Importing all the necessary libraries

```
!pip install category_encoders
!pip install dabl

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import category_encoders as ce
import dabl
import time
import tensorflow as tf

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.preprocessing import label_binarize, LabelEncoder, OneHotEncoder

import plotly.graph_objects as go
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
# Metrics dictionary
accuracy = dict()
precision = dict()
recall = dict()
f1 = dict()
fpr = dict()
tpr = dict()
```

Reading the data

```
df = pd.read_csv('US_Accidents_March23_sampled_500k.csv', nrows=4000)
display(df)
column = df.columns
# all the categorical columns
cat_columns = [colname for colname in df.select_dtypes(["object", "category"])]
len(cat_columns)
```

	ID	Source	Severity	Start_Time	End_Time	Start_Lat
0	A-2047758	Source2	2	2019-06-12 10:10:56	2019-06-12 10:55:58	30.641211

1	A-4694324	Source1	2	2022-12-03 23:37:14.000000000	2022-12-04 01:56:53.000000000	38.990562
2	A-5006183	Source1	2	2022-08-20 13:13:00.000000000	2022-08-20 15:22:45.000000000	34.661189
3	A-4237356	Source1	2	2022-02-21 17:43:04	2022-02-21 19:43:23	43.680592
4	A-6690583	Source1	2	2020-12-04 01:46:00	2020-12-04 04:13:09	35.395484
...	...	...	...	...	...	...
3995	A-3039826	Source2	3	2018-01-01 16:37:09	2018-01-01 17:21:49	34.034649
3996	A-7473792	Source1	2	2018-12-02 11:12:04	2018-12-02 11:41:20	32.793539
3997	A-7207927	Source1	2	2020-03-01 17:50:00	2020-03-01 18:23:34	34.143044
3998	A-6018197	Source1	2	2021-08-19 17:13:00	2021-08-19 22:02:13	34.602169
3999	A-1378470	Source2	3	2020-07-06 21:30:36	2020-07-06 22:24:32	38.614197

4000 rows × 46 columns

```
20
```

```
X.head()
```

	Severity	Start_Lat	Start_Lng	Distance(mi)	Temperature(F)	Humidity(%)	Pres
0	1	0.646508	0.348937	0.000000	0.449064	0.166667	
1	1	0.993655	0.014886	0.257687	0.667360	0.468750	
2	1	0.750957	0.989122	0.000000	0.407484	0.406250	
3	1	0.579510	0.006255	0.000000	0.854470	0.145833	
4	1	0.411929	0.767306	0.000000	0.823285	0.583333	

5 rows × 164 columns

```
# describe categorical columns
df.describe(include='object')
```

	ID	Source	Start_Time	End_Time	Description	Street	City	Cou
count	4000	4000	4000	4000	4000	3992	4000	4
unique	4000	3	3999	4000	3970	2740	1467	
					A crash has			

top	A-2047758	Source1	2020-10-22 23:54:00	2019-06-12 10:55:58	occurred causing no to minimum del...	I-95 N	Miami	Ange
freq	1	2201	2	1	8	44	98	

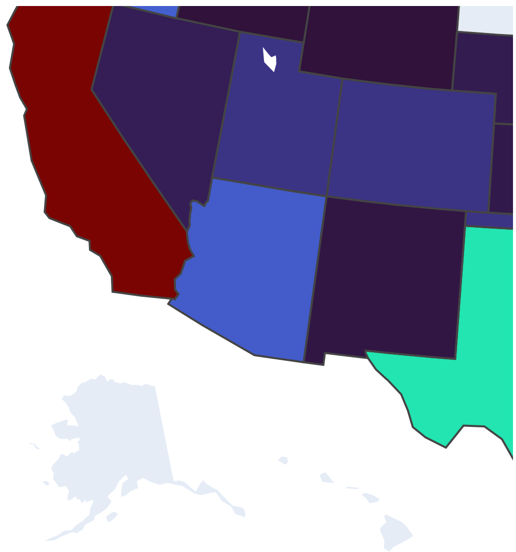
```
df.columns
```

```
Index(['ID', 'Source', 'Severity', 'Start_Time', 'End_Time', 'Start_Lat',
       'Start_Lng', 'End_Lat', 'End_Lng', 'Distance(mi)', 'Description',
       'Street', 'City', 'County', 'State', 'Zipcode', 'Country', 'Timezone',
       'Airport_Code', 'Weather_Timestamp', 'Temperature(F)', 'Wind_Chill(F)',
       'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind_Direction',
       'Wind_Speed(mph)', 'Precipitation(in)', 'Weather_Condition', 'Amenity',
       'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
       'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',
       'Turning_Loop', 'Sunrise_Sunset', 'Civil_Twilight', 'Nautical_Twilight',
       'Astronomical_Twilight'],
      dtype='object')
```

Exploratory Data Analysis

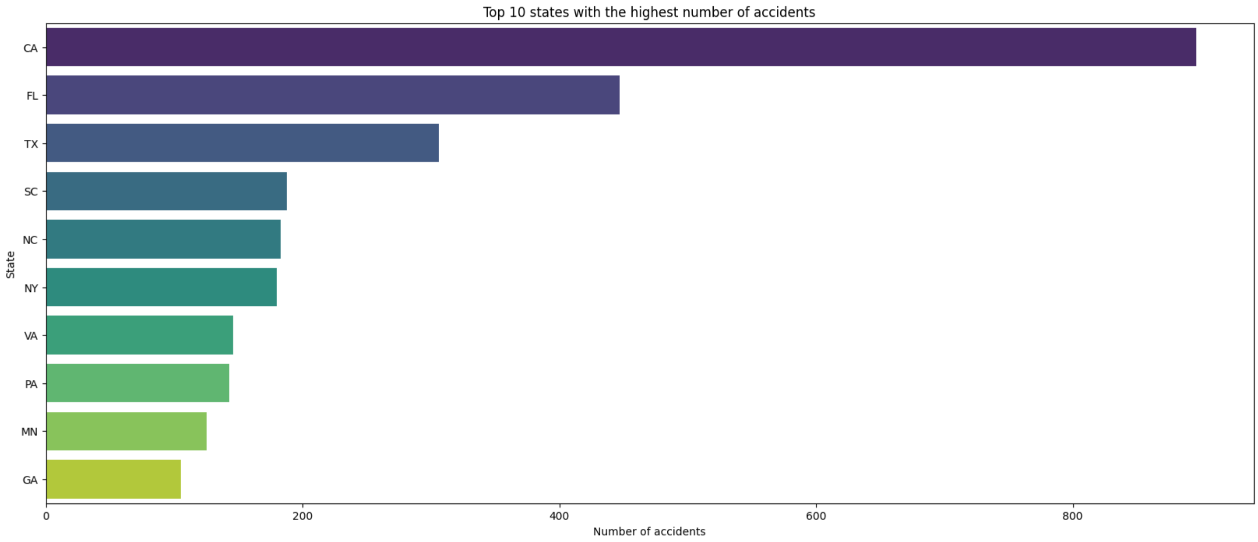
```
state_counts = df["State"].value_counts()
fig = go.Figure(data=go.Choropleth(locations=state_counts.index, z=state_counts.values
fig.update_layout(title_text="Number of US Accidents for each State", geo_scope="usa")
fig.show()
```

