Binary Prediction with a Rainfall Dataset

April 5, 2025

1 Binary Prediction with a Rainfall Dataset

1.0.1 Importing and loading necessary libraries and packages

[1]:
```
import math
import warnings
from itertools import combinations
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style='white', palette='Set2')
pal = sns.color_palette('Set2')
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import␣
 ↪(train_test_split,StratifiedKFold,cross_val_score)
from sklearn.metrics import (roc_curve,roc_auc_score,classification_report,auc)
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils.class_weight import compute_class_weight
import xgboost as xgb
from xgboost import XGBClassifier
xgb.set_config(verbosity=0)
import lightgbm
from lightgbm import LGBMClassifier, plot_importance
import optuna
from optuna.samplers import TPESampler
import logging
optuna.logging.set_verbosity(logging.WARNING)
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore")
```

1.0.2 Loading Data, Data Preprocessing, and Visualizing Data

[2]:
```
file_path = '/Users/nav/Documents/Dataset/Rainfall Dataset/train.csv'
file_path1 = '/Users/nav/Documents/Dataset/Rainfall Dataset/test.csv'
```

[3]:
```
train_df = pd.read_csv(file_path)
test_df = pd.read_csv(file_path1)
```

[4]:
```
train_df.head()
```

[4]:

	id	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	\
0	0	1	1017.4	21.2	20.6	19.9	19.4	87.0	
1	1	2	1019.5	16.2	16.9	15.8	15.4	95.0	
2	2	3	1024.1	19.4	16.1	14.6	9.3	75.0	
3	3	4	1013.4	18.1	17.8	16.9	16.8	95.0	
4	4	5	1021.8	21.3	18.4	15.2	9.6	52.0	

	cloud	sunshine	winddirection	windspeed	rainfall
0	88.0	1.1	60.0	17.2	1
1	91.0	0.0	50.0	21.9	1
2	47.0	8.3	70.0	18.1	1
3	95.0	0.0	60.0	35.6	1
4	45.0	3.6	40.0	24.8	0

[5]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2190 entries, 0 to 2189
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             2190 non-null   int64
 1   day            2190 non-null   int64
 2   pressure       2190 non-null   float64
 3   maxtemp        2190 non-null   float64
 4   temparature    2190 non-null   float64
 5   mintemp        2190 non-null   float64
 6   dewpoint       2190 non-null   float64
 7   humidity       2190 non-null   float64
 8   cloud          2190 non-null   float64
 9   sunshine       2190 non-null   float64
 10  winddirection  2190 non-null   float64
 11  windspeed      2190 non-null   float64
 12  rainfall       2190 non-null   int64
dtypes: float64(10), int64(3)
memory usage: 222.6 KB
```

[6]:
```
train_df.describe()
```

[6]:
```
                id          day     pressure      maxtemp  temparature  \
count  2190.000000  2190.000000  2190.000000  2190.000000  2190.000000
mean   1094.500000   179.948402  1013.602146    26.365799    23.953059
std     632.342866   105.203592     5.655366     5.654330     5.222410
min       0.000000     1.000000   999.000000    10.400000     7.400000
25%     547.250000    89.000000  1008.600000    21.300000    19.300000
50%    1094.500000   178.500000  1013.000000    27.800000    25.500000
75%    1641.750000   270.000000  1017.775000    31.200000    28.400000
max    2189.000000   365.000000  1034.600000    36.000000    31.500000

           mintemp     dewpoint     humidity        cloud     sunshine  \
count  2190.000000  2190.000000  2190.000000  2190.000000  2190.000000
mean     22.170091    20.454566    82.036530    75.721918     3.744429
std       5.059120     5.288406     7.800654    18.026498     3.626327
min       4.000000    -0.300000    39.000000     2.000000     0.000000
25%      17.700000    16.800000    77.000000    69.000000     0.400000
50%      23.850000    22.150000    82.000000    83.000000     2.400000
75%      26.400000    25.000000    88.000000    88.000000     6.800000
max      29.800000    26.700000    98.000000   100.000000    12.100000

       winddirection    windspeed     rainfall
count    2190.000000  2190.000000  2190.000000
mean      104.863151    21.804703     0.753425
std        80.002416     9.898659     0.431116
min        10.000000     4.400000     0.000000
25%        40.000000    14.125000     1.000000
50%        70.000000    20.500000     1.000000
75%       200.000000    27.900000     1.000000
max       300.000000    59.500000     1.000000
```

[7]:
```
train_df.isna().sum()
```

[7]:
```
id              0
day             0
pressure        0
maxtemp         0
temparature     0
mintemp         0
dewpoint        0
humidity        0
cloud           0
sunshine        0
winddirection   0
windspeed       0
rainfall        0
dtype: int64
```

[8]:
```
test_df.head()
```

[8]:

	id	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	cloud	sunshine	winddirection	windspeed
0	2190	1	1019.5	17.5	15.8	12.7	14.9	96.0	99.0	0.0	50.0	24.3
1	2191	2	1016.5	17.5	16.5	15.8	15.1	97.0	99.0	0.0	50.0	35.3
2	2192	3	1023.9	11.2	10.4	9.4	8.9	86.0	96.0	0.0	40.0	16.9
3	2193	4	1022.9	20.6	17.3	15.2	9.5	75.0	45.0	7.1	20.0	50.6
4	2194	5	1022.2	16.1	13.8	6.4	4.3	68.0	49.0	9.2	20.0	19.4

[9]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             730 non-null    int64  
 1   day            730 non-null    int64  
 2   pressure       730 non-null    float64
 3   maxtemp        730 non-null    float64
 4   temparature    730 non-null    float64
 5   mintemp        730 non-null    float64
 6   dewpoint       730 non-null    float64
 7   humidity       730 non-null    float64
 8   cloud          730 non-null    float64
 9   sunshine       730 non-null    float64
 10  winddirection  729 non-null    float64
 11  windspeed      730 non-null    float64
dtypes: float64(10), int64(2)
memory usage: 68.6 KB
```

