Upload the Dataset

```
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 traffic_accidents.csv to traffic_accidents.csv
```

Load the Dataset

```
import pandas as pd
# Read the dataset
df = pd.read_csv('traffic_accidents.csv')
```

Data Exploration

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

	weather_condition	lighting_condition	crash_hour	injuries_total
0	2	3	13	0.0
1	2	1	0	0.0
2	2	3	10	0.0
3	2	3	19	5.0
4	2	3	14	0.0

```
# 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: (209306, 24)
Columns: ['crash_date', 'traffic_control_device', 'weather_condition', 'lighting_condition', 'first_crash_type', 'trafficway_type', 'alignment', 'roadway_surface_cond', 'road_
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209306 entries, 0 to 209305
Data columns (total 24 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   crash_date                     209306 non-null  object
 1   traffic_control_device         209306 non-null  object
 2   weather_condition              209306 non-null  object
 3   lighting_condition             209306 non-null  object
 4   first_crash_type               209306 non-null  object
 5   trafficway_type                209306 non-null  object
 6   alignment                      209306 non-null  object
 7   roadway_surface_cond           209306 non-null  object
 8   road_defect                    209306 non-null  object
 9   crash_type                     209306 non-null  object
 10  intersection_related_i         209306 non-null  object
 11  damage                         209306 non-null  object
 12  prim_contributory_cause        209306 non-null  object
 13  num_units                      209306 non-null  int64
 14  most_severe_injury             209306 non-null  object
 15  injuries_total                 209306 non-null  float64
 16  injuries_fatal                 209306 non-null  float64
 17  injuries_incapacitating        209306 non-null  float64
 18  injuries_non_incapacitating    209306 non-null  float64
 19  injuries_reported_not_evident  209306 non-null  float64
 20  injuries_no_indication         209306 non-null  float64
 21  crash_hour                     209306 non-null  int64
 22  crash_day_of_week              209306 non-null  int64
 23  crash_month                    209306 non-null  int64
dtypes: float64(6), int64(4), object(14)
memory usage: 38.3+ MB
```

	num_units	injuries_total	injuries_fatal	injuries_incapacitating	injuries_non_incapacitating	injuries_reported_not_evident	injuries_no_indication	crash_hour
count	209306.000000	209306.000000	209306.000000	209306.000000	209306.000000	209306.000000	209306.000000	209306.000000
mean	2.063300	0.382717	0.001859	0.038102	0.221241	0.121516	2.244002	13.373047
std	0.396012	0.799720	0.047502	0.233964	0.614960	0.450865	1.241175	5.603830
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	9.000000
50%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	14.000000
75%	2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	3.000000	17.000000
max	11.000000	21.000000	3.000000	7.000000	21.000000	15.000000	49.000000	23.000000

```
Check for Missing Values and Duplicates
```

