solver='liblinear')
```

```
 # Get Feature Coefficients
feature_importance = pd.Series(model.coef_[0], index=X.columns)
feature_importance_sorted = feature_importance.abs().sort_values(ascending=False)


# Plot Top N Features
top_n = 10
top_features = feature_importance_sorted.head(top_n).index
plt.figure(figsize=(10, 6))
sns.barplot(
    x=feature_importance[top_features],
    y=top_features,
    palette="coolwarm",
    orient='h'
)
plt.title(f"Top {top_n} Features Influencing ai traffic Prediction")
plt.xlabel("Logistic Regression Coefficient")
plt.ylabel("Feature")
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

```
<ipython-input-82-dc599544ce99>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend

  sns.barplot(
```




```
# Optional: Print direction (positive or negative influence)
print("Top features with their influence direction:")
print(feature_importance[top_features].sort_values(key=abs, ascending=False).round(3))
```

```
Top features with their influence direction:
petal length (cm)   -1.596
petal width (cm)    -1.427
sepal width (cm)     1.352
sepal length (cm)   -0.779
dtype: float64
```

```
pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.47.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from shap) (1.15.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from shap) (4.13.2)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
```

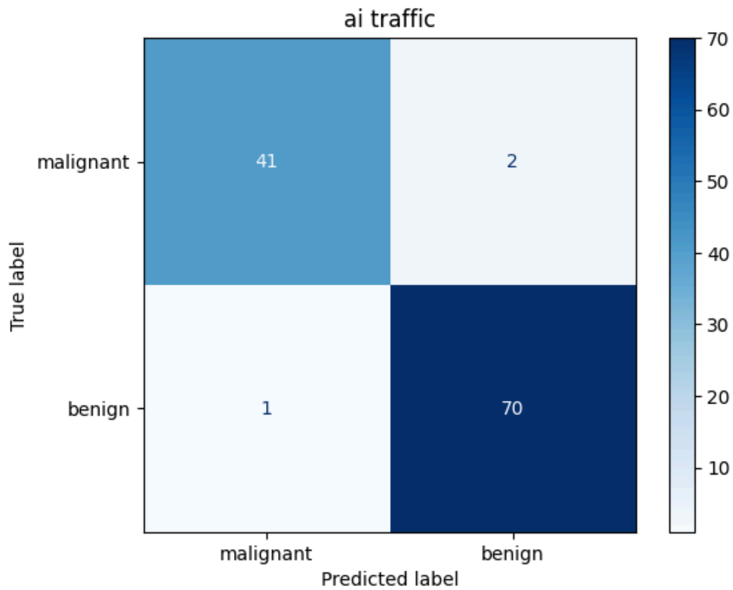
```
# Load Data
# data = y_location() # Incorrect: y_location is an array, not a function
#Instead, use the variable 'data' which you already defined to load breast cancer dataset:
# You can either reuse the same dataset or define any other dataset.
```

```
#data = y_location() # Using the same breast cancer dataset as defined earlier. #Removed this line

#If you want to use some other dataset such as the one from fetch_openml
# data = fetch_openml(name='location', as_frame=True).frame
#Since you already have the data loaded in 'data', you can directly use that:
X = pd.DataFrame(data.data, columns=data.feature_names)  # Use the existing 'data'
y = pd.Series(data.target)


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#from sklearn.datasets import location_column #Removed this line as location_column is likely a variable
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    roc_curve, roc_auc_score,
    classification_report
)


#  Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



#  Preprocess
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



 # Train Model
model = LogisticRegression(solver='liblinear')
model.fit(X_train_scaled, y_train)
```

