```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF

from sklearn.metrics import classification_report

from sklearn.manifold import TSNE
import umap

from scipy.stats import entropy
import shap
```

```
df= pd.read_csv('/content/wd_mee_2.csv')
```

```
/tmp/ipython-input-1603384990.py:1: DtypeWarning: Columns (11,12,13,
  df= pd.read_csv('/content/wd_mee_2.csv')
```

```
df = df[
    ["YEAR", "MN", "DT", "HR",
     "MSLP", "FFF", "AW",
     "RF", "H", "P", "DW", "WAT"]
]
```

```
df.head()
```

	YEAR	MN	DT	HR	MSLP	FFF	AW	RF	H	P	DW	WAT
0	2000	1	1	12	1013.8	0	4	5		10	0.0	
1	2000	1	2	12	1013.8	0	5	3		10	0.0	
2	2000	1	3	12	1013.2	0	5	2		12	0.0	
3	2000	1	4	12	1013.1	4	4	3		8	0.0	
4	2000	1	5	12	1012.8	4	4	4		20	0.0	

Next steps: Generate code with df New interactive sheet

```python
# HR code → hour mapping
hr_map = {0:0, 12:3, 24:6, 36:9, 48:12, 60:15, 72:18, 84:21}
df["hour"] = df["HR"].map(hr_map)

df["time"] = pd.to_datetime(
    df["YEAR"].astype(str) + "-" +
    df["MN"].astype(str).str.zfill(2) + "-" +
    df["DT"].astype(str).str.zfill(2) + " " +
    df["hour"].astype(str).str.zfill(2) + ":00"
)

df = df.sort_values("time").reset_index(drop=True)
```

```python
df = df.rename(columns={
    "FFF": "wind_speed",
    "AW": "avg_wind",
    "MSLP": "mslp",
    "RF": "rainfall",
    "H": "wave_height",
    "P": "wave_period",
    "DW": "wave_direction",
    "WAT": "water_temp"
})
```

```python
df.head(2)
```

	YEAR	MN	DT	HR	mslp	wind_speed	avg_wind	rainfall	wave_heigh
0	2000	1	1	12	1013.8	0	4	5	
1	2000	1	1	48	1010.8	18	11	4	

Next steps: Generate code with df New interactive sheet

```python
df = df.fillna(method="ffill").dropna()
```

```
/tmp/ipython-input-1976003100.py:1: FutureWarning: DataFrame.fillna
  df = df.fillna(method="ffill").dropna()
```

```
df.dtypes
```

```
                        0
YEAR                int64
MN                  int64
DT                  int64
HR                  int64
mslp              float64
wind_speed         object
avg_wind           object
rainfall           object
wave_height        object
wave_period        object
wave_direction    float64
water_temp         object
hour                int64
time       datetime64[ns]

dtype: object
```

```
numeric_cols = [
    "wind_speed",
    "avg_wind",
    "mslp",
    "rainfall",
    "wave_height",
    "wave_period",
    "wave_direction",
    "water_temp"
]

for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors="coerce")
```

```
df.head(2)
```

	YEAR	MN	DT	HR	mslp	wind_speed	avg_wind	rainfall	wave_heigh
0	2000	1	1	12	1013.8	0.0	4.0	5.0	Na
1	2000	1	1	48	1010.8	18.0	11.0	4.0	Na

Next steps: Generate code with df New interactive sheet

```
# Drop columns with too many missing values
df = df.drop(columns=["wave_height", "wave_period", "wave_directior
```

```
df.head()
```

	YEAR	MN	DT	HR	mslp	wind_speed	avg_wind	rainfall	hour	ti
0	2000	1	1	12	1013.8	0.0	4.0	5.0	3	20 01- 03:00
1	2000	1	1	48	1010.8	18.0	11.0	4.0	12	20 01- 12:00
2	2000	1	1	72	1013.1	0.0	NaN	2.0	18	20 01-

Next steps: Generate code with df New interactive sheet

```
df["mslp_mean_8"] = df["mslp"].rolling(8, min_periods=3).mean()
df["pressure_anomaly"] = (df["mslp_mean_8"] - df["mslp"]).clip(lowe
```

```
df["wind_stress"] = df["wind_speed"] ** 2
```

```
df = df.dropna().reset_index(drop=True)
print("After feature engineering:", df.shape)
```

```
After feature engineering: (16251, 13)
```

```
def norm95(x):
    return x / x.quantile(0.95)

df["Vn"] = norm95(df["wind_speed"])
df["Pn"] = norm95(df["pressure_anomaly"])
df["Rn"] = norm95(df["rainfall"])

# Atmospheric Extreme Weather Severity Index
df["EWSI"] = df["Vn"] + df["Pn"] + df["Rn"]
```

```
thr = df["EWSI"].quantile(0.95)
df["extreme"] = (df["EWSI"] > thr).astype(int)

df["extreme"].value_counts()
```

```
         count
extreme
   0     15438
   1       813

dtype: int64
```

LAG FEATURES

```
for k in [1, 2]:
    df[f"wind_lag_{k}"] = df["wind_speed"].shift(k)
    df[f"press_lag_{k}"] = df["pressure_anomaly"].shift(k)

df = df.dropna().reset_index(drop=True)
print("Final dataset:", df.shape)
```

```
Final dataset: (16249, 22)
```

```
    v_thr = df["Vn"].quantile(0.95)
    p_thr = df["Pn"].quantile(0.95)
    r_thr = df["Rn"].quantile(0.95)

    df["wind_ext"]  = (df["Vn"]>v_thr).astype(int)
    df["press_ext"] = (df["Pn"]>p_thr).astype(int)
    df["rain_ext"]  = (df["Rn"]>r_thr).astype(int)


    def label(row):

        if row["EWSI"]<=thr:
            return 0

        h = row["wind_ext"]+row["press_ext"]+row["rain_ext"]

        if h>=2: return 3
        if row["wind_ext"]: return 1
        if row["press_ext"]: return 2

        return 3


    df["extreme_type"] = df.apply(label,axis=1)
```