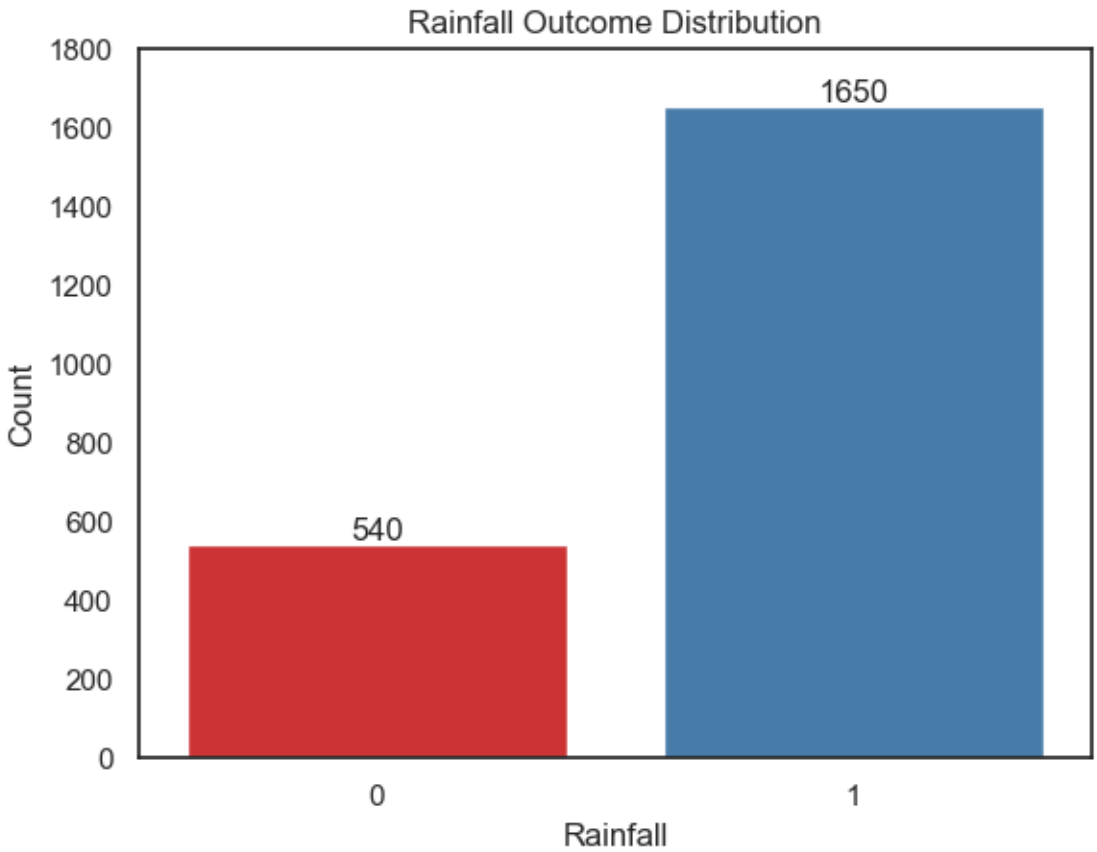
[10]:
```
test_df.isna().sum()
```

[10]:
```
id              0
day             0
pressure        0
maxtemp         0
temparature     0
mintemp         0
dewpoint        0
humidity        0
```

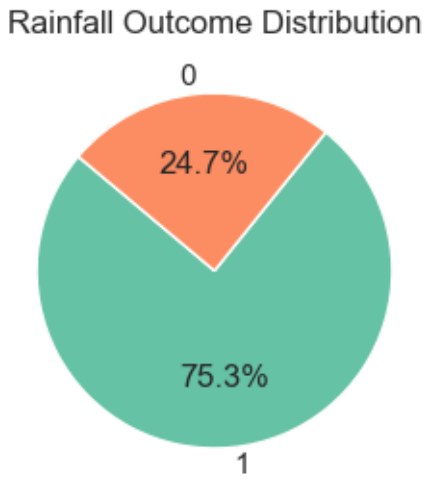
```
cloud            0
sunshine         0
winddirection    1
windspeed        0
dtype: int64
```

[11]:
```
# Fill the missing value in 'winddirection' with the median (since it's␣
 ↪directional)
test_df['winddirection'] = test_df['winddirection'].
 ↪fillna(test_df['winddirection'].median())
```

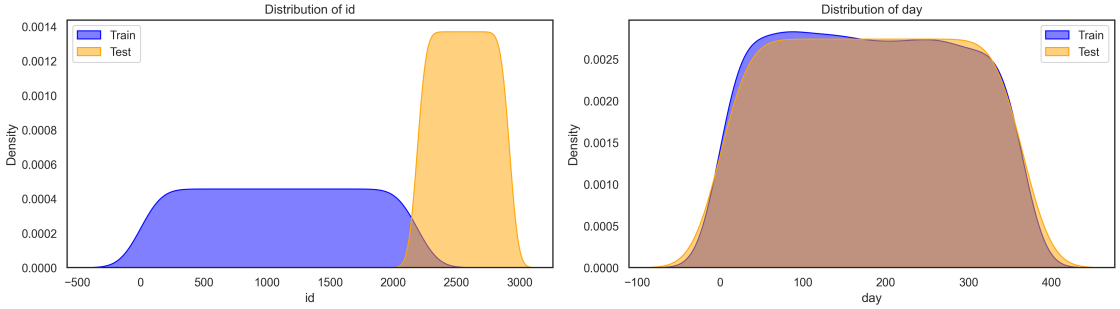
[12]:
```
ax = sns.countplot(x='rainfall', data=train_df, palette='Set1')
for container in ax.containers:
    ax.bar_label(container)

ax.set_ylabel('Count')
ax.set_xlabel('Rainfall')
ax.set_title('Rainfall Outcome Distribution')
ax.set_ylim(0, 1800)
plt.show()
```



