IMPORTING WILDFIRE DATA

```
import pandas as pd

# Load the dataset
file_path = '/content/CANADA_WILDFIRES.csv'
data = pd.read_csv(file_path)

# Display the first few rows and summary info about the dataset
data.head(), data.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 423831 entries, 0 to 423830
Data columns (total 9 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   FID         423831 non-null  int64
 1   SRC_AGENCY  423831 non-null  object
 2   LATITUDE    423831 non-null  float64
 3   LONGITUDE   423831 non-null  float64
 4   REP_DATE    420118 non-null  object
 5   SIZE_HA     423831 non-null  float64
 6   CAUSE       423590 non-null  object
 7   PROTZONE    422821 non-null  object
 8   ECOZ_NAME   423831 non-null  object
dtypes: float64(3), int64(1), object(5)
memory usage: 29.1+ MB
(   FID SRC_AGENCY  LATITUDE  LONGITUDE    REP_DATE  SIZE_HA CAUSE PROTZONE  \
 0    0         BC    59.963   -128.172  1953-05-26      8.0     H
 1    1         BC    59.318   -132.172  1950-06-22      8.0     L
 2    2         BC    59.876   -131.922  1950-06-04  12949.9     H
 3    3         BC    59.760   -132.808  1951-07-15    241.1     H
 4    4         BC    59.434   -126.172  1952-06-12      1.2     H

            ECOZ_NAME
 0  Boreal Cordillera
 1  Boreal Cordillera
 2  Boreal Cordillera
 3  Boreal Cordillera
 4  Boreal Cordillera  ,
 None)
```

NEURAL NETWORK MODEL WITH ONLY WILDFIRE DATA

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
```

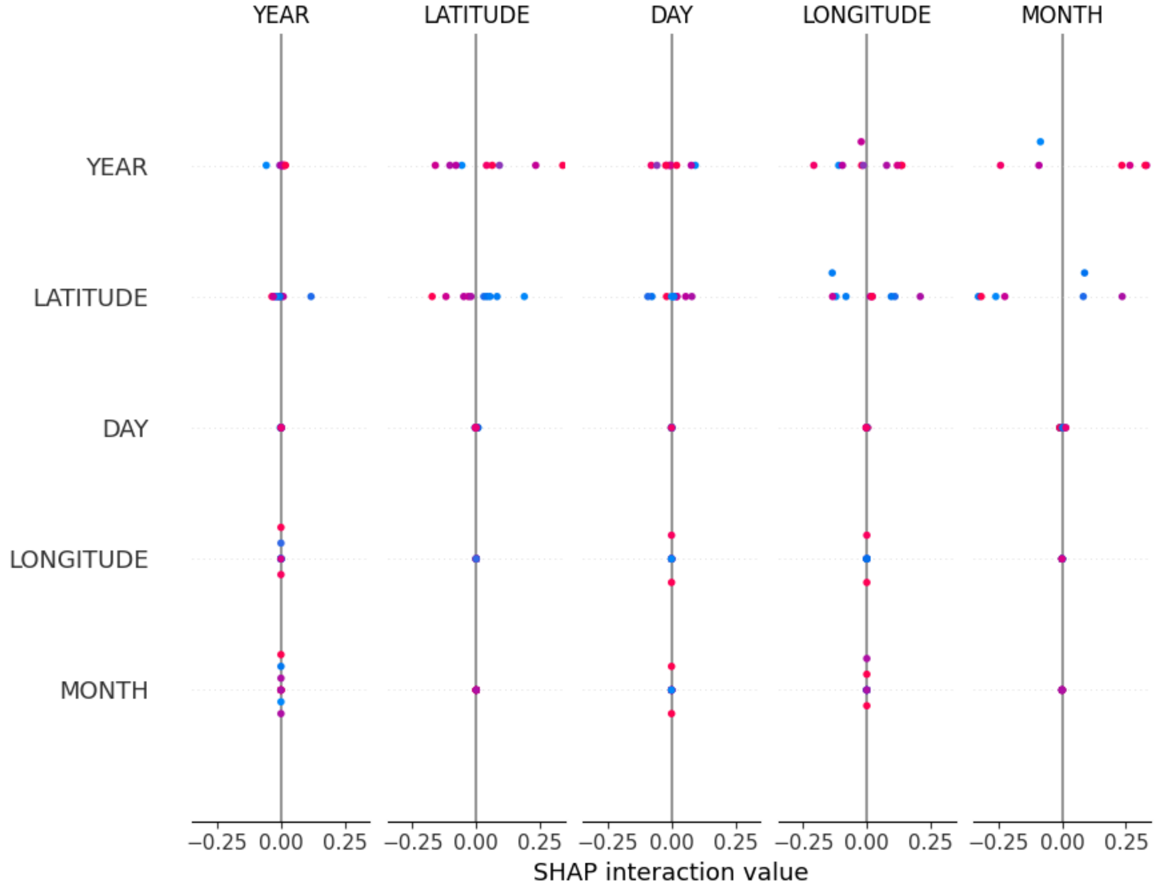
```
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
# Define the dataset path