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

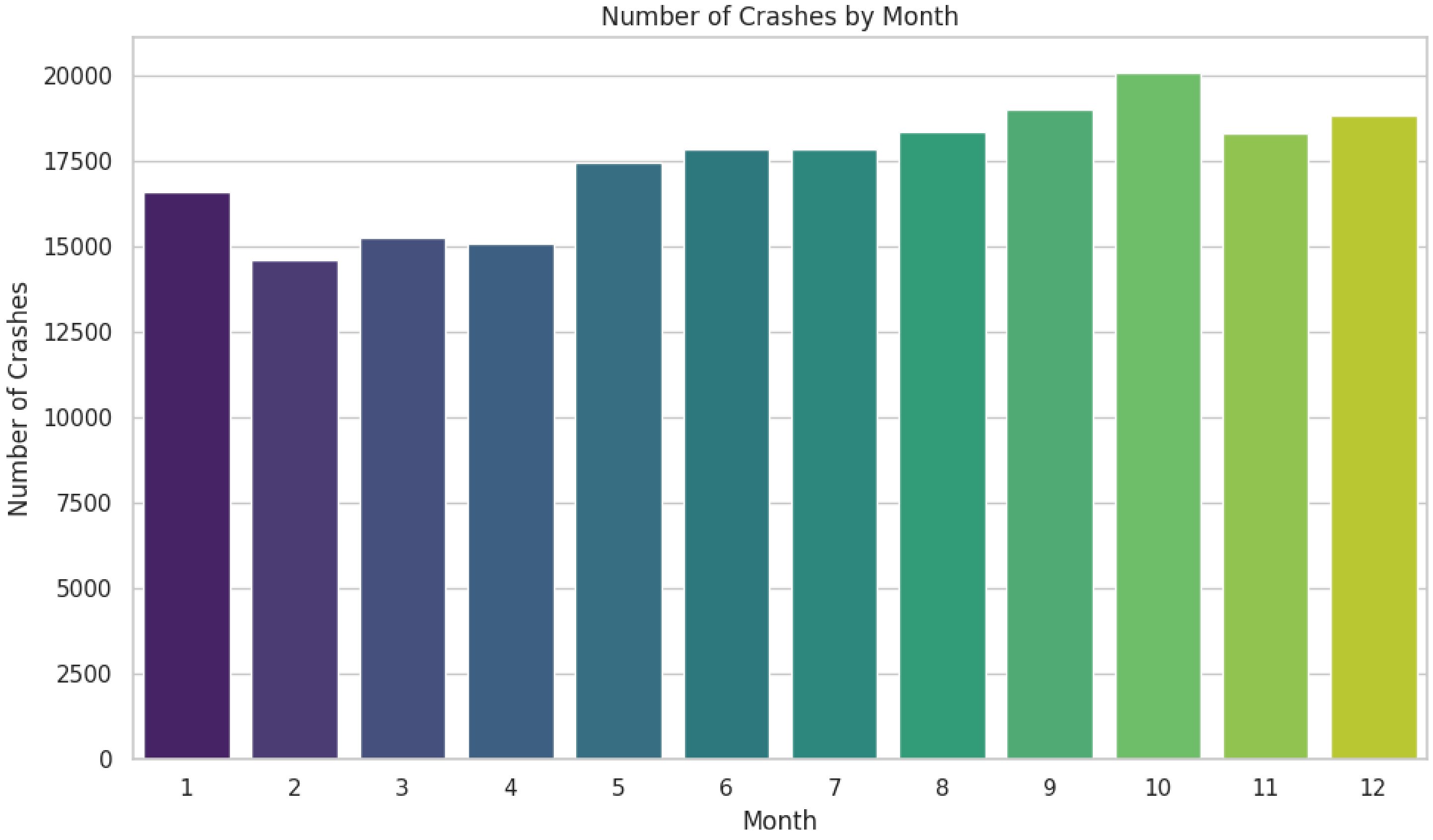
```
crash_date                       0
traffic_control_device           0
weather_condition                0
lighting_condition               0
first_crash_type                 0
trafficway_type                  0
alignment                        0
roadway_surface_cond             0
road_defect                      0
crash_type                       0
intersection_related_i           0
damage                           0
prim_contributory_cause          0
num_units                        0
most_severe_injury               0
injuries_total                   0
injuries_fatal                   0
injuries_incapacitating          0
injuries_non_incapacitating      0
injuries_reported_not_evident    0
injuries_no_indication           0
crash_hour                       0
crash_day_of_week                0
crash_month                      0
dtype: int64
Duplicate rows: 31
```