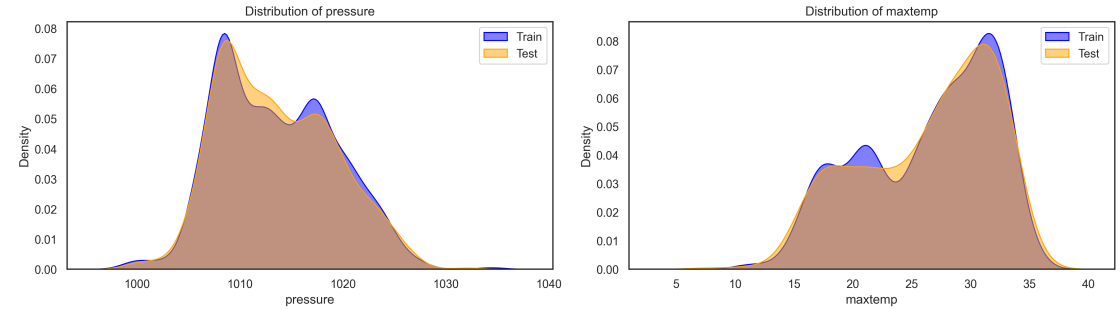
[13]:
```
data = train_df['rainfall'].value_counts()
plt.figure(figsize=(3, 3))
plt.pie(data, labels=data.index, autopct='%1.1f%%', startangle=140)
plt.title("Rainfall Outcome Distribution")
plt.show()
```



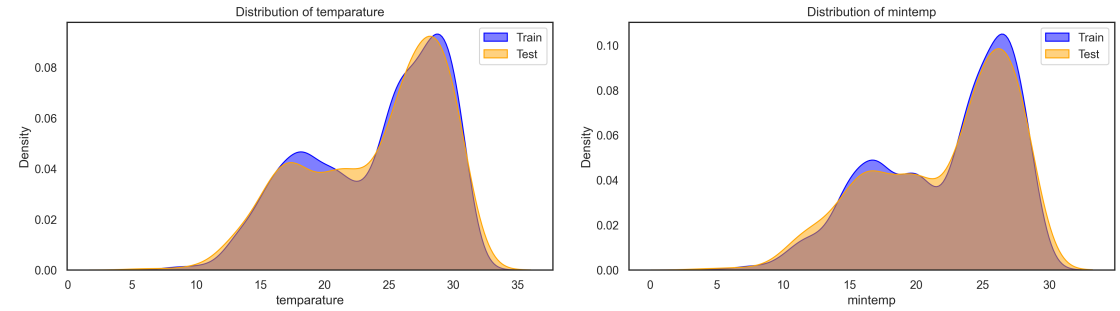
[14]:
```
# Define features
features = ['id', 'day', 'pressure', 'maxtemp', 'temparature', 'mintemp',
            'dewpoint', 'humidity', 'cloud', 'sunshine', 'winddirection',␣
 ↪'windspeed']
fig, ax = plt.subplots(6, 2, figsize=(15, 25), dpi=300)
ax = ax.flatten()

# Plot KDE for both train_df and test_df
for i, feature in enumerate(features):
    sns.kdeplot(train_df[feature], ax=ax[i], fill=True, label='Train',␣
 ↪color='blue', alpha=0.5)
    sns.kdeplot(test_df[feature], ax=ax[i], fill=True, label='Test',␣
 ↪color='orange', alpha=0.5)
    ax[i].set_title(f'Distribution of {feature}')
    ax[i].set_xlabel(feature)
    ax[i].set_ylabel('Density')
    ax[i].legend()
plt.tight_layout()
plt.show()
```


Distribution of id
Train
Test
Density
id

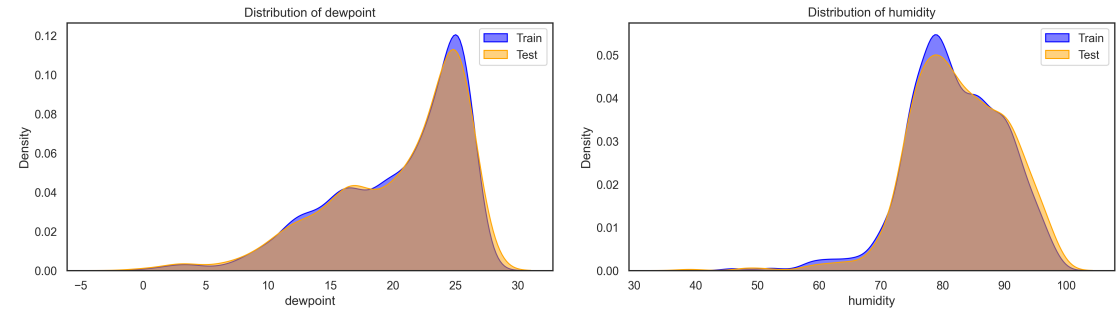
Distribution of day
Train
Test
Density
day


Distribution of pressure
Train
Test
Density
pressure

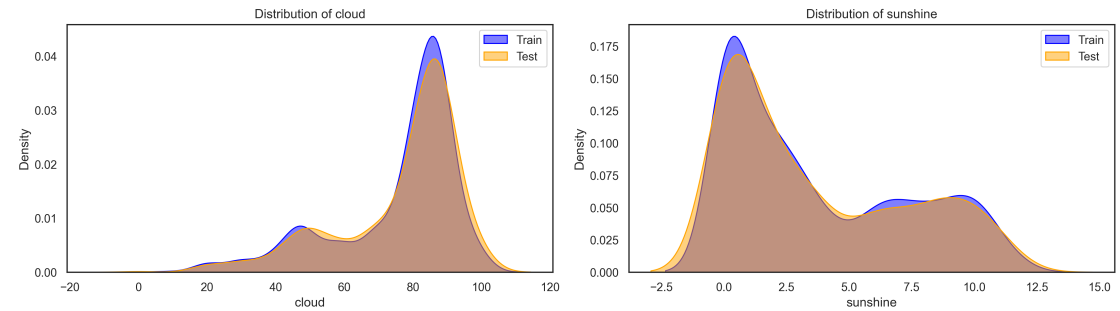
Distribution of maxtemp
Train
Test
Density
maxtemp