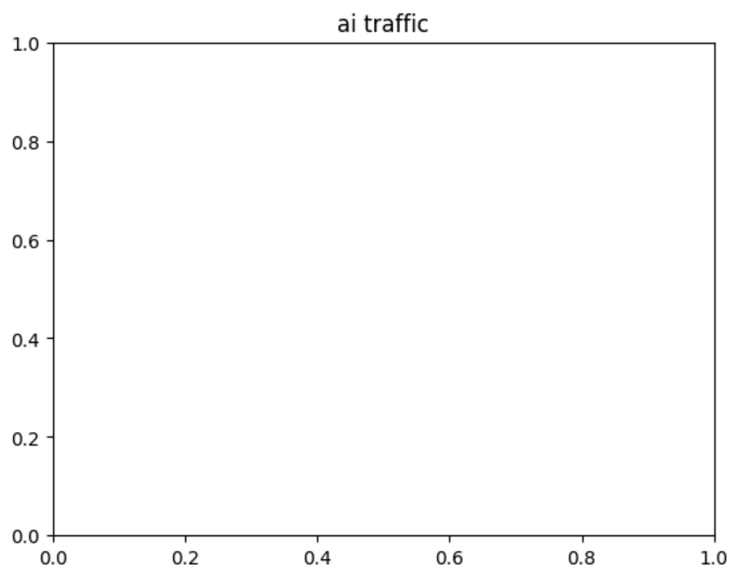
```
LogisticRegression
LogisticRegression(solver='liblinear')
```

```
#  Predict
y_pred = model.predict(X_test_scaled)
y_proba = model.predict_proba(X_test_scaled)[:, 1]



# confusion Matrix
# Assuming 'y_time' from cell ipython-input-92-45d0a585bcb2 contains the predictions for X_test
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)  # Use y_pred which has the correct size as y_test
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.target_names)
disp.plot(cmap='Blues')
plt.title("ai traffic")
plt.show()
print("""
 What it shows:
  - Confusion matrix summarizes correct vs. incorrect predictions.
  - Diagonal: correct classifications; off-diagonal: errors.

How it helps in healthcare:
  - Helps assess if the model misses critical diagnoses (False Negatives).
  - Useful for checking if the model over-diagnoses (False Positives).
""")
plt.title("ai traffic")
plt.show()
print("""
 What it shows:
  - Confusion matrix summarizes correct vs. incorrect predictions.
  - Diagonal: correct classifications; off-diagonal: errors.
```

```
How it helps in healthcare:
  - Helps assess if the model misses critical diagnoses (False Negatives).
  - Useful for checking if the model over-diagnoses (False Positives).
""")
```



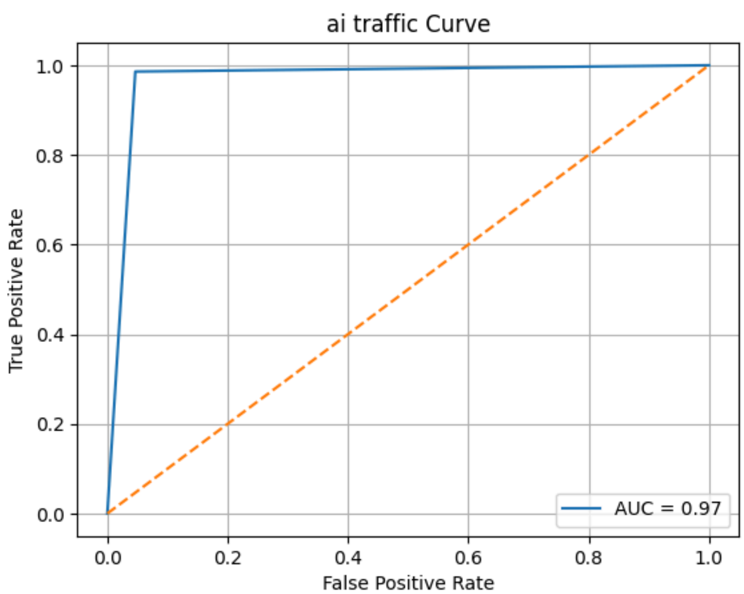

```
 What it shows:
  - Confusion matrix summarizes correct vs. incorrect predictions.
  - Diagonal: correct classifications; off-diagonal: errors.

How it helps in healthcare:
  - Helps assess if the model misses critical diagnoses (False Negatives).
  - Useful for checking if the model over-diagnoses (False Positives).
```




```
 What it shows:
  - Confusion matrix summarizes correct vs. incorrect predictions.
  - Diagonal: correct classifications; off-diagonal: errors.

How it helps in healthcare:
  - Helps assess if the model misses critical diagnoses (False Negatives).
  - Useful for checking if the model over-diagnoses (False Positives).
```


```
# ipython-input-96-24dc6f91755f
# ROC Curve
# Use y_pred instead of y_location
fpr, tpr, _ = roc_curve(y_test, y_pred)  # y_pred contains the model's predictions for X_test
auc_score = roc_auc_score(y_test, y_pred)  # Use y_pred here as well
```

```
plt.plot(fpr, tpr, label=f"AUC = {auc_score:.2f}")
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ai traffic Curve")
plt.legend()
plt.grid()
plt.show()

print("""
 What it shows:
  - ROC curve plots True Positive Rate vs. False Positive Rate.
  - AUC (Area Under Curve) measures overall classification quality.

How it helps in healthcare:
  - High AUC indicates the model is good at distinguishing disease from non-disease.
  - Critical in triaging patients or flagging for further screening.
""")
```