# Drop rows with NaN values to clean the data
data = data.dropna()

# Parse REP_DATE and extract features
data['REP_DATE'] = pd.to_datetime(data['REP_DATE'], errors='coerce')
data['YEAR'] = data['REP_DATE'].dt.year
data['MONTH'] = data['REP_DATE'].dt.month
data['DAY'] = data['REP_DATE'].dt.day

# Drop unnecessary columns
data = data.drop(columns=['REP_DATE', 'FID'])

# Encode categorical variables (excluding CAUSE since it will be the target)
categorical_columns = ['SRC_AGENCY', 'PROTZONE', 'ECOZ_NAME']
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_features = pd.DataFrame(
    encoder.fit_transform(data[categorical_columns]),
    columns=encoder.get_feature_names_out(categorical_columns),
    index=data.index
)

# Combine encoded features with numerical features
numerical_columns = ['LATITUDE', 'LONGITUDE', 'YEAR', 'MONTH', 'DAY']
X = pd.concat([data[numerical_columns], encoded_features], axis=1)

# Encoding the target variable (CAUSE)
y = pd.get_dummies(data['CAUSE'])

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

# Scaling the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the deep learning model
model = Sequential([
    Dense(256, activation='relu', kernel_regularizer=l2(0.001), input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(X_train, y_train, validation_split=0.2, epochs=10, batch_size=32, callbacks=[reduce_lr, early_stopping], verbose=1)