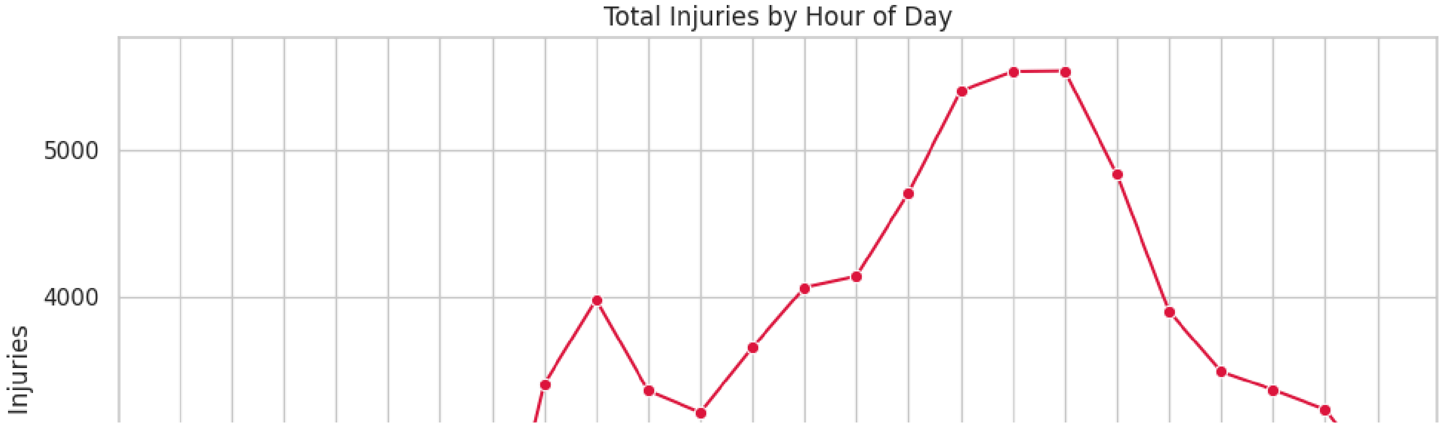
```
Visualize a Few Features
```

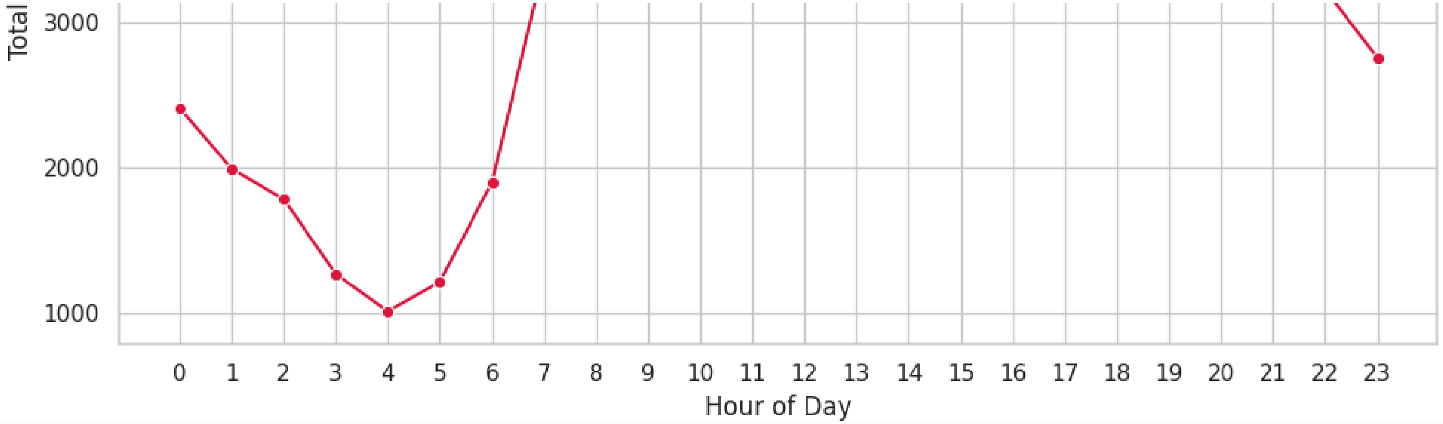
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set a consistent style
sns.set(style="whitegrid")

# 1. Crashes by Month
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x="crash_month", palette="viridis")
plt.title("Number of Crashes by Month")
plt.xlabel("Month")
plt.ylabel("Number of Crashes")
plt.tight_layout()
plt.show()


# 2. Total Injuries by Crash Hour
```

```
injuries_by_hour = df.groupby("crash_hour")["injuries_total"].sum().reset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(data=injuries_by_hour, x="crash_hour", y="injuries_total", marker="o", color="crimson")
plt.title("Total Injuries by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Total Injuries")
plt.xticks(range(0, 24))
plt.tight_layout()
plt.show()
```

```
<ipython-input-6-bb12107624f5>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x="crash_month", palette="viridis")
```