Distribution of temparature
Train
Test
Density
temparature

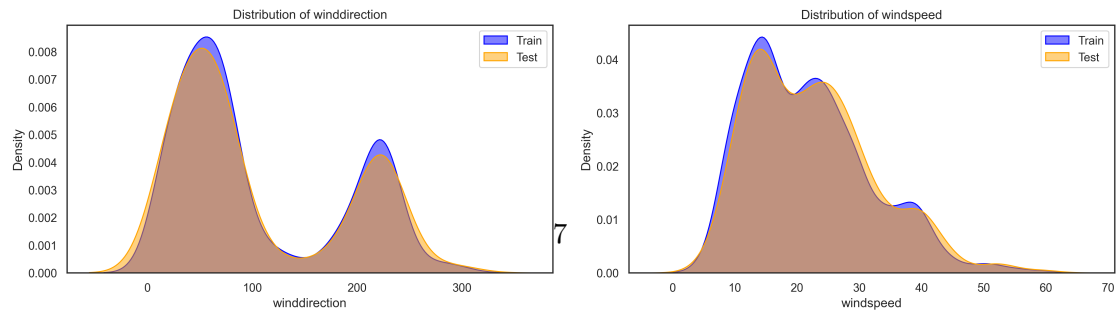
Distribution of mintemp
Train
Test
Density
mintemp


Distribution of dewpoint
Train
Test
Density
dewpoint

Distribution of humidity
Train
Test
Density
humidity


Distribution of cloud
Train
Test
Density
cloud

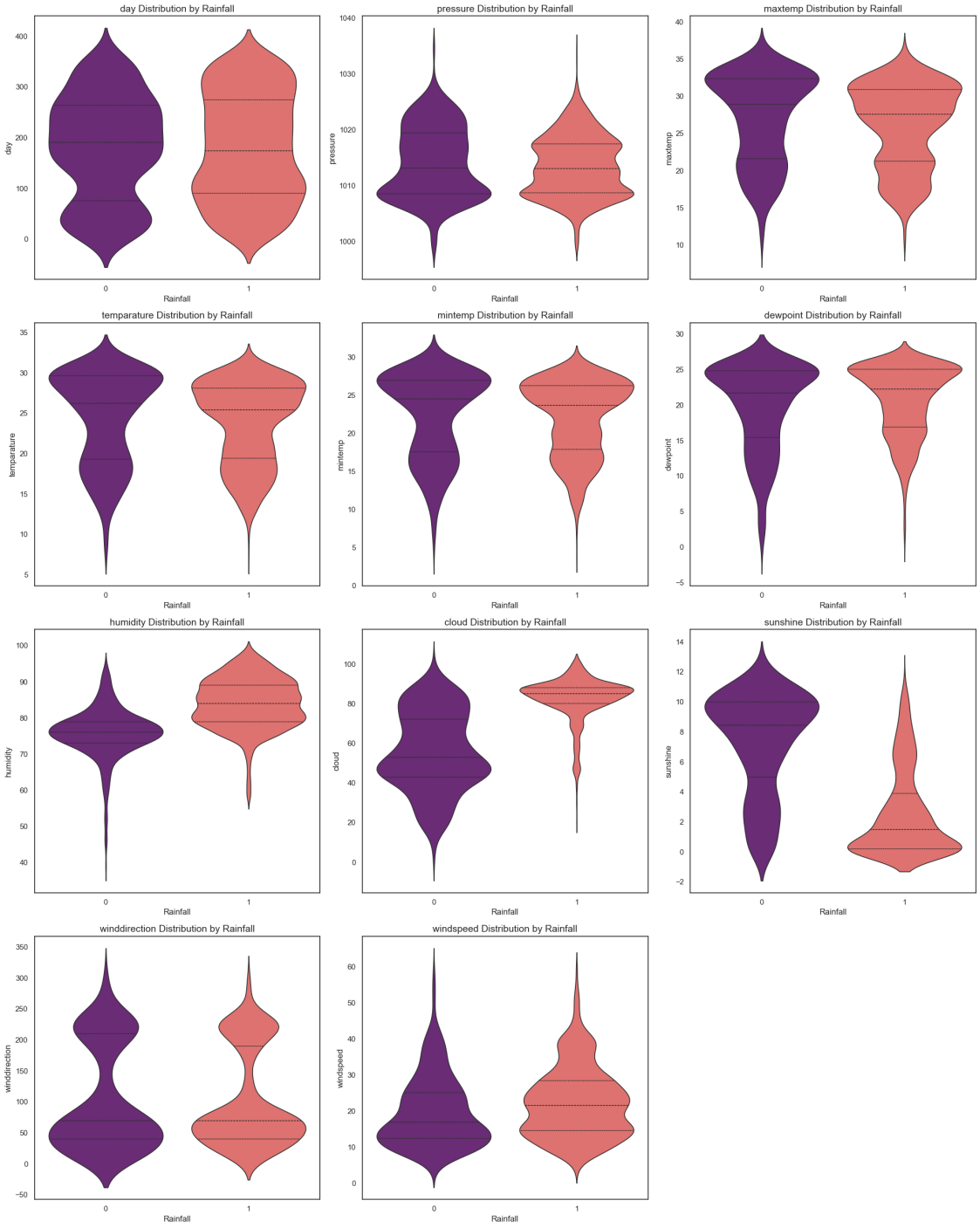
Distribution of sunshine
Train
Test
Density
sunshine


Distribution of winddirection
Train
Test
Density
winddirection

Distribution of windspeed
Train
Test
Density
windspeed

The histograms above illustrate the distribution of each numerical feature for both the training and testing datasets. By comparing the two, we can observe that the train and test sets exhibit similar distribution patterns, indicating that the data split is consistent and representative.