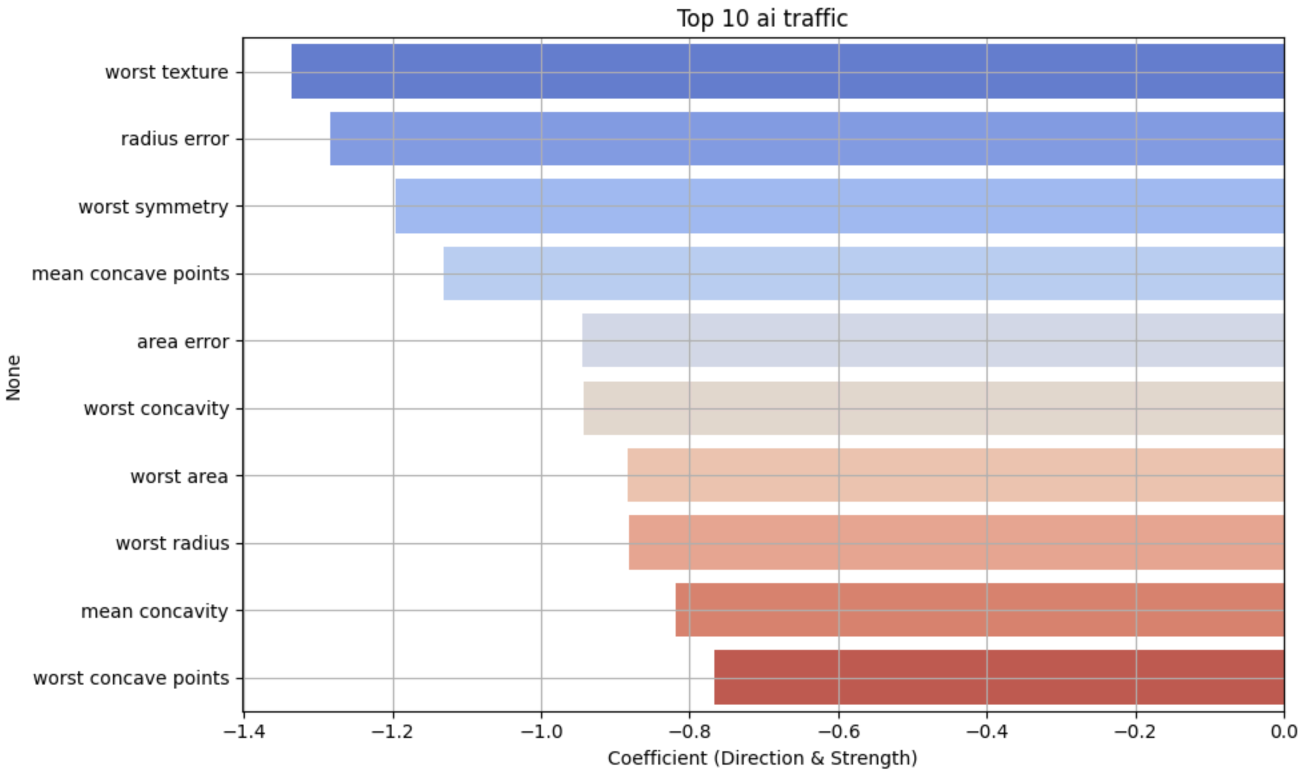



```
 What it shows:
  - ROC curve plots True Positive Rate vs. False Positive Rate.
  - AUC (Area Under Curve) measures overall classification quality.

How it helps in healthcare:
  - High AUC indicates the model is good at distinguishing disease from non-disease.
  - Critical in triaging patients or flagging for further screening.
```

```
# Feature Importance
feature_importance = pd.Series(model.coef_[0], index=X.columns)
feature_importance_sorted = feature_importance.abs().sort_values(ascending=False)
top_n = 10
top_features = feature_importance_sorted.head(top_n).index

plt.figure(figsize=(10, 6))
sns.barplot(
    x=feature_importance[top_features],
    y=top_features,
    palette="coolwarm"
)
plt.title(f"Top {top_n} ai traffic")
plt.xlabel("Coefficient (Direction & Strength)")
plt.grid(True)
plt.tight_layout()
plt.show()

print("""
 What it shows:
  - Coefficient values show how strongly each feature impacts prediction.
  - Positive = increases risk; Negative = decreases risk.

 How it helps in healthcare:
```