# Evaluate the model
eval_results = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {eval_results[1]}")

# Predict and generate classification report
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.values.argmax(axis=1)
print(classification_report(y_test_classes, y_pred_classes, target_names=y.columns))

```

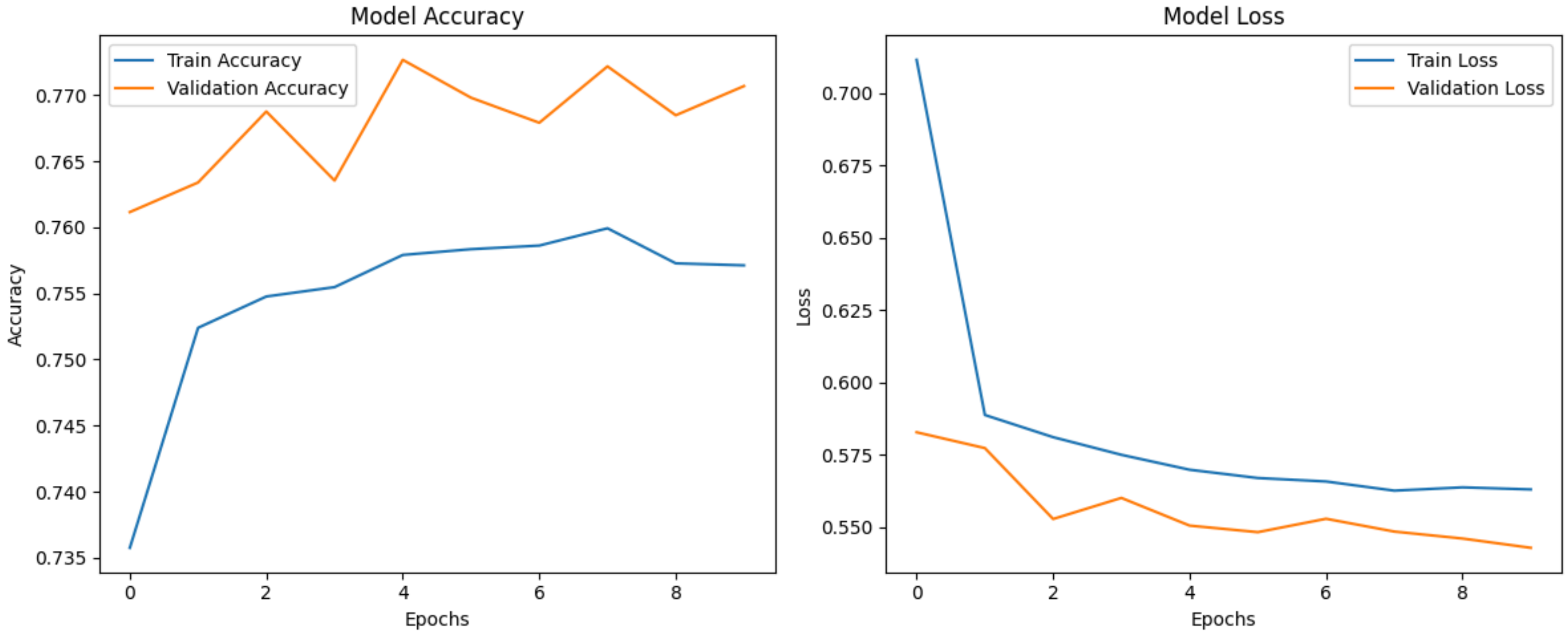
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/3
8378/8378 ━━━━━━━━━━━━━━━━━━━━ 53s 6ms/step - accuracy: 0.6997 - loss: 0.9128 - val_accuracy: 0.7614 - val_loss: 0.5830 - learning_rate: 0.0010
Epoch 2/3
8378/8378 ━━━━━━━━━━━━━━━━━━━━ 70s 4ms/step - accuracy: 0.7518 - loss: 0.5907 - val_accuracy: 0.7592 - val_loss: 0.5905 - learning_rate: 0.0010
Epoch 3/3
8378/8378 ━━━━━━━━━━━━━━━━━━━━ 41s 5ms/step - accuracy: 0.7522 - loss: 0.5859 - val_accuracy: 0.7622 - val_loss: 0.5692 - learning_rate: 0.0010
Test Accuracy: 0.7657507061958313
2618/2618 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
              precision    recall  f1-score   support

           H       0.77      0.83      0.80     46048
        H-PB       0.42      0.64      0.51        78
           L       0.76      0.72      0.74     36344
          RE       0.00      0.00      0.00        11
           U       0.00      0.00      0.00      1293

    accuracy                           0.77     83774
   macro avg       0.39      0.44      0.41     83774
weighted avg       0.75      0.77      0.76     83774

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

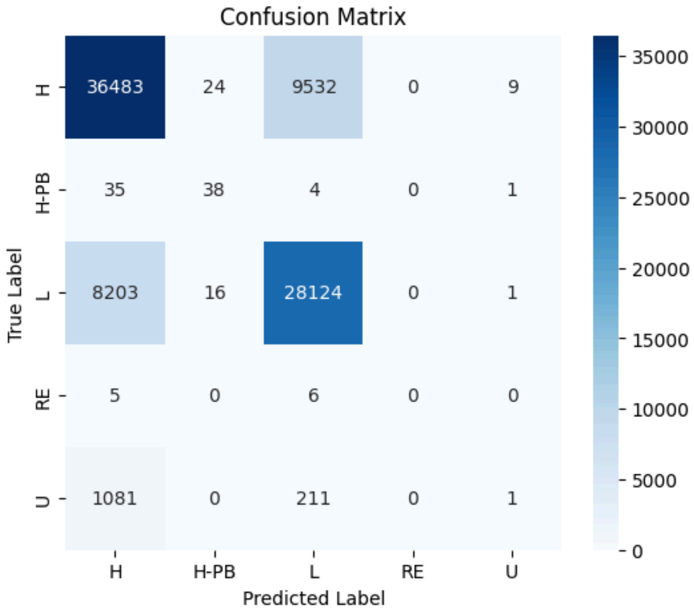
```
# Sample 1000 rows from the training data to use as background samples
X_train_sample = shap.sample(X_train, 10)