[15]:
```
features = [col for col in train_df.columns if col not in ['id', 'rainfall']]
plt.figure(figsize=(20, 25))
for i, col in enumerate(features, 1):
    plt.subplot(4, 3, i)
    sns.violinplot(data=train_df, x='rainfall', y=col, palette='magma',␣
 ↪inner="quartile", linewidth=1.2)
    plt.title(f'{col} Distribution by Rainfall', fontsize=14)
    plt.xlabel('Rainfall', fontsize=12)
    plt.ylabel(col, fontsize=12)

plt.tight_layout()
plt.show()
```

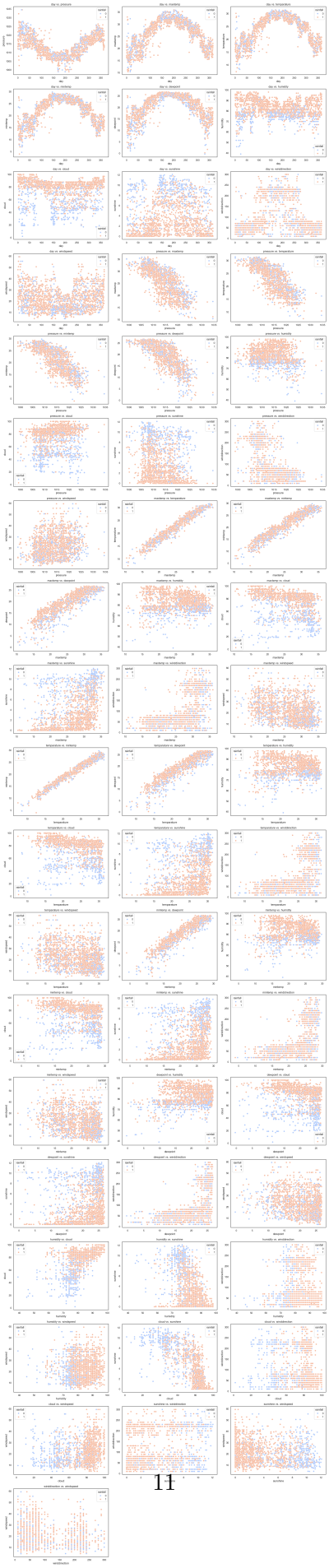


The violin plots above visualize the relationship between the target variable 'rainfall' and features such as 'humidity', 'cloud', and 'sunshine', providing deeper insight into their distributions.

- **Rainfall vs Humidity**: Class 0 (no rainfall) tends to cluster around 75% humidity, while class 1 (rainfall) clusters closer to 85%. This suggests that rainfall is more likely when humidity levels are higher.

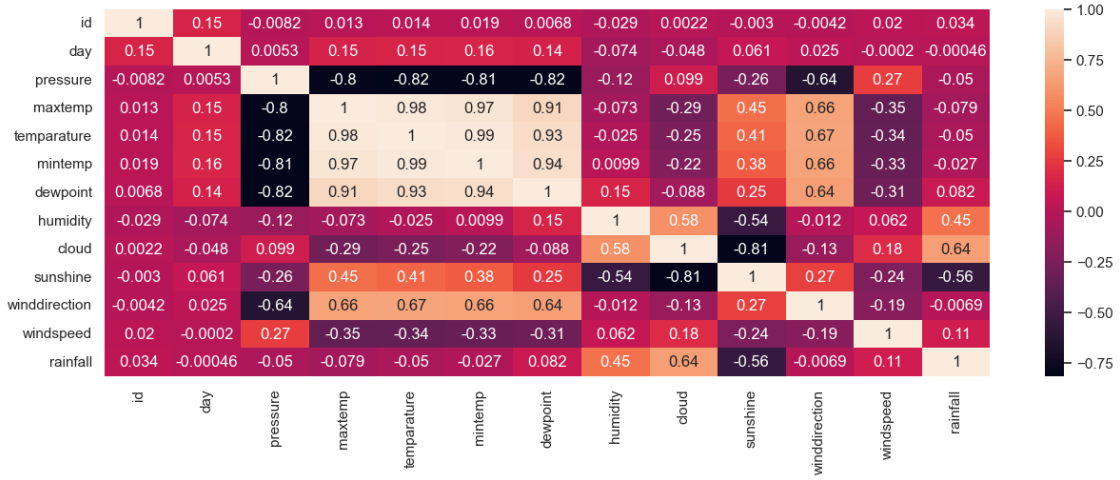
- **Rainfall vs Cloud Coverage**: Class 0 is centered around 55% cloud cover, whereas class 1 clusters around 85%, indicating that increased cloud coverage may be associated with a higher likelihood of rainfall.
- **Rainfall vs Sunshine**: Class 0 tends to have around 8 hours of sunshine, while class 1 is closer to just 1 hour. This implies that rainfall typically occurs on days with significantly less sunshine.

[16]:
```
# Define features and create feature pairs
features = ['day', 'pressure', 'maxtemp', 'temparature', 'mintemp',
            'dewpoint', 'humidity', 'cloud', 'sunshine', 'winddirection',␣
 ↪'windspeed']
pairs = list(combinations(features, 2))
n_cols = 3
n_rows = math.ceil(len(pairs) / n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 5 * n_rows))
axes = axes.flatten()

for i, (x, y) in enumerate(pairs):
    sns.scatterplot(data=train_df, x=x, y=y, hue='rainfall',␣
 ↪palette='coolwarm', ax=axes[i])
    axes[i].set_title(f'{x} vs. {y}')

for j in range(i + 1, len(axes)):
    axes[j].axis('off')

plt.tight_layout()
plt.show()
```


day vs. pressure
day vs. maxtemp
day vs. temperature
day vs. mintemp
day vs. dewpoint
day vs. humidity
day vs. cloud
day vs. sunshine
day vs. windbearing
day vs. windspeed
pressure vs. maxtemp
pressure vs. temperature
pressure vs. mintemp
pressure vs. dewpoint
pressure vs. humidity
pressure vs. cloud
pressure vs. sunshine
pressure vs. windbearing
pressure vs. windspeed
maxtemp vs. temperature
maxtemp vs. mintemp
maxtemp vs. dewpoint
maxtemp vs. humidity
maxtemp vs. cloud
maxtemp vs. sunshine
maxtemp vs. windbearing
maxtemp vs. windspeed
temperature vs. mintemp
temperature vs. dewpoint
temperature vs. humidity
temperature vs. cloud
temperature vs. sunshine
temperature vs. windbearing
temperature vs. windspeed
mintemp vs. dewpoint
mintemp vs. humidity
mintemp vs. cloud
mintemp vs. sunshine
mintemp vs. windbearing
mintemp vs. windspeed
dewpoint vs. humidity
dewpoint vs. cloud
dewpoint vs. sunshine
dewpoint vs. windbearing
dewpoint vs. windspeed
humidity vs. cloud
humidity vs. sunshine
humidity vs. windbearing
humidity vs. windspeed
cloud vs. sunshine
cloud vs. windbearing
cloud vs. windspeed
sunshine vs. windbearing
sunshine vs. windspeed
windbearing vs. windspeed
rainfall

