5. Project Development Phase

Development Environment Setup

Prerequisites

```
# Python 3.x installation
python --version

# Create virtual environment
python -m venv venv

# Activate virtual environment
# Windows:
venv\Scripts\activate
# Linux/Mac:
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt
```

Required Libraries

```
flask==2.3.0
pandas==2.0.0
numpy==1.24.0
scikit-learn==1.2.2
xgboost==1.7.5
matplotlib==3.7.1
seaborn==0.12.2
jupyter==1.0.0
```

Data Preparation Implementation

1. Data Loading

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv('data/WeatherAUS.csv')

# Display basic information
print(df.shape)
print(df.info())
print(df.describe())
```

2. Data Cleaning

```
# Handle missing values
# Drop columns with >40% missing values
threshold = 0.4
missing_ratio = df.isnull().sum() / len(df)
columns_to_drop = missing_ratio[missing_ratio > threshold].index
df = df.drop(columns=columns_to_drop)

# Fill numerical missing values with median
numerical_cols = df.select_dtypes(include=[np.number]).columns
df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].median(
```

```
# Fill categorical missing values with mode
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

3. Feature Engineering

```
from sklearn.preprocessing import LabelEncoder

# Encode categorical variables
label_encoders = {}
categorical_features = ['Location', 'WindGustDir', 'WindDir9am',
                        'WindDir3pm', 'RainToday', 'RainTomorrow']

for col in categorical_features:
    le = LabelEncoder()
```

```
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

4. Data Splitting

```
from sklearn.model_selection import train_test_split

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Model Development Implementation

1. Data Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2. Model Training

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=20,
    random_state=42,
    n_jobs=-1
```

```
)
rf_model.fit(X_train_scaled, y_train)
```

XGBoost

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=10,
    learning_rate=0.1,
    random_state=42
)
xgb_model.fit(X_train_scaled, y_train)
```

3. Model Evaluation

```
from sklearn.metrics import accuracy_score, classification_report, confus

# Make predictions
y_pred = rf_model.predict(X_test_scaled)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

4. Model Serialization

```
import pickle

# Save model
with open('Rainfall.pkl', 'wb') as f:
    pickle.dump(rf_model, f)

# Save scaler
with open('scale.pkl', 'wb') as f:
```

```
    pickle.dump(scaler, f)

# Save encoders
with open('encoder.pkl', 'wb') as f:
    pickle.dump({'features': label_encoders}, f)
```

Web Application Implementation

1. Flask Application Structure

```
project/
├── app.py
├── templates/
│   ├── index.html
│   ├── chance.html
│   └── noChance.html
├── static/
│   └── (future: CSS, JS, images)
├── Rainfall.pkl
├── scale.pkl
├── encoder.pkl
└── imputer.pkl
```

2. Backend Implementation (app.py)

Key Features:

- Model loading at startup
- Input validation
- Data preprocessing
- Prediction logic
- Error handling

Main Functions:

```
def load_artifacts():
    # Load ML model and preprocessors

def index():
    # Render input form
```

```
def predict():
    # Process prediction request
```

3. Frontend Implementation

Input Form (index.html)

Features:

- 21 input fields organized in grid
- Dropdown menus for categorical data
- Number inputs with validation
- Responsive design
- Submit button with styling

Result Pages

chance.html:

- Full-screen rainy background
- Animated raindrops
- "chances of rain today." message
- Check Again button

noChance.html:

- Full-screen beach background
- "No chances of rain today, enjoy your outing." message
- Check Again button

Code Quality Standards

Python Code Style (PEP 8)

- 4 spaces for indentation
- Maximum line length: 79 characters
- Descriptive variable names
- Function docstrings
- Type hints where applicable

HTML/CSS Standards

- Semantic HTML5 elements
- Consistent indentation
- Descriptive class names
- Mobile-first responsive design
- Cross-browser compatibility

Git Workflow

```
# Feature branch workflow
git checkout -b feature/model-training
# Make changes
git add .
git commit -m "Add model training pipeline"
git push origin feature/model-training
# Create pull request
```

Testing Implementation

Unit Tests

```
import unittest

class TestPrediction(unittest.TestCase):
    def test_model_loading(self):
        # Test model loads successfully

    def test_data_preprocessing(self):
        # Test preprocessing pipeline

    def test_prediction(self):
        # Test prediction output
```

Integration Tests

- Test form submission
- Test prediction flow
- Test error handling
- Test result display

Performance Optimization

Backend Optimization

- Load models once at startup
- Use efficient data structures
- Minimize preprocessing steps
- Implement caching if needed

Frontend Optimization

- Minimize CSS/JS files
- Optimize images
- Use CDN for external resources
- Implement lazy loading

Debugging and Troubleshooting

Common Issues

Issue 1: Model not loading

```
# Solution: Check file paths and permissions
import os
print(os.path.exists('Rainfall.pkl'))
```

Issue 2: Prediction errors

```
# Solution: Validate input data format
print(input_data.dtypes)
print(input_data.shape)
```

Issue 3: Template not rendering

```
# Solution: Check template folder structure
print(app.template_folder)
```

Version Control

Git Commits

- Frequent, small commits
- Descriptive commit messages
- Follow conventional commits format

Branching Strategy

- `main` : Production-ready code
- `develop` : Development branch
- `feature/*` : Feature branches
- `bugfix/*` : Bug fix branches

Development Best Practices

1. **Write clean, readable code**
2. **Comment complex logic**
3. **Handle errors gracefully**
4. **Validate all inputs**
5. **Test thoroughly**
6. **Document changes**
7. **Follow coding standards**
8. **Use version control**
9. **Review code before merging**
10. **Keep dependencies updated**