# Create a SHAP explainer with the sampled data
explainer = shap.KernelExplainer(model.predict, X_train_sample)

# Calculate SHAP values
shap_values = explainer.shap_values(X_train_sample)

# Plot SHAP summary plot
shap.summary_plot(shap_values, X_train_sample, feature_names=X.columns)

```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
100%                                          10/10 [00:20<00:00,  2.26s/it]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
653/653 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix, f1_score

# Plot accuracy and loss graphs
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Generate confusion matrix
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test.values, axis=1)
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y.columns, yticklabels=y.columns)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
class_f1_scores = f1_score(y_test_classes, y_pred_classes, average=None)
print('F1 Score for each class:')
for class_name, score in zip(y.columns, class_f1_scores):
    print(f'{class_name}: {score:.4f}')

```






```
F1 Score for each class:
H: 0.7944
H-PB: 0.4872
L: 0.7578
RE: 0.0000
U: 0.0015
```

COMBINING HISTRICAL WILDFIRE DATA FROM 2006 TO 2023 TO OUR WILDFIRE DATA

```
import pandas as pd
import numpy as np
from tqdm import tqdm  # For progress bar


# Load the datasets
wildfire_data = pd.read_excel('/content/fp-historical-wildfire-data-2006-2023.xlsx')
canada_wildfires = pd.read_csv('/content/CANADA_WILDFIRES.csv')
# Select relevant columns
wildfire_data_filtered = wildfire_data[['fire_location_latitude', 'fire_location_longitude', 'reported_date'] + [col for col in wildfire_data.columns if col not in ['fire_location_latitude', 'fire_
canada_wildfires_filtered = canada_wildfires[['LATITUDE', 'LONGITUDE', 'REP_DATE'] + [col for col in canada_wildfires.columns if col not in ['LATITUDE', 'LONGITUDE', 'REP_DATE']]]

# Rename columns for consistency
wildfire_data_filtered.columns = ['LATITUDE', 'LONGITUDE', 'REP_DATE'] + [col for col in wildfire_data_filtered.columns[3:]]
canada_wildfires_filtered.columns = ['LATITUDE', 'LONGITUDE', 'REP_DATE'] + [col for col in canada_wildfires_filtered.columns[3:]]

# Convert date formats dynamically
wildfire_data_filtered['REP_DATE'] = pd.to_datetime(wildfire_data_filtered['REP_DATE']).dt.strftime('%Y-%m-%d')
canada_wildfires_filtered['REP_DATE'] = pd.to_datetime(canada_wildfires_filtered['REP_DATE'], format='mixed').dt.strftime('%Y-%m-%d')

# Set tolerance for latitude/longitude matching (round latitudes and longitudes for better matching)
wildfire_data_filtered['LATITUDE'] = wildfire_data_filtered['LATITUDE'].round(2)
wildfire_data_filtered['LONGITUDE'] = wildfire_data_filtered['LONGITUDE'].round(2)
canada_wildfires_filtered['LATITUDE'] = canada_wildfires_filtered['LATITUDE'].round(2)
canada_wildfires_filtered['LONGITUDE'] = canada_wildfires_filtered['LONGITUDE'].round(2)

# Merge datasets on the three key columns
merged_df = pd.merge(
    canada_wildfires_filtered,
    wildfire_data_filtered,
    on=['LATITUDE', 'LONGITUDE', 'REP_DATE'],
    how='inner'
)

# Save the merged results
merged_df.to_csv("matched_wildfires.csv", index=False)

# Print summary with match count
print(f'\nTotal Number of Matches Found: {len(merged_df)}')
print("Merged data saved as 'matched_wildfires.csv'")

# Print out columns in the merged dataframe to confirm the presence of all columns
print(f"Columns in the matched dataset: {merged_df.columns.tolist()}")