```
Identify Target and Features
```

```
target = [col for col in df.columns if 'injur' in col.lower() and df[col].nunique() > 2][0]
features = [col for col in df.columns if col != target]
print("Target:", target, "\nFeatures:", features)
```

```
Target: most_severe_injury
Features: ['crash_date', 'traffic_control_device', 'weather_condition', 'lighting_condition', 'first_crash_type', 'trafficway_type', 'alignment', 'roadway_surface_cond', 'road
```

```
Convert Categorical Columns to Numerical
```

```
from sklearn.preprocessing import LabelEncoder
for col in df.select_dtypes(include='object'): df[col] = LabelEncoder().fit_transform(df[col].astype(str))
print(df.head())
```

```
   crash_date  traffic_control_device  weather_condition  lighting_condition  \
0      102748                      16                  2                   3
1      110797                      16                  2                   1
2      176858                      16                  2                   3
3      108613                      16                  2                   3
4      113911                      16                  2                   3

   first_crash_type  trafficway_type  alignment  roadway_surface_cond  \
0                17                8          3                     5
1                17                6          3                     0
2                10               15          3                     0
3                 0                6          3                     0
4                10               15          3                     5

   road_defect  crash_type  ...  most_severe_injury  injuries_total  \
0            5           1  ...                   2             0.0
1            1           1  ...                   2             0.0
```

```
2           1           1  ...                    2             0.0
3           1           0  ...                    3             5.0
4           5           1  ...                    2             0.0

   injuries_fatal  injuries_incapacitating  injuries_non_incapacitating  \
0             0.0                      0.0                          0.0
1             0.0                      0.0                          0.0
2             0.0                      0.0                          0.0
3             0.0                      0.0                          5.0
4             0.0                      0.0                          0.0

   injuries_reported_not_evident  injuries_no_indication  crash_hour  \
0                            0.0                     3.0          13
1                            0.0                     2.0           0
2                            0.0                     3.0          10
3                            0.0                     0.0          19
4                            0.0                     3.0          14

   crash_day_of_week  crash_month
0                  7            7
1                  1            8
2                  5           12
3                  4            8
4                  7            8

[5 rows x 24 columns]
```

```
One-Hot Encoding
```

```
# Apply one-hot encoding to all object (categorical) columns
df_onehot = pd.get_dummies(df, drop_first=True)
```

```
Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

```
Train-Test Split
```

```
from sklearn.model_selection import train_test_split

X = df.drop("injuries_total", axis=1)
y = df["injuries_total"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Model Building
```

```
from sklearn.ensemble import RandomForestClassifier
X = pd.get_dummies(df[['weather_condition', 'lighting_condition', 'crash_hour']].dropna())
y = (df.loc[X.index, 'injuries_total'] > 0).astype(int)
model = RandomForestClassifier().fit(X, y)
```

```
Evaluation
```

```
from sklearn.metrics import mean_squared_error, r2_score
y_reg = df['injuries_total'].fillna(0)
model = RandomForestClassifier().fit(X_train, y_reg.loc[y_train.index])
pred = model.predict(X_test)
print("MSE:", mean_squared_error(y_reg.loc[y_test.index], pred))
print("R²:", r2_score(y_reg.loc[y_test.index], pred))
```

```
MSE: 0.026348478333572213
R²: 0.9591348331295516
```

```
Make Predictions from New Input
```

```
new_data = pd.DataFrame([{
    'weather_condition': 'CLEAR',
    'lighting_condition': 'DAYLIGHT',
    'crash_hour': 15
}])
# Ensure new_data_encoded has all columns present in the training data (X)
# Get all the original dummies columns
all_dummy_cols = X_train.columns
# Create new_data_encoded with all columns and fill missing with 0
new_data_encoded = pd.get_dummies(new_data,
                                  columns=['weather_condition', 'lighting_condition']).reindex(columns=all_dummy_cols, fill_value=0)
# Add the 'crash_hour' column, ensuring it's of numeric type
new_data_encoded['crash_hour'] = new_data['crash_hour'].astype(int)

predicted_class = model.predict(new_data_encoded)[0]
print("Predicted injury risk (0=No, 1=Yes):", predicted_class)
```

```
Predicted injury risk (0=No, 1=Yes): 1.0
```

```
Convert to DataFrame and Encode
```

```
# or use your loaded df
df = df[['weather_condition', 'lighting_condition', 'crash_hour', 'injuries_total']].dropna()
X = pd.get_dummies(df[['weather_condition', 'lighting_condition', 'crash_hour']])
y = (df['injuries_total'] > 0).astype(int)
```

```
Predict the Final Grade
```

```
from sklearn.ensemble import RandomForestRegressor
X = pd.get_dummies(df[['weather_condition', 'lighting_condition', 'crash_hour']].dropna())
y = df.loc[X.index, 'injuries_total']
print("Predicted injuries:", RandomForestRegressor().fit(X, y).predict(X.iloc[[0]])[0])
```

```
Predicted injuries: 0.3488025225239783
```

```
Deployment-Building an Interactive App
```

```
pip install gradio
```

```
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)











   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB 1.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB 1.9 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 6.5 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.9-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 85.5 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 4.3 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 4.1 MB/s eta 0:00:00
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starlette, safehttpx, gradio-client, fastapi, grac
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.0 gradio-client-1.10.0 groovy-0.1.2 pydub-0.25.1 python-multipart-0.0.20 ruff-0.11.9 safehttp
```

