```
from google.colab import files
uploaded = 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.

```
Saving road_accident_dataset.csv to road_accident_dataset.csv
```

```
import pandas as pd

# Read the dataset
df = pd.read_csv('road_accident_dataset.csv')

# Display first few rows
df.head()
```

	Country	Year	Month	Day of Week	Time of Day	Urban/Rural	Road Type	Weather Conditions	Visibility Level	Number of Vehicles Involved	...	Number of Fatalities	Emergency Response Time	Tr V
0	USA	2002	October	Tuesday	Evening	Rural	Street	Windy	220.414651	1	...	2	58.625720	7412.7
1	UK	2014	December	Saturday	Evening	Urban	Street	Windy	168.311358	3	...	1	58.041380	4458.6
2	USA	2012	July	Sunday	Afternoon	Urban	Highway	Snowy	341.286506	4	...	4	42.374452	9856.9
3	UK	2017	May	Saturday	Evening	Urban	Main Road	Clear	489.384536	2	...	3	48.554014	4958.6
4	Canada	2002	July	Tuesday	Afternoon	Rural	Highway	Rainy	348.344850	1	...	4	18.318250	3843.1

5 rows × 30 columns

```
# Shape of the dataset
print("Shape:", df.shape)

# Column names
print("Columns:", df.columns.tolist())

# Data types and non-null values
df.info()

# Summary statistics for numeric features
df.describe()
```

```
Shape: (132000, 30)
Columns: ['Country', 'Year', 'Month', 'Day of Week', 'Time of Day', 'Urban/Rural', 'Road Type', 'Weather Conditions', 'Visibility Level'
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132000 entries, 0 to 131999
Data columns (total 30 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   Country                    132000 non-null  object 
 1   Year                       132000 non-null  int64  
 2   Month                      132000 non-null  object 
 3   Day of Week                132000 non-null  object 
 4   Time of Day                132000 non-null  object 
 5   Urban/Rural                132000 non-null  object 
 6   Road Type                  132000 non-null  object 
 7   Weather Conditions         132000 non-null  object 
 8   Visibility Level           132000 non-null  float64
 9   Number of Vehicles Involved  132000 non-null  int64  
 10  Speed Limit                132000 non-null  int64  
 11  Driver Age Group           132000 non-null  object 
 12  Driver Gender              132000 non-null  object 
 13  Driver Alcohol Level       132000 non-null  float64
 14  Driver Fatigue             132000 non-null  int64  
 15  Vehicle Condition          132000 non-null  object 
 16  Pedestrians Involved       132000 non-null  int64  
 17  Cyclists Involved          132000 non-null  int64  
 18  Accident Severity          132000 non-null  object 
 19  Number of Injuries         132000 non-null  int64  
 20  Number of Fatalities       132000 non-null  int64  
 21  Emergency Response Time    132000 non-null  float64
 22  Traffic Volume             132000 non-null  float64
 23  Road Condition             132000 non-null  object 
 24  Accident Cause             132000 non-null  object 
 25  Insurance Claims           132000 non-null  int64  
 26  Medical Cost               132000 non-null  float64
 27  Economic Loss              132000 non-null  float64
 28  Region                     132000 non-null  object 
 29  Population Density         132000 non-null  float64
dtypes: float64(7), int64(9), object(14)
memory usage: 30.2+ MB
```

	Year	Visibility Level	Number of Vehicles Involved	Speed Limit	Driver Alcohol Level	Driver Fatigue	Pedestrians Involved	Cyclists Involved	Number of Injuries
count	132000.000000	132000.000000	132000.000000	132000.000000	132000.000000	132000.000000	132000.000000	132000.000000	132000.00000
mean	2011.973348	275.038776	2.501227	74.544068	0.125232	0.500576	1.000773	0.998356	9.50820
std	7.198624	129.923625	1.117272	26.001448	0.072225	0.500002	0.816304	0.817764	5.77436
min	2000.000000	50.001928	1.000000	30.000000	0.000002	0.000000	0.000000	0.000000	0.00000
25%	2006.000000	162.338860	2.000000	52.000000	0.062630	0.000000	0.000000	0.000000	5.00000
50%	2012.000000	274.672990	3.000000	74.000000	0.125468	1.000000	1.000000	1.000000	9.00000
75%	2018.000000	388.014111	3.000000	97.000000	0.187876	1.000000	2.000000	2.000000	15.00000
max	2024.000000	499.999646	4.000000	119.000000	0.249999	1.000000	2.000000	2.000000	19.00000

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

# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