```

```
Total Number of Matches Found: 26544
Merged data saved as 'matched_wildfires.csv'
Columns in the matched dataset: ['LATITUDE', 'LONGITUDE', 'REP_DATE', 'FID', 'SRC_AGENCY', 'SIZE_HA', 'CAUSE', 'PROTZONE', 'ECOZ_NAME', 'fire_year', 'fire_number', 'fire_name', 'current_size', 'si
```

NULL PERCENTAGE

```
# Calculate percentage of null values in each column
null_percentage = merged_df.isnull().mean() * 100

# Print the null percentage for each column
print("Null Percentage in Each Column:")
print(null_percentage)

```

```
Null Percentage in Each Column:
LATITUDE                        0.000000
LONGITUDE                       0.000000
REP_DATE                        0.000000
FID                             0.000000
SRC_AGENCY                      0.000000
SIZE_HA                         0.000000
CAUSE                           0.060277
PROTZONE                        0.000000
ECOZ_NAME                       0.000000
fire_year                       0.000000
fire_number                     0.000000
fire_name                      97.558770
current_size                    0.000000
size_class                      0.000000
fire_origin                     0.033906
general_cause_desc              0.000000
industry_identifier_desc       98.372514
responsible_group_desc         54.788276
activity_class                 33.630952
true_cause                     41.527275
fire_start_date                 4.132761
det_agent_type                  0.000000
det_agent                       0.000000
discovered_date                15.807715
discovered_size                99.457505
dispatched_resource             0.011302
dispatch_date                   0.011302
start_for_fire_date             0.015069
assessment_resource             0.000000
assessment_datetime             0.000000
assessment_hectares             0.000000
fire_spread_rate               16.839964
fire_type                      16.045057
fire_position_on_slope         17.009494
weather_conditions_over_fire   17.024563
temperature                    17.028330
relative_humidity              17.035865
wind_direction                 17.039632
wind_speed                     17.039632
fuel_type                      35.567360
initial_action_by               0.011302
ia_arrival_at_fire_date        25.467149
ia_access                      57.191832
fire_fighting_start_date       24.886980
fire_fighting_start_size       24.886980
```

```
bucketing_on_fire           25.561332
distance_from_water_source  75.350362
first_bucket_drop_date      75.350362
bh_fs_date                   0.000000
bh_hectares                  0.000000
uc_fs_date                   0.000000
uc_hectares                  0.000000
to_fs_date                  90.683394
to_hectares                 90.683394
ex_fs_date                   0.000000
ex_hectares                  0.000000
dtype: float64
```

REMOVING COLUMNS WITH NULL PERCENTAGE OVER 20%

```python
# Calculate percentage of null values in each column
null_percentage = merged_df.isnull().mean() * 100

# Identify columns with more than 20% null values
columns_to_remove = null_percentage[null_percentage > 20].index

merged_df_cleaned = merged_df.drop(columns=columns_to_remove)

# Print summary
print(f"Columns removed due to >20% null values: {columns_to_remove.tolist()}")
print(f"Remaining columns: {merged_df_cleaned.columns.tolist()}")

```

```
Columns removed due to >20% null values: ['fire_name', 'industry_identifier_desc', 'responsible_group_desc', 'activity_class', 'true_cause', 'discovered_size', 'fuel_type', 'ia_arrival_at_fire_dat
Remaining columns: ['LATITUDE', 'LONGITUDE', 'REP_DATE', 'FID', 'SRC_AGENCY', 'SIZE_HA', 'CAUSE', 'PROTZONE', 'ECOZ_NAME', 'fire_year', 'fire_number', 'current_size', 'size_class', 'fire_origin',
```

```python
merged_df=merged_df_cleaned.dropna()
```

