Assessment Report

on

"Rainfall Prediction"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AI)

By

Name of Team Members:

1. Adyanshi Singh - 202401100300023
2. Achin Sahu - 202401100300011
3. Aditya Singh - 202401100300021
4. Abhishek Deewakar - 202401100300006

Section: A

Under the supervision of

"Bikki Gupta"

KIET Group of Institutions, Ghaziabad

May, 2025

1. Introduction

Rainfall prediction is a critical task in meteorology that aims to forecast the amount and occurrence of rain in a specific region over a future time period. Accurate rainfall forecasting helps in numerous sectors such as:-

- **Agriculture** : - (crop planning and irrigation)
- **Disaster management:-** (flood prediction and preparedness)
- **Urban planning** :- (drainage systems, water storage)
- **Transportation** :- (reducing risks due to adverse weather)

Traditionally, rainfall forecasting relied on physical models and numerical weather simulations, which are computationally intensive and depend heavily on atmospheric physics. However, with the growing availability of weather data and advances in **machine learning (ML)** and **artificial intelligence (AI)**, data-driven models have become increasingly popular.

These ML models can learn patterns and relationships from historical weather data, enabling faster and often more localized predictions. The accuracy of such models continues to improve with better data, more powerful algorithms, and increasing computational resources.

2. Problem Statement

Design and implement a machine learning model that can accurately forecast rainfall using historical weather data. The model should take key atmospheric parameters as input and predict the occurrence and/or amount of rainfall in a given region.

3. Objectives

To develop a machine learning model that can predict **whether it will rain** (classification) or **how much it will rain** (regression) based on various meteorological features such as:

- Temperature
- Humidity
- Atmospheric Pressure
- Wind Speed and Direction
- Cloud Cover
- Historical Rainfall Data

4. Methodology

- **Data Collection**: The user uploads a CSV file containing the dataset.

- **Model Building**:

□ Feed training data into the selected algorithm.

□ Tune hyperparameters using techniques like Grid Search or Random Search.

□ Use **cross-validation** to ensure the model generalizes well.

- **Model Evaluation**:

Evaluate the model's performance using appropriate metrics:

For Classification:

- Accuracy
- Precision, Recall, F1 Score
- Confusion Matrix

- ROC-AUC Curve

---

5. Data Preprocessing

 Handling Missing Values:- Fill or drop rows/columns with missing data.

 Encoding Categorical Features:- Convert text labels to numbers (e.g., wind direction).

 Feature Engineering: - Derive new features such as "day of year", "season", "is_weekend".

 Normalization/Scaling:- Standardize features to improve model performance.

 Train-Test Split:- Divide data (e.g., 80% for training, 20% for testing).

---

6. Model Implementation

Below is a simple **model implementation** using **Python** and the **Random Forest** algorithm for rainfall prediction. We'll demonstrate a **classification** problem: predicting **whether it will rain or not** based on weather data.

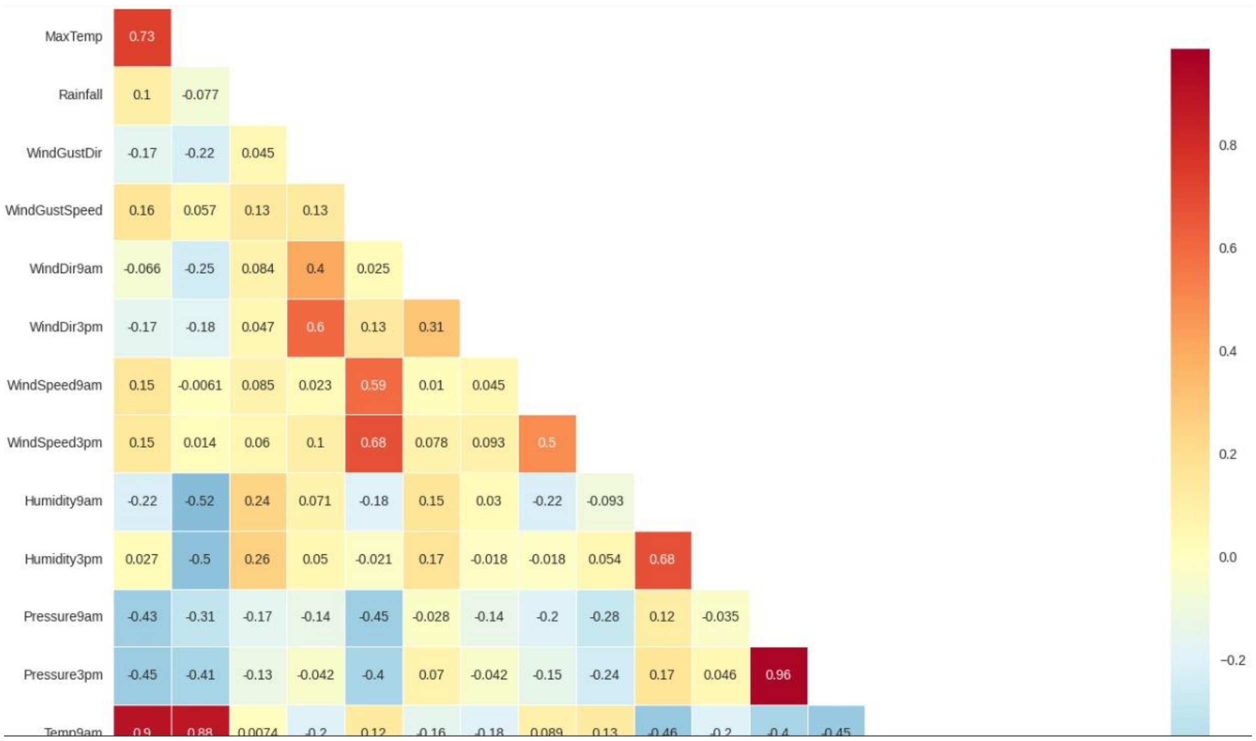
---

7. Evaluation Metrics

The following metrics are used to evaluate the model:

- **Accuracy:** Measures the proportion of correct predictions.
- **Precision:** True Positives / (True Positives + False Positives)
- **Recall:** True Positives / (True Positives + False Negatives)
- **F1-Score**: Harmonic mean of precision and recall.
- **Confusion Matrix:** Visualized using a heatmap.