FEATURES + TARGET

```
    X = df[
        ["wind_speed","pressure_anomaly",
         "rainfall","wind_stress",
         "wind_lag_1","wind_lag_2",
         "press_lag_1","press_lag_2"]
    ]

    y = df["extreme_type"]
```

SCALE

```
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

TRAIN TEST SPLIT

```
split = int(0.7*len(df))

X_train = X_scaled[:split]
X_test  = X_scaled[split:]

y_train = y.iloc[:split]
y_test  = y.iloc[split:]
```

RANDOM FOREST

```
rf = RandomForestClassifier(
    n_estimators=500,
    max_depth=16,
    min_samples_leaf=3,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train,y_train)
```

```
▾                      RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=16,
                       min_samples_leaf=3, n_estimators=500, n_jobs=
                       random_state=42)
```

```
print("Random Forest")
print(classification_report(y_test,rf.predict(X_test)))
```

```
Random Forest
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4571
           1       0.74      0.98      0.84        88
           2       0.81      0.62      0.70        89
           3       0.91      0.94      0.92       127

    accuracy                           0.98      4875
   macro avg       0.86      0.88      0.86      4875
weighted avg       0.98      0.98      0.98      4875
```

UMAP

```
um = umap.UMAP(
    n_neighbors=20,
    min_dist=0.1,
    random_state=42
)

U = um.fit_transform(X_scaled)

df["UMAP1"]=U[:,0]
df["UMAP2"]=U[:,1]
```

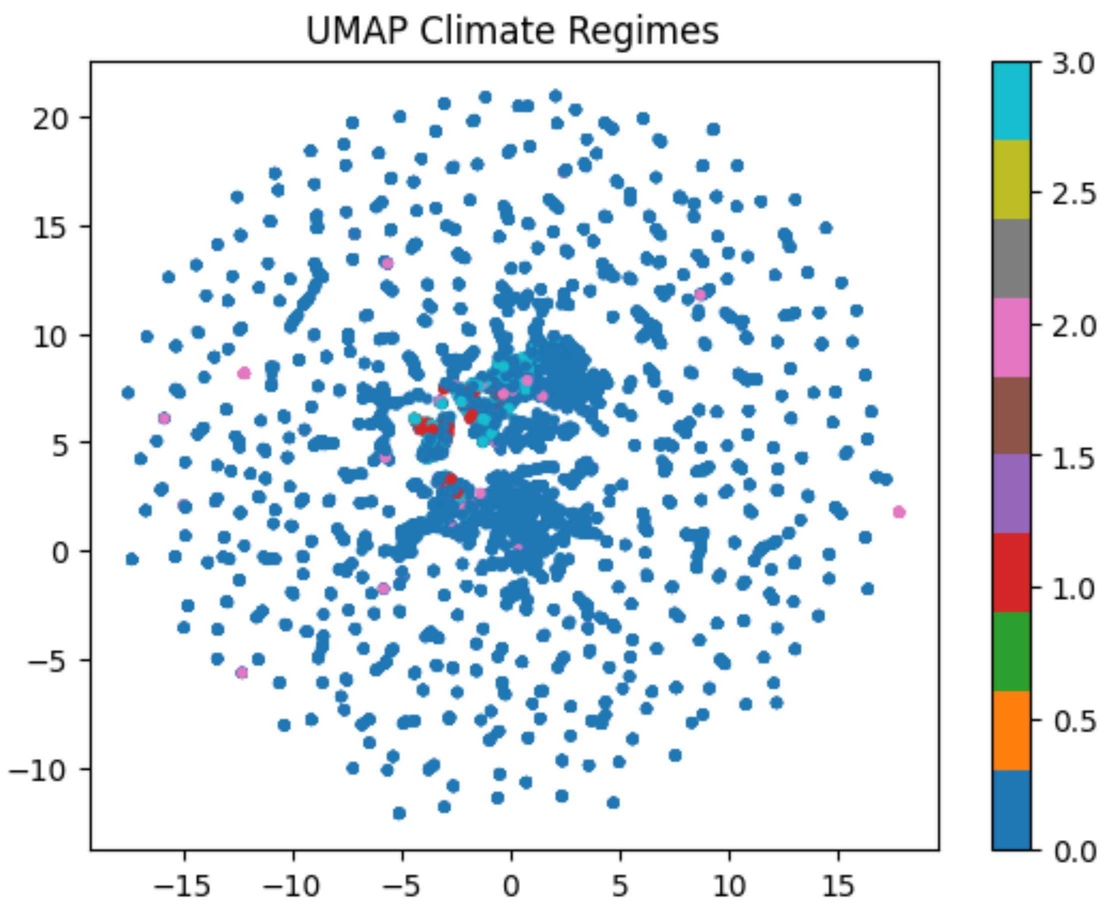
```
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarr
  warn(
```

t-SNE

```
ts = TSNE(
    n_components=2,
    perplexity=40,
    random_state=42
)