FINDING CORRELATION

```python
from sklearn.preprocessing import LabelEncoder

# Convert date columns to numeric (days since Unix epoch)
def convert_dates_to_numeric(df):
    for column in df.select_dtypes(include=['object', 'datetime']):
        if pd.to_datetime(df[column], errors='coerce').notnull().all():
            df[column] = pd.to_datetime(df[column], errors='coerce').apply(lambda x: (x - pd.Timestamp('1970-01-01')).days)
    return df

# Label encode non-numeric columns
def label_encode_columns(df):
    label_encoder = LabelEncoder()
    for column in df.select_dtypes(include=['object']):
        df[column] = label_encoder.fit_transform(df[column].astype(str))
    return df

# Convert date columns to numeric
```

```
merged_df = convert_dates_to_numeric(merged_df)

# Label encode categorical columns (non-numeric)
merged_df = label_encode_columns(merged_df)

# Check if 'CAUSE' exists and is categorical
if 'CAUSE' in merged_df.columns:
    # Label encode the 'CAUSE' column if it's still categorical
    label_encoder = LabelEncoder()
    merged_df['cause_encoded'] = label_encoder.fit_transform(merged_df['CAUSE'])

    # Calculate correlation of all columns with 'cause_encoded'
    correlation = merged_df.corr()

    # Show correlation with the 'cause_encoded' column
    cause_correlation = correlation['cause_encoded']

    # Rank the absolute value of the correlation values in descending order
    absolute_correlation = cause_correlation.abs().sort_values(ascending=False)

    # Print the ranked absolute correlation of each column to 'cause'
    print("Ranked Absolute Correlation to 'cause':")
    print(absolute_correlation)
else:
    print("Column 'CAUSE' not found in the dataset.")

```

```
Ranked Absolute Correlation to 'cause':
cause_encoded                  1.000000
CAUSE                          1.000000
fire_origin                    0.393488
temperature                    0.376415
general_cause_desc             0.335058
LATITUDE                       0.249471
dispatched_resource            0.235939
fire_number                    0.214588
relative_humidity              0.180776
size_class                     0.163690
weather_conditions_over_fire   0.153791
fire_type                      0.146829
fire_spread_rate               0.135771
det_agent                      0.130965
det_agent_type                 0.118438
fire_start_date                0.091869
fire_position_on_slope         0.080628
wind_speed                     0.074361
ex_fs_date                     0.071937
uc_fs_date                     0.071727
bh_fs_date                     0.071481
dispatch_date                  0.071417
assessment_datetime            0.071415
REP_DATE                       0.071412
discovered_date                0.071412
start_for_fire_date            0.071398
fire_year                      0.066169
assessment_resource            0.062938
ECOZ_NAME                      0.051343
SIZE_HA                        0.041290
ex_hectares                    0.041170
```

```
current_size              0.041170
uc_hectares               0.039161
initial_action_by         0.035241
bh_hectares               0.035088
wind_direction            0.030787
assessment_hectares       0.022380
LONGITUDE                 0.012901
PROTZONE                  0.007348
SRC_AGENCY                0.007348
FID                       0.005083
Name: cause_encoded, dtype: float64
```

```
1 merged_df_cleaned
```

	LATITUDE	LONGITUDE	REP_DATE	FID	SRC_AGENCY	SIZE_HA	CAUSE	PROTZONE	ECOZ_NAME	fire_year	...	relative_humidity	wind_direction	wind_speed	initial_action_by	bh_fs_date	bh_hecta
0	59.40	-110.64	2006-06-15	172655	AB	5.70	L		Taiga Shield West	2006	...	65.0	NW	5.0	HAC1H	2006-06-15 15:52:00	
1	59.43	-110.29	2006-06-18	172656	AB	18204.00	L		Taiga Shield West	2006	...	53.0	NW	10.0	HAC1H	2006-06-18 21:30:00	20
2	59.44	-110.28	2006-06-18	172657	AB	0.01	L		Taiga Shield West	2006	...	48.0	NE	7.0	HAC1H	2006-06-18 21:30:00	
3	59.48	-110.33	2006-06-18	172658	AB	450.00	L		Taiga Shield West	2006	...	35.0	NW	5.0	HAC1H	2006-06-18 21:30:00	45
4	59.48	-110.32	2006-06-18	172659	AB	0.01	L		Taiga Shield West	2006	...	25.0	NW	5.0	HAC1H	2006-06-18 18:15:00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
26539	58.89	-114.95	2021-07-08	212113	AB	2728.00	L		Taiga Plain	2021	...	30.0	SW	15.0	FPD Staff	2021-07-11 13:05:00	272
26540	57.15	-113.22	2021-07-12	212114	AB	5960.00	L		Boreal PLain	2021	...	39.0	W	25.0	Air Tanker	2021-07-19 17:00:00	321
26541	53.63	-115.14	2021-06-22	212115	AB	175.00	U		Boreal PLain	2021	...	27.0	NW	20.0	HAC	2021-06-26 11:18:00	18
26542	53.42	-118.31	2006-07-23	420182	PC-JA	2.00	L	Intensive	Montane Cordillera	2006	...	61.0	NW	5.0	HAC1H	2006-07-23 21:30:00	
26543	51.73	-115.53	2021-07-30	423751	PC-BA	0.05	H	Full Response	Montane Cordillera	2021	...	33.0	SE	5.0	Public	2021-07-30 16:02:00	