Confusion Matrix:



---

9. Conclusion

Rainfall prediction using machine learning helps improve the accuracy and timeliness of weather forecasts. By analyzing historical weather data, models like Random Forest can effectively predict rain occurrence or amount, aiding agriculture, disaster management, and water planning. Despite some challenges, ML-based prediction offers a promising tool for better decision-making and risk reduction.

---

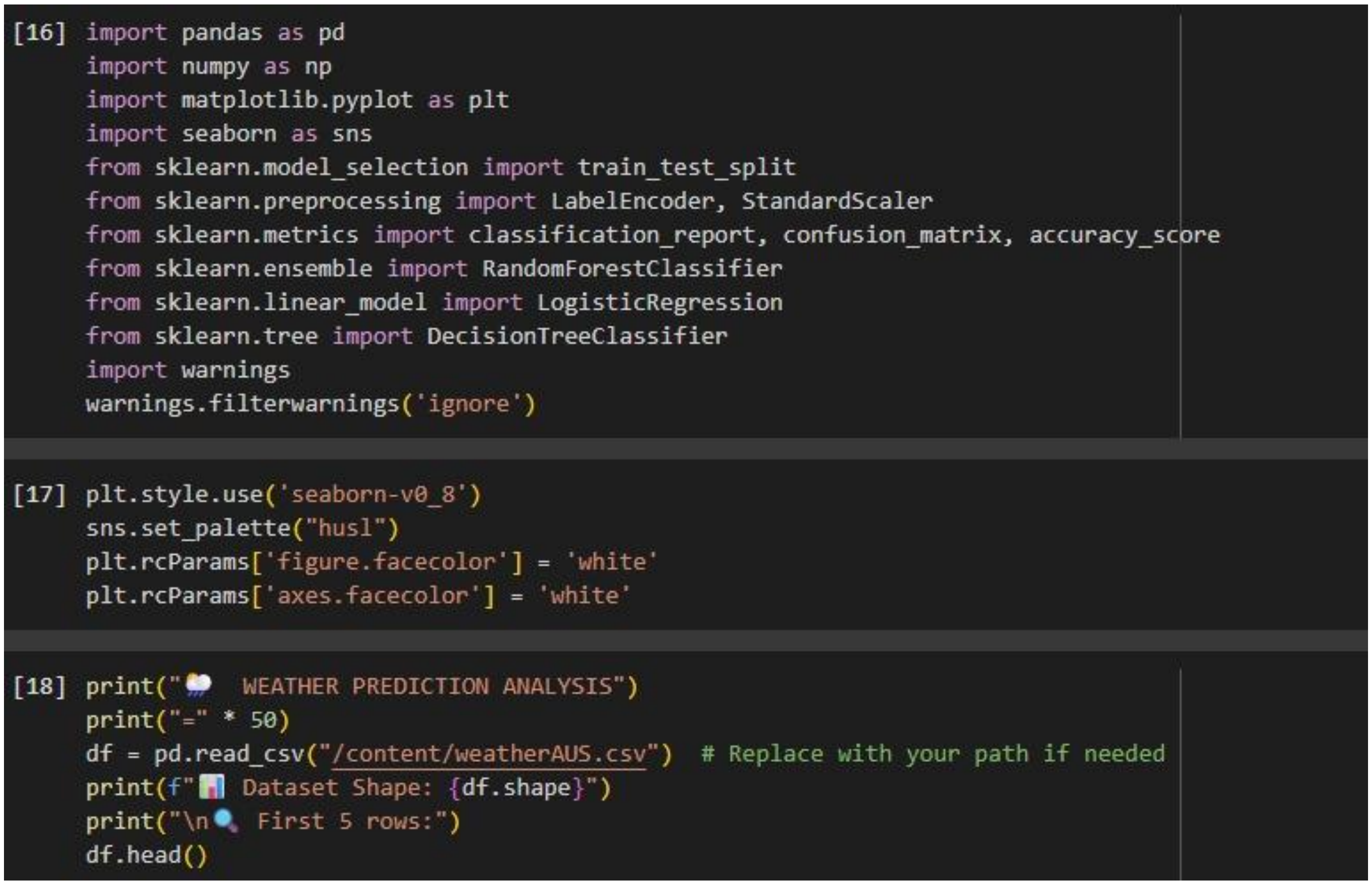
10. References

- Kaggle Datasets. (n.d.). Weather and Rainfall Data. Retrieved from https://www.kaggle.com/
- pandas documentation

- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer
- Research articles on credit risk prediction

---

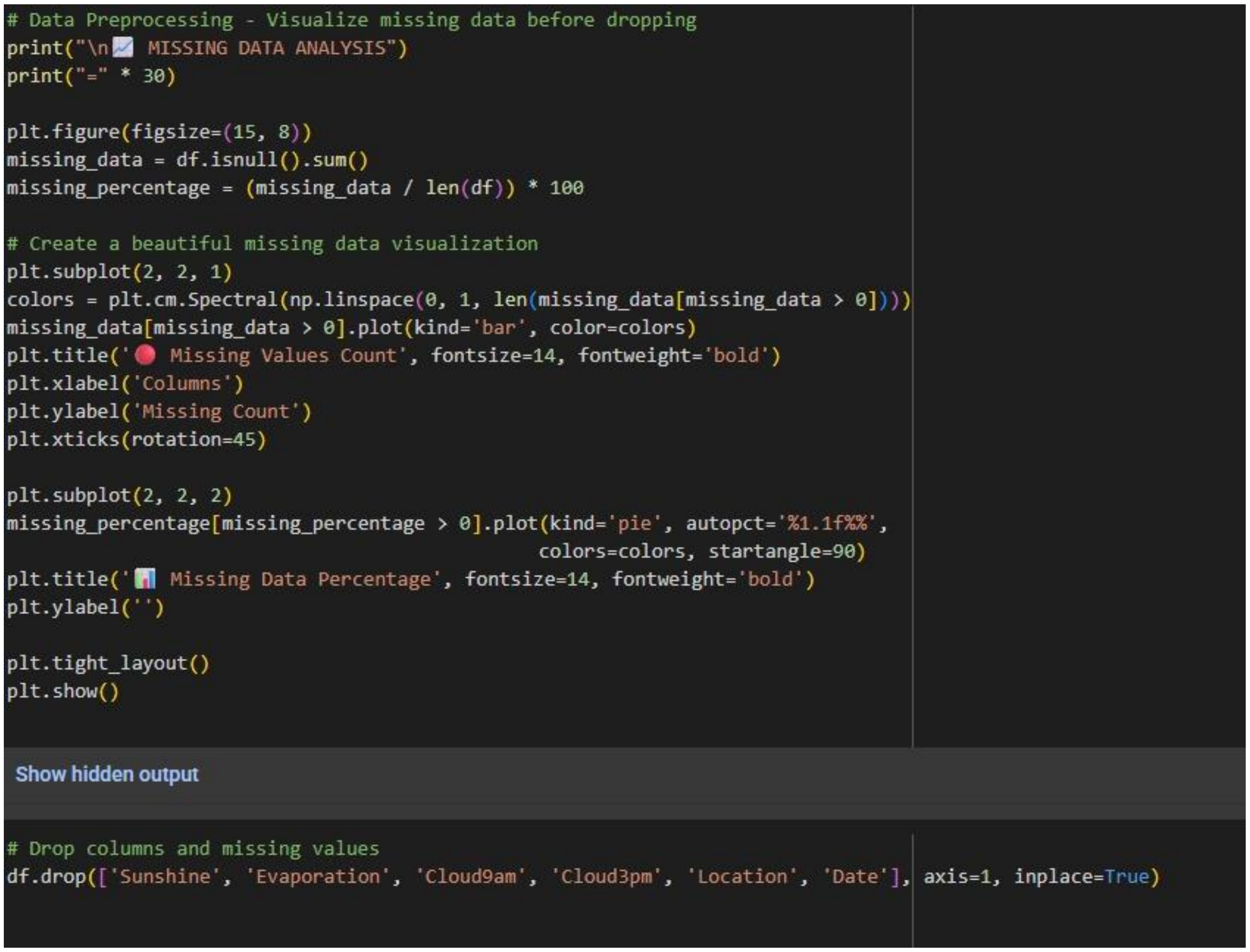
11. Code



```
[16] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder, StandardScaler
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     import warnings
     warnings.filterwarnings('ignore')
```

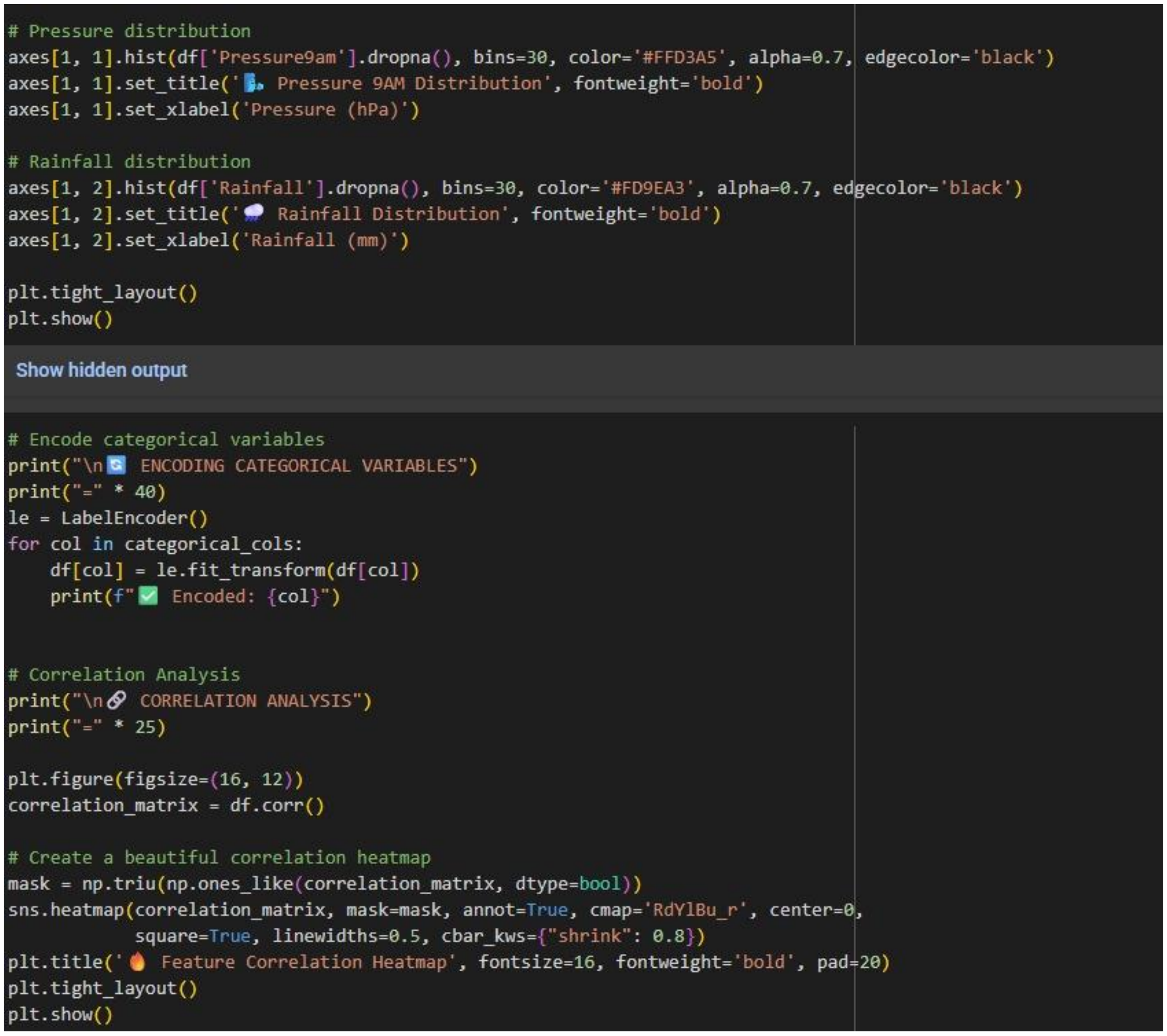
```
[17] plt.style.use('seaborn-v0_8')
     sns.set_palette("husl")
     plt.rcParams['figure.facecolor'] = 'white'
     plt.rcParams['axes.facecolor'] = 'white'
```

```
[18] print("🌧️  WEATHER PREDICTION ANALYSIS")
     print("=" * 50)
     df = pd.read_csv("/content/weatherAUS.csv")  # Replace with your path if needed
     print(f"📊 Dataset Shape: {df.shape}")
     print("\n🔍 First 5 rows:")
     df.head()
```



```
# Data Preprocessing - Visualize missing data before dropping
print("\n📈 MISSING DATA ANALYSIS")
print("=" * 30)

plt.figure(figsize=(15, 8))
missing_data = df.isnull().sum()
missing_percentage = (missing_data / len(df)) * 100

# Create a beautiful missing data visualization
plt.subplot(2, 2, 1)
colors = plt.cm.Spectral(np.linspace(0, 1, len(missing_data[missing_data > 0])))
missing_data[missing_data > 0].plot(kind='bar', color=colors)
plt.title('🔴 Missing Values Count', fontsize=14, fontweight='bold')
plt.xlabel('Columns')
plt.ylabel('Missing Count')
plt.xticks(rotation=45)

plt.subplot(2, 2, 2)
missing_percentage[missing_percentage > 0].plot(kind='pie', autopct='%1.1f%%',
                                               colors=colors, startangle=90)
plt.title('📊 Missing Data Percentage', fontsize=14, fontweight='bold')
plt.ylabel('')

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

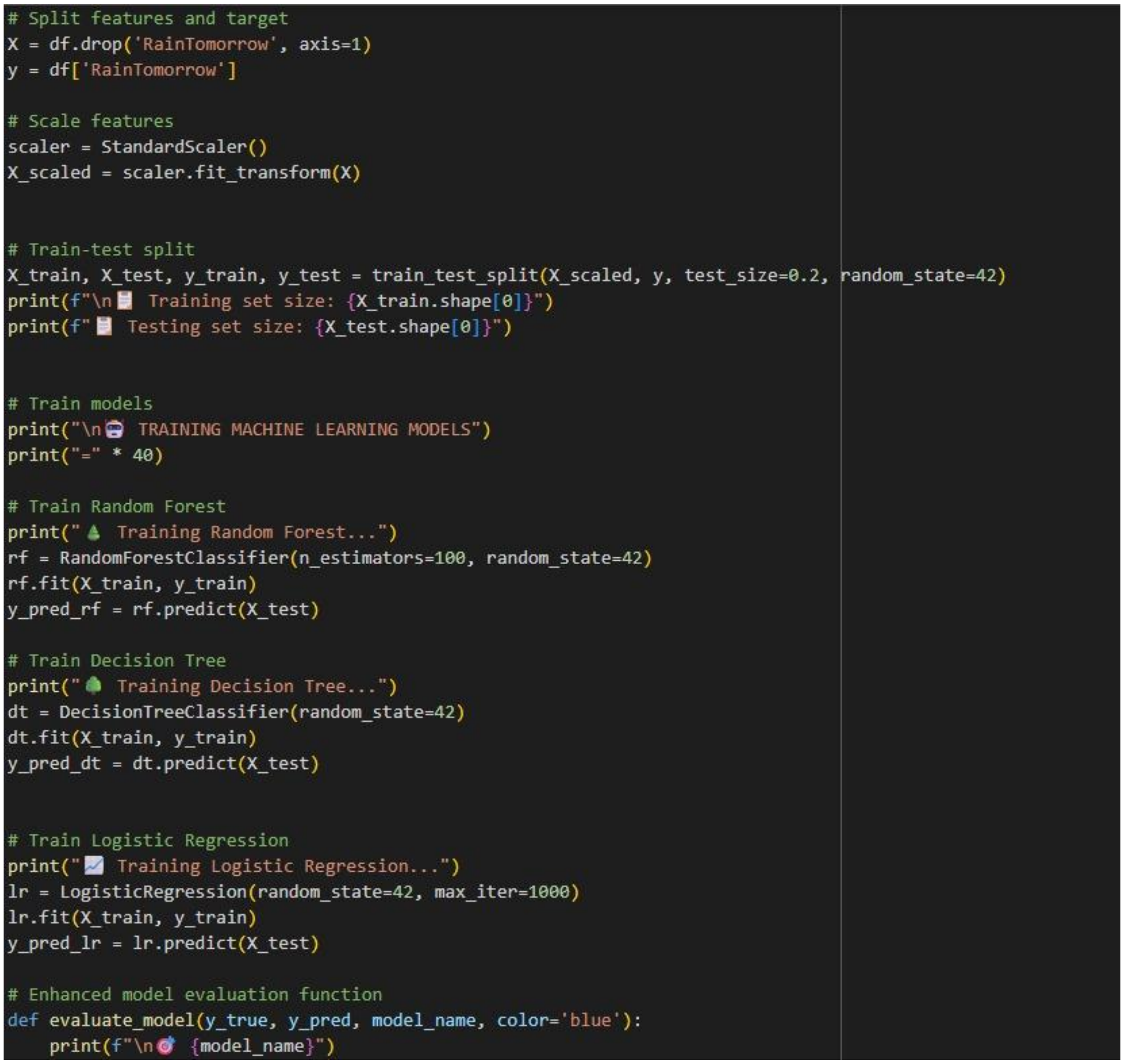
```
# Drop columns and missing values
df.drop(['Sunshine', 'Evaporation', 'Cloud9am', 'Cloud3pm', 'Location', 'Date'], axis=1, inplace=True)
```



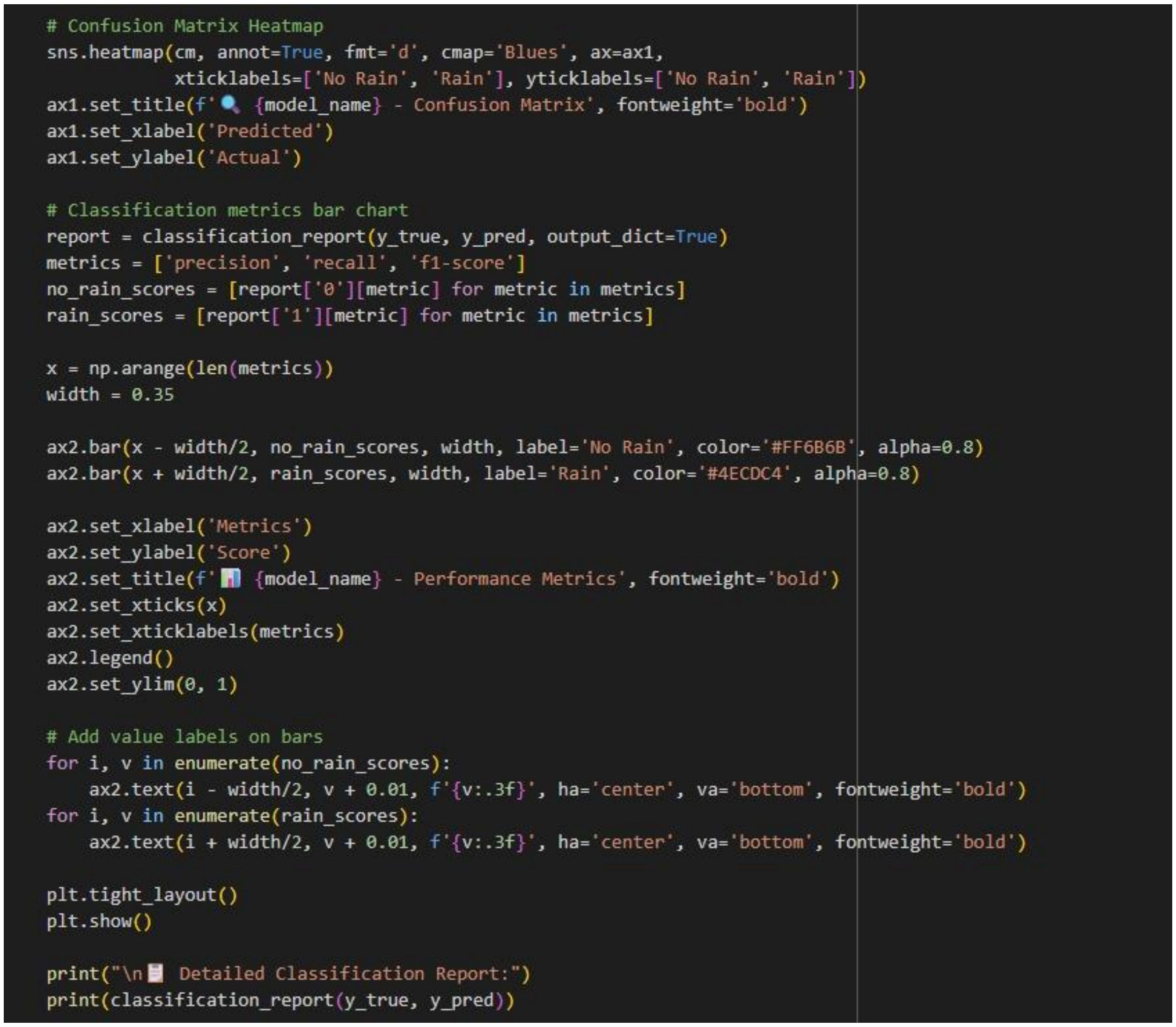
```python
# Pressure distribution
axes[1, 1].hist(df['Pressure9am'].dropna(), bins=30, color='#FFD3A5', alpha=0.7, edgecolor='black')
axes[1, 1].set_title('🌡️ Pressure 9AM Distribution', fontweight='bold')
axes[1, 1].set_xlabel('Pressure (hPa)')

# Rainfall distribution
axes[1, 2].hist(df['Rainfall'].dropna(), bins=30, color='#FD9EA3', alpha=0.7, edgecolor='black')
axes[1, 2].set_title('🌧️ Rainfall Distribution', fontweight='bold')
axes[1, 2].set_xlabel('Rainfall (mm)')

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

Show hidden output

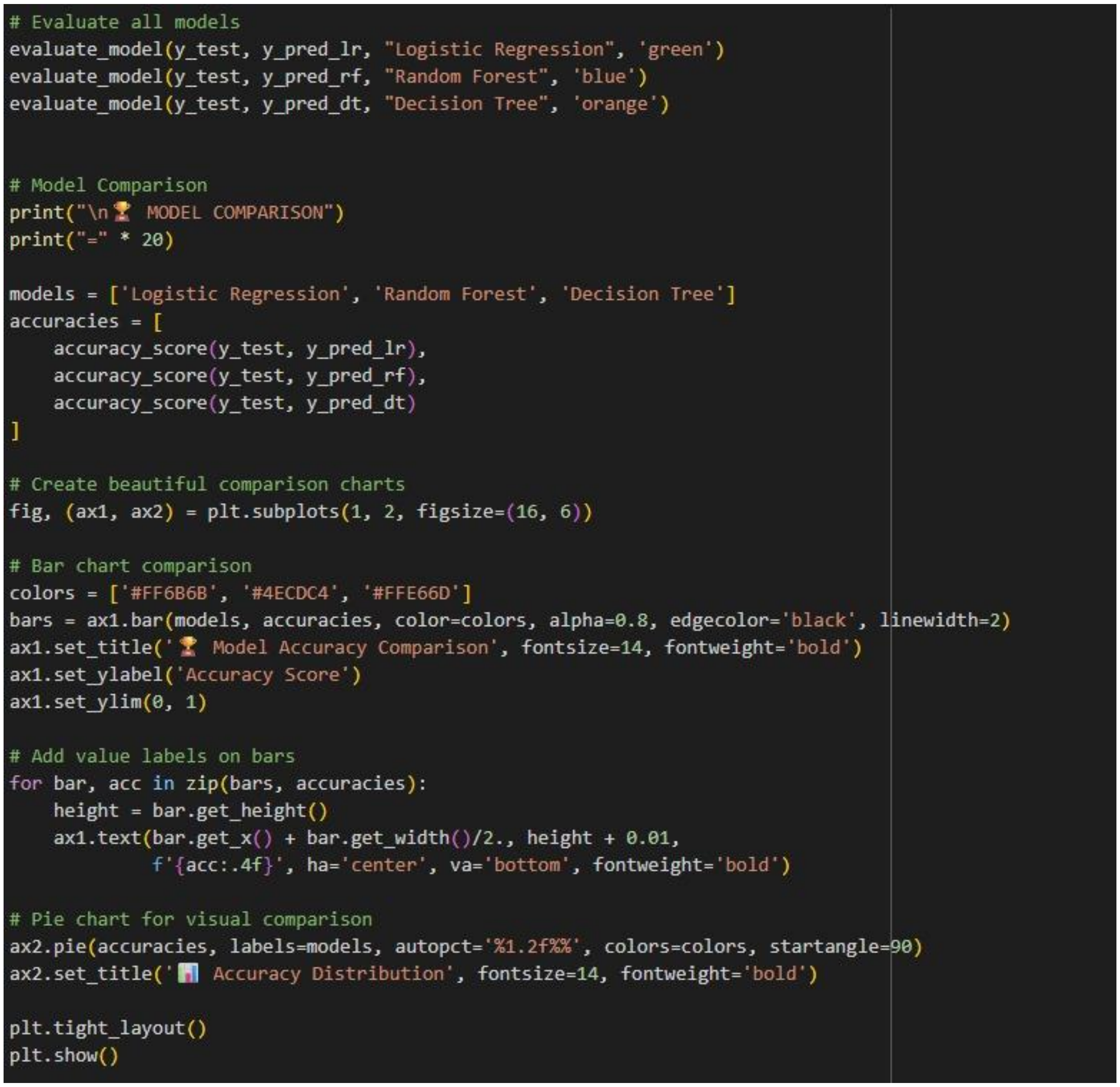
```python
# Encode categorical variables
print("\n🔄 ENCODING CATEGORICAL VARIABLES")
print("=" * 40)
le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])
    print(f"✅ Encoded: {col}")


# Correlation Analysis
print("\n🔗 CORRELATION ANALYSIS")
print("=" * 25)

plt.figure(figsize=(16, 12))
correlation_matrix = df.corr()

# Create a beautiful correlation heatmap
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='RdYlBu_r', center=0,
            square=True, linewidths=0.5, cbar_kws={"shrink": 0.8})
plt.title('🔥 Feature Correlation Heatmap', fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()
```



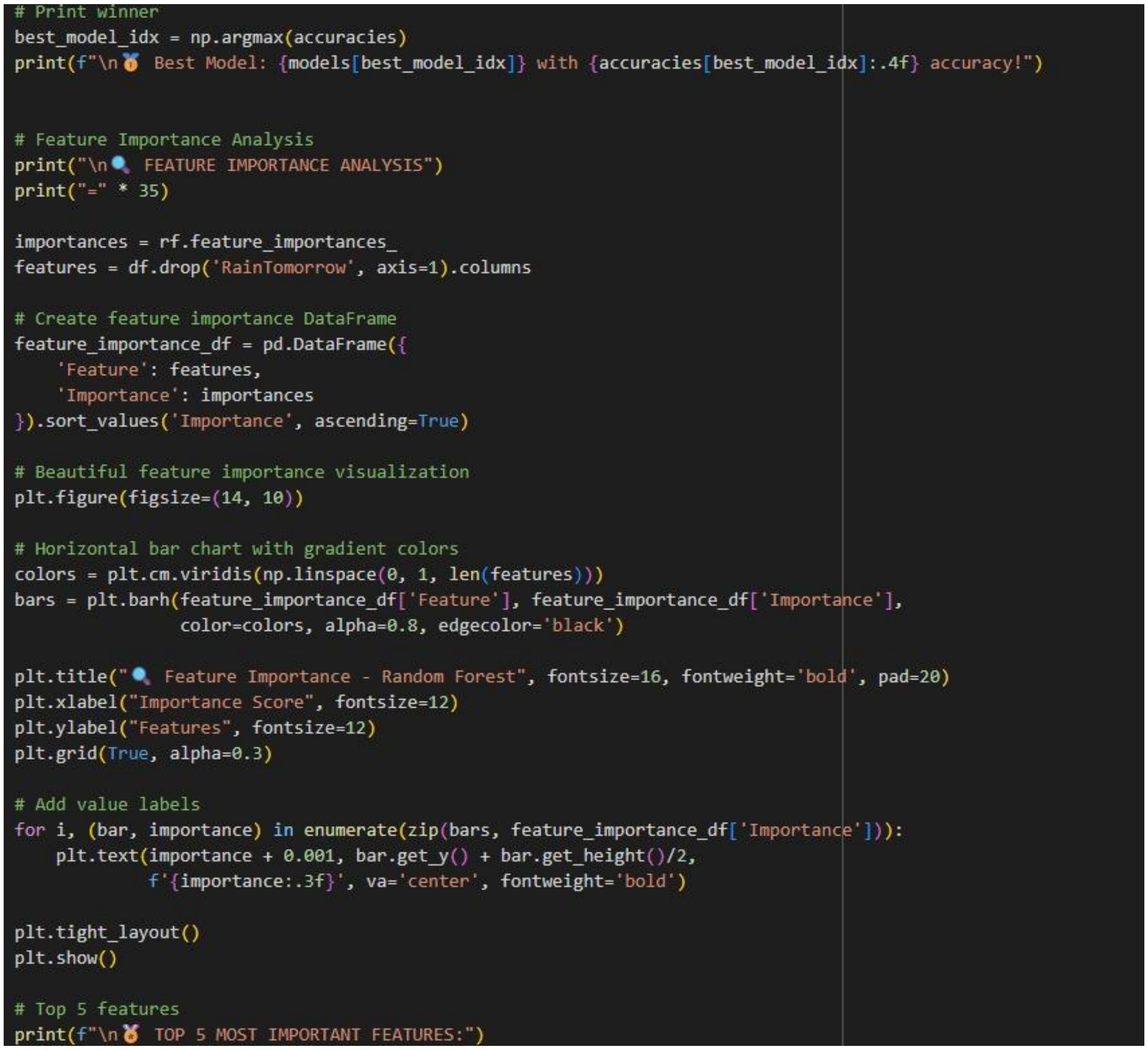
```python
# Split features and target
X = df.drop('RainTomorrow', axis=1)
y = df['RainTomorrow']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
print(f"\n📋 Training set size: {X_train.shape[0]}")
print(f"📋 Testing set size: {X_test.shape[0]}")


# Train models
print("\n🤖 TRAINING MACHINE LEARNING MODELS")
print("=" * 40)

# Train Random Forest
print("🌲 Training Random Forest...")
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Train Decision Tree
print("🌳 Training Decision Tree...")
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)


# Train Logistic Regression
print("📈 Training Logistic Regression...")
lr = LogisticRegression(random_state=42, max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# Enhanced model evaluation function
def evaluate_model(y_true, y_pred, model_name, color='blue'):
    print(f"\n🎯 {model_name}")
```

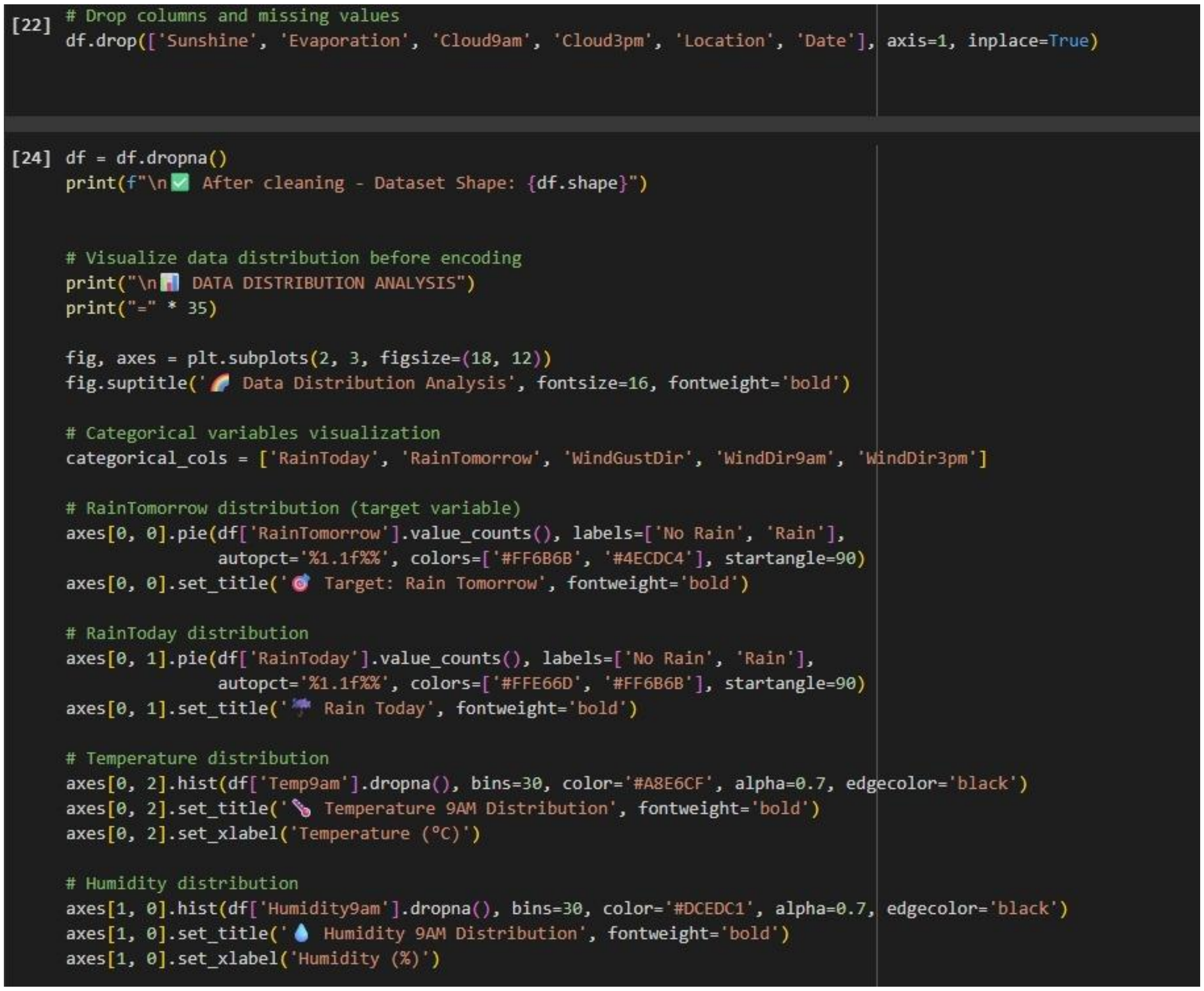


```python
# Confusion Matrix Heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax1,
            xticklabels=['No Rain', 'Rain'], yticklabels=['No Rain', 'Rain'])
ax1.set_title(f'🔍 {model_name} - Confusion Matrix', fontweight='bold')
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')

# Classification metrics bar chart
report = classification_report(y_true, y_pred, output_dict=True)
metrics = ['precision', 'recall', 'f1-score']
no_rain_scores = [report['0'][metric] for metric in metrics]
rain_scores = [report['1'][metric] for metric in metrics]

x = np.arange(len(metrics))
width = 0.35

ax2.bar(x - width/2, no_rain_scores, width, label='No Rain', color='#FF6B6B', alpha=0.8)
ax2.bar(x + width/2, rain_scores, width, label='Rain', color='#4ECDC4', alpha=0.8)

ax2.set_xlabel('Metrics')
ax2.set_ylabel('Score')
ax2.set_title(f'📊 {model_name} - Performance Metrics', fontweight='bold')
ax2.set_xticks(x)
ax2.set_xticklabels(metrics)
ax2.legend()
ax2.set_ylim(0, 1)

# Add value labels on bars
for i, v in enumerate(no_rain_scores):
    ax2.text(i - width/2, v + 0.01, f'{v:.3f}', ha='center', va='bottom', fontweight='bold')
for i, v in enumerate(rain_scores):
    ax2.text(i + width/2, v + 0.01, f'{v:.3f}', ha='center', va='bottom', fontweight='bold')

plt.tight_layout()
plt.show()

print("\n📋 Detailed Classification Report:")
print(classification_report(y_true, y_pred))
```



```python
# Evaluate all models
evaluate_model(y_test, y_pred_lr, "Logistic Regression", 'green')
evaluate_model(y_test, y_pred_rf, "Random Forest", 'blue')
evaluate_model(y_test, y_pred_dt, "Decision Tree", 'orange')


# Model Comparison
print("\n🏆 MODEL COMPARISON")
print("=" * 20)

models = ['Logistic Regression', 'Random Forest', 'Decision Tree']
accuracies = [
    accuracy_score(y_test, y_pred_lr),
    accuracy_score(y_test, y_pred_rf),
    accuracy_score(y_test, y_pred_dt)
]

# Create beautiful comparison charts
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Bar chart comparison
colors = ['#FF6B6B', '#4ECDC4', '#FFE66D']
bars = ax1.bar(models, accuracies, color=colors, alpha=0.8, edgecolor='black', linewidth=2)
ax1.set_title('🏆 Model Accuracy Comparison', fontsize=14, fontweight='bold')
ax1.set_ylabel('Accuracy Score')
ax1.set_ylim(0, 1)

# Add value labels on bars
for bar, acc in zip(bars, accuracies):
    height = bar.get_height()
    ax1.text(bar.get_x() + bar.get_width()/2., height + 0.01,
             f'{acc:.4f}', ha='center', va='bottom', fontweight='bold')

# Pie chart for visual comparison
ax2.pie(accuracies, labels=models, autopct='%1.2f%%', colors=colors, startangle=90)
ax2.set_title('📊 Accuracy Distribution', fontsize=14, fontweight='bold')

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


```python
# Print winner
best_model_idx = np.argmax(accuracies)
print(f"\n🥉 Best Model: {models[best_model_idx]} with {accuracies[best_model_idx]:.4f} accuracy!")


# Feature Importance Analysis
print("\n🔍 FEATURE IMPORTANCE ANALYSIS")
print("=" * 35)

importances = rf.feature_importances_
features = df.drop('RainTomorrow', axis=1).columns

# Create feature importance DataFrame
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values('Importance', ascending=True)

# Beautiful feature importance visualization
plt.figure(figsize=(14, 10))

# Horizontal bar chart with gradient colors
colors = plt.cm.viridis(np.linspace(0, 1, len(features)))
bars = plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
                color=colors, alpha=0.8, edgecolor='black')

plt.title("🔍 Feature Importance - Random Forest", fontsize=16, fontweight='bold', pad=20)
plt.xlabel("Importance Score", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.grid(True, alpha=0.3)

# Add value labels
for i, (bar, importance) in enumerate(zip(bars, feature_importance_df['Importance'])):
    plt.text(importance + 0.001, bar.get_y() + bar.get_height()/2,
             f'{importance:.3f}', va='center', fontweight='bold')

plt.tight_layout()
plt.show()

# Top 5 features
print(f"\n🥉 TOP 5 MOST IMPORTANT FEATURES:")
```



```
[22] # Drop columns and missing values
     df.drop(['Sunshine', 'Evaporation', 'Cloud9am', 'Cloud3pm', 'Location', 'Date'], axis=1, inplace=True)
```

```
[24] df = df.dropna()
     print(f"\n✅ After cleaning - Dataset Shape: {df.shape}")


     # Visualize data distribution before encoding
     print("\n📊 DATA DISTRIBUTION ANALYSIS")
     print("=" * 35)

     fig, axes = plt.subplots(2, 3, figsize=(18, 12))
     fig.suptitle('🌈 Data Distribution Analysis', fontsize=16, fontweight='bold')

     # Categorical variables visualization
     categorical_cols = ['RainToday', 'RainTomorrow', 'WindGustDir', 'WindDir9am', 'WindDir3pm']

     # RainTomorrow distribution (target variable)
     axes[0, 0].pie(df['RainTomorrow'].value_counts(), labels=['No Rain', 'Rain'],
                    autopct='%1.1f%%', colors=['#FF6B6B', '#4ECDC4'], startangle=90)
     axes[0, 0].set_title('🎯 Target: Rain Tomorrow', fontweight='bold')

     # RainToday distribution
     axes[0, 1].pie(df['RainToday'].value_counts(), labels=['No Rain', 'Rain'],
                    autopct='%1.1f%%', colors=['#FFE66D', '#FF6B6B'], startangle=90)
     axes[0, 1].set_title('☔ Rain Today', fontweight='bold')

     # Temperature distribution
     axes[0, 2].hist(df['Temp9am'].dropna(), bins=30, color='#A8E6CF', alpha=0.7, edgecolor='black')
     axes[0, 2].set_title('🌡️ Temperature 9AM Distribution', fontweight='bold')
     axes[0, 2].set_xlabel('Temperature (°C)')

     # Humidity distribution
     axes[1, 0].hist(df['Humidity9am'].dropna(), bins=30, color='#DCEDC1', alpha=0.7, edgecolor='black')
     axes[1, 0].set_title('💧 Humidity 9AM Distribution', fontweight='bold')
     axes[1, 0].set_xlabel('Humidity (%)')
```