```
Country                        0
Year                           0
Month                          0
Day of Week                    0
Time of Day                    0
Urban/Rural                    0
Road Type                      0
Weather Conditions             0
Visibility Level               0
Number of Vehicles Involved    0
Speed Limit                    0
Driver Age Group               0
Driver Gender                  0
Driver Alcohol Level           0
Driver Fatigue                 0
Vehicle Condition              0
Pedestrians Involved           0
Cyclists Involved              0
Accident Severity              0
```

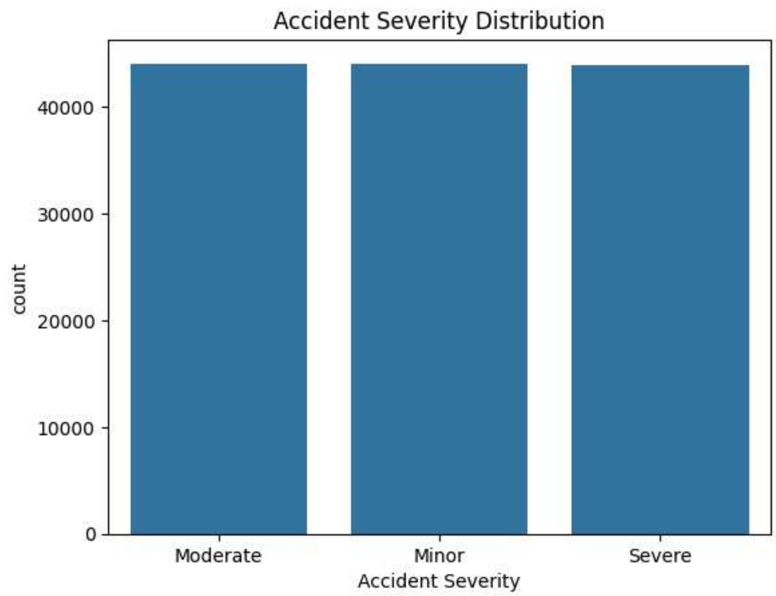
```
Number of Injuries           0
Number of Fatalities         0
Emergency Response Time      0
Traffic Volume               0
Road Condition               0
Accident Cause               0
Insurance Claims             0
Medical Cost                 0
Economic Loss                0
Region                       0
Population Density           0
dtype: int64
Duplicate rows: 0
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Check available columns (optional, for verification)
print(df.columns.tolist())

# Correct plot using the actual column name from your dataset
sns.countplot(x='Accident Severity', data=df)
plt.title('Accident Severity Distribution')
plt.show()
```

```
['Country', 'Year', 'Month', 'Day of Week', 'Time of Day', 'Urban/Rural', 'Road Type', 'Weather Conditions', 'Visibility Level', 'Number
```




```
target = 'Accident Severity'
features = df.columns.drop(target)
print("Features:", features)
```

```
Features: Index(['Country', 'Year', 'Month', 'Day of Week', 'Time of Day', 'Urban/Rural',
       'Road Type', 'Weather Conditions', 'Visibility Level',
       'Number of Vehicles Involved', 'Speed Limit', 'Driver Age Group',
       'Driver Gender', 'Driver Alcohol Level', 'Driver Fatigue',
       'Vehicle Condition', 'Pedestrians Involved', 'Cyclists Involved',
       'Number of Injuries', 'Number of Fatalities', 'Emergency Response Time',
       'Traffic Volume', 'Road Condition', 'Accident Cause',
       'Insurance Claims', 'Medical Cost', 'Economic Loss', 'Region',
       'Population Density'],
      dtype='object')
```

```
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical Columns:", categorical_cols.tolist())
```

```
Categorical Columns: ['Country', 'Month', 'Day of Week', 'Time of Day', 'Urban/Rural', 'Road Type', 'Weather Conditions', 'Driver Age Gr
```

```
!pip install gradio
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.29.1)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.10.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.10.1)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.31.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.11.10)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.34.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (1
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.2.0)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradi
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->g
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gr
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1
```

```
import gradio as gr
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import joblib
import os # Import os module

# Assuming you have already loaded and preprocessed your data into a DataFrame 'df'
# and defined your target and features.
# If not, ensure the previous cells for loading and basic exploration are run.