26544 rows × 40 columns

merging the new columns to our exsting wildfire dataset

```
import pandas as pd
df= pd.read_csv('/content/matched_wildfires (1).csv')
# Assuming your dataset is loaded into a DataFrame called df
columns_to_keep = ['CAUSE','LATITUDE', 'LONGITUDE', 'REP_DATE', 'fire_origin', 'temperature',
                   'dispatched_resource', 'relative_humidity', 'size_class',
                   'weather_conditions_over_fire', 'fire_type', 'fire_spread_rate',
                   'det_agent', 'det_agent_type']

# Filter the DataFrame to keep only the desired columns
data = df[columns_to_keep]

# Display the filtered DataFrame
data
```

	CAUSE	LATITUDE	LONGITUDE	REP_DATE	fire_origin	temperature	dispatched_resource	relative_humidity	size_class	weather_conditions_over_fire	fire_type	fire_spread_rate	det_agent	det_
0	L	59.40	-110.64	2006-06-15	Provincial Land	31.0	HAC	65.0	C	Clear	Crown	3.0	MD	
1	L	59.43	-110.29	2006-06-18	Provincial Land	26.0	HAC	53.0	E	Clear	Surface	3.0	HAC	
2	L	59.44	-110.28	2006-06-18	Provincial Land	28.0	HAC	48.0	A	Clear	Ground	1.0	CF	
3	L	59.48	-110.33	2006-06-18	Provincial Land	24.0	HAC	35.0	E	Clear	Crown	2.0	HAC	
4	L	59.48	-110.32	2006-06-18	Provincial Land	29.0	HAC	25.0	A	Clear	Surface	0.0	HAC	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
26539	L	58.89	-114.95	2021-07-08	Provincial Land	27.0	HAC	30.0	E	Clear	Crown	5.0	FG	
26540	L	57.15	-113.22	2021-07-12	Provincial Land	23.2	FTAC	39.0	E	Clear	Crown	5.0	UAA	

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

```
data = data.dropna()
data
```

	CAUSE	LATITUDE	LONGITUDE	REP_DATE	fire_origin	temperature	dispatched_resource	relative_humidity	size_class	weather_conditions_over_fire	fire_type	fire_spread_rate	det_agent	det_
0	L	59.40	-110.64	2006-06-15	Provincial Land	31.0	HAC	65.0	C	Clear	Crown	3.0	MD	
1	L	59.43	-110.29	2006-06-18	Provincial Land	26.0	HAC	53.0	E	Clear	Surface	3.0	HAC	
2	L	59.44	-110.28	2006-06-18	Provincial Land	28.0	HAC	48.0	A	Clear	Ground	1.0	CF	
3	L	59.48	-110.33	2006-06-18	Provincial Land	24.0	HAC	35.0	E	Clear	Crown	2.0	HAC	
4	L	59.48	-110.32	2006-06-18	Provincial Land	29.0	HAC	25.0	A	Clear	Surface	0.0	HAC	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
26539	L	58.89	-114.95	2021-07-08	Provincial Land	27.0	HAC	30.0	E	Clear	Crown	5.0	FG	
26540	L	57.15	-113.22	2021-07-12	Provincial Land	23.2	FTAC	39.0	E	Clear	Crown	5.0	UAA	

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

creating same Neural network model with our new dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