Number of US Accidents for each State



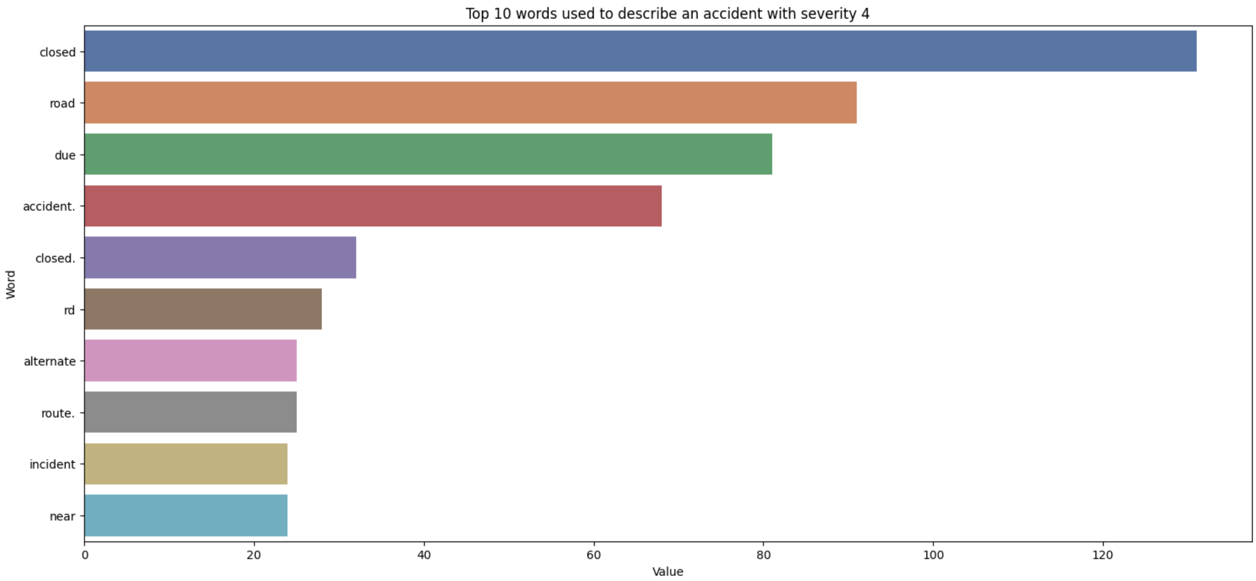
```
plt.figure(figsize=(20, 8))
plt.title("Top 10 states with the highest number of accidents")
sns.barplot(x=state_counts[:10].values, y=state_counts[:10].index, orient="h", palette
plt.xlabel("Number of accidents")
plt.ylabel("State")
plt.show()
```

```
<ipython-input-201-da259d883998>:3: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```



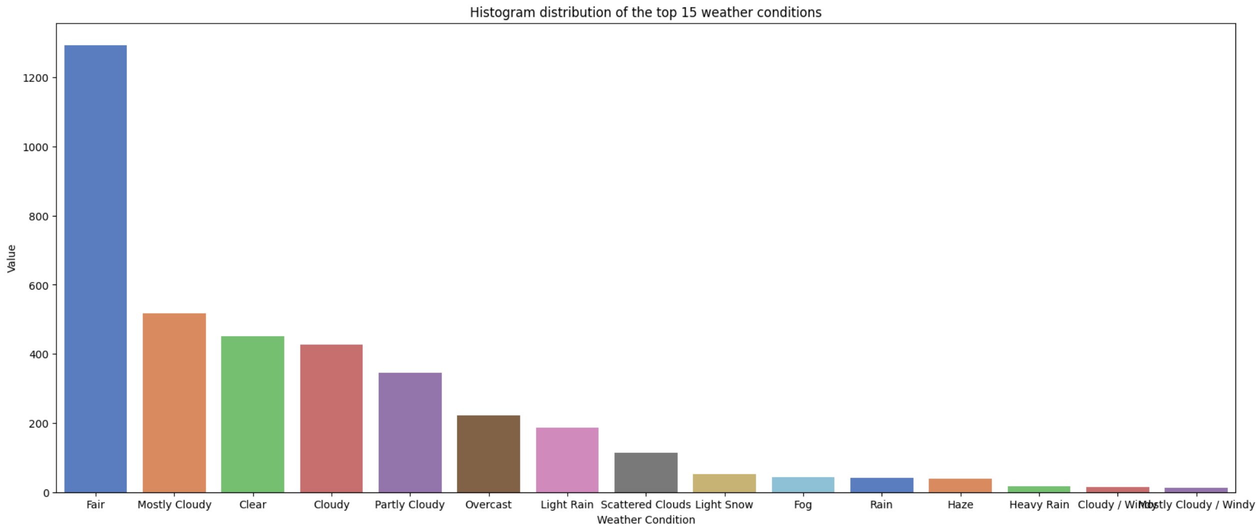
```
stop = stopwords.words("english") + ["-"]

df_s4_desc = df[df["Severity"] == 4]["Description"]
# Split the description
df_words = df_s4_desc.str.lower().str.split(expand=True).stack()