T = ts.fit_transform(X_scaled)

df["TSNE1"]=T[:,0]
df["TSNE2"]=T[:,1]
```

REGIME MAP

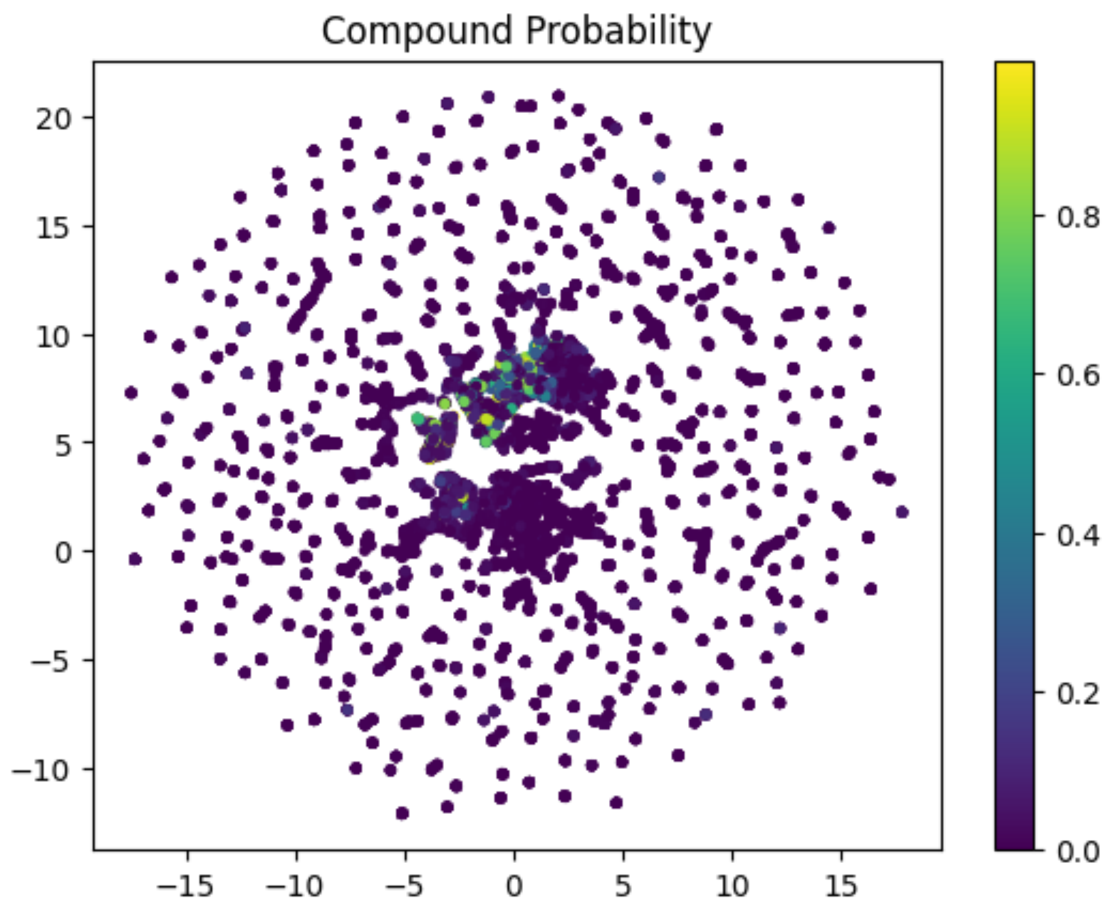
```
    plt.scatter(
        df["UMAP1"],df["UMAP2"],
        c=df["extreme_type"],
        cmap="tab10",s=8)