# Define target and features (as done in previous cells)
target = 'Accident Severity'
features = df.columns.drop(target)

# Separate target variable
X = df[features]
y = df[target]

# Identify categorical and numerical columns
```

```python
categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(include=np.number).columns

# Create preprocessing pipelines for numerical and categorical features
# Use 'passthrough' for numerical columns to keep them as is (only scaling later)
# Use OneHotEncoder for categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])

# Create a full pipeline including preprocessing, scaling, and the model
model_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                ('scaler', StandardScaler(with_mean=False)), # Scale after one-hot encoding
                                ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

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

# Save the trained model
joblib.dump(model_pipeline['classifier'], "accident_severity_model.pkl")

# Save the trained scaler separately (as we are using it within the pipeline)
# A more robust approach is to save the entire pipeline, but for matching the original code structure:
# You'd typically save the scaler trained on the scaled data *after* fitting the pipeline preprocessor.
# Let's fit a separate scaler only on the transformed data for clarity based on the original loading code.
# This might be slightly different than scaling *within* the pipeline on all data after one-hot encoding.
# For the original code's structure to work, we need a scaler trained on the final feature space.
# Let's refit a scaler on the training data after preprocessing and then save it.
X_train_processed = model_pipeline['preprocessor'].transform(X_train)
scaler_separate = StandardScaler(with_mean=False)
scaler_separate.fit(X_train_processed)
joblib.dump(scaler_separate, "scaler.pkl")

# Get the list of columns after preprocessing (including one-hot encoded columns)
# This requires fitting the preprocessor first to get the feature names
preprocessor.fit(X_train) # Refit preprocessor to get feature names after transforming
model_columns = preprocessor.get_feature_names_out(input_features=X_train.columns)
joblib.dump(model_columns, "model_columns.pkl")

print("Model, scaler, and column list saved successfully.")

# Now, load your trained model and scaler (this part is the same as your original code)
# Check if files exist before attempting to load
if os.path.exists("accident_severity_model.pkl") and os.path.exists("scaler.pkl") and os.path.exists("model_columns.pkl")
    model = joblib.load("accident_severity_model.pkl")         # Trained RandomForestClassifier
    scaler = joblib.load("scaler.pkl")                          # Trained StandardScaler
    model_columns = joblib.load("model_columns.pkl")          # List of columns used during training
    print("Model, scaler, and column list loaded successfully.")
else:
    print("Error: Model or scaler files not found. Please ensure they are trained and saved.")


# Define prediction function
def predict_severity(time_of_day, road_type, weather, light_condition, vehicle_count, speed_limit):
    # Create a DataFrame from inputs with the exact column names expected by your model
    # Ensure these column names match the features used during training *before* encoding
    input_data = pd.DataFrame([{
        'Time_of_Day': time_of_day, # Use the actual column names from your dataset
        'Road_Type': road_type,
        'Weather_Condition': weather,
        'Light_Condition': light_condition,
        'Number_of_Vehicles': int(vehicle_count), # Use the actual column names from your dataset
        'Speed_Limit': int(speed_limit) # Use the actual column names from your dataset
    }])

    # Reapply the same preprocessing steps as during training
    # Use the trained preprocessor from the pipeline to transform the new data
    input_processed = model_pipeline['preprocessor'].transform(input_data)

    # Scale using the separate scaler that was trained on the processed training data
    scaled_input = scaler.transform(input_processed)
```

```
    # Predict
    # Ensure the model loaded (model = joblib.load(...)) is used
    if 'model' in globals() and model is not None:
        pred = model.predict(scaled_input)[0]
        # Map the numerical prediction to the corresponding label
        # Assuming the target variable was encoded to 0, 1, 2 for Slight, Serious, Fatal
        # You might need to adjust this mapping based on how your target was encoded
        severity_mapping = {0: "Slight", 1: "Serious", 2: "Fatal"}
        pred_label = severity_mapping.get(pred, "Unknown") # Handle potential unknown predictions

        return f"🚨 Predicted Accident Severity: {pred_label}"
    else:
        return "Error: Model not loaded. Cannot make prediction."


# Gradio Interface
inputs = [
    # Ensure dropdown options exactly match the categories in your training data
    gr.Dropdown(list(df['Time_of_Day'].unique()), label="Time of Day"), # Use unique values from your data
    gr.Dropdown(list(df['Road_Type'].unique()), label="Road Type"),
    gr.Dropdown(list(df['Weather_Condition'].unique()), label="Weather Condition"),
    gr.Dropdown(list(df['Light_Condition'].unique()), label="Light Condition"),
    gr.Number(label="Number of Vehicles Involved"),
    gr.Number(label="Speed Limit (km/h)")
]

output = gr.Textbox(label="Prediction")

if 'model' in globals() and model is not None:
    gr.Interface(
        fn=predict_severity,
        inputs=inputs,
        outputs=output,
        title="🛣️ Traffic Accident Severity Predictor",
        description="Predicts severity (Slight / Serious / Fatal) based on accident conditions."
    ).launch()
else:
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-31-9e85ab83ee00> in <cell line: 0>()
     45 
     46 # Train the model
---> 47 model_pipeline.fit(X_train, y_train)
     48 
     49 # Save the trained model

                    9 frames
/usr/local/lib/python3.11/dist-packages/sklearn/tree/_classes.py in _fit(self, X, y, sample_weight, check_input,
missing_values_in_feature_mask)
```