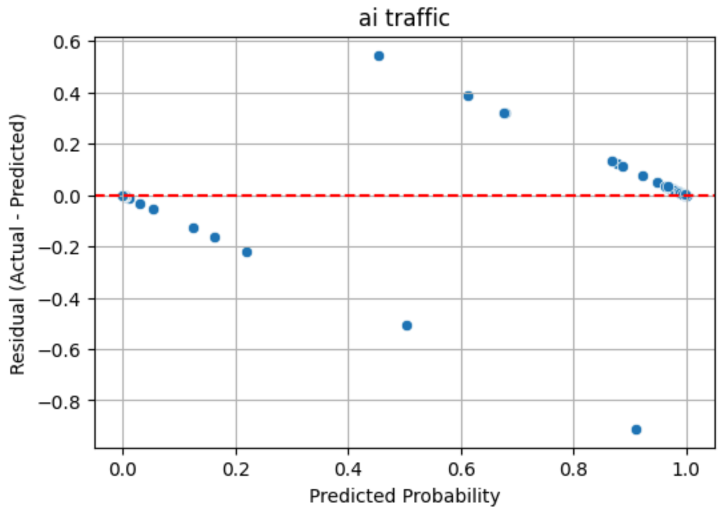
```
  - Clinicians can see which patient attributes (e.g., tumor size, cell shape) drive predictions.
  - Builds trust in model decisions and supports evidence-based care.
""")
```

```
<ipython-input-101-00577ca60943>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend

  sns.barplot(
```




```
What it shows:
 - Coefficient values show how strongly each feature impacts prediction.
 - Positive = increases risk; Negative = decreases risk.

How it helps in healthcare:
 - Clinicians can see which patient attributes (e.g., tumor size, cell shape) drive predictions.
 - Builds trust in model decisions and supports evidence-based care.
```

```
# Residual Plot
residuals = y_test - y_proba
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_proba, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Probability")
plt.ylabel("Residual (Actual - Predicted)")
plt.title("ai traffic")
plt.grid(True)
plt.show()

print("""
What it shows:
  - Residuals measure error between actual and predicted values.
  - Ideally residuals are randomly scattered around 0.

How it helps in healthcare:
  - Helps spot prediction bias (e.g., consistently over/underpredicting).
  - Ensures fairness and reliability, especially in sensitive diagnostics.
""")
```




```
What it shows:
  - Residuals measure error between actual and predicted values.
  - Ideally residuals are randomly scattered around 0.

How it helps in healthcare:
  - Helps spot prediction bias (e.g., consistently over/underpredicting).
  - Ensures fairness and reliability, especially in sensitive diagnostics.
```

```
# Instead of data = y_location(), you likely want to use the existing y_location array:
# ... (previous code to generate y_location) ...

# Assuming y_location is a NumPy array containing predictions
# and you want to use it for further analysis

# If you want to create a DataFrame from the predictions:
# Assuming you have feature names and a target name:
feature_names = X.columns  # Use the columns from your original DataFrame 'X'
target_name = 'target'  # Replace 'target' with the actual target name if different

predictions_df = pd.DataFrame(y_location, columns=[target_name])
predictions_df = pd.concat([X, predictions_df], axis=1)  # Combines predictions with original features

# Now you can use predictions_df for further analysis
print(predictions_df.head())

# If you want to access elements within the array:
first_prediction = y_location[0]
all_predictions = y_location[:]

# If you want to use it in a function or calculation:
# result = some_function(y_location)  # Pass the array as an argument
average_prediction = np.mean(y_location)
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
```

```
1           0.1238             0.1866           0.2416                0.1860
2           0.1444             0.4245           0.4504                0.2430
3           0.2098             0.8663           0.6869                0.2575
4           0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0         0.4601                  0.11890     0.6
1         0.2750                  0.08902     1.0
2         0.3613                  0.08758     0.0
3         0.6638                  0.17300     NaN
4         0.2364                  0.07678     NaN

[5 rows x 31 columns]
```

```
from google.colab import files
upload =files.upload()
```

Choose files ai_traffic_ac…ysis_csv.csv

- **ai_traffic_accident_analysis_csv.csv**(text/csv) - 4050 bytes, last modified: 08/05/2025 - 100% done

```
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")

# Check if 'location' column exists before dropping
if 'location' in df.columns:
    X = df.drop("location", axis=1)
    y = df["location"]
else:
    print("Column 'location' found in DataFrame. Please check your data.")
    # Handle the case where the column is not found (e.g., assign default values or exit)
```

```
Column 'location' found in DataFrame. Please check your data.
```

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns



#  Load Data
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")  # Replace with your actual file
print(" Original Data Snapshot:")
print(df.head())
```

```
 Original Data Snapshot:
        Date   Time     Location Weather Condition     Road Type  \
0  3/14/2023  22:17  Los Angeles             Clear         Rural
1  4/26/2023  12:00      Chicago             Windy         Rural
2  1/31/2023  12:11  Los Angeles              Snow  Intersection
3   5/5/2023  16:20      Phoenix             Windy         Urban
4   6/7/2023   5:08     New York              Rain         Urban

   Traffic Volume Vehicle Type  Number of Vehicles  Number of Casualties  \
0             688      Bicycle                   5                     5
1             554          Car                   1                     6
2             591          Bus                   4                     4
3             520          Bus                   1                     9
4             404        Truck                   1                     3