# If the word is not in the stopwords list
counts = df_words[~df_words.isin(stop)].value_counts()[:10]

plt.figure(figsize=(18, 8))
plt.title("Top 10 words used to describe an accident with severity 4")
sns.barplot(x=counts.values, y=counts.index, palette="deep")
plt.xlabel("Value")
plt.ylabel("Word")
plt.show()
```

```
<ipython-input-202-7d7fa243e6ac>:12: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```



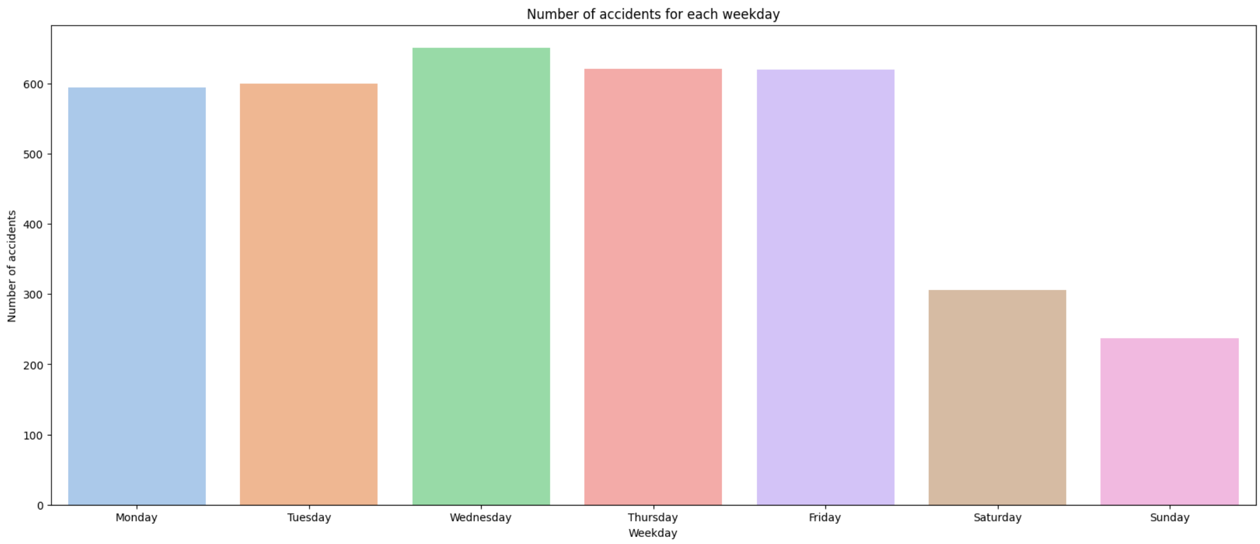
```
counts = df["Weather_Condition"].value_counts()[:15]
plt.figure(figsize=(20, 8))
plt.title("Histogram distribution of the top 15 weather conditions")
sns.barplot(x=counts.index, y=counts.values, palette="muted")
plt.xlabel("Weather Condition")
plt.ylabel("Value")
plt.show()
```

```
<ipython-input-203-7186130da21b>:4: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```



```
# Define the list of weekdays
weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunda

# Convert 'Start_Time' column to datetime with format and handle errors
counts = pd.to_datetime(df['Start_Time'], format='%Y-%m-%d %H:%M:%S', errors='coerce')

# Reindex the counts to ensure all weekdays are present
counts = counts.reindex(weekdays)

# Plot the counts
plt.figure(figsize=(20, 8))
plt.title("Number of accidents for each weekday")
sns.barplot(x=counts.index, y=counts.values, order=weekdays, palette="pastel")
plt.xlabel("Weekday")
plt.ylabel("Number of accidents")
plt.show()
```

```
<ipython-input-204-12d0e8a71489>:13: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```



```
import dabl
dabl.detect_types(df)
```

Data Preprocessing

```
X = df
X.head()
```

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	
0	A-2047758	Source2	2	2019-06-12 10:10:56	2019-06-12 10:55:58	30.641211	-
1	A-4694324	Source1	2	2022-12-03 23:37:14.000000000	2022-12-04 01:56:53.000000000	38.990562	-
2	A-5006183	Source1	2	2022-08-20 13:13:00.000000000	2022-08-20 15:22:45.000000000	34.661189	-1
3	A-4237356	Source1	2	2022-02-21 17:43:04	2022-02-21 19:43:23	43.680592	-
4	A-6690583	Source1	2	2020-12-04 01:46:00	2020-12-04 04:13:09	35.395484	-

5 rows × 46 columns

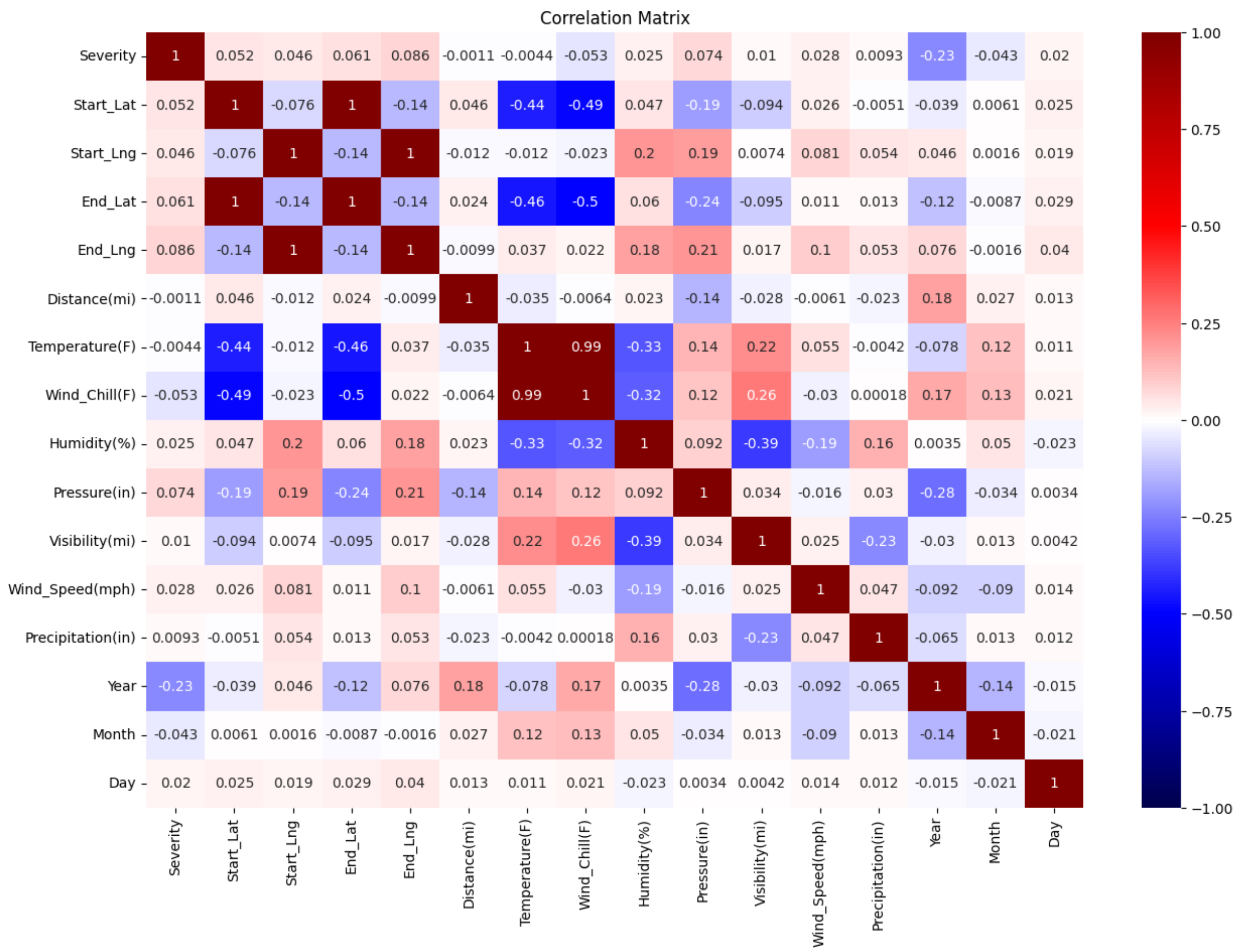
```
# Cast Start_Time to datetime with format and handle errors
X["Start_Time"] = pd.to_datetime(X["Start_Time"], format='%Y-%m-%d %H:%M:%S', errors='

# Extract year, month, weekday, and day
X["Year"] = X["Start_Time"].dt.year
X["Month"] = X["Start_Time"].dt.month
X["Weekday"] = X["Start_Time"].dt.day_name()
X["Day"] = X["Start_Time"].dt.day

X.head()
```

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_L
0	A-2047758	Source2	2	2019-06-12 10:10:56	2019-06-12 10:55:58	30.641211	-91.1534
1	A-4694324	Source1	2	NaT	2022-12-04 01:56:53.000000000	38.990562	-77.3990
2	A-5006183	Source1	2	NaT	2022-08-20 15:22:45.000000000	34.661189	-120.4928
3	A-4237356	Source1	2	2022-02-21	2022-02-21	43.680592	-92.9933

3				17:43:04	19:43:23		
4	A-6690583	Source1	2	2020-12-04 01:46:00	2020-12-04 04:13:09	35.395484	-118.9851

5 rows × 50 columns

```
# Select only numeric columns
numeric_columns = X.select_dtypes(include=['number'])

# Compute correlation matrix
corr_matrix = numeric_columns.corr()

# Plot the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, vmin=-1, vmax=1, cmap="seismic", xticklabels=True, yticklabel
            cbar_kws={"orientation": "vertical"})
plt.gca().patch.set(hatch="X", edgecolor="#666")
plt.title('Correlation Matrix')
plt.show()
```



```
features_to_drop = ["ID", "Source", "Start_Time", "End_Time", "End_Lat", "End_Lng", "D
X = X.drop(features_to_drop, axis=1)
X.head()
```

	Severity	Start_Lat	Start_Lng	Distance(mi)	City	Temperature(F)	Humid
0	2	30.641211	-91.153481	0.000	Zachary	77.0	
1	2	38.990562	-77.399070	0.056	Sterling	45.0	
2	2	34.661189	-120.492822	0.022	Lompoc	68.0	
3	2	43.680592	-92.993317	1.054	Austin	27.0	
4	2	35.395484	-118.985176	0.046	Bakersfield	42.0	

5 rows × 30 columns

```
print("Number of rows:", len(X.index))
X.drop_duplicates(inplace=True)
print("Number of rows after drop of duplicates:", len(X.index))
```

```
Number of rows: 4000
Number of rows after drop of duplicates: 4000
```

```
# Assuming df is your DataFrame
df['Side'] = 'Unknown'  # Add 'Unknown' as default value for all rows
# Assuming df is your DataFrame and you want to copy values from the 'Direction' colum
df['Side'] = df['Pressure(in)']
```

```
# Print the column names of the DataFrame
print(df.columns)

# Check if the 'Side' column exists in the DataFrame
```

```
if 'Side' in df.columns:
    # Perform operations involving the 'Side' column
    # For example:
    # side_counts = df['Side'].value_counts()
    pass
else:
    print("'Side' column does not exist in the DataFrame")
#df["Side"].value_counts()
```

```
Index(['Severity', 'Start_Lat', 'Start_Lng', 'Distance(mi)', 'City',
       'Temperature(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)',
       'Wind_Direction', 'Wind_Speed(mph)', 'Precipitation(in)',
       'Weather_Condition', 'Amenity', 'Bump', 'Crossing', 'Give_Way',
       'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
       'Traffic_Calming', 'Traffic_Signal', 'Civil_Twilight', 'Year', 'Month',
       'Weekday', 'Day'],
      dtype='object')
'Side' column does not exist in the DataFrame
```

```
"""
X = X[X["Side "] != " "]
X["Side "].value_counts()
"""
```

```
'\nX = X[X["Side "] != " "]\nX["Side "].value_counts()\n'
```

```
X[["Pressure(in)", "Visibility(mi)"]].describe().round(2)
```

	Pressure(in)	Visibility(mi)
count	3931.00	3913.00
mean	29.54	9.13
std	0.97	2.72
min	20.37	0.00
25%	29.36	10.00
50%	29.85	10.00
75%	30.03	10.00
max	30.71	70.00

```
X = X[X["Pressure(in)"] != 0]
X = X[X["Visibility(mi)"] != 0]
X[["Pressure(in)", "Visibility(mi)"]].describe().round(2)
```

	Pressure(in)	Visibility(mi)
count	3925.00	3907.00
mean	29.54	9.14

std	0.97	2.70
min	20.37	0.06
25%	29.36	10.00
50%	29.86	10.00
75%	30.03	10.00
max	30.71	70.00

```
unique_weather = X["Weather_Condition"].unique()

print(len(unique_weather))
print(unique_weather)
```

```
49
['Fair' 'Wintry Mix' 'Light Rain' 'Cloudy' 'Mostly Cloudy' 'Partly Cloudy'
 'Clear' 'Scattered Clouds' 'Fog' 'Overcast' 'Light Snow' 'T-Storm' nan
 'Thunderstorms and Rain' 'Thunder' 'Light Rain with Thunder' 'Rain'
 'Showers in the Vicinity' 'Mostly Cloudy / Windy' 'Heavy Rain'
 'Cloudy / Windy' 'Light Drizzle' 'Heavy T-Storm' 'Light Rain / Windy'
 'Smoke' 'Haze' 'Blowing Dust / Windy' 'N/A Precipitation'
 'Thunder in the Vicinity' 'Snow' 'Heavy Thunderstorms and Rain'
 'Shallow Fog' 'Light Freezing Drizzle' 'Fair / Windy' 'Patches of Fog'
 'Light Snow / Windy' 'Blowing Snow / Windy' 'Thunderstorm' 'Drizzle'
 'T-Storm / Windy' 'Partly Cloudy / Windy' 'Heavy Rain / Windy'
 'Heavy Snow / Windy' 'Mist' 'Light Thunderstorms and Rain' 'Rain / Windy'
 'Light Freezing Rain' 'Heavy Snow' 'Light Ice Pellets']
```

```
X.loc[X["Weather_Condition"].str.contains("Thunder|T-Storm", na=False), "Weather_Condi
X.loc[X["Weather_Condition"].str.contains("Snow|Sleet|Wintry", na=False), "Weather_Con
X.loc[X["Weather_Condition"].str.contains("Rain|Drizzle|Shower", na=False), "Weather_C
X.loc[X["Weather_Condition"].str.contains("Wind|Squalls", na=False), "Weather_Conditio
X.loc[X["Weather_Condition"].str.contains("Hail|Pellets", na=False), "Weather_Conditio
X.loc[X["Weather_Condition"].str.contains("Fair", na=False), "Weather_Condition"] = "C
X.loc[X["Weather_Condition"].str.contains("Cloud|Overcast", na=False), "Weather_Condit
X.loc[X["Weather_Condition"].str.contains("Mist|Haze|Fog", na=False), "Weather_Conditi
X.loc[X["Weather_Condition"].str.contains("Sand|Dust", na=False), "Weather_Condition"]
X.loc[X["Weather_Condition"].str.contains("Smoke|Volcanic Ash", na=False), "Weather_Co
X.loc[X["Weather_Condition"].str.contains("N/A Precipitation", na=False), "Weather_Con

print(X["Weather_Condition"].unique())
```

```
['Clear' 'Snow' 'Rain' 'Cloudy' 'Fog' 'Thunderstorm' nan 'Windy' 'Smoke'
 'Hail']
```

```
X["Wind_Direction"].unique()
```

```
array(['NW', 'W', 'ENE', 'CALM', 'SW', 'VAR', 'S', 'E', 'WSW', 'NNE',
       'Variable', 'West', 'N', 'ESE', 'NNW', 'North', 'SSW', 'WNW', 'NE',
       'Calm', 'SE', 'East', 'SSE', 'South', nan], dtype=object)
```

```
X.loc[X["Wind_Direction"] == "CALM", "Wind_Direction"] = "Calm"
```

```
X.loc[X["Wind_Direction"] == "VAR", "Wind_Direction"] = "Variable"
X.loc[X["Wind_Direction"] == "East", "Wind_Direction"] = "E"
X.loc[X["Wind_Direction"] == "North", "Wind_Direction"] = "N"
X.loc[X["Wind_Direction"] == "South", "Wind_Direction"] = "S"
X.loc[X["Wind_Direction"] == "West", "Wind_Direction"] = "W"

X["Wind_Direction"] = X["Wind_Direction"].map(lambda x : x if len(x) != 3 else x[1:],

X["Wind_Direction"].unique()
```

```
array(['NW', 'W', 'NE', 'Calm', 'SW', 'Variable', 'S', 'E', 'N', 'SE',
       nan], dtype=object)
```

```
X.isna().sum()
```

```
Severity                 0
Start_Lat                0
Start_Lng                0
Distance(mi)             0
City                     0
Temperature(F)          85
Humidity(%)             89
Pressure(in)            69
Visibility(mi)          87
Wind_Direction          86
Wind_Speed(mph)        293
Precipitation(in)     1200
Weather_Condition       94
Amenity                  0
Bump                     0
Crossing                 0
Give_Way                 0
Junction                 0
No_Exit                  0
Railway                  0
Roundabout               0
Station                  0
Stop                     0
Traffic_Calming          0
Traffic_Signal           0
Civil_Twilight           6
Year                   369
Month                  369
Weekday                369
Day                    369
dtype: int64
```

```
features_to_fill = ["Temperature(F)", "Humidity(%)", "Pressure(in)", "Visibility(mi)",
X[features_to_fill] = X[features_to_fill].fillna(X[features_to_fill].mean())

X.dropna(inplace=True)

X.isna().sum()
```

```
Severity              0
Start_Lat             0
Start_Lng             0
Distance(mi)          0
```

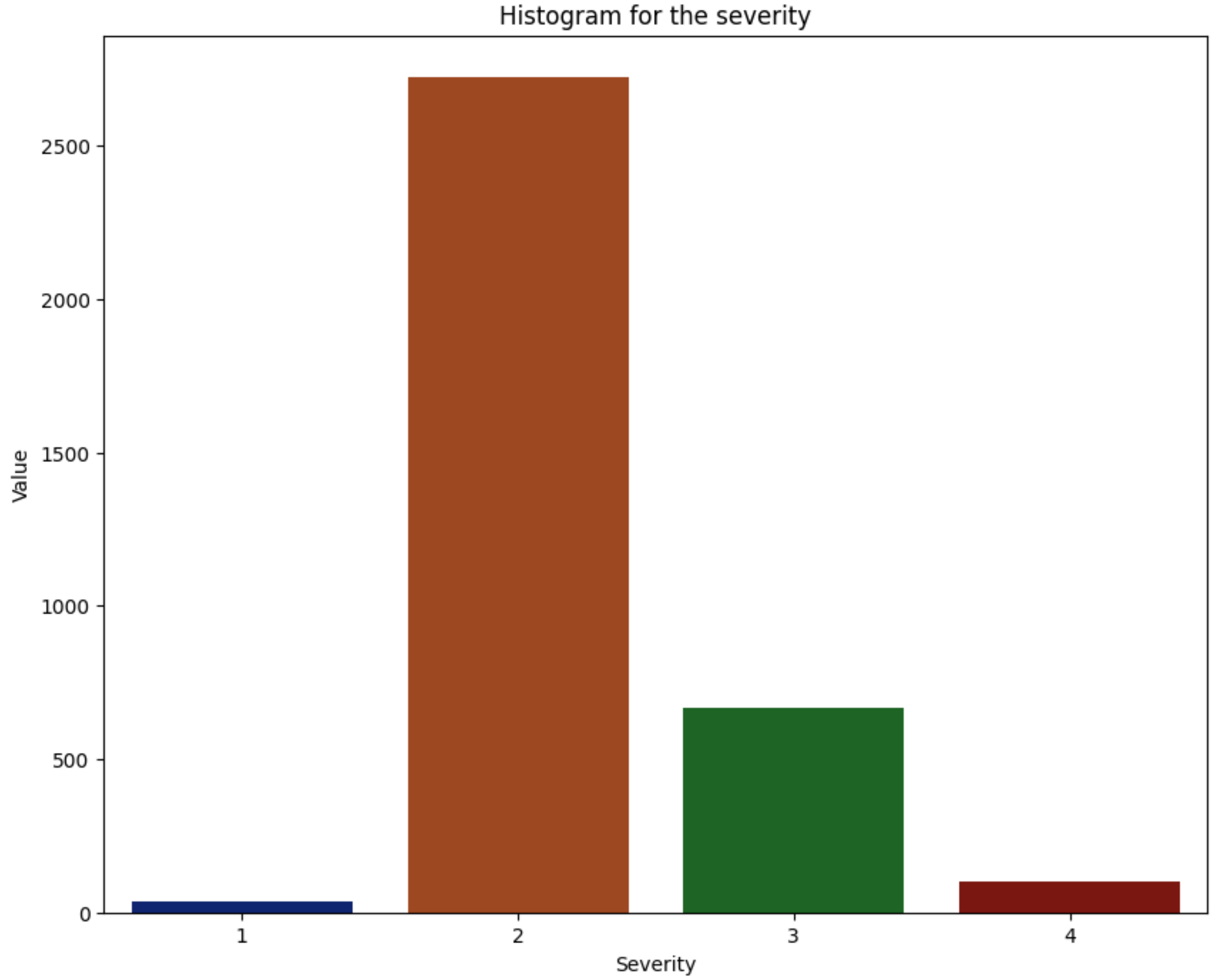
```
Distance(mi)         0
City                 0
Temperature(F)       0
Humidity(%)          0
Pressure(in)         0
Visibility(mi)       0
Wind_Direction       0
Wind_Speed(mph)      0
Precipitation(in)    0
Weather_Condition    0
Amenity              0
Bump                 0
Crossing             0
Give_Way             0
Junction             0
No_Exit              0
Railway              0
Roundabout           0
Station              0
Stop                 0
Traffic_Calming      0
Traffic_Signal       0
Civil_Twilight       0
Year                 0
Month                0
Weekday              0
Day                  0
dtype: int64
```

```
X.describe().round(2)
```

	Severity	Start_Lat	Start_Lng	Distance(mi)	Temperature(F)	Humidity(%)
count	3524.00	3524.00	3524.00	3524.00	3524.00	3524.00
mean	2.24	36.24	-94.92	0.44	62.18	65.01
std	0.51	5.15	17.37	1.14	18.47	22.98
min	1.00	24.88	-124.35	0.00	-35.00	4.00
25%	2.00	33.41	-117.24	0.00	50.00	48.00
50%	2.00	35.75	-87.97	0.01	64.00	67.00
75%	2.00	40.14	-80.46	0.35	75.90	84.00
max	4.00	48.90	-70.21	18.97	115.00	100.00

```
severity_counts = X["Severity"].value_counts()

plt.figure(figsize=(10, 8))
plt.title("Histogram for the severity")
sns.barplot(x=severity_counts.index, y=severity_counts.values, palette="dark")
plt.xlabel("Severity")
plt.ylabel("Value")
plt.show()
```

```
<ipython-input-223-52f12ea84180>:5: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```

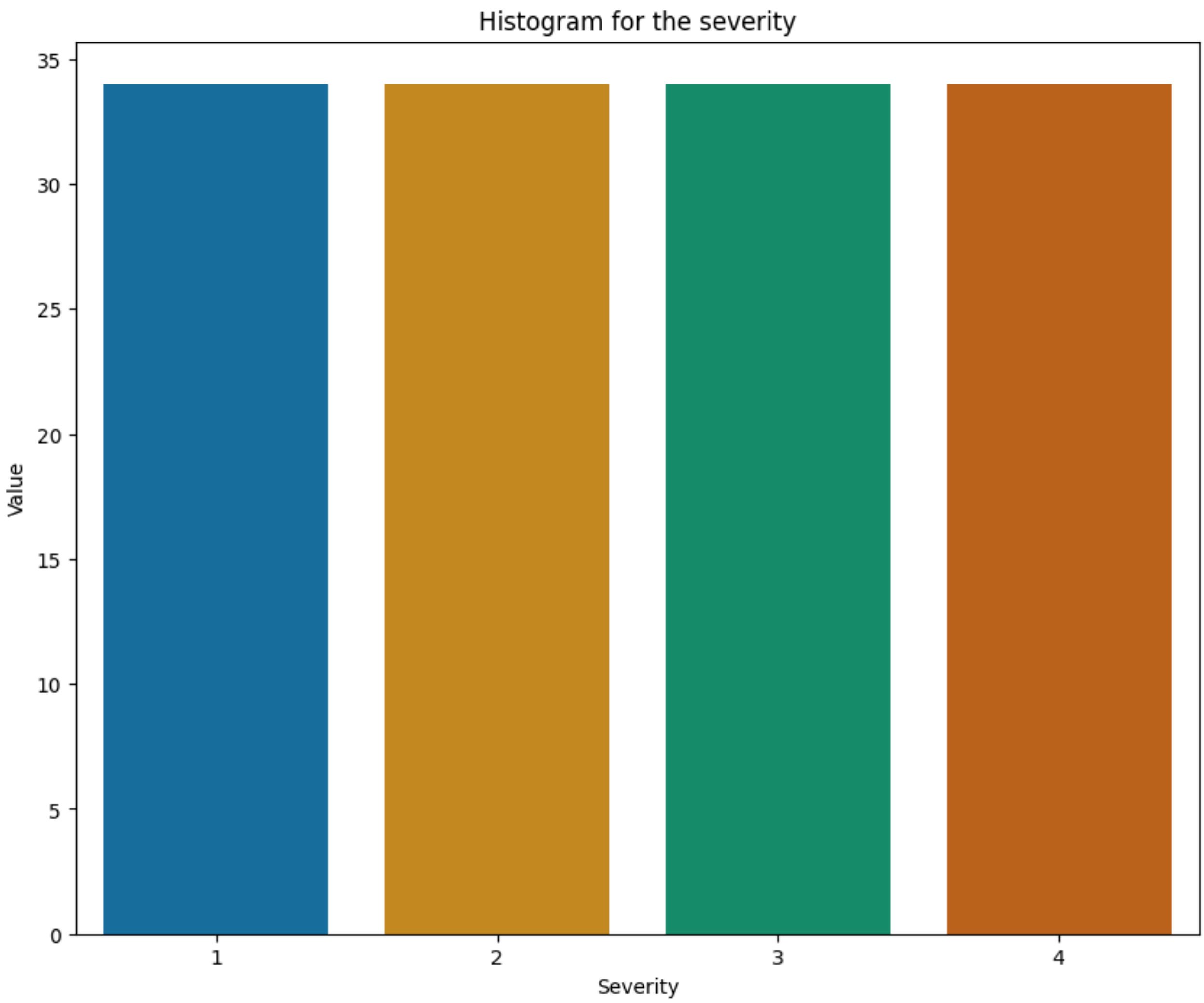



```
size = len(X[X["Severity"]==1].index)
df = pd.DataFrame()
for i in range(1,5):
    S = X[X["Severity"]==i]
    df = pd.concat([df, S.sample(size, random_state=42)], ignore_index=True)
X = df
```

```
severity_counts = X["Severity"].value_counts()
```

```
severity_counts = X['Severity'].value_counts()
plt.figure(figsize=(10, 8))
plt.title("Histogram for the severity")
sns.barplot(x=severity_counts.index, y=severity_counts.values, palette="colorblind")
plt.xlabel("Severity")
plt.ylabel("Value")
plt.show()
```

```
<ipython-input-225-873c45145721>:4: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
```




```
# Select only numeric features for scaling
```

```
numeric_features = ['Temperature(F)', 'Distance(mi)', 'Humidity(%)', 'Pressure(in)',
                    'Visibility(mi)', 'Wind_Speed(mph)', 'Precipitation(in)',
                    'Start_Lng', 'Start_Lat', 'Year', 'Month', 'Day']

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Scale the selected numeric features
X[numeric_features] = scaler.fit_transform(X[numeric_features])

# Display the first few rows of the scaled DataFrame
X.head()
```

	Severity	Start_Lat	Start_Lng	Distance(mi)	City	Temperature(F)	Humic
0	1	0.646508	0.348937	0.000000	Brighton	0.449064	0
1	1	0.993655	0.014886	0.257687	Seattle	0.667360	0
2	1	0.750957	0.989122	0.000000	Westborough	0.407484	0
3	1	0.579510	0.006255	0.000000	Santa Rosa	0.854470	0
4	1	0.411929	0.767306	0.000000	Mountain Rest	0.823285	0

5 rows × 30 columns

```
# Ensure all columns in categorical_features exist in X
missing_columns = [cat for cat in categorical_features if cat not in X.columns]
if missing_columns:
    print("The following columns are missing in X:", missing_columns)
else:
    # Convert columns to categorical dtype
    for cat in categorical_features:
        X[cat] = X[cat].astype("category")

    # Display DataFrame info
    X.info()
```

```
The following columns are missing in X: ['Side']
```

```
# Print the names of the features
print("Features in the dataset:")
for feature in df.columns:
    print(feature)
```

```
Features in the dataset:
Severity
Start_Lat
Start_Lng
Distance(mi)
City
Temperature(F)
Humidity(%)
Pressure(in)
```

```
Visibility(mi)
Wind_Direction
Wind_Speed(mph)
Precipitation(in)
Weather_Condition
Amenity
Bump
Crossing
Give_Way
Junction
No_Exit
Railway
Roundabout
Station
Stop
Traffic_Calming
Traffic_Signal
Civil_Twilight
Year
Month
Weekday
Day
```

Model : Artificial Neural Network

```
X.shape
```

```
(136, 30)
```

```
print(X_encoded.columns)
print(X_sample.columns)
print(y_sample)
```

```
Index([], dtype='object')
Index(['Start_Lat', 'Start_Lng', 'Distance(mi)', 'City', 'Temperature(F)',
       'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind_Direction',
       'Wind_Speed(mph)', 'Precipitation(in)', 'Weather_Condition', 'Amenity',
       'Bump', 'Crossing', 'Give_Way', 'Junction', 'No_Exit', 'Railway',
       'Roundabout', 'Station', 'Stop', 'Traffic_Calming', 'Traffic_Signal',
       'Civil_Twilight', 'Year', 'Month', 'Weekday', 'Day', 'Side'],
      dtype='object')
1060    4
510     2
182     1
942     3
30      1
       ..
1222    4
145     1
445     2
1217    4
1125    4
Name: Severity, Length: 813, dtype: int64
```

```
# Drop the unseen features from X_encoded
```

```
# Drop the unseen features from X_encoded
"""X_encoded = X_encoded.drop(columns=['Civil_Twilight_Day', 'Civil_Twilight_Night',
       'Weekday_Friday', 'Weekday_Monday', 'Weekday_Saturday',
       'Weekday_Sunday', 'Weekday_Thursday', 'Weekday_Tuesday',
       'Weekday_Wednesday',])
"""
#X_encoded = X_encoded.drop(columns=['City', 'Civil_Twilight', 'Weather_Condition', 'W

# Add the missing features to X_encoded
X_encoded['City_Aberdeen'] = 0
X_encoded['City_Abington'] = 0
X_encoded['City_Adamstown'] = 0
X_encoded['City_Adamsville'] = 0
X_encoded['City_Addison'] = 0
```

```
# Step 2: If the shapes are different, subset y to align with X
if X.shape[0] != y.shape[0]:
    y = y.iloc[:len(X)]  # Keep only the first len(X) rows of y

    # Confirm the shapes after alignment
    print("\nAfter aligning the number of samples:")
    print("Shape of X:", X.shape)
    print("Shape of y:", y.shape)
```

```
After aligning the number of samples:
Shape of X: (136, 30)
Shape of y: (136,)
```

```
# Print the first few rows of X
print(X.head())

# Print the number of rows and columns in X
print(X.shape)

# Print the data types of each column in X
print(X.dtypes)
```

```
   Severity  Start_Lat  Start_Lng  Distance(mi)  Temperature(F)  Humidity(%)  \
0         1   0.646508   0.348937      0.000000        0.449064     0.166667
1         1   0.993655   0.014886      0.257687        0.667360     0.468750
2         1   0.750957   0.989122      0.000000        0.407484     0.406250
3         1   0.579510   0.006255      0.000000        0.854470     0.145833
4         1   0.411929   0.767306      0.000000        0.823285     0.583333

   Pressure(in)  Visibility(mi)  Wind_Speed(mph)  Precipitation(in)  ...  \
0      0.191877        0.662162         0.226415                0.0  ...
1      0.928571        0.662162         0.113208                0.0  ...
2      0.745098        0.662162         0.490566                0.0  ...
3      0.896359        0.662162         0.301887                0.0  ...
4      0.743697        0.662162         0.490566                0.0  ...

   Weather_Condition_Windy  Civil_Twilight_Day  Civil_Twilight_Night  \
0                      0.0                 1.0                   0.0
1                      0.0                 1.0                   0.0
2                      0.0                 1.0                   0.0
3                      0.0                 1.0                   0.0
```

```
4                     0.0                1.0                   0.0

   Weekday_Friday  Weekday_Monday  Weekday_Saturday  Weekday_Sunday  \
0             0.0             0.0               0.0             0.0
1             0.0             0.0               0.0             0.0
2             0.0             0.0               0.0             0.0
3             0.0             0.0               0.0             0.0
4             0.0             0.0               0.0             0.0

   Weekday_Thursday  Weekday_Tuesday  Weekday_Wednesday
0               0.0              1.0                0.0
1               0.0              1.0                0.0
2               0.0              1.0                0.0
3               0.0              0.0                1.0
4               1.0              0.0                0.0

[5 rows x 164 columns]
(136, 164)
Severity               int64
Start_Lat            float64
Start_Lng            float64
Distance(mi)         float64
Temperature(F)       float64
                      ...
Weekday_Saturday     float64
Weekday_Sunday       float64
Weekday_Thursday     float64
Weekday_Tuesday      float64
Weekday_Wednesday    float64
Length: 164, dtype: object
```

```
categorical_columns = ['City', 'Wind_Direction', 'Weather_Condition', 'Weekday', 'Day'

encoder = OneHotEncoder(sparse=False)
X_encoded = pd.DataFrame(encoder.fit_transform(X[categorical_columns]))
X_encoded.columns = encoder.get_feature_names_out(categorical_columns)

# Drop the original categorical columns from X
X = X.drop(columns=categorical_columns)

# Concatenate the encoded columns with the remaining columns in X
X = pd.concat([X, X_encoded], axis=1)
```

```
# Encode categorical variables using one-hot encoding
categorical_columns = X.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(sparse=False)
X_encoded = pd.DataFrame(encoder.fit_transform(X[categorical_columns]))
X_encoded.columns = encoder.get_feature_names_out(categorical_columns)

boolean_indexer = ~X.index.isin(X[X['City'].isin(['Aberdeen'])].index)
print(len(boolean_indexer))

# Drop original categorical columns from X
X = X.drop(columns=categorical_columns)

# Concatenate X_encoded with X
```

```
# Concatenate X_encoded with X
X = pd.concat([X, X_encoded], axis=1)

for column in X:
  if X[column].dtype == "object":
    print(column)

X = X[~X.isin(['Aberdeen'])]
y = y[boolean_indexer]

le = LabelEncoder()
X['City'] = le.fit_transform(X['City'])
X['Wind_Direction'] = le.fit_transform(X['Wind_Direction'])
X['Weather_Condition'] = le.fit_transform(X['Weather_Condition'])
X['Weekday'] = le.fit_transform(X['Weekday'])
X['Day'] = le.fit_transform(X['Day'])
```

```
print("Missing values in X:", X.isnull().sum().sum())
print("Missing values in y:", y.isnull().sum())
```

```
"""mlp = MLPClassifier(random_state=42, verbose=False)
parameters = [{"hidden_layer_sizes": [(64, 32), (32, 64, 32)], "max_iter": [200], "solve
grid = GridSearchCV(mlp, parameters, verbose=5, n_jobs=-1)
"""
#Starting the timer
start_time = time.perf_counter()

mlp = MLPClassifier(random_state=42, verbose=False)
params = [{"hidden_layer_sizes": [(8, 16)], "max_iter": [450], "solver": ["lbfgs"], "act
grid_search = GridSearchCV(mlp, params, n_jobs=-1, verbose=5, cv=2, return_train_score =
grid_search.fit(X, y)

print("Best parameters scores:")
print(grid_search.best_params_)
print("Mean Train Score:", grid_search.cv_results_['mean_train_score'][0])
print("Mean Validation score:", grid_search.best_score_) #Mean cross-validation score of
print("Test score: ", grid_search.best_estimator_.score(X_test,y_test)) #Expected: 0.708

#Ending the timer
end_time = time.perf_counter()
total_time = end_time-start_time

print("It took {} secs for completing Multilayer Perceptron Training.".format(total_time
```

Double-click (or enter) to edit