[17]:
```
plt.figure(figsize = (15,5))
sns.heatmap(train_df.corr(), annot = True)
plt.show
```

[17]: <function matplotlib.pyplot.show(close=None, block=None)>



[18]:
```
target_column = train_df.columns[-1]
target_corr = train_df.corr()[target_column].sort_values(ascending=False)
print(target_corr.to_frame())
```

```
              rainfall
rainfall      1.000000
cloud         0.641191
humidity      0.454213
windspeed     0.111625
dewpoint      0.081965
id            0.033674
day          -0.000462
winddirection -0.006939
mintemp      -0.026841
temparature  -0.049660
pressure     -0.049886
maxtemp      -0.079304
sunshine     -0.555287
```

- **Positive Correlations**:
  - Some features show strong positive relationships (correlation values near +1), meaning they tend to increase together.
  - For example, **Cloud (0.641)** and **Humidity (0.454)** show moderately strong positive

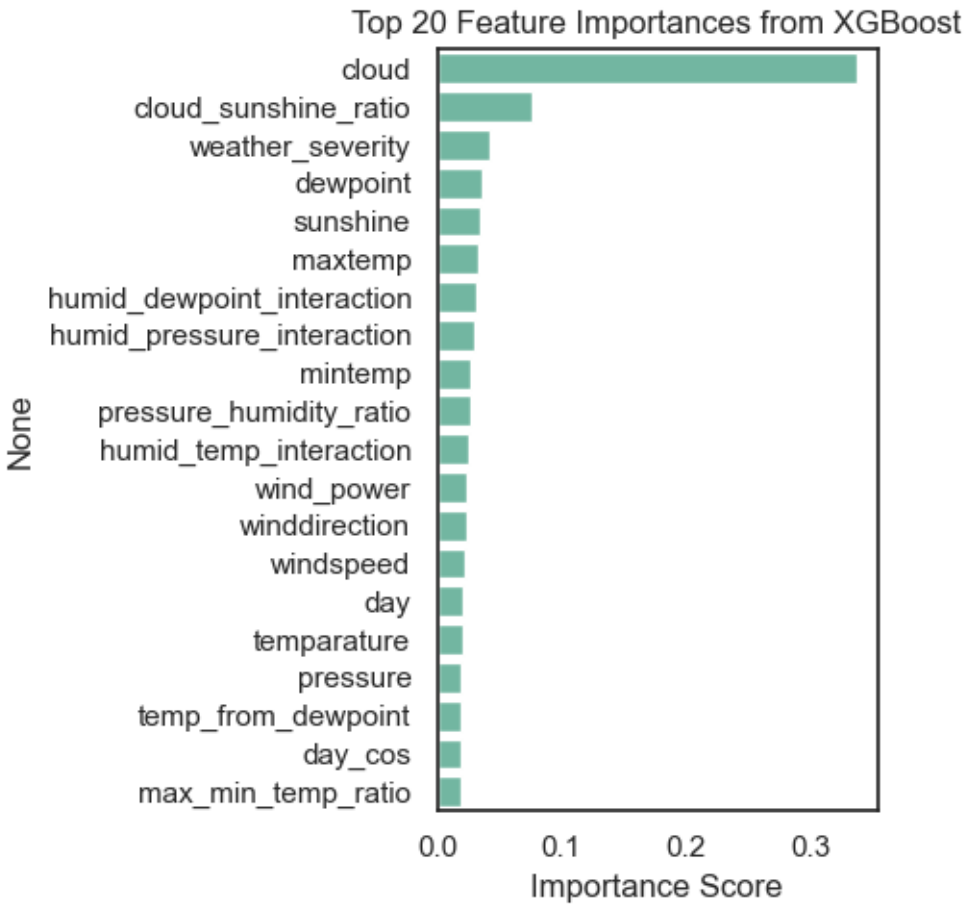
correlations with the target variable **Rainfall**, indicating that higher cloud coverage and humidity are generally associated with more rainfall.

- **Negative Correlations**:
  - Strong negative values (close to -1) suggest an inverse relationship, where one variable increases as the other decreases.
- **Low or No Correlation**:
  - Features like **Windspeed (0.112)** and **Dewpoint (0.082)** show very weak positive correlations with Rainfall, indicating minimal predictive value.
- **Multicollinearity Risk**:
  - High correlations between independent variables could indicate multicollinearity, which may affect the stability and interpretability of predictive models.
- **Cloud coverage** and **humidity** are the most influential factors in predicting rainfall.
- **Wind speed** and **dewpoint** show weak relationships and may contribute less to model performance.
- When building predictive models, consider prioritizing features with stronger correlations to the target.

1.0.3 Feature Engineering

[19]:
```
# create the feature engineering function
def create_features(df):
    df_new = df.copy()

    df_new['temp_range'] = df_new['maxtemp'] - df_new['mintemp']
    df_new['temp_ratio'] = df_new['temparature'] / (df_new['maxtemp'] + 1e-5)
    df_new['temp_from_dewpoint'] = df_new['temparature'] - df_new['dewpoint']
    df_new['max_min_temp_ratio'] = df_new['maxtemp'] / (df_new['mintemp'] +␣
↪1e-5)

    df_new['humid_temp_interaction'] = df_new['humidity'] *␣
↪df_new['temparature']
    df_new['humid_pressure_interaction'] = df_new['humidity'] *␣
↪df_new['pressure']
    df_new['humid_dewpoint_interaction'] = df_new['humidity'] *␣
↪df_new['dewpoint']

    df_new['cloud_sunshine_ratio'] = df_new['cloud'] / (df_new['sunshine'] + 1)
    df_new['cloud_coverage_rate'] = df_new['cloud'] / 100

    df_new['pressure_temp_ratio'] = df_new['pressure'] / (df_new['temparature']␣
↪+ 1e-5)
    df_new['pressure_humidity_ratio'] = df_new['pressure'] /␣
↪(df_new['humidity'] + 1e-5)

    df_new['weather_severity'] = (df_new['cloud'] * df_new['humidity']) /␣
↪(df_new['pressure'] * (df_new['sunshine'] + 1))
```

```
    df_new['temp_humidity_index'] = (df_new['temparature'] *␣
 ↪df_new['humidity']) / 100
    df_new['pressure_temp_humidity'] = (df_new['pressure'] *␣
 ↪df_new['temparature']) / (df_new['humidity'] + 1e-5)

    df_new['day_sin'] = np.sin(2 * np.pi * df_new['day'] / 365)
    df_new['day_cos'] = np.cos(2 * np.pi * df_new['day'] / 365)

    # Wind feature
    df_new['wind_power'] = df_new['windspeed'] * np.abs(np.cos(np.
 ↪radians(df_new['winddirection'])))

    return df_new

# Apply feature engineering
train_fe = create_features(train_df)

# Prepare data for feature selection
X = train_fe.drop(columns=['id', 'rainfall'])
y = train_fe['rainfall']
X_train, X_val, y_train, y_val = train_test_split(X, y, stratify=y, test_size=0.
 ↪2, random_state=42)

# Train XGBoost model
xgb_model = XGBClassifier(n_estimators=200, max_depth=4, learning_rate=0.05,␣
 ↪use_label_encoder=False, eval_metric='logloss')
xgb_model.fit(X_train, y_train)

xgb_importance = pd.Series(xgb_model.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)

plt.figure(figsize=(5, 5))
sns.barplot(x=xgb_importance[:20], y=xgb_importance.index[:20])
plt.title("Top 20 Feature Importances from XGBoost")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()
```