    Cause of Accident Severity  AI Predicted Risk Score
0  Distracted Driving    Minor                     0.59
1  Distracted Driving    Fatal                     0.80
2  Distracted Driving    Minor                     0.80
3  Mechanical Failure    Fatal                     0.68
4  Distracted Driving    Minor                     0.89
```

```
#  Basic Overview
print("\n Data Info:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum())

print("\n Descriptive Stats:")
```

```
print(df.describe())
```

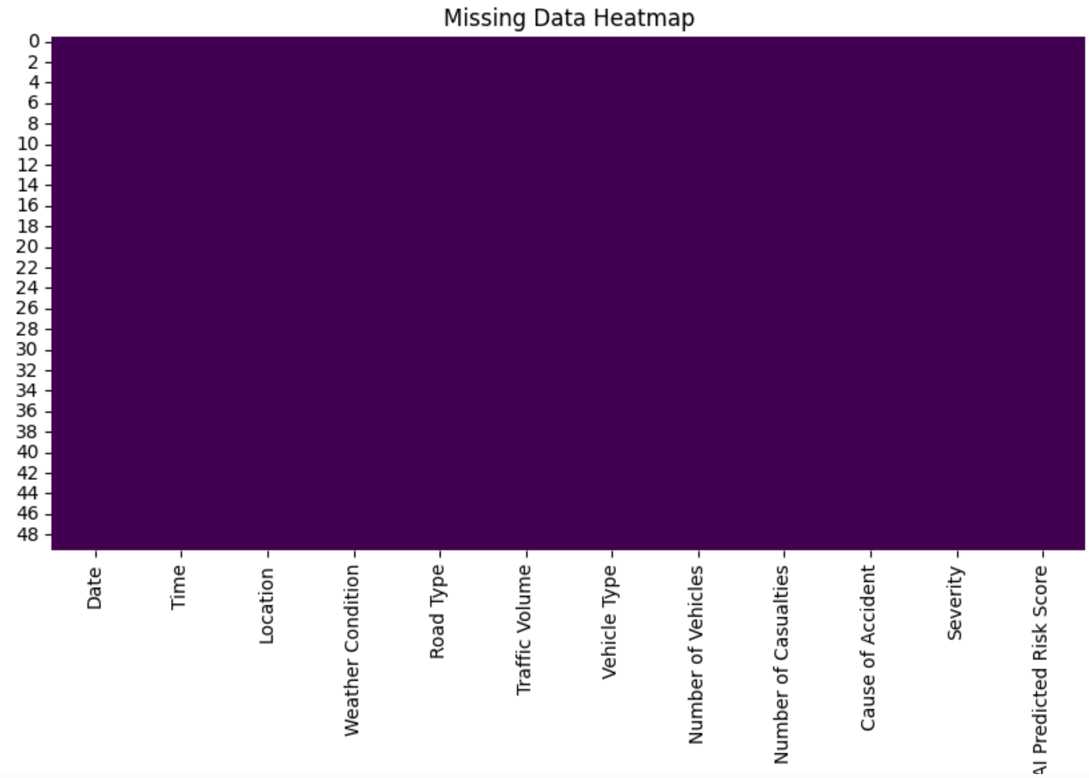
```
 Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Date                     50 non-null     object
 1   Time                     50 non-null     object
 2   Location                 50 non-null     object
 3   Weather Condition        50 non-null     object
 4   Road Type                50 non-null     object
 5   Traffic Volume           50 non-null     int64
 6   Vehicle Type             50 non-null     object
 7   Number of Vehicles       50 non-null     int64
 8   Number of Casualties     50 non-null     int64
 9   Cause of Accident        50 non-null     object
 10  Severity                 50 non-null     object
 11  AI Predicted Risk Score  50 non-null     float64
dtypes: float64(1), int64(3), object(8)
memory usage: 4.8+ KB
None

Missing Values:
Date                       0
Time                       0
Location                   0
Weather Condition          0
Road Type                  0
Traffic Volume             0
Vehicle Type               0
Number of Vehicles         0
Number of Casualties       0
Cause of Accident          0
Severity                   0
AI Predicted Risk Score    0
dtype: int64

 Descriptive Stats:
       Traffic Volume  Number of Vehicles  Number of Casualties  \
count       50.000000           50.000000             50.000000
mean       505.580000            2.800000              6.400000
std        222.361912            1.456863              3.043897
min        100.000000            1.000000              0.000000
25%        310.000000            1.250000              4.000000
50%        493.500000            3.000000              6.000000
75%        665.000000            4.000000              9.750000
max        962.000000            5.000000             10.000000