# Assuming your dataset is loaded into the 'data' DataFrame
# df = pd.read_csv('your_dataset.csv') # Un-comment this to load the data if it's in a CSV file

# Clean data by dropping rows with NaN values


# Parse REP_DATE and extract features (Year, Month, Day)
data['REP_DATE'] = pd.to_datetime(data['REP_DATE'], errors='coerce')
data['YEAR'] = data['REP_DATE'].dt.year
data['MONTH'] = data['REP_DATE'].dt.month
data['DAY'] = data['REP_DATE'].dt.day

# Drop unnecessary columns
data = data.drop(columns=['REP_DATE'])

# Encode categorical variables (excluding CAUSE as it's the target)
```

```
categorical_columns = ['fire_origin', 'dispatched_resource', 'size_class',
                       'weather_conditions_over_fire', 'fire_type', 'det_agent',
                       'det_agent_type']
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_features = pd.DataFrame(
    encoder.fit_transform(data[categorical_columns]),
    columns=encoder.get_feature_names_out(categorical_columns),
    index=data.index
)

# Combine encoded features with numerical features
numerical_columns = ['LATITUDE', 'LONGITUDE', 'YEAR', 'MONTH', 'DAY', 'temperature',
                     'relative_humidity', 'fire_spread_rate']
X = pd.concat([data[numerical_columns], encoded_features], axis=1)

# Encode the target variable (CAUSE)
y = pd.get_dummies(data['CAUSE'])
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the deep learning model
model = Sequential([
    Dense(256, activation='relu', kernel_regularizer=l2(0.001), input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='softmax')  # Softmax for multi-class classification
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Callbacks for reducing learning rate and early stopping
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=10, batch_size=32,
                    callbacks=[reduce_lr, early_stopping], verbose=1)

# Evaluate the model
eval_results = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {eval_results[1]}")

# Predict and generate a classification report
y_pred = model.predict(X_test)
```

```
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.values.argmax(axis=1)
print(classification_report(y_test_classes, y_pred_classes, target_names=y.columns))

```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['YEAR'] = data['REP_DATE'].dt.year
<ipython-input-56-9ea52bba118f>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['MONTH'] = data['REP_DATE'].dt.month
<ipython-input-56-9ea52bba118f>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['DAY'] = data['REP_DATE'].dt.day
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
440/440 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.6795 - loss: 1.4899 - val_accuracy: 0.8891 - val_loss: 0.7511 - learning_rate: 0.0010
Epoch 2/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8646 - loss: 0.8046 - val_accuracy: 0.8843 - val_loss: 0.6774 - learning_rate: 0.0010
Epoch 3/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8764 - loss: 0.6936 - val_accuracy: 0.8846 - val_loss: 0.6228 - learning_rate: 0.0010
Epoch 4/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8728 - loss: 0.6400 - val_accuracy: 0.8937 - val_loss: 0.5708 - learning_rate: 0.0010
Epoch 5/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8773 - loss: 0.5833 - val_accuracy: 0.8962 - val_loss: 0.5239 - learning_rate: 0.0010
Epoch 6/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8829 - loss: 0.5305 - val_accuracy: 0.8911 - val_loss: 0.4895 - learning_rate: 0.0010
Epoch 7/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8803 - loss: 0.4999 - val_accuracy: 0.8922 - val_loss: 0.4721 - learning_rate: 0.0010
Epoch 8/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8847 - loss: 0.4710 - val_accuracy: 0.8920 - val_loss: 0.4469 - learning_rate: 0.0010
Epoch 9/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8844 - loss: 0.4564 - val_accuracy: 0.8971 - val_loss: 0.4318 - learning_rate: 0.0010
Epoch 10/10
440/440 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8902 - loss: 0.4268 - val_accuracy: 0.8937 - val_loss: 0.4290 - learning_rate: 0.0010
Test Accuracy: 0.889015257358551
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           H       0.89      0.94      0.91      2678
        H-PB       0.00      0.00      0.00         4
           L       0.88      0.89      0.88      1583
          RE       0.00      0.00      0.00        10
           U       0.00      0.00      0.00       122
```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use

```
# Plot accuracy and loss graphs
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()


```