1.0.4 Model Training and Evaluation Using Selected Features

[20]:
```
# Top 25 features selected previously
top_xgb_features = ['cloud', 'weather_severity', 'humid_temp_interaction',␣
 ↪'cloud_sunshine_ratio', 'dewpoint',
                     'humid_pressure_interaction', 'pressure_humidity_ratio',␣
 ↪'maxtemp', 'sunshine', 'mintemp',
                     'humid_dewpoint_interaction', 'temp_from_dewpoint',␣
 ↪'day_sin', 'pressure_temp_humidity',
                     'wind_power', 'day_cos', 'temp_range', 'temparature',␣
 ↪'pressure', 'windspeed', 'winddirection',
                     'temp_ratio', 'day', 'max_min_temp_ratio',␣
 ↪'pressure_temp_ratio']

# Prepare inputs
X = train_fe[top_xgb_features]
y = train_fe['rainfall']
X_scaled = MinMaxScaler().fit_transform(X)
```

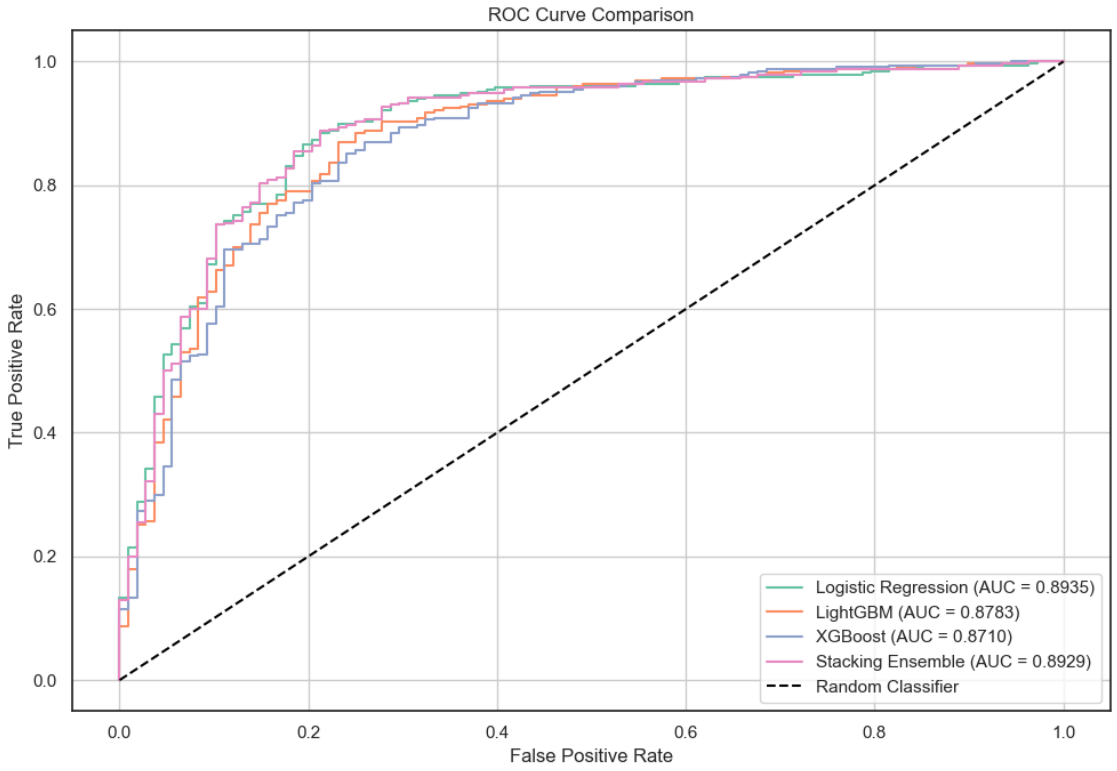
```
# Class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y), y=y)
weights_dict = dict(zip(np.unique(y), class_weights))

# Initialize models
logreg_model = LogisticRegression(max_iter=1000, class_weight=weights_dict)
lgb_model = LGBMClassifier(n_estimators=100, class_weight='balanced',␣
 ↪random_state=42, verbosity=-1)
xgb_model = XGBClassifier(n_estimators=100, use_label_encoder=False,␣
 ↪eval_metric='logloss', scale_pos_weight=class_weights[1]/class_weights[0])
stack_model = StackingClassifier(
    estimators=[('lr', Pipeline([('scaler', MinMaxScaler()), ('logreg',␣
 ↪LogisticRegression(max_iter=1000, class_weight=weights_dict))])),
        ('lgb', lgb_model),('xgb',␣
 ↪xgb_model)],final_estimator=LogisticRegression())

# Cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = {'Logistic Regression': [], 'LightGBM': [], 'XGBoost': [], 'CatBoost':
 ↪ [], 'Stacking Ensemble': []}

for train_idx, val_idx in skf.split(X, y):
    X_tr, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_tr, y_val = y.iloc[train_idx], y.iloc[val_idx]

    logreg_model.fit(MinMaxScaler().fit_transform(X_tr), y_tr)
    lgb_model.fit(X_tr, y_tr)
    xgb_model.fit(X_tr, y_tr)
    stack_model.fit(X_tr, y_tr)

    results['Logistic Regression'].append(roc_auc_score(y_val, logreg_model.
 ↪predict_proba(MinMaxScaler().fit_transform(X_val))[:, 1]))
    results['LightGBM'].append(roc_auc_score(y_val, lgb_model.
 ↪predict_proba(X_val)[:, 1]))
    results['XGBoost'].append(roc_auc_score(y_val, xgb_model.
 ↪predict_proba(X_val)[:, 1]))
    results['Stacking Ensemble'].append(roc_auc_score(y_val, stack_model.
 ↪predict_proba(X_val)[:, 1]))

# Calculate mean AUC for each model
mean_auc_results = {model: np.mean(scores) for model, scores in results.items()}
mean_auc_results
```