Create a Prediction Function

```python
def predict_injuries(weather, lighting, hour):
    from sklearn.ensemble import RandomForestRegressor
    df_clean = df[['weather_condition', 'lighting_condition', 'crash_hour', 'injuries_total']].dropna()
    X = pd.get_dummies(df_clean[['weather_condition', 'lighting_condition', 'crash_hour']])
    y = df_clean['injuries_total'].loc[X.index]
    model = RandomForestRegressor().fit(X, y)
    new_input = pd.DataFrame([{'weather_condition': weather, 'lighting_condition': lighting, 'crash_hour': hour}])
    new_input = pd.get_dummies(new_input).reindex(columns=X.columns, fill_value=0)
    return model.predict(new_input)[0]
```

Create the Gradio Interface

```python
import gradio as gr
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

# Load the dataset (assuming 'traffic_accidents.csv' is in the current directory)
df = pd.read_csv('traffic_accidents.csv')

def predict_injuries(weather, lighting, hour):
    df_clean = df[['weather_condition', 'lighting_condition', 'crash_hour', 'injuries_total']].dropna()
    X = pd.get_dummies(df_clean[['weather_condition', 'lighting_condition', 'crash_hour']])
    y = df_clean['injuries_total'].loc[X.index]
    model = RandomForestRegressor().fit(X, y)
    new_input = pd.DataFrame([{'weather_condition': weather, 'lighting_condition': lighting, 'crash_hour': hour}])
    new_input = pd.get_dummies(new_input).reindex(columns=X.columns, fill_value=0)
    return model.predict(new_input)[0]


# Create Gradio interface
import gradio as gr
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

# Load the dataset (assuming 'traffic_accidents.csv' is in the current directory)
df = pd.read_csv('/content/traffic_accidents.csv')

def predict_injuries(weather, lighting, hour):
    df_clean = df[['weather_condition', 'lighting_condition', 'crash_hour', 'injuries_total']].dropna()
    X = pd.get_dummies(df_clean[['weather_condition', 'lighting_condition', 'crash_hour']])
    y = df_clean['injuries_total'].loc[X.index]
    model = RandomForestRegressor().fit(X, y)
    new_input = pd.DataFrame([{'weather_condition': weather, 'lighting_condition': lighting, 'crash_hour': hour}])
    new_input = pd.get_dummies(new_input).reindex(columns=X.columns, fill_value=0)
    return model.predict(new_input)[0]


# Create Gradio interface
gr.Interface(
    fn=predict_injuries,
    inputs=[
        gr.Dropdown(choices=df['weather_condition'].dropna().unique().tolist(), label="Weather Condition"),
        gr.Dropdown(choices=df['lighting_condition'].dropna().unique().tolist(), label="Lighting Condition"),
        gr.Slider(0, 23, step=1, label="Crash Hour")
    ],
    outputs=gr.Number(label="Predicted Injuries")
).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically setting `share=True` (you can turn this

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ad7642650602f6a098.gradio.live
```