    plt.colorbar()
    plt.title("UMAP Climate Regimes")
    plt.show()
```



The UMAP plot visualizes the underlying structure of atmospheric conditions in a reduced two-dimensional space. Normal conditions form a widespread cluster, while extreme events are grouped within specific regions, indicating distinct climatic regimes. Compound extremes are concentrated in compact zones, reflecting the combined influence of multiple hazards. This confirms that extreme events are not randomly distributed but exhibit organized patterns in the feature space.

PROBABILITY MAP

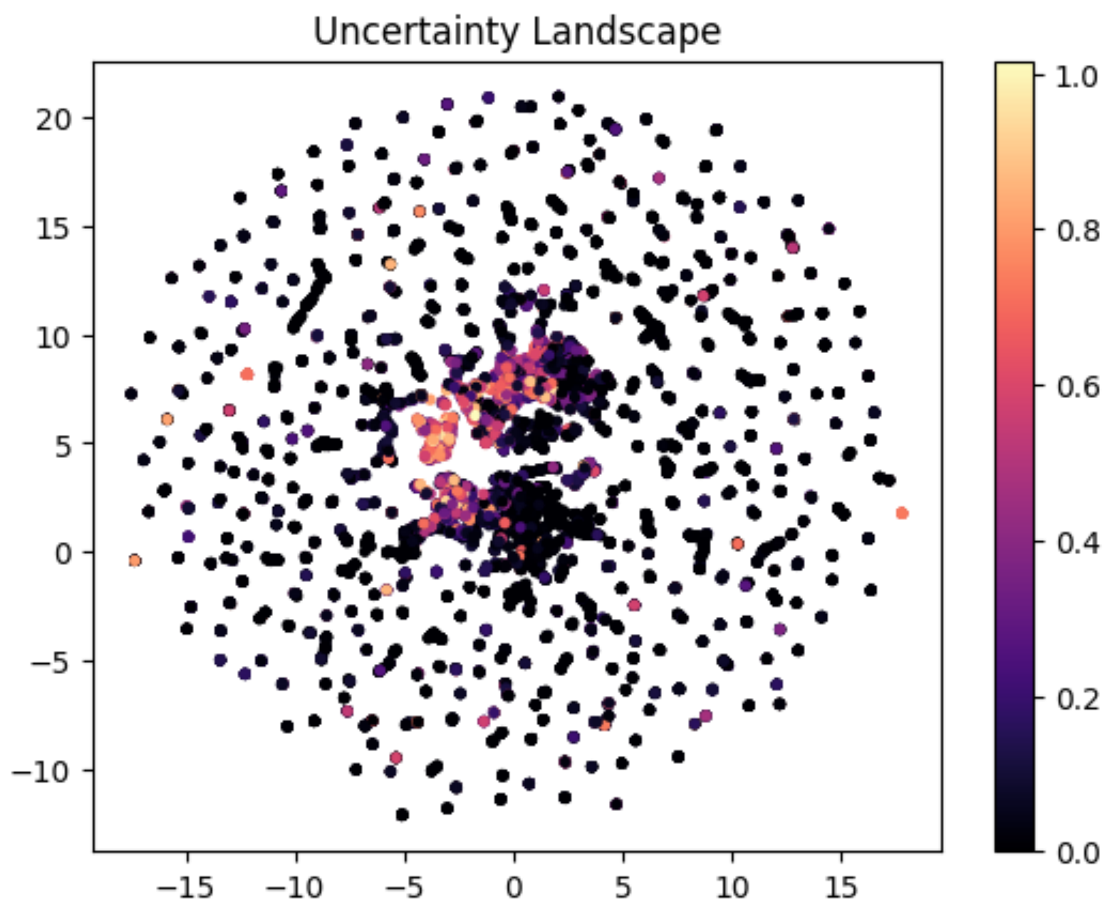
```
prob = rf.predict_proba(X_scaled)[:,3]

plt.scatter(
    df["UMAP1"],df["UMAP2"],
    c=prob,cmap="viridis",s=8)

plt.colorbar()
plt.title("Compound Probability")
plt.show()
```