[20]:
```
{'Logistic Regression': np.float64(0.8922166105499437),
 'LightGBM': np.float64(0.8716442199775534),
```

```
 'XGBoost': np.float64(0.8661223344556678),
 'CatBoost': np.float64(nan),
 'Stacking Ensemble': np.float64(0.8905667789001122)}
```

1.0.5 ROC Curve Comparison of Classification Models

[21]:
```
plt.figure(figsize=(10, 7))

# Plot ROC for each model
for name, model in [('Logistic Regression', logreg_model),
                    ('LightGBM', lgb_model),
                    ('XGBoost', xgb_model),
                    ('Stacking Ensemble', stack_model)]:

    # Predict probabilities
    if name == 'Logistic Regression':
        probas = model.predict_proba(MinMaxScaler().fit_transform(X_val))[:, 1]
    else:
        probas = model.predict_proba(X_val)[:, 1]

    fpr, tpr, _ = roc_curve(y_val, probas)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.4f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```



The ROC curve comparison illustrates the performance of four classification models: Logistic Regression, LightGBM, XGBoost, and a Stacking Ensemble. Among these, **Logistic Regression achieved the highest AUC score of 0.8935**, indicating that the selected features have a strong linear relationship with the target variable. The **Stacking Ensemble closely followed with an AUC of 0.8929**, but did not significantly outperform Logistic Regression, likely due to limited diversity among base models.

Both **LightGBM (AUC = 0.8783)** and **XGBoost (AUC = 0.8710)** also performed well, but slightly lagged behind, suggesting that the additional complexity of these gradient boosting models did not offer substantial gains over the simpler model.

All models significantly outperformed the random classifier baseline, demonstrating strong discriminatory power. Overall, the results highlight the effectiveness of the top 25 selected features and suggest that **Logistic Regression offers a highly effective and interpretable solution for this classification task**.

1.0.6 Final Model Prediction and Submission Preparation

[22]:
```
# apply feature engineering to the test set
test_fe = create_features(test_df)

# Select only the top features
X_train_final = train_fe[top_xgb_features]
y_train_final = train_fe['rainfall']
```

```
X_test_final = test_fe[top_xgb_features]

# Normalize features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_final)
X_test_scaled = scaler.transform(X_test_final)

# Final Logistic Regression model
final_logreg = LogisticRegression(max_iter=1000, class_weight='balanced')
final_logreg.fit(X_train_scaled, y_train_final)

# Predict probabilities on test set
rainfall_probs = final_logreg.predict_proba(X_test_scaled)[:, 1]

# Prepare submission DataFrame
submission_df = pd.DataFrame({
    'id': test_df['id'],
    'rainfall_probability': rainfall_probs
})

# Save to CSV
submission_path = '/Users/nav/Documents/Jupyter Notebooks /
 ↪logreg_rainfall_submission.csv'
submission_df.to_csv(submission_path, index=False)
```

1.0.7 Hyperparameter Optimization and Final Submission Using Logistic Regression

[23]:
```
# Scale features
X = train_fe[top_xgb_features]
y = train_fe['rainfall']
X_scaled = MinMaxScaler().fit_transform(X)

def objective(trial):
    C = trial.suggest_loguniform('C', 1e-3, 10)
    penalty = trial.suggest_categorical('penalty', ['l2'])
    solver = trial.suggest_categorical('solver', ['lbfgs', 'liblinear', 'saga'])

    model = LogisticRegression(
        C=C,
        penalty=penalty,
        solver=solver,
        class_weight='balanced',
        max_iter=1000
    )

    return cross_val_score(model, X_scaled, y, cv=5, scoring='roc_auc').mean()
```

```
study = optuna.create_study(direction='maximize', sampler=TPESampler(seed=42))
study.optimize(objective, n_trials=30)

print("Best AUC:", study.best_value)
print("Best Params:", study.best_params)
```

```
Best AUC: 0.8922109988776654
Best Params: {'C': 1.4773204665513213, 'penalty': 'l2', 'solver': 'lbfgs'}
```

[24]:
```
# Retrain the final optimized logistic regression model
optimized_logreg = LogisticRegression(
    C=1.4773204665513213,
    penalty='l2',
    solver='lbfgs',
    class_weight='balanced',
    max_iter=1000
)

# Fit on full training set (already filled NaNs and scaled)
X_train_final = train_fe[top_xgb_features]
X_test_final = test_fe[top_xgb_features]
y_train_final = train_fe['rainfall']

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_final)
X_test_scaled = scaler.transform(X_test_final)

# Fit and predict
optimized_logreg.fit(X_train_scaled, y_train_final)
rainfall_probs_optimized = optimized_logreg.predict_proba(X_test_scaled)[:, 1]

# Create final submission DataFrame
optimized_submission_df = pd.DataFrame({
    'id': test_df['id'],
    'rainfall_probability': rainfall_probs_optimized})

optimized_submission_path = '/Users/nav/Documents/Jupyter Notebooks /
 ↪logreg_rainfall_submission2.csv'
optimized_submission_df.to_csv(optimized_submission_path, index=False)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: