Generate 10 random numbers using numpy Close

```
from IPython import get_ipython
from IPython.display import display
from google.colab import files
import pandas as pd
import io # Import the io module

uploaded = files.upload()

# Print the keys to see the actual uploaded filenames
print(uploaded.keys())

# Access the content of the uploaded file from the 'uploaded' dictionary
# Instead of hardcoding the key, get the first key from the uploaded dictionary
# This is safer in case the filename is slightly different or truncated
uploaded_filename = list(uploaded.keys())[0]
file_content = uploaded[uploaded_filename]

# Use io.BytesIO to treat the file content as a file-like object
df = pd.read_csv(io.BytesIO(file_content))

# Now the dataframe 'df' contains the data from the uploaded CSV file
```

Choose files traffic_accid…taset (1).csv

- **traffic_accidents dataset (1).csv**(text/csv) - 967468 bytes, last modified: 14/05/2025 - 100% done

```
Saving traffic accidents dataset (1).csv to traffic accidents dataset (1).csv
```

```
# prompt: handling missing values for the above dataset

# Option 5: Fill missing values with the median of the column (for numerical data)
for col in df.select_dtypes(include=['number']).columns:
  df[col] = df[col].fillna(df[col].median())

# Option 6: Fill missing values with the mode of the column (for categorical data)
for col in df.select_dtypes(include=['object']).columns:
  # mode() can return multiple values if there's a tie, so we take the first one [0]
  if not df[col].mode().empty:
    df[col] = df[col].fillna(df[col].mode()[0])

# Display the dataframe after filling NaNs with median and mode
display(df)

# Identify missing values again to verify
print("\nMissing values after handling:")
print(df.isnull().sum())
```

NameError ✕ •••

Please explain the error:

```
NameError: name 'df' is not defined
```

Hypothesis

The `NameError: name 'df' is not defined` traceback indicates that the variable `df` was accessed before it was assigned a value. Looking at the provided code snippets and the traceback, the most likely cause is that the first code block, which defines and loads the dataframe into the variable `df`, was not executed successfully before the second code block was run.

In a Jupyter Notebook or similar environment, cells need to be run sequentially for variables defined in earlier cells to be available in later cells. If the cell containing the `files.upload()` and `pd.read_csv()` code was skipped or failed to run, the `df` variable would not be created, leading to the `NameError` in the subsequent cell that tries to use it.

Suggested Changes

To fix this error, ensure that the first cell which loads the CSV data into the `df` variable is executed successfully *before* executing the second cell that attempts to use `df`.

Here is the code from the first cell. Make sure this cell is run successfully.

```
from IPython import get_ipython
from IPython.display import display
from google.colab import files
import pandas as pd
import io # Import the io module

uploaded = files.upload()

# Print the keys to see the actual uploaded f
print(uploaded.keys())

# Access the content of the uploaded file fro
# Instead of hardcoding the key, get the firs
# This is safer in case the filename is sligh
uploaded_filename = list(uploaded.keys())[0]
file_content = uploaded[uploaded_filename]

# Use io.BytesIO to treat the file content as
df = pd.read_csv(io.BytesIO(file_content))

# Now the dataframe 'df' contains the data fr
```

Use code with caution

After successfully running the above cell, you can then run the second cell which performs the missing value handling. The `df` variable will now be defined and accessible.

```
# prompt: handling missing values for the abo

# Identify missing values
print(df.isnull().sum())

# Option 1: Drop rows with missing values
df_dropped_rows = df.dropna()
```

	Accident ID	Date	Time	Location	Latitude	Longitude	Weather Condition	Condi
0	b0dd6f57	2023-04-19	06:39	Mumbai, India	13.488432	-73.290682	Snow	S
1	debfad09	2023-01-17	02:47	São Paulo, Brazil	-37.798317	-32.244242	Clear	
2	6d69aa36	2024-04-09	02:55	Sydney, Australia	33.767869	104.869018	Rain	S
3	425bb1f0	2023-10-10	11:23	Tokyo, Japan	-0.378031	-165.825855	Storm	
4	90d5cf62	2023-01-02	12:07	Beijing, China	41.254879	-30.776959	Storm	S
...	...	...	...	...	...	...	...	
9995	2d26c7e2	2023-01-10	18:41	Paris, France	-41.344055	109.335620	Storm	
9996	4d236cfd	2023-04-04	16:48	São Paulo, Brazil	-60.765148	-10.432225	Storm	
9997	1d32722f	2024-09-30	14:43	Beijing, China	-11.161278	-72.164379	Snow	U Constru
9998	64722572	2024-10-27	18:34	Sydney, Australia	-17.153524	20.803006	Storm	U Constru
9999	96272c1b	2024-10-26	10:30	Toronto, Canada	21.917486	100.486079	Storm	

10000 rows × 11 columns

```
Missing values after handling:
Accident ID          0
Date                 0
Time                 0
Location             0
Latitude             0
Longitude            0
Weather Condition    0
Road Condition       0
Vehicles Involved    0
Casualties           0
```

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

```
# prompt: feature selection for the above dataset

import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif, f_regression
from sklearn.preprocessing import LabelEncoder

# Assuming 'df' is your pandas DataFrame from the preceding code
# Identify target variable (replace 'target_column' with your actual target column r
# For demonstration, let's assume the last column is the target.
target_column = df.columns[-1] # Or explicitly set your target column name
X = df.drop(columns=[target_column])
y = df[target_column]

# Handle categorical features in X by label encoding or one-hot encoding
# For simplicity in demonstration, we'll use label encoding. For real-world, conside
# Also, make sure the target variable 'y' is numerical if using f_regression or f_cl
# If 'y' is categorical, use LabelEncoder.
if y.dtype == 'object':
    le = LabelEncoder()
    y = le.fit_transform(y)

# Label encode object type columns in X
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    # Fit and transform only on non-null values to avoid errors
    X[col] = X[col].astype(str) # Convert to string first to handle potential NaNs t
    X[col] = le.fit_transform(X[col])

# Select the top K features using f_classif (for classification) or f_regression (fc
# Choose the appropriate test based on your target variable type
# Here we assume classification for demonstration. If regression, use f_regression.
# Ensure there are no NaNs in X before applying SelectKBest
X = X.fillna(X.mean()) # Fill any remaining NaNs in feature columns for safety
```

```
# Option 2: Drop columns with missing values
df_dropped_cols = df.dropna(axis=1)

# Option 3: Fill missing values with a specif
df_filled_zero = df.fillna(0)
df_filled_placeholder = df.fillna('Missing')

# Option 4: Fill missing values with the mean
for col in df.select_dtypes(include=['number'
  df[col] = df[col].fillna(df[col].mean())

# Option 5: Fill missing values with the medi
# for col in df.select_dtypes(include=['numbe
#   df[col] = df[col].fillna(df[col].median()

# Option 6: Fill missing values with the mode
# for col in df.select_dtypes(include=['objec
#   df[col] = df[col].fillna(df[col].mode()[0

# Display the modified dataframes (choose the
# display(df_dropped_rows)
# display(df_dropped_cols)
# display(df_filled_zero)
# display(df_filled_placeholder)
display(df) # Displaying the dataframe after
```

Use code with caution

Rate this answer

```
k_features = 10 # Number of features to select (adjust as needed)

if len(X) > 0: # Ensure X is not empty
  if len(X.columns) > 0: # Ensure X has columns
    if len(y) == len(X): # Ensure X and y have the same number of samples
      try:
        # Use f_classif for classification problems
        selector = SelectKBest(score_func=f_classif, k=min(k_features, X.shape[1]))
        fit = selector.fit(X, y)

        # Get the scores and p-values
        scores = fit.scores_
        pvalues = fit.pvalues_

        # Create a DataFrame to display the scores and p-values
        feature_scores = pd.DataFrame({'Features': X.columns, 'Score': scores, 'P-va

        # Sort the features by score in descending order
        feature_scores = feature_scores.sort_values(by='Score', ascending=False)

        print("\nFeature Scores and P-values (Sorted by Score):")
        display(feature_scores)

        # Get the selected feature names
        selected_features = X.columns[selector.get_support()]
        print("\nSelected Features:")
        print(list(selected_features))

        # Create a new dataframe with only the selected features and the target
        df_selected = df[list(selected_features) + [target_column]]
        print("\nDataFrame with Selected Features:")
        display(df_selected.head())

      except ValueError as e:
        print(f"Error during feature selection: {e}")
        print("This might be due to issues with data types or variance in features/
        print("Check your data and ensure appropriate preprocessing (like handling 
    else:
      print("Mismatch between the number of samples in features (X) and target (y).
  else:
    print("The feature DataFrame (X) is empty.")
else:
  print("The DataFrame (df) is empty or contains no data after preprocessing.")
```

```
Feature Scores and P-values (Sorted by Score):
```

	Features	Score	P-value
0	Accident ID	1.406421	0.218368
5	Longitude	1.305397	0.258359
9	Casualties	1.218644	0.297343
1	Date	1.078692	0.369842
4	Latitude	0.883381	0.491120
3	Location	0.757050	0.580752
2	Time	0.641434	0.668086
8	Vehicles Involved	0.456586	0.808759
7	Road Condition	0.398846	0.849929
6	Weather Condition	0.249591	0.940184

```
Selected Features:
['Accident ID', 'Date', 'Time', 'Location', 'Latitude', 'Longitude', 'Weather C

DataFrame with Selected Features:
```

	Accident ID	Date	Time	Location	Latitude	Longitude	Weather Condition	Road Condition
0	b0dd6f57	2023-04-19	06:39	Mumbai, India	13.488432	-73.290682	Snow	Snowy
1	debfad09	2023-01-17	02:47	São Paulo, Brazil	-37.798317	-32.244242	Clear	Icy
2	6d69aa36	2024-04-09	02:55	Sydney, Australia	33.767869	104.869018	Rain	Snowy
3	425bb1f0	2023-10-10	11:23	Tokyo, Japan	-0.378031	-165.825855	Storm	Wet
4	90d5cf62	2023-	12:07	Beijing,	41.254879	-30.776959	Storm	Snowy

Next steps: Generate code with feature_scores | View recommended plots | New interactive s

```
# prompt: visualisation like histograms for the above dataset

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your pandas DataFrame from the preceding code
# Choose numerical columns for histograms
numerical_cols = df.select_dtypes(include=['number']).columns

# Create histograms for each numerical column
print("\nGenerating Histograms for Numerical Features:")
for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.histplot(data=df, x=col, kde=True)
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# You might also want to visualize the distribution of the target variable if it's nu
if target_column in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.histplot(data=df, x=target_column, kde=True)
    plt.title(f'Histogram of {target_column}')
    plt.xlabel(target_column)
    plt.ylabel('Frequency')
    plt.show()

# For categorical columns, you might use count plots
categorical_cols = df.select_dtypes(include=['object']).columns

print("\nGenerating Count Plots for Categorical Features:")
for col in categorical_cols:
    plt.figure(figsize=(10, 6))
    sns.countplot(data=df, y=col, order=df[col].value_counts().index) # Use y for hor
    plt.title(f'Count Plot of {col}')
    plt.xlabel('Count')
    plt.ylabel(col)
```

```
        plt.show()

    # If the target variable is categorical, visualize its distribution
    if target_column in df.columns and df[target_column].dtype == 'object':
         plt.figure(figsize=(10, 6))
         sns.countplot(data=df, y=target_column, order=df[target_column].value_counts().i
         plt.title(f'Count Plot of {target_column}')
         plt.xlabel('Count')
         plt.ylabel(target_column)
         plt.show()
```