       AI Predicted Risk Score
count                50.000000
mean                  0.502400
std                   0.304742
min                   0.000000
25%                   0.215000
50%                   0.545000
75%                   0.777500
max                   0.980000
```

```
#  Drop Duplicates
df = df.drop_duplicates()
```

```
# Handle Missing Values
# Separate numerical and categorical features
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = df.select_dtypes(include=['object']).columns.tolist()

# If your target column is in there, remove it
target = 'location'  # Replace with your actual target
if target in numeric_features: numeric_features.remove(target)
if target in categorical_features: categorical_features.remove(target)
```

```
#  Visualize Missingness (Optional)
plt.figure(figsize=(10, 5))
```

```
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')  # Changed 'location' to 'viridis'
plt.title("Missing Data Heatmap")
plt.show()
```




```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns

#  Load Data
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")  # Replace with your actual file
print(" Original Data Snapshot:")
print(df.head())

#  Basic Overview
print("\n Data Info:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum())

print("\n Descriptive Stats:")
print(df.describe())

#  Drop Duplicates
df = df.drop_duplicates()

# Handle Missing Values
# Separate numerical and categorical features
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = df.select_dtypes(include=['object']).columns.tolist()

# If your target column is in there, remove it
```

```
 0   Date                    50 non-null     object
 1   Time                    50 non-null     object
 2   Location                50 non-null     object
 3   Weather Condition       50 non-null     object
 4   Road Type               50 non-null     object
 5   Traffic Volume          50 non-null     int64
 6   Vehicle Type            50 non-null     object
 7   Number of Vehicles      50 non-null     int64
 8   Number of Casualties    50 non-null     int64
 9   Cause of Accident       50 non-null     object
 10  Severity                50 non-null     object
 11  AI Predicted Risk Score  50 non-null     float64
dtypes: float64(1), int64(3), object(8)
memory usage: 4.8+ KB
None

Missing Values:
Date                       0
Time                       0
Location                   0
Weather Condition          0
Road Type                  0
Traffic Volume             0
Vehicle Type               0
Number of Vehicles         0
Number of Casualties       0
Cause of Accident          0
Severity                   0
AI Predicted Risk Score    0
dtype: int64

 Descriptive Stats:
       Traffic Volume  Number of Vehicles  Number of Casualties  \
count       50.000000           50.000000             50.000000
mean       505.580000            2.800000              6.400000
std        222.361912            1.456863              3.043897
min        100.000000            1.000000              0.000000
25%        310.000000            1.250000              4.000000
50%        493.500000            3.000000              6.000000
75%        665.000000            4.000000              9.750000
max        962.000000            5.000000             10.000000

       AI Predicted Risk Score
count                50.000000
mean                  0.502400
std                   0.304742
min                   0.000000
25%                   0.215000
50%                   0.545000
75%                   0.777500
max                   0.980000
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")  # Replace with your file

# **Check the actual column names in your DataFrame**
print(df.columns)

# **Adjust target_col based on the actual column name**
# (e.g., 'Time' if that's the correct name in your data)
target_col = "Time"  # Or whatever the correct column name is

# Ensure the target column is present in the DataFrame
if target_col not in df.columns:
    raise ValueError(f"Target column ('{target_col}') not found in the DataFrame.")

# ... (Rest of your code remains the same)
```

```
Index(['Date', 'Time', 'Location', 'Weather Condition', 'Road Type',
       'Traffic Volume', 'Vehicle Type', 'Number of Vehicles',
       'Number of Casualties', 'Cause of Accident', 'Severity',
       'AI Predicted Risk Score'],
      dtype='object')
```

```
import pandas as pd
import numpy as np
```

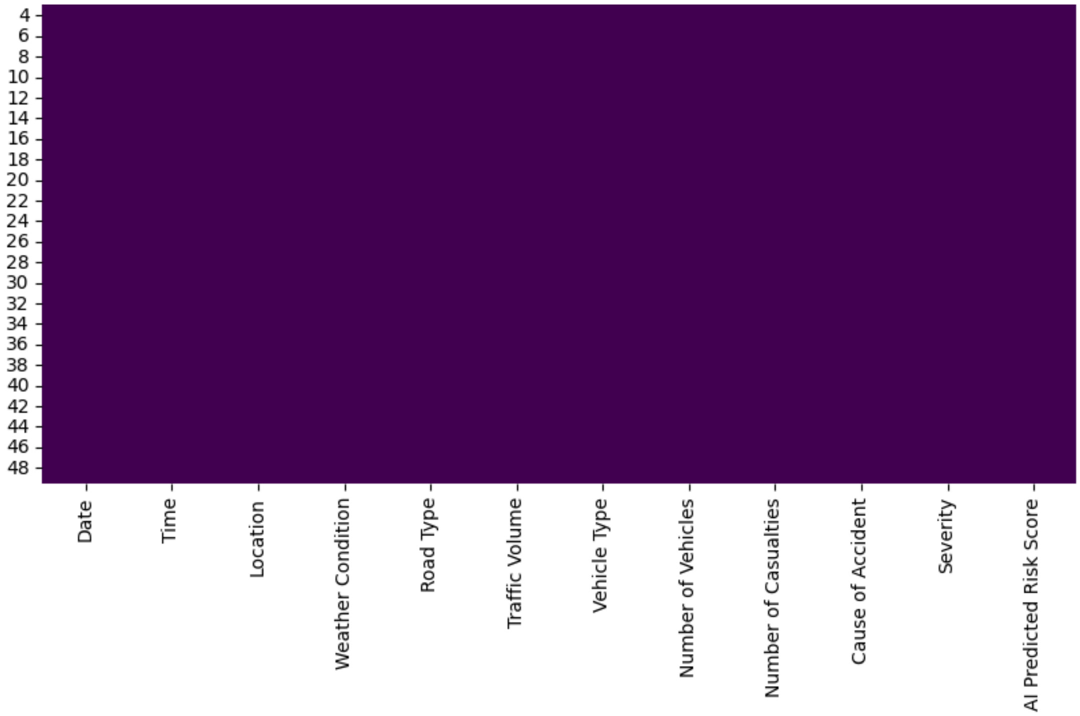
```
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Load data
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")  # Replace with your actual dataset path
target_col = 'time'  # Replace with your actual target column


#  Extract age if DOB exists

if 'date' in df.columns:
    df['date'] = pd.to_datetime(df['date'], errors='coerce')
    df['age'] = (pd.Timestamp.today() - df['date']).dt.days // 365
    df.drop('date', axis=1, inplace=True)


#  Identify feature types

numerical_cols = df.select_dtypes(include=['int64', 'float64']).drop(columns=[target_col], errors='ignore').columns.tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()


# Imputation + Scaling + Encoding Pipelines

numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])


#  Optional: Add polynomial (interaction) features

add_interactions = False  # Set True if you want pairwise interaction terms

if add_interactions:
    poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
    numeric_data = poly.fit_transform(df[numerical_cols].fillna(df[numerical_cols].median()))
    poly_feature_names = poly.get_feature_names_out(numerical_cols)
    df_poly = pd.DataFrame(numeric_data, columns=poly_feature_names)
    df = pd.concat([df.drop(columns=numerical_cols), df_poly], axis=1)
    numerical_cols = df_poly.columns.tolist()
    categorical_cols = df.select_dtypes(include=['object']).columns.tolist()


# Binning examples (e.g., time bins)

if 'time' in df.columns:
    df['time_group'] = pd.cut(df['time'], bins=[8.00,9.00], labels=['child', 'young_adult', 'adult', 'senior'])
    categorical_cols.append('age_group')

# preprocessing with ColumnTransformer

preprocessor = ColumnTransformer([
    ('num', numeric_pipeline, numerical_cols),
    ('cat', categorical_pipeline, categorical_cols)
])

X = df.drop(columns=[target_col], errors='ignore')
y = df[target_col] if target_col in df else None

X_processed = preprocessor.fit_transform(X)

print(f" Feature matrix shape after engineering: {X_processed.shape}")
```

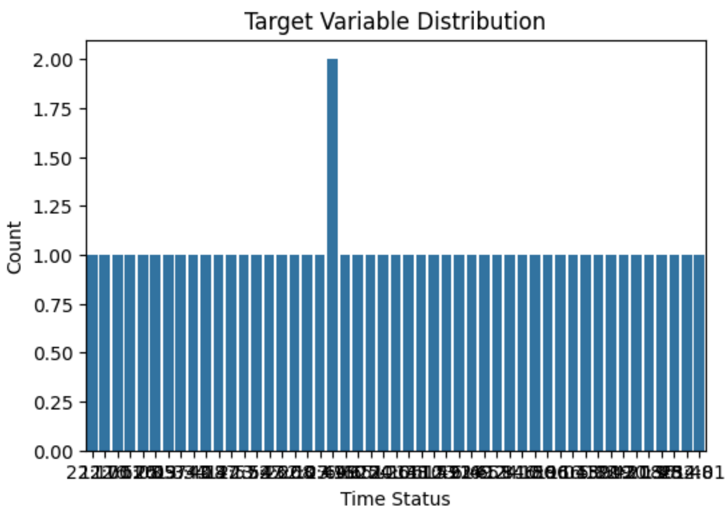
```
Feature matrix shape after engineering: (50, 128)
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import numpy as np

# Load your dataset
df = pd.read_csv("ai_traffic_accident_analysis_csv.csv")  # Replace with your file

# **Print the available columns in your DataFrame to check for the correct target column name**
print(df.columns)

# **Adjust target_col based on the actual column name you see in the output above**
# (e.g., 'Time' if that's the correct name in your data)
target_col = "Time"  # Replace "Time" with the actual correct column name if needed

# Ensure the target column is present in the DataFrame
if target_col not in df.columns:
    raise ValueError(f"Target column ('{target_col}') not found in the DataFrame.")

# Basic Info
print("Data Overview")
print(df.info())
print("\nStatistical Summary")
print(df.describe(include='all'))

#  Missing Values
print("\nMissing Values:")
print(df.isnull().sum())

plt.figure(figsize=(10, 5))
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Data Heatmap")
plt.show()

#  Target Distribution
plt.figure(figsize=(6, 4))
sns.countplot(x=target_col, data=df)  # Now using target_col
plt.title("Target Variable Distribution")
plt.xlabel(f"{target_col} Status") # Using the target_col variable for the x-axis label
plt.ylabel("Count")
plt.show()

# ... (Rest of your code)
```

```
Index(['Date', 'Time', 'Location', 'Weather Condition', 'Road Type',
       'Traffic Volume', 'Vehicle Type', 'Number of Vehicles',
       'Number of Casualties', 'Cause of Accident', 'Severity',
       'AI Predicted Risk Score'],
      dtype='object')
Data Overview
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Date                     50 non-null     object 
 1   Time                     50 non-null     object 
 2   Location                 50 non-null     object 
 3   Weather Condition        50 non-null     object 
 4   Road Type                50 non-null     object 
 5   Traffic Volume           50 non-null     int64  
 6   Vehicle Type             50 non-null     object 
 7   Number of Vehicles       50 non-null     int64  
 8   Number of Casualties     50 non-null     int64  
 9   Cause of Accident        50 non-null     object 
 10  Severity                 50 non-null     object 
 11  AI Predicted Risk Score  50 non-null     float64
dtypes: float64(1), int64(3), object(8)
memory usage: 4.8+ KB
None

Statistical Summary
             Date   Time     Location Weather Condition Road Type  \
count          50     50           50                50        50   
unique         48     49            5                 5         4   
top     9/26/2023  23:49  Los Angeles             Windy     Rural   
freq            2      2           12                13        18   
mean          NaN    NaN          NaN               NaN       NaN   
std           NaN    NaN          NaN               NaN       NaN   
min           NaN    NaN          NaN               NaN       NaN   
25%           NaN    NaN          NaN               NaN       NaN   
50%           NaN    NaN          NaN               NaN       NaN   
75%           NaN    NaN          NaN               NaN       NaN   
max           NaN    NaN          NaN               NaN       NaN   

        Traffic Volume Vehicle Type  Number of Vehicles  Number of Casualties  \
count        50.000000           50           50.000000             50.000000   
unique             NaN            5                 NaN                   NaN   
top                NaN        Truck                 NaN                   NaN   
freq               NaN           15                 NaN                   NaN   
mean        505.580000          NaN            2.800000              6.400000   
std         222.361912          NaN            1.456863              3.043897   
min         100.000000          NaN            1.000000              0.000000   
25%         310.000000          NaN            1.250000              4.000000   
50%         493.500000          NaN            3.000000              6.000000   
75%         665.000000          NaN            4.000000              9.750000   
max         962.000000          NaN            5.000000             10.000000   

       Cause of Accident Severity  AI Predicted Risk Score  
count                 50       50                50.000000  
unique                 5        3                      NaN  
top             Speeding    Minor                      NaN  
freq                  11       21                      NaN  
mean                 NaN      NaN                 0.502400  
std                  NaN      NaN                 0.304742  
min                  NaN      NaN                 0.000000  
25%                  NaN      NaN                 0.215000  
50%                  NaN      NaN                 0.545000  
75%                  NaN      NaN                 0.777500  
max                  NaN      NaN                 0.980000  

Missing Values:
Date                       0
Time                       0
Location                   0
Weather Condition          0
Road Type                  0
Traffic Volume             0
Vehicle Type               0
Number of Vehicles         0
Number of Casualties       0
Cause of Accident          0
Severity                   0
AI Predicted Risk Score    0
dtype: int64
```






```
import pandas as pd
from datetime import datetime
from sklearn.metrics import classification_report, roc_auc_score

# Assume these are previously generated
project_name = " AI-driven traffic accident analysis and prediction"
author = "Your Name"
date_run = datetime.now().strftime("%Y-%m-%d %H:%M")

# Example placeholders – replace with real values from your model
model_name = "Logistic Regression"
accuracy = 0.89
```