The compound probability map shows that high-probability regions are primarily concentrated in the central and densely populated zones of the UMAP feature space, where multiple atmospheric variables exhibit strong interactions. These regions correspond to transitional regimes between normal and extreme conditions, characterized by elevated wind stress, pressure anomaly, and rainfall intensity. In contrast, peripheral and sparsely distributed regions display consistently low probabilities, indicating stable background conditions with limited compound hazard potential. The spatial localization of high-risk zones demonstrates that compound extreme events arise under specific multivariate configurations rather than being uniformly distributed across the feature space.

UNCERTAININITY MAP

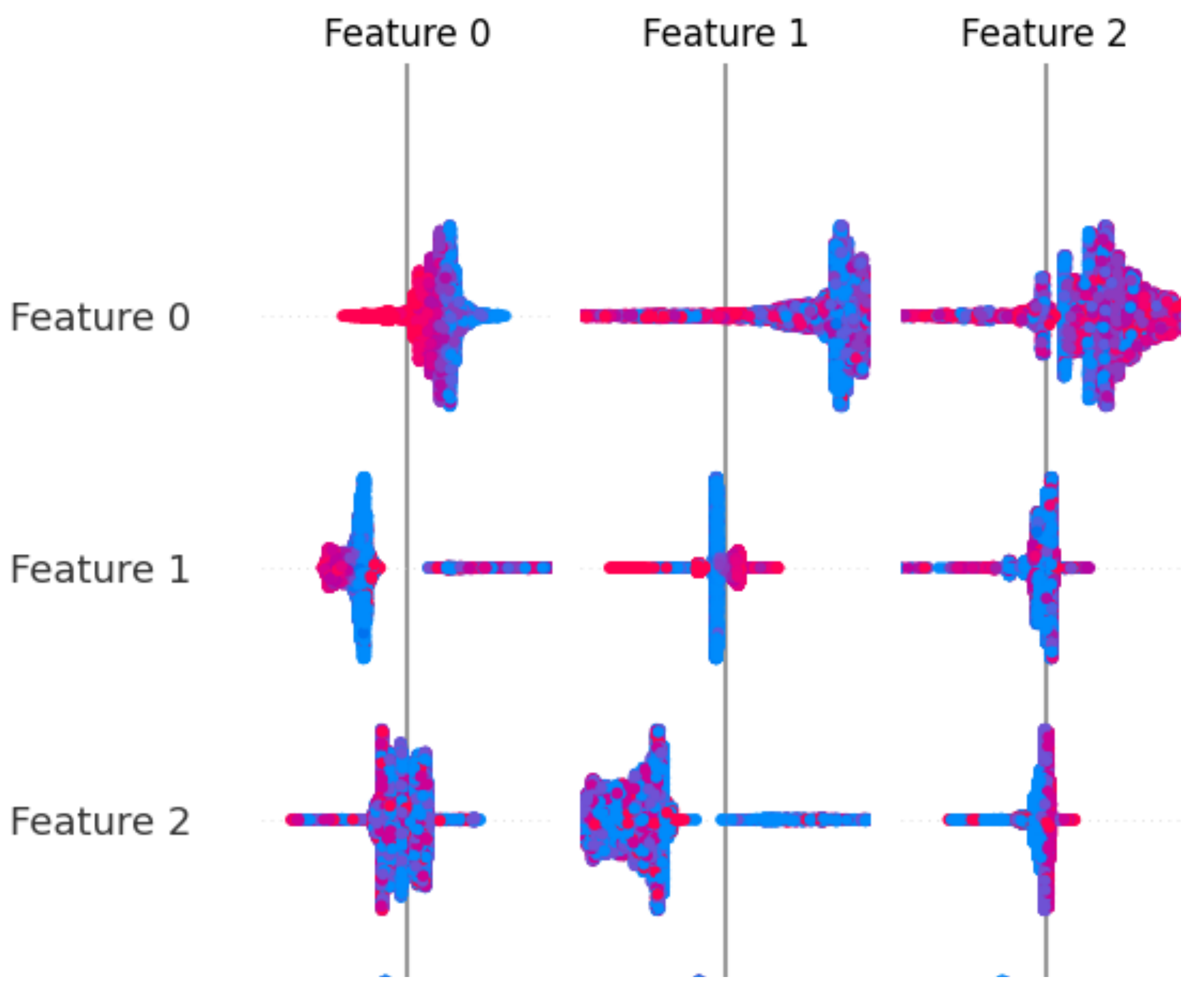
```
unc = entropy(rf.predict_proba(X_scaled).T)

plt.scatter(
    df["UMAP1"],df["UMAP2"],
    c=unc,cmap="magma",s=8)

plt.colorbar()
plt.title("Uncertainty Landscape")
plt.show()
```

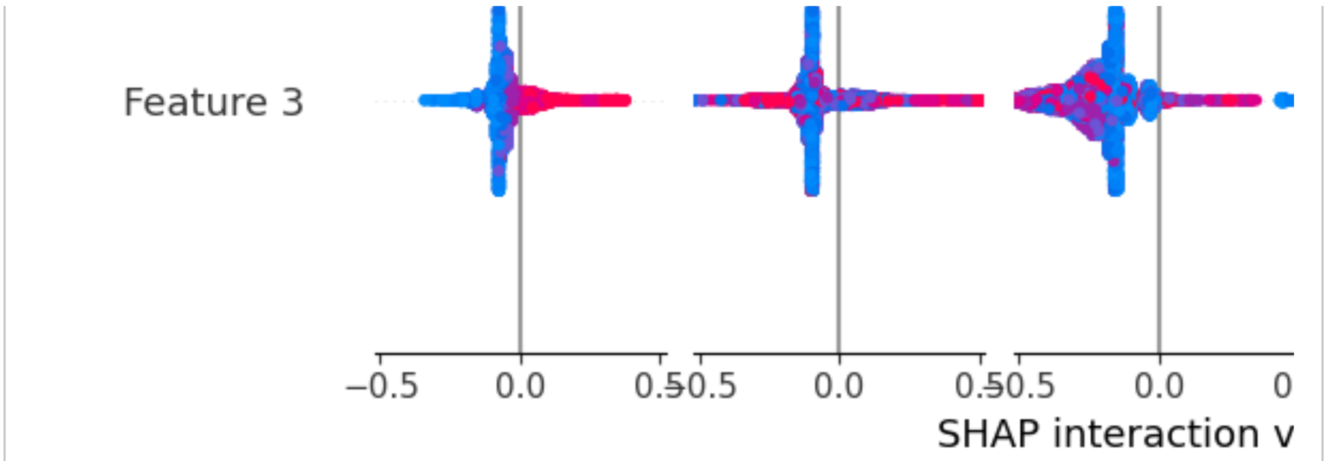


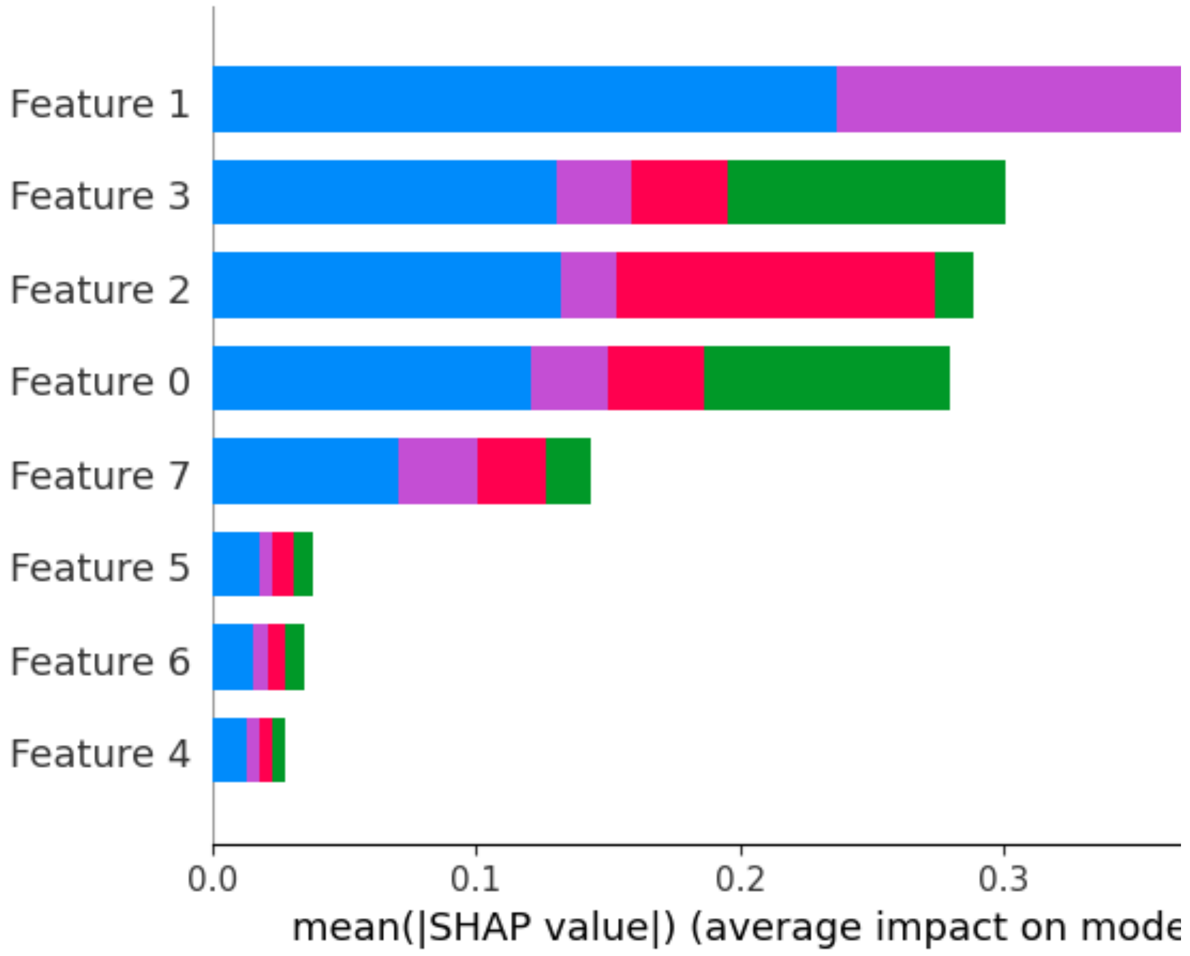
This figure presents the uncertainty landscape of the Random Forest classifier in the UMAP-reduced feature space, where color intensity represents the entropy-based prediction uncertainty. Regions of high uncertainty are primarily concentrated in transitional zones between normal and extreme regimes, indicating atmospheric states with overlapping characteristics. These areas correspond to conditions where the model exhibits ambiguity due to competing influences of multiple meteorological drivers. In contrast, low-uncertainty regions dominate the peripheral and densely clustered zones, reflecting stable regimes with well-defined classification patterns. This spatial distribution highlights that model uncertainty is systematically associated with regime boundaries rather than being randomly distributed.

SHAP

```
    expl = shap.TreeExplainer(rf)

    sv = expl.shap_values(X_train)

    shap.summary_plot(sv,X_train)
    shap.summary_plot(sv,X_train,plot_type="bar")
```



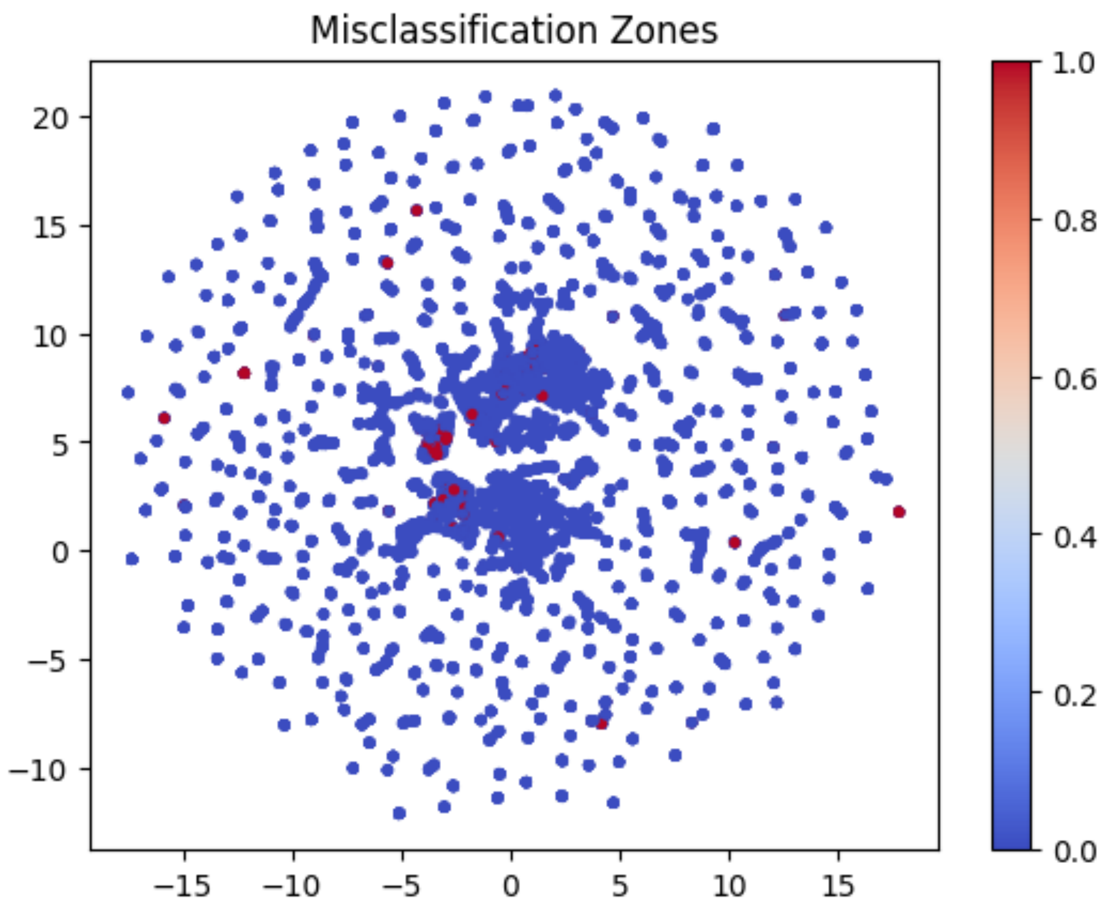

Feature 3
−0.5
0.0
0.5
−0.5
0.0
0.5
−0.5
0.0
0
SHAP interaction v


Feature 1
Feature 3
Feature 2
Feature 0
Feature 7
Feature 5
Feature 6
Feature 4
0.0
0.1
0.2
0.3
mean(|SHAP value|) (average impact on mode

The first graph illustrates the SHAP interaction values between pairs of input features, showing how combinations of variables jointly influence the model's predictions. Each subplot represents the interaction effect between two features, while the horizontal spread indicates the magnitude and direction of their contribution. The color gradient reflects the feature values, highlighting how low and high values modify the interaction strength. The concentration of points near zero suggests weak interactions for many feature pairs, whereas wider distributions indicate strong nonlinear dependencies. This plot demonstrates that compound extreme predictions are driven not only by individual variables but also by their interactions, particularly among the dominant meteorological drivers.

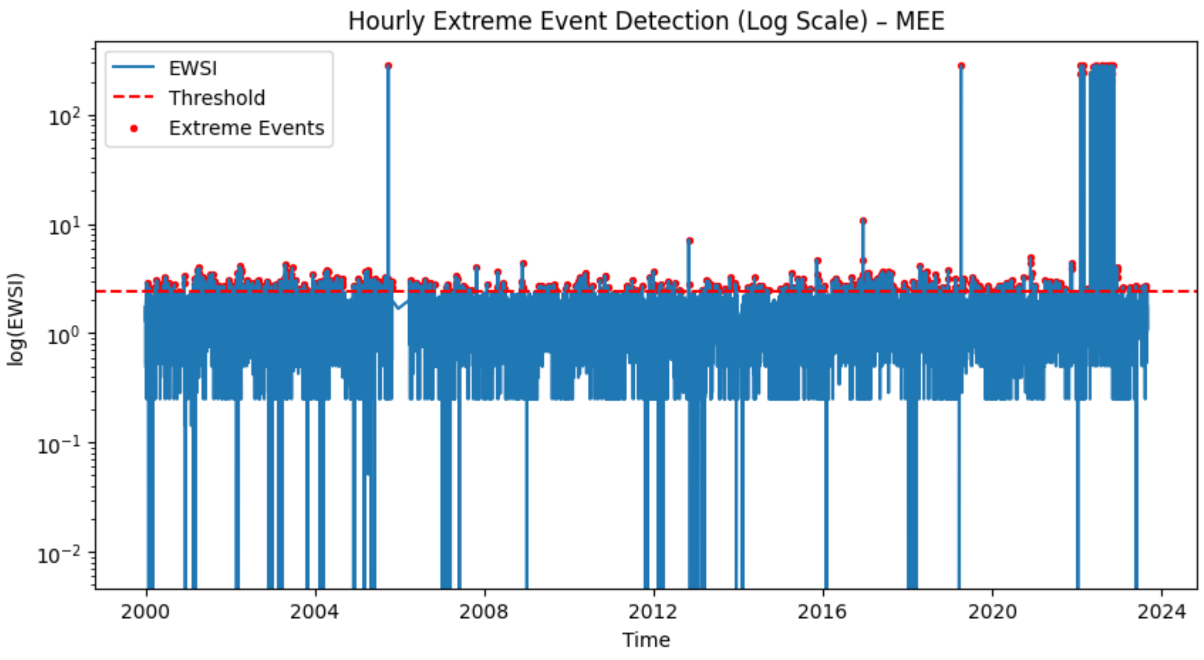
The other graph presents the class-wise mean absolute SHAP values for each feature, representing their average contribution to the prediction of different extreme event categories. The stacked bars show how individual variables influence normal, wind-dominated, pressure-dominated, and compound extreme classes. The results indicate that a small subset of features accounts for most of the predictive power, with Feature 1, Feature 3, and Feature 2 exhibiting the strongest overall influence. Differences in color composition across bars reveal that feature relevance varies by class, reflecting distinct physical drivers for different extreme regimes. This analysis enhances model interpretability by linking classification outcomes to dominant atmospheric variables.

```
#ERROR MAP

df["pred"] = rf.predict(X_scaled)
df["err"] = (df["pred"]!=df["extreme_type"])

plt.scatter(
    df["UMAP1"],df["UMAP2"],
    c=df["err"],cmap="coolwarm",s=8)

plt.colorbar()
plt.title("Misclassification Zones")
plt.show()
```



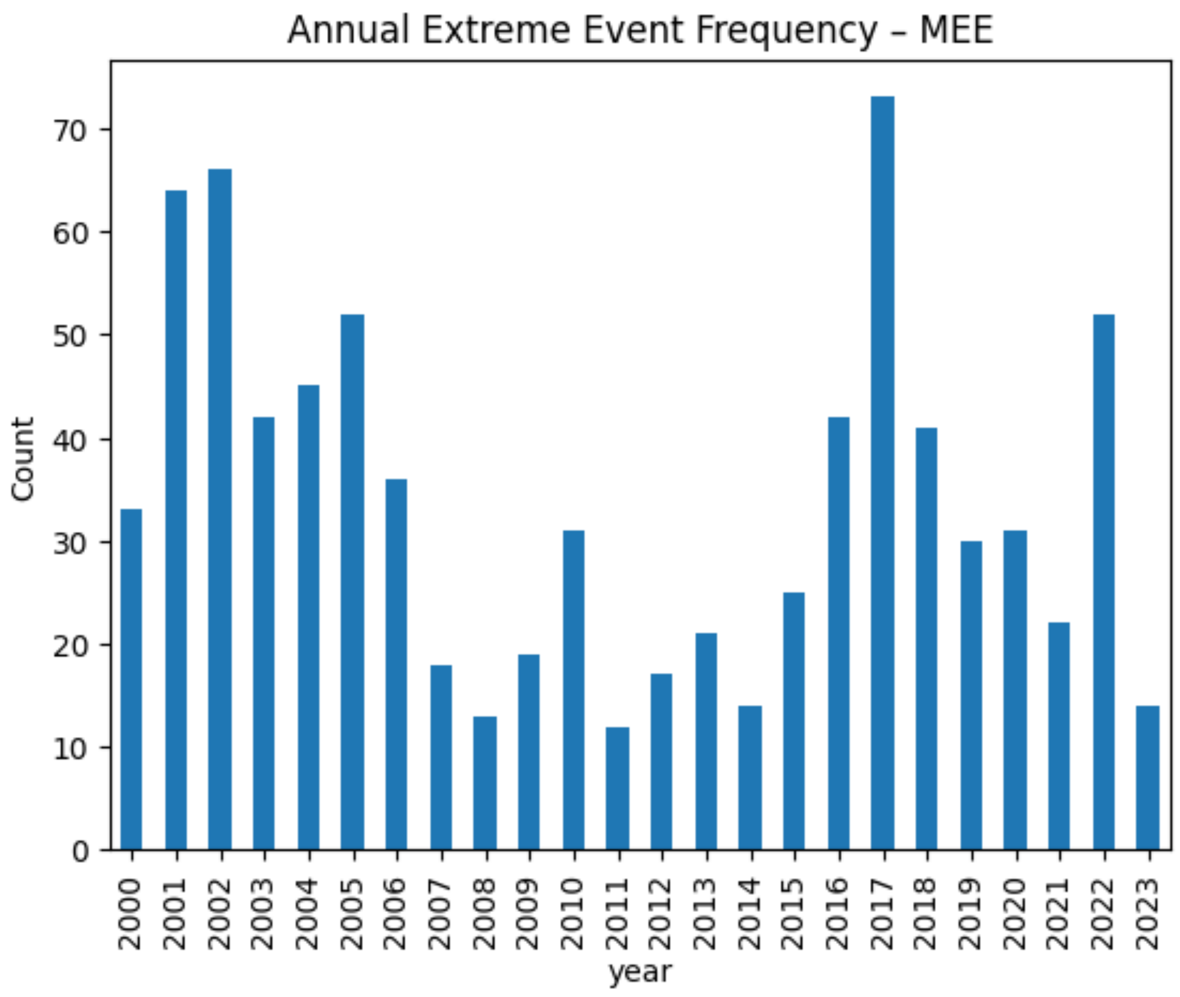
This graph illustrates the spatial distribution of misclassified observations in the UMAP-reduced feature space, where correctly classified instances are shown in blue and misclassified instances are highlighted in red. The majority of data points are accurately classified, indicating strong overall model performance. Misclassifications are primarily concentrated within the central transitional regions, where atmospheric conditions exhibit overlapping characteristics between different extreme regimes. These zones correspond to complex multivariate interactions that increase classification ambiguity. In contrast, peripheral and well-defined regions show minimal errors, reflecting stable atmospheric states with high prediction reliability. This analysis demonstrates that classification errors are systematically associated with regime boundaries rather than occurring randomly.

```
plt.figure(figsize=(10,5))

plt.plot(df["time"], df["EWSI"], label="EWSI")
plt.yscale("log")

plt.axhline(thr, color="red", linestyle="--", label="Threshold")

plt.scatter(
    df.loc[df["extreme"] == 1, "time"],
    df.loc[df["extreme"] == 1, "EWSI"],
    s=8, color="red", label="Extreme Events"
)

plt.legend()
plt.title("Hourly Extreme Event Detection (Log Scale) – MEE")
plt.xlabel("Time")
plt.ylabel("log(EWSI)")
plt.show()
```



The above figure illustrates the temporal evolution of the Extreme Weather Severity Index (EWSI) at the Meenambaakam station on a logarithmic scale. The majority of observations lie within a low-amplitude regime corresponding to normal atmospheric conditions, while rare, high-amplitude excursions represent extreme states. A data-driven threshold defined by the 95th percentile of EWSI is used to identify extreme events, which are shown to occur in temporally clustered bursts. The heavy-tailed and intermittent nature of the series motivates both the percentile-based extreme definition and the inclusion of temporal context in subsequent classification models.

```
df["year"] = df["time"].dt.year
annual = df.groupby("year")["extreme"].sum()

annual.plot(kind="bar")
plt.title("Annual Extreme Event Frequency – MEE")
plt.ylabel("Count")
plt.show()
```



The above figure shows the annual frequency of hourly extreme atmospheric events detected at the MEE station. The results reveal pronounced interannual variability, with certain years exhibiting substantially higher counts of extreme conditions. These variations indicate changes in the persistence and clustering of extreme atmospheric regimes rather than isolated events. The absence of a monotonic trend suggests that the extreme event occurrence is governed by year-to-year dynamical variability rather than a simple linear progression.